Introduction à la programmation

de tablettes Android par l'exemple

-

Version 2

Serge Tahé, IstiA - université d'Angers
novembre 2014

Table des matières

1 APPRENTISSAGE DE LA PROGRAMMATION ANDROID 6
1.1 Introduction 6
1.1.1 Contenu 6
1.1.2 Pré-requis 7
1.1.3 Les outils utilisés 7
1.2 Exemple-01 : un projet Android basique 8
1.2.1 Création du projet 8
1.2.2 Le manifeste de l'application 11
1.2.3 L'activité principale 13
1.2.4 Exécution de l'application 15
1.3 Exemple-02 : une application Maven / Android basique 19
1.4 Exemple-03 : une application Maven [Android Annotations] basique 27
1.5 Exemple-04 : Vues et événements 34
1.5.1 Création du projet 34
1.5.2 Construire une vue 34
1.5.3 Gestion des événements 41
1.6 Exemple-05 : navigation entre vues 44
1.6.1 Création du projet 44
1.6.2 Ajout d'une seconde activité 44
1.6.3 Navigation de la vue n° 1 à la vue n° 2 46
1.6.4 Construction de la vue n° 2 47
1.6.5 L'activité [SecondActivity] 48
1.6.6 Navigation de la vue n° 2 vers la vue n° 1 51
1.7 Exemple-06 : navigation par onglets 53
1.7.1 Création du projet 53
1.7.2 L'activité 55
1.7.3 Personnalisation des fragments 60
1.8 Exemple-07 : utilisation de la bibliothèque [Android Annotations] avec Gradle 61
1.8.1 Création du projet 61
1.8.2 Configuration du projet 63
1.8.3 Ajout d'une première annotation 66
1.8.4 Refactoring des fragments 68
1.8.5 Un nouveau fragment 71
1.8.6 Désactiver le Swipe ou Balayage 74
1.9 Exemple-08 : La navigation entre vues revisitée 78
1.9.1 Création du projet 78
1.9.2 Configuration Gradle du projet 79
1.9.3 Création des fragment et des vues associées 81
1.9.4 Gestion des fragments par l'activité 84
1.9.5 Mise en place de la navigation 87
1.9.6 Ecriture finale des fragments 88
1.9.7 Conclusion 90
1.10 Exemple-09 : une architecture à deux couches 91
1.10.1 Création du projet 91
1.10.2 La vue [vue1] 92
1.10.3 Le fragment [Vue1Fragment] 95
1.10.4 L'activité [MainActivity] 96
1.10.5 La couche [métier] 97
1.10.6 L'activité [MainActivity] revisitée 99
1.10.7 Le fragment [Vue1Fragment] revisité 100
1.10.8 Exécution 102
1.11 Exemple-10 : architecture client / serveur 103
1.11.1 Le serveur [web / jSON] 103
1.11.1.1 Création du projet 103
1.11.1.2 La couche [métier] 107
1.11.1.3 Le service web / jSON 109
1.11.1.4 Configuration du projet Spring 113
1.11.1.5 Exécution du serveur web 114
1.11.1.6 Génération du jar exécutable du projet 118
1.11.2 Le client Android du serveur web / jSON 121
1.11.2.1 Création du projet 121

1.11.2.2 La couche [DAO]....122
1.11.2.2.1 L'interface [IDao] de la couche [DAO]....122
1.11.2.2.2 La classe [AleaResponse]....123
1.11.2.2.3 La classe [WebClient]....123
1.11.2.2.4 Implémentation de la couche [DAO]....124
1.11.2.3 Les dépendances Gradle....126
1.11.2.4 Le manifeste de l'application Android....127
1.11.2.5 L'activité [MainActivity]....128
1.11.2.6 La vue [vue1.xml]....130
1.11.2.7 Le fragment [Vue1Fragment]....133
1.11.2.8 Exécution du projet....137
1.11.3 Conversion du projet [Gradle] en projet [Maven]....140
1.12 Exemple-11 : composants de saisie de données....146
1.12.1 Création du projet....146
1.12.2 La vue XML du formulaire....147
1.12.3 Les chaînes de caractères du formulaire....151
1.12.4 Le fragment du formulaire....152
1.12.5 Exécution du projet....155
1.13 Exemple-12 : utilisation d'un patron de vues....156
1.13.1 Création du projet....156
1.13.2 Le patron des vues....156
1.14 Exemple-13 : le composant [ListView]....161
1.14.1 Création du projet....161
1.14.2 La vue [Vue1] initiale....162
1.14.3 La vue répétée par le [ListView]....164
1.14.4 Le fragment [Vue1Fragment]....165
1.14.5 L'adaptateur [ListAdapter] du [ListView]....167
1.14.6 Retirer un élément de la liste....168
1.14.7 La vue XML [Vue2]....169
1.14.8 Le fragment [Vue2Fragment]....170
1.14.9 Exécution....171
1.14.10 Amélioration....171
1.15 Exemple-14 : utiliser un menu....173
1.15.1 Création du projet....173
1.15.2 La définition XML du menu....173
1.15.3 La gestion du menu dans le fragment [Vue1Fragment]....174
1.15.4 La gestion du menu dans le fragment [Vue2Fragment]....175
1.15.5 Exécution....176
1.16 Annexes....177
1.16.1 La tablette Android....177
1.16.2 Installation d'un JDK....178
1.16.3 Installation du SDK Manager d'Android....179
1.16.4 Installation du gestionnaire d'émulateurs Genymotion....181
1.16.5 Installation de Maven....182
1.16.6 Installation de l'IDE IntellijIDEA Community Edition....183
1.16.7 Utilisation des exemples....189
1.16.8 Installation du plugin Chrome [Advanced Rest Client]....198
1.16.9 Gestion du jSON en Java....199
2 ÉTUDE DE CAS....203
2.1 Le projet....203
2.2 Les vues du client Android....203
2.3 L'architecture du projet....205
2.4 La base de données....206
2.4.1 La table [MEDECINS]....206
2.4.2 La table [CLIENTS]....207
2.4.3 La table [CRENEAUX]....207
2.4.4 La table [RV]....208
2.4.5 Génération de la base....208
2.5 Le serveur web / jSON....210
2.5.1 Mise en oeuvre....211
2.5.2 Sécurisation du service web....212
2.5.3 Liste des médecins....212
2.5.4 Liste des clients....215

2.5.5 Liste des créneaux d'un médecin....216
2.5.6 Liste des rendez-vous d'un médecin....216
2.5.7 L'agenda d'un médecin....217
2.5.8 Obtenir un médecin par son identifiant....218
2.5.9 Obtenir un client par son identifiant....218
2.5.10 Obtenir un créneau par son identifiant....219
2.5.11 Obtenir un rendez-vous par son identifiant....219
2.5.12 Ajouter un rendez-vous....219
2.5.13 Supprimer un rendez-vous....221
2.6 Le client Android....222
2.6.1 Architecture du projet IntellijIDEA....222
2.6.2 Configuration Gradle....223
2.6.3 La couche [DAO]....224
2.6.4 Implémentation des échanges client / serveur....225
2.6.5 L'activité....230
2.6.6 La session....234
2.6.7 Le patron des vues....235
2.6.8 La classe mère des fragments....237
2.6.9 Gestion de la vue de configuration....238
2.6.10 Gestion de la vue d'accueil....244
2.6.11 Gestion de la vue Agenda....249
2.6.12 La vue d'ajout d'un rendez-vous....257
2.7 Conclusion....262
2.8 Annexes....263
2.8.1 Installation de [WampServer]....263
3 TP 1 : GESTION BASIQUE D'UNE FICHE DE PAIE....266
3.1 Introduction....266
3.2 La base de données....266
3.2.1 Définition....266
3.2.2 Génération....267
3.2.3 Modélisation Java de la base....268
3.3 Installation du serveur web / jSON....269
3.3.1 Installation....270
3.3.2 Les URL du service web/jSON....271
3.3.3 Les réponses jSON du service web/jSON....273
3.4 Tests du client Android....275
3.5 Travail à faire....277
4 TP 2 - PILOTER DES ARDUINOS AVEC UNE TABLETTE ANDROID....280
4.1 Architecture du projet....280
4.2 Le matériel....280
4.2.1 L'Arduino....280
4.2.2 La tablette....282
4.2.3 L'émulateur [Genymotion]....283
4.3 Programmation des Arduinos....283
4.4 Le serveur web / jSON Java....287
4.4.1 Installation....287
4.4.2 Les URL exposées par le service web / jSON....289
4.4.3 Les tests du service web / jSON....292
4.5 Tests du client Android....295
4.6 Le client Android du service web / jSON....298
4.6.1 L'architecture du client....298
4.6.2 Le projet IntellijIDEA du client....299
4.6.3 Les cinq vues XML....301
4.6.4 Les cinq fragments....302
4.6.5 La classe [MainActivity]....302
4.6.6 La vue XML [config]....305
4.6.7 Le fragment [ConfigFragment]....309
4.6.8 Vérification des saisies....310
4.6.9 Gestion des onglets....311
4.6.10 Affichage de la liste des Arduinos....313
4.6.11 Un patron pour afficher un Arduino....317
4.6.12 La communication entre vues....321
4.6.13 La couche [DAO]....326

4.6.13.1 Implémentation....327
4.6.13.2 Mise en oeuvre....329
4.6.13.3 Exécution du projet....331
4.6.13.4 Refactorisation de la classe [MyFragment]....331
4.7 Travail à faire....332
4.8 Annexes....333
4.8.1 Intallation de l'IDE Arduino....333
4.8.2 Intallation du pilote (driver) de l'Arduino....333
4.8.3 Tests de l'IDE....333
4.8.4 Connexion réseau de l'Arduino....335
4.8.5 Test d'une application réseau....336
4.8.6 La bibliothèque aJson....339

1 Apprentissage de la programmation Android

1.1 Introduction

1.1.1 Contenu

Ce document est la réécriture de plusieurs documents existants :

- Android pour les développeurs J2EE ;
- Introduction à la programmation de tablettes Android par l'exemple ;
- Commander un Arduino avec une tablette Android ;

et introduit les nouveautés suivantes :

- utilisation de la bibliothèque [Android Annotations] [http://androidannotations.org/]. Cette bibliothèque réduit la quantité de code à écrire ;
- utilisation de l'IDE [IntellijIDEA Community Edition], version 13.1.5 [http://www.jetbrains.com/idea/download/] à la place d'Eclipse ;

Les exemples suivants sont présentés :

Exemple	Nature
1	Un projet Android basique
2	Une application Maven / Android basique
3	Une application [Android Annotations] basique
4	Vues et événements
5	Navigation entre vues
6	Navigation par onglets
7	Utilisation de la bibliothèque [Android Annotations] avec Gradle
8	Navigation entre vues revisitée
9	Architecture à deux couches
10	Architecture client / serveur
11	Composants de saisie de données
12	Utilisation d'un patron de vues
13	Le composant ListView
14	Utiliser un menu
étude de cas	Gestion des rendez-vous d'un cabinet de médecins
TP 1	Gestion basique d'une paie
TP 2	Commande de cartes Arduino

Ce document est utilisé en dernière année de l'école d'ingénieurs IstiA de l'université d'Angers [istia.univ-angers.fr]. Cela explique le ton parfois un peu particulier du texte. TP 1 et TP2 sont des textes de TP dont on ne donne que les grandes lignes de la solution. Celle-ci est à construire par le lecteur.

Le code source des exemples est disponible à l'URL [http://tahe.ftp-developpez.com/fichiers-archive/android-exemples-intellij-aa.zip].

Ce document est un document de formation partielle à la programmation Android. Il ne se veut pas exhaustif. Il cible essentiellement les débutants.

Le site de référence pour la programmation Android est à l'URL [http://developer.android.com/guide/components/index.html]. C'est là qu'il faut aller pour avoir une vue d'ensemble de la programmation Android.

1.1.2 Pré-requis

Les pré-requis à une utilisation optimale du document sont les suivants :

1. bonne maîtrise du langage Java ;
2. connaissances élémentaires d'IntellijIDEA et des projets Maven ;

Le point 2 n'est pas bloquant. Son apprentissage peut se faire en suivant les exemples pas à pas.

1.1.3 Les outils utilisés

Les exemples qui suivent ont été testés dans l'environnement suivant :

- machine Windows 8.1 pro 64 bits ;
- JDK 1.8 ;
- Android SDK API 19 et 20 ;
- IntellijIDEA Community Edition 14.0.3 ;
- tablette Samsung Galaxy Tab 2 ;

Pour suivre ce document vous devez installer :

- un JDK (cf paragraphe 1.16.2, page 178) ;
- un SDK Android (cf paragraphe 1.16.3, page 179) ;
- le gestionnaire d'émulateurs Genymotion (cf paragraphe 1.16.4, page 181) ;
- le gestionnaire de dépendances Maven (cf paragraphe 1.16.5, page 182) ;
- l'IDE [IntellijIDEA Community Edition] (cf paragraphe 1.16.6, page 183) ;

Les projets IntellijIDEA des exemples sont disponibles à l'URL [http://tahe.ftp-developpez.com/fichiers-archive/android-exemples-intellij-aa.zip]. Pour les exécuter, vous devez suivre la procédure du paragraphe 1.16.7, page 189.

Google préconise d'utiliser [Android Studio] [https://developer.android.com/sdk/installing/studio.html] pour les développements Android. [Android Studio] est dérivé de [IntellijIDEA] et les deux produits sont très semblables. [IntellijIDEA Community Edition] a été préféré ici à [Android Studio] pour deux raisons :

- [Android Studio] n'est pas 'Maven friendly'. Les projets [Android Studio] utilisent de préférence Gradle pour gérer leurs dépendances. Ce ne serait pas gênant s'il était possible également d'utiliser Maven. Ce n'est pas le cas ;
- [Android Studio] est orienté clients Android. Dans une application client / serveur, il n'offre pas de facilités pour créer le serveur. Quand on veut créer un projet, c'est uniquement un projet Android ;

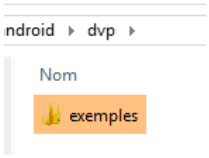
[IntellijIDEA Community Edition] n'offre pas ces deux inconvénients et par ailleurs il bénéficie régulièrement des améliorations apportées à [Android Studio].

1.2 Exemple-01 : un projet Android basique

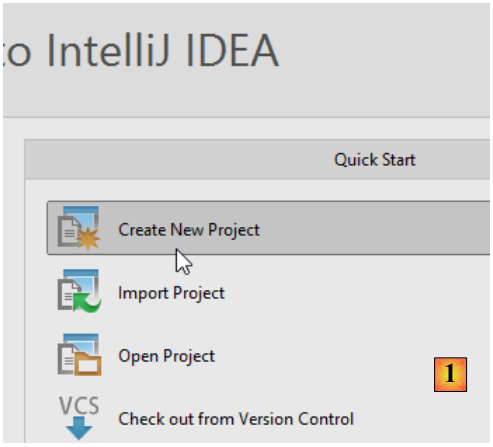
Créons avec IntellijIDEA un premier projet Android.

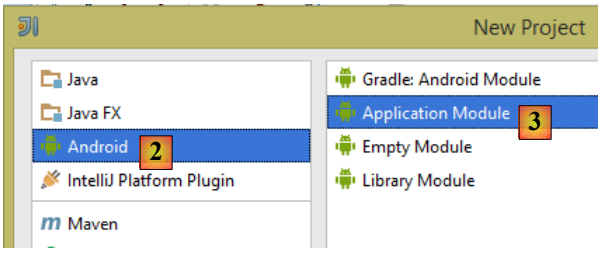
1.2.1 Création du projet

Tout d'abord créons un dossier [exemples] vide où seront placés tous nos projets :

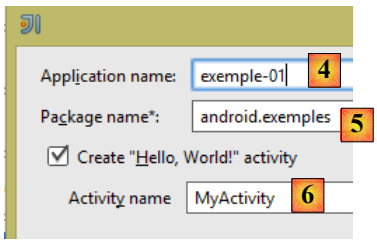


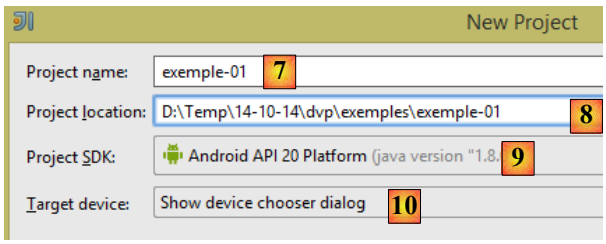
puis créons un projet avec IntellijIDEA] :





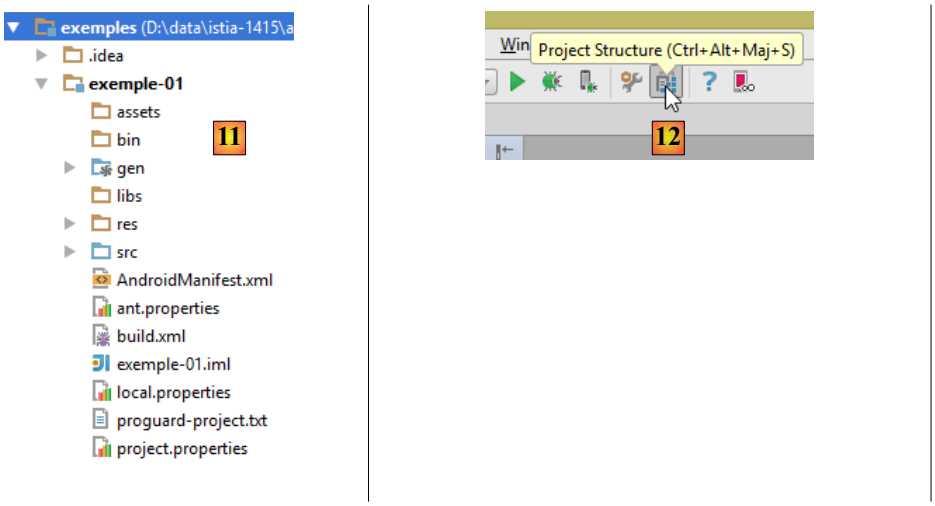
- en [1-3], on crée un nouveau projet Android de type [Application Module] ;



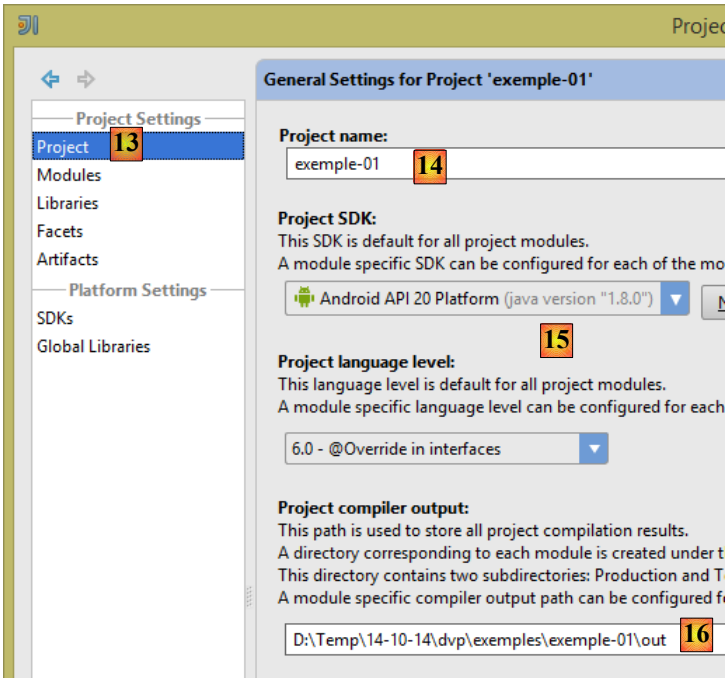


- en [4], on donne un nom à l'application Android ;
- en [5], on donne un nom au package dans lequel sera l'activité Android ;
- en [6], on donne un nom à cette activité ;
- en [7], on donne un nom au projet ;
- en [8], on fixe le dossier de celui-ci ;
- en [9], on choisit une plateforme Android. Celles présentées dans la liste sont celles installées avec le SDK (cf paragraphe 1.16.3, page 179) ;
- en [10], on fixe le périphérique d'exécution de l'application (une tablette, un émulateur, ...). [Show device chooser dialog] permet de donner cette information au dernier moment, juste avant l'exécution de l'application ;

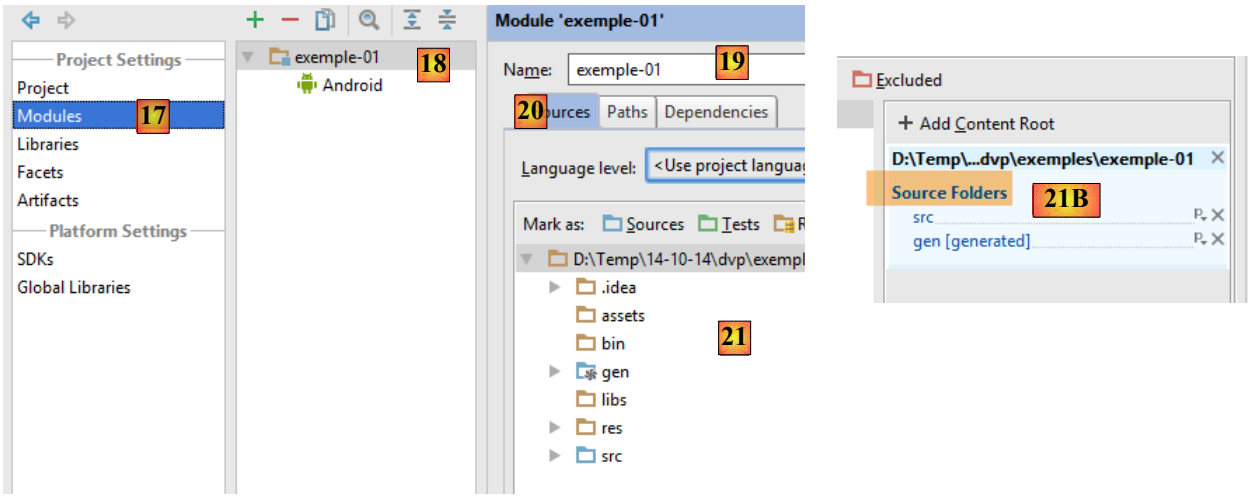
Le projet créé est le suivant [11] :



- en [12], on accède aux propriétés du module ;

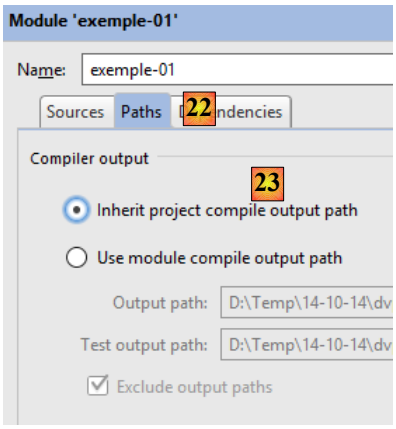


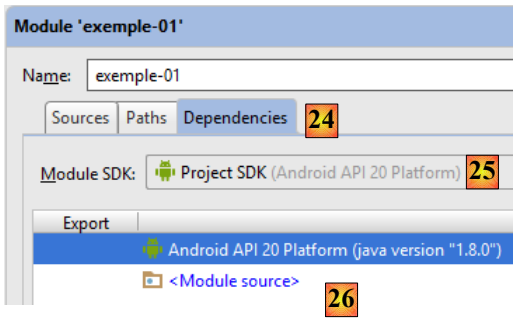
- en [13-16], la configuration du projet ;



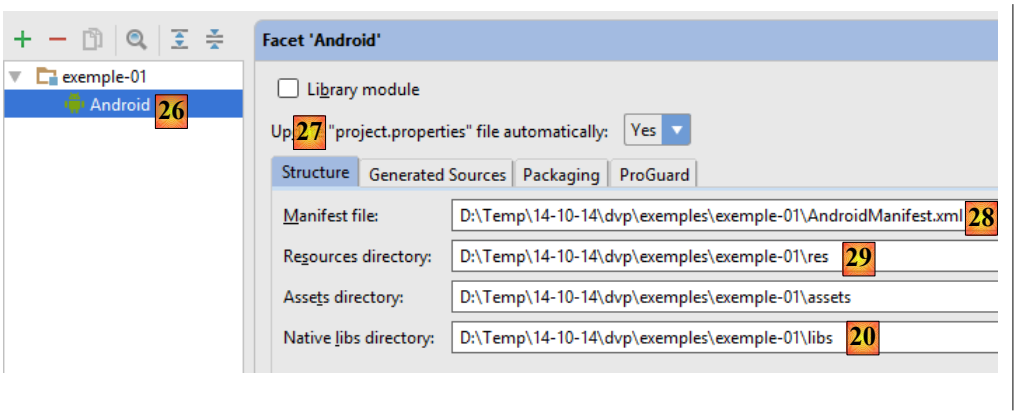
- en [17-19], un projet IntellijIDEA est composé d'un ou de plusieurs modules. Dans tout ce document, les projets n'auront qu'un module ;
- en [20-21B], la configuration du code source du module ;

En [20-21], il est important de vérifier les dossiers qui contiennent du code source. Ils sont en bleu. La compilation d'un projet peut échouer si certains dossiers contenant du code source n'ont pas été renseignés. C'est typiquement le cas lorsqu'on importe un projet provenant d'un autre IDE (Eclipse par exemple).



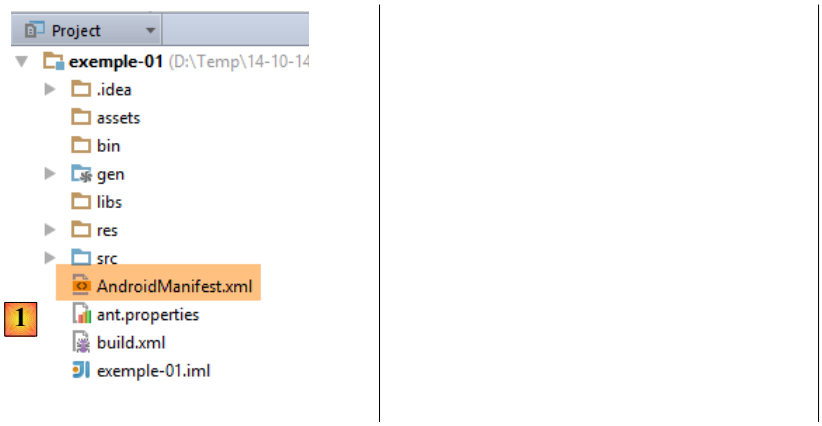


- en [22-23], la configuration du dossier où seront stockés les produits issus de la compilation. La configuration faite par défaut ici indique que ce dossier est celui configuré pour le projet parent du module (cf plus haut page 9, pastille 16) ;
- en [24-26], les dépendances du module Android. Ici, il n'y en a pas en-dehors de l'API 20 du SDK (25) ;



- en [26-27], la structure du module Android ;
- en [28], le fichier [AndroidManifest.xml] qui configure le projet Android ;
- en [29], le dossier des ressources du projet Android ;
- en [30], un dossier de bibliothèques Java. Toute archive Java placée dans ce dossier fera partie du Classpath du projet ;

1.2.2 Le manifeste de l'application

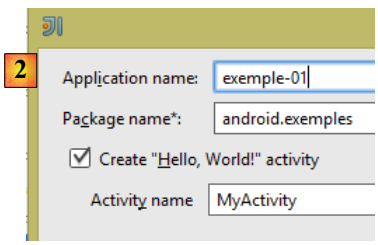


Le fichier [AndroidManifest.xml] [1] fixe les caractéristiques de l'application Android. Son contenu est ici le suivant :

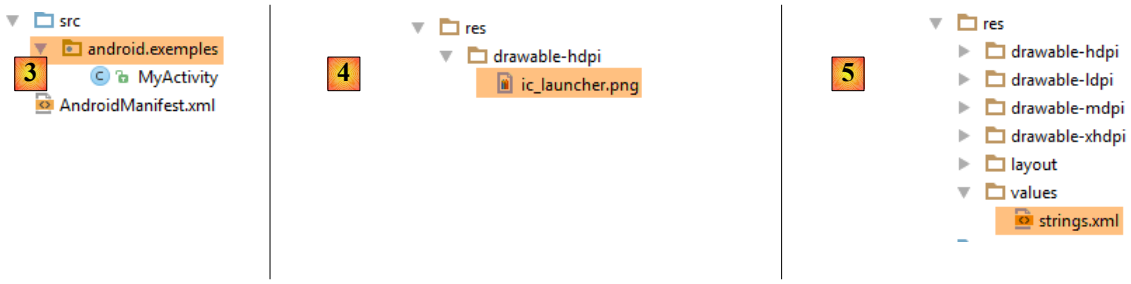
```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.            package="android.exemples"
4.            android:versionCode="1"
5.            android:versionName="1.0">
6.
7.    <uses-sdk android:minSdkVersion="20"/>
8.    <application
9.      android:label="@string/app_name"
10.     android:icon="@drawable/ic_launcher">
11.     <activity
12.       android:name="MyActivity"
13.       android:label="@string/app_name">
14.       <intent-filter>
15.         <action android:name="android.intent.action.MAIN"/>
16.         <category android:name="android.intent.category.LAUNCHER"/>
17.       </intent-filter>
18.     </activity>
19.   </application>
20. </manifest>
```

- ligne 3 : le paquetage du projet Android ;
- ligne 12 : le nom de l'activité ;

Ces deux renseignements viennent des saisies faites lors de la création du projet [2] :



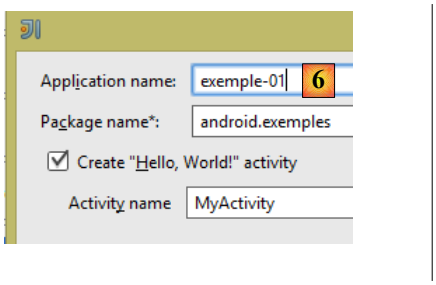
- ligne 3 : le paquetage du projet Android. Un certain nombre de classes sont automatiquement générées dans ce paquetage [3] ;



- ligne 7 : la version minimale d'Android pouvant exécuter l'application. Ici la version 20 ;
- ligne 10 : l'icône [4] de l'application. Elle peut être changée ;
- ligne 14 : le libellé de l'aplication. Il se trouve dans le fichier [strings.xml] [5] :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.    <string name="app_name">exemple-01</string>
4.  </resources>
```

Le fichier [strings.xml] contient les chaînes de caractères utilisées par l'application. Ligne 3, le nom de l'application provient de la saisie faite lors de la construction du projet [6] :



- ligne 11 : une balise d'activité. Une application Android peut avoir plusieurs activités ;
- ligne 15 : l'activité est désignée comme étant l'activité principale ;
- ligne 16 : et elle doit apparaître dans la liste des applications qu'il est possible de lancer sur l'appareil Android.

Nous modifions le fichier [AndroidManifest.xml] de la façon suivante :

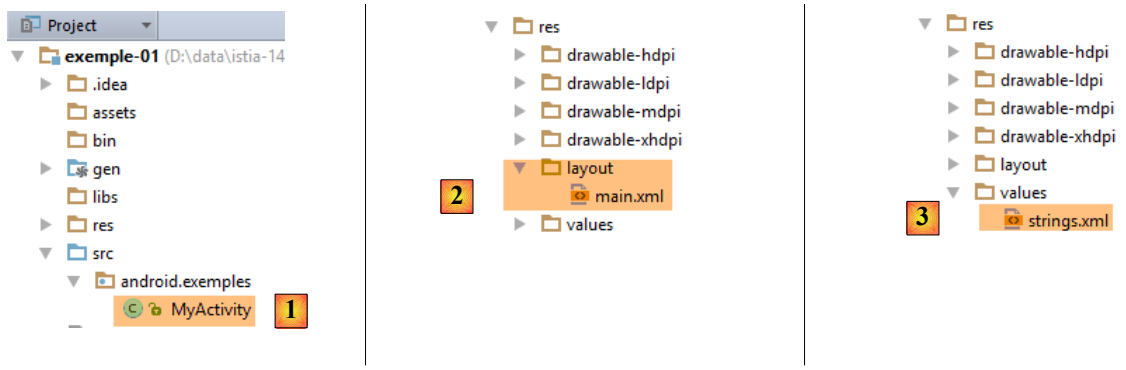
```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.            package="android.exemples"
4.            android:versionCode="1"
5.            android:versionName="1.0">
6.    <uses-sdk
7.      android:minSdkVersion="11"
8.      android:targetSdkVersion="20"/>
9.    <application
10.     android:label="@string/app_name"
11.     android:icon="@drawable/ic_launcher">
12.     <activity android:name="MyActivity"
13.               android:label="@string/app_name">
14.       <intent-filter>
15.         <action android:name="android.intent.action.MAIN"/>
16.         <category android:name="android.intent.category.LAUNCHER"/>
17.       </intent-filter>
18.     </activity>
19.   </application>
20. </manifest>
```

- en [11], on indique que le périphérique d'exécution doit être propulsé par une API >= 11 ;
- en [12], on indique que le périphérique d'exécution cible est un périphérique propulsé par l'API 20 ;

La syntaxe générale est la suivante :

```
<uses-sdk android:minSdkVersion="integer"
          android:targetSdkVersion="integer"
          android:maxSdkVersion="integer" />
```

1.2.3 L'activité principale



Une application Android repose sur une ou plusieurs activités. Ici une activité [1] a été générée : [MyActivity]. Une activité peut afficher une ou plusieurs vues selon son type. La classe [MyActivity] générée est la suivante :

```
1.  package android.exemples;
2.
3.  import android.app.Activity;
4.  import android.os.Bundle;
5.
6.  public class MyActivity extends Activity {
7.    @Override
8.    public void onCreate(Bundle savedInstanceState) {
9.      super.onCreate(savedInstanceState);
10.     setContentView(R.layout.main);
11.   }
12. }
```

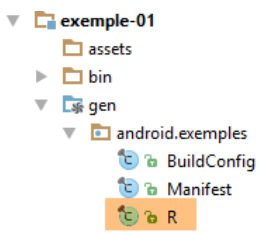
- ligne 6 : la classe [MyActivity] étend la classe [Activity]. C'est le cas de toutes les activités ;
- ligne 8 : la méthode [onCreate] est exécutée lorsque l'activité est créée. C'est avant l'affichage de la vue associée à l'activité ;
- ligne 9 : la méthode [onCreate] de la classe parente est appelée. Il faut toujours le faire ;
- ligne 10 : le fichier [main.xml] [2] est la vue associée à l'activité. La définition XML de cette vue est la suivante :

```
a)  <?xml version="1.0" encoding="utf-8"?>
b)  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
c)                android:orientation="vertical"
d)                android:layout_width="fill_parent"
e)                android:layout_height="fill_parent"
f)    >
g)
h)    <TextView
i)      android:layout_width="fill_parent"
j)      android:layout_height="wrap_content"
k)      android:text="Hello World, MyActivity"
l)      />
m)  </LinearLayout>
```

- lignes **a-f** : le gestionnaire de mise en forme. Celui qui a été choisi par défaut est le type [LinearLayout]. Dans ce type de conteneur, les composants sont placés les uns sous les autres. Le conteneur conseillé est [RelativeLayout] où on place les composants les uns par rapport aux autres (à droite de, à gauche de, dessous, au-dessus) ;
- lignes **h-l** : un composant de type [TextView] qui sert à afficher du texte ;
- ligne **k** : le texte affiché. Il est déconseillé de mettre du texte en dur dans les vues. Il est préférable de déplacer ces textes dans le fichier [res/values/strings.xml] [3] :

Le texte affiché sera donc [*Hello World, MyActivity*]. Où sera-t-il affiché ? Le conteneur [LinearLayout] va remplir l'écran. Le [TextView] qui est son seul et unique élément est affiché en haut et à gauche de ce conteneur, donc en haut et à gauche de l'écran ;

Que signifie [R.layout.main] ligne 10 ? Chaque ressource Android (vues, fragments, composants, ...) se voit attribuée un identifiant. Ainsi une vue [V.xml] se trouvant dans le dossier [res / layout] sera identifiée par [R.layout.V]. R est une classe générée dans le dossier [gen] :



La classe [R] est la suivante :

```
1.  /*___Generated_by_IDEA___*/
2.
3.  package android.exemples;
4.
5.  /* This stub is only used by the IDE. It is NOT the R class actually packed into the APK */
6.  public final class R {
7.  }
```

Cette classe est un squelette et n'est pas celle embarquée dans le binaire [APK]. Lorsqu'on développe avec Eclipse on a accès à la classe [R] :

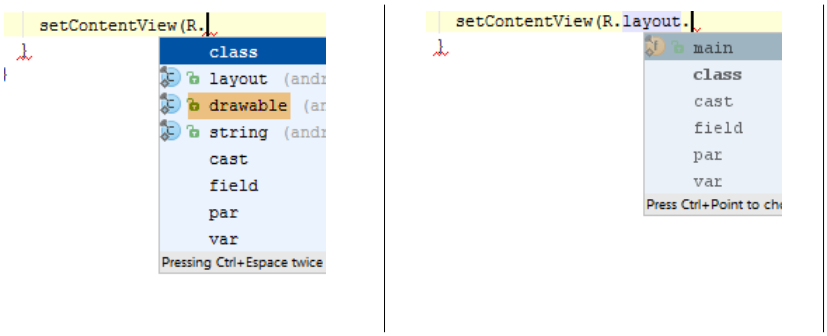
```
1.  package android.exemples;
2.
3.  public final class R {
4.      public static final class attr {
5.      }
6.      public static final class drawable {
7.          public static final int ic_launcher=0x7f020000;
8.      }
9.      public static final class layout {
10.         public static final int main=0x7f030000;
11.     }
12.     public static final class string {
13.         public static final int app_name=0x7f040000;
14.     }
15. }
```

- ligne 10 : l'attribut [R.layout.main] est l'identifiant de la vue [res / layout / main.xml] ;
- ligne 12 : l'attribut [R.string.app_name] est l'identifiant de la chaîne [app_name] dans le fichier [res / values / string.xml] :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">exemple-02</string>
</resources>
```

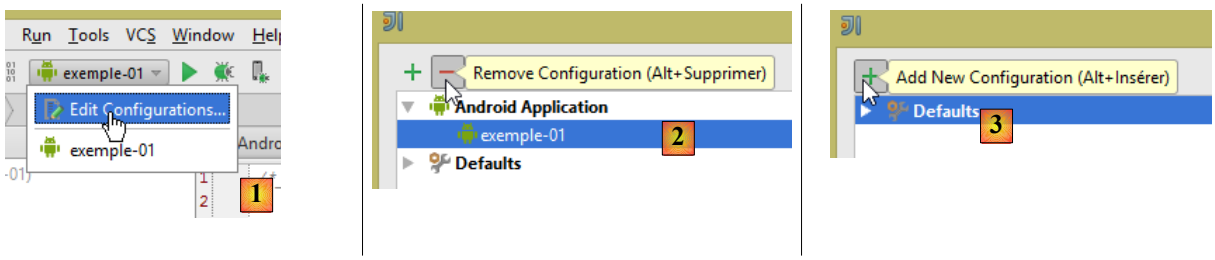
- ligne 7 : l'attribut [R.drawable.ic_launcher] est l'identifiant de l'image [res / drawable-hdpi / ic_launcher] et de celles de même nom dans les dossiers [drawable-ldpi, drawable-mdpi, drawable-xhdpi] ;

Avec [IntellijIDEA], on se souviendra donc que lorsqu'on référence [R.layout.main], on référence un attribut de la classe [R]. L'IDE nous aide à connaître les différents éléments de cette classe :

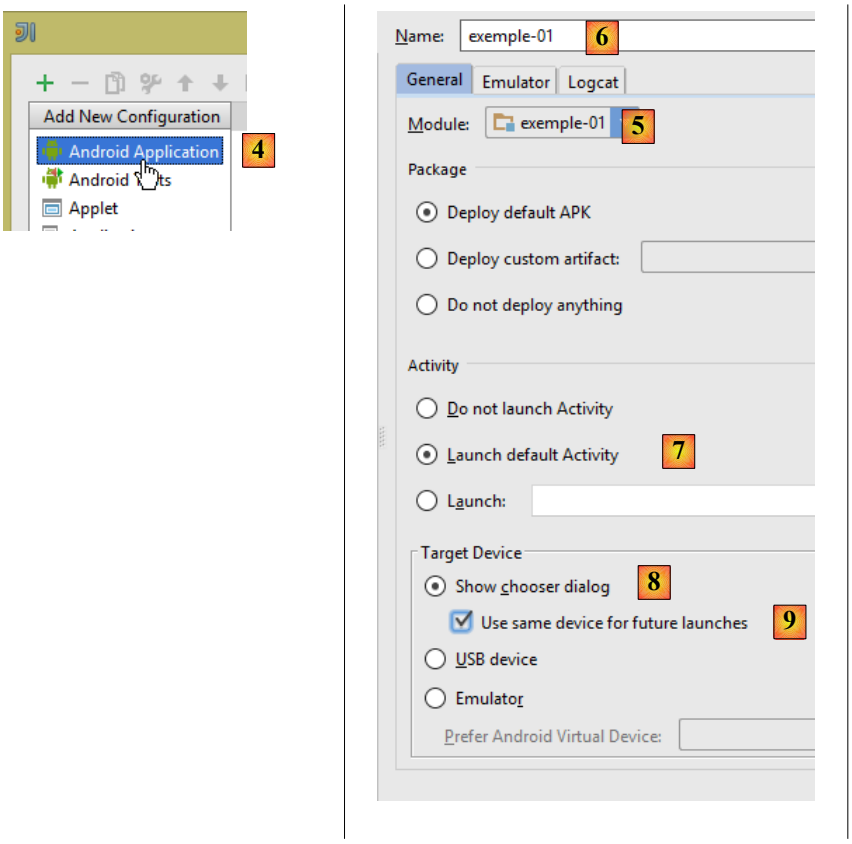


1.2.4 Exécution de l'application

Pour exécuter une application Android, il nous faut créer une configuration d'exécution :



- en [1], choisir [Edit Configurations] ;
- le projet a été créé avec une configuration [exemple-01] que nous allons supprimer [2] pour la recréer ;
- en [3], créer une nouvelle configuration d'exécution ;

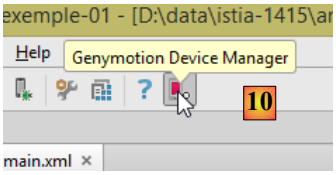


- en [4], choisir [Android Application] ;

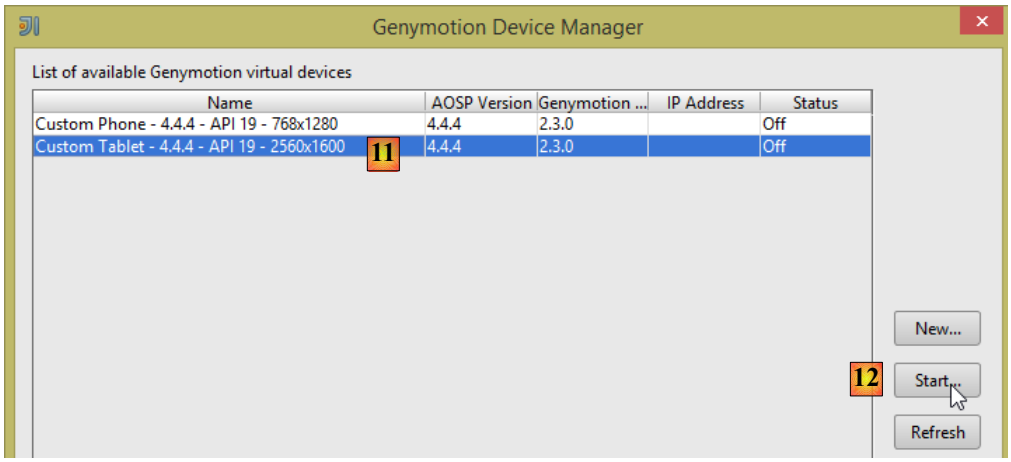
- en [5], dans la liste déroulante choisir le projet [exemple-01] ;
- en [6], donner un nom à cette configuration d'exécution ;
- en [7], sélectionner [Launch Default Activity]. Cette activité par défaut est celle définie dans le fichier [AndroidManifest.xml] (ligne 2 ci-dessous) :

```
1.      <activity
2.        android:name="MyActivity"
3.        android:label="@string/app_name">
4.        <intent-filter>
5.          <action android:name="android.intent.action.MAIN"/>
6.          <category android:name="android.intent.category.LAUNCHER"/>
7.        </intent-filter>
8.  </activity>
```

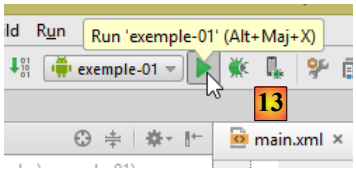
- en [8], sélectionner [Show Chooser Dialog] qui permet de choisir après la compilation le périphérique d'exécution de l'application (émulateur, tablette) ;
- en [9], on indique que ce choix doit être mémorisé ;
- validez la configuration ;

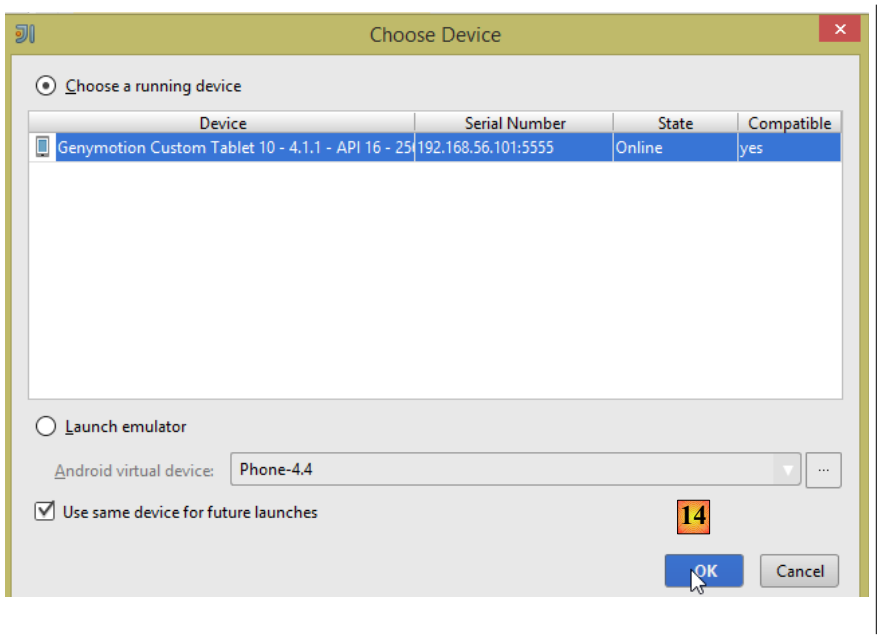


- en [10], lancer le gestionnaire des émulateurs [Genymotion] (cf paragraphe 1.16.4, page 181) ;



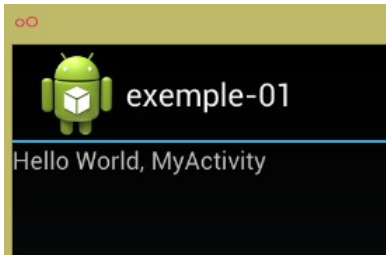
- en [11], sélectionner un émulateur de tablette et lancez le [12] ;



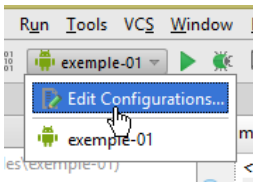


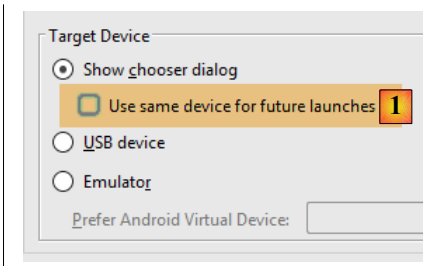
- en [13], exécutez la configuration d'exécution [exemple-01] ;
- en [14] est présenté le formulaire de choix du périphérique d'exécution. Un seul est ici disponible : l'émulateur [Genymotion] lancé précédemment ;

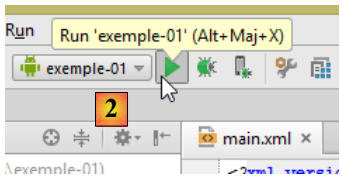
L'émulateur logiciel affiche au bout d'un moment la vue suivante :



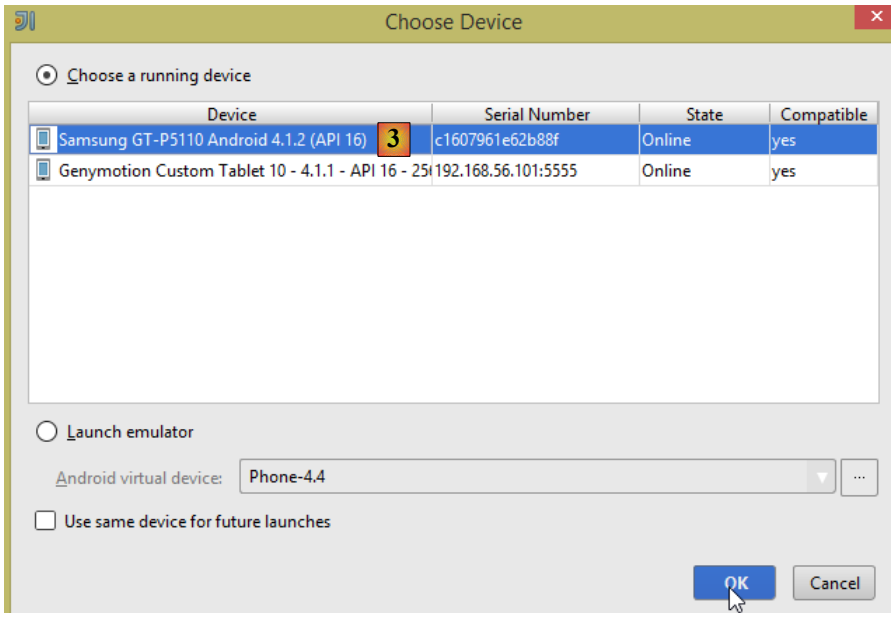
Branchez maintenant une tablette Android sur un port USB du PC puis modifiez la configuration d'exécution [exemple-01] de la façon suivante :







- en [1], décochez la phrase [Use same device...] puis validez la nouvelle configuration ;
- en [2], réexécutez l'application ;

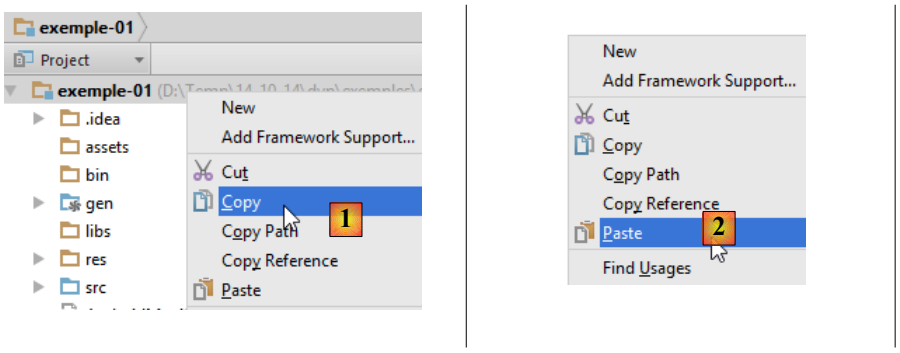


- en [3], sélectionnez la tablette Android et testez l'application. Vous devez obtenir la même chose que sur l'émulateur.

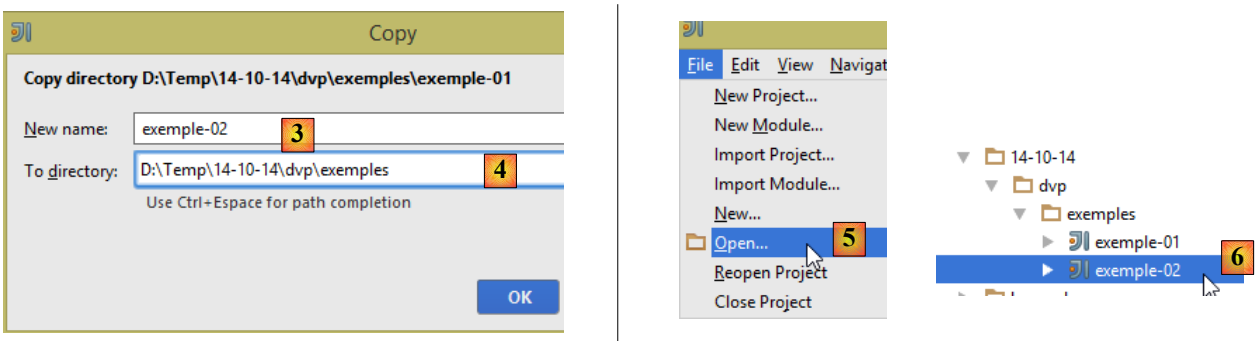
1.3 Exemple-02 : une application Maven / Android basique

Par la suite, les dépendances de certains de nos projets Android vont être gérées avec Maven [http://maven.apache.org/]. Maven est un outil qui sait faire beaucoup de choses qui facilitent la vie des développeurs. Nous n'allons l'utiliser que pour une seule de ses fonctionnalités : la gestion des dépendances d'un projet, ç-à-d la liste des bibliothèques Java (jars) qui doivent être présentes dans le Classpath du projet pour qu'il soit compilable et exécutable.

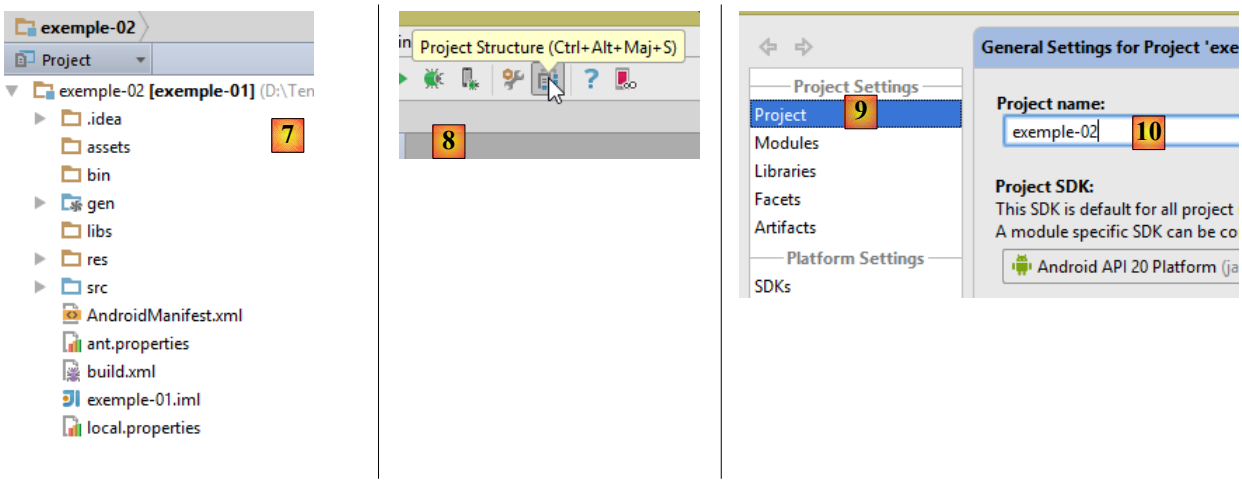
Nous allons dupliquer le module [exemple-01] :



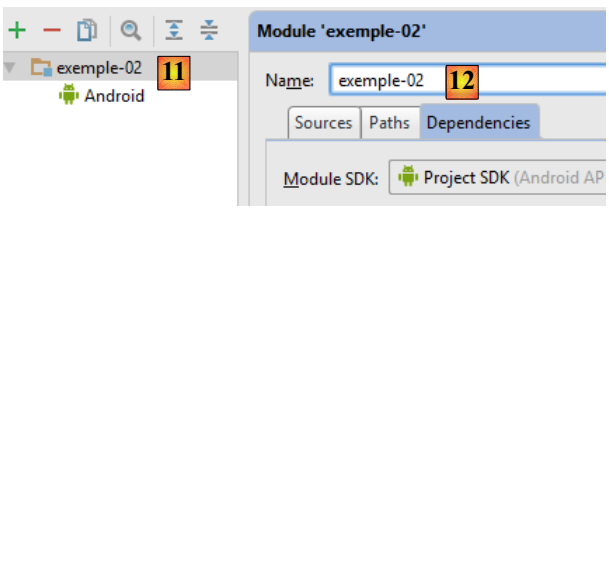
- en [1], clic droit sur le projet [exemple-01] puis sélectionner [Copy] ;
- en [2], clic droit dans la fenêtre du projet puis sélectionner [Paste] ;

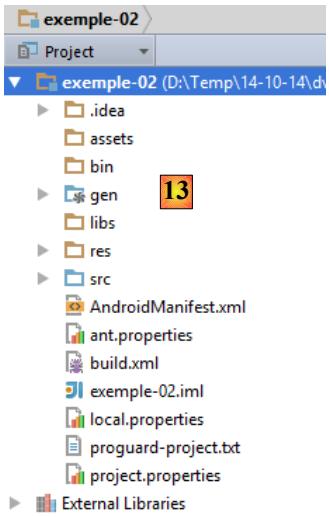


- en [3], donner un nom au nouveau module ;
- en [4], fixer le dossier **parent** de ce nouveau module. Ici ce sera le dossier [exemples] ;
- en [5-6], ouvrez le projet [exemple-02] que vous venez de créer ;

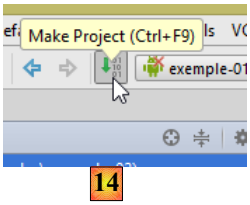


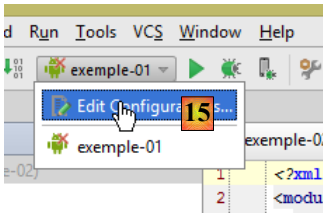
- en [7], le projet [exemple-02] ;
- en [8-10], on change le nom du projet ;



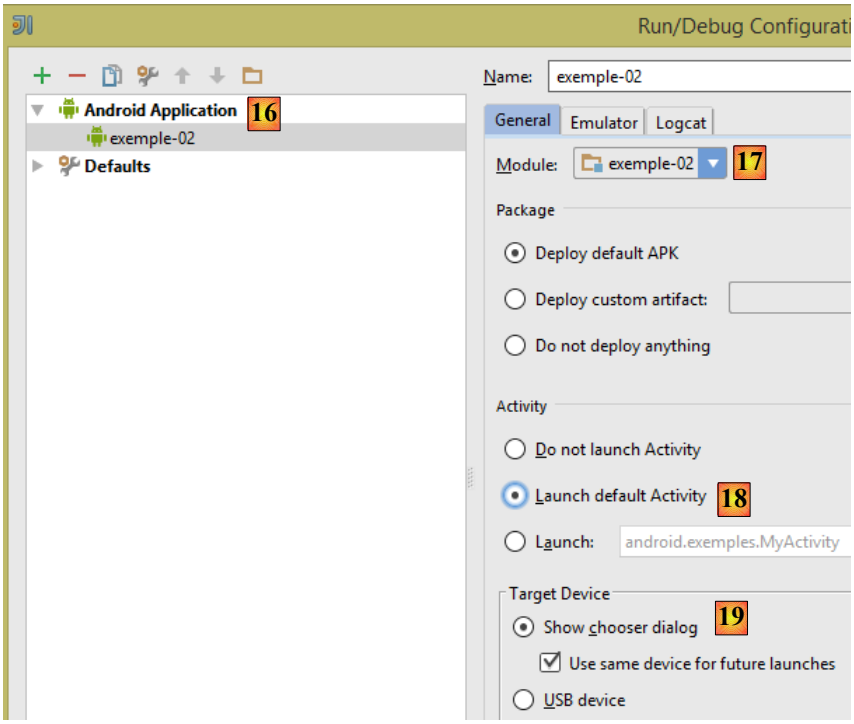


- en [11-12], on change le nom du module ;
- en [13], le nouveau projet. Vous pouvez remarquer que son nom a changé (pastilles 13 et 7) ;

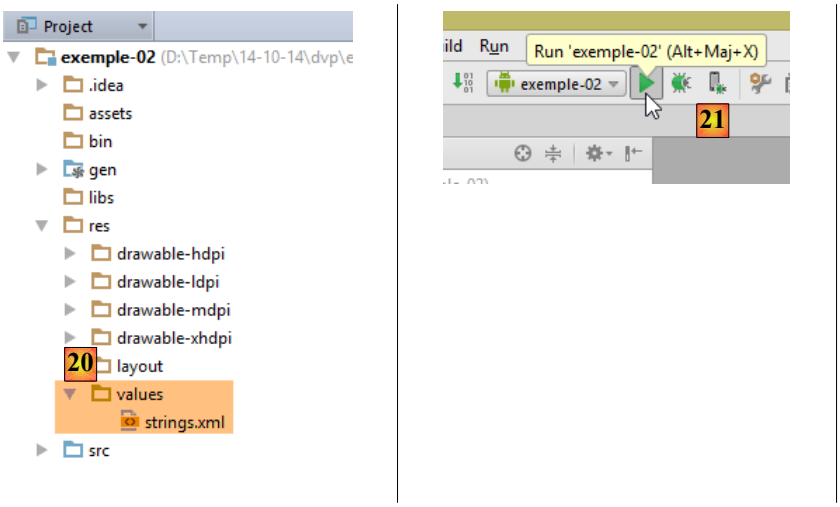




- en [14], on compile le projet ;
- en [15], on édite les configurations d'exécution ;



- en [16-19], on crée une configuration d'exécution nommée [exemple-02] ;

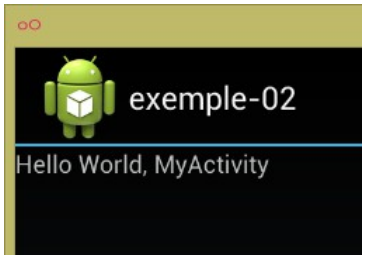


- en [20], on modifie le fichier [strings.xml] ;

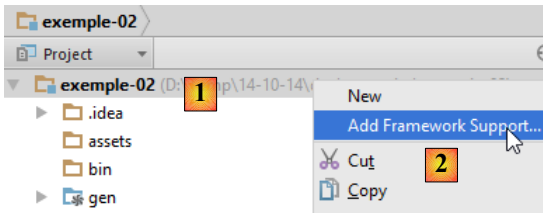
```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.    <string name="app_name">exemple-01</string>
4.  </resources>
```

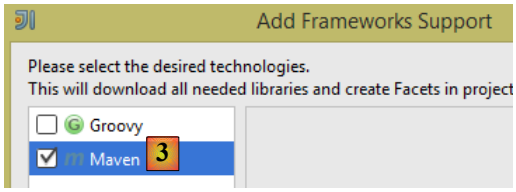
Ligne 3, changez la chaîne [exemple-01] en [exemple-02].

- en [21], exécutez la configuration d'exécution [exemple-02]. Vous devez obtenir la vue suivante :

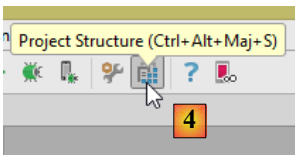


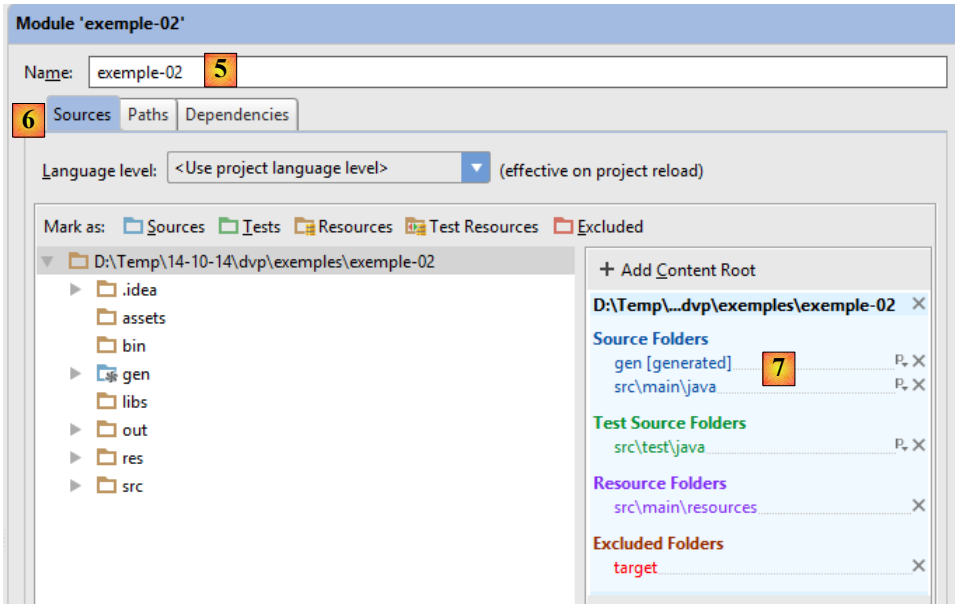
Nous avons désormais une copie du module [exemple-01] que nous transformons maintenant en module Maven.



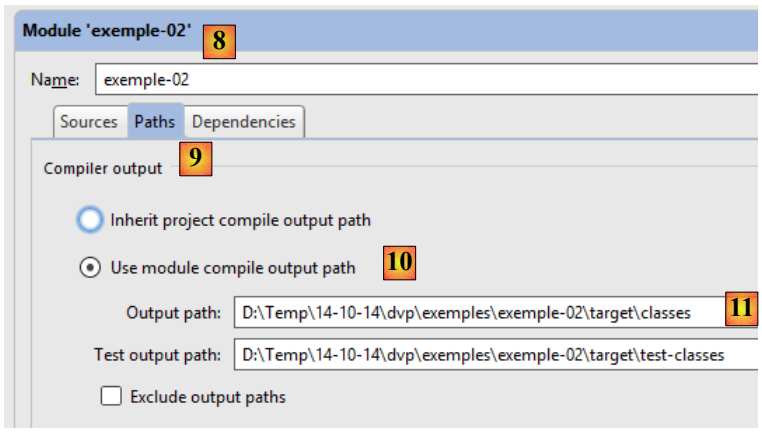


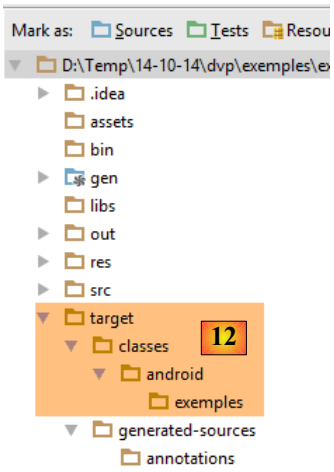
- en [1-3], on ajoute un support Maven au projet ;



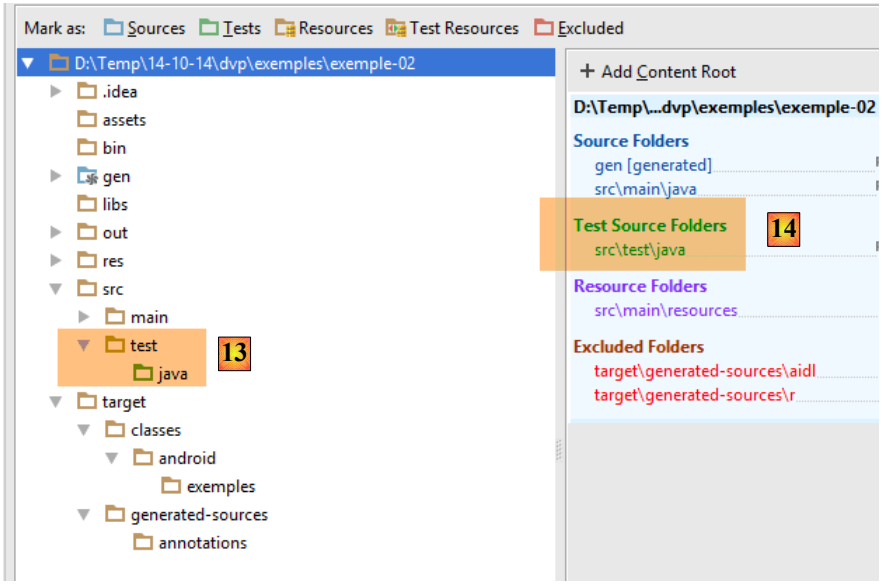


- en [4], on accède à la structure du projet ;
- en [5-7], on peut voir que la structure du module [exemple-02] a changé. Avant, le dossier [src] contenait les codes source. Maintenant c'est le dossier [src / main / java] (7) ;
- en [7], un nouveau dossier [gen] va contenir du code source. Ces codes source vont être générés par le processus de [build] de Maven ;

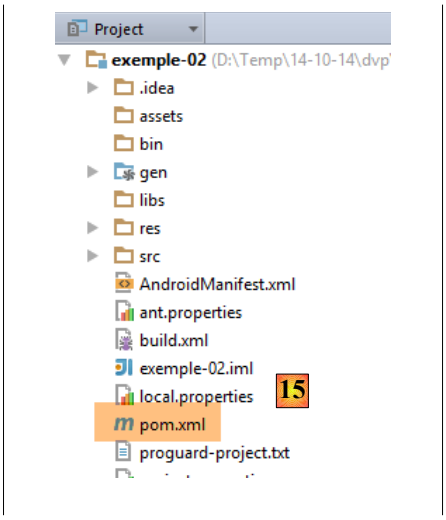




- en [8-12], les produits issus du [build] Maven iront dans le dossier [target / classes] ;



- en [13-14], un dossier [src / test / java] pourra contenir les classes de test du projet. Ces classes ne sont pas incluses dans le produit du [build] ;



- en [15], un fichier [pom.xml] a été ajouté. C'est le fichier de configuration des projets Maven. C'est dans ce fichier qu'on va contrôler les dépendances Maven.

Le contenu de ce fichier sera le suivant :

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4.    <modelVersion>4.0.0</modelVersion>
5.    <groupId>android.exemples</groupId>
6.    <artifactId>exemple-02</artifactId>
7.    <version>1.0-SNAPSHOT</version>
8.    <packaging>apk</packaging>
9.    <name>exemple-02</name>
10.
11.   <properties>
12.     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
13.   </properties>
14.
15.   <dependencies>
16.     <!-- Android -->
17.     <dependency>
18.       <groupId>com.google.android</groupId>
19.       <artifactId>android</artifactId>
20.       <version>4.1.1.4</version>
21.       <scope>provided</scope>
22.     </dependency>
23.   </dependencies>
24.   <!-- build -->
25.   <build>
26.     <finalName>${project.artifactId}</finalName>
27.     <pluginManagement>
28.       <plugins>
29.         <plugin>
30.           <groupId>com.jayway.maven.plugins.android.generation2</groupId>
31.           <artifactId>android-maven-plugin</artifactId>
32.           <version>3.8.2</version>
33.           <extensions>true</extensions>
34.         </plugin>
35.       </plugins>
36.     </pluginManagement>
37.     <plugins>
38.       <plugin>
39.         <groupId>com.jayway.maven.plugins.android.generation2</groupId>
40.         <artifactId>android-maven-plugin</artifactId>
41.         <configuration>
42.           <sdk>
43.             <platform>20</platform>
44.           </sdk>
45.         </configuration>
46.       </plugin>
```

```
47.     </plugins>
48.   </build>
49. </project>
```

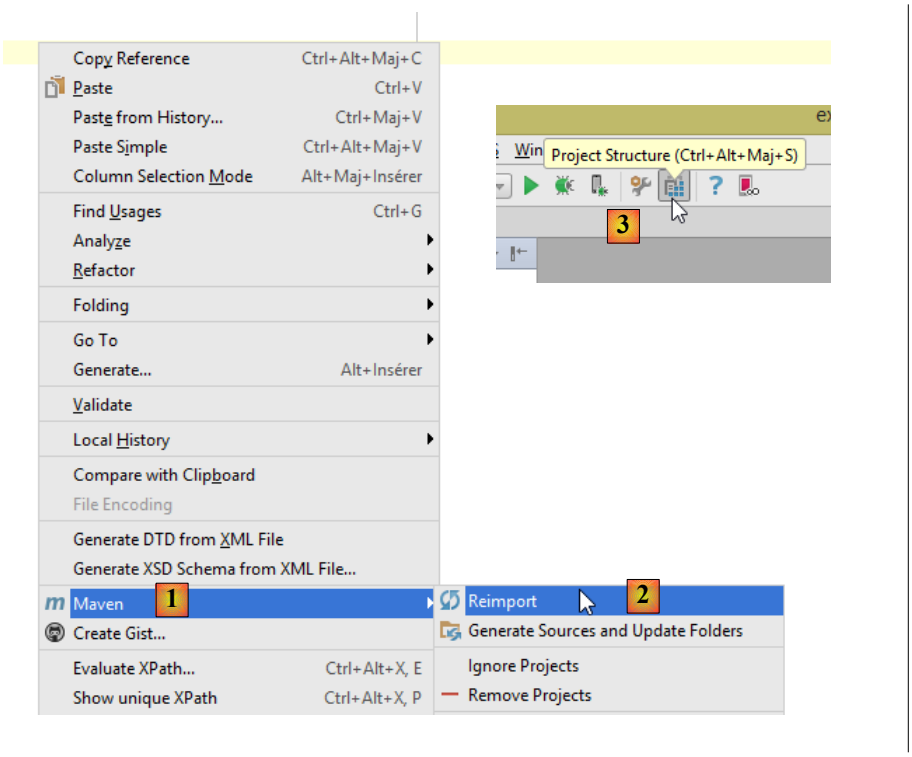
Une compilation Maven (un Build) produit un artifact identifié par plusieurs informations :

- un identifiant [artifactId] (ligne 6) ;
- un groupe [groupId] (ligne 5). On peut produire plusieurs artifacts appartenant au même groupe ;
- une version [version] (ligne 7). Pour un même artifact, on peut produire des versions successives n'apportant pas les mêmes fonctionnalités ;
- le type du fichier de l'artifact (ligne 8). Le type [jar] est le type par défaut pour les applications Java. Ici le type [apk] (Android package) est destiné aux applications Android. C'est ce type qui est déployé sur les périphériques Android ;
- un nom (ligne 9). Cette information est facultative ;
- lignes 5-9 : identification de l'artifact produit par la compilation du module [exemple-02]. Les informations entrées ici n'ont pas de liens avec celles définies dans le fichier [AndroidManifest.xml]. Ici par exemple le [groupId] [android.exemples] est aussi le nom donné au package de l'application Android dans le fichier [AndroidManifest.xml]. Ca n'a rien d'obligatoire ;
- lignes 15-23 : les dépendances nécessaires au module [exemple-02] ;
- lignes 17-22 : pour être compilé, le module a besoin des bibliothèques d'Android ;
- ligne 21 : cette dépendance n'est nécessaire que pour la compilation. Elle n'est pas nécessaire lorsque l'[apk] produit est déployé sur un périphérique Android qui fournit alors cette dépendance. Ce fonctionnement est obtenu avec la balise :

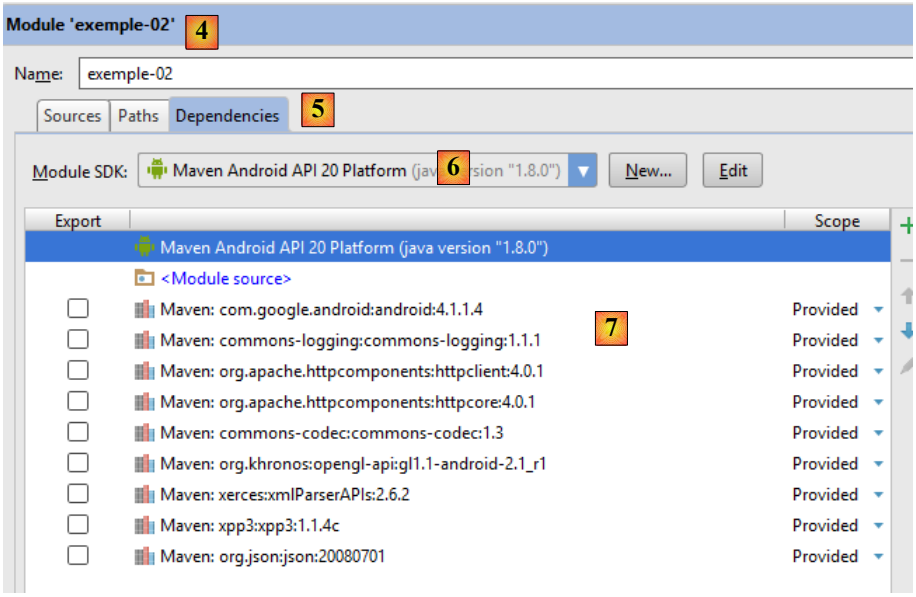
```
<scope>provided</scope>
```

- lignes 25-48 : configurent le processus de compilation (build) du projet Maven. On fera attention à la ligne 43. Le numéro de plateforme doit être le numéro d'une des API téléchargées avec le SDK Manager d'Android. Ici, nous n'avons téléchargé que l'API 20.

Pour télécharger ces dépendances, il faut forcer un peu les choses :

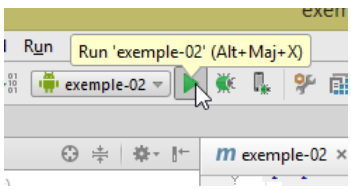


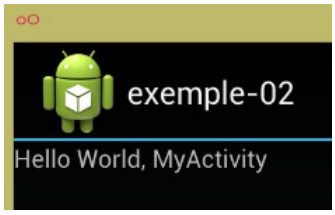
- en [1-2], on force Maven à relire son fichier de configuration [pom.xml] et à télécharger les dépendances ;
- en [3], on accède à la structure du projet Maven ;



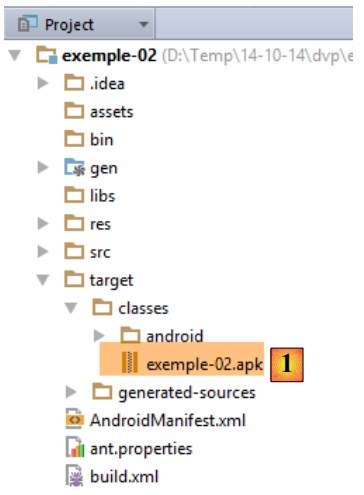
- en [4,7], on retrouve les dépendances inscrites dans le fichier [pom.xml] ;

A ce point, on peut exécuter le nouveau module [exemple-02] :





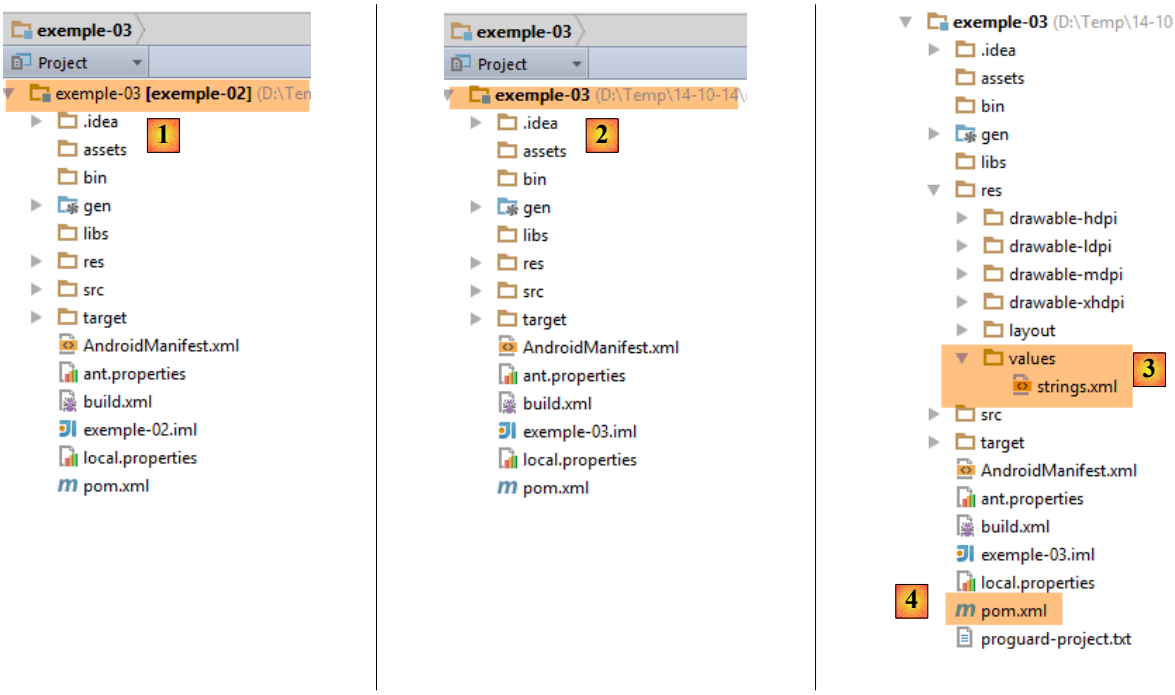
La compilation Maven a produit l'[apk] [1] du module dans le dossier [target] :



- en [1], le binaire [apk] du module [exemple-02]. Ce binaire est transportable et exécutable sur tout périphérique Android. On peut également le faire glisser de l'IDE [IntellijIDEA] et le déposer sur l'émulateur [Genymotion] qui va alors l'enregistrer et l'exécuter ;

1.4 Exemple-03 : une application Maven [Android Annotations] basique

Nous allons maintenant introduire la bibliothèque [Android Annotations] qui facilite l'écriture des applications Android. Pour cela on duplique le module [exemple-02] dans [exemple-03]. On suivra la procédure décrite pour dupliquer [exemple-01] dans [exemple-02] mais il faut auparavant **supprimer** le dossier [target] du module [exemple-02] :



- en [1], le projet créé. En suivant la démarche utilisée précédemment on renomme le projet et le module en [exemple-03] [2] (notez la différence surlignée) ;

Nous continuons la personnalisation du projet [exemple-03] avec les modifications suivantes :

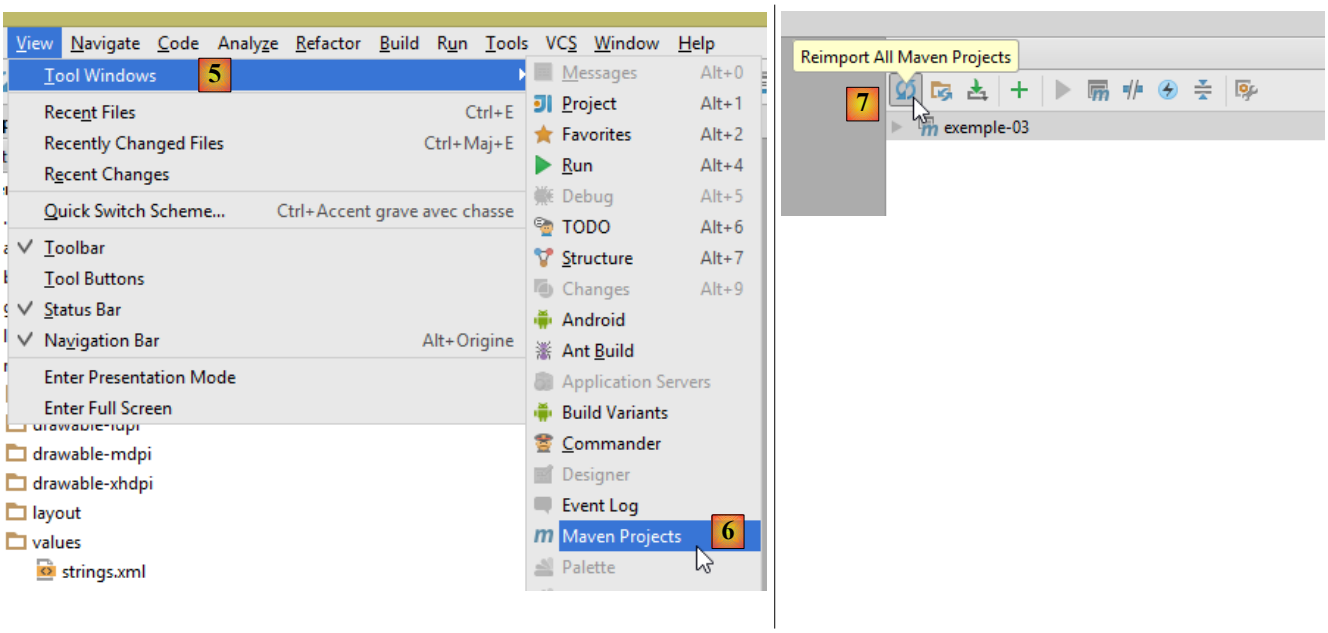
- le fichier [res / values / strings.xml] [3] est modifié :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.    <string name="app_name">exemple-03</string>
4.  </resources>
```

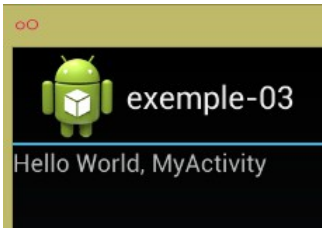
- le fichier [pom.xml] [4] est modifié :

```
1.    <modelVersion>4.0.0</modelVersion>
2.    <groupId>android.exemples</groupId>
3.    <artifactId>exemple-03</artifactId>
4.    <version>1.0-SNAPSHOT</version>
5.    <packaging>apk</packaging>
6.  <name>exemple-03</name>
```

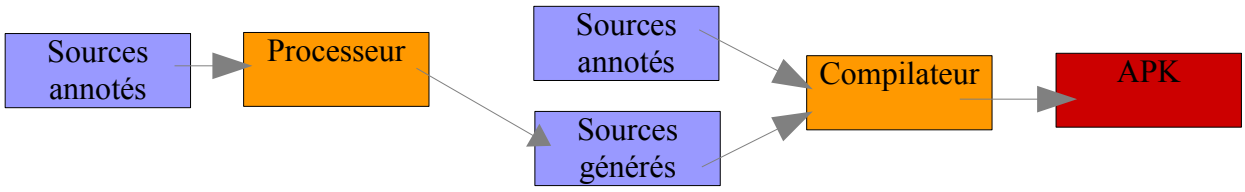
Nous rafraîchissons le nouveau projet [5-7] :



Ceci fait, on crée une configuration d'exécution nommée [exemple-03] et on l'exécute :



Nous allons maintenant introduire la bibliothèque [Android Annotations] que nous appellerons par facilité **AA**. Cette bibliothèque introduit de nouvelles classes pour annoter les sources Android. Ces annotations vont être utilisées par un processeur qui va créer de nouvelles classes Java dans le module, classes qui participeront à la compilation de celui-ci au même titre que les classes écrites par le développeur. On a ainsi la chaîne de compilation suivante :

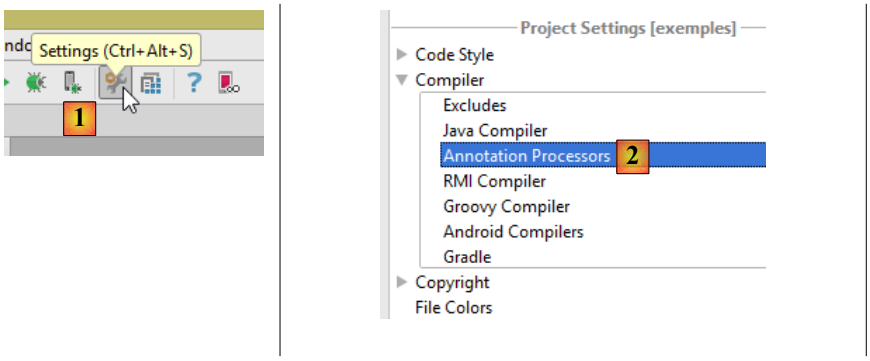


Nous allons tout d'abord mettre dans le fichier [pom.xml] les dépendances sur le compilateur d'annotations AA (processeur ci-dessus) :

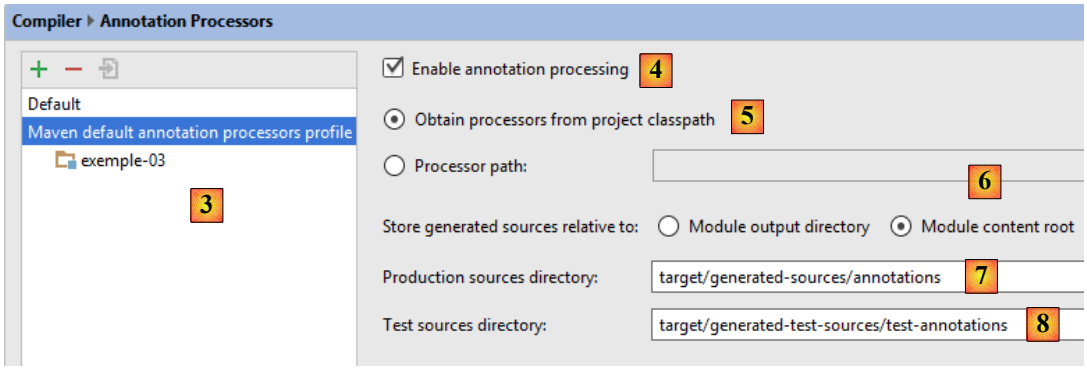
```
1.    <dependencies>
2.      <!-- Android -->
3.      <dependency>
4.        <groupId>com.google.android</groupId>
5.        <artifactId>android</artifactId>
6.        <version>4.1.1.4</version>
7.        <scope>provided</scope>
8.      </dependency>
9.      <!-- Android annotations-->
10.     <dependency>
11.       <groupId>org.androidannotations</groupId>
12.       <artifactId>androidannotations</artifactId>
```

```
13.       <version>3.1</version>
14.     </dependency>
15.     <dependency>
16.       <groupId>org.androidannotations</groupId>
17.       <artifactId>androidannotations-api</artifactId>
18.       <version>3.1</version>
19.     </dependency>
20. </dependencies>
```

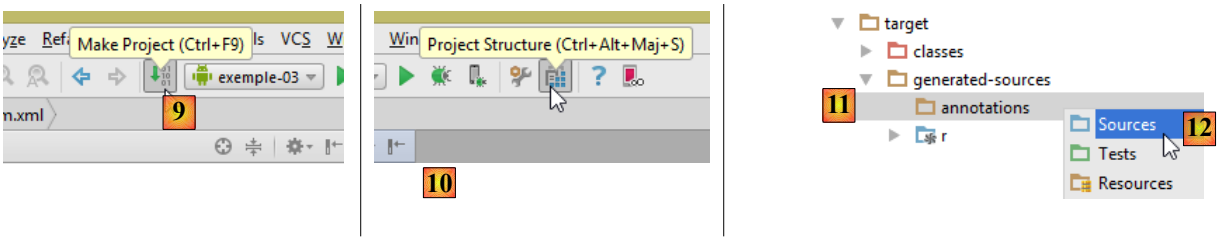
- les lignes 10-19 ajoutent les deux dépendances qui forment la bibliothèque AA ;



- en [1], on accède à la configuration du projet [exemples] ;
- en [2], on configure l'option [Compiler / Annotation Processors] qui sert à configurer les processeurs d'annotations tels que AA ;

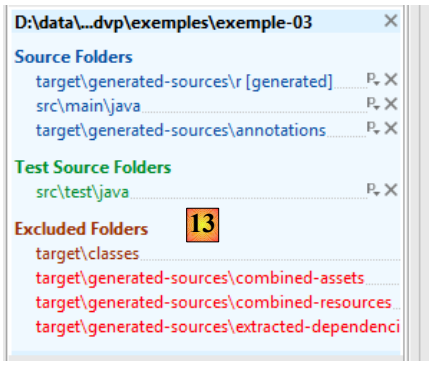


- en [3], on configure le module [exemple-03] ;
- en [4], on autorise les processeurs d'annotations ;
- en [5], on indique que ceux-ci seront trouvés dans le Classpath du projet. Il faut se rappeler que AA étant une dépendance Maven, elle sera automatiquement dans le Classpath du projet ;
- en [7], on indique dans quel dossier doivent être placés les produits du processeur d'annotations ;
- en [6], on indique la racine de [7]. Donc [7] sera <module>/target/generated-sources/annotations où <module> est le dossier du module configuré ;
- en [8], une opération similaire pour les sources de tests ;



- en [9], on compile le module [exemple-03] ;
- en [10], on accède aux propriétés du module [exemple-03] ;

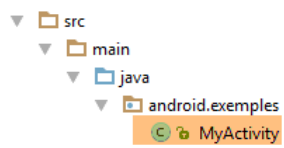
- en [11-12], on fait du dossier [target/generated-sources/annotations] un dossier de sources à compiler. C'est en effet dans ce dossier que AA va placer les classes issues du traitement des annotations Java trouvées dans les codes source ;



- en [13], un récapitulatif des configurations faites.

A ce stade, vérifiez que la configuration d'exécution [exemple-03] fonctionne toujours.

Nous allons maintenant introduire une première annotation de la bibliothèque AA dans la classe [MainActivity] :



La classe [MyActivity] est pour l'instant la suivante :

```
1.  package android.exemples;
2.
3.  import android.app.Activity;
4.  import android.os.Bundle;
5.
6.  public class MyActivity extends Activity {
7.    @Override
8.    public void onCreate(Bundle savedInstanceState) {
9.      super.onCreate(savedInstanceState);
10.     setContentView(R.layout.main);
11.   }
12. }
```

Nous avons déjà expliqué ce code au paragraphe 1.2.4, page 15. Nous le modifions de la façon suivante :

```
1.  package android.exemples;
2.
3.  import android.app.Activity;
4.  import android.os.Bundle;
5.  import org.androidannotations.annotations.EActivity;
6.
7.  @EActivity(R.layout.main)
8.  public class MyActivity extends Activity {
9.    @Override
10.   public void onCreate(Bundle savedInstanceState) {
11.     super.onCreate(savedInstanceState);
12.   }
13. }
```

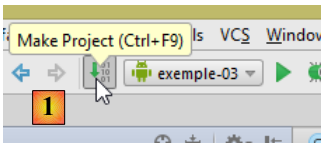
- ligne 7 : l'annotation [@EActivity] est une annotation AA (ligne 7). Son paramètre est la vue associée à l'activité ;

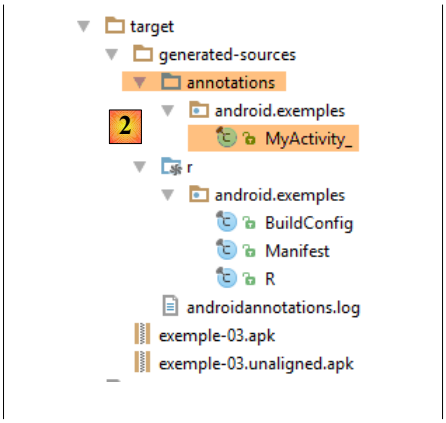
Cette annotation va produire une classe [MyActivity_] dérivée de la classe [MyActivity] et c'est cette classe qui sera la véritable activité. Nous devons donc modifier le manifeste du projet [AndroidManifest.xml] de la façon suivante :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.            package="android.exemples"
4.            android:versionCode="1"
5.            android:versionName="1.0">
6.    <uses-sdk
7.      android:minSdkVersion="11"
8.      android:targetSdkVersion="20"/>
9.    <application
10.     android:label="@string/app_name"
11.     android:icon="@drawable/ic_Launcher">
12.     <activity android:name="MyActivity_"
13.               android:label="@string/app_name">
14.       <intent-filter>
15.         <action android:name="android.intent.action.MAIN"/>
16.         <category android:name="android.intent.category.LAUNCHER"/>
17.       </intent-filter>
18.     </activity>
19.   </application>
20. </manifest>
```

- ligne 12 : la nouvelle activité ;

Ceci fait, nous pouvons compiler le projet [1] :





- en [2], on voit la classe [MyActivity_] générée dans le dossier [annotations] ;

La classe [MyActivity_] générée est la suivante :

```
1.  //
2.  // DO NOT EDIT THIS FILE, IT HAS BEEN GENERATED USING AndroidAnnotations 3.1.
3.  //
4.
5.
6.  package android.exemples;
7.
8.  import android.content.Context;
9.  import android.exemples.R.layout;
10. import android.os.Bundle;
11. import android.view.View;
12. import android.view.ViewGroup.LayoutParams;
13. import org.androidannotations.api.builder.ActivityIntentBuilder;
14. import org.androidannotations.api.view.HasViews;
15. import org.androidannotations.api.view.OnViewChangedNotifier;
16.
17. public final class MyActivity_
18.     extends MyActivity
19.     implements HasViews
20. {
21.
22.     private final OnViewChangedNotifier onViewChangedNotifier_ = new OnViewChangedNotifier();
```

```
23.
24.     @Override
25.     public void onCreate(Bundle savedInstanceState) {
26.         OnViewChangedNotifier previousNotifier =
    OnViewChangedNotifier.replaceNotifier(onViewChangedNotifier_);
27.         init_(savedInstanceState);
28.         super.onCreate(savedInstanceState);
29.         OnViewChangedNotifier.replaceNotifier(previousNotifier);
30.         setContentView(layout.main);
31.     }
32.
33.     private void init_(Bundle savedInstanceState) {
34.     }
35.
36.     @Override
37.     public void setContentView(int layoutResID) {
38.         super.setContentView(layoutResID);
39.         onViewChangedNotifier_.notifyViewChanged(this);
40.     }
41.
42.     @Override
43.     public void setContentView(View view, LayoutParams params) {
44.         super.setContentView(view, params);
45.         onViewChangedNotifier_.notifyViewChanged(this);
46.     }
47.
48.     @Override
49.     public void setContentView(View view) {
50.         super.setContentView(view);
51.         onViewChangedNotifier_.notifyViewChanged(this);
52.     }
53.
54.     public static MyActivity_.IntentBuilder_ intent(Context context) {
55.         return new MyActivity_.IntentBuilder_(context);
56.     }
57.
58.     public static MyActivity_.IntentBuilder_ intent(android.app.Fragment fragment) {
59.         return new MyActivity_.IntentBuilder_(fragment);
60.     }
61.
62.     public static MyActivity_.IntentBuilder_ intent(android.support.v4.app.Fragment supportFragment) {
63.         return new MyActivity_.IntentBuilder_(supportFragment);
64.     }
65.
66.     public static class IntentBuilder_
67.         extends ActivityIntentBuilder<MyActivity_.IntentBuilder_>
68.     {
69.
70.         private android.app.Fragment fragment_;
71.         private android.support.v4.app.Fragment fragmentSupport_;
72.
73.         public IntentBuilder_(Context context) {
74.             super(context, MyActivity_.class);
75.         }
76.
77.         public IntentBuilder_(android.app.Fragment fragment) {
78.             super(fragment.getActivity(), MyActivity_.class);
79.             fragment_ = fragment;
80.         }
81.
82.         public IntentBuilder_(android.support.v4.app.Fragment fragment) {
83.             super(fragment.getActivity(), MyActivity_.class);
84.             fragmentSupport_ = fragment;
85.         }
86.
87.         @Override
88.         public void startForResult(int requestCode) {
89.             if (fragmentSupport_!= null) {
90.                 fragmentSupport_.startActivityForResult(intent, requestCode);
91.             } else {
92.                 if (fragment_!= null) {
93.                     fragment_.startActivityForResult(intent, requestCode);
94.                 } else {
95.                     super.startForResult(requestCode);
96.                 }
```

```
97.             }
98.         }
99.
100.    }
101.
102.}
```

- lignes 17-18 : la classe [MyActivity_] étend la classe [MyActivity] ;

Nous ne chercherons pas à expliquer le code des classes générées par AA. Elles gèrent la complexité que les annotations cherchent à cacher. Mais il peut être parfois bon de l'examiner lorsqu'on veut comprendre comment sont 'traduites' les annotations qu'on utilise.

On peut désormais exécuter de nouveau la configuration [exemple-03]. On obtient le même résultat qu'auparavant. Nous allons désormais partir de ce projet que nous dupliquerons pour présenter les notions importantes de la programmation Android.

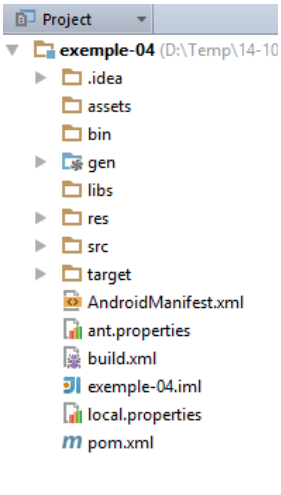
1.5 Exemple-04 : Vues et événements

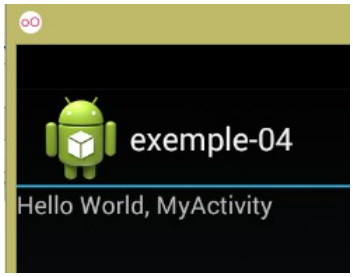
1.5.1 Création du projet

On suivra la procédure décrite pour dupliquer [exemple-02] dans [exemple-03] au paragraphe 1.4, page 27 :

Nous :

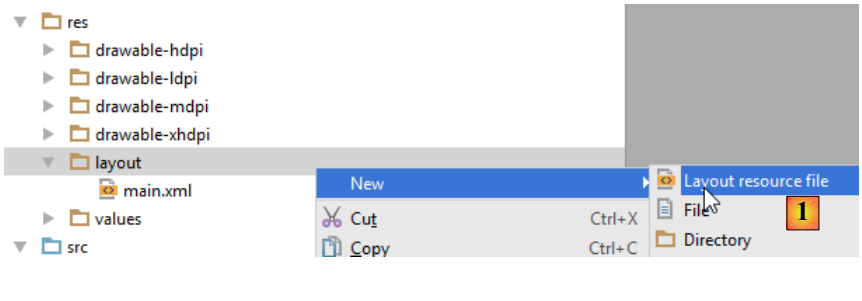
- dupliquons le module [exemple-03] dans [exemple-04] (après avoir supprimé le dossier [target] de [exemple-03]) ;
- chargeons le module [exemple-04] ;
- changeons le nom du projet et du module en [exemple-04] (structure du projet) ;
- changeons le nom du projet dans les fichiers [pom.xml, strings.xml] ;
- rafraîchissons le projet Maven ;
- le compilons ;
- créons une configuration d'exécution nommée [exemple-04] ;
- exécutons celle-ci ;

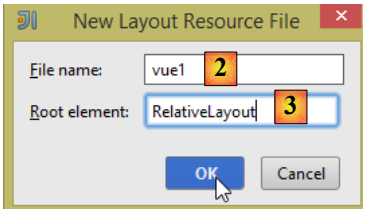




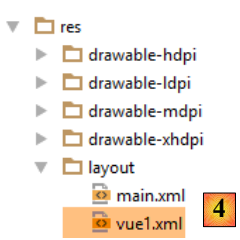
1.5.2 Construire une vue

Nous allons maintenant modifier, avec l'éditeur graphique, la vue affichée par le module [exemple-04] :





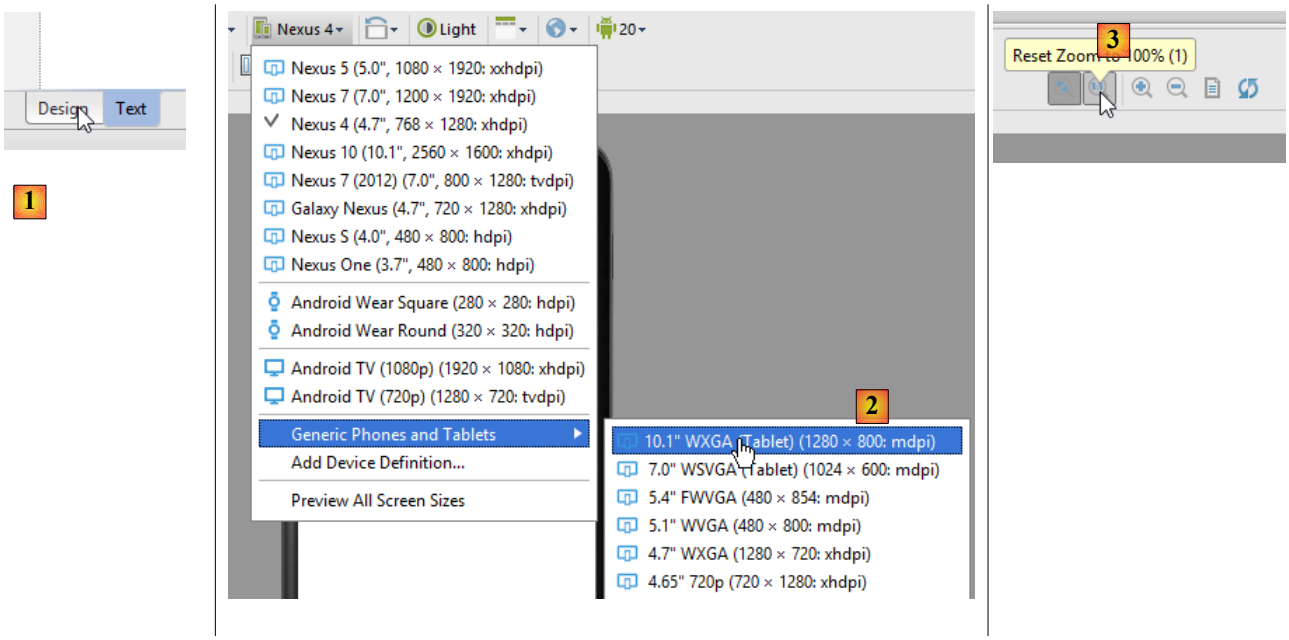
- en [1], créez une nouvelle vue XML [clic droit sur layout / New / Layout resource File] ;
- en [2], nommez la vue ;
- en [3], indiquez la balise racine de la vue. Ici, nous choisissons un conteneur [RelativeLayout]. Dans ce conteneur de composants, ceux-ci sont placés les uns par rapport aux autres : " à droite de ", " à gauche de ", " au-dessous de ", " au-dessus de " ;



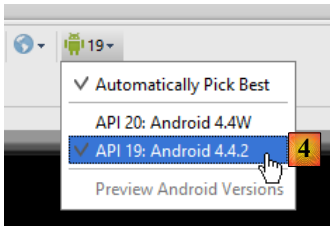
Le fichier [vue1.xml] généré [4] est le suivant :

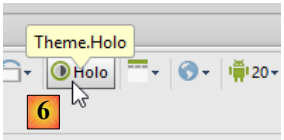
```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.                  android:layout_width="match_parent"
4.                  android:layout_height="match_parent">
5.
6.  </RelativeLayout>
```

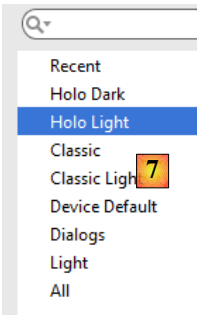
- ligne 2 : un conteneur [RelativeLayout] vide qui occupera toute la largeur de la tablette (ligne 3) et toute sa hauteur (ligne 4) ;



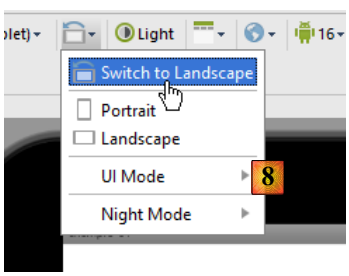
- en [1], sélectionnez l'onglet [Design] dans la vue [vue1.xml] affichée ;
- en [2], mettez-vous en mode tablette ;
- en [3], mettez-vous à l'échelle 1 de la tablette ;



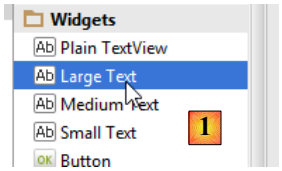


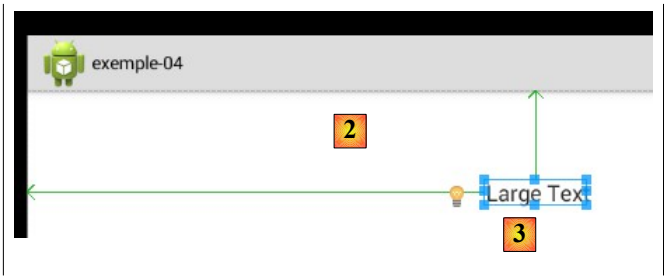


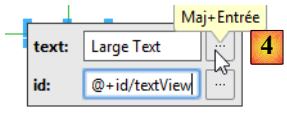
- en [4], choisissez l'API 19 car avec l'API 20, la vue n'est pas affichée ;
- en [6], changez le style des vues (Holo : blanc sur fond noir) ;
- en [7], choisissez le style [Holo Light] (noir sur fond blanc) ;



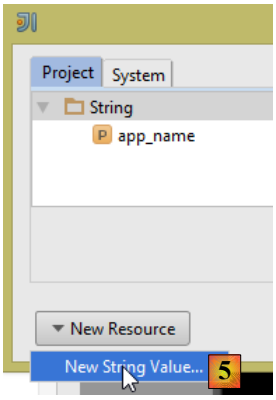
- en [8], choisissez le mode 'paysage' pour la tablette ;
- la copie d'écran [9] résume les choix faits.

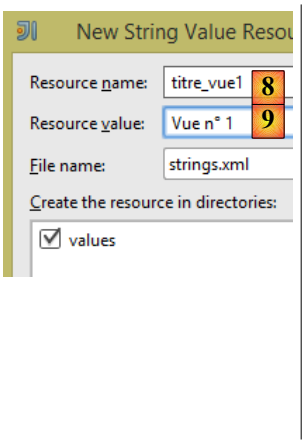


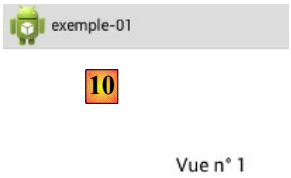




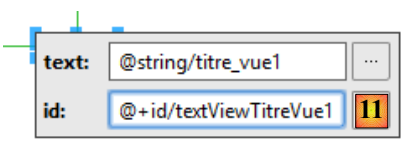
- en [1], prendre un [Large Text] et le tirer sur la vue [2] ;
- en [3], double-cliquer sur le composant ;
- en [4], modifier le texte affiché. Plutôt que de le mettre en 'dur' dans la vue XML, nous allons l'externaliser dans le fichier [res / values / string.xml]

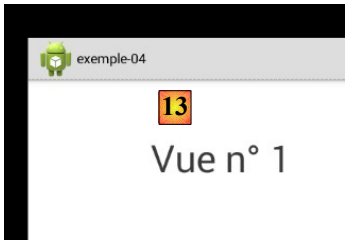






- en [5], on ajoute une nouvelle valeur dans le fichier [strings.xml] ;
- en [8], on donne un identifiant à la chaîne ;
- en [9], on donne la valeur de la chaîne ;
- en [10], la nouvelle vue après validation de l'étape précédente ;





- après un double clic sur le composant, on change son identifiant [11] ;
- en [12], dans les propriétés du composant, on change la taille des caractères [50sp] ;
- en [13], la nouvelle vue ;

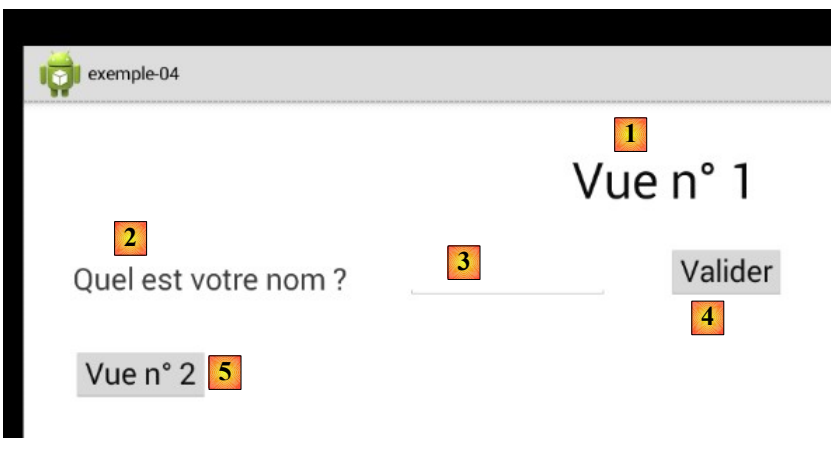
Le fichier [vue1.xml] a évolué comme suit :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.
3.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
4.                  android:layout_width="match_parent"
5.                  android:layout_height="match_parent">
6.
7.    <TextView
8.      android:layout_width="wrap_content"
9.      android:layout_height="wrap_content"
10.     android:textAppearance="?android:attr/textAppearanceLarge"
11.     android:text="@string/titre_vue1"
12.     android:id="@+id/textViewTitreVue1"
13.     android:layout_marginTop="121dp"
14.     android:layout_alignParentTop="true"
15.     android:layout_alignParentLeft="true"
16.     android:layout_marginLeft="201dp"
17.     android:textSize="50sp"/>
18. </RelativeLayout>
```

- les modifications faites dans l'interface graphique sont aux lignes 11, 12 et 17. Les autres attributs du [TextView] sont des valeurs par défaut ou bien découlent du positionnement du composant dans la vue ;
- lignes 8-9 : la taille du composant est celle du texte qu'elle contient (wrap_content) en hauteur et largeur ;
- lignes 13-14 : le haut du composant est aligné avec le haut de la vue (ligne 14), 121 pixels dessous (ligne 13) ;
- lignes 10-12 : le côté gauche du composant est aligné avec la gauche de la vue (ligne 15), 201 pixels à droite (ligne 16) ;

En général, les tailles exactes des marges gauche, droite, haute et basse seront fixées directement dans le XML.

En procédant de la même façon, créez la vue suivante [1] :

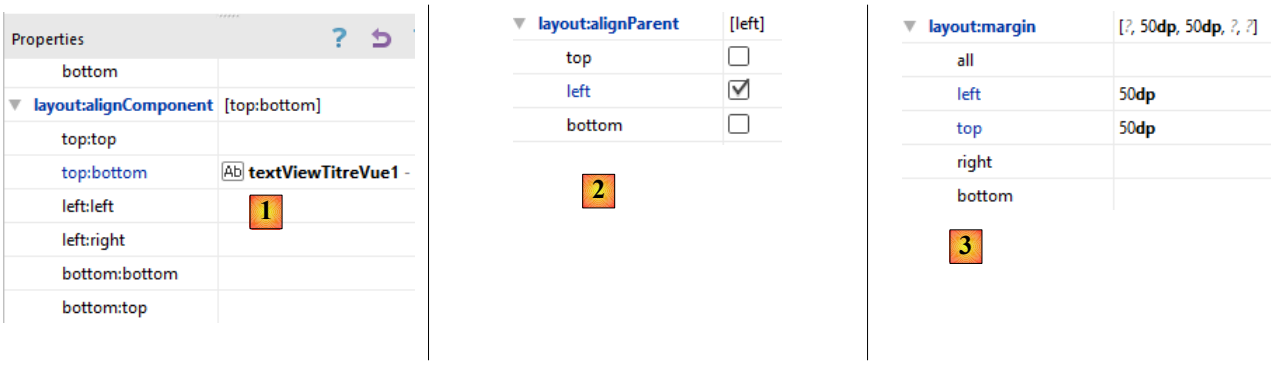


Les composants sont les suivants :

N°	Id	Type	Rôle
1	*textViewTitreVue1*	TextView	Titre de la vue
2	*textView1*	TextView	une question
3	*editTextNom*	EditText	saisie d'un nom
4	*buttonValider*	Button	pour valider la saisie
5	*buttonVue2*	Button	pour passer à la vue n° 2

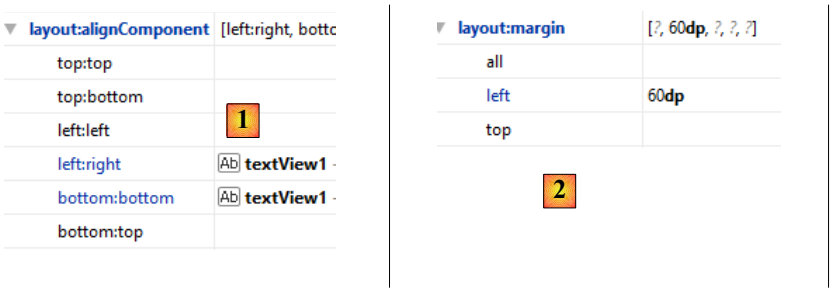
Placer les composants les uns par rapport aux autres est un exercice qui peut s'avérer frustrant, les réactions de l'éditeur graphique étant parfois surprenantes. Il peut être préférable d'utiliser les propriétés des composants :

Le composant [*textView1*] doit être placé à 50 pixels sous le titre et à 50 pixels du bord gauche du conteneur :



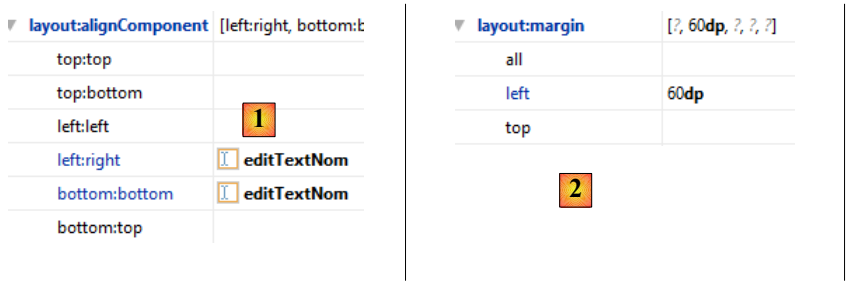
- en [1], le bord supérieur (top) du composant est aligné par rapport au bord inférieur (bottom) du composant [*textViewTitreVue1*] à une distance de 50 pixels (top) ;
- en [2], le bord gauche (left) du composant est aligné par rapport au bord gauche du conteneur à une distance de 50 pixels [3] (left) ;

Le composant [*editTextNom*] doit être placé à 60 pixels à droite du composant [*textView1*] et aligné par le bas sur ce même composant ;



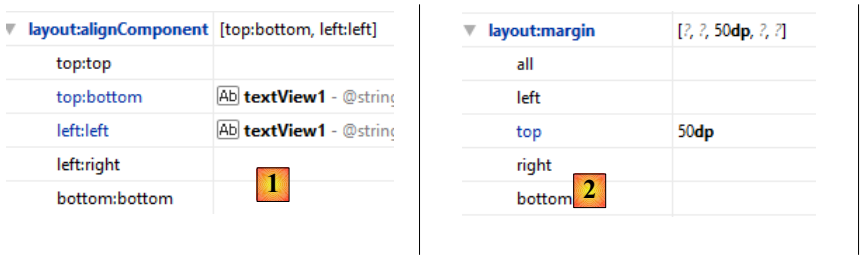
- en [1], le bord gauche (left) du composant est aligné par rapport au bord droit (right) du composant [*textView1*] à une distance de 60 pixels [2] (left). Il est aligné sur le bord inférieur (bottom:bottom) du composant [*textView1*] [1] ;

Le composant [*buttonValider*] doit être placé à 60 pixels à droite du composant [*editTextNom*] et aligné par le bas sur ce même composant ;



- en [1], le bord gauche (left) du composant est aligné par rapport au bord droit (right) du composant [*editTextNom*] à une distance de 60 pixels [2] (left). Il est aligné sur le bord inférieur du composant (bottom:bottom) [*editTextNom*] [1] ;

Le composant [*buttonVu2*] doit être placé à 50 pixels sous le composant [*textView1*] et aligné par la gauche sur ce même composant ;



- en [1], le bord gauche (left) du composant est aligné par rapport au bord gauche (left) du composant [*textView1*] et est placé dessous (top:bottom) à une distance de 50 pixels [2] (top) ;

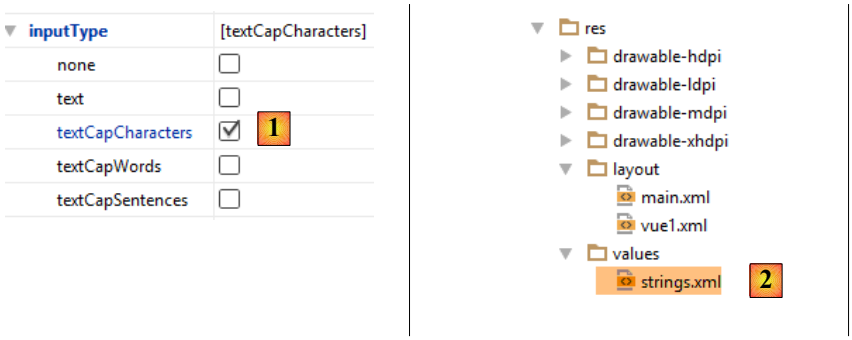
Le fichier XML généré est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.
3.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
4.                  android:layout_width="match_parent"
5.                  android:layout_height="match_parent">
6.
7.    <TextView
8.      android:layout_width="wrap_content"
9.      android:layout_height="wrap_content"
10.     android:textAppearance="?android:attr/textAppearanceLarge"
11.     android:text="@string/titre_vue1"
12.     android:id="@+id/textViewTitreVue1"
13.     android:layout_marginTop="49dp"
14.     android:textSize="50sp"
15.     android:layout_gravity="center|left"
16.     android:layout_alignParentTop="true"
17.     android:layout_centerHorizontal="true"/>
18.
19.   <TextView
20.     android:layout_width="wrap_content"
21.     android:layout_height="wrap_content"
22.     android:text="@string/txt_nom"
23.     android:id="@+id/textView1"
24.     android:layout_below="@+id/textViewTitreVue1"
25.     android:layout_alignParentLeft="true"
26.     android:layout_marginLeft="50dp"
27.     android:layout_marginTop="50dp"
28.     android:textSize="30sp"/>
29.
30.   <EditText
31.     android:layout_width="wrap_content"
32.     android:layout_height="wrap_content"
33.     android:id="@+id/editTextNom"
34.     android:inputType="textCapCharacters"
35.     android:minWidth="200dp"
36.     android:layout_toRightOf="@+id/textView1"
37.     android:layout_marginLeft="60dp"
38.     android:layout_alignBottom="@+id/textView1"/>
39.
40.   <Button
41.     android:layout_width="wrap_content"
42.     android:layout_height="wrap_content"
43.     android:text="@string/btn_valider"
44.     android:id="@+id/buttonValider"
45.     android:layout_alignBottom="@+id/editTextNom"
46.     android:layout_toRightOf="@+id/editTextNom"
47.     android:textSize="30sp"
48.     android:layout_marginLeft="60dp"/>
49.
50.   <Button
```

```
51.     android:layout_width="wrap_content"
52.     android:layout_height="wrap_content"
53.     android:text="@string/btn_vue2"
54.     android:id="@+id/buttonVue2"
55.     android:layout_below="@+id/textView1"
56.     android:layout_alignLeft="@+id/textView1"
57.     android:layout_marginTop="50dp"
58.     android:textSize="30sp"/>
59.
60. </RelativeLayout>
```

On y retrouve tout ce qui a été fait de façon graphique. Une autre façon de créer une vue est alors d'écrire directement ce fichier. Lorsqu'on est habitué, cela peut être plus rapide que d'utiliser l'éditeur graphique.

- ligne 34, on trouve une information que nous n'avons pas montrée. Elle est donnée via les propriétés du composant [*editTextNom*] [1] :



Tous les textes proviennent du fichier [strings.xml] [2] suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.    <string name="app_name">exemple-01</string>
4.    <string name="titre_vue1">Vue n° 1</string>
5.    <string name="txt_nom">Quel est votre nom ?</string>
6.    <string name="btn_valider">Valider</string>
7.    <string name="btn_vue2">Vue n° 2</string>
8.  </resources>
```

Maintenant, modifions l'activité [MainActivity] pour que cette vue soit affichée au démarrage de l'application :

```
1.  package android.exemples;
2.
3.  import android.app.Activity;
4.  import android.os.Bundle;
5.  import org.androidannotations.annotations.EActivity;
6.
7.  @EActivity(R.layout.vue1)
8.  public class MyActivity extends Activity {
9.    @Override
10.   public void onCreate(Bundle savedInstanceState) {
11.     super.onCreate(savedInstanceState);
12.   }
13. }
```

- ligne 7 : c'est la vue [vue1.xml] qui est désormais affichée par l'activité ;

Modifiez le fichier [AndroidManifest.xml] de la façon suivante :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.            package="android.exemples"
4.            android:versionCode="1"
5.            android:versionName="1.0">
6.    <uses-sdk
7.      android:minSdkVersion="11"
8.      android:targetSdkVersion="20"/>
```

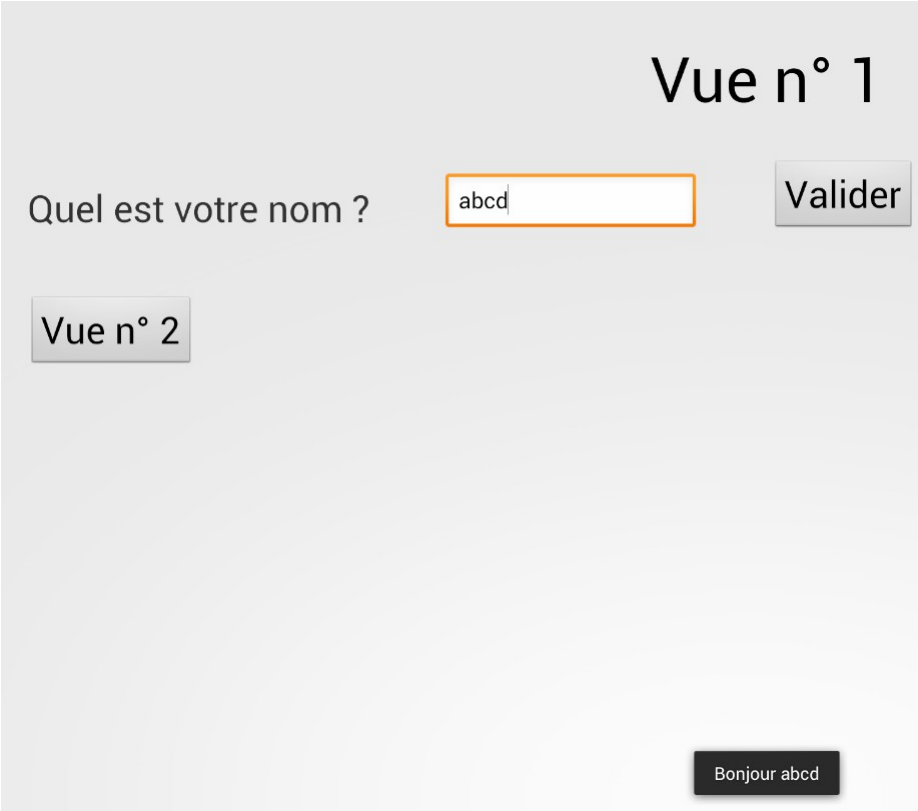
```
9.    <application
10.     android:label="@string/app_name"
11.     android:icon="@drawable/ic_launcher"
12.     android:theme="@android:style/Theme.Light">
13.     <activity android:name="MyActivity_"
14.               android:label="@string/app_name"
15.               android:windowSoftInputMode="stateHidden">
16.       <intent-filter>
17.         <action android:name="android.intent.action.MAIN"/>
18.         <category android:name="android.intent.category.LAUNCHER"/>
19.       </intent-filter>
20.     </activity>
21.   </application>
22. </manifest>
```

- ligne 12 : modifier le thème de l'application en choisissant le thème [Light] (textes noirs sur fond blanc) ;
- ligne 15 : cette ligne de configuration empêche le clavier d'apparaître dès l'affichage de la vue [vue1]. En effet celle-ci a un champ de saisie qui a le focus lors de l'affichage de la vue. Ce focus fait apparaître par défaut le clavier virtuel ;

Exécutez l'application et vérifiez que c'est bien la vue [vue1.xml] qui est affichée :

1.5.3 Gestion des événements

Gérons maintenant le clic sur le bouton [Valider] de la vue [Vue1] :



Le code de [MainActivity] évolue comme suit :

```
1.  package android.exemples;
2.
3.  import android.app.Activity;
4.  import android.os.Bundle;
5.  import android.widget.EditText;
6.  import android.widget.Toast;
7.  import org.androidannotations.annotations.Click;
8.  import org.androidannotations.annotations.EActivity;
9.  import org.androidannotations.annotations.ViewById;
10.
11. @EActivity(R.layout.vue1)
12. public class MyActivity extends Activity {
13.
14.   // les éléments de l'interface visuelle
15.   @ViewById(R.id.editTextNom)
16.   protected EditText editTextNom;
17.
18.   @Override
19.   public void onCreate(Bundle savedInstanceState) {
20.     super.onCreate(savedInstanceState);
21.   }
22.
23.   // gestionnaire d'évt
24.   @Click(R.id.buttonValider)
25.   protected void doValider() {
26.     // on affiche le nom saisi
27.     Toast.makeText(this, String.format("Bonjour %s", editTextNom.getText().toString()),
    Toast.LENGTH_LONG).show();
28.   }
29. }
```

- lignes 15-16 : on associe le champ [**protected** EditText editTextNom] au composant d'identifiant [R.id.editTextNom] de l'interface visuelle. Le champ associé au composant doit être accessible dans la classe dérivée [MyActivity_] et pour cette raison ne peut être de portée [private]. Le champ identifié par [R.id.editTextNom] provient de la vue [vue1.xml] :

```
<EditText
  android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:id="@+id/editTextNom"
    android:minWidth="200dp"
    android:layout_toRightOf="@+id/textView1"
    android:layout_marginLeft="60dp"
    android:layout_alignBottom="@+id/textView1"
    android:inputType="textCapCharacters"/>
```

- ligne 24 : l'annotation [@Click(R.id.buttonValider)] désigne la méthode qui gère l'événement 'Click' sur le bouton d'identifiant [R.id.buttonValider]. Cet identifiant provient également de la vue [vue1.xml] :

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/btn_valider"
    android:id="@+id/buttonValider"
    android:layout_alignBottom="@+id/editTextNom"
    android:layout_toRightOf="@+id/editTextNom"
    android:textSize="30sp"
    android:layout_marginLeft="60dp"/>
```

- ligne 27 : affiche le nom saisi :
    - *Toast.makeText(...).show()* : affiche un texte à l'écran,
    - le 1er paramètre de *makeText* est l'activité,
    - le second paramètre est le texte à afficher dans la boîte qui va être affichée par *makeText,*
    - le troisième paramètre est la durée de vie de la boîte affichée : *Toast.LENGTH_LONG* ou *Toast.LENGTH_SHORT* ;

Exécutez le module [exemple-04] et vérifiez qu'il se passe quelque chose lorsque vous cliquez sur le bouton [Validez].

1.6 Exemple-05 : navigation entre vues

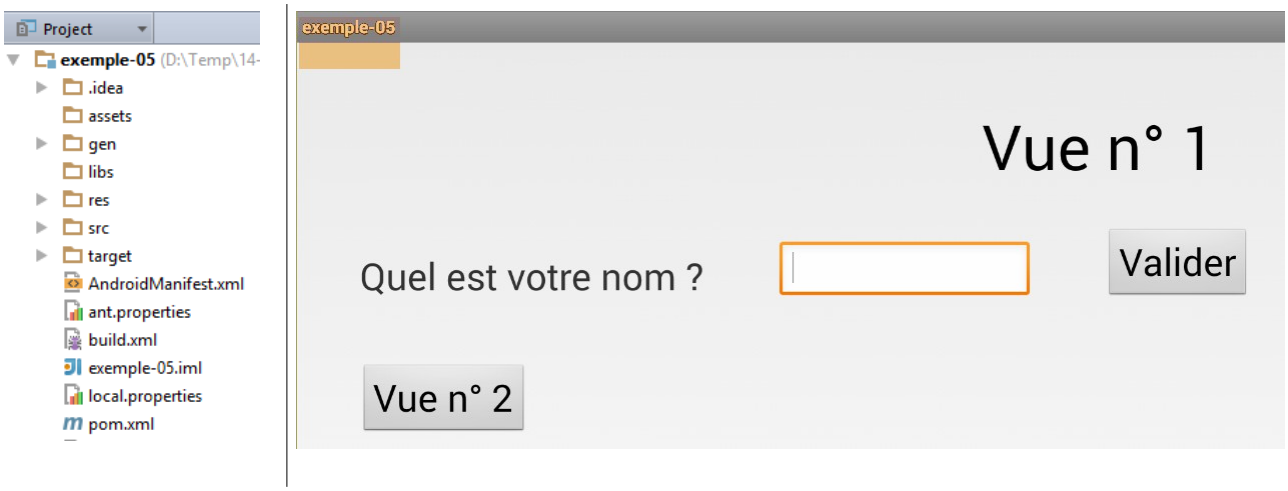
Dans le projet précédent, le bouton [Vue n° 2] n'a pas été exploité. On se propose de l'exploiter en créant une seconde vue et en montrant comment naviguer d'une vue à l'autre. Il y a plusieurs façons de résoudre ce problème. Celle qui est proposée ici est d'associer chaque vue à une activité. Une autre méthode est d'avoir une unique activité de type [FragmentActivity] qui affiche des vues de type [Fragment]. Ce sera la méthode utilisée dans des applications à venir.

1.6.1 Création du projet

On suivra la procédure décrite pour dupliquer [exemple-02] dans [exemple-03] au paragraphe 1.4, page 27 :

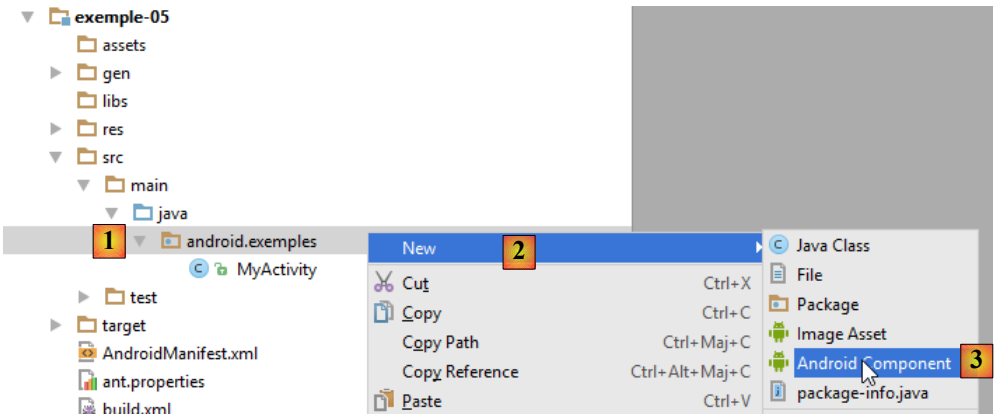
Nous :

- dupliquons le module [exemple-04] dans [exemple-05] (après avoir supprimé le dossier [target] de [exemple-04]) ;
- chargeons le module [exemple-045 ;
- changeons le nom du projet et du module en [exemple-05] (structure du projet) ;
- changeons le nom du projet dans les fichiers [pom.xml, strings.xml] ;
- rafraîchissons le projet Maven ;
- le compilons ;
- créons une configuration d'exécution nommée [exemple-05] ;
- exécutons celle-ci ;

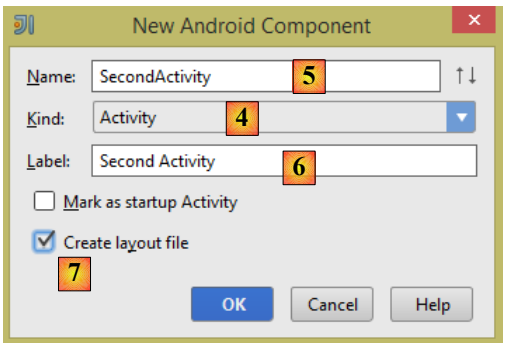


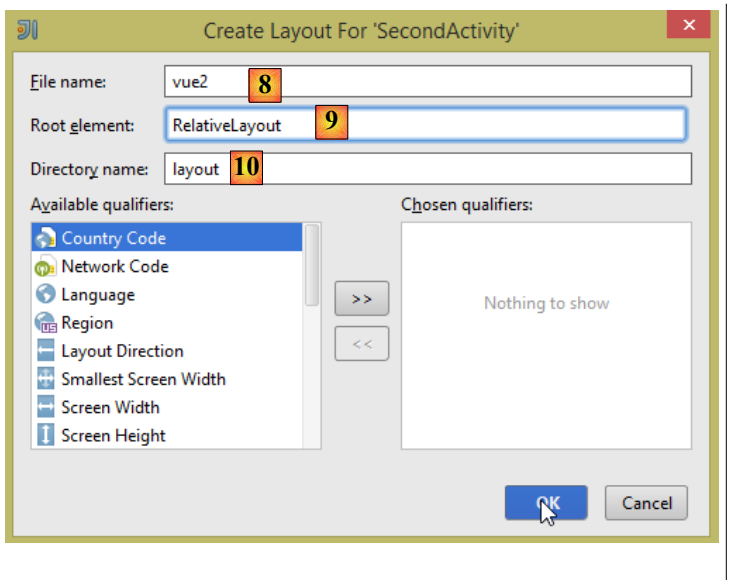
1.6.2 Ajout d'une seconde activité

Pour gérer une seconde vue, nous allons créer une seconde activité. C'est elle qui gèrera la vue n° 2. On est là dans un modèle **une vue = une activité**. Il y a d'autres modèles possibles.

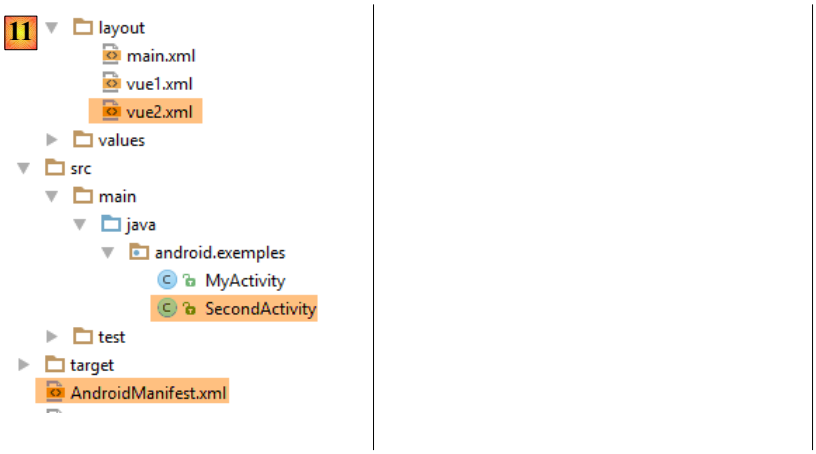


- en [1-3], on crée un nouveau composant Android ;





- en [4], ce nouveau composant Android sera une activité ;
- en [5], le nom de la classe qui sera générée ;
- en [6], le label de l'activité. Il sera placé dans la barre de titre de la fenêtre associée à l'activité ;
- en [7], on demande la création d'une vue qui sera associée à cette nouvelle activité ;
- en [8], le nom de la vue (vue2.xml) ;
- en [9], le type de mise en forme utilisée pour la disposition des composants sur la vue ;
- en [10], le dossier où sera placée la vue ;



- en [11], les fichiers impactés par la configuration précédente ;

L'activité [SecondActivity] est la suivante :

```
1.  package android.exemples;
2.
3.  import android.app.Activity;
4.  import android.os.Bundle;
5.
6.  public class SecondActivity extends Activity {
7.    @Override
8.    public void onCreate(Bundle savedInstanceState) {
9.      super.onCreate(savedInstanceState);
10.     setContentView(R.layout.vue2);
11.   }
12. }
```

- ligne 10 : l'activité est associée à la vue [vue2.xml] ;

La vue [vue2.xml] est la suivante :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.
3.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
4.                  android:layout_width="match_parent"
5.                  android:layout_height="match_parent">
6.
7.  </RelativeLayout>
```

C'est donc une vue vide avec un gestionnaire de disposition de type [RelativeLayout] (ligne 3).

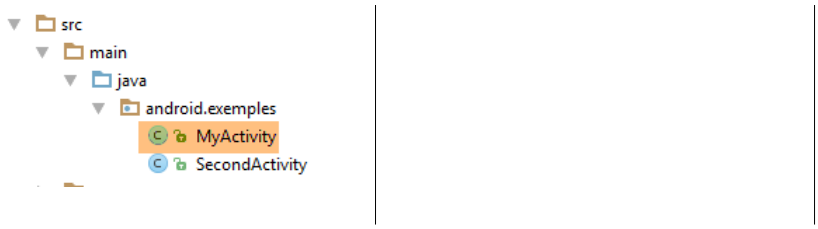
Le manifeste du module Android [AndroidManifest.xml] évolue de la façon suivante :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.            package="android.exemples"
4.            android:versionCode="1"
5.            android:versionName="1.0">
6.    <uses-sdk
7.      android:minSdkVersion="11"
8.      android:targetSdkVersion="20"/>
9.    <application
10.     android:label="@string/app_name"
11.     android:icon="@drawable/ic_launcher"
12.     android:theme="@android:style/Theme.Light">
13.     <activity android:name="MyActivity_"
14.               android:label="@string/app_name"
15.               android:windowSoftInputMode="stateHidden">
16.       <intent-filter>
17.         <action android:name="android.intent.action.MAIN"/>
18.         <category android:name="android.intent.category.LAUNCHER"/>
19.       </intent-filter>
20.     </activity>
21.     <activity
22.       android:name=".SecondActivity"
23.       android:label="Second Activity"/>
24.   </application>
25. </manifest>
```

Aux lignes 21-23, une seconde activité est enregistrée avec les informations données dans l'assistant de création de celle-ci.

1.6.3 Navigation de la vue n° 1 à la vue n° 2

Revenons au code de la classe [MainActivity] qui affiche la vue ° 1. Le passage à la vue n° 2 n'est actuellement pas géré :



Nous la gérons de la façon suivante :

```
1.  // naviguer vers la vue n° 2
2.    @Click(R.id.buttonVue2)
3.    protected void navigateToView2() {
4.      // on navigue vers la vue n° 2 en lui passant le nom saisi dans la vue n° 1
5.      // on crée un Intent
6.      Intent intent = new Intent();
7.      // on associe cet Intent à une activité
8.      intent.setClass(this, SecondActivity.class);
9.      // on associe des informations à cet Intent
10.     intent.putExtra("NOM", editTextNom.getText().toString().trim());
```

```
11.     // on lance l'Intent, ici une activité de type [SecondActivity]
12.     startActivity(intent);
13.   }
```

- lignes 2-3 : la méthode [navigateToView2] gère le 'clic' sur le bouton identifié par [R.id.buttonVue2] défini dans la vue [vue1.xml] :

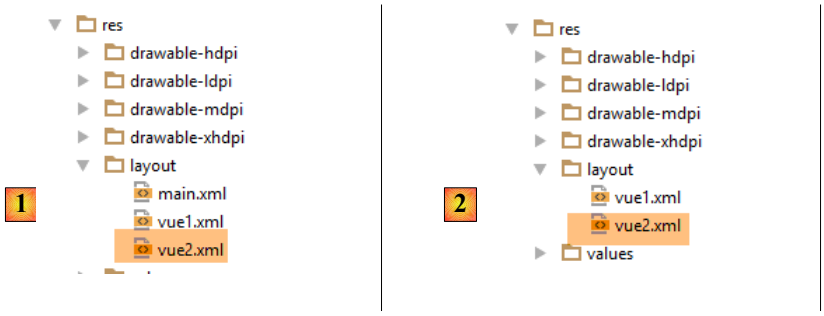
```
1.    <Button
2.      android:layout_width="wrap_content"
3.      android:layout_height="wrap_content"
4.      android:text="@string/btn_vue2"
5.      android:id="@+id/buttonVue2"
6.      android:layout_below="@+id/textView1"
7.      android:layout_alignLeft="@+id/textView1"
8.      android:layout_marginTop="50dp"
9.  android:textSize="30sp"/>
```

Les commentaires décrivent les étapes à réaliser pour le changement de vue :

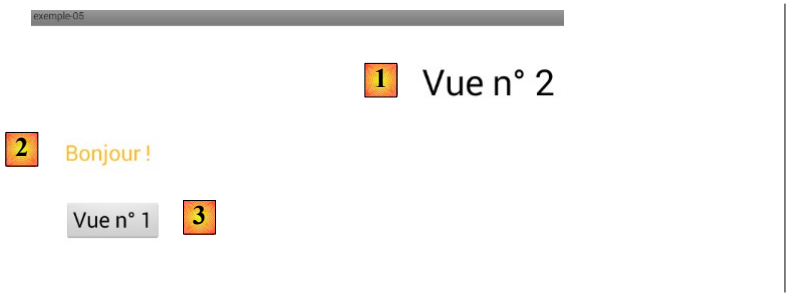
1. ligne 6 : créer un objet de type [Intent]. Cet objet va permettre de préciser et l'activité à lancer et les informations à lui passer ;
2. ligne 8 : associer l'Intent à une activité, ici une activité de type [SecondActivity] qui sera chargée d'afficher la vue n° 2. Il faut se souvenir que l'activité [MainActivity] affiche elle la vue n° 1. Donc on a **une vue = une activité**. Il nous faudra définir le type [SecondActivity] ;
3. ligne 10 : de façon facultative, mettre des informations dans l'objet [Intent]. Celles-ci sont destinées à l'activité [SecondActivity] qui va être lancée. Les paramètres de [Intent.putExtra] sont (Object clé , Object valeur). On notera que la méthode [EditText.getText()] qui rend le texte saisi dans la zone de saisie ne rend pas un type [String] mais un type [Editable]. Il faut utiliser la méthode [toString] pour avoir le texte saisi ;
4. ligne 12 : lancer l'activité définie par l'objet [Intent].

Exécutez le projet [exemple-05] et vérifiez que vous obtenez bien la vue n° 2 :

1.6.4 Construction de la vue n° 2



- en [1-2], nous supprimons la vue [main.xml] qui ne nous sert plus, puis nous modifions la vue [vue2.xml] de la façon suivante :



Les composants sont les suivants :

N°	Id	Type	Rôle
1	*textViewTitreVue2*	TextView	Titre de la vue
2	*textViewBonjour*	TextView	un texte
5	*btn_vue1*	Button	pour passer à la vue n° 1

Le fichier XML [vue2.xml] est le suivant :

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
                android:layout_width="match_parent"
                android:layout_height="match_parent">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="@string/titre_vue2"
    android:id="@+id/textViewTitreVue2"
    android:layout_marginTop="50dp"
    android:textSize="50sp"
    android:layout_gravity="center|left"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"/>

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textViewBonjour"
    android:layout_centerVertical="true"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textViewTitreVue2"
    android:layout_marginTop="50dp"
    android:layout_marginLeft="50dp"
    android:textSize="30sp"
    android:text="Bonjour !"
    android:textColor="#ffffb91b"/>

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/btn_vue1"
    android:id="@+id/buttonVue1"
    android:layout_marginTop="50dp"
    android:textSize="30sp"
    android:layout_alignLeft="@+id/textViewBonjour"
    android:layout_below="@+id/textViewBonjour"/>

</RelativeLayout>
```

Exécutez le projet [exemple-05] et vérifiez que vous obtenez bien la nouvelle vue en cliquant sur le bouton [Vue n° 2].

1.6.5 L'activité [SecondActivity]

Dans [MainActivity], nous avons écrit le code suivant :

```
    // naviguer vers la vue n° 2
```

```
2.      protected void navigateToView2() {
3.         // on navigue vers la vue n° 2 en lui passant le nom saisi dans la vue n° 1
4.         // on crée un Intent
5.         Intent intent = new Intent();
6.         // on associe cet Intent à une activité
7.         intent.setClass(this, SecondActivity.class);
8.         // on associe des informations à cet Intent
9.         intent.putExtra("NOM", edtNom.getText().toString().trim());
10.        // on lance l'Intent, ici une activité de type [SecondActivity]
11.        startActivity(intent);
12. }
```

Ligne 9, nous avons mis pour [SecondActivity] des informations qui n'ont pas été exploitées. Nous les exploitons maintenant et cela se passe dans le code de [SecondActivity] :

Le code de [SecondActivity] évolue comme suit :

```
1.  package android.exemples;
2.
3.  import android.app.Activity;
4.  import android.content.Intent;
5.  import android.os.Bundle;
6.  import android.widget.TextView;
7.  import org.androidannotations.annotations.AfterViews;
8.  import org.androidannotations.annotations.Click;
9.  import org.androidannotations.annotations.EActivity;
10. import org.androidannotations.annotations.ViewById;
11.
12. @EActivity(R.layout.vue2)
13. public class SecondActivity extends Activity {
14.
15.   // composants de l'interface visuelle
16.   @ViewById
17.   protected TextView textViewBonjour;
18.
19.   @Override
20.   public void onCreate(Bundle savedInstanceState) {
21.     super.onCreate(savedInstanceState);
22.   }
23.
24.   @AfterViews
25.   protected void initActivity() {
26.     // on récupère l'intent s'il existe
27.     Intent intent = getIntent();
28.     if (intent != null) {
29.       Bundle extras = intent.getExtras();
30.       if (extras != null) {
31.         // on récupère le nom
32.         String nom = extras.getString("NOM");
33.         if (nom != null) {
34.           // on l'affiche
35.           textViewBonjour.setText(String.format("Bonjour %s !", nom));
36.         }
37.       }
38.     }
39.   }
40.
41. }
```

- ligne 12 : on utilise l'annotation [@EActivity] pour indiquer que la classe [SecondActivity] est une activité associée à la vue [vue2.xml] ;

- ligne 16 : on récupère une référence sur le composant [TextView] identifié par [R.id.textViewBonjour]. Ici, on n'a pas écrit [@ViewById(R.id.textViewBonjour)]. Dans ce cas, AA suppose que l'identifiant du composant est identique au champ annoté, ici le champ [textViewBonjour] ;
- ligne 24 : l'annotation [@AfterViews] annote une méthode qui doit être exécutée après que les champs annotés par [@ViewById] ont été initialisés. Dans la méthode [OnCreate] (ligne 20), on ne peut pas utiliser ces champs car ils n'ont pas encore été initialisés. Dans le projet [exemple-05], on bascule d'une activité à une autre et il n'était a priori pas clair si la méthode annotée [@AfterViews] allait être exécutée une fois à l'instanciation initiale de l'activité ou à chaque fois que l'activité est démarrée. Les tests ont montré que la seconde hypothèse était vérifiée ;
- ligne 27 : la classe [Activity] a une méthode [getIntent] qui rend l'objet [Intent] associé à l'activité ;
- ligne 29 : la méthode [Intent.getExtras] rend un type [Bundle] qui est une sorte de dictionnaire contenant les informations associées à l'objet [Intent] de l'activité ;
- ligne 32 : on récupère le nom placé dans l'objet [Intent] de l'activité ;
- ligne 35 : on l'affiche.

Revenons sur la classe [SecondActivity]. Parce que nous avons écrit :

```
@EActivity(R.layout.vue2)
public class SecondActivity extends Activity {
```

AA va générer une classe [SecondActivity_] dérivée de [SecondActivity] et c'est cette classe qui sera la véritable activité. Cela nous amène à faire des modifications dans :

[MyActivity]

```
1.    // naviguer vers la vue n° 2
2.    @Click(R.id.buttonVue2)
3.    protected void navigateToView2() {
4.  ..
5.      // on associe cet Intent à une activité
6.      intent.setClass(this, SecondActivity_.class);
7.      ...
8.  }
```

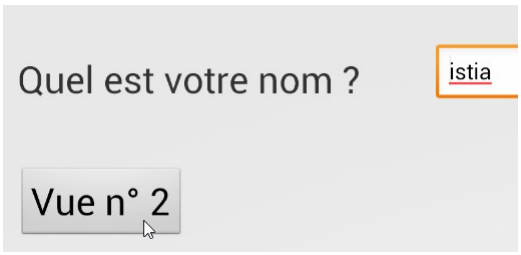
- ligne 6, on doit remplacer [SecondActivity] par [SecondActivity_] ;

[AndroidManifest.xml]

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.            package="android.exemples"
4.            android:versionCode="1"
5.            android:versionName="1.0">
6.    <uses-sdk
7.      android:minSdkVersion="11"
8.      android:targetSdkVersion="20"/>
9.    <application
10.     android:label="@string/app_name"
11.     android:icon="@drawable/ic_launcher"
12.     android:theme="@android:style/Theme.Light">
13.     <activity android:name="MyActivity_"
14.               android:label="@string/app_name"
15.               android:windowSoftInputMode="stateHidden">
16.       <intent-filter>
17.         <action android:name="android.intent.action.MAIN"/>
18.         <category android:name="android.intent.category.LAUNCHER"/>
19.       </intent-filter>
20.     </activity>
21.     <activity
22.       android:name=".SecondActivity_"
23.       android:label="Second Activity"/>
24.   </application>
25. </manifest>
```

- ligne 22, on doit remplacer [SecondActivity] par [SecondActivity_] ;

Testez cette nouvelle version. Tapez un nom dans la vue n° 1 et vérifiez que la vue n° 2 l'affiche bien.



1.6.6 Navigation de la vue n° 2 vers la vue n° 1

Pour naviguer de la vue n° 1 à la vue n° 2 nous allons suivre la procédure vue précédemment :

- mettre le code de navigation dans l'activité [SecondActivity] qui affiche la vue n° 2 ;
- écrire la méthode [@AfterViews] dans l'activité [MyActivity] qui affiche la vue n° 1 ;

Le code de [SecondActivity] évolue comme suit :

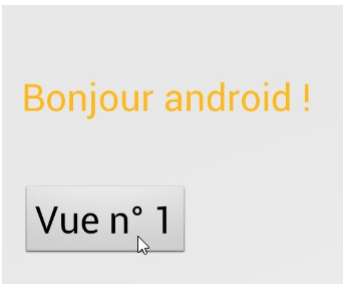
```
1.    @Click(R.id.buttonVue1)
2.    protected void navigateToView1() {
3.      // on crée un Intent pour l'activité [MyActivity_]
4.      Intent intent1 = new Intent();
5.      intent1.setClass(this, MyActivity_.class);
6.      // on récupère l'Intent de l'activité courante [SecondActivity]
7.      Intent intent2 = getIntent();
8.      if (intent2 != null) {
9.        Bundle extras2 = intent2.getExtras();
10.       if (extras2 != null) {
11.         // on met le nom dans l'Intent de [MyActivity_]
12.         intent1.putExtra("NOM", extras2.getString("NOM"));
13.       }
14.       // on lance [MyActivity_]
15.       startActivity(intent1);
16.     }
17. }
```

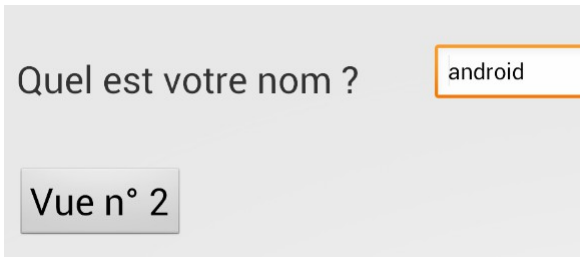
- lignes 1-2 : on associe la méthode [navigateToView1] au clic sur le bouton [btn_vue1] ;
- ligne 4 : on crée un nouvel [Intent] ;
- ligne 5 : associé à l'activité [MyActivity_] ;
- ligne 7 : on récupère l'Intent associé à [SecondActivity] ;
- ligne 9 : on récupère les informations de cet Intent ;
- ligne 12 : la clé [NOM] est récupérée dans [intent2] pour être mise dans [intent1] avec la même valeur associée ;
- ligne 15 : l'activité [MyActivity_] est lancée.

Dans le code de [MyActivity] on ajoute la méthode [@AfterViews] suivante :

```
1.    @AfterViews
2.    protected void initActivity() {
3.      // on récupère l'intent s'il existe
4.      Intent intent = getIntent();
5.      if (intent != null) {
6.        Bundle extras = intent.getExtras();
7.        if (extras != null) {
8.          // on récupère le nom
9.          String nom = extras.getString("NOM");
10.         if (nom != null) {
11.           // on l'affiche
12.           editTextNom.setText(nom);
13.         }
14.       }
15.     }
16. }
```

Faites ces modifications et testez votre application. Maintenant quand on revient de la vue n° 2 à la vue n° 1, on doit retrouver le nom saisi initialement, ce qui n'était pas le cas jusqu'à maintenant.



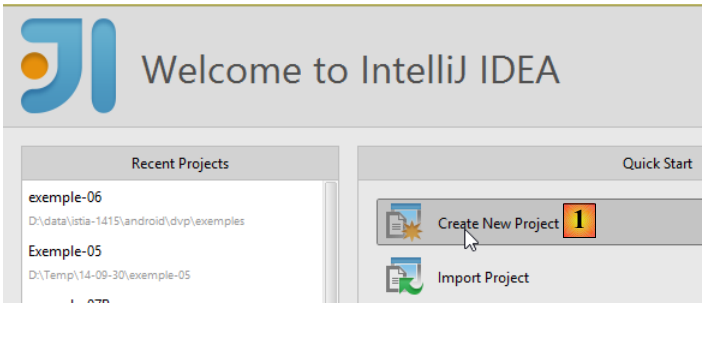


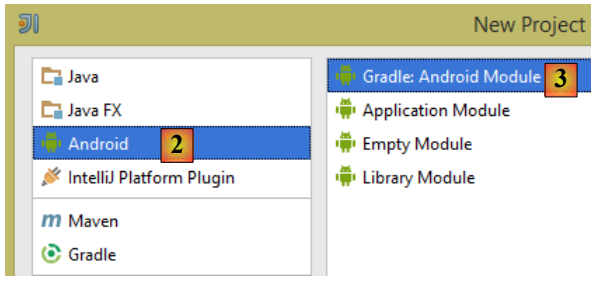
1.7 Exemple-06 : navigation par onglets

Nous allons maintenant explorer les interfaces à onglets.

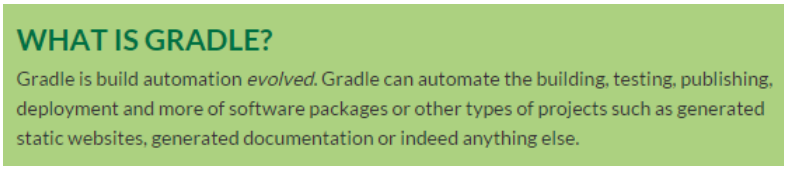
1.7.1 Création du projet

Nous créons un nouveau projet :

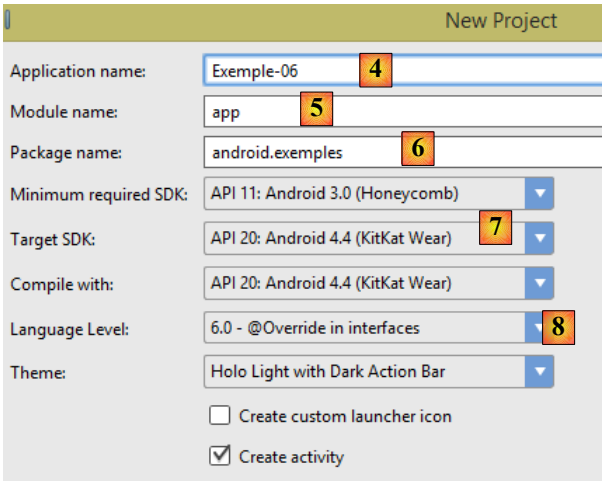


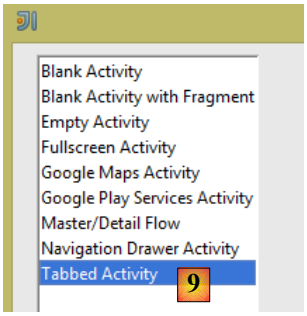


- en [1-3], nous créons un projet de type [Gradle:Android Module] ;

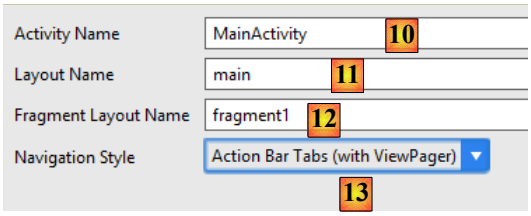


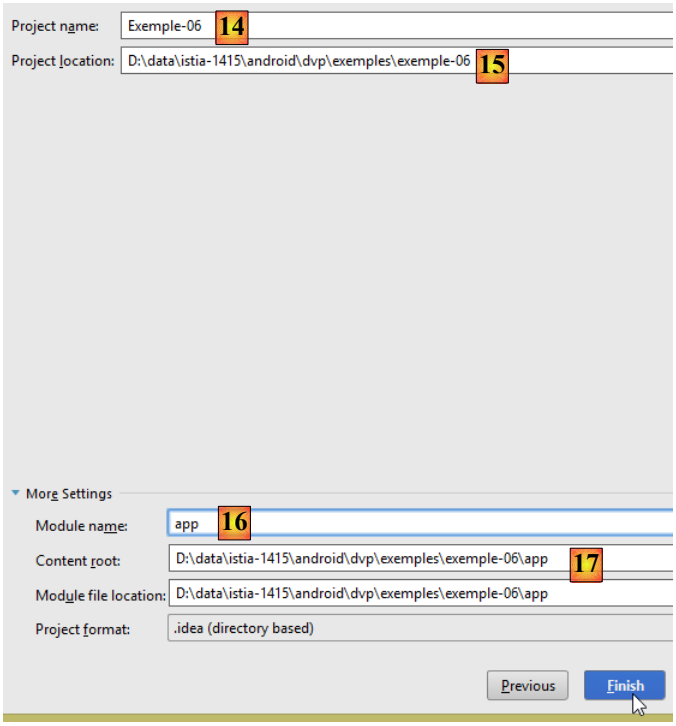
Dans [Android Studio] qui est dérivé de [IntellijIDEA], Google a opté pour [Gradle] plutôt que Maven pour construire les projets Android. [IntellijIDEA] propose également cette solution. Pour nous, le changement visible sera que les dépendances d'un projet seront gérées par [Gradle] plutôt que [Maven] et que l'architecture du projet est alors différente.





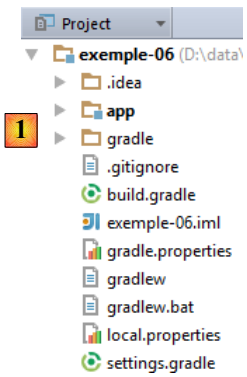
- en [4], on donne un nom au projet ;
- en [5], on donne un nom au module dans ce projet ;
- en [6], on donne un nom au package Android ;
- en [7], on garde les valeurs par défaut ;
- en [8], on choisit un JDK ;
- en [9], on choisit une activité avec onglets ;

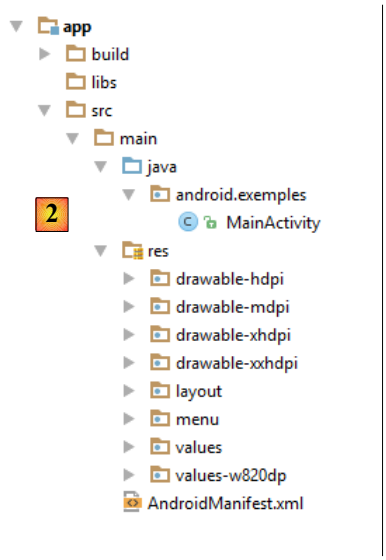




- en [10], on nomme l'activité unique du projet ;
- en [11], la vue associée sera [main.xml]. Cette vue agira comme un patron (template) dans lequel viendront s'insérer des fragments, qui sont des vues partielles ;
- en [12], un fragment ;
- en [13], le type d'onglets désiré ;
- en [14], le nom du projet ;
- en [15], le dossier de celui-ci ;
- en [16], le nom du module de ce projet ;
- en [17], on garde les valeurs proposées ;

Le projet créé est alors le suivant :

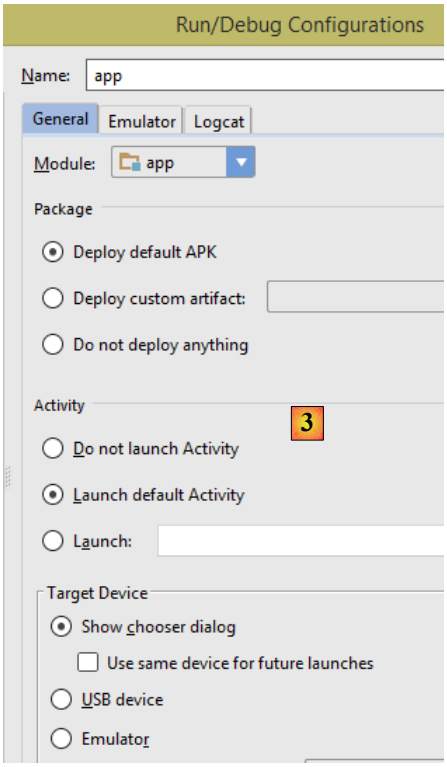


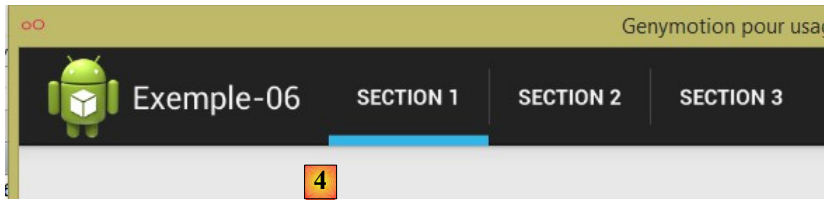


- en [1], on trouve un projet [exemple-06] du nom de son dossier et un module [app] ;

- en [2], le module [app]. C'est là que tout se passe et qu'on retrouve les éléments déjà rencontrés dans les exemples précédents ;

Une configuration d'exécution [app], du nom du module, a été automatiquement créée [3] :

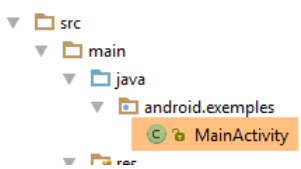




On peut l'exécuter. S'affiche alors une fenêtre avec trois onglets pour l'instant vides [4].

1.7.2 L'activité

Le code généré pour l'activité est assez complexe. C'est une caractéristique de la programmation Android. Tout devient vite assez complexe.



Le code de [MainActivity] est le suivant :

```
1.  package android.exemples;
2.
3.  import android.os.Bundle;
4.  import android.support.v4.app.Fragment;
5.  import android.support.v4.app.FragmentManager;
6.  import android.support.v4.app.FragmentPagerAdapter;
7.  import android.support.v4.app.FragmentTransaction;
8.  import android.support.v4.view.ViewPager;
9.  import android.support.v7.app.ActionBar;
10. import android.support.v7.app.ActionBarActivity;
11. import android.view.*;
12.
13. import java.util.Locale;
14.
```

```
15. public class MainActivity extends ActionBarActivity implements ActionBar.TabListener {
16.
17.   // le gestionnaire de fragments ou sections
18.   SectionsPagerAdapter mSectionsPagerAdapter;
19.
20.   // le conteneur des fragments
21.   ViewPager mViewPager;
22.
23.   @Override
24.   protected void onCreate(Bundle savedInstanceState) {
25.     // classique
26.     super.onCreate(savedInstanceState);
27.     setContentView(R.layout.main);
28.
29.     // la barre d'onglets
30.     final ActionBar actionBar = getSupportActionBar();
31.     actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
32.
33.     // instanciation du gestionnaire de fragments
34.     mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
35.
36.     // on récupère la référence du conteneur de fragments
37.     mViewPager = (ViewPager) findViewById(R.id.pager);
38.     // qu'on associe à notre gestionnaire de fragments
39.     mViewPager.setAdapter(mSectionsPagerAdapter);
40.
41.   // notre gestionnaire de fragments
42.   // à redéfinir pour chaque application
43.   // doit définir les méthodes suivantes
44.   // getItem, getCount, getPageTitle
45.   public class SectionsPagerAdapter extends FragmentPagerAdapter {
46. ...
47.   }
48.
49.   // un fragment est une vue affichée par un conteneur de fragments
50.   public static class PlaceholderFragment extends Fragment {
51.     ...
52.   }
53.
54. }
```

- ligne 15 : l'activité dérive de la classe [ActionBarActivity]. C'est nouveau. Dans les exemples précédents, elle dérivait de la classe [Activity]. La classe [ActionBarActivity] permet de gérer la barre des actions [ActionBar] qui se trouve en haut de la fenêtre de l'activité ;
- ligne 21 : Android fournit un conteneur de vues de type [`android.support.v4.view.ViewPager`] (ligne 8). Il faut fournir à ce conteneur un gestionnaire de vues ou fragments. C'est le développeur qui le fournit ;
- ligne 18 : le gestionnaire de fragments utilisé dans cet exemple. Son implémentation est aux lignes 45-47 ;
- ligne 24 : la méthode exécutée à la création de l'activité ;
- ligne 27 : la vue [main.xml] est associée à l'activité. Cette vue est un conteneur dans lequel vont venir s'afficher des [Fragments] :

```
1.  <android.support.v4.view.ViewPager xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools"
3.      android:id="@+id/pager"
4.      android:layout_width="match_parent"
5.      android:layout_height="match_parent"
6.  tools:context=".MainActivity" />
```

Les fragments vont venir s'insérer dans le conteneur d'id [pager] de la ligne 3.

- ligne 34 : le gestionnaire de fragments est instancié. Le paramètre du constructeur est la classe Android [`android.support.v4.app.FragmentManager`] (ligne 5) ;
- ligne 37 : on récupère dans la vue [main.xml] la référence du conteneur de fragments ;
- ligne 39 : le gestionnaire de fragments est lié au conteneur de fragments ;

Le gestionnaire de fragments [SectionsPagerAdapter] est le suivant :

```
1.  // notre gestionnaire de fragments
2.    // à redéfinir pour chaque application
3.    // doit définir les méthodes suivantes
4.    // getItem, getCount, getPageTitle
5.    public class SectionsPagerAdapter extends FragmentPagerAdapter {
```

```
6.
7.      // constructeur
8.       public SectionsPagerAdapter(FragmentManager fm) {
9.         super(fm);
10.     }
11.
12.     @Override
13.     public Fragment getItem(int position) {
14.       // fragment n° position
15.       return PlaceholderFragment.newInstance(position + 1);
16.     }
17.
18.     // rend le nombre de fragments à gérer
19.     @Override
20.     public int getCount() {
21.       // 3 fragments
22.       return 3;
23.     }
24.
25.     // rend le titre du fragment n° position
26.     @Override
27.     public CharSequence getPageTitle(int position) {
28.       Locale l = Locale.getDefault();
29.       switch (position) {
30.         case 0:
31.           return getString(R.string.title_section1).toUpperCase(l);
32.         case 1:
33.           return getString(R.string.title_section2).toUpperCase(l);
34.         case 2:
35.           return getString(R.string.title_section3).toUpperCase(l);
36.       }
37.       return null;
38.     }
39.   }
```

- ligne 5 : le gestionnaire de fragments étend la classe Android [`android.support.v4.app.FragmentPagerAdapter`]. Le constructeur nous est imposé. Nous devons définir trois méthodes :
    - **int getCount()** : rend le nombre de fragments à gérer,
    - **Fragment getItem(i)** : rend le fragment n° i,
    - **CharSequence getPageTitle(i)** : rend le titre du fragment n° i ;
- lignes 20-23 : **getCount** rend le nombre de fragments gérés, ici trois ;
- ligne 13 : **getItem(i)** rend le fragment n° i. Ici tous les fragments seront identiques de type [PlaceholderFragment] ;
- ligne 27 : **getPageTitle(int i)** rend le titre du fragment n° i ;
- lignes 29-36 : les titres des trois fragments sont trouvés dans le fichier [strings.xml] :

```
1.      <string name="title_section1">Section 1</string>
2.      <string name="title_section2">Section 2</string>
3.  <string name="title_section3">Section 3</string>
```

Les trois fragments créés sont du type [PlaceholderFragment] suivant :

```
1.  // un fragment est une vue affichée par un conteneur de fragments
2.    public static class PlaceholderFragment extends Fragment {
3.
4.      // n° de fragment
5.      private static final String ARG_SECTION_NUMBER = "section_number";
6.
7.      // rend un type [PlaceholderFragment]
8.      public static PlaceholderFragment newInstance(int sectionNumber) {
9.        // instanciation fragment
10.       PlaceholderFragment fragment = new PlaceholderFragment();
11.       // arguments du fragment
12.       Bundle args = new Bundle();
13.       args.putInt(ARG_SECTION_NUMBER, sectionNumber);
14.       fragment.setArguments(args);
15.       // on rend le fragment instancié
16.       return fragment;
17.     }
18.
19.     public PlaceholderFragment() {
20.     }
```

```
21.
22.     @Override
23.     public View onCreateView(LayoutInflater inflater, ViewGroup container,
24.                              Bundle savedInstanceState) {
25.       // le fragment est associé à la vue [fragment1]
26.       View rootView = inflater.inflate(R.layout.fragment1, container, false);
27.       // on retourne la vue
28.       return rootView;
29.     }
30.   }
```

- ligne 2 : la classe [PlaceholderFragment] est statique. Sa méthode [newInstance] permet d'obtenir des instances de type [PlaceholderFragment] ;
- lignes 8-17 : la méthode [newInstance] crée et retourne un objet de type [PlaceholderFragment] ;
- ligne 14 : le fragment est créé avec un argument mais dans le code généré, celui-ci ne n'est jamais utilisé. On aurait donc tout aussi bien pu créer des fragments sans argument ;

Un fragment doit définir la méthode [onCreateView] de la ligne 23. Cette méthode doit rendre la vue associée au fragment.

- ligne 26 : la vue [fragment1.xml] est associée au fragment ;
- ligne 28 : on rend la vue ainsi créée ;

Pour l'instant, à dessein nous n'avons pas parlé des onglets. Les notions de fragments et de conteneur de fragments sont indépendantes des onglets. On peut les utiliser avec des vues sans onglets. La gestion des onglets est faite aux lignes suivantes :

```
1.  // on crée autant d'onglets qu'il y a de fragments affichés par le conteneur
2.      for (int i = 0; i < mSectionsPagerAdapter.getCount(); i++) {
3.        // actionBar est la barre d'onglets
4.        // actionBar.newTab() crée un nouvel onglet
5.        // actionBar.newTab().setText() donne un titre à cet onglet
6.        // actionBar.newTab().setText().setTabListener(this) indique que cette classe (le fragment) gère les
    évts des onglets
7.        actionBar.addTab(
8.          actionBar.newTab()
9.            .setText(mSectionsPagerAdapter.getPageTitle(i))
10.           .setTabListener(this));
11.     }
12.   }
13.
14.   @Override
15.   public void onTabSelected(ActionBar.Tab tab, FragmentTransaction fragmentTransaction) {
16.     // un onglet a été sélectionné - on change le fragment affiché par le conteneur de fragments
17.     mViewPager.setCurrentItem(tab.getPosition());
18.   }
19.
20.   @Override
21.   public void onTabUnselected(ActionBar.Tab tab, FragmentTransaction fragmentTransaction) {
22.   }
23.
24.   @Override
25.   public void onTabReselected(ActionBar.Tab tab, FragmentTransaction fragmentTransaction) {
26.   }
```

On notera ligne 17, la façon de changer de fragment dans le conteneur de fragments.

Enfin il y dans le code généré des sections inutiles pour l'instant :

```
1.    @Override
2.    protected void onCreate(Bundle savedInstanceState) {
3.     ...
4.
5.      // gestionnaire du changement de fragment
6.      mViewPager.setOnPageChangeListener(new ViewPager.SimpleOnPageChangeListener() {
7.        @Override
8.        public void onPageSelected(int position) {
9.          // on sélectionne l'onglet du fragment dans la barre d'actions
10.         actionBar.setSelectedNavigationItem(position);
11.       }
12.     });
13. ....
14. }
```

- lignes 6-12 : on gère un changement de fragment. Dans ce cas, on change l'onglet sélectionné. Dans notre cas, on change de fragment en cliquant un onglet donc celui-ci est automatiquement sélectionné. Ce que fait la ligne 10 est donc redondant dans ce cas. Ce gestionnaire d'événement serait utile si on changeait de fragment par programmation. Dans ce cas, l'onglet de ce fragment serait automatiquement sélectionné ;

```
1.    @Override
2.    public boolean onCreateOptionsMenu(Menu menu) {
3.      // Inflate the menu; this adds items to the action bar if it is present.
4.      getMenuInflater().inflate(R.menu.main, menu);
5.      return true;
6.    }
7.
8.    @Override
9.    public boolean onOptionsItemSelected(MenuItem item) {
10.     // Handle action bar item clicks here. The action bar will
11.     // automatically handle clicks on the Home/Up button, so long
12.     // as you specify a parent activity in AndroidManifest.xml.
13.     int id = item.getItemId();
14.     if (id == R.id.action_settings) {
15.       return true;
16.     }
17.     return super.onOptionsItemSelected(item);
18. }
```

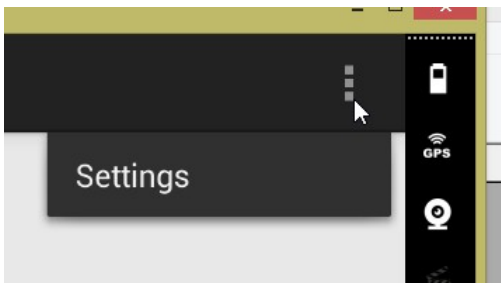
Ce code gère un menu défini (ligne 4) associé au menu [res / menu / main.xml] :

```
1.  <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:app="http://schemas.android.com/apk/res-auto"
3.      xmlns:tools="http://schemas.android.com/tools"
4.      tools:context="android.exemples.MainActivity" >
5.      <item android:id="@+id/action_settings"
6.          android:title="@string/action_settings"
7.          android:orderInCategory="100"
8.          app:showAsAction="never" />
9.  </menu>
```

Ce menu a une unique option définie lignes 5-8 et dont l'intitulé [*action_settings*] (ligne 6) est défini dans [res / values / strings.xml] (ligne 9) :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.
4.      <string name="app_name">Exemple-06</string>
5.      <string name="title_section1">Section 1</string>
6.      <string name="title_section2">Section 2</string>
7.      <string name="title_section3">Section 3</string>
8.      <string name="hello_world">Hello world!</string>
9.      <string name="action_settings">Settings</string>
10.
11. </resources>
```

A l'exécution, ce menu apparaît en haut / droite de la fenêtre :



1.7.3 Personnalisation des fragments

Revenons sur le code de création des trois fragments :

```
1.  public static class PlaceholderFragment extends Fragment {
2.
3.      // n° de fragment
4.      private static final String ARG_SECTION_NUMBER = "section_number";
5.
6.      // rend un type [PlaceholderFragment]
7.      public static PlaceholderFragment newInstance(int sectionNumber) {
8.        // instanciation fragment
9.        PlaceholderFragment fragment = new PlaceholderFragment();
10.       // arguments du fragment
11.       Bundle args = new Bundle();
12.       args.putInt(ARG_SECTION_NUMBER, sectionNumber);
13.       fragment.setArguments(args);
14.       // on rend le fragment instancié
15.       return fragment;
16.     }
17.
18.     public PlaceholderFragment() {
19.     }
20.
21.     @Override
22.     public View onCreateView(LayoutInflater inflater, ViewGroup container,
23.                              Bundle savedInstanceState) {
24.       // le fragment est associé à la vue [fragment1]
25.       View rootView = inflater.inflate(R.layout.fragment1, container, false);
26.       // on retourne la vue
27.       return rootView;
28.     }
29.   }
```

- lignes 11-13 : nous avons vu qu'un argument était passé au fragment créé mais que celui-ci n'en faisait rien ;

Le fichier [fragment1.xml] contient un composant [TextView] (lignes 11-14) que nous allons utiliser pour afficher l'argument du fragment :

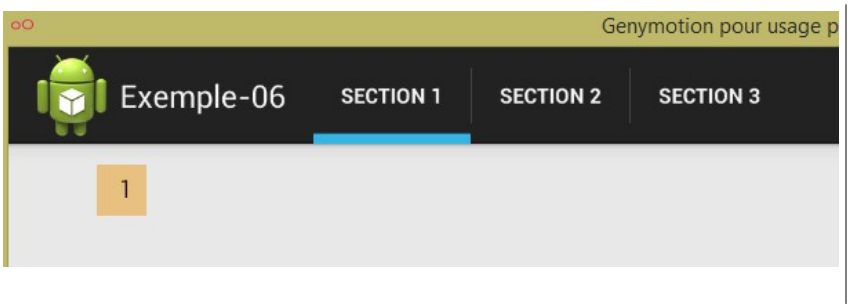
```
1.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools"
3.      android:layout_width="match_parent"
4.      android:layout_height="match_parent"
5.      android:paddingLeft="@dimen/activity_horizontal_margin"
6.      android:paddingRight="@dimen/activity_horizontal_margin"
7.      android:paddingTop="@dimen/activity_vertical_margin"
8.      android:paddingBottom="@dimen/activity_vertical_margin"
9.      tools:context="android.exemples.MainActivity$PlaceholderFragment">
10.
11.     <TextView
12.         android:id="@+id/section_label"
13.         android:layout_width="wrap_content"
14.         android:layout_height="wrap_content" />
15.
16. </RelativeLayout>
```

La méthode [onCreateView] évolue de la façon suivante :

```
1.      @Override
2.      public View onCreateView(LayoutInflater inflater, ViewGroup container,
3.                               Bundle savedInstanceState) {
4.        // le fragment est associé à la vue [fragment1]
5.        View rootView = inflater.inflate(R.layout.fragment1, container, false);
6.        // on récupère le [TextView]
7.        TextView textView = (TextView) rootView.findViewById(R.id.section_label);
8.        // pour l'initialiser avec l'argument du fragment
9.        textView.setText(Integer.toString(getArguments().getInt(ARG_SECTION_NUMBER)));
10.       // on retourne la vue
11.       return rootView;
12. }
```

- ligne 9 : on récupère l'information que l'activité a placée dans les arguments du fragment associée à la clé [ARG_SECTION_NUMBER] ;

A l'exécution, on obtient désormais la chose suivante :

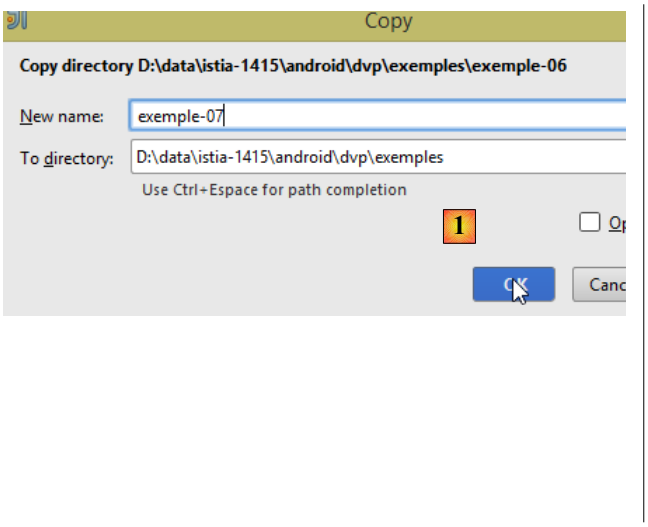


1.8 Exemple-07 : utilisation de la bibliothèque [Android Annotations] avec Gradle

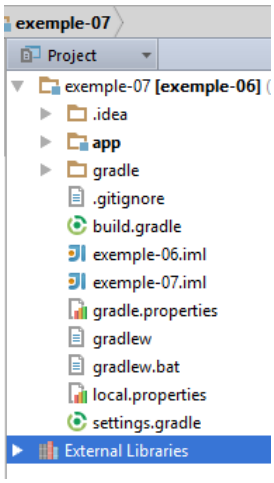
1.8.1 Création du projet

Nous allons dupliquer le projet [exemple-06] dans [exemple-07] pour introduire dans ce dernier les annotations Android :

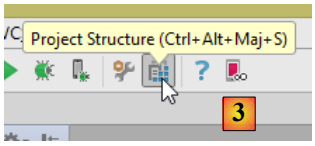
- commençons par supprimer le dossier [app / build] ;
- puis faisons un Copy / Paste [1] :

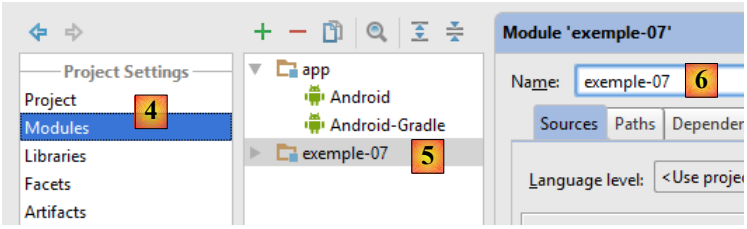


2

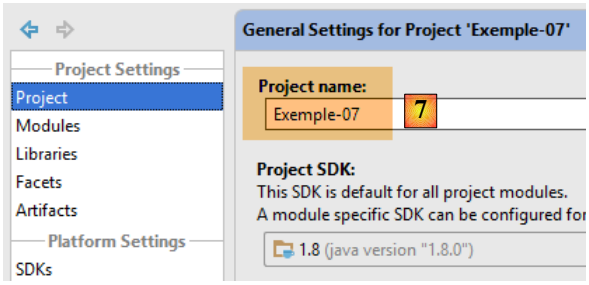


Puis nous ouvrons le dossier [exemple-07] (File / Open Project) ainsi créé [2]. Nous pouvons voir que le projet lié au dossier [exemple-07] s'appelle [exemple-06]. Nous commençons par changer cela :





- en [3], on accède à la structure du projet ;
- en [4-6], on change le nom du module [exemple-06] en [exemple-07] (dans [6]) ;



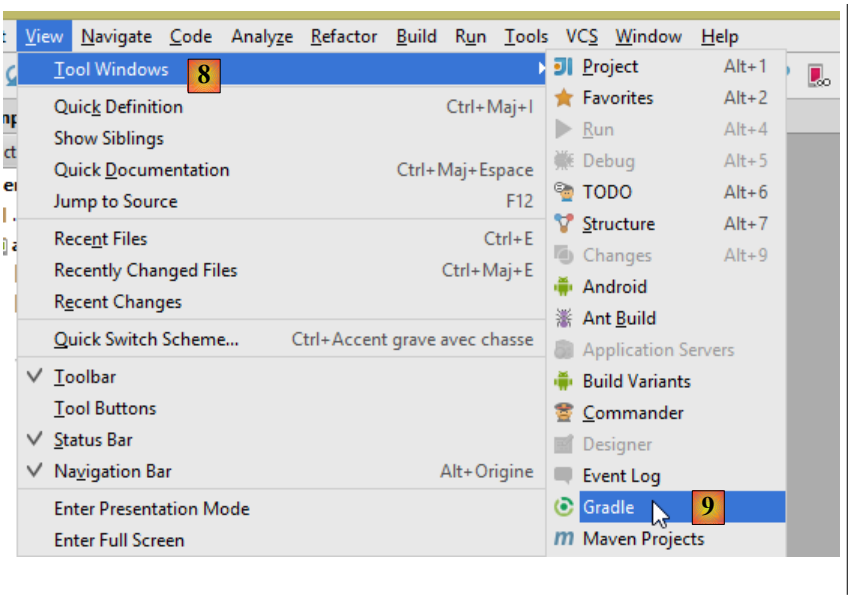
- en [7], nous changeons également le nom du projet ;

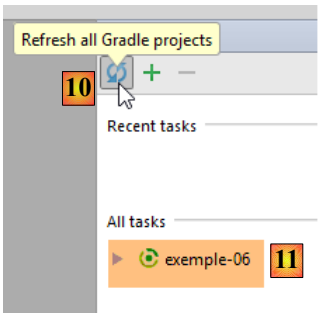
Puis dans le fichier [res / values / strings.xml] :

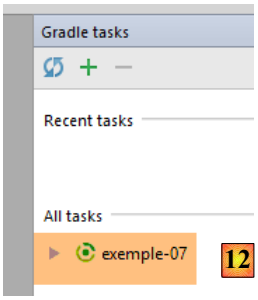
```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.
4.     <string name="app_name">Exemple-07</string>
5.     <string name="title_section1">Section 1</string>
6.     <string name="title_section2">Section 2</string>
7.     <string name="title_section3">Section 3</string>
8.     <string name="hello_world">Hello world!</string>
9.     <string name="action_settings">Settings</string>
10. </resources>
```

- ligne 4 : on change le nom de l'application ;

Puis on rafraîchit le projet [Gradle] [8-11] :

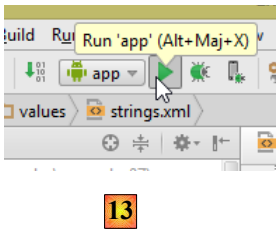


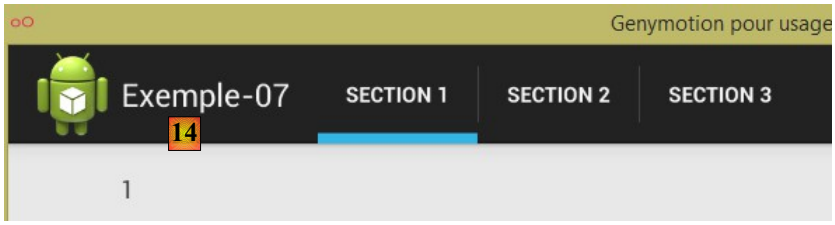




- en [12], la fenêtre [Gradle] a été rafraîchie et affiche [exemple-07] au lieu de [exemple-06] [9] ;

On peut alors exécuter [13-14] le module [app] du projet [exemple-07] :





Parfois, cela ne marche pas et on a toujours en [14] la chaîne [exemple-06]. Il faut alors vérifier deux points (non liés l'un à l'autre - l'ordre n'a pas d'importance) :

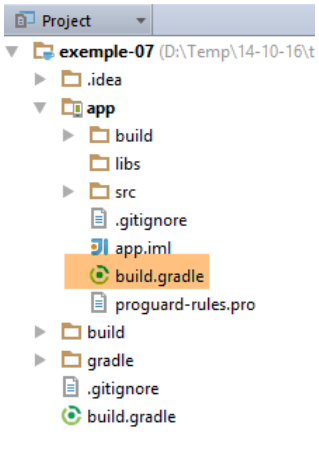
- dans [res / values / strings.xml], on doit avoir la chaîne :

```
<string name="app_name">exemple-07</string>
```

- vous devez rafraîchir le projet [Gradle] ;

1.8.2 Configuration du projet

La configuration de la compilation du projet (build) est faite par le fichier [build.gradle] du module [app] :



Ce fichier est pour l'instant le suivant (peut être différent selon votre version d'IntellijIDEA) :

```
1.  buildscript {
2.    repositories {
3.      jcenter()
```

```
4.    }
5.    dependencies {
6.      classpath 'com.android.tools.build:gradle:1.0.0'
7.    }
8.  }
9.
10. apply plugin: 'com.android.application'
11.
12. repositories {
13.   jcenter()
14. }
15.
16. dependencies {
17.   compile fileTree(dir: 'libs', include: ['*.jar'])
18.   compile 'com.android.support:appcompat-v7:20.+'
19. }
20.
21. android {
22.   compileSdkVersion 20
23.   buildToolsVersion "20.0.0"
24.   defaultConfig {
25.     applicationId "android.exemples"
26.     minSdkVersion 11
27.     targetSdkVersion 20
28.     versionCode 1
29.     versionName "1.0"
30.   }
31.
32.   compileOptions {
33.     sourceCompatibility JavaVersion.VERSION_1_6
34.     targetCompatibility JavaVersion.VERSION_1_6
35.   }
36.
37.   buildTypes {
38.     release {
39.       minifyEnabled false
40.       proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
41.     }
42.   }
43. }
```

Mes explications seront succinctes.

- lignes 16-19 : les dépendances du projet. Elles peuvent changer avec chaque nouveau projet ;
- ligne 17 : les jars du dossier [app / libs] font partie des dépendances ;
- ligne 18 : cette dépendance correspond à la dépendance Maven suivante :

```
1.      <dependency>
2.        <groupId>com.android.support</groupId>
3.        <artifactId>appcompat-v7</artifactId>
4.        <version>20</version>
5.        <scope>compile</scope>
6.  </dependency>
```

La notation [20.+] de la ligne 40 indique toute version 20 : 20, 20.1, 20.2, ... Les versions récentes d'Intellij déconseillent pourtant cette syntaxe ;

Le fichier [build.gradle] est en fait un script Groovy [http://groovy.codehaus.org/] qui va être exécuté pour générer les binaires du projet. On peut logiquement imaginer que cela amène de la souplesse vis à vis de la description XML statique d'un build Maven, par exemple faire des builds sous condition à l'aide de tests Groovy.

Ce fichier Gradle doit être modifié pour prendre en charge la bibliothèque [Android Annotations]. Sur le site de cette bibliothèque on trouve le script suivant [https://github.com/excilys/androidannotations/wiki/Building-Project-Gradle] (5 mars 2015) :

```
1.  buildscript {
2.      repositories {
3.        mavenCentral()
4.      }
5.      dependencies {
6.          // replace with the current version of the Android plugin
7.          classpath 'com.android.tools.build:gradle:1.0.0'
```

```
        // replace with the current version of the android-apt plugin
        classpath 'com.neenbedankt.gradle.plugins:android-apt:1.4'
    }
}

repositories {
    mavenCentral()
    mavenLocal()
}

apply plugin: 'com.android.application'
apply plugin: 'android-apt'
def AAVersion = 'XXX'

dependencies {
    apt "org.androidannotations:androidannotations:$AAVersion"
    compile "org.androidannotations:androidannotations-api:$AAVersion"
}

apt {
    arguments {
        androidManifestFile variant.outputs[0].processResources.manifestFile
        // if you have multiple outputs (when using splits), you may want to have other index than 0

        resourcePackageName 'com.myproject.package'

        // If you're using Android NBS flavors you should use the following line instead of hard-coded packageName
        // resourcePackageName android.defaultConfig.packageName

        // You can set optional annotation processing options here, like these commented options:
        // logLevel 'INFO'
        // logFile '/var/log/aa.log'
    }
}

android {
    compileSdkVersion 19
    buildToolsVersion "20.0.0"

    defaultConfig {
        minSdkVersion 9
        targetSdkVersion 19
    }

    // This is only needed if you project structure doesn't fit the one found here
    // http://tools.android.com/tech-docs/new-build-system/user-guide#TOC-Project-Structure
    sourceSets {
        main {
            // manifest.srcFile 'src/main/AndroidManifest.xml'
            // java.srcDirs = ['src/main/java', 'build/generated/source/apt/${variant.dirName}']
            // resources.srcDirs = ['src/main/resources']
            // res.srcDirs = ['src/main/res']
            // assets.srcDirs = ['src/main/assets']
        }
    }
}
```

Lorsqu'on fusionne ce fichier avec les informations du précédent, on arrive au script Groovy suivant :

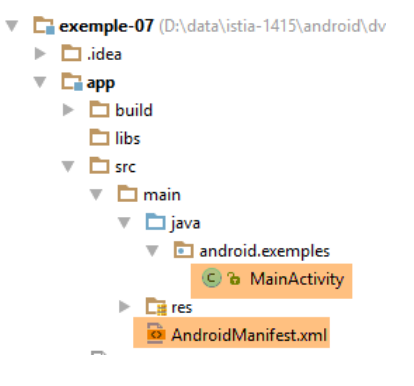
```
buildscript {
  repositories {
    mavenCentral()
    mavenLocal()
  }

  dependencies {
    // replace with the current version of the Android plugin
    classpath 'com.android.tools.build:gradle:1.0.0'
    // Since Android's Gradle plugin 0.11, you have to use android-apt >= 1.3
    classpath 'com.neenbedankt.gradle.plugins:android-apt:1.4'
  }
}

```

```
15. apply plugin: 'com.android.application'
16. apply plugin: 'android-apt'
17. def AAVersion = '3.1'
18.
19. dependencies {
20.   apt "org.androidannotations:androidannotations:$AAVersion"
21.   compile "org.androidannotations:androidannotations-api:$AAVersion"
22.   compile 'com.android.support:appcompat-v7:20.+'
23.   compile fileTree(dir: 'libs', include: ['*.jar'])
24. }
25.
26. repositories {
27.   jcenter()
28. }
29.
30. apt {
31.   arguments {
32.     androidManifestFile variant.outputs[0].processResources.manifestFile
33.     resourcePackageName android.defaultConfig.applicationId
34.   }
35. }
36.
37.
38. android {
39.   compileSdkVersion 20
40.   buildToolsVersion "20.0.0"
41.
42.   defaultConfig {
43.     applicationId "android.exemples"
44.     minSdkVersion 11
45.     targetSdkVersion 20
46.     versionCode 1
47.     versionName "1.0"
48.   }
49.
50.   compileOptions {
51.     sourceCompatibility JavaVersion.VERSION_1_6
52.     targetCompatibility JavaVersion.VERSION_1_6
53.   }
54.
55.   buildTypes {
56.     release {
57.       minifyEnabled false
58.       proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
59.     }
60.   }
61. }
```

On fera attention aux différentes versions des lignes 9, 11, 17, 22, 39, 40, 44, 45. Exécutez le projet et vérifiez que vous obtenez toujours l'interface avec onglets.

1.8.3 Ajout d'une première annotation

Nous allons créer une première annotation de la bibliothèque AA. Ceci sera fait dans [MainActivity] :



La classe [MainActivity] évolue de la façon suivante :

```
1.  @EActivity(R.layout.main)
2.  public class MainActivity extends ActionBarActivity implements ActionBar.TabListener {
3.
4.    // le gestionnaire de fragments ou sections
5.    SectionsPagerAdapter mSectionsPagerAdapter;
6.
7.    // le conteneur des fragments
8.    @ViewById(R.id.pager)
9.    ViewPager mViewPager;
10.
11.   @Override
12.   protected void onCreate(Bundle savedInstanceState) {
13.     // classique
14.     super.onCreate(savedInstanceState);
15.   }
16.
17.   @AfterViews
18.   protected void initActivity() {
19.     // la barre d'onglets
20.     final ActionBar actionBar = getSupportActionBar();
21.     actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
22.
23.     // instanciation du gestionnaire de fragments
24.     mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
25.
26.     // qu'on associe à notre gestionnaire de fragments
27.     mViewPager.setAdapter(mSectionsPagerAdapter);
28. ...
29.   }
```

- ligne 1 : l'annotation [@EActivity] fait de [MainActivity] une classe gérée par AA. Son paramètre [R.layout.main] est l'identifiant de la vue [main.xml] associée à l'activité ;
- lignes 8-9 : le composant identifié par [R.id.pager] est injecté dans le champ [mViewPager]. On rappelle que ce composant est dans la vue [main.xml] ;
- lignes 12-15 : la méthode [onCreate] est réduite à sa plus simple expression ;
- lignes 17-29 : le code qui était auparavant dans la méthode [onCreate] migre dans une méthode de nom quelconque mais annotée par [@AfterViews] (ligne 17). Dans la méthode ainsi annotée, on est assuré que tous les composants de l'interface visuelle annotés par [@ViewById] ont été initialisés ;

On se rappelle que l'annotation [@EActivity] va générer une classe [MainActivity_] qui sera la véritable activité du projet. Il faut donc modifier le fichier [AndroidManifest.xml] de la façon suivante :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.      package="android.exemples" >
4.
5.      <application
6.          android:allowBackup="true"
7.          android:icon="@drawable/ic_launcher"
8.          android:label="@string/app_name"
9.          android:theme="@style/AppTheme" >
10.         <activity
11.             android:name="android.exemples.MainActivity_"
12.             android:label="@string/app_name" >
13.             <intent-filter>
14.                 <action android:name="android.intent.action.MAIN" />
15.
16.                 <category android:name="android.intent.category.LAUNCHER" />
17.             </intent-filter>
18.         </activity>
19.     </application>
20.
21. </manifest>
```

- ligne 11 : la nouvelle activité.

A ce point, exécutez de nouveau le projet et vérifiez que vous obtenez toujours l'interface avec onglets.

1.8.4 Refactoring des fragments

Nous allons revoir la gestion des fragments du projet. Pour l'instant la classe [PlaceholderFragment] est une classe interne statique de l'activité [MainActivity]. Nous allons revenir à un cas d'utilisation plus usuel, celui où les fragments sont définis dans des classes externes. Par ailleurs, nous introduisons les annotations AA pour les fragments.

Le projet [exemple-07] évolue de la façon suivante :

Ci-dessus, on voit apparaître la classe [PlaceholderFragment] qui a été externalisée en-dehors de la classe [MainActivity].

La classe [MainActivity] perd sa classe interne [PlaceholderFragment] et voit son gestionnaire de fragments évoluer de la façon suivante :

```
1.    // notre gestionnaire de fragments
2.    // à redéfinir pour chaque application
3.    // doit définir les méthodes suivantes
4.    // getItem, getCount, getPageTitle
5.    public class SectionsPagerAdapter extends FragmentPagerAdapter {
6.
7.      // n° de fragment
8.      private static final String ARG_SECTION_NUMBER = "section_number";
9.      // nombre de fragments
10.     private static final int FRAGMENTS_COUNT = 3;
11.
12.     // les fragments
13.     private Fragment[] fragments;
14.
15.     // constructeur
16.     public SectionsPagerAdapter(FragmentManager fm) {
17.       super(fm);
18.       // initialisation du tableau des fragments
19.       fragments = new Fragment[FRAGMENTS_COUNT];
20.       for (int i = 0; i < fragments.length; i++) {
21.         // on crée un fragment
22.         fragments[i] = new PlaceholderFragment_();
23.         // on peut passer des arguments au fragment
24.         Bundle args = new Bundle();
25.         args.putInt(ARG_SECTION_NUMBER, i + 1);
26.         fragments[i].setArguments(args);
27.       }
28.     }
29.
30.
31.     @Override
32.     public Fragment getItem(int position) {
33.       // fragment n° position
34.       return fragments[position];
35.     }
36.
37.     // rend le nombre de fragments à gérer
38.     @Override
39.     public int getCount() {
40.       // 3 fragments
41.       return fragments.length;
42.     }
43.
44.     // rend le titre du fragment n° position
45.     @Override
46.     public CharSequence getPageTitle(int position) {
```

```
47.       Locale l = Locale.getDefault();
48.       switch (position) {
49.         case 0:
50.           return getString(R.string.title_section1).toUpperCase(l);
51.         case 1:
52.           return getString(R.string.title_section2).toUpperCase(l);
53.         case 2:
54.           return getString(R.string.title_section3).toUpperCase(l);
55.       }
56.       return null;
57.     }
58. }
```

- ligne 13 : les fragments sont mis dans un tableau ;
- lignes 16-28 : l'initialisation du tableau des fragments se fait dans le constructeur. . Ils sont de type [PlaceholderFragment_] (ligne 22) et non [PlaceholderFragment]. Ils vont être en effet annotés par une annotation AA qui va produire une classe [PlaceholderFragment_] dérivée de [PlaceholderFragment] et c'est cette classe que l'activité doit utiliser. Chaque fragment créé se voit passer un argument entier qui sera affiché par le fragment ;

La classe [PlaceholderFragment] devient la suivante :

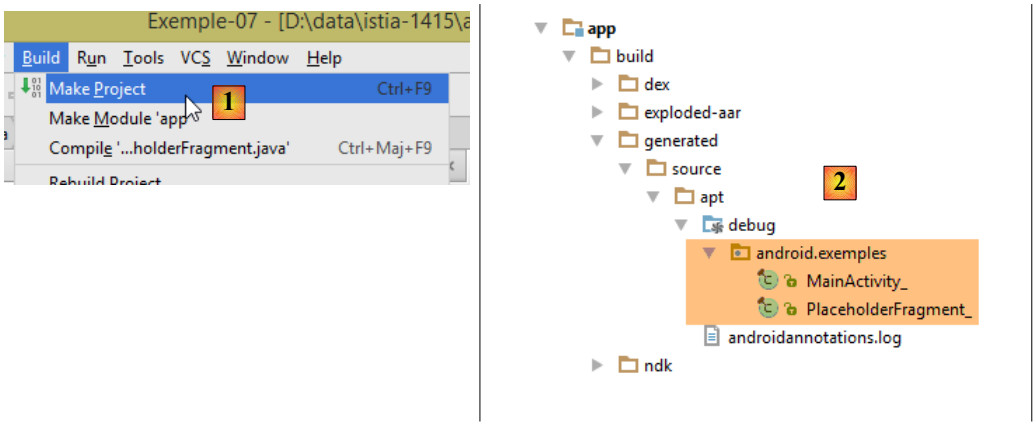
```
1.  package android.exemples;
2.
3.  import android.support.v4.app.Fragment;
4.  import android.widget.TextView;
5.  import org.androidannotations.annotations.EFragment;
6.  import org.androidannotations.annotations.ViewById;
7.
8.  // un fragment est une vue affichée par un conteneur de fragments
9.  @EFragment(R.layout.fragment1)
10. public class PlaceholderFragment extends Fragment {
11.
12.   // composant de l'interface visuelle
13.   @ViewById(R.id.section_label)
14.   protected TextView textViewInfo;
15.
16.   // n° de fragment
17.   private static final String ARG_SECTION_NUMBER = "section_number";
18.
19.   @Override
20.   public void onResume() {
21.     super.onResume();
22.     if (textViewInfo != null) {
23.       textViewInfo.setText(Integer.toString(getArguments().getInt(ARG_SECTION_NUMBER)));
24.     }
25.   }
26. }
```

- ligne 9 : le fragment est annoté avec l'annotation [@EFragment] dont le paramètre est l'identifiant de la vue XML associée au fragment, ici la vue [fragment1.xml] ;
- lignes 13-14 : injectent dans le champ [textViewInfo] la référence du composant de [fragment1.xml] identifié par [R.id.section_label] qui est un type [TextView] (ligne 12 ci-dessous) :

```
1.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools"
3.      android:layout_width="match_parent"
4.      android:layout_height="match_parent"
5.      android:paddingLeft="@dimen/activity_horizontal_margin"
6.      android:paddingRight="@dimen/activity_horizontal_margin"
7.      android:paddingTop="@dimen/activity_vertical_margin"
8.      android:paddingBottom="@dimen/activity_vertical_margin"
9.      tools:context="android.exemples.MainActivity$PlaceholderFragment">
10.
11.     <TextView
12.         android:id="@+id/section_label"
13.         android:layout_width="wrap_content"
14.         android:layout_height="wrap_content" />
15.
16. </RelativeLayout>
```

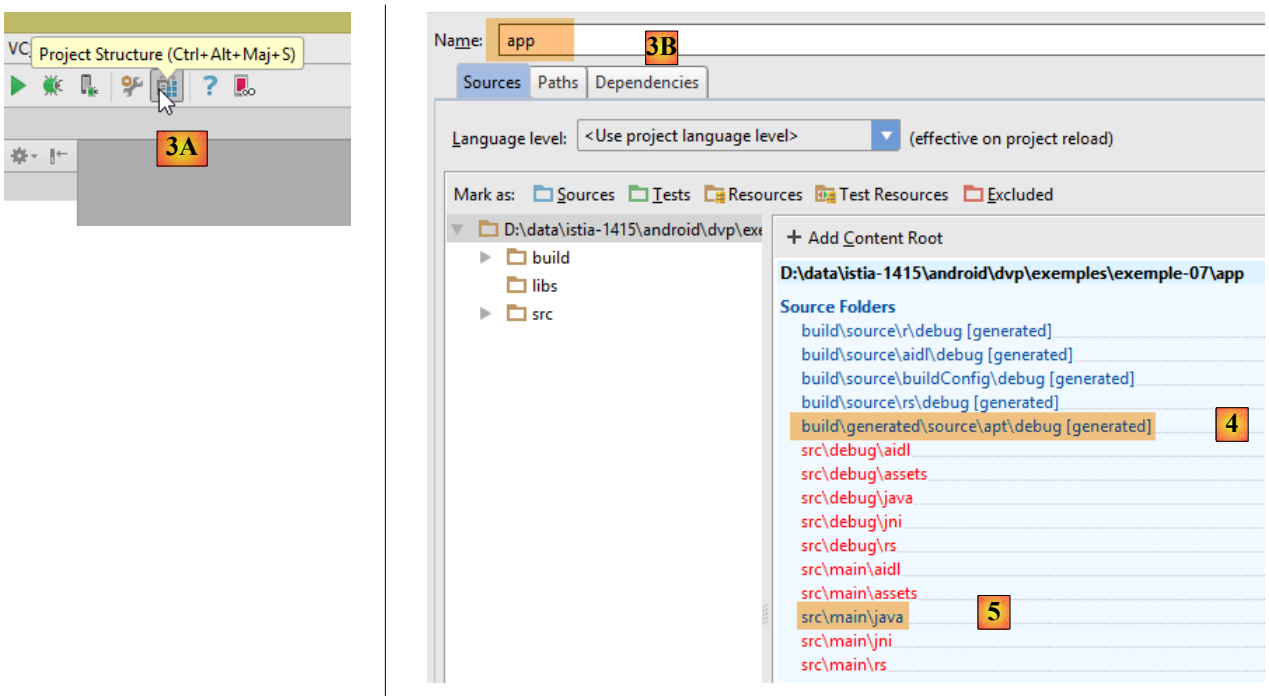
- lignes 20-24 : la méthode [onResume] est exécutée avant l'affichage de la vue associée au fragment. On peut l'utiliser pour mettre à jour l'interface visuelle qui va être affichée ;
- ligne 21 : on doit appeller la méthode de même nom de la classe parent ;
- ligne 22 : il y a une difficulté à savoir si la méthode [onResume] peut ou non être exécutée avant l'initialisation du champ de la ligne 22. Par précaution, on fait un test de nullité ;
- ligne 23 : on met à jour l'information du champ [textViewInfo] avec l'argument entier passé au fragment lors de sa création ;

Compilons [Make] [1] ce projet :



- en [2], on voit que les classes générées par la bibliothèque AA le sont dans le dossier [app / build / generated / source / apt / debug] ;

Dans les propriétés du projet, en [3A-5], on voit que ce dossier fait partie des dossiers utilisés pour la compilation du projet :



Exécutez le projet [exemple-07] et vérifiez qu'il fonctionne toujours.

1.8.5 Un nouveau fragment

Apprenons à créer un fragment et à l'afficher. Tout d'abord copions la vue [vue1.xml] du projet [exemple-05] dans le projet [exemple-07] [1] :

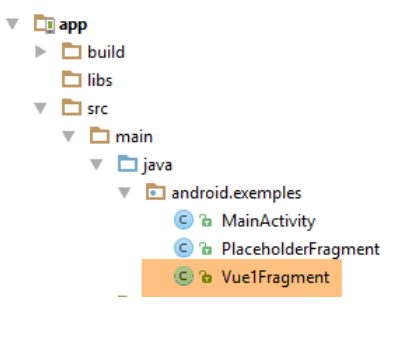
- en [1], la vue [vue1.xml] ;
- en [2], la vue présente des erreurs provenant de textes absents dans le fichier [res/values/strings.xml] ;

En [3], on rajoute les textes manquants en les prenant dans le fichier [res/values/strings.xml] du projet [exemple-05] :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.
4.    <string name="app_name">Exemple-07</string>
5.    <string name="title_section1">Section 1</string>
6.    <string name="title_section2">Section 2</string>
7.    <string name="title_section3">Section 3</string>
8.    <string name="hello_world">Hello world!</string>
9.    <string name="action_settings">Settings</string>
10.   <!-- vue 1 -->
11.   <string name="titre_vue1">Vue n° 1</string>
12.   <string name="txt_nom">Quel est votre nom ?</string>
13.   <string name="btn_valider">Valider</string>
14.   <string name="btn_vue2">Vue n° 2</string>
15. </resources>
```

- ci-dessus, on a ajouté les lignes 10-14 ;

Maintenant, nous créons la classe [Vue1Fragment] qui va être le fragment chargé d'afficher la vue [vue1.xml] :



La classe [Vue1Fragment] sera la suivante :

```
1.  package android.exemples;
2.
3.  import android.support.v4.app.Fragment;
4.  import android.widget.EditText;
5.  import android.widget.Toast;
6.  import org.androidannotations.annotations.Click;
7.  import org.androidannotations.annotations.EFragment;
8.  import org.androidannotations.annotations.ViewById;
9.
10. @EFragment(R.layout.vue1)
11. public class Vue1Fragment extends Fragment {
12.
13.   // les éléments de l'interface visuelle
14.   @ViewById(R.id.editTextNom)
15.   protected EditText editTextNom;
16.
17.   // gestionnaire d'évt
18.   @Click(R.id.buttonValider)
19.   protected void doValider() {
20.     // on affiche le nom saisi
21.     Toast.makeText(getActivity(), String.format("Bonjour %s", editTextNom.getText().toString()),
    Toast.LENGTH_LONG).show();
22.   }
23. }
```

- ligne 10 : l'annotation [@EFragment] fait que le frament utilisé par l'activité sera en réalité la classe [Vue1Fragment_]. il faut s'en souvenir. Le fragment est associé à la vue [vue1.xml] ;
- lignes 14-15 : le composant identifié par [R.id.editTextNom] est injecté dans le champ [editTextNom] de la ligne 15 ;
- lignes 18-20 : la méthode [doValider] gère l'événement 'click' sur le bouton identifié par [R.id.buttonValider] ;
- ligne 21 : le premier paramètre de [Toast.makeText] est de type [Activity]. La méthode [Fragment.getActivity()] permet d'avoir l'activité dans laquelle se trouve le fragment. Il s'agit de [MainActivity] puisque dans cette architecture, nous n'avons qu'une activité qui affiche différentes vues ou fragments ;

Dans la classe [MainActivity], le gestionnaire de fragments évolue de la façon suivante :

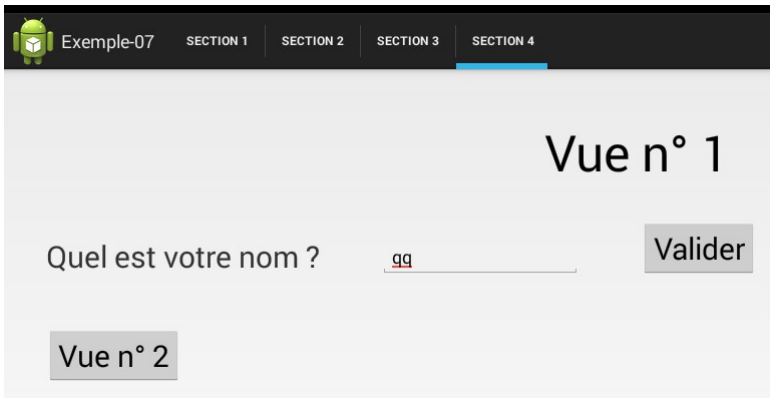
```
1.  public class SectionsPagerAdapter extends FragmentPagerAdapter {
2.
3.      // n° de fragment
4.      private static final String ARG_SECTION_NUMBER = "section_number";
5.      // nombre de fragments
6.      private static final int FRAGMENTS_COUNT = 4;
7.
8.      // les fragments
9.      private Fragment[] fragments;
10.
11.     // constructeur
12.     public SectionsPagerAdapter(FragmentManager fm) {
13.       super(fm);
14.       // initialisation du tableau des fragments
15.       fragments = new Fragment[FRAGMENTS_COUNT];
16.       for (int i = 0; i < fragments.length-1; i++) {
17.         // on crée un fragment
18.         fragments[i] = new PlaceholderFragment_();
19.         // on peut passer des arguments au fragment
20.         Bundle args = new Bundle();
```

```
21.         args.putInt(ARG_SECTION_NUMBER, i + 1);
22.         fragments[i].setArguments(args);
23.       }
24.       // un fragment de +
25.       fragments[fragments.length - 1] = new Vue1Fragment_();
26.     }
27.
28.
29.     @Override
30.     public Fragment getItem(int position) {
31.       // fragment n° position
32.       return fragments[position];
33.     }
34.
35.     // rend le nombre de fragments à gérer
36.     @Override
37.     public int getCount() {
38.       // nb de fragments
39.       return fragments.length;
40.     }
41.
42.     // rend le titre du fragment n° position
43.     @Override
44.     public CharSequence getPageTitle(int position) {
45.       Locale l = Locale.getDefault();
46.       switch (position) {
47.         case 0:
48.           return getString(R.string.title_section1).toUpperCase(l);
49.         case 1:
50.           return getString(R.string.title_section2).toUpperCase(l);
51.         case 2:
52.           return getString(R.string.title_section3).toUpperCase(l);
53.         case 3:
54.           return getString(R.string.title_section4).toUpperCase(l);
55.       }
56.       return null;
57.     }
58.   }
```

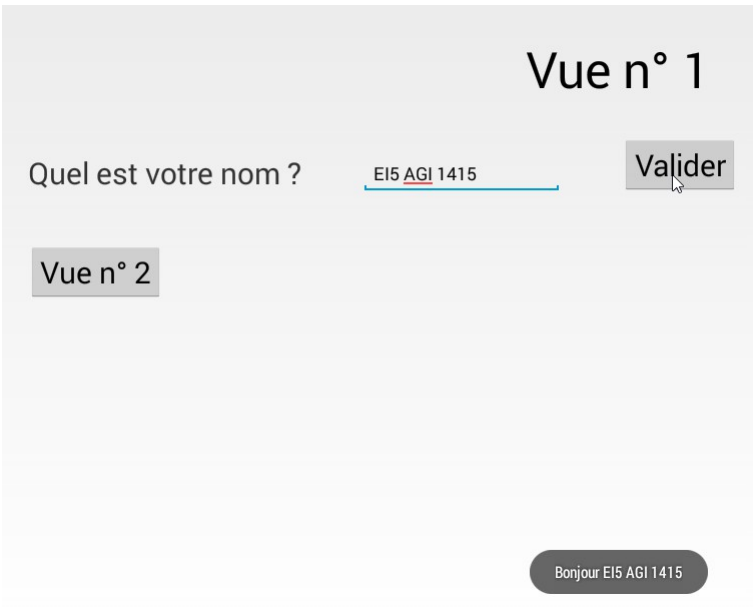
- ligne 6 : il y a désormais quatre fragments ;
- ligne 25 : le fragment supplémentaire est de type [Vue1Fragment_] (attention à l'underscore) ;
- lignes 53-54 : le titre de la quatrième vue définie dans [strings.xml] :

```
1.     <!-- vue 1 -->
2.     <string name="titre_vue1">Vue n° 1</string>
3.     <string name="txt_nom">Quel est votre nom ?</string>
4.     <string name="btn_valider">Valider</string>
5.     <string name="btn_vue2">Vue n° 2</string>
6.   <string name="title_section4">Section 4</string>
```

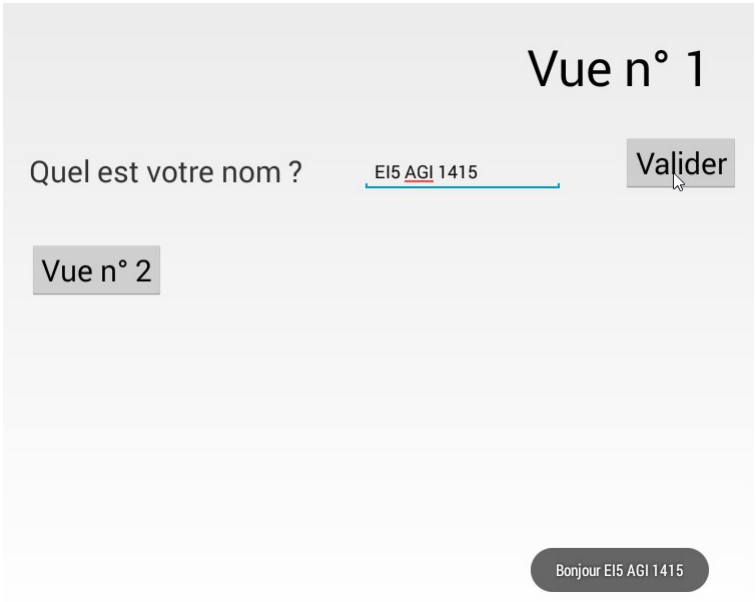
Compilez puis exécutez le projet [exemple-07]. Vous devez avoir une vue de plus :



Si nous tapons un nom et que nous validons, on obtient la vue suivante :



Maintenant passez sur un autre onglet et revenez sur l'onglet [Section 4]. On obtient la vue suivante :

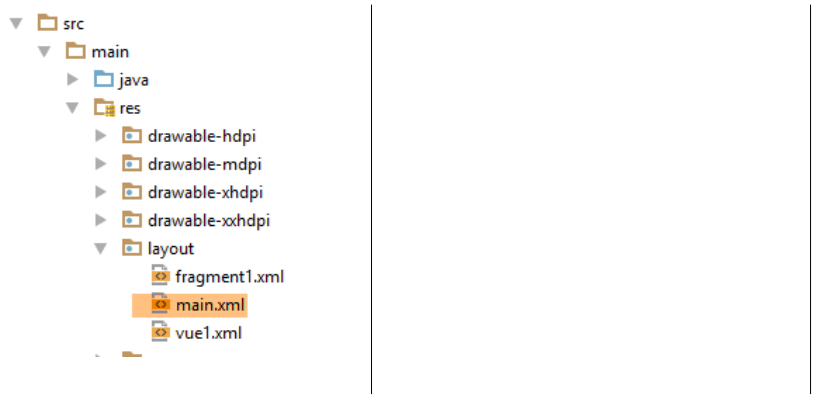


Ci-dessus, le nom saisi a été conservé. Le conteneur de fragments **garde les fragments en mémoire** et ne les régénère pas de nouveau lorsqu'ils doivent être réaffichés. On retrouve ainsi le fragment dans l'état où on l'a laissé.

1.8.6 Désactiver le Swipe ou Balayage

Dans l'application précédente, lorsque vous balayez l'émulateur avec la souris vers la gauche ou la droite, la vue courante laisse alors place à la vue de droite ou de gauche selon les cas. Dans les applications que nous allons écrire, ce comportement ne sera pas souhaitable. On voudra passer d'une vue à une autre seulement si certaines conditions sont remplies. Nous allons apprendre à désactiver le balayage des vues (swipe).

Revenons sur la vue XML principale [main] :



Le code XML de la vue est le suivant :

```
1.  <android.support.v4.view.ViewPager xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools"
3.      android:id="@+id/pager"
4.      android:layout_width="match_parent"
5.      android:layout_height="match_parent"
6.  tools:context="android.exemples.MainActivity" />
```

La ligne 1 désigne la classe qui gère les pages de l'activité. On retrouve cette classe dans l'activité [MainActivity] :

```
1.  package android.exemples;
2.
3.  ...
4.  import android.support.v4.view.ViewPager;
5.
6.  import java.util.Locale;
7.
8.  @EActivity(R.layout.main)
9.  public class MainActivity extends ActionBarActivity implements ActionBar.TabListener {
10.
11.   // le gestionnaire de fragments ou sections
12.   SectionsPagerAdapter mSectionsPagerAdapter;
13.
14.   // le conteneur des fragments
15.   @ViewById(R.id.pager)
16.   ViewPager mViewPager;
17. ...
```

Ligne 16, le gestionnaire de pages est de type [android.support.v4.view.ViewPager] (ligne 4). Pour désactiver le balayage, on est amené à dériver cette classe de la façon suivante :


```
1.  package android.exemples;
2.
3.  import android.content.Context;
4.  import android.support.v4.view.ViewPager;
5.  import android.util.AttributeSet;
6.  import android.view.MotionEvent;
7.
```

```
8.  public class MyPager extends ViewPager {
9.
10.   // contrôle le swipe
11.   boolean isSwipeEnabled;
12.
13.   public MyPager(Context context) {
14.     super(context);
15.   }
16.
17.   public MyPager(Context context, AttributeSet attrs) {
18.     super(context, attrs);
19.   }
20.
21.   @Override
22.   public boolean onInterceptTouchEvent(MotionEvent event) {
23.     // swipe autorisé ?
24.     if (isSwipeEnabled) {
25.       return super.onInterceptTouchEvent(event);
26.     } else {
27.       return false;
28.     }
29.   }
30.
31.   @Override
32.   public boolean onTouchEvent(MotionEvent event) {
33.     // swipe autorisé ?
34.     if (isSwipeEnabled) {
35.       return super.onTouchEvent(event);
36.     } else {
37.       return false;
38.     }
39.   }
40.
41.   // setter
42.   public void setSwipeEnabled(boolean isSwipeEnabled) {
43.     this.isSwipeEnabled = isSwipeEnabled;
44.   }
45.
46. }
```

- ligne 8 : la classe [MyPager] étend la classe [ViewPager] ;
- sur un balayage de la main, les gestionnaires d'événements des lignes 22 et 32 peuvent être appelés. Elles rendent toutes deux un booléen. Il leur suffit de rendre le booléen [false] pour inhiber le balayage ;
- ligne 11 : le booléen qui sert à indiquer si on accepte ou non le balayage de la main.

Ceci fait, il faut utiliser désormais notre nouveau gestionnaire de pages. Cela se fait dans la vue XML [main.xml] et dans l'activité principale [MainActivity]. Dans [main.xml] on écrit :

```
1.  <android.exemples.MyPager xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools"
3.      android:id="@+id/pager"
4.      android:layout_width="match_parent"
5.      android:layout_height="match_parent"
6.   tools:context="android.exemples.MainActivity" />
```

Ligne 1, on utilise la nouvelle classe. Dans [MainActivity], le code évolue comme suit :

```
1.  @EActivity(R.layout.main)
2.  public class MainActivity extends ActionBarActivity implements ActionBar.TabListener {
3.
4.    ...
5.
6.    // le conteneur des fragments
7.    @ViewById(R.id.pager)
8.    MyPager mViewPager;
9.
10.   @AfterViews
11.   protected void initActivity() {
12.
13.     // on inhibe le swipe entre fragments
14.     mViewPager.setSwipeEnabled(false);
15.
```

```
16.     // la barre d'onglets
17.     final ActionBar actionBar = getSupportActionBar();
18.     actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
19. ...
20.   }
21. ...
```

- ligne 8 : le gestionnaire de pages a désormais le type [MyPager] ;
- ligne 14 : on inhibe ou non le balayage de la main.

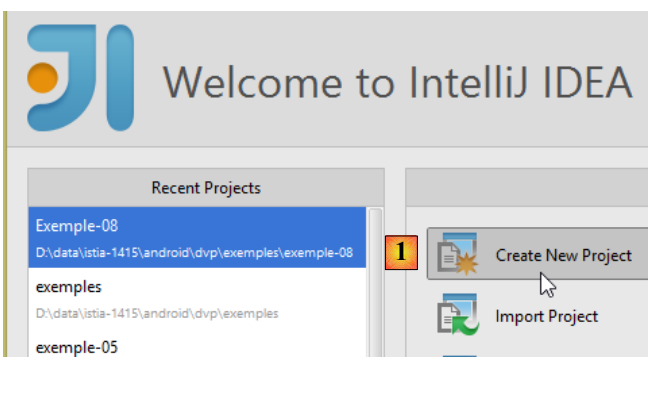
Testez cette nouvelle version. Inhibez ou non le balayage et constatez la différence de comportement des vues. Dans toutes les applications à venir, le balayage sera inhibé. Nous ne le rappellerons pas.

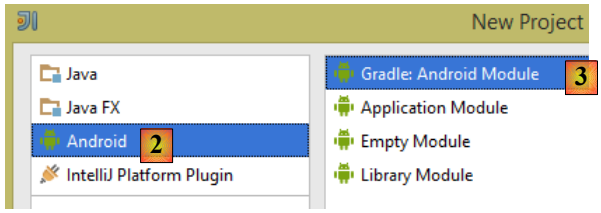
1.9 Exemple-08 : La navigation entre vues revisitée

Dans le projet [exemple-05] nous avons introduit la navigation entre vues. Il s'agissait alors d'une navigation entre activités : **1 vue = 1 activité**. Nous nous proposons ici d'avoir 1 activité avec plusieurs vues. L'activité sera de type [FragmentActivity] et la vue de type [Fragment].

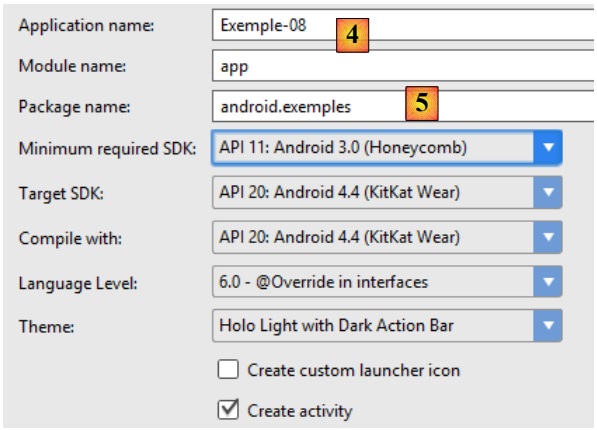
1.9.1 Création du projet

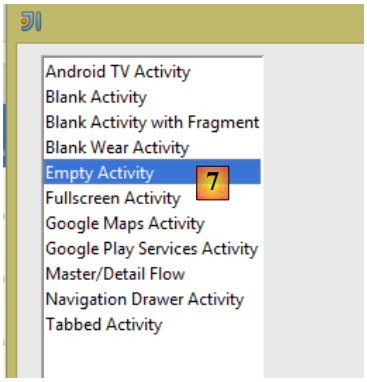
Nous créons un nouveau projet [exemple-08] :



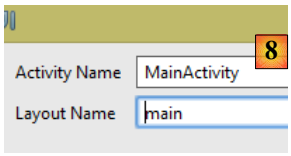


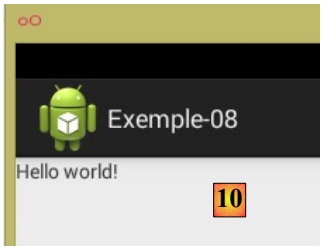
- en [1], on crée un nouveau projet ;
- en [2-3], de type [Gradle:Android Module] ;

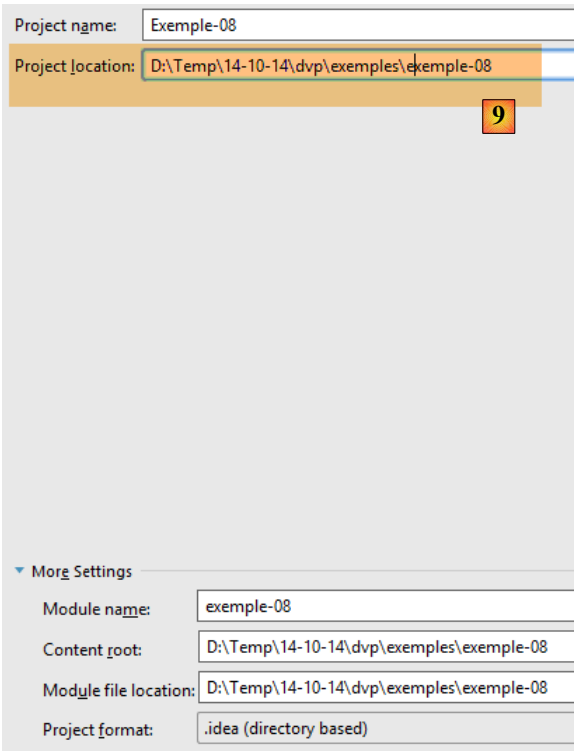




- en [4], on donne un nom au projet ;
- en [5], le package des sources ;
- en [7], on prend le modèle [Empty Activity] qui va créer un projet avec une activité qui ne fait quasiment rien ;

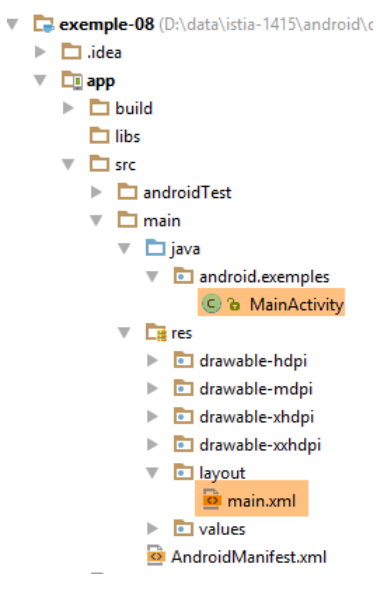






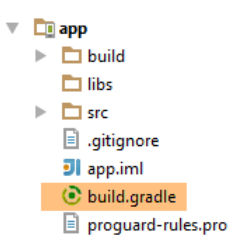
- en [8], on nomme l'activité et sa vue ;
- en [9], on place le projet dans le dossier <exemples> comme les projets précédents ;
- en [10], on exécute le projet ;

L'arborescence du nouveau projet est la suivante :



1.9.2 Configuration Gradle du projet

Nous allons utiliser la bibliothèque AA. Il nous faut donc un fichier [build.gradle] adapté.



Nous remplaçons le fichier [build.gradle] ci-dessus par celui du projet [exemple-07] :

```
1.  buildscript {
2.    repositories {
3.      mavenCentral()
4.      mavenLocal()
5.    }
6.
7.    dependencies {
8.      // replace with the current version of the Android plugin
9.      classpath 'com.android.tools.build:gradle:1.0.0'
10.     // Since Android's Gradle plugin 0.11, you have to use android-apt >= 1.3
11.     classpath 'com.neenbedankt.gradle.plugins:android-apt:1.4'
12.   }
13. }
14.
15. apply plugin: 'com.android.application'
16. apply plugin: 'android-apt'
17. def AAVersion = '3.1'
18.
19. dependencies {
20.   apt "org.androidannotations:androidannotations:$AAVersion"
21.   compile "org.androidannotations:androidannotations-api:$AAVersion"
22.   compile 'com.android.support:appcompat-v7:20.+'
23.   compile fileTree(dir: 'libs', include: ['*.jar'])
24. }
25.
26. repositories {
27.   jcenter()
28. }
29.
30. apt {
31.   arguments {
32.     androidManifestFile variant.outputs[0].processResources.manifestFile
33.     resourcePackageName android.defaultConfig.applicationId
34.   }
35. }
36.
37. android {
38.   compileSdkVersion 20
39.   buildToolsVersion "20.0.0"
40.   defaultConfig {
41.     applicationId "android.exemples"
42.     minSdkVersion 11
43.     targetSdkVersion 20
44.     versionCode 1
45.     versionName "1.0"
46.   }
47.
48.   compileOptions {
49.     sourceCompatibility JavaVersion.VERSION_1_6
50.     targetCompatibility JavaVersion.VERSION_1_6
51.   }
52.
53.   buildTypes {
54.     release {
55.       minifyEnabled false
56.       proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
57.     }
58.   }
59. }
```

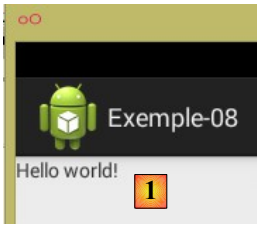
Ajoutons l'annotation [@EActivity] à l'activité [MainActivity] (ligne 7 ci-dessous) :

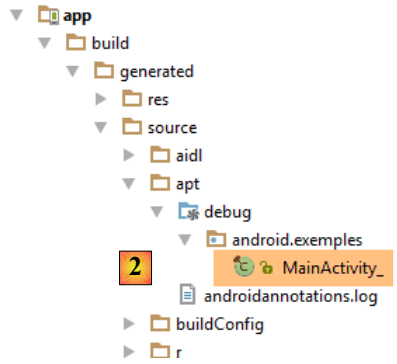
```
1.  package android.exemples;
2.
3.  import android.app.Activity;
4.  import android.os.Bundle;
5.  import org.androidannotations.annotations.EActivity;
6.
7.  @EActivity(R.layout.main)
8.  public class MainActivity extends Activity {
9.
10.   @Override
11.   protected void onCreate(Bundle savedInstanceState) {
12.     super.onCreate(savedInstanceState);
13.   }
14. }
```

Pour finir, modifions l'activité dans [AndroidManifest.xml] (ligne 11 ci-dessous - notez l'underscore de [MainActivity_]) :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.      package="android.exemples" >
4.
5.      <application
6.          android:allowBackup="true"
7.          android:icon="@drawable/ic_launcher"
8.          android:label="@string/app_name"
9.          android:theme="@style/AppTheme" >
10.         <activity
11.             android:name=".MainActivity_"
12.             android:label="@string/app_name" >
13.             <intent-filter>
14.                 <action android:name="android.intent.action.MAIN" />
15.
16.                 <category android:name="android.intent.category.LAUNCHER" />
17.             </intent-filter>
18.         </activity>
19.     </application>
20.
21. </manifest>
```

A ce stade, exécutez le projet [exemple-08]. Il doit fonctionner avec cette nouvelle configuration [1] :

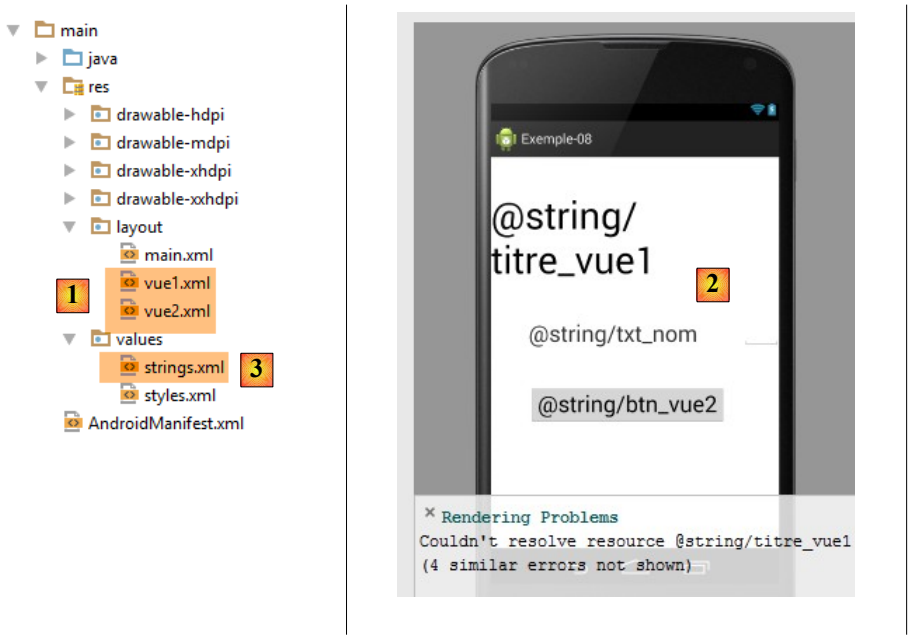




- notez en [2], la génération de la classe [MainActivity_] ;

1.9.3 Création des fragment et des vues associées

L'application aura deux vues, celles du projet [exemple-05]. Nous nous contentons de copier les vues XML [vue1.xml, vue2.xml] de ce projet dans le nouveau :

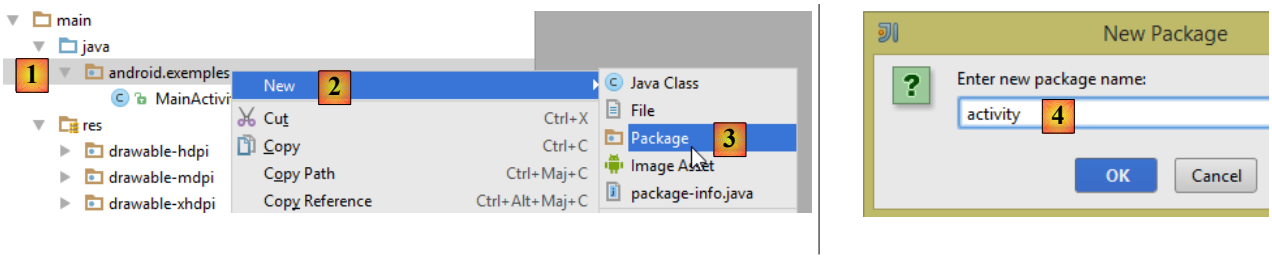


- en [1], les nouvelles vues. Lorsqu'on essaie de les éditer, des erreurs apparaissent [2]. Il nous faut modifier le fichier [strings.xml] [3] pour y ajouter les chaînes référencées par les différentes vues :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.    <string name="app_name">exemple-08</string>
4.    <string name="hello_world">Hello world!</string>
5.    <string name="titre_vue1">Vue n° 1</string>
6.    <string name="txt_nom">Quel est votre nom ?</string>
7.    <string name="btn_valider">Valider</string>
8.    <string name="btn_vue2">Vue n° 2</string>
9.    <string name="titre_vue2">Vue n° 2</string>
10.   <string name="btn_vue1">Vue n° 1</string>
11. </resources>
```

Ce changement du contenu peut parfois ne pas être pris en compte par l'IDE qui continue alors à signaler des erreurs sur les vues. Rafraîchir le projet [Gradle] (View / Tool windows / Gradle) ou quitter / relancer l'IDE résoud en général le problème.

Maintenant, nous devons créer les fragments. Nous allons organiser le code en packages :



- en [1-5], nous créons un package [activity] ;
- en [6], nous refaisons la même opération pour le package [fragments] ;

- en [7], nous déplaçons la classe [MainActivity] dans le package [activity] ;

Parce que l'activité a changé d'emplacement, il faut modifier le fichier [AndroidManifest.xml] (ligne 11 ci-dessous) :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.      package="android.exemples" >
4.
5.      <application
6.          android:allowBackup="true"
7.          android:icon="@drawable/ic_launcher"
8.          android:label="@string/app_name"
9.          android:theme="@style/AppTheme" >
10.         <activity
11.             android:name=".activity.MainActivity_"
12.             android:label="@string/app_name" >
13.             <intent-filter>
14.                 <action android:name="android.intent.action.MAIN" />
15.
16.                 <category android:name="android.intent.category.LAUNCHER" />
17.             </intent-filter>
18.         </activity>
19.     </application>
20.
21. </manifest>
```

Dans une activité avec fragments, la vue XML [main.xml] est le conteneur des fragments. Pour l'instant ce n'est pas le cas :

```
1.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools"
3.      android:layout_width="match_parent"
4.      android:layout_height="match_parent"
5.      tools:context="${relativePackage}.${activityClass}">
6.
7.      <TextView
8.          android:text="@string/hello_world"
9.          android:layout_width="wrap_content"
10.         android:layout_height="wrap_content" />
11.
12. </RelativeLayout>
```

Nous remplaçons ce contenu par le contenu de la vue [main.xml] du projet [Exemple-07], vue qui était un conteneur de fragments :

```
1.  <android.exemples.MyPager xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools"
3.      android:id="@+id/pager"
4.      android:layout_width="match_parent"
5.      android:layout_height="match_parent"
6.  tools:context="android.exemples.MainActivity" />
```

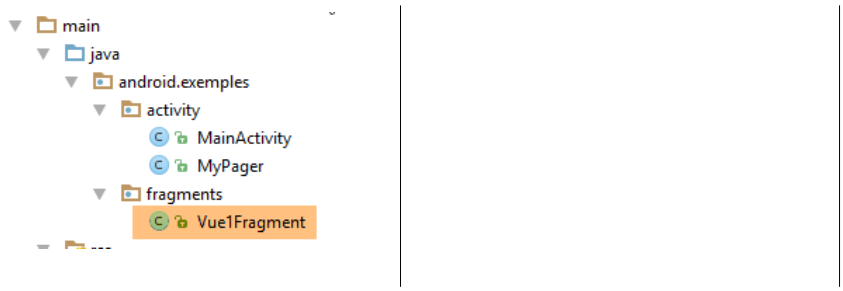
- ligne 1 ci-dessus, la classe [android.exemples.MyPager] est utilisée pour gérer les fragments. Nous la copions du projet [Exemple-07], dans le package [activity] du projet [Exemple-08] :

Du coup, le nom complet de la classe est [android.exemples.activity.MyPager] (ligne 1 ci-dessous) :

```
1.  <android.exemples.activity.MyPager xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools"
```

```
3.      android:id="@+id/pager"
4.      android:layout_width="match_parent"
5.      android:layout_height="match_parent"
6.  tools:context="android.exemples.MainActivity" />
```

Maintenant créons le fragment [Vue1Fragment] associé à la vue XML [vue1.xml] :

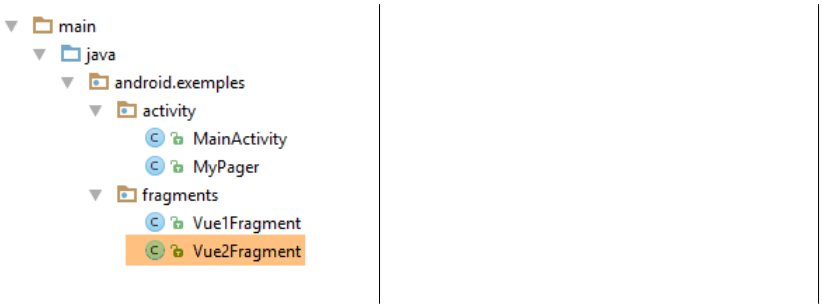


La classe [Vue1Fragment] est la suivante :

```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.support.v4.app.Fragment;
5.  import org.androidannotations.annotations.EFragment;
6.
7.  @EFragment(R.layout.vue1)
8.  public class Vue1Fragment extends Fragment {
9.  }
```

- ligne 7 : le fragment est associé à la vue XML [vue1.xml] ;

Nous dupliquons la classe [Vue1Fragment] dans [Vue2Fragment] :



et nous modifions le code copié de la façon suivante :

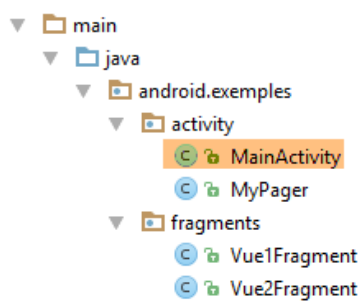
```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.support.v4.app.Fragment;
5.  import org.androidannotations.annotations.EFragment;
6.
7.  @EFragment(R.layout.vue2)
8.  public class Vue1Fragment extends Fragment {
9.  }
```

- ligne 7 : le fragment est associé à la vue XML [vue2.xml] ;

A ce stade, nous avons un projet compilable mais pas encore exécutable. Notre activité n'est pas capable de gérer les fragments que nous avons créés. Nous allons la modifier.

1.9.4 Gestion des fragments par l'activité

L'activité [MainActivity] est pour l'instant la suivante :




```
1.  package android.exemples.activity;
2.
3.  import android.app.Activity;
4.  import android.exemples.R;
5.  import android.os.Bundle;
6.  import org.androidannotations.annotations.EActivity;
7.
8.  @EActivity(R.layout.main)
9.  public class MainActivity extends Activity {
10.
11.   @Override
12.   protected void onCreate(Bundle savedInstanceState) {
13.     super.onCreate(savedInstanceState);
14.   }
15. }
```

Nous allons lui ajouter le gestionnaire de fragments que nous avons utilisé dans l'activité du projet [Exemple-07] et que nous adaptons au nouveau contexte :

```
1.  package android.exemples.activity;
2.
3.  import android.exemples.R;
4.  import android.exemples.fragments.Vue1Fragment_;
5.  import android.exemples.fragments.Vue2Fragment_;
6.  import android.os.Bundle;
7.  import android.support.v4.app.Fragment;
8.  import android.support.v4.app.FragmentActivity;
9.  import android.support.v4.app.FragmentManager;
10. import android.support.v4.app.FragmentPagerAdapter;
11. import org.androidannotations.annotations.AfterViews;
12. import org.androidannotations.annotations.EActivity;
13. import org.androidannotations.annotations.ViewById;
14.
15. @EActivity(R.layout.main)
16. public class MainActivity {
17.
18.   ...
19.
20.   // notre gestionnaire de fragments à redéfinir pour chaque application
21.   // doit définir les méthodes suivantes : getItem, getCount, getPageTitle
22.   public class SectionsPagerAdapter extends FragmentPagerAdapter {
23.
24.     // les fragments
25.     private final Fragment[] fragments = {new Vue1Fragment_(), new Vue2Fragment_()};
26.     private final String[] titres = {getString(R.string.titre_vue1), getString(R.string.titre_vue2)};
27.
28.     // constructeur
29.     public SectionsPagerAdapter(FragmentManager fm) {
30.       super(fm);
31.     }
32.
33.
34.     @Override
35.     public Fragment getItem(int position) {
36.       // fragment n° position
37.       return fragments[position];
38.     }
```

```
39.
40.     // rend le nombre de fragments à gérer
41.     @Override
42.     public int getCount() {
43.       // nb de fragments
44.       return fragments.length;
45.     }
46.
47.     // rend le titre du fragment n° position
48.     @Override
49.     public CharSequence getPageTitle(int position) {
50.       return titres[position];
51.     }
52.   }
53.
54. }
```

- les deux fragments sont définis ligne 25. On les met dans un tableau ;
- les titres des deux fragments sont définis ligne 26. La méthode [getString] permet d'aller récupérer une chaîne de caractères dans le fichier [strings.xml] (lignes 5 et 9 ci-dessous) :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.    <string name="app_name">exemple-08</string>
4.    <string name="hello_world">Hello world!</string>
5.    <string name="titre_vue1">Vue n° 1</string>
6.    <string name="txt_nom">Quel est votre nom ?</string>
7.    <string name="btn_valider">Valider</string>
8.    <string name="btn_vue2">Vue n° 2</string>
9.    <string name="titre_vue2">Vue n° 2</string>
10.   <string name="btn_vue1">Vue n° 1</string>
11. </resources>
```

- le reste est standard ;

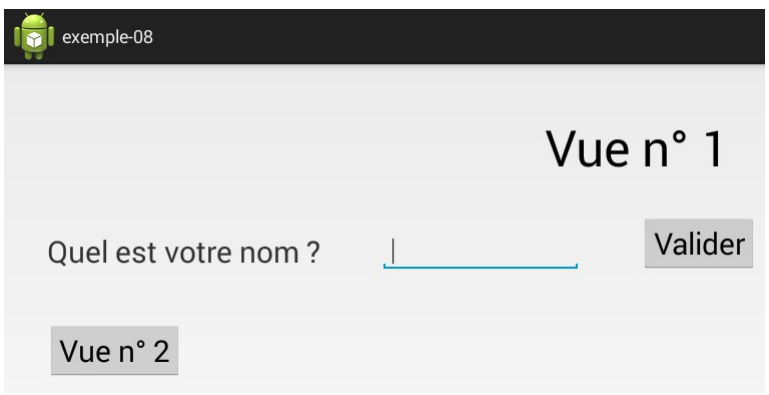
Ce gestionnaire de fragments doit maintenant être utilisé par l'activité [MainActivity] :

```
1.  package android.exemples.activity;
2.
3.  import android.exemples.R;
4.  import android.exemples.fragments.Vue1Fragment_;
5.  import android.exemples.fragments.Vue2Fragment_;
6.  import android.os.Bundle;
7.  import android.support.v4.app.Fragment;
8.  import android.support.v4.app.FragmentActivity;
9.  import android.support.v4.app.FragmentManager;
10. import android.support.v4.app.FragmentPagerAdapter;
11. import org.androidannotations.annotations.AfterViews;
12. import org.androidannotations.annotations.EActivity;
13. import org.androidannotations.annotations.ViewById;
14.
15. @EActivity(R.layout.main)
16. public class MainActivity extends FragmentActivity {
17.
18.   // le gestionnaire de fragments ou sections
19.   SectionsPagerAdapter mSectionsPagerAdapter;
20.
21.   // le conteneur des fragments
22.   @ViewById(R.id.pager)
23.   MyPager mViewPager;
24.
25.   @Override
26.   protected void onCreate(Bundle savedInstanceState) {
27.     super.onCreate(savedInstanceState);
28.   }
29.
30.   @AfterViews
31.   protected void initActivity() {
32.
33.     // on inhibe le swipe entre fragments
34.     mViewPager.setSwipeEnabled(false);
35.
36.     // instanciation du gestionnaire de fragments
```

```
37.     mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
38.
39.     // qu'on associe à notre conteneur de fragments
40.     mViewPager.setAdapter(mSectionsPagerAdapter);
41.
42.   }
43.
44.   // notre gestionnaire de fragments à redéfinir pour chaque application
45.   // doit définir les méthodes suivantes : getItem, getCount, getPageTitle
46.   public class SectionsPagerAdapter extends FragmentPagerAdapter {
47. ...
48. }
```

- lignes 22-23 : la référence du composant de type [MyPager] de la vue [main.xml] est injectée dans le champ [mViewPager] de la ligne 23 ;
- lignes 30-31 : la méthode [initActivity] va être exécutée après cette injection ;
- ligne 34 : le balayage est inhibé ;
- ligne 37 : le gestionnaire de fragments est instancié. Le paramètre du constructeur est le résultat d'une méthode [getSupportFragmentManager] qui n'existe pas par défaut dans une activité. Il faut étendre la classe [FragmentActivity] (ligne 16) pour l'obtenir ;
- ligne 40 : le conteneur de fragments est associé au gestionnaire de fragments ;

Exécutez le projet. Le premier fragment est affiché. Pour nous c'est la vue [Vue n° 1].



1.9.5 Mise en place de la navigation

C'est la classe [MainActivity] qui va assurer la navigation. Cette activité est accessible à tous les fragments. Aussi l'utiliserons-nous également pour passer de l'information d'un fragment à un autre. Le code de [MainActivity] évolue comme suit :

```
1.    // navigation
2.    public void navigateToView(int i) {
3.      // on affiche le fragment n° i
4.      mViewPager.setCurrentItem(i);
5.  }
```

- ligne 4 : la navigation entre fragments est une fonctionnalité du conteneur de fragments ;

Le fragment [Vue1Fragment] évolue comme suit :

```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.exemples.activity.MainActivity;
5.  import android.support.v4.app.Fragment;
6.  import org.androidannotations.annotations.AfterViews;
7.  import org.androidannotations.annotations.Click;
8.  import org.androidannotations.annotations.EFragment;
9.
10. @EFragment(R.layout.vue1)
11. public class Vue1Fragment extends Fragment {
```

```
12.
13.   // activité
14.   private MainActivity activity;
15.
16.   @AfterViews
17.   public void initFragment(){
18.     // on mémorise l'activité
19.     activity=(MainActivity) getActivity();
20.   }
21.
22.   @Click(R.id.buttonVue2)
23.   protected void showVue2(){
24.     activity.navigateToView(1);
25.   }
26. }
```

- lignes 16-17 : l'exécution de la méthode [initFragment] a lieu après les injections de composants de l'interface visuelle. Ici, il n'y en a pas. Aux tests, on constate que la ligne 19 rend bien l'activité de l'application. Ce n'est pas vrai si on met cette instruction dans le constructeur ;
- lignes 22-23 : la méthode [showVue2] gère l'événement 'click' sur le bouton [Vue n° 2] ;
- ligne 24 : on navigue avec la méthode [navigateToView] de l'activité ;

On retrouve un code analogue pour le fragment [Vue2Fragment] :

```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.exemples.activity.MainActivity;
5.  import android.support.v4.app.Fragment;
6.  import org.androidannotations.annotations.AfterViews;
7.  import org.androidannotations.annotations.Click;
8.  import org.androidannotations.annotations.EFragment;
9.
10. @EFragment(R.layout.vue2)
11. public class Vue2Fragment extends Fragment {
12.   // activité
13.   private MainActivity activity;
14.
15.   @AfterViews
16.   public void initFragment() {
17.     // on mémorise l'activité
18.     activity = (MainActivity) getActivity();
19.   }
20.
21.   @Click(R.id.buttonVue1)
22.   protected void showVue1() {
23.     activity.navigateToView(0);
24.   }
25.
26. }
```

Exécutez de nouveau le projet et vérifiez la navigation.

1.9.6 Ecriture finale des fragments

Le fragment [Vue1Fragment] doit gérer certains événements :

```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.exemples.activity.MainActivity;
5.  import android.support.v4.app.Fragment;
6.  import android.widget.EditText;
7.  import android.widget.Toast;
8.  import org.androidannotations.annotations.AfterViews;
9.  import org.androidannotations.annotations.Click;
10. import org.androidannotations.annotations.EFragment;
11. import org.androidannotations.annotations.ViewById;
12.
13. @EFragment(R.layout.vue1)
14. public class Vue1Fragment extends Fragment {
15.
```

```
16.   // activité
17.   private MainActivity activity;
18.   // les éléments de l'interface visuelle
19.   @ViewById(R.id.editTextNom)
20.   protected EditText editTextNom;
21.
22.   @AfterViews
23.   public void initFragment() {
24.     // on mémorise l'activité
25.     activity = (MainActivity) getActivity();
26.   }
27.
28.   @Click(R.id.buttonVue2)
29.   protected void showVue2() {
30.     // on récupère le nom saisi
31.     String nom=editTextNom.getText().toString().trim();
32.     // on le met dans l'activité
33.     activity.setNom(nom);
34.     // on navigue
35.     activity.navigateToView(1);
36.   }
37.
38.   @Click(R.id.buttonValider)
39.   protected void doValider() {
40.     // on récupère le nom saisi
41.     String nom=editTextNom.getText().toString().trim();
42.     // on le met dans l'activité
43.     activity.setNom(nom);
44.     // on l'affiche
45.     Toast.makeText(activity, String.format("Bonjour %s",nom), Toast.LENGTH_LONG).show();
46.   }
47.
48. }
```

- lignes 38-46 : on gère le 'click' sur le bouton [Valider] ;
- ligne 43 : le nom saisi est mémorisé dans l'activité ;
- ligne 45 : le nom saisi est affiché ;

La vue [Vue2Fragment] évolue de la façon suivante :

```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.exemples.activity.MainActivity;
5.  import android.support.v4.app.Fragment;
6.  import android.widget.TextView;
7.  import org.androidannotations.annotations.AfterViews;
8.  import org.androidannotations.annotations.Click;
9.  import org.androidannotations.annotations.EFragment;
10. import org.androidannotations.annotations.ViewById;
11.
12. @EFragment(R.layout.vue2)
13. public class Vue2Fragment extends Fragment {
14.   // activité
15.   private MainActivity activity;
16.   // composants de l'interface visuelle
17.   @ViewById
18.   protected TextView textViewBonjour;
19.
20.   @AfterViews
21.   public void initFragment() {
22.     // on mémorise l'activité
23.     activity = (MainActivity) getActivity();
24.   }
25.
26.   @Click(R.id.buttonVue1)
27.   protected void showVue1() {
28.     // on navigue
29.     activity.navigateToView(0);
30.   }
31.
32.   // juste avant affichage
33.   public void setMenuVisibility(final boolean visible) {
```

```
34.     super.setMenuVisibility(visible);
35.     if (visible) {
36.       // la vue est visible - on affiche le nom saisi dans la vue 1
37.       textViewBonjour.setText(String.format("Bonjour %s !", activity.getNom()));
38.     }
39.   }
40. }
```

Lorsque la vue n° 2 s'affiche, il faut afficher le nom saisi dans la vue n° 1. Un fragment n'est créé qu'une fois. Aussi les méthodes telles que [onStart, onResume] ne sont-elles appelées qu'une fois, lors de la création initiale du fragment. Il nous faut un événement qui nous dise que le fragment est devenu visible. C'est l'événement géré par la méthode [setMenuVisibility] des lignes 33-38.

La classe [MainActivity] s'enrichit d'un nouveau champ [nom] utilisé pour la communication inter-fragments (ligne 6 ci-dessous) ;

```
1.    // le conteneur des fragments
2.    @ViewById(R.id.pager)
3.    MyPager mViewPager;
4.
5.    // données partagées entre fragments
6.    protected String nom;
7.
8.      // getters et setters
9.    public String getNom() {
10.     return nom;
11.   }
12.
13.   public void setNom(String nom) {
14.     this.nom = nom;
15. }
16. ...
```

Exécutez le projet et vérifiez qu'il fonctionne.

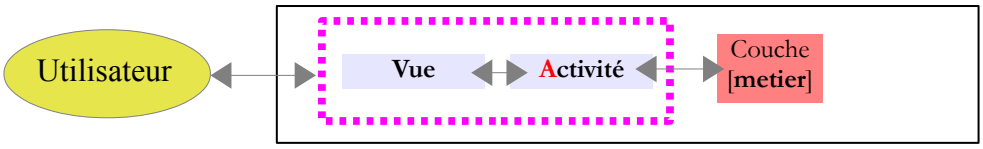
1.9.7 Conclusion

A ce point, nous avons un début d'architecture cohérent pour une application à plusieurs vues :

- une seule activité gère toutes les vues ;
- elle gère la navigation et la transmission d'information entre celles-ci.

1.10 Exemple-09 : une architecture à deux couches

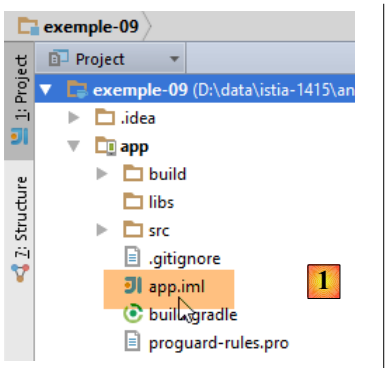
Nous allons construire une application à une vue ayant l'architecture suivante :

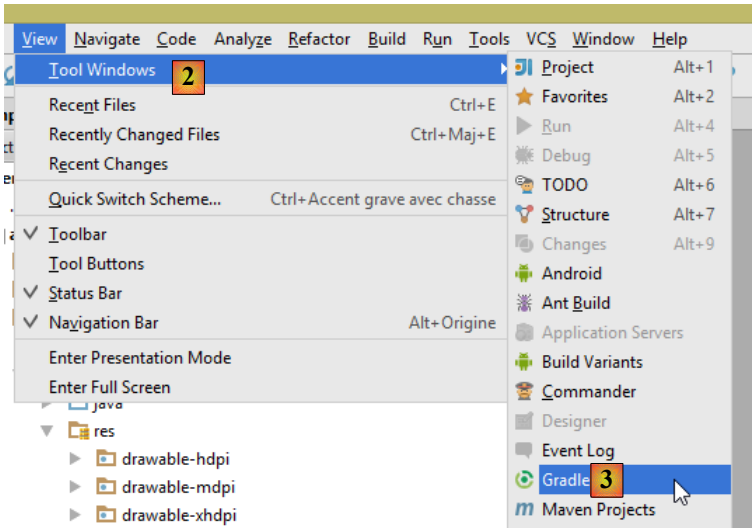


1.10.1 Création du projet

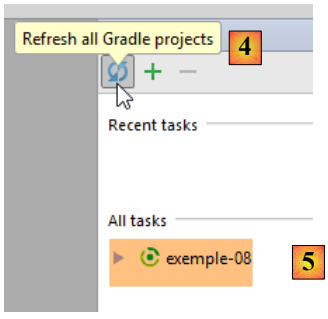
Pour créer le nouveau projet, nous :

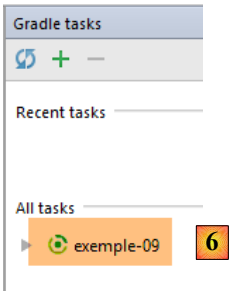
- dupliquons le projet [exemple-08] dans [exemple-09] (copy / paste) ;
- chargeons le nouveau projet [exemple-09] ;
- modifions le nom du projet et le nom du module en [exemple-09] (project structure) ;
- modifions le nom du projet dans [res / values / strings.xml] ;





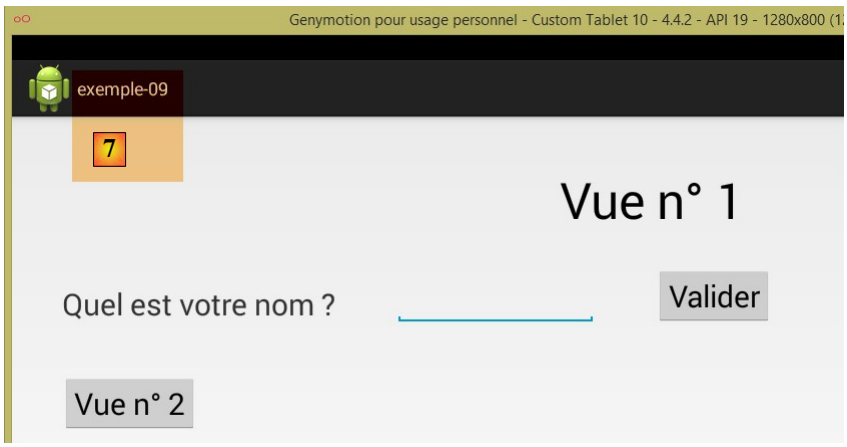
- en [2-3], on affiche la fenêtre de [Gradle] ;





- en [4], on rafraîchit les projets [Gradle] ;
- le nom du projet courant passe de [exemple-08] [5] à [exemple-09] [6] ;

Puis on compile et exécute l'application :



- en [7], on obtient la vue attendue avec le nom [exemple-09] du nouveau projet ;

Parfois, cela ne marche pas et on a toujours en [7] la chaîne [exemple-08]. Il faut alors vérifier deux points (non liés l'un à l'autre - l'ordre n'a pas d'importance) :

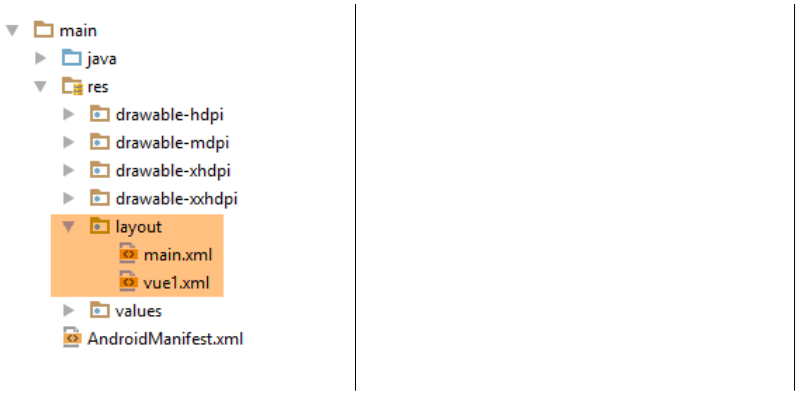
- dans [res / values / strings.xml], on doit avoir la chaîne :

```
<string name="app_name">exemple-09</string>
```

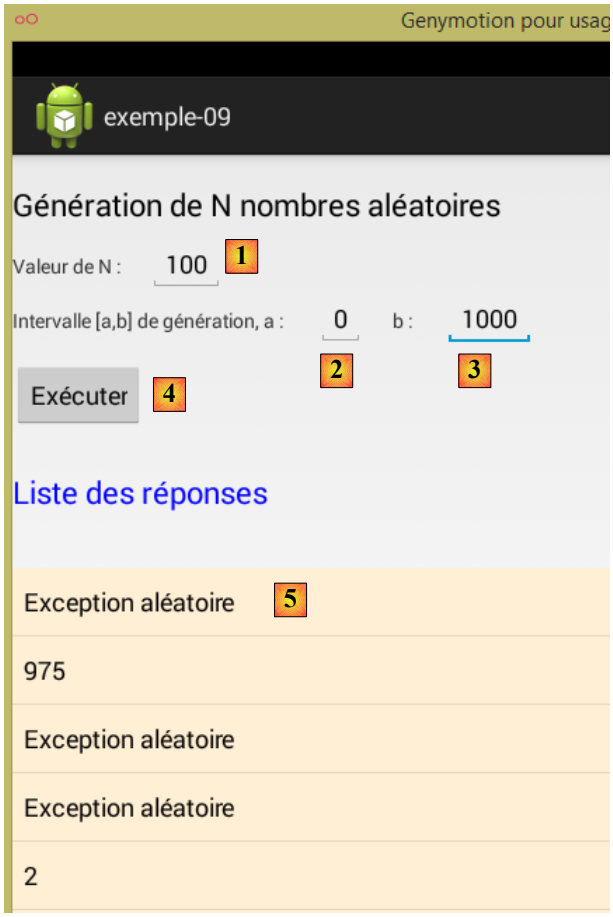
- vous devez rafraîchir le projet [Gradle] ;

1.10.2 La vue [vue1]

L'application n'aura qu'une vue [vue1.xml]. Aussi supprimons-nous l'autre vue [vue2.xml] :



Nous allons créer la vue [vue1.xml] qui permettra de générer des nombres aléatoires :



Ses composants sont les suivants :

N°	Id	Type	Rôle
1	edtNbAleas	EditText	nombre de nombres aléatoires à générer dans l'intervalle entier [a,b]
2	edtA	EditText	valeur de a
2	edtB	EditText	valeur de b
4	btnExécuter	Button	lance la génération des nombres
5	ListView	lstReponses	liste des nombres générés dans l'ordre inverse de leur génération. On voit d'abord le dernier généré ;

Son code XML est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:tools="http://schemas.android.com/tools"
4.      android:id="@+id/RelativeLayout1"
5.      android:layout_width="match_parent"
6.      android:layout_height="match_parent"
7.      android:orientation="vertical" >
8.
9.      <TextView
10.         android:id="@+id/txt_Titre2"
11.         android:layout_width="wrap_content"
12.         android:layout_height="wrap_content"
13.         android:layout_marginTop="20dp"
14.         android:text="@string/aleas"
15.         android:textAppearance="?android:attr/textAppearanceLarge" />
16.
17.     <TextView
18.         android:id="@+id/txt_nbaleas"
19.         android:layout_width="wrap_content"
20.         android:layout_height="wrap_content"
21.         android:layout_below="@+id/txt_Titre2"
22.         android:layout_marginTop="20dp"
```

```
23.         android:text="@string/txt_nbaleas" />
24.
25.     <EditText
26.         android:id="@+id/edt_nbaleas"
27.         android:layout_width="wrap_content"
28.         android:layout_height="wrap_content"
29.         android:layout_alignBaseline="@+id/txt_nbaleas"
30.         android:layout_marginLeft="20dp"
31.         android:layout_toRightOf="@+id/txt_nbaleas"
32.         android:inputType="number" />
33.
34.     <TextView
35.         android:id="@+id/txt_errorNbAleas"
36.         android:layout_width="wrap_content"
37.         android:layout_height="wrap_content"
38.         android:layout_alignBaseline="@+id/edt_nbaleas"
39.         android:layout_marginLeft="20dp"
40.         android:layout_toRightOf="@+id/edt_nbaleas"
41.         android:text="@string/txt_errorNbAleas"
42.         android:textColor="@color/red" />
43.
44.     <TextView
45.         android:id="@+id/txt_a"
46.         android:layout_width="wrap_content"
47.         android:layout_height="wrap_content"
48.         android:layout_below="@+id/txt_nbaleas"
49.         android:layout_marginTop="20dp"
50.         android:text="@string/txt_a" />
51.
52.     <EditText
53.         android:id="@+id/edt_a"
54.         android:layout_width="wrap_content"
55.         android:layout_height="wrap_content"
56.         android:layout_alignBaseline="@+id/txt_a"
57.         android:layout_marginLeft="20dp"
58.         android:layout_toRightOf="@+id/txt_a"
59.         android:inputType="number" />
60.
61.     <TextView
62.         android:id="@+id/txt_b"
63.         android:layout_width="wrap_content"
64.         android:layout_height="wrap_content"
65.         android:layout_alignBaseline="@+id/txt_a"
66.         android:layout_marginLeft="20dp"
67.         android:layout_toRightOf="@+id/edt_a"
68.         android:text="@string/txt_b" />
69.
70.     <EditText
71.         android:id="@+id/edt_b"
72.         android:layout_width="wrap_content"
73.         android:layout_height="wrap_content"
74.         android:layout_alignBaseline="@+id/txt_a"
75.         android:layout_marginLeft="20dp"
76.         android:layout_toRightOf="@+id/txt_b"
77.         android:inputType="number" />
78.
79.     <TextView
80.         android:id="@+id/txt_errorIntervalle"
81.         android:layout_width="wrap_content"
82.         android:layout_height="wrap_content"
83.         android:layout_alignBaseline="@+id/edt_b"
84.         android:layout_marginLeft="20dp"
85.         android:layout_toRightOf="@+id/edt_b"
86.         android:text="@string/txt_errorIntervalle"
87.         android:textColor="@color/red" />
88.
89.
90.     <Button
91.         android:id="@+id/btn_Executer"
92.         android:layout_width="wrap_content"
93.         android:layout_height="wrap_content"
94.         android:layout_alignParentLeft="true"
95.         android:layout_below="@+id/txt_a"
96.         android:layout_marginTop="20dp"
97.         android:text="@string/btn_executer" />
```

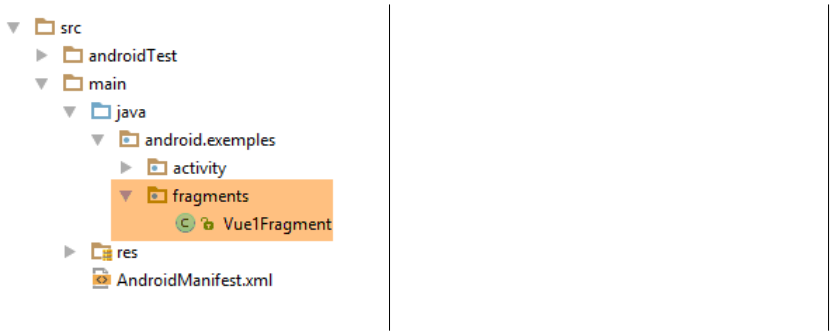
```
98.
99.
100.    <TextView
101.        android:id="@+id/txt_Reponses"
102.        android:layout_width="wrap_content"
103.        android:layout_height="wrap_content"
104.        android:layout_below="@+id/btn_Executer"
105.        android:layout_marginTop="30dp"
106.        android:text="@string/list_reponses"
107.        android:textAppearance="?android:attr/textAppearanceLarge"
108.        android:textColor="@color/blue" />
109.
110.    <ListView
111.        android:id="@+id/lst_reponses"
112.        android:layout_width="match_parent"
113.        android:layout_height="match_parent"
114.        android:layout_alignParentLeft="true"
115.        android:layout_below="@+id/txt_Reponses"
116.        android:layout_marginTop="40dp"
117.        android:background="@color/wheat"
118.        android:clickable="true"
119.        tools:listitem="@android:layout/simple_list_item_1" >
120.    </ListView>
121.
122.</RelativeLayout>
123.
```

La vue précédente utilise des libellés définis dans [res / values / strings.xml] :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.
4.    <string name="app_name">exemple-09</string>
5.    <string name="titre_vue1">Vue n° 1</string>
6.    <string name="action_settings">Settings</string>
7.    <string name="list_reponses">Liste des réponses</string>
8.    <string name="btn_executer">Exécuter</string>
9.    <string name="aleas">Génération de N nombres aléatoires</string>
10.   <string name="txt_nbaleas">Valeur de N :</string>
11.   <string name="txt_a">"Intervalle [a,b] de génération, a : "</string>
12.   <string name="txt_b">"b : "</string>
13.   <string name="txt_dummy">Dummy</string>
14.   <string name="txt_errorNbAleas">Tapez un nombre entier >=1</string>
15.   <string name="txt_errorIntervalle">Les bornes de l\'intervalle doivent être entières et b>=a</string>
16.
17. </resources>
```

1.10.3 Le fragment [Vue1Fragment]

Nous supprimons la classe [Vue2Fragment] qui était associée à la vue [vue2.xml] que nous avons supprimée :



puis nous modifions le fragment [Vue1Fragment] de la façon suivante :

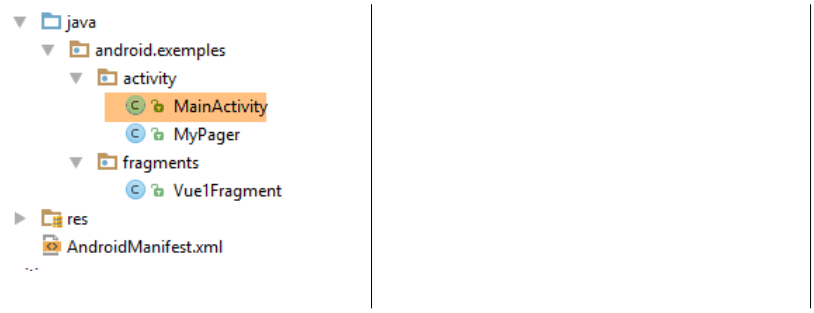
```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
```

```
4.  import android.support.v4.app.Fragment;
5.  import org.androidannotations.annotations.EFragment;
6.
7.  @EFragment(R.layout.vue1)
8.  public class Vue1Fragment extends Fragment {
9.  }
```

On a donc un fragment qui affiche la vue [vue1.xml] (ligne 7) mais ne fait rien d'autre.

1.10.4 L'activité [MainActivity]

Nous devons modifier le gestionnaire de fragments de l'activité : il n'y a plus qu'un fragment :



Le gestionnaire de fragments est modifié de la façon suivante :

```
1.  // notre gestionnaire de fragments à redéfinir pour chaque application
2.    // doit définir les méthodes suivantes : getItem, getCount, getPageTitle
3.    public class SectionsPagerAdapter extends FragmentPagerAdapter {
4.
5.      // les fragments
6.      private final Fragment[] fragments = {new Vue1Fragment_()};
7.      private final String[] titres = {getString(R.string.titre_vue1)};
8.
9.      // le reste ne change pas
10. ...
11.   }
```

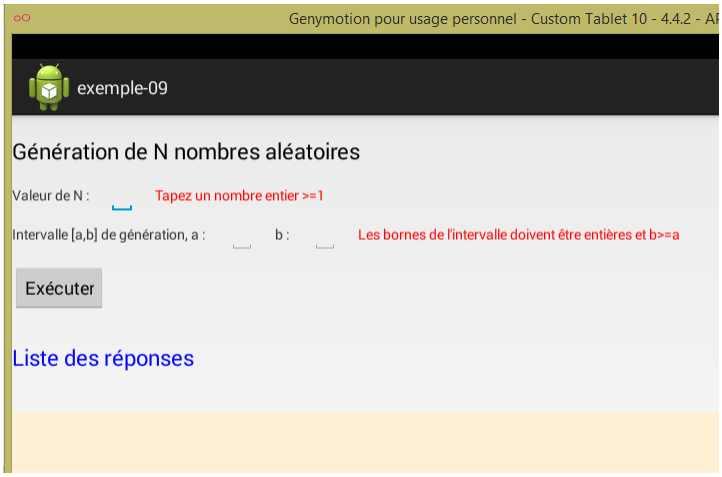
- lignes 6-7 : il n'y a plus que le fragment de type [Vue1Fragment] ;

Par ailleurs, les données stockées dans l'activité pour permettre la communication entre les deux fragments sont supprimées :

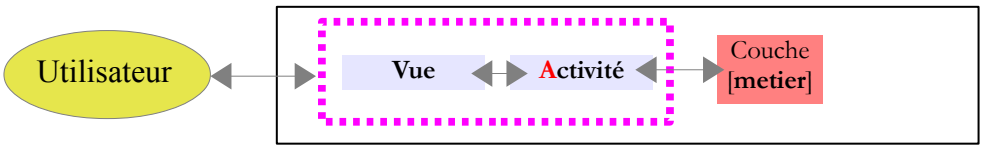
```
1.  @EActivity(R.layout.main)
2.  public class MainActivity extends FragmentActivity {
3.
4.    // le gestionnaire de fragments ou sections
5.    SectionsPagerAdapter mSectionsPagerAdapter;
6.
7.    // le conteneur des fragments
8.    @ViewById(R.id.pager)
9.    MyPager mViewPager;
10.
11.   // données partagées entre fragments
12.   protected String nom;
13.
14. ...
```

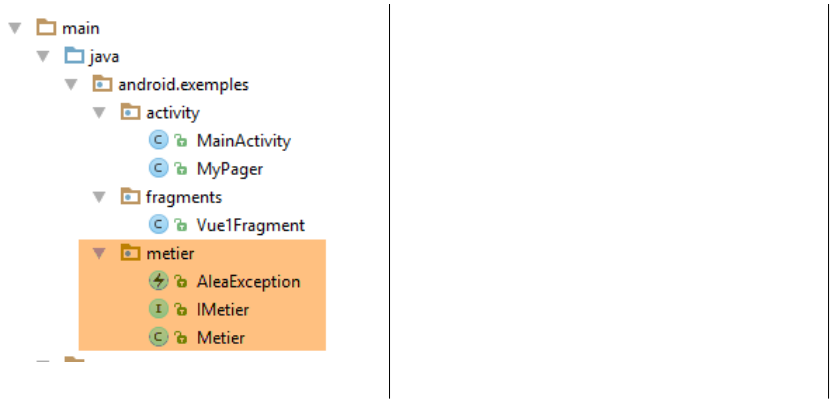
- les lignes 11-12 sont supprimées ainsi que les getter / setter associés.

A ce stade, l'application doit pouvoir être exécutée :



1.10.5 La couche [métier]





La couche [métier] présente l'interface [IMetier] suivante :

```
1.  package istia.st.android.metier;
2.
3.  import java.util.List;
4.
5.  public interface IMetier {
6.
7.     public List<Object> getAleas(int a, int b, int n);
8.  }
```

La méthode [getAleas(a,b,n)] renvoie normalement n nombres entiers aléatoires dans l'intervalle [a,b]. On a prévu également qu'elle renvoie une fois sur trois une exception, exception également insérée dans les réponses rendues par la méthode. Au final celle-ci rend une liste d'objets de type [Exception] ou [Integer].

L'implémentation [Metier] de cette interface est la suivante :

```
1.  package istia.st.android.metier;
2.
3.  import java.util.ArrayList;
```

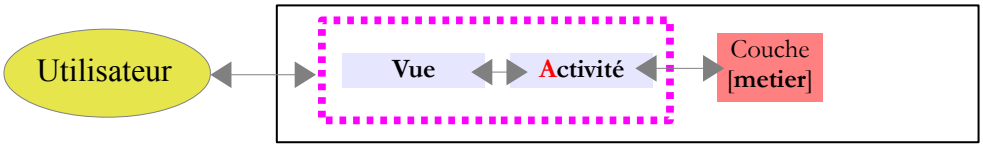
```
4.  import java.util.List;
5.  import java.util.Random;
6.
7.  @EBean(scope = EBean.Scope.Singleton)
8.  public class Metier implements IMetier {
9.
10.    @Override
11.    public List<Object> getAleas(int a, int b, int n) {
12.       // la liste des objets
13.       List<Object> réponses = new ArrayList<Object>();
14.       // qqs vérifications
15.       if (n < 1) {
16.          réponses.add(new AleaException("Le nombre d'entier aléatoires demandé doit être supérieur ou égal
    à 1"));
17.       }
18.       if (a < 0) {
19.          réponses.add(new AleaException("Le nombre a de l'intervalle [a,b] doit être supérieur à 0"));
20.       }
21.       if (b < 0) {
22.          réponses.add(new AleaException("Le nombre b de l'intervalle [a,b] doit être supérieur à 0"));
23.       }
24.       if (a >= b) {
25.          réponses.add(new AleaException("Dans l'intervalle [a,b], on doit avoir a< b"));
26.       }
27.       // erreur ?
28.       if (réponses.size() != 0) {
29.          return réponses;
30.       }
31.       // on génère les nombres aléatoires
32.       Random random = new Random();
33.       for (int i = 0; i < n; i++) {
34.          // on génère une exception aléatoire 1 fois / 3
35.          int nombre = random.nextInt(3);
36.          if (nombre == 0) {
37.             réponses.add(new AleaException("Exception aléatoire"));
38.          } else {
39.             // sinon on rend un nombre aléatoire entre deux bornes [a,b]
40.             réponses.add(Integer.valueOf(a + random.nextInt(b - a + 1)));
41.          }
42.       }
43.       // résultat
44.       return réponses;
45.    }
46. }
```

- ligne 7 : on utilise l'annotation AA [@EBean] sur la classe [Metier] afin de pouvoir injecter des références de celle-ci dans la couche [Présentation]. L'attribut (**scope = EBean.Scope.Singleton**) fait que la classe [Metier] ne sera instanciée qu'en un seul exemplaire. C'est donc toujours la même référence qui est injectée si on l'injecte plusieurs fois dans la couche [Présentation].

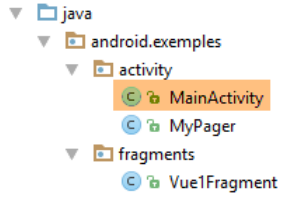
Le type [AleaException] utilisé par la classe [Metier] est la suivante :

```
1.  package istia.st.android.metier;
2.
3.  public class AleaException extends RuntimeException {
4.
5.     public AleaException() {
6.     }
7.
8.     public AleaException(String detailMessage) {
9.        super(detailMessage);
10.    }
11.
12.    public AleaException(Throwable throwable) {
13.       super(throwable);
14.    }
15.
16.    public AleaException(String detailMessage, Throwable throwable) {
17.       super(detailMessage, throwable);
18.    }
19.
20. }
```

1.10.6 L'activité [MainActivity] revisitée



Nous allons continuer à mettre dans l'activité les éléments qui doivent être partagés par les fragments / vues qu'elle gère. Aussi est-ce dans l'activité qu'on injectera la référence de la couche [métier] plutôt que dans les fragments. Par ailleurs l'activité implémentera l'interface [IMetier] de la couche [métier]. Ainsi un fragment n'aura-t-il que l'activité comme interlocuteur. Nous détaillons ce concept maintenant.

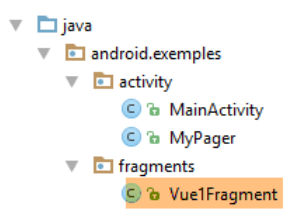


La classe [MainActivity] évolue de la façon suivante :

```
1.  @EActivity(R.layout.main)
2.  public class MainActivity extends FragmentActivity implements IMetier {
3.
4.    // la couche [metier]
5.    @Bean(Metier.class)
6.    protected IMetier metier;
7.
8.    // le gestionnaire de fragments ou sections
9.    SectionsPagerAdapter mSectionsPagerAdapter;
10.
11.   // le conteneur des fragments
12.   @ViewById(R.id.pager)
13.   MyPager mViewPager;
14.
15.   @Override
16.   protected void onCreate(Bundle savedInstanceState) {
17.     super.onCreate(savedInstanceState);
18.   }
19.
20.   @AfterViews
21.   protected void initActivity() {
22. ...
23.
24.   }
25.
26.   // navigation
27.   public void navigateToView(int i) {
28.     // on affiche le fragment n° i
29.     mViewPager.setCurrentItem(i);
30.   }
31.
32.   @Override
33.   public List<Object> getAleas(int a, int b, int n) {
34.     return metier.getAleas(a, b, n);
35.   }
36.
37.   // notre gestionnaire de fragments à redéfinir pour chaque application
38.   // doit définir les méthodes suivantes : getItem, getCount, getPageTitle
39.   public class SectionsPagerAdapter extends FragmentPagerAdapter {
40. ...
41.
42. }
```

- lignes 5-6 : la couche [métier] est injectée dans l'activité. On utilise pour cela l'annotation AA [@Bean] dont le paramètre est la classe portant l'annotation AA [@EBean] ;
- ligne 2 : l'activité implémente l'interface [IMetier] de la couche [métier] ;
- lignes 32-35 : implémentation de l'unique méthode de l'interface [IMetier]. On se contente de déléguer l'appel à la couche [métier] ;

1.10.7 Le fragment [Vue1Fragment] revisité



Le code de la classe [Vue1Fragment] est le suivant :

```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.exemples.activity.MainActivity;
5.  import android.support.v4.app.Fragment;
6.  import android.widget.ArrayAdapter;
7.  import android.widget.EditText;
8.  import android.widget.ListView;
9.  import android.widget.TextView;
10. import org.androidannotations.annotations.AfterViews;
11. import org.androidannotations.annotations.Click;
12. import org.androidannotations.annotations.EFragment;
13. import org.androidannotations.annotations.ViewById;
14.
15. import java.util.ArrayList;
16. import java.util.List;
17.
18. @EFragment(R.layout.vue1)
19. public class Vue1Fragment extends Fragment {
20.
21.   // les éléments de l'interface visuelle
22.   @ViewById(R.id.lst_reponses)
23.   ListView listReponses;
24.   @ViewById(R.id.edt_nbaleas)
25.   EditText edtNbAleas;
26.   @ViewById(R.id.edt_a)
27.   EditText edtA;
28.   @ViewById(R.id.edt_b)
29.   EditText edtB;
30.   @ViewById(R.id.txt_errorNbAleas)
31.   TextView txtErrorAleas;
32.   @ViewById(R.id.txt_errorIntervalle)
33.   TextView txtErrorIntervalle;
34.
35.   // liste des reponses à une commande
36.   private List<String> reponses = new ArrayList<String>();
37.   // les saisies
38.   private int nbAleas;
39.   private int a;
40.   private int b;
41.   // l'activité
42.   private MainActivity activité;
43.
44.   @AfterViews
45.   void initFragment() {
46.     // on note l'activité
47.     activité = (MainActivity) getActivity();
48.     // au départ pas de messages d'erreur
49.     txtErrorAleas.setText("");
50.     txtErrorIntervalle.setText("");
```

```
51.   }
52.
53.   @Click(R.id.btn_Executer)
54.   void doExecuter() {
55.    ...
56.   }
57.
58.   // on vérifie la validité des données saisies
59.   private boolean isPageValid() {
60.   ...
61. }
```

- lignes 22-23 : on récupère toutes les références des composants de la vue [vue1.xml] ;
- ligne 45 : à cause de l'annotation AA [@AfterViews], la méthode [initFragment] s'exécute une fois que toutes les références précédentes ont été initialisées ;
- ligne 47 : on récupère l'activité. On se rappelle que celle-ci sera le seul interlocuteur du fragment ;
- lignes 49-50 : on efface les messages d'erreur ;

La méthode [doExecuter] traite le 'click' sur le bouton [Exécuter]. Son code est le suivant :

```
1.  @Click(R.id.btn_Executer)
2.    void doExecuter() {
3.      // on efface les éventuels msg d'erreur précédents
4.      txtErrorAleas.setText("");
5.      txtErrorIntervalle.setText("");
6.      // on teste la validité des saisies
7.      if (!isPageValid()) {
8.        return;
9.      }
10.     // on efface les reponses précédentes
11.     reponses.clear();
12.     // on demande les nombres aléatoires à l'activité
13.     List<Object> data = activité.getAleas(a, b, nbAleas);
14.     // on crée une liste de String à partir de ces données
15.     List<String> strings = new ArrayList<String>();
16.     for (Object o : data) {
17.       if (o instanceof Exception) {
18.         strings.add(((Exception) o).getMessage());
19.       } else {
20.         strings.add(o.toString());
21.       }
22.     }
23.     // on affiche les reponses
24.     listReponses.setAdapter(new ArrayAdapter<String>(activité, android.R.layout.simple_list_item_1,
    android.R.id.text1, strings));
25.   }
```

- lignes 7-9 : avant d'exécuter l'action demandée, on vérifie que les valeurs saisies sont correctes ;
- ligne 13 : la liste des nombres aléatoires est demandée à l'activité. On obtient une liste d'objets où chaque objet est de type [Integer] ou [AleaException] ;
- lignes 15-22 : à partir de la liste d'objets obtenue, on crée la liste de [String] qui va être affichée par le composant de type [ListView] de la vue ;

```
1.   @ViewById(R.id.lst_reponses)
2. ListView listReponses;
```

- ligne 24 : le composant [listRéponses] de type [ListView] va afficher la liste de [String] que nous venons de construire.

La signature de [ListView.setAdapter] est la suivante :

```
public void setAdapter (ListAdapter adapter)
```

[ListAdapter] est une interface. La classe [ArrayAdapter] est une classe implémentant cette interface. Le constructeur utilisé ici est le suivant :

```
public ArrayAdapter (Context context, int resource, int textViewResourceId, List<T> objects)
```

- [context] est l'activité qui affiche le [ListView] ;

- [resource] est l'entier identifiant la vue utilisée pour afficher un élément du [ListView]. Cette vue peut avoir une complexité quelconque. C'est le développeur qui la construit en fonction de ses besoins ;
- [textViewResourceId] est l'entier identifiant un composant [TextView] dans la vue [resource]. La chaîne affichée le sera par ce composant ;
- [objects] : la liste d'objets affichés par le [ListView]. La méthode [toString] des objets est utilisée pour afficher l'objet dans le [TextView] identifié par [textViewResourceId] dans la vue identifiée par [resource].

Le travail du développeur est de créer la vue [resource] qui va afficher chaque élément du [ListView]. Pour le cas simple où on ne désire afficher qu'une simple chaîne de caractères comme ici, Android fournit la vue identifiée par [android.R.layout.*simple_list_item_1*]. Celle-ci contient un composant [TextView] identifié par [android.R.id.*text1*]. C'est la méthode utillisée ligne 24 pour afficher la liste [strings].

La validité des valeurs saisies est vérifiée par la méthode [isPageValid] suivante :

```
1.  // on vérifie la validité des données saisies
2.      private boolean isPageValid() {
3.          // saisie du nombre de nombres aléatoires
4.          nbAleas = 0;
5.          Boolean erreur = false;
6.          int nbErreurs = 0;
7.          try {
8.              nbAleas = Integer.parseInt(edtNbAleas.getText().toString());
9.              erreur = (nbAleas < 1);
10.         } catch (Exception ex) {
11.             erreur = true;
12.         }
13.         // erreur ?
14.         if (erreur) {
15.             nbErreurs++;
16.             txtErrorAleas.setText(R.string.txt_errorNbAleas);
17.         }
18.         // saisie de a
19.         a = 0;
20.         erreur = false;
21.         try {
22.             a = Integer.parseInt(edtA.getText().toString());
23.         } catch (Exception ex) {
24.             erreur = true;
25.         }
26.         // erreur ?
27.         if (erreur) {
28.             nbErreurs++;
29.             txtErrorIntervalle.setText(R.string.txt_errorIntervalle);
30.         }
31.         // saisie de b
32.         b = 0;
33.         erreur = false;
34.         try {
35.             b = Integer.parseInt(edtB.getText().toString());
36.             erreur = b < a;
37.         } catch (Exception ex) {
38.             erreur = true;
39.         }
40.         // erreur ?
41.         if (erreur) {
42.             nbErreurs++;
43.             txtErrorIntervalle.setText(R.string.txt_errorIntervalle);
44.         }
45.         // retour
46.         return (nbErreurs == 0);
47.     }
```

Ci-dessus on a vérifié simplement que les valeurs saisies étaient des nombres entiers. La couche [métier] est plus exigeante. Si vous entrez un nombre **a** supérieur au nombre **b**, la couche [métier] vous renverra une exception.

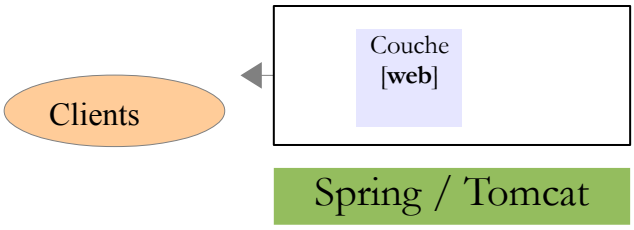
1.10.8 Exécution

Exécutez le projet [exemple-09] et vérifiez son bon fonctionnement.

1.11 Exemple-10 : architecture client / serveur

Nous abordons une architecture courante pour une application Android, celle où l'application Android communique avec des services web distants. On aura maintenant l'architecture suivante :

On a ajouté à l'application Android une couche [DAO] pour communiquer avec le serveur distant. Elle communiquera avec le serveur qui génère les nombres aléatoires affichés par la tablette Android. Ce serveur aura une architecture à deux couches suivante :



Les clients interrogent certaines URL de la couche [web / jSON] et reçoivent une réponse texte au format jSON (**J**ava**S**cript **O**bject **N**otation). Ici notre service web traitera une unique URL de type [/a/b] qui renverra un nombre aléatoire dans l'intervalle [a,b]. Nous allons décrire l'application dans l'ordre suivant :

Le serveur

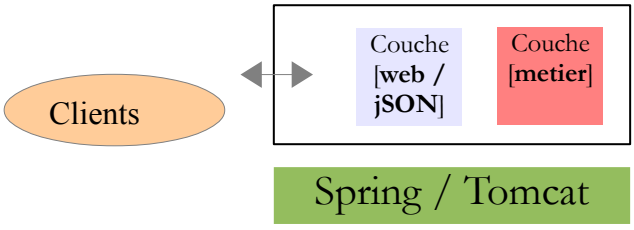
- sa couche [métier] ;
- son service [web / jSON] implémenté avec Spring MVC ;

Le client

- sa couche [DAO]. Il n'y aura pas de couche [métier] ;

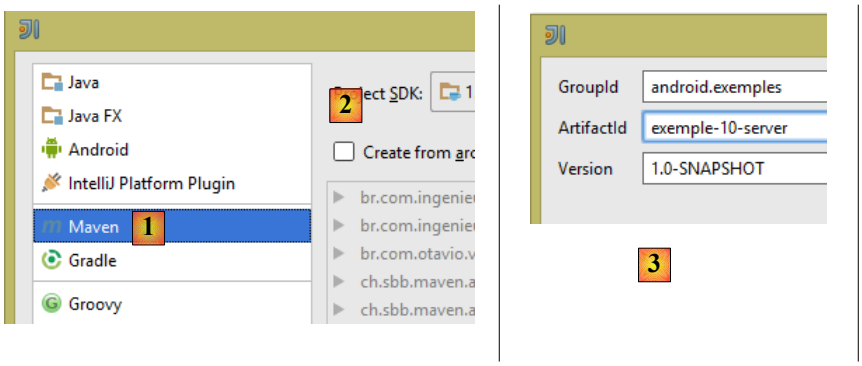
1.11.1 Le serveur [web / jSON]

Nous voulons construire l'architecture suivante :

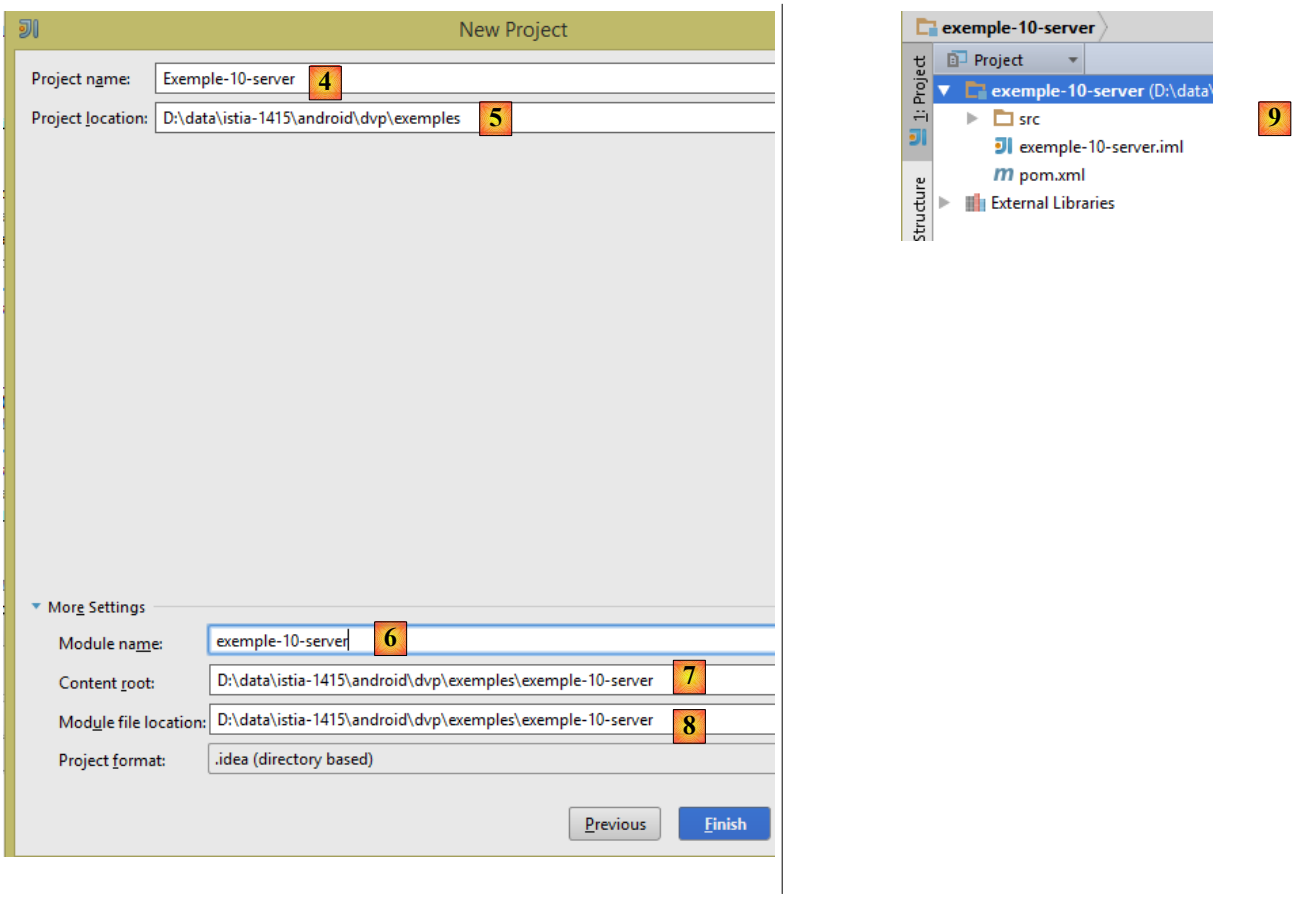


1.11.1.1 Création du projet

Nous créons un nouveau projet Maven [exemple-10-server] :



- en [1], on crée un projet Maven sans archétype [2] ;
- en [3], le projet Maven a les propriétés suivantes :
  - [GroupId] : [android.exemples] ;
  - [ArtifactId] : [exemple-10-server] ;
  - [Version] : [1.0-SNAPSHOT] ;



- en [4], le nom du projet Intellij ;
- en [5], le dossier parent du dossier qui va être créé pour le nouveau projet ;
- en [6], le nom du module Intellij ;
- laisser les valeurs par défaut qui s'inscrivent en [7-8] ;
- en [9], le projet généré ;

Le fichier [pom.xml] qui configure le projet Maven est pour l'instant le suivant :

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <project xmlns="http://maven.apache.org/POM/4.0.0"
3.           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
4.          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
    4.0.0.xsd">
5.    <modelVersion>4.0.0</modelVersion>
6.
7.    <groupId>android.exemples</groupId>
8.    <artifactId>exemple-10-server</artifactId>
9.    <version>1.0-SNAPSHOT</version>
10. </project>
```

- les lignes 7-9 reprennent les données saisies dans l'assistant de création ;

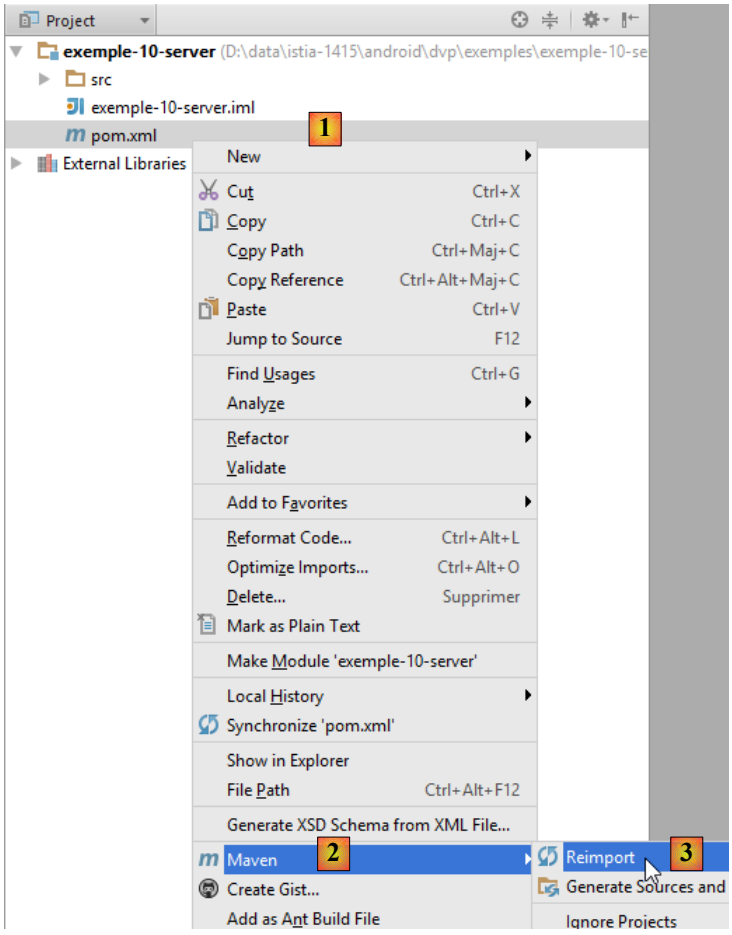
Nous modifions ce fichier de la façon suivante :

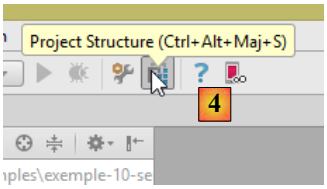
```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <project xmlns="http://maven.apache.org/POM/4.0.0"
3.           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
    4.0.0.xsd">
5.    <modelVersion>4.0.0</modelVersion>
6.
7.    <groupId>android.exemples</groupId>
8.    <artifactId>exemple-10-server</artifactId>
9.    <version>1.0-SNAPSHOT</version>
10.
11.   <properties>
12.     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
13.     <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
14.     <start-class>istia.st.aleas.config.Application</start-class>
15.   </properties>
16.
17.   <parent>
18.     <groupId>org.springframework.boot</groupId>
19.     <artifactId>spring-boot-starter-parent</artifactId>
20.     <version>1.1.1.RELEASE</version>
21.   </parent>
22.
23.   <dependencies>
24.     <dependency>
25.       <groupId>org.springframework.boot</groupId>
26.       <artifactId>spring-boot-starter-web</artifactId>
27.     </dependency>
28.   </dependencies>
29.
30.   <build>
31.     <plugins>
32.       <plugin>
33.         <artifactId>maven-compiler-plugin</artifactId>
34.       </plugin>
35.       <plugin>
36.         <groupId>org.springframework.boot</groupId>
37.         <artifactId>spring-boot-maven-plugin</artifactId>
38.       </plugin>
39.     </plugins>
40.   </build>
41.
42.   <repositories>
43.     <repository>
44.       <id>spring-snapshots</id>
45.       <url>http://repo.spring.io/libs-snapshot</url>
46.       <snapshots>
47.         <enabled>true</enabled>
48.       </snapshots>
49.     </repository>
50.   </repositories>
51.   <pluginRepositories>
52.     <pluginRepository>
53.       <id>spring-snapshots</id>
54.       <url>http://repo.spring.io/libs-snapshot</url>
55.       <snapshots>
56.         <enabled>true</enabled>
57.       </snapshots>
58.     </pluginRepository>
```

```
59.   </pluginRepositories>
60.
61. </project>
```

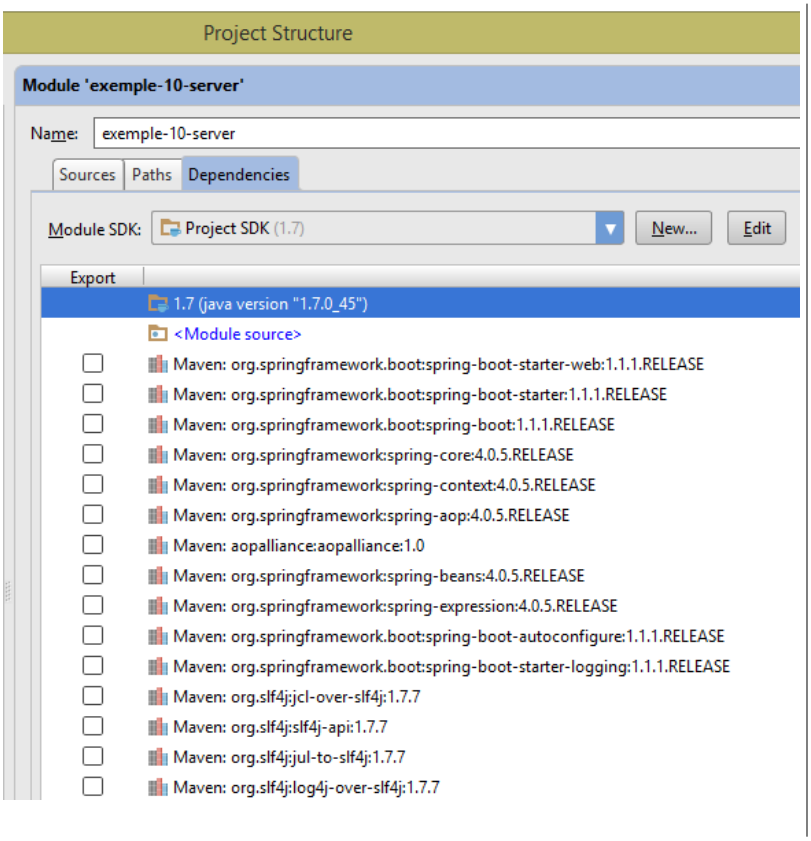
- lignes 17-21 : une dépendance sur le framework [Spring Boot], une branche de l'écosystème Spring. Ce framework [http://projects.spring.io/spring-boot/] permet une configuration minimale de Spring. Selon les archives présentes dans le Classpath du projet, [Spring Boot] infère une configuration plausible ou probable pour celui-ci. Ainsi si Hibernate est dans le Classpath du projet, alors [Spring Boot] infèrera que l'implémentation JPA utilisée va être Hibernate et configurera Spring dans ce sens. Le développeur n'a plus à le faire. Il ne lui reste alors à faire que les configurations que [Spring Boot] n'a pas faites par défaut ou celles que [Spring Boot] a faites par défaut mais qui doivent être précisées. Dans tous les cas c'est la configuration faite par le développeur qui a le dernier mot ;
- lignes 17-21 : définissent un projet Maven parent. Ce dernier, [Spring Boot], définit les dépendances les plus courantes pour les divers types de projets Spring qui peuvent exister. Lorsqu'on utilise l'une d'elles dans le fichier [pom.xml], sa version n'a pas à être précisée. C'est la version définie dans le projet parent qui sera utilisée ;
- lignes 24-27 : définissent une dépendance sur l'artifact [spring-boot-starter-web]. Cet artifact amène avec lui toutes les archives nécessaires à un projet Spring MVC. Parmi celles-ci on trouve l'archive d'un serveur Tomcat. C'est lui qui sera utilisé pour déployer l'application web. On notera que la version de la dépendance n'a pas été mentionnée. C'est celle mentionnée dans le projet parent qui sera utilisée ;

Pour mettre à jour le projet, il faut forcer le téléchargement des dépendances :





Regardons [4] les dépendances amenées par ce fichier [pom.xml] :

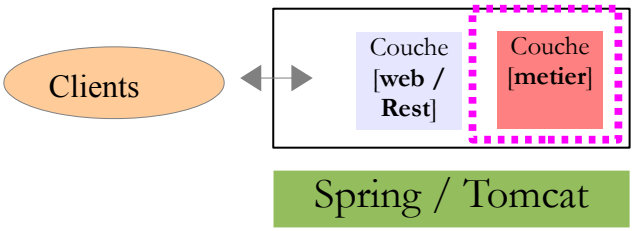


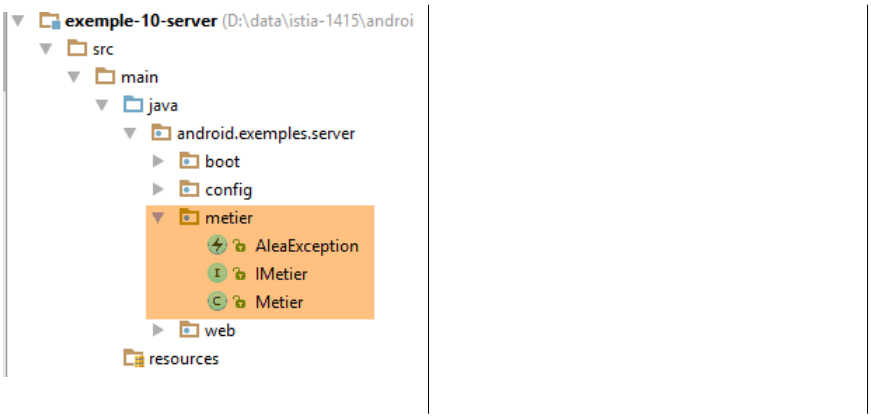
Elles sont très nombreuses. Spring Boot pour le web a inclus les dépendances dont une application web Spring MVC aura probablement besoin. Cela veut dire que certaines sont peut être inutiles. Spring Boot est idéal pour un tutoriel :

- il amène les dépendances dont nous aurons probablement besoin ;
- nous allons voir qu'il simplifie considérablement la configuration du projet Spring MVC ;
- il amène un serveur Tomcat embarqué [1] ce qui nous évite le déploiement de l'application sur un serveur web externe ;
- il permet de générer un jar exécutable incluant toutes les dépendances ci-dessus. Ce jar peut être transporté d'une plate-forme à une autre sans reconfiguration.

On trouvera de nombreux exemples utilisant Spring Boot sur le site de l'écosystème Spring [http://spring.io/guides]. Maintenant que notre projet est configuré, nous pouvons passer au code.

1.11.1.2 La couche [métier]





La couche [métier] aura l'interface [IMetier] suivante :

```
1.  package android.exemples.server.metier;
2.
3.  public interface IMetier {
4.    // nombre aléatoire dans [a,b]
5.     public int getAlea(int a, int b);
6.  }
```

- ligne 5 : la méthode qui génère 1 nombre aléatoire dans [a,b]

Le code de la classe [Metier] implémentant cette interface est le suivant :

```
1.  package android.exemples.server.metier;
2.
3.  import org.springframework.stereotype.Service;
4.
5.  import java.util.Random;
6.
7.  @Service
8.  public class Metier implements IMetier {
9.
10.   @Override
11.   public int getAlea(int a, int b) {
12.     // qqs vérifications
13.     if (a < 0) {
14.       throw new AleaException("Le nombre a de l'intervalle [a,b] doit être supérieur à 0", 2);
15.     }
16.     if (b < 0) {
17.       throw new AleaException("Le nombre b de l'intervalle [a,b] doit être supérieur à 0", 3);
18.     }
19.     if (a >= b) {
20.       throw new AleaException("Dans l'intervalle [a,b], on doit avoir a< b", 4);
21.     }
22.     // génération résultat
23.     Random random = new Random();
24.     int nombre = random.nextInt(3);
25.     if (nombre == 0) {
26.       // on génère une exception aléatoire 1 fois / 3
27.       throw new AleaException("Exception aléatoire", 5);
28.     } else {
29.       // sinon on rend un nombre aléatoire entre les deux bornes [a,b]
30.       return a + random.nextInt(b - a + 1);
31.     }
32.   }
33. }
```

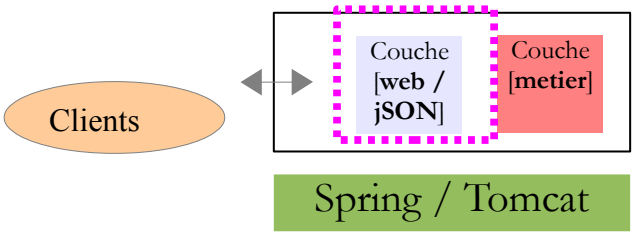
Nous ne commentons pas la classe : elle est analogue à celle rencontrée dans l'exemple précédent. On notera simplement ligne 7 l'annotation Spring [@Service] qui va faire que Spring va instancier la classe en un unique exemplaire et rendre sa référence disponible pour d'autres composants Spring. D'autres annotations Spring auraient pu être utilisées ici pour le même effet.

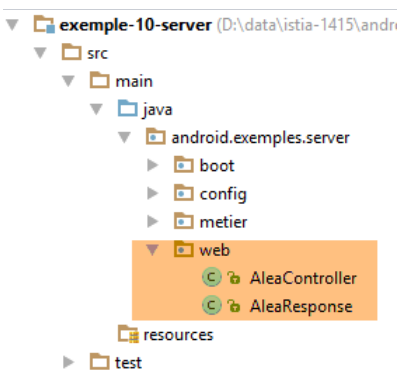
La classe [Metier] lance des exceptions de type [AleaException] :

```
1.  package android.exemples.server.metier;
2.
3.  public class AleaException extends RuntimeException {
4.
5.    // code d'erreur
6.    private int code;
7.
8.    // constructeurs
9.    public AleaException() {
10.   }
11.
12.   public AleaException(String detailMessage, int code) {
13.     super(detailMessage);
14.     this.code = code;
15.   }
16.
17.   public AleaException(Throwable throwable, int code) {
18.     super(throwable);
19.     this.code = code;
20.   }
21.
22.   public AleaException(String detailMessage, Throwable throwable, int code) {
23.     super(detailMessage, throwable);
24.     this.code = code;
25.   }
26.
27.   // getters et setters
28.
29.   public int getCode() {
30.     return code;
31.   }
32.
33.   public void setCode(int code) {
34.     this.code = code;
35.   }
36. }
```

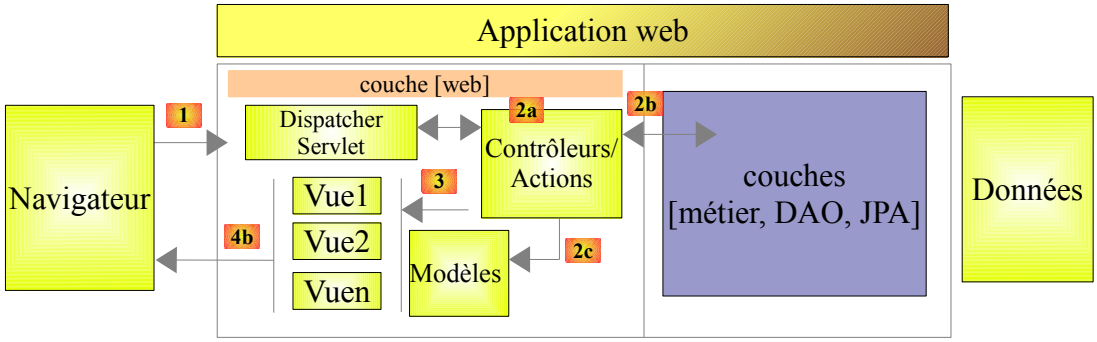
- ligne 3 : [AleaException] étend la classe [RuntimeException]. C'est donc une exception non contrôlée (pas d'obligation de la gérer avec un try / catch) ;
- ligne 6 : on ajoute à la classe [RuntimeException] un code d'erreur ;

1.11.1.3 Le service web / jSON





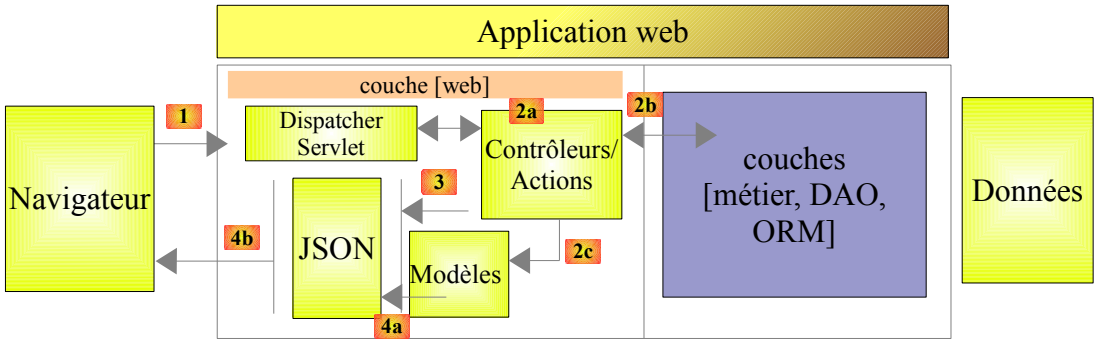
Le service web / jSON est implémenté par Spring MVC. Spring MVC implémente le modèle d'architecture dit MVC (Modèle – Vue – Contrôleur) de la façon suivante :



Le traitement d'une demande d'un client se déroule de la façon suivante :

1. **demande** - les URL demandées sont de la forme *http://machine:port/contexte/Action/param1/param2/....?p1=v1&p2=v2&...* La [Dispatcher Servlet] est la classe de Spring qui traite les URL entrantes. Elle "route" l'URL vers l'action qui doit la traiter. Ces actions sont des méthodes de classes particulières appelées [Contrôleurs]. Le **C** de MVC est ici la chaîne [Dispatcher Servlet, Contrôleur, Action]. Si aucune action n'a été configurée pour traiter l'URL entrante, la servlet [Dispatcher Servlet] répondra que l'URL demandée n'a pas été trouvée (erreur 404 NOT FOUND) ;
2. **traitement**
    - l'action choisie peut exploiter les paramètres *parami* que la servlet [Dispatcher Servlet] lui a transmis. Ceux-ci peuvent provenir de plusieurs sources :
        - du chemin [*/param1/param2/...*] de l'URL,
        - des paramètres [*p1=v1&p2=v2*] de l'URL,
        - de paramètres postés par le navigateur avec sa demande ;
    - dans le traitement de la demande de l'utilisateur, l'action peut avoir besoin de la couche [metier] [2b]. Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est :
        - une page d'erreur si la demande n'a pu être traitée correctement
        - une page de confirmation sinon
    - l'action demande à une certaine vue de s'afficher [3]. Cette vue va afficher des données qu'on appelle le **modèle de la vue**. C'est le **M** de MVC. L'action va créer ce modèle M [2c] et demander à une vue **V** de s'afficher [3] ;
3. **réponse** - la vue **V** choisie utilise le modèle **M** construit par l'action pour initialiser les parties dynamiques de la réponse HTML qu'elle doit envoyer au client puis envoie cette réponse.

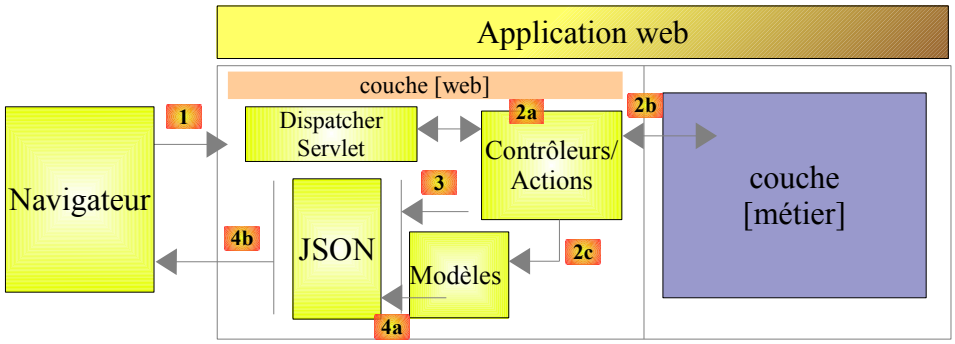
Pour un service web / jSON, l'architecture précédente est légèrement modifiée :



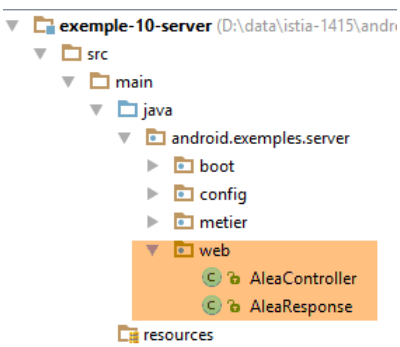
- en [4a], le modèle qui est une classe Java est transformé en chaîne jSON par une bibliothèque jSON ;
- en [4b], cette chaîne jSON est envoyée au navigateur ;

Un exemple de sérialisation d'un objet Java en chaîne jSON et de désérialisation d'une chaîne jSON en objet Java est présenté en annexes au paragraphe 1.16.9, page 199.

Revenons à la couche [web] de notre application :



Dans notre application, il n'y a qu'un contrôleur :



Le service web / jSON enverra à ses clients une réponse de type [AleaResponse] suivant :

```
1.  package android.exemples.server.web;
2.
3.  import java.util.List;
4.
5.  public class AleaResponse {
6.
7.    // data
8.    private int erreur;
9.    private List<String> messages;
10.   private int alea;
11.
```

```
12.   // getters et setters
13. ...
14. }
```

- ligne 8 : un code d'erreur (0 si pas d'erreur) ;
- ligne 9 : si *erreur!=0*, une liste de messages d'erreur, usuellement ceux de la pile d'exceptions si exception il y a eu, *null* si pas d'erreur ;
- ligne 10 : le nombre aléatoire généré ;

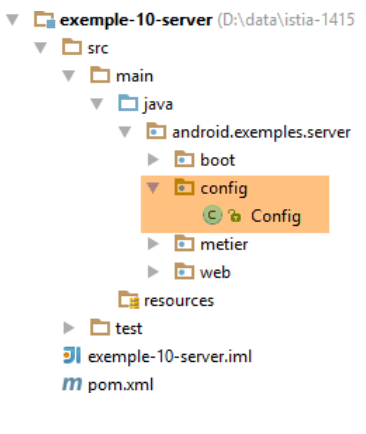
Le contrôleur [AleaController] est le suivant :

```
1.  package android.exemples.server.web;
2.
3.  import android.exemples.server.metier.AleaException;
4.  import android.exemples.server.metier.IMetier;
5.  import org.springframework.beans.factory.annotation.Autowired;
6.  import org.springframework.web.bind.annotation.PathVariable;
7.  import org.springframework.web.bind.annotation.RequestMapping;
8.  import org.springframework.web.bind.annotation.RequestMethod;
9.  import org.springframework.web.bind.annotation.RestController;
10.
11. import java.util.ArrayList;
12. import java.util.List;
13.
14. @RestController
15. public class AleaController {
16.
17.   // couche métier
18.   @Autowired
19.   private IMetier metier;
20.
21.   // nombres aléatoires
22.   @RequestMapping(value = "/{a}/{b}", method = RequestMethod.GET)
23.   public AleaResponse getAlea(@PathVariable("a") int a, @PathVariable("b") int b) {
24.
25.     // la réponse
26.     AleaResponse response = new AleaResponse();
27.     // on utilise la couche métier
28.     try {
29.       response.setAlea(metier.getAlea(a, b));
30.       response.setErreur(0);
31.     } catch (AleaException e) {
32.       response.setErreur(e.getCode());
33.       response.setMessages(getMessagesFromException(e));
34.     }
35.     // on rend la réponse
36.     return response;
37.   }
38.
39.   private List<String> getMessagesFromException(AleaException e) {
40.     // liste des messages
41.     List<String> messages = new ArrayList<String>();
42.     // on parcourt la pile des exceptions
43.     Throwable th = e;
44.     while (th != null) {
45.       messages.add(e.getMessage());
46.       th = th.getCause();
47.     }
48.     // on rend le résultat
49.     return messages;
50.   }
51.
52. }
```

- ligne 14 : l'annotation [@RestController] implique diverses choses :
    - que la classe contient des méthodes qui vont traiter des requêtes pour certaines URL de l'application web,
    - que les méthodes de la classe génèrent elles-mêmes la réponse envoyée au client,
    - que cette réponse sera une chaîne de caractères au format jSON (JavaScript Object Notation) ;

- ligne 18 : l'annotation [@Autowired] demande à Spring d'injecter dans le champ, un composant de type [IMetier]. Ce sera la classe [Metier] précédente. C'est parce que nous avons mis à celle-ci l'annotation [@Service] qu'elle est gérée comme un composant Spring ;
- ligne 23 : la méthode qui génère le nombre aléatoire. Son nom n'a pas d'importance. Lorsqu'elle s'exécute, les champs de la ligne 22 ont été initialisés par Spring MVC. Nous verrons comment. Par ailleurs, si elle s'exécute, c'est parce que le serveur web a reçu une requête HTTP GET pour l'URL de la ligne 22 ;
- ligne 22 : l'URL traitée est de la forme **/{a}/{b}** où {x} représente une variable. Les variables {a} et {b} sont affectées aux paramètres de la méthode ligne 16. Cela se fait via l'annotation **@PathVariable(" x ")**. On notera que {a} et {b}sont des composantes d'une URL et sont donc de type *String*. La conversion de *String* vers le type des paramètres peut échouer. Spring MVC lance alors une exception. **Résumons** : si avec un navigateur je demande l'URL /100/200, la méthode *getAlea* de la ligne 16 s'exécutera avec les paramètres entiers a=100, b=200 ;
- ligne 23 : on notera que le type de la réponse est **AleaResponse**. Parce que la classe a été taguée avec [@RestController], cette réponse sera transformée en jSON avant d'être envoyée au client ;
- ligne 29 : on demande à la couche [métier] un nombre aléatoire dans l'intervalle [a,b]. On se souvient que la méthode *[metier].getAlea* peut lancer une exception ;
- ligne 30 : pas d'erreur ;
- ligne 32 : code d'erreur ;
- ligne 33 : la liste des messages de la réponse est celle de la pile d'exceptions (lignes 39-50). Ici, nous savons que la pile ne contient qu'une exception mais nous avons voulu montrer une méthode plus générique ;
- ligne 36 : la réponse de type [AleaResponse] est rendue. Elle va être automatiquement sérialisée en jSON puis envoyée au client ;

1.11.1.4 *Configuration du projet Spring*



Il existe diverses façons de configurer Spring :
- avec des fichiers XML ;
- avec du code Java ;
- avec un mix des deux ;

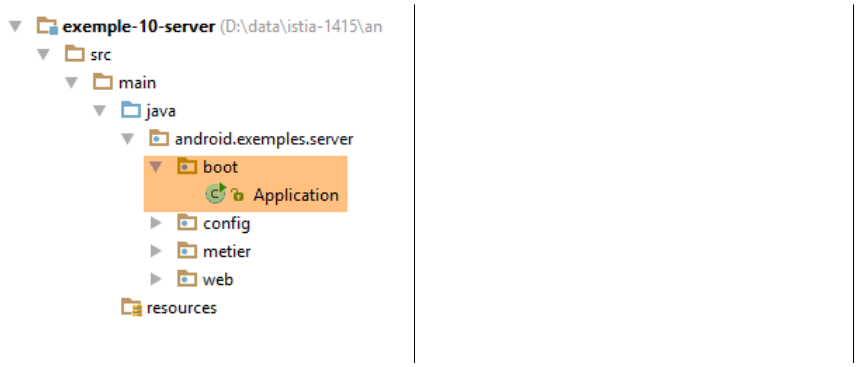
Nous choisissons de configurer notre application web avec du code Java. C'est la classe [Config] suivante qui assure cette configuration :

```
1.  package android.exemples.server.config;
2.
3.  import org.springframework.boot.SpringApplication;
4.  import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
5.  import org.springframework.context.annotation.ComponentScan;
6.
7.  @ComponentScan(basePackages = { "android.exemples.server.metier", "android.exemples.server.web" })
8.  @EnableAutoConfiguration
9.  public class Config {
10. }
```

- ligne 7 : on dit à Spring dans quels packages il va trouver les deux composants qu'il doit gérer :
- le composant [Metier] annoté [@Service] dans le package [android.exemples.server.metier],
- le composant [AleaController] annoté [@RestController] dans le package [android.exemples.server.web] ;
- ligne 8 : on demande à Spring Boot de configurer l'application Spring MVC. C'est là qu'intervient la magie de Spring Boot. Sur la base des packages présents dans le ClassPath du projet, il va faire des hypothèses et configurer Spring MVC. Comme

tout l'écosystème Spring, Spring MVC est extrêmement flexible et cela se paie par de la configuration. Ici, Spring Boot utilise la notion de " convention over configuration " qui dit que si on ne s'écarte pas de certaines conventions, la configuration devrait être minimale.

1.11.1.5 *Exécution du serveur web*

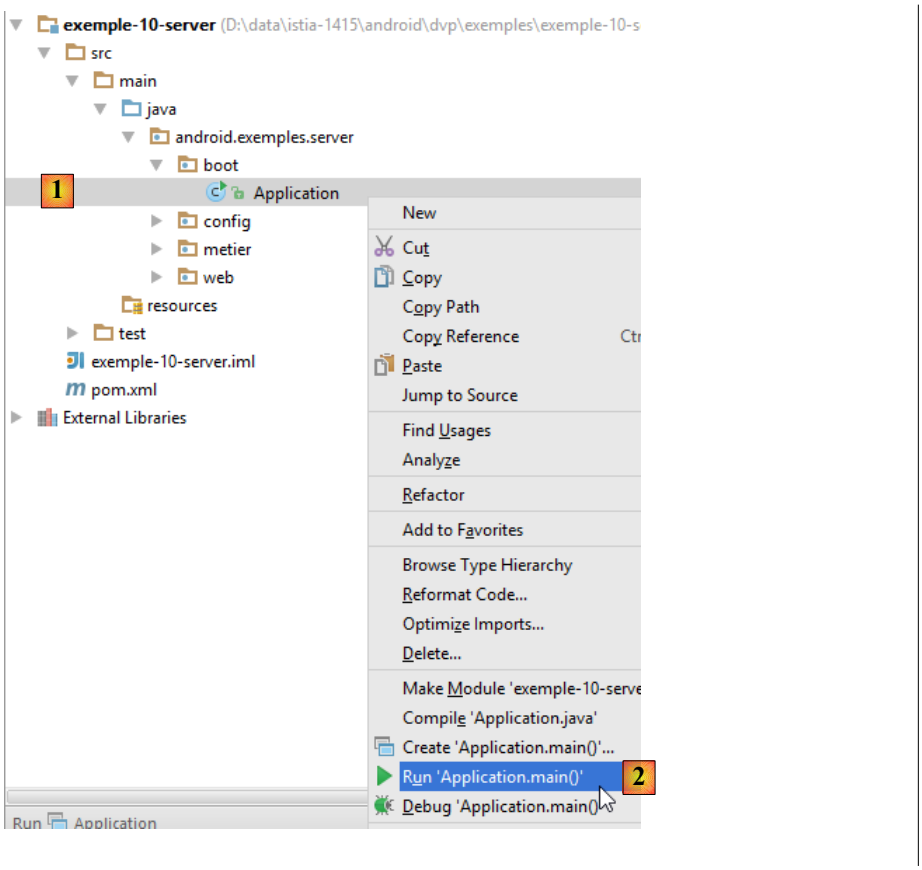


Le projet s'exécute à partir de la classe exécutable [Application] suivante :

```
1.  package android.exemples.server.boot;
2.
3.  import android.exemples.server.config.Config;
4.  import org.springframework.boot.SpringApplication;
5.
6.  public class Application {
7.    public static void main(String[] args) {
8.      // exécution application
9.      SpringApplication.run(Config.class, args);
10.   }
11.
12. }
```

- la classe [Application] est une classe exécutable (lignes 7-10) ;
- ligne 9 : la méthode statique [SpringApplication.run] est une méthode de [spring Boot] (ligne 4) qui va lancer l'application. Son premier paramètre est la classe Java qui configure le projet. Ici la classe [Config] que nous venons de décrire. Le second paramètre est le tableau d'arguments passé à la méthode [main] (ligne 7) ;

On peut lancer l'application web de diverses façons dont la suivante :



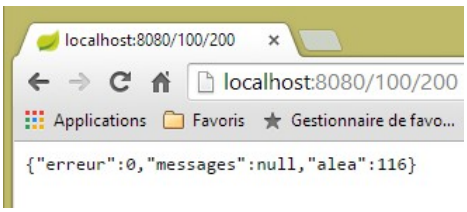
Dans la console, apparaissent alors un certain nombre de logs :

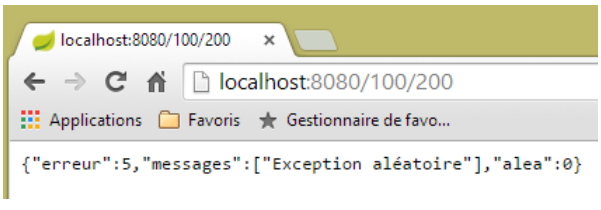
```
.   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.1.1.RELEASE)

2014-10-07 09:13:42.194  INFO 7408 --- [           main] a.exemples.server.boot.Application       :
Starting Application on Gportpers3 with PID 7408 (D:\data\istia-1415\android\dvp\exemples\exemple-10-
server\target\classes started by ST in D:\data\istia-1415\android\dvp\exemples)
2014-10-07 09:13:42.294  INFO 7408 --- [           main] ationConfigEmbeddedWebApplicationContext :
Refreshing
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@23edbbb9:
startup date [Tue Oct 07 09:13:42 CEST 2014]; root of context hierarchy
2014-10-07 09:13:42.996  INFO 7408 --- [           main] o.s.b.f.s.DefaultListableBeanFactory     :
Overriding bean definition for bean 'beanNameViewResolver': replacing [Root bean: class [null]; scope=;
abstract=false; lazyInit=false; autowireMode=3; dependencyCheck=0; autowireCandidate=true;
primary=false;
factoryBeanName=org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$WhitelabelErrorView
Configuration; factoryMethodName=beanNameViewResolver; initMethodName=null;
destroyMethodName=(inferred); defined in class path resource
[org/springframework/boot/autoconfigure/web/ErrorMvcAutoConfiguration$WhitelabelErrorViewConfiguration.c
lass]] with [Root bean: class [null]; scope=; abstract=false; lazyInit=false; autowireMode=3;
dependencyCheck=0; autowireCandidate=true; primary=false;
factoryBeanName=org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration$WebMvcAutoConfigurati
onAdapter; factoryMethodName=beanNameViewResolver; initMethodName=null; destroyMethodName=(inferred);
defined in class path resource
[org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration$WebMvcAutoConfigurationAdapter.class
]]
2014-10-07 09:13:44.251  INFO 7408 --- [           main] .t.TomcatEmbeddedServletContainerFactory :
Server initialized with port: 8080
2014-10-07 09:13:44.665  INFO 7408 --- [           main] o.apache.catalina.core.StandardService   :
Starting service Tomcat
2014-10-07 09:13:44.666  INFO 7408 --- [           main] org.apache.catalina.core.StandardEngine  :
Starting Servlet Engine: Apache Tomcat/7.0.54
```

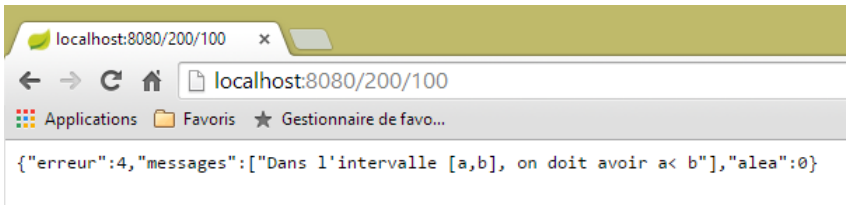
```
15. 2014-10-07 09:13:44.866  INFO 7408 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       :
    Initializing Spring embedded WebApplicationContext
16. 2014-10-07 09:13:44.866  INFO 7408 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root
    WebApplicationContext: initialization completed in 2575 ms
17. 2014-10-07 09:13:45.748  INFO 7408 --- [ost-startStop-1] o.s.b.c.e.ServletRegistrationBean        :
    Mapping servlet: 'dispatcherServlet' to [/]
18. 2014-10-07 09:13:45.750  INFO 7408 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean  :
    Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
19. 2014-10-07 09:13:46.802  INFO 7408 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  :
    Mapped URL path [/**/favicon.ico] onto handler of type [class
    org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
20. 2014-10-07 09:13:46.946  INFO 7408 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/{a}/{b}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    android.exemples.server.web.AleaResponse android.exemples.server.web.AleaController.getAlea(int,int)
21. 2014-10-07 09:13:46.950  INFO 7408 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/error],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>>
    org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequ
    est)
22. 2014-10-07 09:13:46.951  INFO 7408 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/error],methods=[],params=[],headers=[],consumes=[],produces=[text/html],custom=[]}" onto
    public org.springframework.web.servlet.ModelAndView
    org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServlet
    Request)
23. 2014-10-07 09:13:46.979  INFO 7408 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  :
    Mapped URL path [/**] onto handler of type [class
    org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
24. 2014-10-07 09:13:46.979  INFO 7408 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  :
    Mapped URL path [/webjars/**] onto handler of type [class
    org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
25. 2014-10-07 09:13:47.294  INFO 7408 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        :
    Registering beans for JMX exposure on startup
26. 2014-10-07 09:13:47.335  INFO 7408 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer :
    Tomcat started on port(s): 8080/http
27. 2014-10-07 09:13:47.337  INFO 7408 --- [           main] a.exemples.server.boot.Application       :
    Started Application in 6.081 seconds (JVM running for 6.897)
```

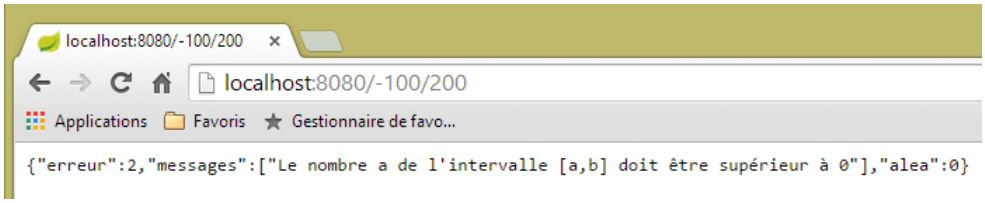
- lignes 12-14 : le serveur embarqué Tomcat est lancé ;
- lignes 15-19 : la servlet [DispatcherServlet] de Spring MVC est chargée et configurée ;
- ligne 20 : l'URL [/{a}/{b}] du serveur web est détectée ;

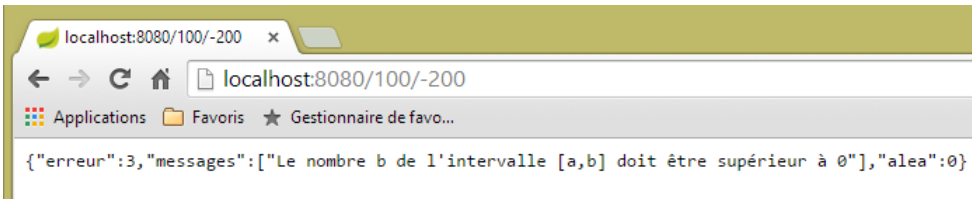
Maintenant, prenons un navigateur et testons l'URL du service REST :





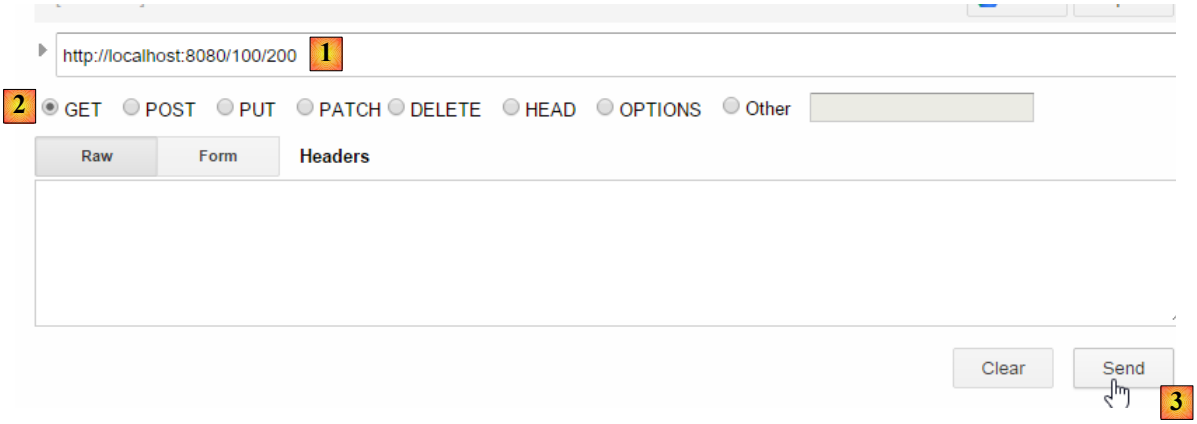




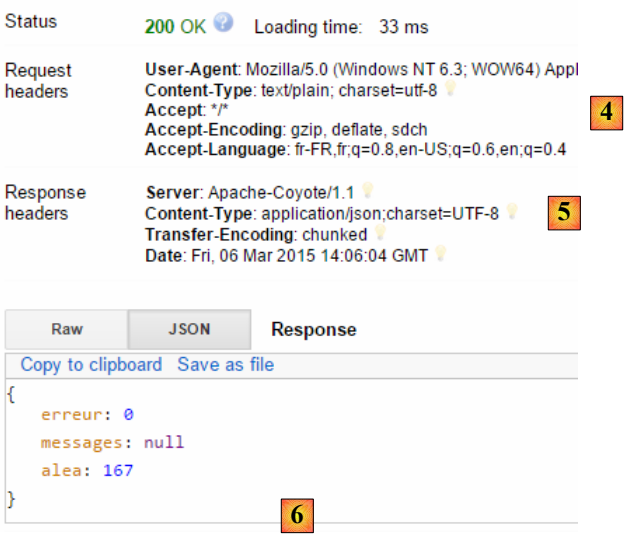


Nous obtenons à chaque fois, la représentation jSON d'un objet de type [AleaResponse].

Au lieu de prendre un navigateur standard, prenons maintenant l'extension [Advanced Rest Client] du navigateur Chrome (voir annexes, paragraphe 1.16.8, page 198) :



- en [1], l'URL demandée ;
- en [2], au moyen d'un GET ;
- en [3], on envoie la requête ;



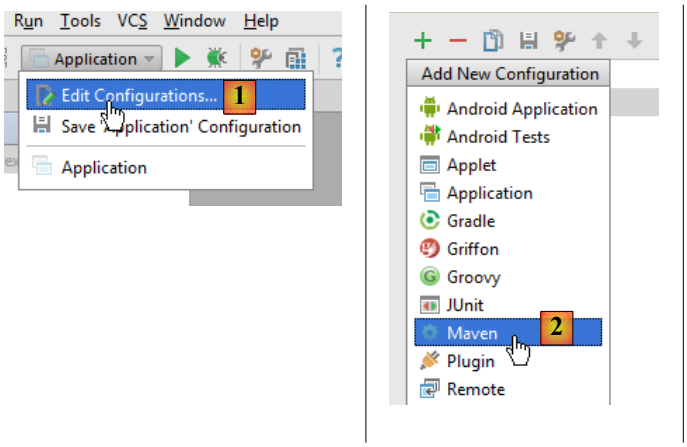
- en [4], les entêtes HTTP envoyés par [Advanced Rest Client] ;
- en [5], les entêtes HTTP de la réponse du serveur. On remarquera que celui-ci indique que le document envoyé est une chaîne jSON ;
- en [6], la chaîne jSON reçue ;

1.11.1.6 Génération du jar exécutable du projet

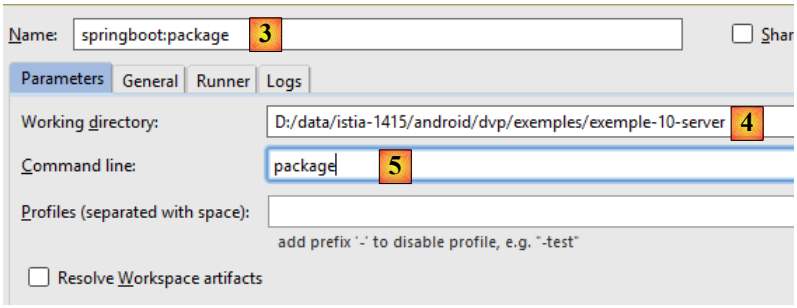
Dans le fichier [pom.xml], on trouve les lignes suivantes :

```
1.        <plugin>
2.          <groupId>org.springframework.boot</groupId>
3.          <artifactId>spring-boot-maven-plugin</artifactId>
4.  </plugin>
```

[spring-boot-maven-plugin] est un plugin pour Maven qui permet de générer le jar exécutable d'un projet Spring Boot. Il y a différentes façon de le générer. Procédons ainsi :



- en [1-2], on crée une nouvelle configuration Maven ;



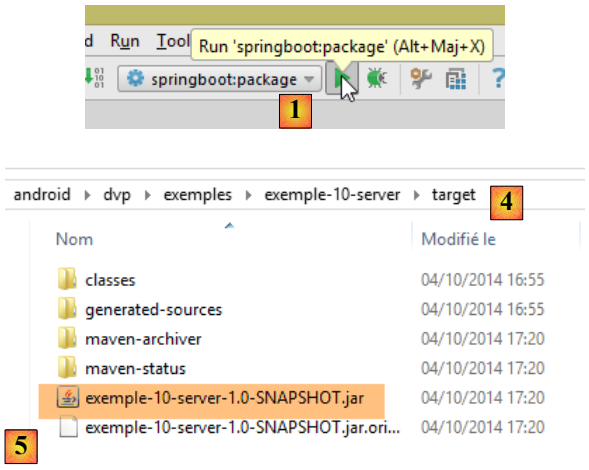
- en [3], donnez un nom à la configuration d'exécution ;
- en [4], naviguez jusqu'au dossier du projet ;
- en [5], le Build Maven aura un but (goal) : **package** (pour générer le jar exécutable) ;

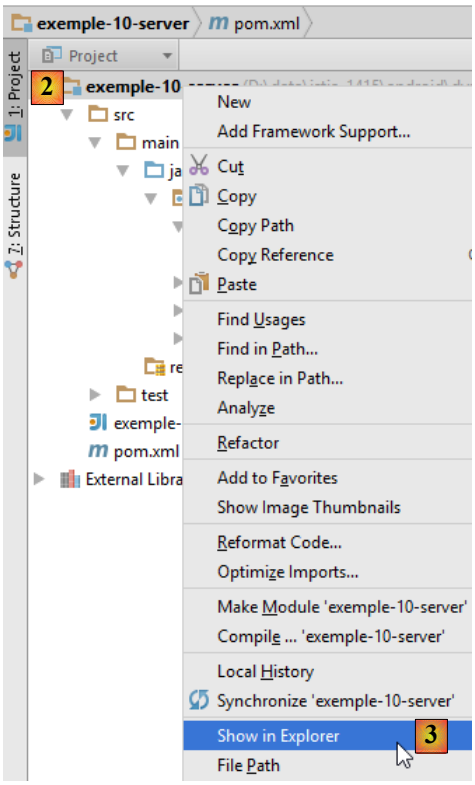
Avant d'exécuter cette configuration Maven, il faut vérifier le fichier [pom.xml] :

```
1.    <properties>
2.      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
3.      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
4.      <start-class>android.exemples.server.boot.Application</start-class>
5.  </properties>
```

- ligne 4, on doit préciser le nom complet de la classe exécutable du projet. Cette information sera en effet enregistrée dans le jar produit par l'exécution du projet Maven ;

On exécute cette configuration [1] :





- une fois l'exécution terminée, on ouvre [2-3] le dossier du projet dans l'explorateur windows ;
- on trouve [5] l'archive jar produite par l'exécution de la configuration Maven dans le fichier [target] [4] du projet ;

Ouvrez une console DOS et placez-vous sur le dossier [target] [4] ci-dessus, puis tapez la commande suivante :

```
...\exemples\exemple-10-server\target>java -jar exemple-10-server-1.0-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.1.1.RELEASE)

2014-10-07 09:23:12.190  INFO 8028 --- [           main] a.exemples.server.boot.Application       :
Starting Application on Gportpers3 with PID 8028 (D:\data\istia-1415\android\dvp\exemples\exemple-10-
server\target\exemple-10-server-1.0-SNAPSHOT.jar started by ST in D:\data\istia-
1415\android\dvp\exemples\exemple-10-server\target)
2014-10-07 09:23:12.255  INFO 8028 --- [           main] ationConfigEmbeddedWebApplicationContext :
Refreshing
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@5bfdde41: startup
date [Tue Oct 07 09:23:12 CEST 2014]; root of context hierarchy
2014-10-07 09:23:13.136  INFO 8028 --- [           main] o.s.b.f.s.DefaultListableBeanFactory     :
Overriding bean definition for bean 'beanNameViewResolver': replacing [Root bean: class [null]; scope=;
abstract=false; lazyInit=false; autowireMode=3; dependencyCheck=0; autowireCandidate=true; primary=false;
factoryBeanName=org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$WhitelabelErrorViewCo
nfiguration; factoryMethodName=beanNameViewResolver; initMethodName=null; destroyMethodName=(inferred);
defined in class path resource
[org/springframework/boot/autoconfigure/web/ErrorMvcAutoConfiguration$WhitelabelErrorViewConfiguration.cla
ss]] with [Root bean: class [null]; scope=; abstract=false; lazyInit=false; autowireMode=3;
dependencyCheck=0; autowireCandidate=true; primary=false;
factoryBeanName=org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration$WebMvcAutoConfiguration
Adapter; factoryMethodName=beanNameViewResolver; initMethodName=null; destroyMethodName=(inferred);
defined in class path resource
[org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration$WebMvcAutoConfigurationAdapter.class]]
2014-10-07 09:23:14.520  INFO 8028 --- [           main] .t.TomcatEmbeddedServletContainerFactory : Server
initialized with port: 8080
2014-10-07 09:23:14.798  INFO 8028 --- [           main] o.apache.catalina.core.StandardService   :
Starting service Tomcat
2014-10-07 09:23:14.798  INFO 8028 --- [           main] org.apache.catalina.core.StandardEngine  :
Starting Servlet Engine: Apache Tomcat/7.0.54
2014-10-07 09:23:14.966  INFO 8028 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       :
Initializing Spring embedded WebApplicationContext
2014-10-07 09:23:14.967  INFO 8028 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root
WebApplicationContext: initialization completed in 2716 ms
2014-10-07 09:23:15.841  INFO 8028 --- [ost-startStop-1] o.s.b.c.e.ServletRegistrationBean        :
Mapping servlet: 'dispatcherServlet' to [/]
2014-10-07 09:23:15.844  INFO 8028 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean  :
Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2014-10-07 09:23:16.369  INFO 8028 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped
URL path [/**/favicon.ico] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2014-10-07 09:23:16.592  INFO 8028 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"{[/{a}/{b}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
android.exemples.server.web.AleaResponse android.exemples.server.web.AleaController.getAlea(int,int)
2014-10-07 09:23:16.596  INFO 8028 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"{[/error],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>>
org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletReques
t)
2014-10-07 09:23:16.597  INFO 8028 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"{[/error],methods=[],params=[],headers=[],consumes=[],produces=[text/html],custom=[]}" onto public
org.springframework.web.servlet.ModelAndView
org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRe
quest)
2014-10-07 09:23:16.655  INFO 8028 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped
URL path [/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2014-10-07 09:23:16.655  INFO 8028 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped
URL path [/webjars/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2014-10-07 09:23:17.245  INFO 8028 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        :
Registering beans for JMX exposure on startup
2014-10-07 09:23:17.317  INFO 8028 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat
started on port(s): 8080/http
```

```
29. 2014-10-07 09:23:17.320  INFO 8028 --- [           main] a.exemples.server.boot.Application       :
    Started Application in 5.88 seconds (JVM running for 6.674)
```

- ligne 1 : on utilise l'exécutable [java.exe] qui est ici dans le PATH du DOS. Si ce n'est pas le cas, utilisez le chemin complet de l'exécutable, en général "C:\Program Files\java\jdkxxx\bin\java.exe" ;
- à partir de la ligne 3, les logs de Spring Boot ;

Prenez un navigateur et demandez l'URL [localhost:8080/100/200].

1.11.2 Le client Android du serveur web / jSON

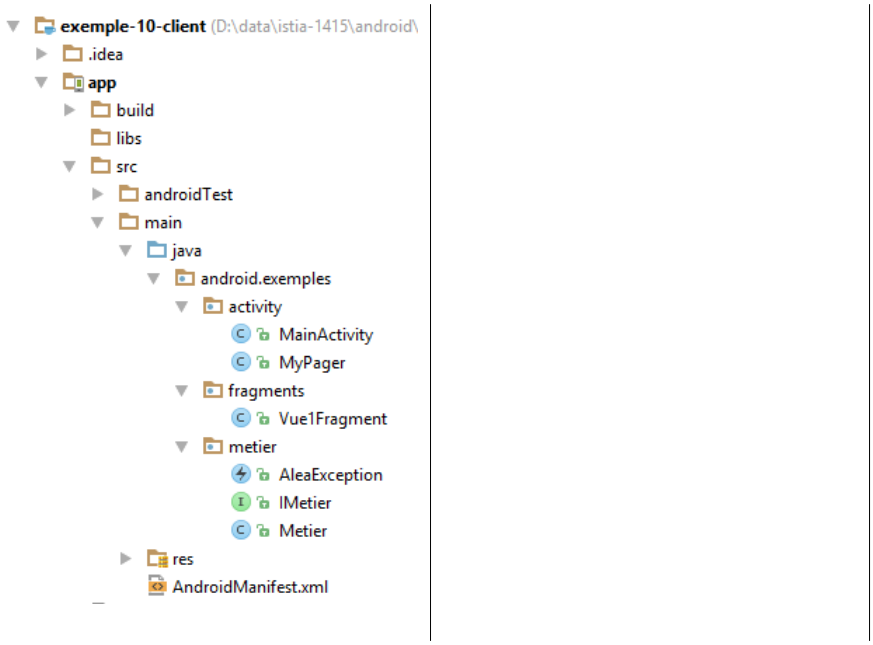
Le client Android aura l'architecture suivante :

Le client aura deux composantes :

1. une couche [Présentation] (vue+activité) analogue à celle que nous avons étudiée dans l'exemple [exemple-09] ;
2. la couche [DAO] qui s'adresse au service [web / jSON] que nous avons étudié précédemment.

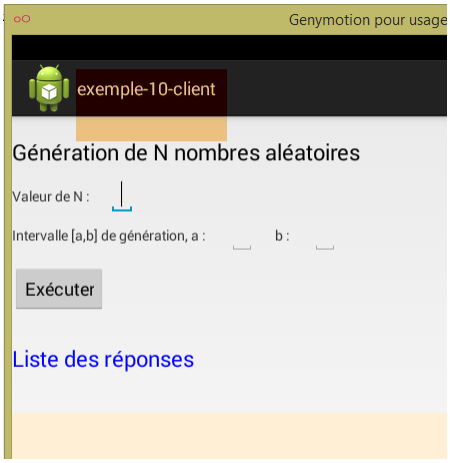
1.11.2.1 Création du projet

Le projet Android s'appellera [exemple-10-client] et est obtenu par recopie du projet [exemple-09]. En effet, on veut réutiliser certains éléments de ce projet



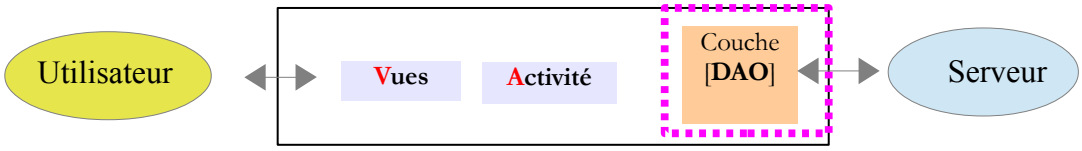
Rappelons les étapes de la recopie (cf paragraphe 1.10.1, page 91) :

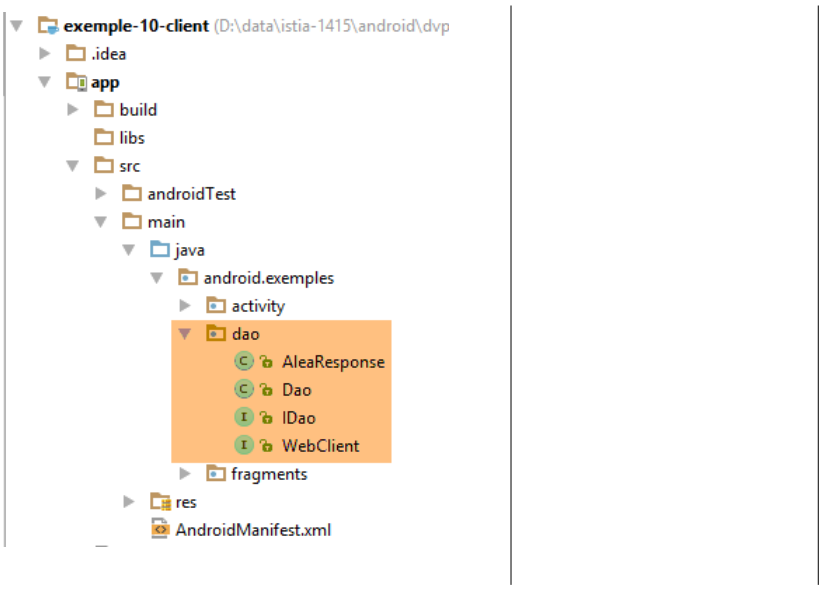
1. copier / coller le projet [exemple-09] dans [exemple-10-client] ;
2. modifier les noms du projet et le nom du module ;
3. changer le nom du projet dans [src/main/res/values/strings.xml] ;
4. rafraîchir le projet [Gradle] ;
5. compiler le projet
6. l'exécuter ;



Dans la suite, le lecteur est invité à créer le projet qui suit.

1.11.2.2 La couche [DAO]





1.11.2.2.1 L'interface [IDao] de la couche [DAO]

L'interface de la couche [DAO] sera la suivante :

```
1.  package android.exemples.dao;
2.
3.  public interface IDao {
4.
5.    // nombre aléatoire
6.    public AleaResponse getAlea(int a, int b);
7.    // URL du service web
8.    public void setUrlServiceWebJson(String url);
9.    // délai d'attente (ms) max de la réponse du serveur
```

```
10.   public void setTimeout(int timeout);
11. }
```

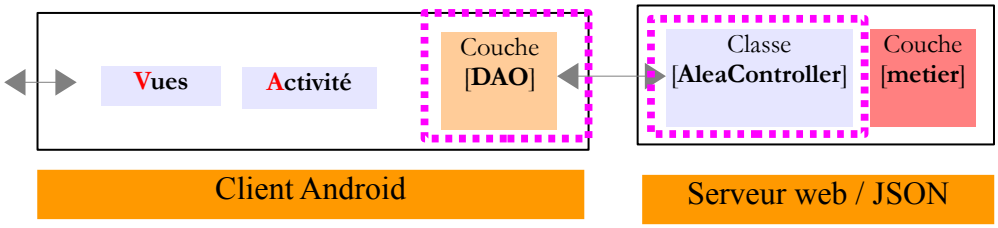
- ligne 8 : l'URL du service web / jSON de génération de nombres aléatoires ;
- ligne 6 : la méthode pour obtenir un nombre aléatoire dans l'intervalle [a,b] de ce service web ;
- ligne 10 : on se fixe un délai d'attente maximal pour attendre la réponse du serveur ;

1.11.2.2.2 La classe [AleaResponse]

La méthode [getAlea] de l'interface [IDao] rend une réponse du type [AleaResponse] suivant :

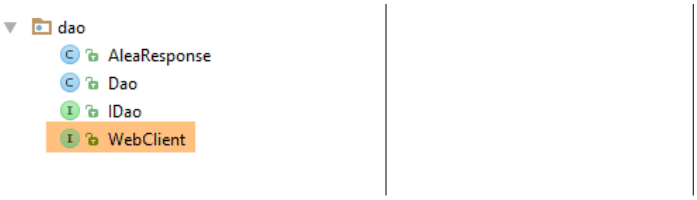
```
1.  package android.exemples.dao;
2.
3.  import java.util.List;
4.
5.  public class AleaResponse {
6.
7.    // data
8.    private int erreur;
9.    private List<String> messages;
10.   private int alea;
11.
12.   // getters et setters
13. ...
14. }
```

C'est la classe [AleaResponse] déjà utilisée côté serveur. En fait, d'un point de vue programmation, tout se passe comme si la couche [DAO] du client communiquait directement avec le contrôleur [AleaController] du service web :



La communication réseau entre client et serveur ainsi que la sérialisation / désérialisation des objets Java sont transparents au programmeur.

1.11.2.2.3 La classe [WebClient]



La classe [WebClient] s'occupe de dialoguer avec le service web. Son code est le suivant :

```
1.  package android.exemples.dao;
2.
3.  import org.androidannotations.annotations.rest.Get;
4.  import org.androidannotations.annotations.rest.Rest;
5.  import org.androidannotations.api.rest.RestClientRootUrl;
6.  import org.androidannotations.api.rest.RestClientSupport;
7.  import org.springframework.http.converter.json.MappingJacksonHttpMessageConverter;
8.  import org.springframework.web.client.RestTemplate;
9.
10. @Rest(converters = {MappingJacksonHttpMessageConverter.class})
11. public interface WebClient extends RestClientRootUrl, RestClientSupport {
12.
13.   // 1 nombre aléatoire dans l'intervalle [a,b]
14.   @Get("/{a}/{b}")
```

```
15.    public AleaResponse getAlea(int a, int b);
16. }
```

- ligne 10 : l'annotation [@Rest] est une annotation AA. La bibliothèque AA va implémenter elle-même cette interface. Celle-ci doit implémenter les appels aux URL exposées par le service web / jSON ;
- lignes 14-15 : le service web n'expose qu'une URL (dans [AleaController]) :

```
1.    // nombres aléatoires
2.    @RequestMapping(value = "/{a}/{b}", method = RequestMethod.GET)
3.  public AleaResponse getAlea(@PathVariable("a") int a, @PathVariable("b") int b) {
```

Pour chaque URL exposée par le service web, on peut ajouter dans l'interface [WebClient] ci-dessus, une méthode reprenant la signature de l'URL exposée ;

- ligne 15 : la signature de la méthode appelée dans [AleaController] ;
- ligne 14 : une annotation AA indiquant que l'URL doit être appelée avec une méthode HTTP GET. Le paramètre de l'annotation [@Get] est la forme de l'URL attendue par le service web. Il suffit de reprendre le paramètre [value] de l'annotation [@RequestMapping] de la méthode appelée dans [AleaController].

Dans le cas d'une requête HTTP POST, la méthode d'appel aurait la signature suivante :

```
@Post("/{a}/{b}")
  public AleaResponse getAlea(T body,int a, int b);
```

où [T body] est l'objet posté. Celui-ci sera automatiquement sérialisé en jSON. Côté serveur, on aura la signature suivante :

```
1.    // nombres aléatoires
2.    @RequestMapping(value = "/{a}/{b}", method = RequestMethod.POST, consumes = "application/json")
3. public AleaResponse getAlea(@PathVariable("a") int a, @PathVariable("b") int b, @RequestBody T
body) {
```

- ligne 2 : on précise qu'on attend une requête HTTP POST et que le corps de cette requête (objet posté) doit être transmis sous forme d'une chaîne jSON ;
- ligne 3 : La valeur postée sera récupérée dans le paramètre [@RequestBody T body] de la méthode.

Revenons au code de la classe [WebClient] :

```
1.  @Rest(converters = {MappingJacksonHttpMessageConverter.class})
2.  public interface WebClient extends RestClientRootUrl, RestClientSupport {
3.  ...
4.  }
```

- ligne 1 : on précise la bibliothèque jSON utilisée, ici la bibliothèque [Jackson] ;
- il nous faut pouvoir indiquer l'URL du service web à contacter. Ceci est obtenu en étendant l'interface [RestClientRootUrl] fourni par AA. Cette interface expose une méthode [setRootUrl(urlServiceWeb] qui permet de fixer l'URL du service web à contacter ;
- par ailleurs, nous voulons contrôler l'appel au service web car nous voulons limiter le temps d'attente de la réponse. Pour cela, nous étendons l'interface [RestClientSupport] qui expose la méthode [setRestTemplate] qui nous permettra de fixer ce temps d'attente ;

1.11.2.2.4 Implémentation de la couche [DAO]

L'interface [IDao] est implémentée avec la classe [Dao] suivante :

```
1.  package android.exemples.dao;
2.
3.  import org.androidannotations.annotations.EBean;
4.  import org.androidannotations.annotations.rest.RestService;
5.  import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;
```

```
6.  import org.springframework.http.converter.json.MappingJacksonHttpMessageConverter;
7.  import org.springframework.web.client.RestTemplate;
8.
9.  import java.util.ArrayList;
10. import java.util.List;
11.
12. @EBean
13. public class Dao implements IDao {
14.
15.   // client du service REST
16.   @RestService
17.   WebClient webClient;
18.
19.   @Override
20.   public AleaResponse getAlea(int a, int b) {
21. ....
22.   }
23.
24.   @Override
25.   public void setUrlServiceWebJson(String urlServiceWebJson) {
26.     ...
27.   }
28.
29.   @Override
30.   public void setTimeout(int timeout) {
31.   ...
32.   }
33.
34. }
```

- ligne 12 : nous annotons la classe [Dao] avec l'annotation [@EBean] pour en faire un bean AA qu'on va pouvoir injecter ailleurs ;
- lignes 16-17 : nous injectons l'implémentation qui sera faite de l'interface [WebClient] que nous avons décrite. C'est l'annotation [@RestService] qui assure cette injection ;
- les autres méthodes implémentent l'interface [IDao] (ligne 13) ;

Méthode [getAlea]

La méthode [getAlea] est la suivante :

```
1.  @Override
2.    public AleaResponse getAlea(int a, int b) {
3.      // exécution service
4.      AleaResponse info;
5.      try {
6.        info= webClient.getAlea(a, b);
7.      } catch (Exception ex) {
8.        // cas d'erreur
9.        info = new AleaResponse();
10.       info.setErreur(-1);
11.       info.setMessages(getMessagesFromException(ex));
12.     }
13.     // résultat
14.     return info;
15.   }
16.
17.   private List<String> getMessagesFromException(Exception ex) {
18.     // on crée une liste avec les msg d'erreur de la pile d'exceptions
19.     List<String> messages = new ArrayList<String>();
20.     Throwable th = ex;
21.     while (th != null) {
22.       messages.add(th.getMessage());
23.       th = th.getCause();
24.     }
25.     return messages;
26.   }
```

- ligne 6 : on se contente d'appeler la méthode de même signature dans la classe implémentant l'interface [WebClient] ;
- lignes 9-11 : le cas où il se produit une exception ;

Méthode [setUrlServiceWebJson]

La méthode [setUrlServiceWebJson] est la suivante :

```
1.    @Override
2.    public void setUrlServiceWebJson(String urlServiceWebJson) {
3.      // on fixe l'URL du service REST
4.      webClient.setRootUrl(urlServiceWebJson);
5.  }
```

- ligne 4 : on fixe l'URL du service web via la méthode [setRootUrl] de l'interface [WebClient]. C'est parce que cette interface étend l'interface [RestClientRootUrl] que cette méthode existe ;

Méthode [setTimeout]

La méthode [setTimeout] est la suivante :

```
1.    @Override
2.    public void setTimeout(int timeout) {
3.      // on fixe le timeout des requêtes du client REST
4.      HttpComponentsClientHttpRequestFactory factory = new HttpComponentsClientHttpRequestFactory();
5.      factory.setReadTimeout(timeout);
6.      factory.setConnectTimeout(timeout);
7.      RestTemplate restTemplate = new RestTemplate(factory);
8.      restTemplate.getMessageConverters().add(new MappingJacksonHttpMessageConverter());
9.      webClient.setRestTemplate(restTemplate);
10. }
```

- l'interface [WebClient] va être implémentée par une classe AA utilisant la dépendance Gradle [org.springframework.android:spring-android-rest-template]. [spring-android-rest-template] implémente le dialogue du client avec le serveur web / jSON au moyen d'une classe de type [RestTemplate] ;
- ligne 4 : la classe [HttpComponentsClientHttpRequestFactory] est fournie par la dépendance [spring-android-rest-template]. Elle va nous permettre de fixer le délai d'attente maximum de la réponse du serveur (lignes 5-6) ;
- ligne 7 : nous construisons l'objet de type [RestTemplate] qui va être le support de la communication avec le service web. Nous lui passons comme paramètre l'objet [factory] qui vient d'être construit ;
- ligne 8 : le dialogue client / serveur peut prendre diverses formes. Les échanges se font par lignes de texte et nous devons indiquer à l'objet de type [RestTemplate] ce qu'il doit faire avec cette ligne de texte. Pour cela, nous lui fournissons des convertisseurs, des classes capables de traiter les lignes de texte. Le choix du convertisseur se fait en général via les entêtes HTTP qui accompagnent la ligne de texte. Ici, nous savons que nous recevons uniquement des lignes de texte au format jSON. Par ailleurs, nous avons vu page 117, que le serveur envoyait l'entête HTTP :

  ```
  Content-Type: application/json;charset=UTF-8
  ```

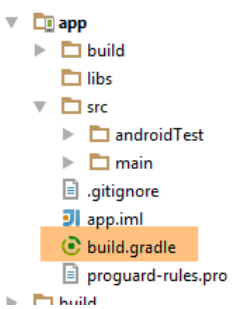
  Ligne 8, l'unique convertisseur du [RestTemplate] sera un convertisseur jSON implémenté avec la bibliothèque [Jackson]. Il y a une bizarrerie à propos de ces convertisseurs : AA nous impose de l'avoir également dans l'annotation du client web [WebClient] :

  ```
  1. @Rest(converters = {MappingJacksonHttpMessageConverter.class})
  2. public interface WebClient extends RestClientRootUrl, RestClientSupport {
  ```

  Ligne 1, on est obligé de préciser un convertisseur alors même que nous le précisons par programmation.
- ligne 9 : l'objet [RestTemplate] ainsi construit est injecté dans l'implémentation de l'interface [WebClient] et c'est cet objet qui va opérer le dialogue client / serveur ;

1.11.2.3 Les dépendances Gradle

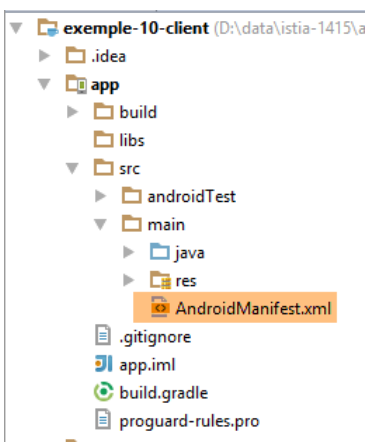
La couche [DAO] a besoin de dépendances que nous inscrivons dans le fichier [app / build.gradle] :




```
1.  dependencies {
2.    apt "org.androidannotations:androidannotations:$AAVersion"
3.    compile "org.androidannotations:androidannotations-api:$AAVersion"
4.    compile 'com.android.support:support-v13:19.1.0'
5.    compile 'com.android.support:appcompat-v7:19.1.0'
6.    compile 'org.springframework.android:spring-android-rest-template:1.0.1.RELEASE'
7.    compile 'org.codehaus.jackson:jackson-mapper-asl:1.9.9'
8.  }
```

- la dépendance de la ligne 6 amène l'objet [RestTemplate] qui gère le dialogue client / serveur ;
- la dépendance de la ligne 7 amène la bibliothèque jSON [Jackson] ;

1.11.2.4 *Le manifeste de l'application Android*



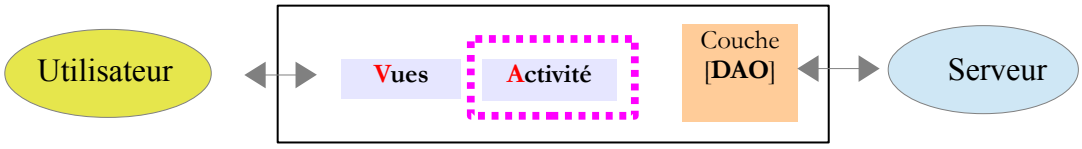
Le fichier [AndroidManifest.xml] doit évoluer. En effet, par défaut, les accès Internet sont désactivés. Il faut les activer par une directive spéciale :

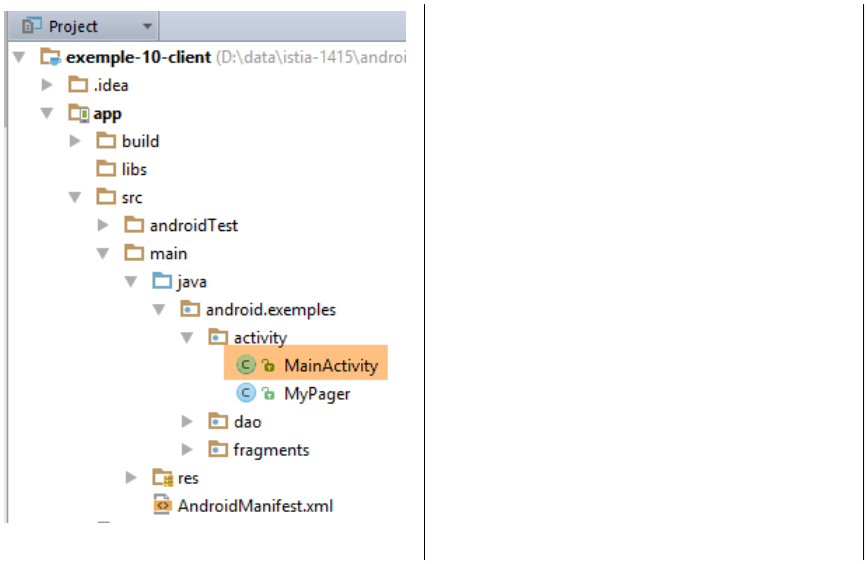
```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.            package="android.exemples">
4.
5.    <uses-sdk
6.      android:minSdkVersion="11"
7.      android:targetSdkVersion="20"/>
8.
9.    <uses-permission android:name="android.permission.INTERNET"/>
10.
11.   <application
12.     android:allowBackup="true"
13.     android:icon="@drawable/ic_launcher"
14.     android:label="@string/app_name"
15.     android:theme="@style/AppTheme">
16.     <activity
17.       android:name=".activity.MainActivity_"
18.       android:label="@string/app_name"
```

```
19.       android:windowSoftInputMode="stateHidden">
20.       <intent-filter>
21.         <action android:name="android.intent.action.MAIN"/>
22.
23.         <category android:name="android.intent.category.LAUNCHER"/>
24.       </intent-filter>
25.     </activity>
26.   </application>
27.
28. </manifest>
```

- ligne 9 : les accès internet sont autorisés ;

1.11.2.5 *L'activité [MainActivity]*



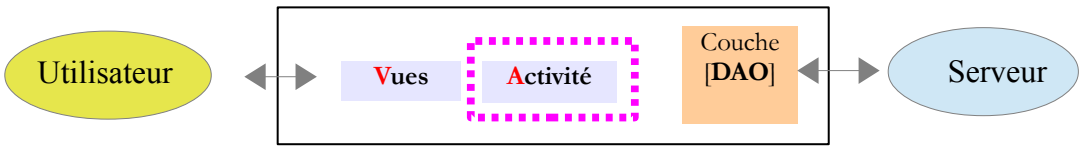


L'activité [MainActivity] bouge peu vis à vis de ce qu'elle était dans [exemple-09] :

```
1.  package android.exemples.activity;
2.
3.  import android.exemples.R;
4.  import android.exemples.dao.AleaResponse;
5.  import android.exemples.dao.Dao;
6.  import android.exemples.dao.IDao;
7.  import android.exemples.fragments.Vue1Fragment_;
8.  import android.os.Bundle;
9.  import android.support.v4.app.Fragment;
10. import android.support.v4.app.FragmentActivity;
11. import android.support.v4.app.FragmentManager;
12. import android.support.v4.app.FragmentPagerAdapter;
13. import android.view.Window;
14. import org.androidannotations.annotations.*;
15.
16. @EActivity(R.layout.main)
17. public class MainActivity extends FragmentActivity implements IDao {
18.
19.   // la couche [DAO]
20.   @Bean(Dao.class)
21.   protected IDao dao;
```

```
22.
23.   // délai d'attente avant service
24.   private int delay;
25.   // délai d'attente de la réponse
26.   private final int TIMEOUT = 1000;
27.
28.   // le gestionnaire de fragments ou sections
29.   SectionsPagerAdapter mSectionsPagerAdapter;
30.
31.   // le conteneur des fragments
32.   @ViewById(R.id.pager)
33.   MyPager mViewPager;
34.
35.   @Override
36.   protected void onCreate(Bundle savedInstanceState) {
37.     super.onCreate(savedInstanceState);
38.     // il faut installer le sablier avant de créer la vue
39.     requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
40.   }
41.
42.   @AfterViews
43.   protected void initActivity() {
44.
45.     // ne change pas
46.
47.   }
48.
49.   @AfterInject
50.   protected void initDao() {
51.     // on paramètre la couche [DAO]
52.     setTimeout(TIMEOUT);
53.   }
54.
55.   // implémentation interface [IDao] ----------------------------------------
56.
57.   @Override
58.   public AleaResponse getAlea(int a, int b) {
59.     // attente
60.     waitSomeTime(delay);
61.     // service
62.     return dao.getAlea(a, b);
63.   }
64.   @Override
65.   public void setUrlServiceWebJson(String url) {
66.     dao.setUrlServiceWebJson(url);
67.   }
68.
69.   @Override
70.   public void setTimeout(int timeout) {
71.     dao.setTimeout(timeout);
72.   }
73.
74.   // notre gestionnaire de fragments à redéfinir pour chaque application
75.   // doit définir les méthodes suivantes : getItem, getCount, getPageTitle
76.   public class SectionsPagerAdapter extends FragmentPagerAdapter {
77.
78.     // ... ne change pas
79.   }
80.
81.   private void waitSomeTime(int delay) {
82.     try {
83.       Thread.sleep(delay);
84.     } catch (InterruptedException e) {
85.       e.printStackTrace();
86.     }
87.   }
88.
89.   public void setDelay(int delay) {
90.     this.delay = delay;
91.   }
92. }
```

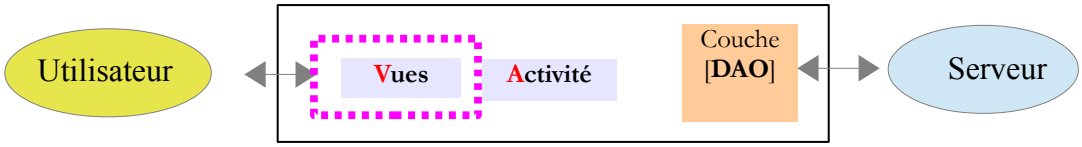
- ligne 17 : l'activité implémente l'interface [IDao]. C'est un choix d'architecture correspondant au schéma suivant :

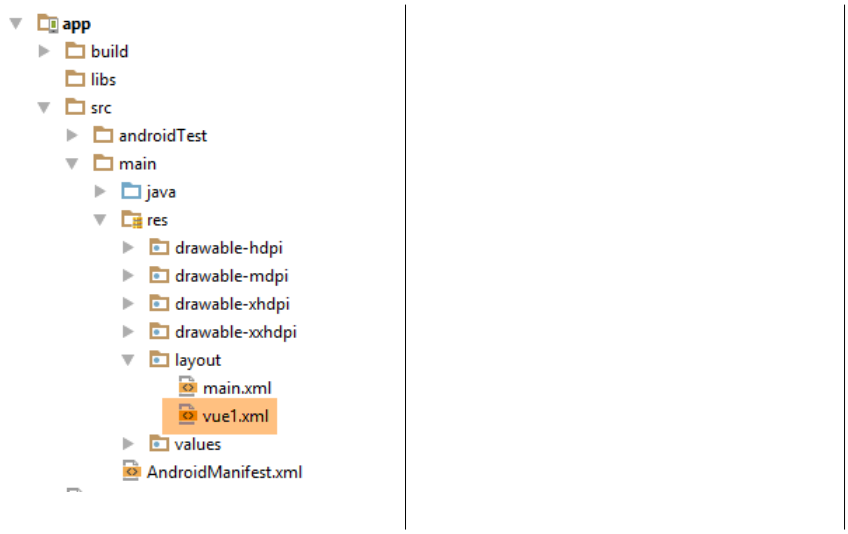


La vue n'a que l'activité comme interlocuteur. C'est celle-ci qui assure le dialogue avec la couche [DAO] ;

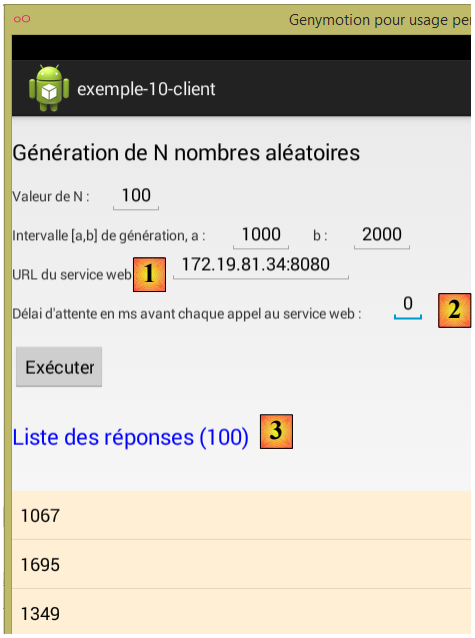
- lignes 20-21 : injection d'une référence sur la couche [DAO] ;
- lignes 57-72 : implémentation de l'interface [IDao] ;
- ligne 60 : avant de demander un nombre aléatoire à la couche [DAO], on s'arrêtera [delay] millisecondes ;
- lignes 49-53 : une fois obtenue la référence sur la couche [DAO] (@AfterInject), on fixe le [timeout] de celle-ci ;
- lignes 38-39 : on installe un widget qui permet de signaler une attente avec une roue qui tourne ;

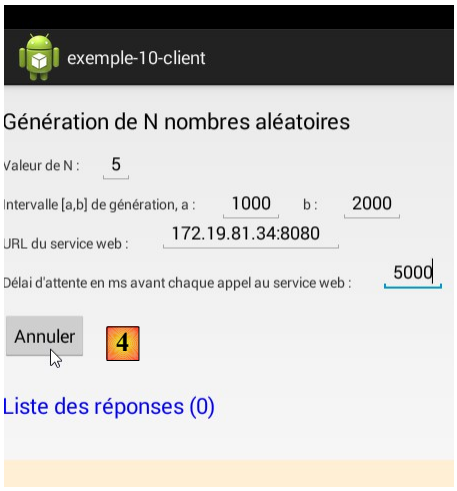
1.11.2.6 *La vue [vue1.xml]*

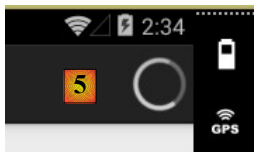




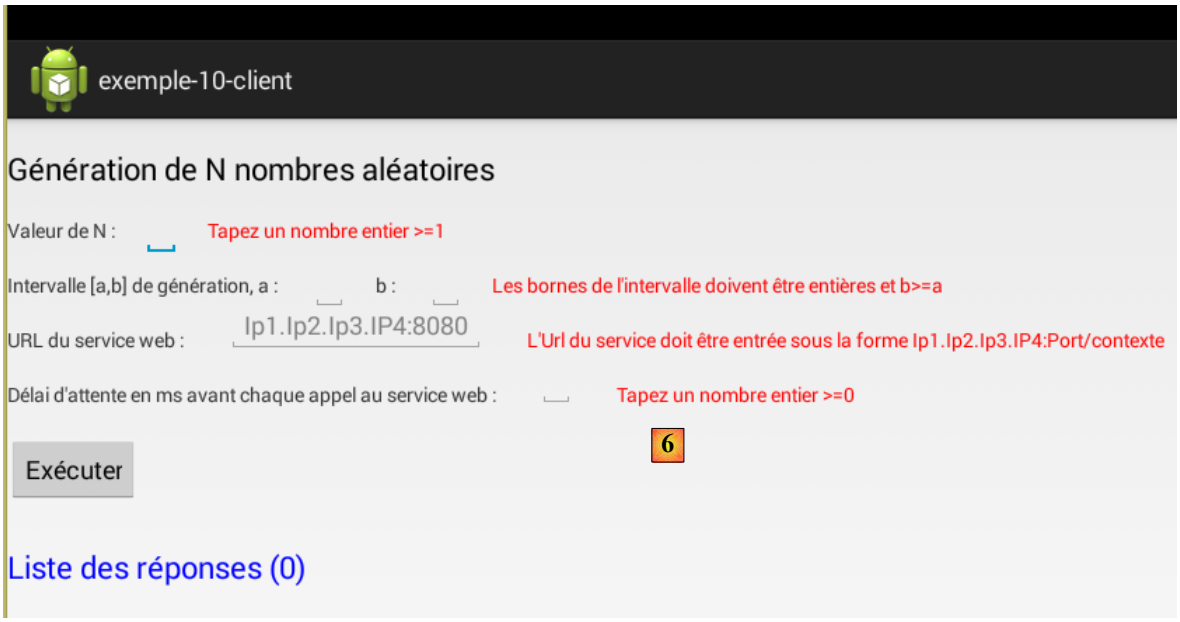
La vue [vue1.xml] évolue de la façon suivante :





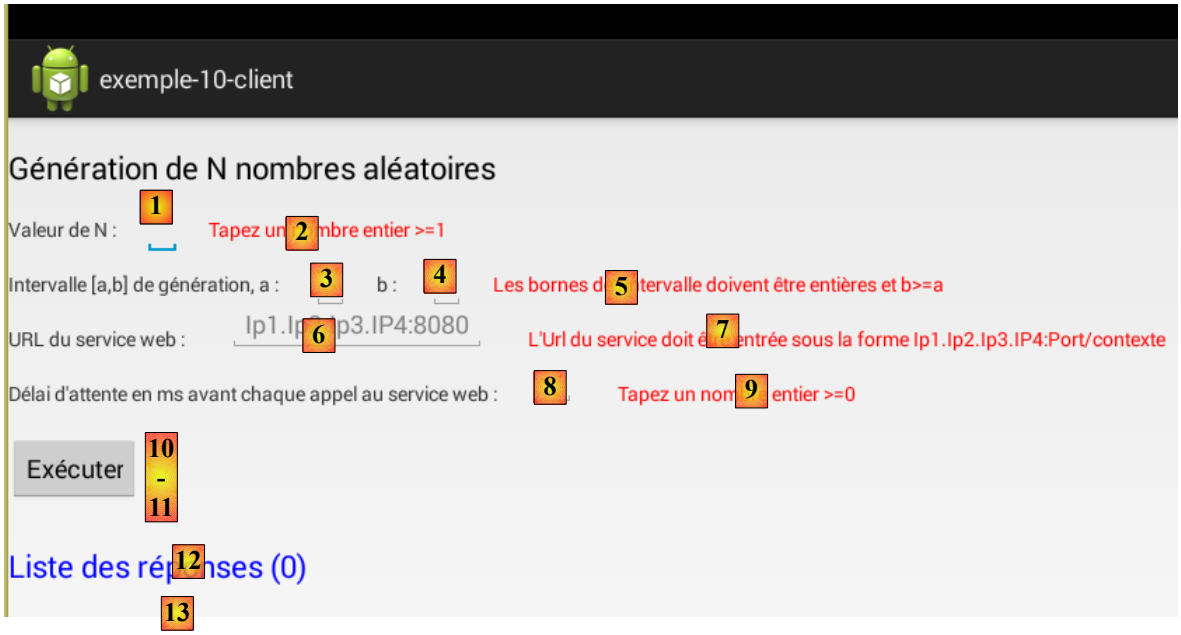


- en [1], l'utilisateur doit préciser l'URL du service web ainsi que le délai d'attente [2] avant chaque appel au service web ;
- en [3], les réponses sont comptées ;
- en [4], l'utilisateur peut annuler sa demande ;
- en [5], un indicateur d'attente s'affiche lorsque les nombres sont demandés. Il s'efface lorsqu'ils ont été tous reçus ou que l'opération a été annulée ;



- en [6], la validité des saisies est contrôlée ;

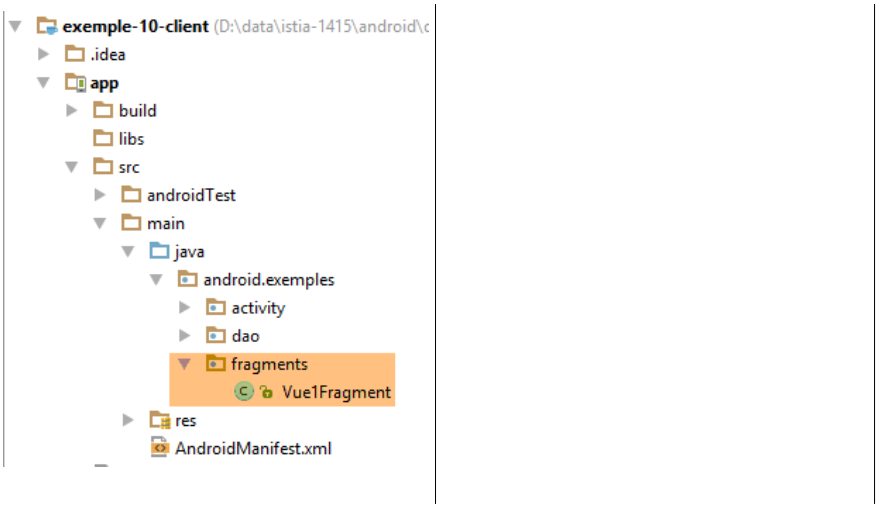
Le lecteur est invité à charger le fichier [vue1.xml] à partir des exemples. Pour la suite, nous donnons l'identifiant des nouveaux composants :



n°	Type	Id
1	EditText	edt_nbaleas
2	TextView	txt_errorNbAleas
3	EditText	edt_a
4	EditText	edt_b
5	TextView	txt_errorIntervalle
6	EditText	editTextUrlServiceWeb
7	TextView	textViewErreurUrl
8	EditText	editTextDelay
9	TextView	textViewErreurDelay
10	Button	btn_Executer
11	Button	btn_Annuler
12	TextView	txt_Reponses
13	ListView	lst_reponses

Les boutons [10-11] sont physiquement l'un sur l'autre. A un moment donné, on ne rendra visible que l'un des deux.

1.11.2.7 *Le fragment [Vue1Fragment]*



Le squelette du fragment [Vue1Fragment] est le suivant :

```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.exemples.activity.MainActivity_;
5.  import android.exemples.dao.AleaResponse;
6.  import android.support.v4.app.Fragment;
7.  import android.view.View;
8.  import android.widget.*;
9.  import org.androidannotations.annotations.*;
10. import org.androidannotations.api.BackgroundExecutor;
11.
12. import java.net.URI;
13. import java.net.URISyntaxException;
14. import java.util.ArrayList;
15. import java.util.List;
16.
17. @EFragment(R.layout.vue1)
18. public class Vue1Fragment extends Fragment {
19.
20.   // les éléments de l'interface visuelle
21.   @ViewById(R.id.editTextUrlServiceWeb)
22.   EditText edtUrlServiceRest;
23.   @ViewById(R.id.textViewErreurUrl)
24.   TextView txtMsgErreurUrlServiceWeb;
25.   @ViewById(R.id.editTextDelay)
26.   EditText edtDelay;
27.   @ViewById(R.id.textViewErreurDelay)
28.   TextView textViewErreurDelay;
29.   @ViewById(R.id.lst_reponses)
30.   ListView listReponses;
31.   @ViewById(R.id.txt_Reponses)
32.   TextView infoReponses;
33.   @ViewById(R.id.edt_nbaleas)
34.   EditText edtNbAleas;
35.   @ViewById(R.id.edt_a)
36.   EditText edtA;
37.   @ViewById(R.id.edt_b)
38.   EditText edtB;
39.   @ViewById(R.id.txt_errorNbAleas)
40.   TextView txtErrorAleas;
41.   @ViewById(R.id.txt_errorIntervalle)
42.   TextView txtErrorIntervalle;
43.   @ViewById(R.id.btn_Executer)
44.   Button btnExecuter;
45.   @ViewById(R.id.btn_Annuler)
46.   Button btnAnnuler;
47.
48.   // l'activité principale
```

```
49.   private MainActivity_ activité;
50.
51.   @AfterViews
52.   void initFragment() {
53.     // activité
54.     activité = (MainActivity_) getActivity();
55.     // au départ pas de messages d'erreur
56.     txtErrorAleas.setText("");
57.     txtErrorIntervalle.setText("");
58.     txtMsgErreurUrlServiceWeb.setText("");
59.     textViewErreurDelay.setText("");
60.     // boutons [Annuler] caché, [Exécuter] visible
61.     btnAnnuler.setVisibility(View.INVISIBLE);
62.     btnExecuter.setVisibility(View.VISIBLE);
63.   }
64.
65. }
```

- lignes 21-46 : les références sur les composants de la vue [vue1.xml] (ligne 17) ;
- ligne 49 : la référence sur l'unique activité [MainActivity_] qui sera générée par AA ;
- lignes 56-59 : les messages d'erreur sont effacés au cas où par construction, ils ne seraient pas vides. Une autre façon d'arriver au même résultat est de cacher le composant comme il est fait ligne 61 ;
- lignes 61-62 : on cache le bouton [Annuler] (ligne 61) et on affiche le bouton [Exécuter] (ligne 62). On rappelle qu'ils sont physiquement l'un sur l'autre ;

Le 'clic' sur le bouton [Exécuter] provoque l'exécution de la méthode suivante :

```
1.    // les saisies
2.    private int nbAleas;
3.    private int a;
4.    private int b;
5.    private String urlServiceWebJson;
6.    private int delay;
7.
8.    // données locales
9.    private int nbInfos;
10.   private List<String> reponses;
11.
12. ...
13. @Click(R.id.btn_Executer)
14.   protected void doExecuter() {
15.     // on efface les reponses précédentes
16.     reponses = new ArrayList<String>();
17.     listReponses.setAdapter(new ArrayAdapter<String>(activité, android.R.layout.simple_list_item_1,
    android.R.id.text1, reponses));
18.     // on remet à 0 le compteur de réponses
19.     nbInfos=0;
20.     infoReponses.setText(String.format("Liste des réponses (%s)",nbInfos));
21.     // on teste la validité des saisies
22.     if (!isPageValid()) {
23.       return;
24.     }
25.     // initialisation activité
26.     activité.setUrlServiceWebJson(urlServiceWebJson);
27.     activité.setDelay(delay);
28.     // on demande les nombres aléatoires
29.     for (int i = 0; i < nbAleas; i++) {
30.       getAlea(a, b);
31.     }
32.     // on commence l'attente
33.     beginWaiting();
34.   }
35.
36.   @Background(id = "alea")
37.   void getAlea(int a, int b) {
38.     showInfo(activité.getAlea(a, b));
39. }
```

- lignes 16-17 : on efface la précédente liste de réponses du serveur. Pour cela, ligne 17, le composant [ListView listReponses] est associé à une liste vide [reponses] ;
- lignes 19-20 : on affiche un compteur nul pour le nombre de réponses ;

- lignes 22-24 : on récupère les saisies des lignes [2-6] et on vérifie leur validité. Si l'une d'elles est invalide, la méthode est abandonnée (ligne 23) et pour l'utilisateur il y a retour à l'interface visuelle ;
- lignes 26-27 : si les données saisies sont toutes valides, alors on transmet à l'activité l'URL du service web (ligne 26) ainsi que le délai d'attente avant chaque appel au service (ligne 27). Ces informations sont nécessaires à la couche [DAO] et on rappelle que c'est l'activité qui communique avec celle-ci ;
- lignes 29-31 : les nombres aléatoires sont demandés un par un à la méthode [getAlea] de la ligne 37 ;
- ligne 36 : la méthode [getAlea] est annotée avec l'annotation AA [@Background], ce qui fait qu'elle va être exécutée dans un autre thread (flux d'exécution, process) que celui dans lequel s'exécute l'interface visuelle. Il est en effet obligatoire d'exécuter tout appel internet dans un thread différent de celui de l'interface visuelle. Ainsi, à un moment donné, on pourra avoir plusieurs threads :
  - celui qui affiche l'interface visuelle UI (User Interface) et gère ses événements,
  - les [nbAleas] threads qui chacun demandent un nombre aléatoire au service web. Ces threads sont lancés de façon asynchrone : le thread de l'UI lance un thread [getAlea] (ligne 37) qui demande un nombre aléatoire au service web et n'attend pas sa fin. Celle-ci lui sera signalée par un événement. Ainsi, les [nbAleas] threads vont être lancés en parallèle. Il est possible de configurer l'application pour qu'elle ne lance qu'un thread à la fois. Il y a alors une file d'attente des threads pour être exécutés ;

  Ligne 36, le paramètre [id] donne un nom au thread généré. Ici les [nbAleas] threads portent tous le même nom [alea]. Cela va nous permettre de les annuler tous en même temps. Ce paramètre est facultatif si on ne gère pas l'annulation du thread ;
- ligne 38 : la méthode [getAlea] de l'activité est appelée. Elle le sera donc dans un thread à part de celui de l'UI. Celui-ci fera l'appel au service web et n'attendra pas la réponse. Il sera prévenu plus tard par un événement que la réponse est disponible. C'est à ce moment que ligne 38, la méthode [showInfo] sera appelée avec comme paramètre la réponse reçue ;
- ligne 33 : on se met en attente des résultats :
  - un indicateur d'attente va être affiché,
  - le bouton [Annuler] va remplacer le bouton [Exécuter]. Parce que les threads lancés sont asynchrones, le thread de l'UI ne les attend pas et la ligne 33 est exécutée avant leur fin. Une fois la méthode [beginWaiting] terminée, l'UI peut de nouveau répondre aux sollicitations de l'utilisateur tel que le clic sur le bouton [Annuler]. Si les threads lancés avaient été synchrones, on arriverait à la ligne 33 qu'une fois tous les thread terminés. L'annulation de ceux-ci n'aurait alors plus de sens ;

La méthode [showInfo] est la suivante :

```
1.  @UiThread
2.    protected void showInfo(AleaResponse info) {
3.      // une info de plus
4.      nbInfos++;
5.      infoReponses.setText(String.format("Liste des réponses (%s)",nbInfos));
6.      // a-t-on terminé ?
7.      if (nbInfos == nbAleas) {
8.        // on termine l'attente
9.        cancelWaiting();
10.     }
11.     if (info.getErreur() == 0) {
12.       // on ajoute l'information à la liste des reponses
13.       reponses.add(0,String.valueOf(info.getAlea()));
14.     } else {
15.       // on ajoute un msg d'erreur à la liste des reponses
16.       StringBuffer message = new StringBuffer();
17.       for (String msg : info.getMessages()) {
18.         message.append(String.format("[%s]", msg));
19.       }
20.       reponses.add(message.toString());
21.     }
22.     // on affiche les reponses
23.     listReponses.setAdapter(new ArrayAdapter<String>(activité, android.R.layout.simple_list_item_1,
    android.R.id.text1, reponses));
24.   }
```

- la méthode [showInfo] est appelée à l'intérieur du thread [getAlea] annotée par [@Background]. Cette méthode va mettre à jour l'interface visuelle UI. Elle ne peut le faire qu'en étant exécutée à l'intérieur du thread de l'UI. C'est la signification de l'annotation [@UIThread] de la ligne 1 ;
- ligne 2 : la méthode reçoit un type [AleaResponse] (cf page 123) ;
- lignes 4-5 : on incrémente le compteur de réponses et on l'affiche ;
- lignes 7-10 : si on a reçu toutes les réponses attendues, alors on termine l'attente (fin du signal d'attente, le bouton [Exécuter] remplace le bouton [Annuler]) ;
- lignes 11-12 : s'il n'y a pas eu d'erreur, alors on ajoute le nombre aléatoire reçu à la liste des réponses affichée par le composant [ListView listReponses] (ligne 23) ;

- lignes 15-19 : s'il y a eu erreur, on concatène tous les messages d'erreur de l'objet [AleaResponse] en une unique chaîne de caractères qui est ajoutée à la liste des réponses affichée par le composant [ListView listReponses] (ligne 23) ;

Le 'clic' sur le bouton [Annuler] est géré avec la méthode suivante :

```
1.    @Click(R.id.btn_Annuler)
2.    protected void doAnnuler() {
3.      // on annule la tâche asynchrone
4.      BackgroundExecutor.cancelAll("alea", true);
5.      // fin de l'attente
6.      cancelWaiting();
7.  }
```

- ligne 4 : annule toutes les tâches identifiées par la chaîne [alea]. Le second paramètre [true] signifie qu'elles doivent être annulées même si elles ont déjà été lancées. L'identifiant [alea] est celui utilisé pour qualifier la méthode [getAlea] du fragment (ligne 1 ci-dessous) :

```
1.    @Background(id = "alea")
2.    void getAlea(int a, int b) {
3.      showInfo(activité.getAlea(a, b));
4.  }
```

La gestion de l'attente est gérée par les deux méthodes suivantes :

```
1.    // gestion de l'attente
2.    private void beginWaiting() {
3.      // le bouton [Annuler] remplace le bouton [Exécuter]
4.      btnExecuter.setVisibility(View.INVISIBLE);
5.      btnAnnuler.setVisibility(View.VISIBLE);
6.      // on met le sablier
7.      activité.setProgressBarIndeterminateVisibility(true);
8.    }
9.
10.   void cancelWaiting() {
11.     // le bouton [Exécuter] remplace le bouton [Annuler]
12.     btnAnnuler.setVisibility(View.INVISIBLE);
13.     btnExecuter.setVisibility(View.VISIBLE);
14.     // on enlève le sablier
15.     activité.setProgressBarIndeterminateVisibility(false);
16. }
```

La validité des saisies est gérée par la méthode [pageValid] suivante :

```
1.  // on vérifie la validité des données saisies
2.    private boolean isPageValid() {
3.      // au départ, pas d'erreurs
4.      Boolean erreur = false;
5.      int nbErreurs = 0;
6.      // saisie de l'URL du service web
7.      urlServiceWebJson = String.format("http://%s", edtUrlServiceRest.getText().toString().trim());
8.      // on vérifie sa validité
9.      txtMsgErreurUrlServiceWeb.setText("");
10.     URI service = null;
11.     try {
12.       service = new URI(urlServiceWebJson);
13.       erreur = service.getHost() == null && service.getPort() == -1;
14.     } catch (URISyntaxException ex) {
15.       erreur = true;
16.     }
17.     // erreur ?
18.     if (erreur) {
19.       // affichage msg d'erreur
20.       txtMsgErreurUrlServiceWeb.setText(getResources().getString(R.string.txt_MsgErreurUrlServiceWeb));
21.       // une erreur de +
22.       nbErreurs++;
23.     }
24.     // saisie du temps d'attente
25.     textViewErreurDelay.setText("");
26.     erreur = false;
27.     delay = 0;
28.     try {
29.       delay = Integer.parseInt(edtDelay.getText().toString());
```

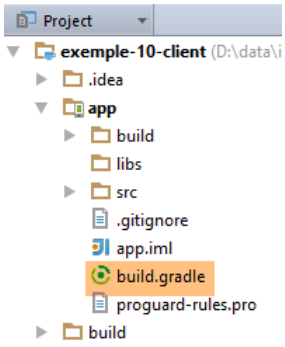
```
30.       erreur = (delay < 0);
31.     } catch (Exception ex) {
32.       erreur = true;
33.     }
34.     // erreur ?
35.     if (erreur) {
36.       nbErreurs++;
37.       textViewErreurDelay.setText(R.string.txt_ErreurDelay);
38.     }
39.     // saisie du nombre de nombres aléatoires
40.     erreur = false;
41.     nbAleas = 0;
42.     txtErrorAleas.setText("");
43.     try {
44.       nbAleas = Integer.parseInt(edtNbAleas.getText().toString());
45.       erreur = (nbAleas < 1);
46.     } catch (Exception ex) {
47.       erreur = true;
48.     }
49.     // erreur ?
50.     if (erreur) {
51.       nbErreurs++;
52.       txtErrorAleas.setText(R.string.txt_errorNbAleas);
53.     }
54.     // saisie de a
55.     txtErrorIntervalle.setText("");
56.     a = 0;
57.     erreur = false;
58.     try {
59.       a = Integer.parseInt(edtA.getText().toString());
60.     } catch (Exception ex) {
61.       erreur = true;
62.     }
63.     // erreur ?
64.     if (erreur) {
65.       nbErreurs++;
66.       txtErrorIntervalle.setText(R.string.txt_errorIntervalle);
67.     }
68.     // saisie de b
69.     b = 0;
70.     erreur = false;
71.     try {
72.       b = Integer.parseInt(edtB.getText().toString());
73.       erreur = b < a;
74.     } catch (Exception ex) {
75.       erreur = true;
76.     }
77.     // erreur ?
78.     if (erreur) {
79.       nbErreurs++;
80.       txtErrorIntervalle.setText(R.string.txt_errorIntervalle);
81.     }
82.     // retour
83.     return (nbErreurs == 0);
84.   }
```

1.11.2.8 Exécution du projet

Lorsqu'on essaie de compiler le projet [exemple-10-client], on a l'erreur suivante :

```
1.  Error:Gradle: duplicate files during packaging of APK D:\data\istia-1415\android\dvp\exemples\exemple-10-
    client\app\build\outputs\apk\app-debug-unaligned.apk
2.  Error:Gradle: Execution failed for task ':app:packageDebug'.
3.  > Duplicate files copied in APK META-INF/ASL2.0
4.      File 1: C:\Users\Serge Tahé\.gradle\caches\modules-2\files-2.1\org.codehaus.jackson\jackson-mapper-
    asl\1.9.9\174678f285c155bea65c76ae9ac302f63b4aece1\jackson-mapper-asl-1.9.9.jar
5.      File 2: C:\Users\Serge Tahé\.gradle\caches\modules-2\files-2.1\org.codehaus.jackson\jackson-mapper-
    asl\1.9.9\174678f285c155bea65c76ae9ac302f63b4aece1\jackson-mapper-asl-1.9.9.jar
```

On trouve la solution à ce problème sur StackOverflow [http://stackoverflow.com/questions/20673625/android-gradle-plugin-0-7-0-duplicate-files-during-packaging-of-apk]. Il faut modifier le fichier [build.gradle] du projet de la façon suivante :



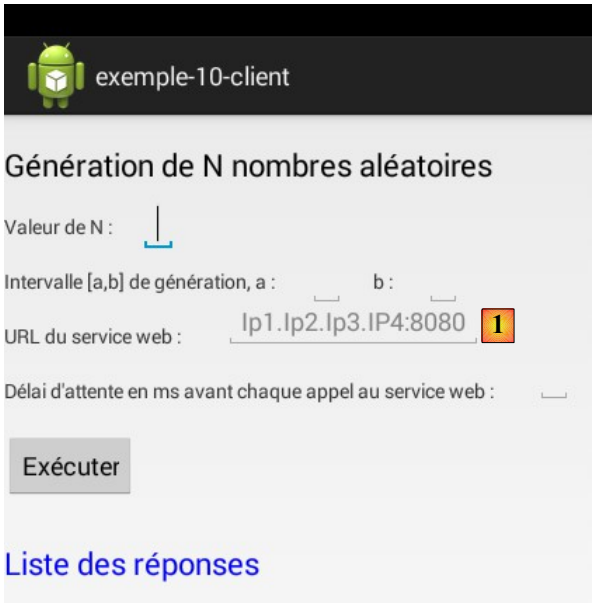

```
1.  android {
2.    compileSdkVersion 20
3.    buildToolsVersion "20.0.0"
4.
5.    packagingOptions {
6.      exclude 'META-INF/ASL2.0'
7.    }
8.
9.    defaultConfig {
10.     minSdkVersion 11
11.     targetSdkVersion 20
12.     versionCode 1
13.     versionName "1.0"
14.   }
15.   buildTypes {
16.     release {
17.       runProguard false
18.       proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
19.     }
20.   }
```

La modification nécessaire est faite aux lignes 5-7. Elle exclut du processus de compilation le fichier [META-INF/ASL2.0]. Lorsqu'on reconstruit le projet, on a exactement le même type d'erreur mais sur un autre fichier cette fois qu'on exclut à son tour. On répète l'opération jusqu'à ne plus avoir d'erreurs. Pour ma part, je suis arrivé aux exclusions suivantes :

```
packagingOptions {
  exclude 'META-INF/ASL2.0'
  exclude 'META-INF/NOTICE'
  exclude 'META-INF/LICENSE'
  exclude 'META-INF/notice.txt'
  exclude 'META-INF/license.txt'
}
```

Une fois le projet construit, on peut l'exécuter :

1. lancer le serveur [exemple-10-server] ;
2. lancer le client [exemple-10-client] ;



Pour savoir quoi mettre en [1], procéder comme suit. Ouvrez une fenêtre [DOS] et tapez la commande suivante :

```
1.  dos>ipconfig
2.
3.  Configuration IP de Windows
4.
5.  ....
6.
7.  Carte Ethernet Connexion au réseau local :
8.
9.     Suffixe DNS propre à la connexion. . . : ad.univ-angers.fr
10.    Adresse IPv6 de liaison locale. . . . .: fe80::698b:455a:925:6b13%4
11.    Adresse IPv4. . . . . .: 172.19.81.34
12.    Masque de sous-réseau. . . . . : 255.255.0.0
13.    Passerelle par défaut. . . . . : 172.19.0.254
14.
15. Carte réseau sans fil Wi-Fi :
16.
17.    Statut du média. . . . . : Média déconnecté
18.    Suffixe DNS propre à la connexion. . . :
19. ...
```

La ligne 11 donne l'adresse IP de votre poste. Utilisez-la si vous utilisez l'émulateur pour tester l'application. Si vous êtes connecté à un réseau wifi, utilisez l'adresse wifi. Si vous utilisez la tablette pour tester l'application, utilisez l'adresse wifi du PC. Dans ce cas, inhibez le pare-feu de votre PC (s'il y en a un) qui par défaut empêche toute connexion venant de l'extérieur (donc de la tablette).

Testez l'application dans les cas suivants :

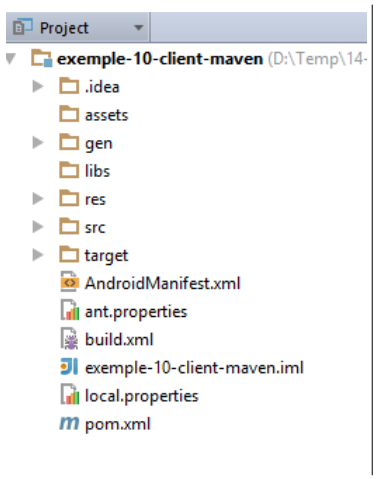
- 100 nombres aléatoires dans l'intervalle [1000, 2000] sans délai d'attente ;
- 2000 nombres aléatoires dans l'intervalle [10000, 20000] sans délai d'attente et annulez l'attente avant la fin de la génération ;
- 5 nombres aléatoires dans l'intervalle [100,200] avec un délai d'attente de 5000 ms et annulez l'attente avant la fin de la génération ;

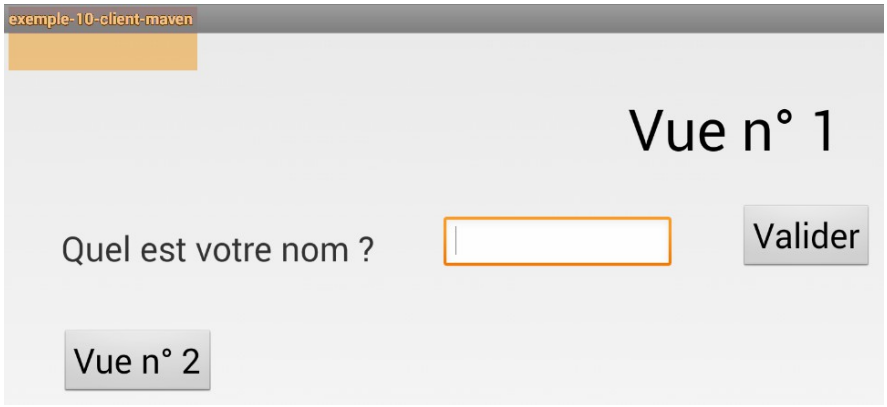
1.11.3 Conversion du projet [Gradle] en projet [Maven]

On se propose ici, de convertir le projet précédent configuré avec [Gradle] en un projet Maven. Tout d'abord, nous partons du dernier projet Maven que nous avons créé. C'est [exemple-05].

Nous

- dupliquons le module [exemple-05] dans [exemple-10-client-maven] (après avoir supprimé le dossier [target] de [exemple-05]) ;
- chargeons le module [exemple-10-client-maven] ;
- changeons le nom du projet et du module en [exemple-10-client-maven] (structure du projet) ;
- changeons le nom du projet dans les fichiers [pom.xml, strings.xml] ;
- rafraîchissons le projet Maven (View / Tool windows / Maven) ;
- le compilons ;
- créons une configuration d'exécution nommée [exemple-10-client-maven] ;
- exécutons celle-ci ;





Dans le projet [exemple-10-client], nous avions les dépendances Gradle suivantes :

```
1.  dependencies {
2.    apt "org.androidannotations:androidannotations:$AAVersion"
3.    compile "org.androidannotations:androidannotations-api:$AAVersion"
4.    compile 'com.android.support:appcompat-v7:20.+'
5.    compile fileTree(dir: 'libs', include: ['*.jar'])
6.    compile 'org.springframework.android:spring-android-rest-template:1.0.1.RELEASE'
7.    compile 'org.codehaus.jackson:jackson-mapper-asl:1.9.9'
8.  }
```

Il nous faut reproduire ces dépendances dans le fichier [pom.xml] :

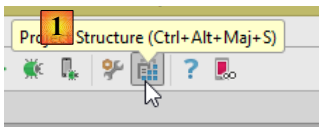
```
1.    <dependencies>
2.      <!-- Android -->
3.      <dependency>
4.        <groupId>com.google.android</groupId>
5.        <artifactId>android</artifactId>
6.        <version>4.1.1.4</version>
7.        <scope>provided</scope>
8.      </dependency>
9.      <!-- Android annotations-->
10.     <dependency>
11.       <groupId>org.androidannotations</groupId>
12.       <artifactId>androidannotations</artifactId>
13.       <version>3.1</version>
14.     </dependency>
15.     <dependency>
16.       <groupId>org.androidannotations</groupId>
17.       <artifactId>androidannotations-api</artifactId>
18.       <version>3.1</version>
19.     </dependency>
20.     <!-- Spring RestTemplate-->
```

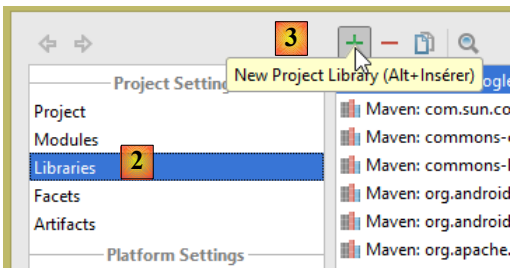
```
21.     <dependency>
22.       <groupId>org.springframework.android</groupId>
23.       <artifactId>spring-android-rest-template</artifactId>
24.       <version>1.0.1.RELEASE</version>
25.     </dependency>
26.     <!-- bibliothèque Jackson -->
27.     <dependency>
28.       <groupId>org.codehaus.jackson</groupId>
29.       <artifactId>jackson-mapper-asl</artifactId>
30.       <version>1.9.9</version>
31.     </dependency>
32. </dependencies>
```

Une dépendance n'a pas été reproduite dans le fichier [pom.xml] :

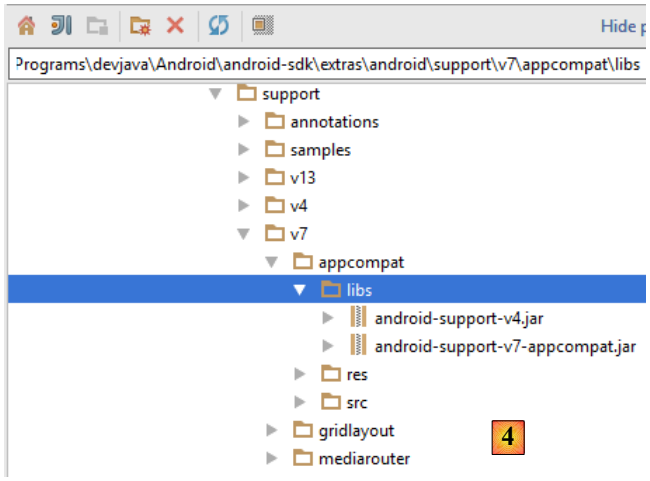
```
compile 'com.android.support:appcompat-v7:20.+'
```

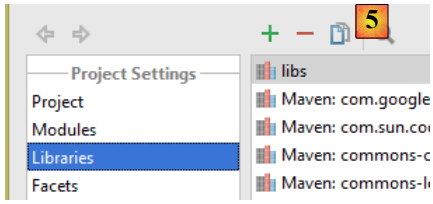
Cette dépendance n'existe pas dans le dépôt central de Maven. Les archives correspondantes sont dans l'arborescence du SDK d'Android. Il y a plusieurs façons de résoudre ce problème. Nous en montrons une :



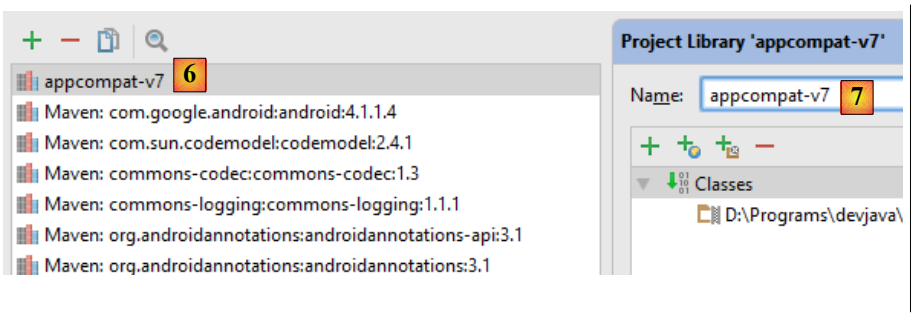


- en [1], nous accédons à la structure du projet ;
- et en [2-3], nous lui ajoutons une bibliothèque ;



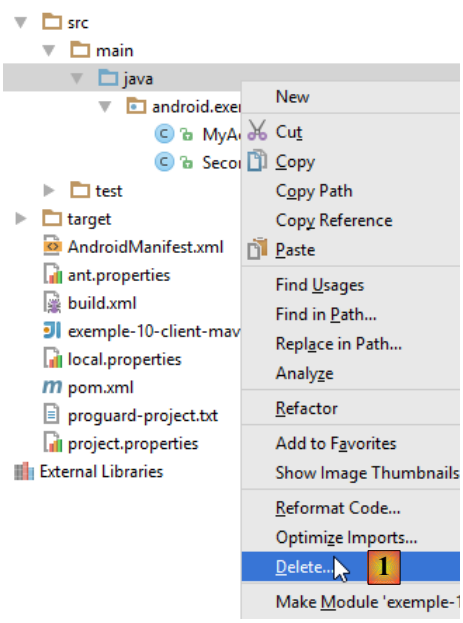


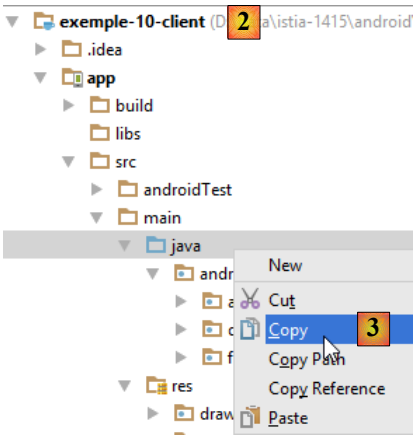
- en [4], nous sélectionnons le dossier [<sdk>\extras\android\support\v7\appcompat\libs] où <sdk> est le dossier d'installation du SDK d'Android (cf paragraphe 1.16.3, page 179). Ce dossier contient deux archives ;
- en [5], la nouvelle bibliothèque va apparaître ;



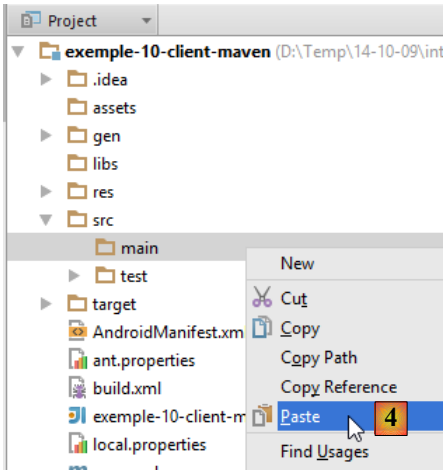
- en [6-7], on lui donne un nom explicite ;

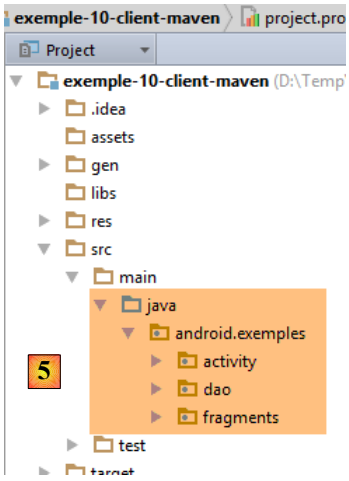
Nous allons maintenant récupérer les sources du projet [exemple-10-client] par un (copy / paste) entre les deux projets :





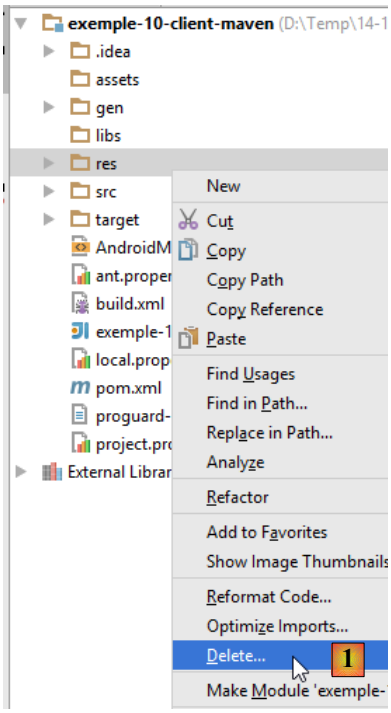
- en [1], dans le projet [exemple-10-client-maven], on supprime le dossier [java] ;
- en [2-3], dans le projet [exemple-10-client], on copie le dossier [java] ;

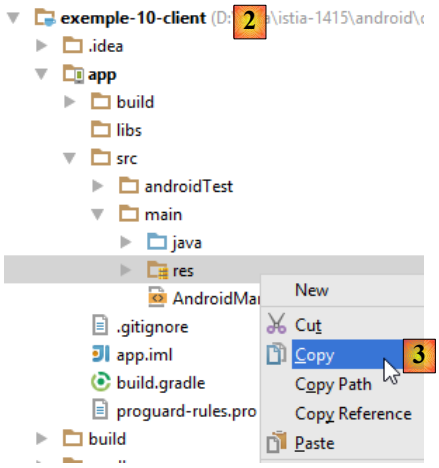




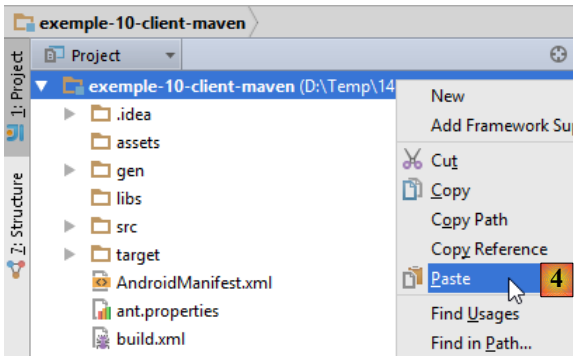
- en [4-5], on colle le nouveau dossier [java] ;

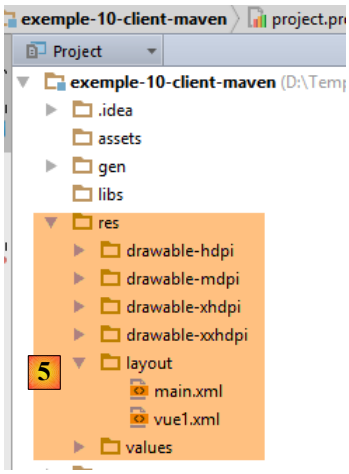
On recommence une opération similaire pour le dossier [res] :



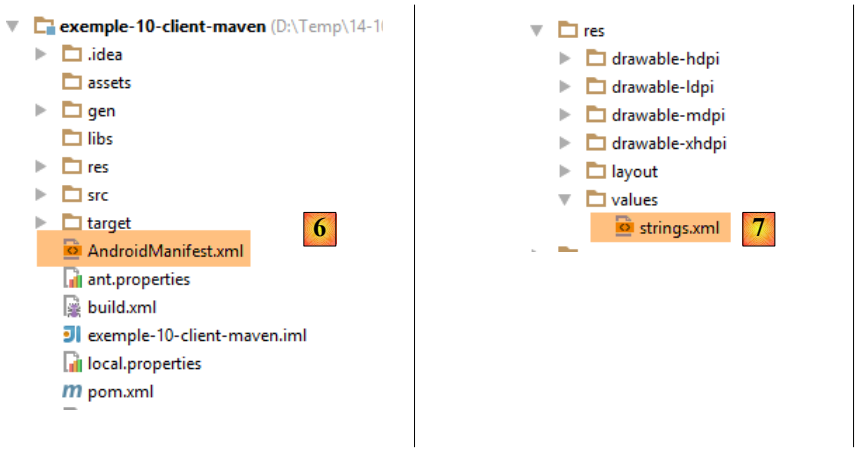


- en [1], dans le projet [exemple-10-client-maven], on supprime le dossier [res] ;
- en [2-3], dans le projet [exemple-10-client], on copie le dossier [res] ;





- en [4-5], on colle le nouveau dossier [res] ;

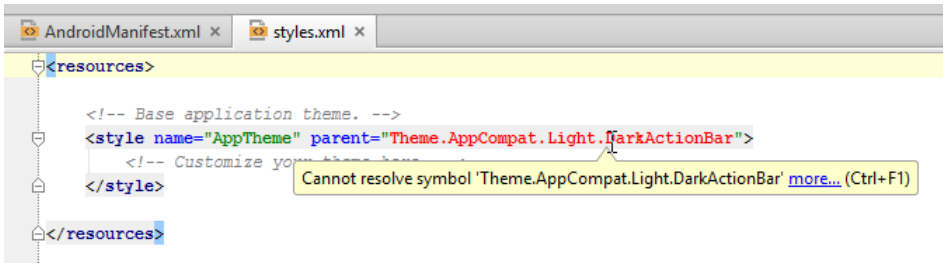


- en [6], nous copions le fichier [AndroidManifes.xml] du projet [exemple-10-client] dans [exemple-10-client-maven].
- en [7], nous modifions le fichier [res / strings.xml] ;

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.
4.    <string name="app_name">exemple-10-client-maven</string>
5.    <string name="titre_vue1">Vue n° 1</string>
6.    ....
7.  </resources>
```

- ligne 4 : nous modifions le nom qui sera affiché dans la fenêtre de l'application ;

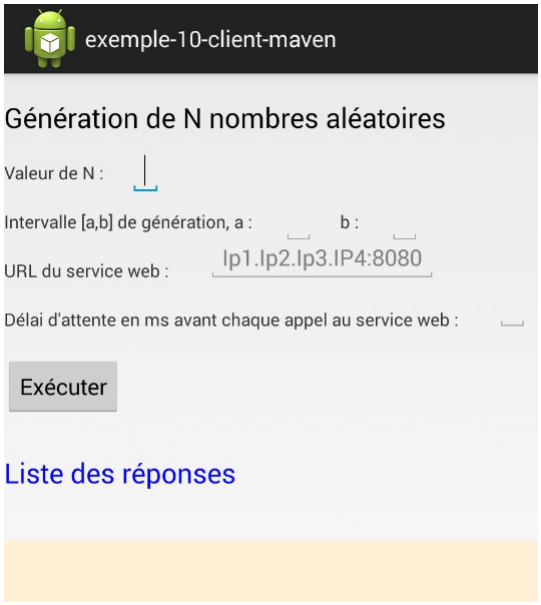
A ce stade, il peut être bon de rafraîchir le projet Maven (View / Tools windows / Maven) voire de fermer le projet puis de le réouvrir car l'IDE Intellij peut parfois être 'perdu' par ce jeu. Ensuite nous pouvons tenter la construction de l'application. Nous rencontrons alors l'erreur suivante dans le fichier [res / values/ styles.xml] (peut dépendre de la version d'IntellijIDEA :



Modifiez alors le fichier de la façon suivante :

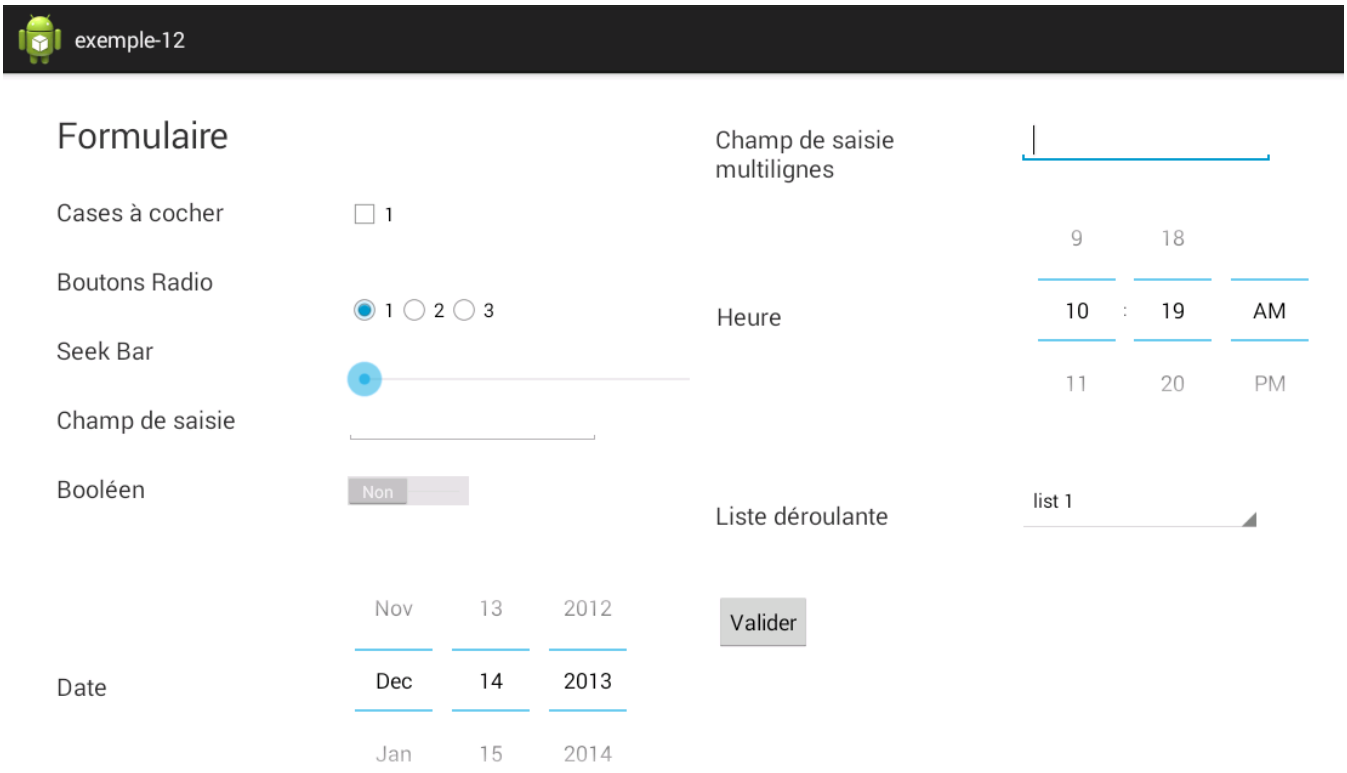
```
1.  <resources>
2.
3.      <!-- Base application theme. -->
4.      <style name="AppTheme" parent="android:Theme.Holo.Light.DarkActionBar">
5.          <!-- Customize your theme here. -->
6.      </style>
7.
8.  </resources>
```

puis reconstruisez l'application. Normalement, il n'y a plus d'erreurs. Exécutez alors l'application :


exemple-10-client-maven
Génération de N nombres aléatoires
Valeur de N :
Intervalle [a,b] de génération, a :
b :
URL du service web :
lp1.lp2.lp3.IP4:8080
Délai d'attente en ms avant chaque appel au service web :
Exécuter
Liste des réponses

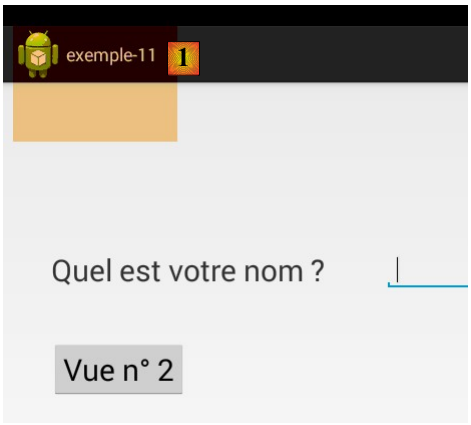
1.12 Exemple-11 : composants de saisie de données

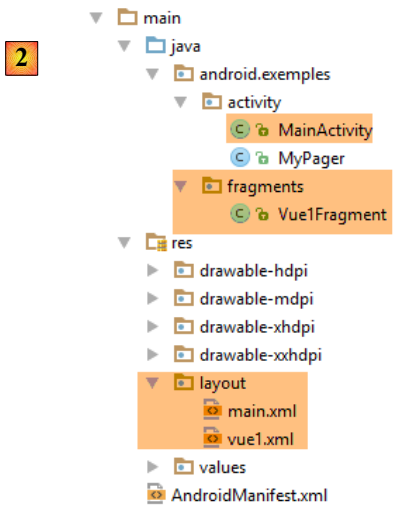
Nous allons écrire un nouveau projet pour présenter quelques composants usuels dans les formulaires de saisie de données :



1.12.1 Création du projet

Nous créons un nouveau projet [exemple-11] par recopie du projet [exemple-08]. Pour cela suivez la démarche indiquée au paragraphe 1.10.1, page 91 qui décrit la création du projet [exemple-09] à partir du projet [exemple-08]. A la fin de la démarche, lorsque vous exécutez le projet [exemple-11] vous devez obtenir le résultat suivant [1] :



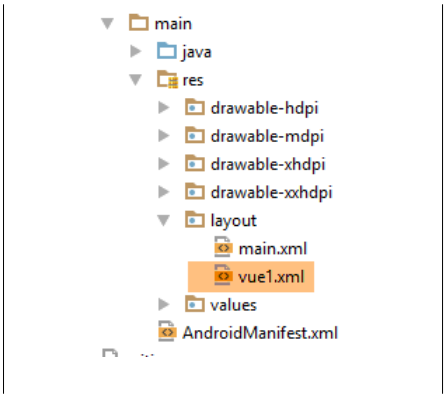


Le nouveau projet n'aura qu'une vue [formulaire.xml]. Aussi supprimons-nous la vue [vue2.xml] et son fragment associé [Vue2Fragment] [2]. Nous prenons en compte cette modification dans le gestionnaire de fragments de [Mainactivity] :

```
    // notre gestionnaire de fragments à redéfinir pour chaque application
    // doit définir les méthodes suivantes : getItem, getCount, getPageTitle
    public class SectionsPagerAdapter extends FragmentPagerAdapter {

      // les fragments
      private final Fragment[] fragments = {new Vue1Fragment_()};
      private final String[] titres = {getString(R.string.titre_vue1)};
  ....
  }
```

Réexécutez le projet. Il doit faire apparaître le vue n° 1 comme précédemment. Nous allons travailler à partir de ce projet.

1.12.2 La vue XML du formulaire



La vue XML du formulaire est dans [vue1.xml] :

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="match_parent"
            android:layout_height="match_parent" >

  <RelativeLayout
    android:layout_width="match_parent" android:layout_height="wrap_content" >

    <TextView
      android:id="@+id/textViewFormulaireTitre"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:layout_marginLeft="50dp"
      android:layout_marginTop="30dp"
      android:text="@string/titre_vue1"
      android:textSize="30sp" />

    <Button
      android:id="@+id/formulaireButtonValider"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignLeft="@+id/TextViewFormulaireCombo"
      android:layout_below="@+id/TextViewFormulaireCombo"
      android:layout_marginTop="50dp"
      android:text="@string/formulaire_valider" />

    <TextView
      android:id="@+id/textViewFormulaireCheckBox"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignLeft="@+id/textViewFormulaireTitre"
      android:layout_below="@+id/textViewFormulaireTitre"
      android:layout_marginTop="30dp"
      android:text="@string/formulaire_checkbox"
      android:textSize="20sp" />
```

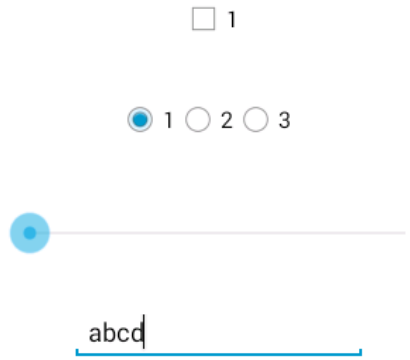
```
38.
39.     <TextView
40.       android:id="@+id/textViewFormulaireRadioButton"
41.       android:layout_width="wrap_content"
42.       android:layout_height="wrap_content"
43.       android:layout_alignLeft="@+id/textViewFormulaireCheckBox"
44.       android:layout_below="@+id/textViewFormulaireCheckBox"
45.       android:layout_marginTop="30dp"
46.       android:text="@string/formulaire_radioButton"
47.       android:textSize="20sp" />
48.
49.     <TextView
50.       android:id="@+id/textViewFormulaireSeekBar"
51.       android:layout_width="wrap_content"
52.       android:layout_height="wrap_content"
53.       android:layout_alignLeft="@+id/textViewFormulaireRadioButton"
54.       android:layout_below="@+id/textViewFormulaireRadioButton"
55.       android:layout_marginTop="30dp"
56.       android:text="@string/formulaire_seekBar"
57.       android:textSize="20sp" />
58.
59.     <TextView
60.       android:id="@+id/textViewFormulaireEdtText"
61.       android:layout_width="wrap_content"
62.       android:layout_height="wrap_content"
63.       android:layout_alignLeft="@+id/textViewFormulaireSeekBar"
64.       android:layout_below="@+id/textViewFormulaireSeekBar"
65.       android:layout_marginTop="30dp"
66.       android:text="@string/formulaire_saisie"
67.       android:textSize="20sp" />
68.
69.     <TextView
70.       android:id="@+id/textViewFormulaireBool"
71.       android:layout_width="wrap_content"
72.       android:layout_height="wrap_content"
73.       android:layout_alignLeft="@+id/textViewFormulaireEdtText"
74.       android:layout_below="@+id/textViewFormulaireEdtText"
75.       android:layout_marginTop="30dp"
76.       android:text="@string/formulaire_bool"
77.       android:textSize="20sp" />
78.
79.     <TextView
80.       android:id="@+id/textViewFormulaireDate"
81.       android:layout_width="wrap_content"
82.       android:layout_height="200dp"
83.       android:layout_alignLeft="@+id/textViewFormulaireBool"
84.       android:layout_below="@+id/textViewFormulaireBool"
85.       android:layout_marginTop="50dp"
86.       android:gravity="center"
87.       android:text="@string/formulaire_date"
88.       android:textSize="20sp" />
89.
90.     <TextView
91.       android:id="@+id/textViewFormulaireMultilignes"
92.       android:layout_width="150dp"
93.       android:layout_height="100dp"
94.       android:gravity="center"
95.       android:layout_alignBaseline="@+id/textViewFormulaireTitre"
96.       android:layout_alignParentTop="true"
97.       android:layout_marginLeft="500dp"
98.       android:layout_toRightOf="@+id/textViewFormulaireTitre"
99.       android:text="@string/formulaire_multilignes"
100.      android:textSize="20sp" />
101.
102.    <TextView
103.      android:id="@+id/textViewFormulaireTime"
104.      android:layout_width="wrap_content"
105.      android:layout_height="200dp"
106.      android:gravity="center"
107.      android:layout_alignLeft="@+id/textViewFormulaireMultilignes"
108.      android:layout_below="@+id/textViewFormulaireMultilignes"
109.      android:layout_marginTop="50dp"
110.      android:text="@string/formulaire_time"
111.      android:textSize="20sp" />
112.
```

```
113.    <TextView
114.      android:id="@+id/TextViewFormulaireCombo"
115.      android:layout_width="wrap_content"
116.      android:layout_height="wrap_content"
117.      android:layout_alignLeft="@+id/textViewFormulaireTime"
118.      android:layout_below="@+id/textViewFormulaireTime"
119.      android:layout_marginTop="50dp"
120.      android:text="@string/formulaire_combo"
121.      android:textSize="20sp" />
122.
123.    <CheckBox
124.      android:id="@+id/formulaireCheckBox1"
125.      android:layout_width="wrap_content"
126.      android:layout_height="wrap_content"
127.      android:layout_alignBaseline="@+id/textViewFormulaireCheckBox"
128.      android:layout_marginLeft="100dp"
129.      android:layout_toRightOf="@+id/textViewFormulaireCheckBox"
130.      android:text="@string/formulaire_checkbox1" />
131.
132.    <RadioGroup
133.      android:id="@+id/formulaireRadioGroup"
134.      android:layout_width="wrap_content"
135.      android:layout_height="wrap_content"
136.      android:layout_alignBaseline="@+id/textViewFormulaireRadioButton"
137.      android:layout_alignLeft="@+id/formulaireCheckBox1"
138.      android:orientation="horizontal" >
139.
140.      <RadioButton
141.        android:id="@+id/formulaireRadioButton1"
142.        android:layout_width="wrap_content"
143.        android:layout_height="wrap_content"
144.        android:text="@string/formulaire_radiobutton1" />
145.
146.      <RadioButton
147.        android:id="@+id/formulaireRadioButton2"
148.        android:layout_width="wrap_content"
149.        android:layout_height="wrap_content"
150.        android:text="@string/formulaire_radionbutton2" />
151.
152.      <RadioButton
153.        android:id="@+id/formulaireRadionButton3"
154.        android:layout_width="wrap_content"
155.        android:layout_height="wrap_content"
156.        android:text="@string/formulaire_radiobutton3" />
157.    </RadioGroup>
158.
159.    <SeekBar
160.      android:id="@+id/formulaireSeekBar"
161.      android:layout_width="300dp"
162.      android:layout_height="wrap_content"
163.      android:layout_alignBaseline="@+id/textViewFormulaireSeekBar"
164.      android:layout_alignLeft="@+id/formulaireCheckBox1" />
165.
166.    <EditText
167.      android:id="@+id/formulaireEditText1"
168.      android:layout_width="wrap_content"
169.      android:layout_height="wrap_content"
170.      android:layout_alignBaseline="@+id/textViewFormulaireEdtText"
171.      android:layout_alignLeft="@+id/formulaireCheckBox1"
172.      android:ems="10"
173.      android:inputType="text" >
174.    </EditText>
175.
176.    <Switch
177.      android:id="@+id/formulaireSwitch1"
178.      android:layout_width="wrap_content"
179.      android:layout_height="wrap_content"
180.      android:layout_alignBaseline="@+id/textViewFormulaireBool"
181.      android:layout_alignLeft="@+id/formulaireCheckBox1"
182.      android:text="@string/formulaire_switch"
183.      android:textOff="Non"
184.      android:textOn="Oui" />
185.
186.    <TimePicker
187.      android:id="@+id/formulaireTimePicker1"
```

```
188.      android:layout_width="wrap_content"
189.      android:layout_height="wrap_content"
190.      android:layout_alignBottom="@+id/textViewFormulaireTime"
191.      android:layout_alignLeft="@+id/formulaireEditTextMultiLignes" />
192.
193.    <EditText
194.      android:id="@+id/formulaireEditTextMultiLignes"
195.      android:layout_width="wrap_content"
196.      android:layout_height="100dp"
197.      android:layout_alignBaseline="@+id/textViewFormulaireMultilignes"
198.      android:layout_alignBottom="@+id/textViewFormulaireMultilignes"
199.      android:layout_marginLeft="100dp"
200.      android:layout_toRightOf="@+id/textViewFormulaireMultiLignes"
201.      android:ems="10"
202.      android:inputType="textMultiLine" >
203.    </EditText>
204.
205.    <Spinner
206.      android:id="@+id/formulaireDropDownList"
207.      android:layout_width="200dp"
208.      android:layout_height="50dp"
209.      android:layout_alignBottom="@+id/TextViewFormulaireCombo"
210.      android:layout_alignLeft="@+id/formulaireEditTextMultiLignes" >
211.    </Spinner>
212.
213.    <DatePicker
214.      android:id="@+id/formulaireDatePicker1"
215.      android:layout_width="wrap_content"
216.      android:layout_height="wrap_content"
217.      android:layout_alignBottom="@+id/textViewFormulaireDate"
218.      android:layout_alignLeft="@+id/formulaireCheckBox1"
219.      android:calendarViewShown="false">
220.    </DatePicker>
221.
222.    <TextView
223.      android:id="@+id/textViewSeekBarValue"
224.      android:layout_width="30dp"
225.      android:layout_height="wrap_content"
226.      android:layout_alignBaseline="@+id/textViewFormulaireSeekBar"
227.      android:layout_marginLeft="30dp"
228.      android:layout_toRightOf="@+id/formulaireSeekBar"
229.      android:text="" />
230.  </RelativeLayout>
231.
232.</ScrollView>
```

Les principaux composants du formulaire sont les suivants :

- ligne 2 : un layout [ScrollView] vertical. Il permet de présenter un formulaire plus grand que l'écran de la tablette. On obtient la totalité du formulaire par défilement ;
- lignes 123-130 : une case à cocher
- lignes 132-157 : un groupe de trois boutons radio
- lignes 159-164 : une barre de recherche
- lignes 166-174 : une boîte de saisie



- lignes 176-184 : un switch oui / non

- lignes 186-191 : une boîte de saisie de l'heure

- lignes 193-203 : une boîte de saisie multi-lignes

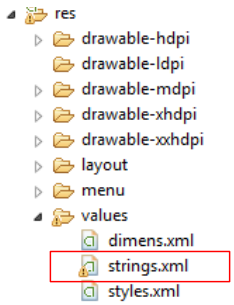
- lignes 205-211 : une liste déroulante

- lignes 213-219 : une boîte de saisie d'une date

- tous les autres composants sont des [TextView] qui affichent des textes.

1.12.3 Les chaînes de caractères du formulaire

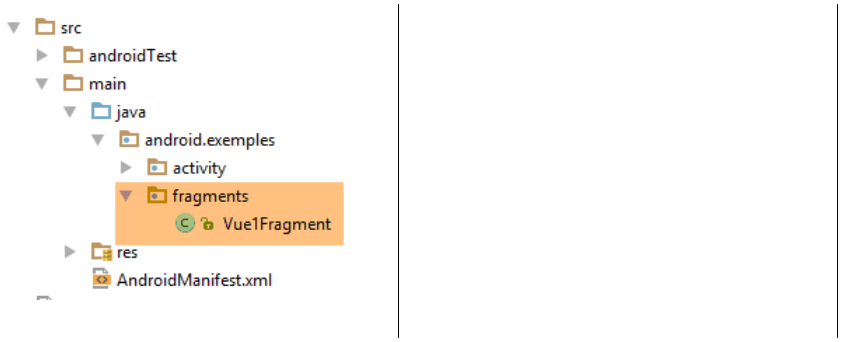
Les chaînes de caractères du formulaire sont définies dans le fichier [res / values / strings.xml] suivant :




```
1.  <?xml version="1.0" encoding="utf-8"?>
```

```
<resources>

    <string name="app_name">exemple-11</string>
    <string name="action_settings">Settings</string>
    <string name="titre_vue1">Vue n° 1</string>
    <string name="formulaire_checkbox">Cases à cocher</string>
    <string name="formulaire_radioButton">Boutons Radio</string>
    <string name="formulaire_seekBar">Seek Bar</string>
    <string name="formulaire_saisie">Champ de saisie</string>
    <string name="formulaire_bool">Booléen</string>
    <string name="formulaire_date">Date</string>
    <string name="formulaire_time">Heure</string>
    <string name="formulaire_multilignes">Champ de saisie multilignes</string>
    <string name="formulaire_listview">Liste</string>
    <string name="formulaire_combo">Liste déroulante</string>
    <string name="formulaire_checkbox1">1</string>
    <string name="formulaire_checkbox2">2</string>
    <string name="formulaire_radiobutton1">1</string>
    <string name="formulaire_radionbutton2">2</string>
    <string name="formulaire_radiobutton3">3</string>
    <string name="formulaire_switch"></string>
    <string name="formulaire_valider">Valider</string>

</resources>
```

1.12.4 Le fragment du formulaire



La classe [Vue1Fragment] est la suivante :

```
package android.exemples.fragments;

import android.annotation.SuppressLint;
import android.app.AlertDialog;
import android.exemples.R;
import android.exemples.activity.MainActivity;
import android.support.v4.app.Fragment;
import android.widget.*;
import android.widget.SeekBar.OnSeekBarChangeListener;
import org.androidannotations.annotations.AfterViews;
import org.androidannotations.annotations.Click;
import org.androidannotations.annotations.EFragment;
import org.androidannotations.annotations.ViewById;

import java.util.ArrayList;
import java.util.List;

// un fragment est une vue affichée par un conteneur de fragments
@EFragment(R.layout.vue1)
public class Vue1Fragment extends Fragment {

  // les champs de la vue affichée par le fragment
  @ViewById(R.id.formulaireDropDownList)
  Spinner dropDownList;
  @ViewById(R.id.formulaireButtonValider)
  Button buttonValider;
  @ViewById(R.id.formulaireCheckBox1)
```

```
28.   CheckBox checkBox1;
29.   @ViewById(R.id.formulaireRadioGroup)
30.   RadioGroup radioGroup;
31.   @ViewById(R.id.formulaireSeekBar)
32.   SeekBar seekBar;
33.   @ViewById(R.id.formulaireEditText1)
34.   EditText saisie;
35.   @ViewById(R.id.formulaireSwitch1)
36.   Switch switch1;
37.   @ViewById(R.id.formulaireDatePicker1)
38.   DatePicker datePicker1;
39.   @ViewById(R.id.formulaireTimePicker1)
40.   TimePicker timePicker1;
41.   @ViewById(R.id.formulaireEditTextMultiLignes)
42.   EditText multiLignes;
43.   @ViewById(R.id.formulaireRadioButton1)
44.   RadioButton radioButton1;
45.   @ViewById(R.id.formulaireRadioButton2)
46.   RadioButton radioButton2;
47.   @ViewById(R.id.formulaireRadionButton3)
48.   RadioButton radioButton3;
49.   @ViewById(R.id.textViewSeekBarValue)
50.   TextView seekBarValue;
51.
52.   // l'activité
53.   private MainActivity activité;
54.
55.   @AfterViews
56.   void initFragment() {
57.     // on récupère l'unique activité
58.     activité = (MainActivity) getActivity();
59.
60.     // on coche le premier bouton
61.     radioButton1.setChecked(true);
62.     // le calendrier
63.     datePicker1.setCalendarViewShown(false);
64.     // le seekBar
65.     seekBar.setMax(100);
66.     seekBar.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {
67.
68.       public void onStopTrackingTouch(SeekBar seekBar) {
69.       }
70.
71.       public void onStartTrackingTouch(SeekBar seekBar) {
72.       }
73.
74.       public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
75.         seekBarValue.setText(String.valueOf(progress));
76.       }
77.     });
78.     // la liste déroulante
79.     List<String> list = new ArrayList<String>();
80.     list.add("list 1");
81.     list.add("list 2");
82.     list.add("list 3");
83.     ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(activité,
    android.R.layout.simple_spinner_item, list);
84.     dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
85.     dropDownList.setAdapter(dataAdapter);
86.   }
87.
88.
89.   @SuppressLint("DefaultLocale")
90.   @Click(R.id.formulaireButtonValider)
91.   protected void doValider() {
92.     ...
93.   }
94.
95.   private void doAfficher(List<String> messages) {
96. ...
97.
98. }
```

- lignes 23-50 : on récupère les références de tous les composants du formulaire XML [vue1] (ligne 19) ;

- ligne 61 : la méthode [setChecked] permet de cocher un bouton radio ou une case à cocher ;
- ligne 65 : *[SeekBar].setMax()* permet de fixer la valeur maximale de la barre de réglage. La valeur minimale est 0 ;
- lignes 66-77 : on gère les événements de la barre de réglage. On veut, à chaque changement opéré par l'utilisateur, afficher la valeur de la règle dans le [TextView] de la ligne 50 ;
- ligne 74 : le paramètre [progress] représente la valeur de la règle ;
- ligne 63 : par défaut le composant [DatePicker] affiche et une boîte de saisie de la date et un calendrier. La ligne 63 élimine le calendrier ;
- lignes 79-82 : une liste de [String] qu'on va associer à une liste déroulante ;
- lignes 83-85 : cette liste est associée à la liste déroulante ;
- lignes 90-91 : on associe la méthode [doValider] au clic sur le bouton [Valider] ;

La méthode [doValider] a pour but d'afficher les valeurs saisies par l'utilisateur. Son code est le suivant :

```
1.  @Click(R.id.formulaireButtonValider)
2.  protected void doValider() {
3.    // liste des messages à afficher
4.    List<String> messages = new ArrayList<String>();
5.    // case à cocher
6.    boolean isChecked = checkBox1.isChecked();
7.    messages.add(String.format("CheckBox1 [checked=%s]", isChecked));
8.    // les boutons radio
9.    int id = radioGroup.getCheckedRadioButtonId();
10.   String radioGroupText = id == -1 ? "" : ((RadioButton) activité.findViewById(id)).getText().toString();
11.   messages.add(String.format("RadioGroup [checked=%s]", radioGroupText));
12.   // le SeekBar
13.   int progress = seekBar.getProgress();
14.   messages.add(String.format("SeekBar [value=%d]", progress));
15.   // le champ de saisie
16.   String texte = String.valueOf(saisie.getText());
17.   messages.add(String.format("Saisie simple [value=%s]", texte));
18.   // le switch
19.   boolean état = switch1.isChecked();
20.   messages.add(String.format("Switch [value=%s]", état));
21.   // la date
22.   int an = datePicker1.getYear();
23.   int mois = datePicker1.getMonth() + 1;
24.   int jour = datePicker1.getDayOfMonth();
25.   messages.add(String.format("Date [%d, %d, %d]", jour, mois, an));
26.   // le texte multi-lignes
27.   String lignes = String.valueOf(multiLignes.getText());
28.   messages.add(String.format("Saisie multi-lignes [value=%s]", lignes));
29.   // l'heure
30.   int heure = timePicker1.getCurrentHour();
31.   int minutes = timePicker1.getCurrentMinute();
32.   messages.add(String.format("Heure [%d, %d]", heure, minutes));
33.   // liste déroulante
34.   int position = dropDownList.getSelectedItemPosition();
35.   String selectedItem = String.valueOf(dropDownList.getSelectedItem());
36.   messages.add(String.format("DropDownList [position=%d, item=%s]", position, selectedItem));
37.   // affichage
38.   doAfficher(messages);
39. }
```

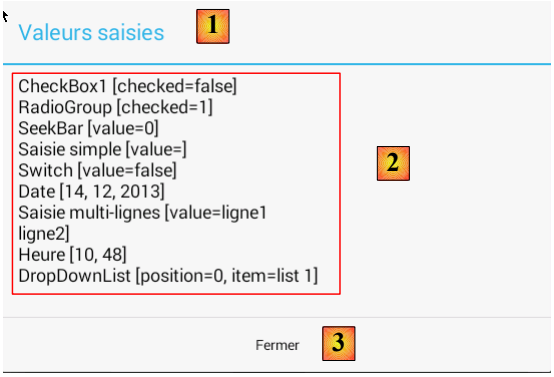
- ligne 4 : les valeurs saisies vont être cumulées dans une liste de messages ;
- ligne 6 : la méthode *[CheckBox].isCkecked()* permet de savoir si une case est cochée ou non ;
- ligne 9 : la méthode *[RadioGroup].getCheckedButtonId()* permet d'obtenir l'id du bouton radio qui a été coché ou -1 si aucun n'a été coché ;
- ligne 10 : le code [activité.*findViewById(id)*] permet de retrouver le bouton radio coché et d'avoir ainsi son libellé ;
- ligne 13 : la méthode *[SeekBar].getProgress()]* permet d'avoir la valeur d'une barre de réglage ;
- ligne 19 : la méthode *[Switch].isChecked()* permet de savoir si un switch est *On* (true) ou *Off* (false) ;
- ligne 22 : la méthode *[DatePicker].getYear()* permet d'avoir l'année choisie avec un objet [DatePicker] ;
- ligne 23 : la méthode *[DatePicker].getMonth()* permet d'avoir le mois choisi avec un objet [DatePicker] dans l'intervalle [0,11] ;
- ligne 24 : la méthode *[DatePicker].getDayOfMonh()* permet d'avoir le jour du mois choisi avec un objet [DatePicker] dans l'intervalle [1,31] ;
- ligne 30 : la méthode *[TimePicker].getCurrentHour()* permet d'avoir l'heure choisie avec un objet [TimePicker] ;
- ligne 31 : la méthode *[TimePicker].getCurrentMinute()* permet d'avoir les minutes choisies avec un objet [TimePicker] ;
- ligne 34 : la méthode *[Spinner].getSelectedItemPosition()* permet d'avoir la position de l'élément sélectionné dans une liste déroulante ;

- ligne 35 : la méthode *[Spinner].getSelectedItem()* permet d'avoir l'objet sélectionné dans une liste déroulante ;

La méthode [doAfficher] qui affiche la liste des valeurs saisies est la suivante :

```
1.     private void doAfficher(List<String> messages) {
2.        // on construit le texte à affiche
3.        StringBuilder texte = new StringBuilder();
4.        for (String message : messages) {
5.           texte.append(String.format("%s\n", message));
6.        }
7.        // on l'affiche
8.        new AlertDialog.Builder(activité).setTitle("Valeurs
   saisies").setMessage(texte).setNeutralButton("Fermer", null).show();
9.  }
```

- ligne 1 : la méthode reçoit une liste de messages à afficher ;
- lignes 3-6 : un objet [StringBuilder] est construit à partir de ces messages. Pour concaténer des chaînes, le type [StringBuilder] est plus efficace que le type [String] ;
- ligne 8 : une boîte de dialogue affiche le texte de la ligne 3 :



- ligne 8 :
  - *setTitle* affiche [1],
  - *setMessage* affiche [2],
  - *setNeutralButton* affiche [3]. Le 1er paramètre est le libellé du bouton, le second une référence sur le gestionnaire du clic sur ce bouton. Ici nous n'en avons pas. Un clic sur le bouton fermera simplement la boîte de dialogue ;

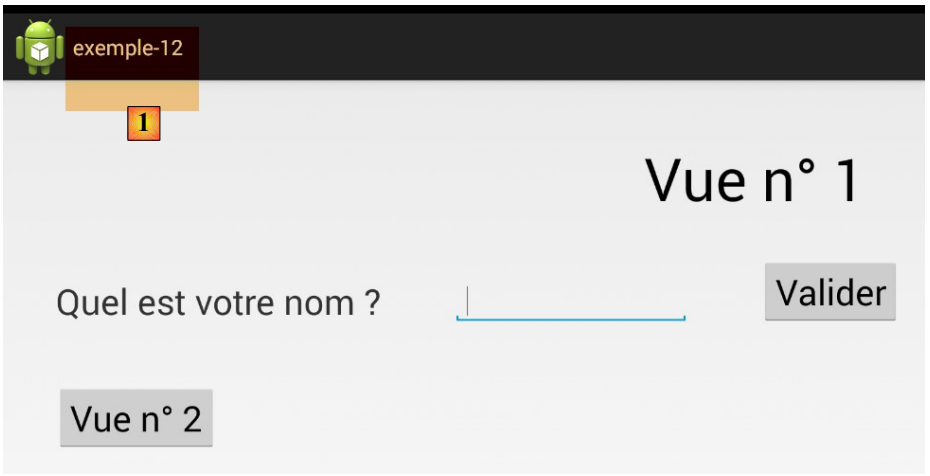
1.12.5 Exécution du projet

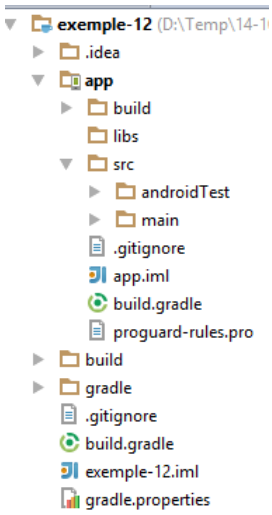
Exécutez le projet et testez les différents composants de saisie.

1.13 Exemple-12 : utilisation d'un patron de vues

1.13.1 Création du projet

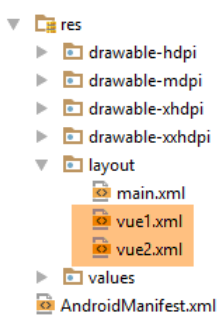
Nous créons un nouveau projet [exemple-12] par recopie du projet [exemple-08]. Pour cela suivez la démarche indiquée au paragraphe 1.10.1, page 91 qui décrit la création du projet [exemple-09] à partir du projet [exemple-08]. A la fin de la démarche, lorsque vous exécutez le projet [exemple-12] vous devez obtenir le résultat suivant [1] :

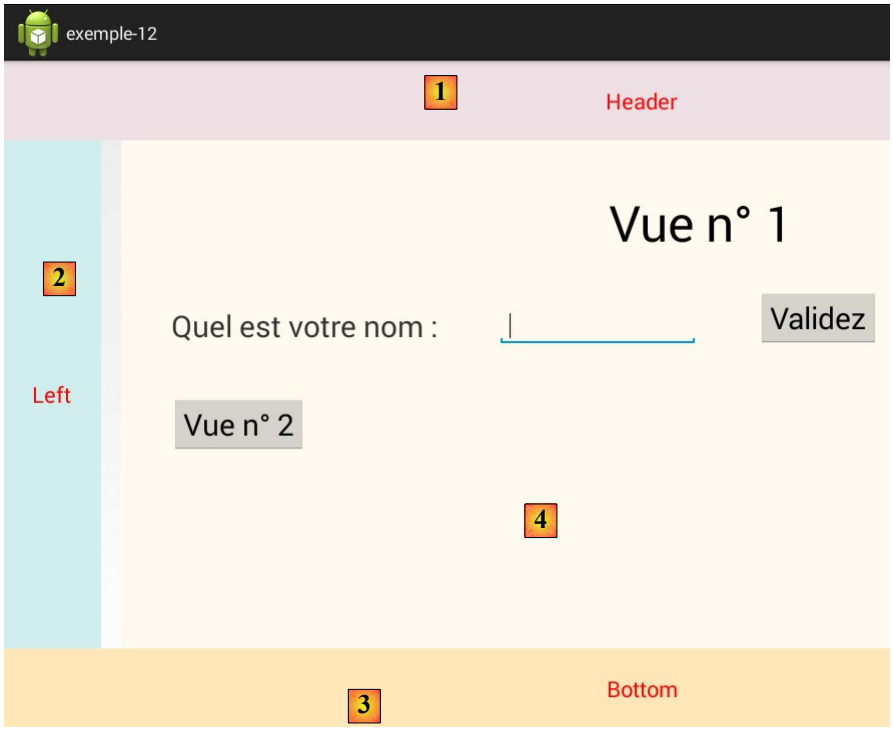




1.13.2 Le patron des vues

Nous voulons reprendre les deux vues du projet et les inclure dans un patron :

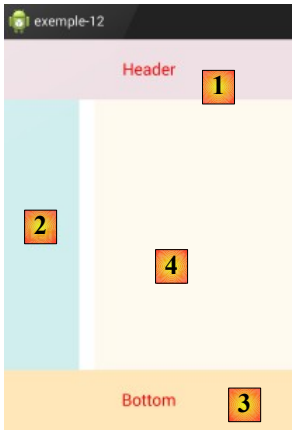


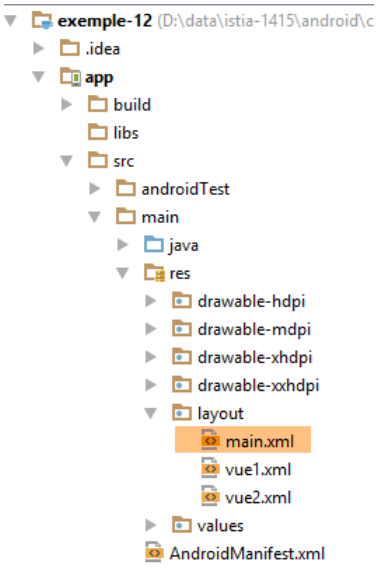


Chacune des deux vues sera structurée de la même façon :

- en [1], un entête ;
- en [2], une colonne de gauche qui pourrait contenir des liens ;
- en [3], un bas de page ;
- en [4], un contenu.

Ceci est obtenu en modifiant la vue de base [main.xml] de l'activité ;





Le code XML de la vue [main] est le suivant :

```
1.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.                xmlns:tools="http://schemas.android.com/tools"
3.                android:layout_width="match_parent"
4.                android:layout_height="match_parent"
```

```
5.                android:gravity="center"
6.                android:orientation="vertical" >
7.
8.    <LinearLayout
9.      android:id="@+id/header"
10.     android:layout_width="match_parent"
11.     android:layout_height="100dp"
12.     android:layout_weight="0.1"
13.     android:background="@color/lavenderblushh2" >
14.
15.     <TextView
16.       android:id="@+id/textViewHeader"
17.       android:layout_width="match_parent"
18.       android:layout_height="wrap_content"
19.       android:layout_gravity="center"
20.       android:gravity="center_horizontal"
21.       android:text="@string/txt_header"
22.       android:textAppearance="?android:attr/textAppearanceLarge"
23.       android:textColor="@color/red" />
24.   </LinearLayout>
25.
26.   <LinearLayout
27.     android:layout_width="match_parent"
28.     android:layout_height="fill_parent"
29.     android:layout_weight="0.8"
30.     android:orientation="horizontal" >
31.
32.     <LinearLayout
33.       android:id="@+id/left"
34.       android:layout_width="100dp"
35.       android:layout_height="match_parent"
36.       android:background="@color/lightcyan2" >
37.
38.       <TextView
39.         android:id="@+id/txt_left"
40.         android:layout_width="fill_parent"
41.         android:layout_height="fill_parent"
42.         android:gravity="center_vertical|center_horizontal"
43.         android:text="@string/txt_left"
44.         android:textAppearance="?android:attr/textAppearanceLarge"
45.         android:textColor="@color/red" />
46.     </LinearLayout>
47.
48.     <android.exemples.activity.MyPager
49.       xmlns:android="http://schemas.android.com/apk/res/android"
50.       xmlns:tools="http://schemas.android.com/tools"
51.       android:id="@+id/pager"
52.       android:layout_width="match_parent"
53.       android:layout_height="match_parent"
54.       android:layout_marginLeft="20dp"
55.       android:background="@color/floral_white"
56.       tools:context=".MainActivity" />
57.   </LinearLayout>
58.
59.   <LinearLayout
60.     android:id="@+id/bottom"
61.     android:layout_width="match_parent"
62.     android:layout_height="100dp"
63.     android:layout_weight="0.1"
64.     android:background="@color/wheat1" >
65.
66.     <TextView
67.       android:id="@+id/textViewBottom"
68.       android:layout_width="fill_parent"
69.       android:layout_height="fill_parent"
70.       android:gravity="center_vertical|center_horizontal"
71.       android:text="@string/txt_bottom"
72.       android:textAppearance="?android:attr/textAppearanceLarge"
73.       android:textColor="@color/red" />
74.   </LinearLayout>
75.
76. </LinearLayout>
```

- l'entête [1] est obtenu avec les lignes 8-24 ;

- la bande gauche [2] est obtenue avec les lignes 32-46 ;
- le bas de page [3] est obtenu avec les lignes 66-74 ;
- le contenu [4] est obtenu avec les lignes 48-57. On notera le nom [*pager*] de ce conteneur (ligne 51) ainsi que le nom de la classe [android.exemples.activity.MyPager] référencée ligne 48 ;

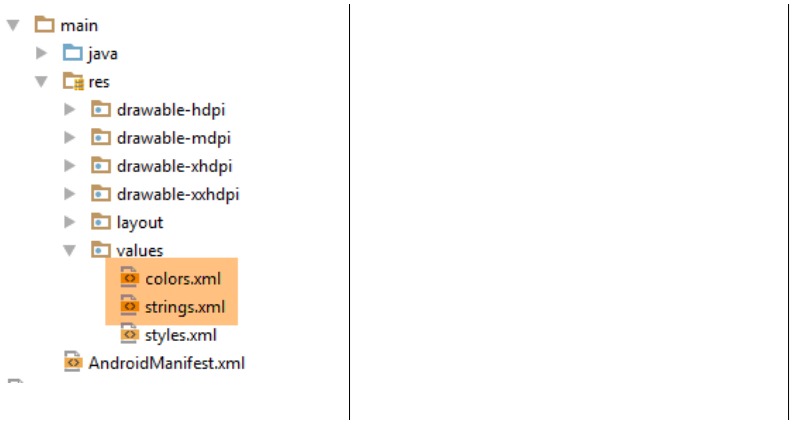
Maintenant rappelons le code dans l'activité [MainActivity] qui affiche une vue :

```
1.  // le conteneur des fragments
2.    @ViewById(R.id.pager)
3.    MyPager mViewPager;
4.
5.    ...
6.    @AfterViews
7.    protected void initActivity() {
8.
9.      // on inhibe le swipe entre fragments
10.     mViewPager.setSwipeEnabled(false);
11.
12.     // instanciation du gestionnaire de fragments
13.     mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
14.
15.     // qu'on associe à notre conteneur de fragments
16.     mViewPager.setAdapter(mSectionsPagerAdapter);
17.
18.   }
```

- ligne 16 : le composant d'id [*pager*] (lignes 2-3) est le conteneur qui recevra les vues gérées par le [ViewPager]. Le reste (entête, bande gauche, bas de page) est conservé.

On a là une méthode analogue à celle des templates des facelets de JSF2 (Java Server Faces).

La vue XML [main] utilise des informations trouvées dans les fichiers [res / values / colors.xml] et [res / values / strings.xml] :



Le fichier [colors.xml] est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.
4.      <color name="red">#FF0000</color>
5.      <color name="blue">#0000FF</color>
6.      <color name="wheat">#FFEFD5</color>
7.      <color name="floral_white">#FFFAF0</color>
8.      <color name="lavenderblushh2">#EEE0E5</color>
9.      <color name="lightcyan2">#D1EEEE</color>
10.     <color name="wheat1">#FFE7BA</color>
11.
12. </resources>
```

et le fichier [strings.xml] le suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
```

```
3.
4.      <string name="app_name">exemple-12</string>
5.      <string name="action_settings">Settings</string>
6.      <string name="titre_vue1">Vue n° 1</string>
7.      <string name="textView_nom">Quel est votre nom :</string>
8.      <string name="btn_Valider">Validez</string>
9.      <string name="btn_vue2">Vue n° 2</string>
10.     <string name="titre_vue2">Vue n° 2</string>
11.     <string name="btn_vue1">Vue n° 1</string>
12.     <string name="textView_bonjour">"Bonjour "</string>
13.     <string name="txt_header">Header</string>
14.     <string name="txt_left">Left</string>
15.     <string name="txt_bottom">Bottom</string>
16.
17. </resources>
```

Créez un contexte d'exécution pour ce projet et exécutez-le.

1.14 Exemple-13 : le composant [ListView]

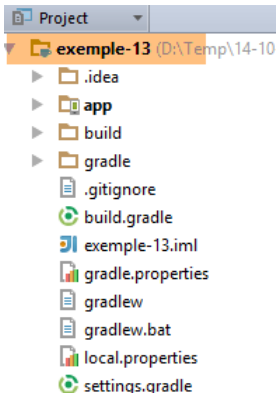
Le composant [ListView] permet de répéter une vue particulière pour chaque élément d'une liste. La vue répétée peut être d'une complexité quelconque, d'une simple chaîne de caractères à une vue permettant de saisir des informations pour chaque élément de la liste. Nous allons créer le [ListView] suivant :

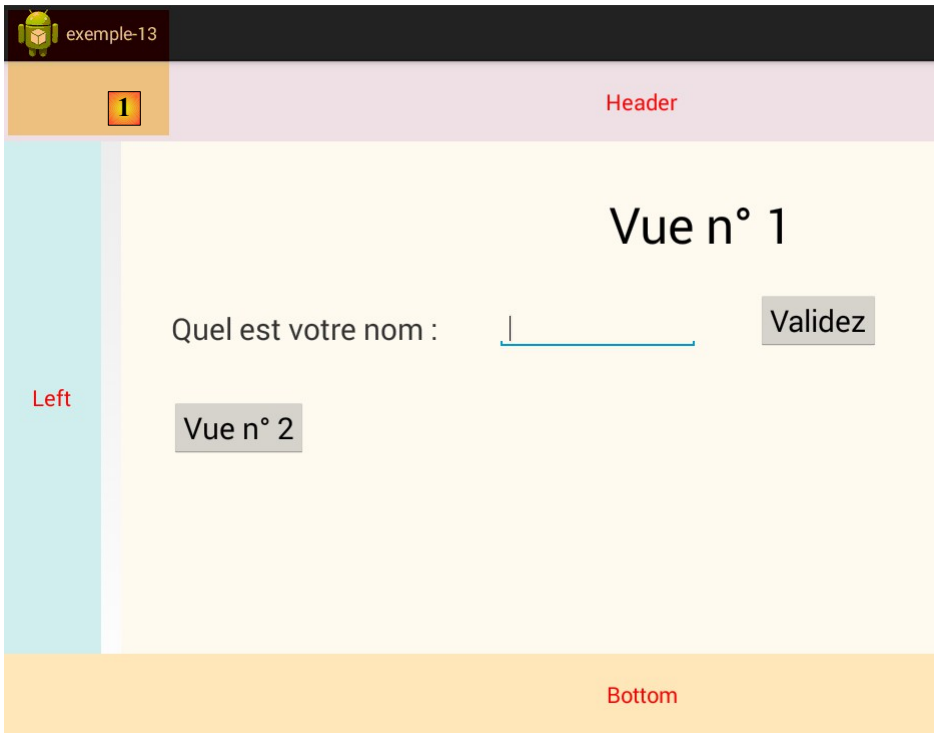
Chaque vue de la liste a trois composants :

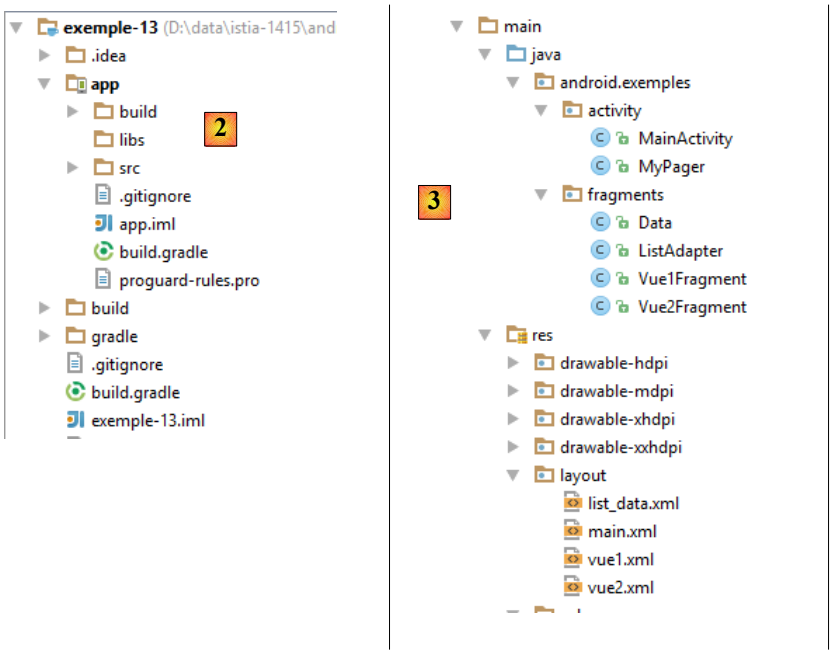
- un [TextView] d'information ;
- un [CheckBox] ;
- un [TextView] cliquable ;

1.14.1 Création du projet

Le projet [exemple-13] est tout d'abord obtenu par recopie du projet précédent [exemple-12] [1-2] (suivez la démarche faite déjà de nombreuses fois) :

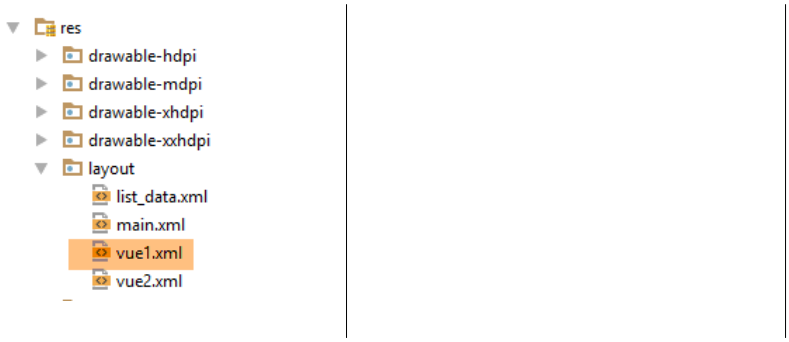


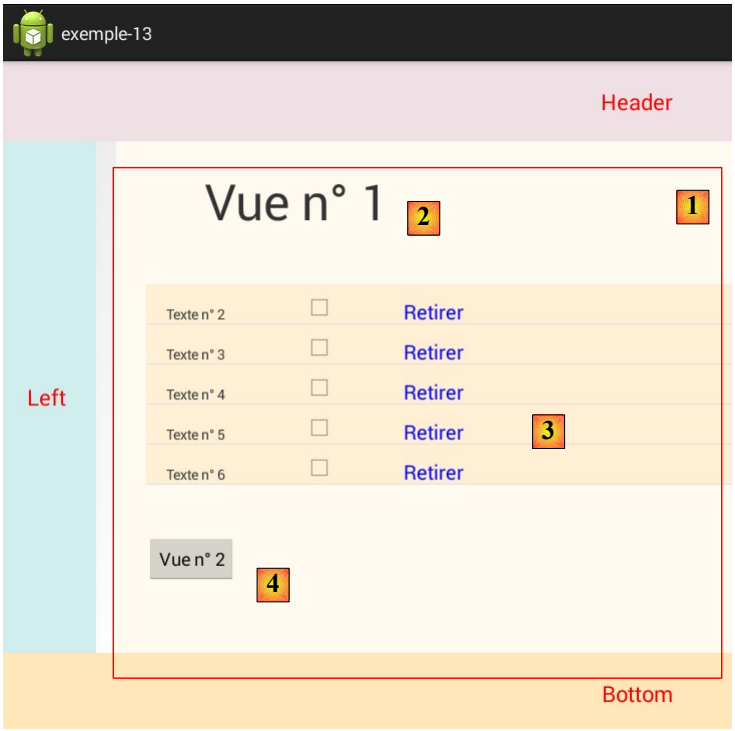




Nous allons faire évoluer le projet comme indiqué en [3].

1.14.2 La vue [Vue1] initiale





La vue XML [vue1.xml] affiche la zone [1] ci-dessus. Son code est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="match_parent"
4.      android:layout_height="match_parent" >
5.
6.      <TextView
7.          android:id="@+id/textView_titre"
8.          android:layout_width="wrap_content"
9.          android:layout_height="wrap_content"
10.         android:layout_alignParentLeft="true"
11.         android:layout_alignParentTop="true"
12.         android:layout_marginLeft="88dp"
13.         android:layout_marginTop="26dp"
14.         android:text="@string/vue1_titre"
15.         android:textSize="50sp" />
16.
17.     <Button
18.         android:id="@+id/button_vue2"
19.         android:layout_width="wrap_content"
20.         android:layout_height="wrap_content"
21.         android:layout_alignLeft="@+id/listView1"
22.         android:layout_below="@+id/listView1"
23.         android:layout_marginTop="50dp"
24.         android:text="@string/btn_vue2" />
25.
26.     <ListView
27.         android:id="@+id/listView1"
28.         android:layout_width="600dp"
29.         android:layout_height="200dp"
30.         android:layout_alignParentLeft="true"
31.         android:layout_below="@+id/textView_titre"
32.         android:layout_marginLeft="30dp"
33.         android:layout_marginTop="50dp" >
34.     </ListView>
35.
36. </RelativeLayout>
```

- lignes 6-15 : le composant [TextView] [2] ;
- lignes 26-34 : le composant [ListView] [3] ;
- lignes 17-24 : le composant [Button] [4] ;

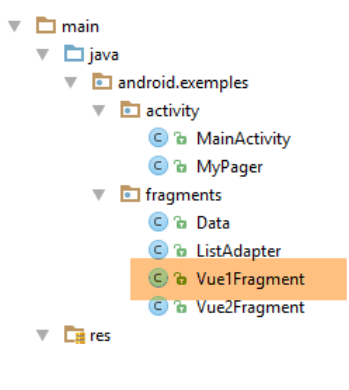
1.14.3 La vue répétée par le [ListView]

La vue répétée par le [ListView] est la vue [list_data] suivante :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:id="@+id/RelativeLayout1"
4.      android:layout_width="match_parent"
5.      android:layout_height="match_parent"
6.      android:background="@color/wheat" >
7.
8.      <TextView
9.          android:id="@+id/txt_Libellé"
10.         android:layout_width="100dp"
11.         android:layout_height="wrap_content"
12.         android:layout_marginLeft="20dp"
13.         android:layout_marginTop="20dp"
14.         android:text="@string/txt_dummy" />
15.
16.     <CheckBox
17.         android:id="@+id/checkBox1"
18.         android:layout_width="wrap_content"
19.         android:layout_height="wrap_content"
20.         android:layout_alignBottom="@+id/txt_Libellé"
21.         android:layout_marginLeft="37dp"
22.         android:layout_toRightOf="@+id/txt_Libellé"
23.         android:text="@string/txt_dummy" />
24.
25.     <TextView
26.         android:id="@+id/textViewRetirer"
27.         android:layout_width="wrap_content"
28.         android:layout_height="wrap_content"
29.         android:layout_alignBaseline="@+id/txt_Libellé"
30.         android:layout_alignBottom="@+id/txt_Libellé"
31.         android:layout_marginLeft="68dp"
32.         android:layout_toRightOf="@+id/checkBox1"
33.         android:text="@string/txt_retirer"
34.         android:textColor="@color/blue"
35.         android:textSize="20sp" />
36.
37. </RelativeLayout>
```

- lignes 8-14 : le composant [TextView] [1] ;
- lignes 16-23 : le composant [CheckBox] [2] ;
- lignes 25-35 : le composant [TextView] [3] ;

1.14.4 Le fragment [Vue1Fragment]



Le fragment [Vue1Fragment] gère la vue XML [vue1]. Son code est le suivant :

```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.exemples.activity.MainActivity;
5.  import android.support.v4.app.Fragment;
6.  import android.view.View;
7.  import android.widget.ListView;
8.  import org.androidannotations.annotations.AfterViews;
9.  import org.androidannotations.annotations.Click;
10. import org.androidannotations.annotations.EFragment;
11. import org.androidannotations.annotations.ViewById;
12.
13. import java.util.List;
14.
15. @EFragment(R.layout.vue1)
16. public class Vue1Fragment extends Fragment {
17.
18.    // les champs de la vue affichée par le fragment
19.   @ViewById(R.id.listView1)
20.   ListView listView;
21.    // l'activité
22.    private MainActivity activité;
23.    // l'adaptateur de liste
24.    private ListAdapter adapter;
25.
26.    @AfterViews
27.    void initFragment() {
28.       // on récupère l'unique activité
29.       activité = (MainActivity) getActivity();
30.       // on associe des données au [ListView]
31.       adapter = new ListAdapter(activité, R.layout.list_data, activité.getListe(), this);
32.       listView.setAdapter(adapter);
33.    }
34.
35.   @Click(R.id.button_vue2)
36.    void navigateToView2() {
37.       // on navigue vers la vue 2
38.       activité.navigateToView(1);
39.    }
40.
41.    public void doRetirer(int position) {
42.       ...
43.    }
44. }
```

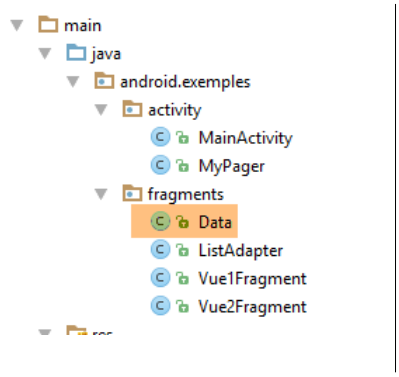
- ligne 15 : la vue XML [vue1] est associée au fragment ;
- lignes 19-20 : on récupère une référence sur le composant [ListView] de [vue1] ;
- lignes 31-32 : on associe à ce [ListView] une source de données de type [ListAdapter] (ligne 31). Nous allons construire cette classe. Elle dérive de la classe [ArrayAdapter] que nous avons déjà eu l'occasion d'utiliser pour associer des données à un [ListView] ;
- ligne 31 : nous passons diverses informations au constructeur de [ListAdapter] :

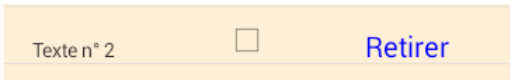
- une référence sur l'activité courante,
- l'id de la vue qui sera instanciée pour chaque élément de la liste,
- une source de données pour alimenter la liste,
- une référence sur le fragment. Celle-ci sera utilisée pour faire gérer le clic sur un lien [Retirer] du [ListView] par la méthode [doRetirer] de la ligne 41 ;

La source de données est définie dans [MainActivity] ;

```
1.     // une liste de données
2.     private List<Data> liste;
3.
4.     // constructeur
5.     public MainActivity() {
6.        // on crée une liste de données
7.        liste = new ArrayList<Data>();
8.        for (int i = 0; i < 20; i++) {
9.           liste.add(new Data("Texte n° " + i, false));
10.       }
11. }
```

Le constructeur de la classe [MainActivity] crée 20 données du type [Data] suivant :



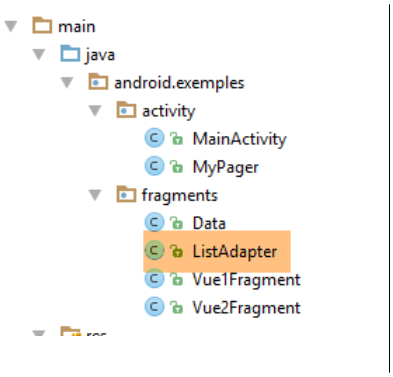



```
1.  package android.exemples.fragments;
2.
3.  public class Data {
4.
5.     // données
6.     private String texte;
7.     private boolean isChecked;
8.
9.     // constructeur
10.    public Data(String texte, boolean isCkecked) {
11.       this.texte = texte;
12.       this.isChecked = isCkecked;
13.    }
14.
15.    // getters et setters
16.    public String getTexte() {
17.       return texte;
18.    }
19.
20.    public void setTexte(String texte) {
21.       this.texte = texte;
22.    }
23.
24.    public boolean isChecked() {
25.       return isChecked;
26.    }
27.
28.    public void setChecked(boolean isChecked) {
29.       this.isChecked = isChecked;
```

```
30.    }
31.
32. }
```

- ligne 6 : le texte qui va alimenter le premier [TextView] de [list_data] ;
- ligne 7 : le booléen qui va servir à cocher ou non le [checkBox] de [list_data] ;

1.14.5 L'adaptateur [ListAdapter] du [ListView]



La classe [ListAdapter]

- configure la source de données du [ListView] ;
- gère l'affichage des différents éléments du [ListView] ;
- gère les événements de ces éléments ;

Son code est le suivant :

```
1.  package android.exemples.fragments;
2.
3.  import java.util.List;
4.  ...
5.  public class ListAdapter extends ArrayAdapter<Data> {
6.
7.     // le contexte d'exécution
8.     private Context context;
9.     // l'id du layout d'affichage d'une ligne de la liste
10.    private int layoutResourceId;
11.    // les données de la liste
12.    private List<Data> data;
13.    // le fragment qui affiche le [ListView]
14.    private Vue1Fragment fragment;
15.    // l'adapteur
16.    final ListAdapter adapter = this;
17.
18.    // constructeur
19.    public ListAdapter(Context context, int layoutResourceId, List<Data> data, Vue1Fragment fragment) {
20.       super(context, layoutResourceId, data);
21.       // on mémorise les infos
22.       this.context = context;
23.       this.layoutResourceId = layoutResourceId;
24.       this.data = data;
25.       this.fragment = fragment;
26.    }
27.
28.    @Override
29.    public View getView(final int position, View convertView, ViewGroup parent) {
30. ...
31.    }
32. }
```

- ligne 5 : la classe [ListAdapter] étend la classe [ArrayAdapter] ;
- ligne 19 : le constructeur ;
- ligne 20 : ne pas oublier d'appeler le constructeur de la classe parent [ArrayAdapter] avec les trois premiers paramètres ;
- lignes 22-25 : on mémorise les informations du constructeur ;

- ligne 29 : la méthode [getView] va être appelée de façon répétée par le [ListView] pour générer la vue de l'élément n° [position]. Le résultat [View] rendu est une référence sur la vue créée.

Le code de la méthode [getView] est le suivant :

```
1.  @Override
2.      public View getView(final int position, View convertView, ViewGroup parent) {
3.          // on crée la ligne
4.          View row = ((Activity) context).getLayoutInflater().inflate(layoutResourceId, parent, false);
5.          // le texte
6.          TextView textView = (TextView) row.findViewById(R.id.txt_Libellé);
7.          textView.setText(data.get(position).getTexte());
8.          // la case à cocher
9.          CheckBox checkBox = (CheckBox) row.findViewById(R.id.checkBox1);
10.         checkBox.setChecked(data.get(position).isChecked());
11.         // le lien [Retirer]
12.         TextView txtRetirer = (TextView) row.findViewById(R.id.textViewRetirer);
13.         txtRetirer.setOnClickListener(new OnClickListener() {
14.
15.             public void onClick(View v) {
16.                 fragment.doRetirer(position);
17.             }
18.         });
19.         // on gère le clic sur la case à cocher
20.         checkBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
21.
22.             public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
23.                 data.get(position).setChecked(isChecked);
24.             }
25.         });
26.         // on rend la ligne
27.         return row;
28. }
```

- ligne 2 : la méthode reçoit trois paramètres. Nous n'allons utiliser que le premier ;
- ligne 4 : on crée la vue de l'élément n° [position]. C'est la vue [list_data] dont l'id a été passé comme deuxième paramètre au constructeur. Ensuite on procède comme d'habitude. On récupère les références des composants de la vue qu'on vient d'instancier ;
- ligne 6 : on récupère la référence du [TextView] n° 1 ;
- ligne 7 : on lui assigne un texte provenant de la source de données qui a été passée comme troisième paramètre au constructeur ;
- ligne 9 : on récupère la référence du [CheckBox] n° 2 ;
- ligne 10 : on le coche ou non avec une valeur provenant de la source de données du [ListView] ;
- ligne 12 : on récupère la référence du [TextView] n° 3 ;
- lignes 13-18 : on gère le clic sur le lien [Retirer] ;
- ligne 16 : c'est la méthode *[Vue1Fragment].doRetirer* qui va gérer ce clic. Il paraît en effet plus logique de faire gérer cet événement par le fragment qui affiche le [ListView]. Il a une vue d'ensemble que n'a pas la classe [ListAdapter]. La référence du fragment [Vue1Fragment] avait été passée comme quatrième paramètre au constructeur de la classe ;
- lignes 20-25 : on gère le clic sur la case à cocher. L'action faite sur elle est répercutée sur la donnée qu'elle affiche. Ceci pour la raison suivante. Le [ListView] est une liste qui n'affiche qu'une partie de ces éléments. Ainsi un élément de la liste est-il parfois caché, parfois affiché. Lorsque l'élément n° i doit être affiché, la méthode [getView] de la ligne 2 ci-dessus est appelée pour la position n° i. La ligne 10 va recalculer l'état de la case à cocher à partir de la donnée à laquelle elle est liée. Il faut donc que celle-ci mémorise l'état de la case à cocher au fil du temps ;

1.14.6 Retirer un élément de la liste

Le clic sur le lien [Retirer] est géré dans le fragment [Vue1Fragment] par la méthode [doRetirer] suivante :

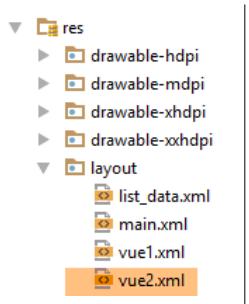
```
1.      public void doRetirer(int position) {
2.          // on enlève l'élément n° [position] dans la liste
3.          List<Data> liste = activité.getListe();
4.          liste.remove(position);
5.          // on note la position du scroll pour y revenir
6.          // lire
7.          // [http://stackoverflow.com/questions/3014089/maintain-save-restore-scroll-position-when-
    returning-to-a-listview]
8.          // position du 1er élément visible complètement ou non
9.          int firstPosition = listView.getFirstVisiblePosition();
10.         // offset Y de cet élément par rapport au haut du ListView
```

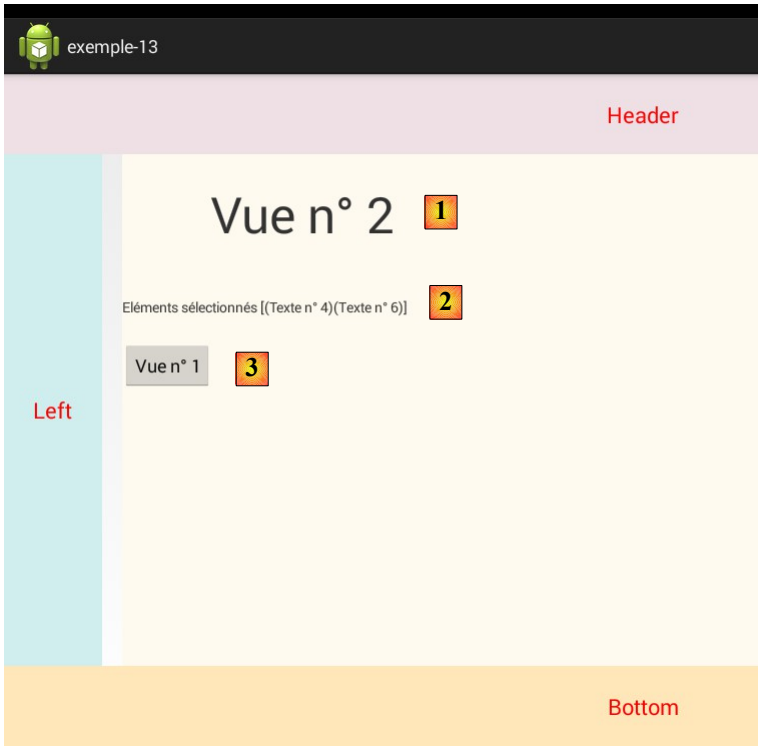
```
11.         // mesure la hauteur de la partie éventuellement cachée
12.         View v = listView.getChildAt(0);
13.         int top = (v == null) ? 0 : v.getTop();
14.         // on réassocie des données au [ListView]
15.         listView.setAdapter(adapter);
16.         // on se positionne au bon endroit du ListView
17.         listView.setSelectionFromTop(firstPosition, top);
18.
19. }
```

- ligne 1 : on reçoit la position dans le [ListView] du lien [Retirer] qui a été cliqué ;
- ligne 3 : on récupère la liste de données ;
- ligne 4 : on retire l'élément de n° [position] ;
- ligne 15 : on réassocie les nouvelles données au [ListView]. Sans cela, visuellement rien ne change.
- lignes 5-13, 17 : une gymnastique assez complexe. Sans elle, il se passe la chose suivante :
    - le [ListView] affiche les lignes 15-18 de la liste de données,
    - on supprime la ligne 16,
    - la ligne 15 ci-dessus le réinitialise totalement et le [ListView] affiche alors les lignes 0-3 de la liste de données ;

Avec les lignes ci-dessus, la suppression se fait et le [ListView] reste positionné sur la ligne qui suit la ligne supprimée.

1.14.7 La vue XML [Vue2]





Le code XML de la vue est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="match_parent"
4.      android:layout_height="match_parent" >
5.
6.      <TextView
7.          android:id="@+id/textView_titre"
8.          android:layout_width="wrap_content"
9.          android:layout_height="wrap_content"
10.         android:layout_alignParentLeft="true"
11.         android:layout_alignParentTop="true"
12.         android:layout_marginLeft="88dp"
```

```
13.         android:layout_marginTop="26dp"
14.         android:text="@string/vue2_titre"
15.         android:textSize="50sp" />
16.
17.     <Button
18.         android:id="@+id/button_vue1"
19.         android:layout_width="wrap_content"
20.         android:layout_height="wrap_content"
21.         android:layout_below="@+id/textViewResultats"
22.         android:layout_marginTop="25dp"
23.         android:text="@string/btn_vue1" />
24.
25.     <TextView
26.         android:id="@+id/textViewResultats"
27.         android:layout_width="wrap_content"
28.         android:layout_height="wrap_content"
29.         android:layout_below="@+id/textView_titre"
30.         android:layout_marginTop="50dp"
31.         android:text="" />
32.
33. </RelativeLayout>
```

- lignes 6-15 : le composant [TextView] n° 1 ;
- lignes 25-31 : le composant [TextView] n° 2 ;
- lignes 17-23 : le composant [Button] n° 3 ;

1.14.8 Le fragment [Vue2Fragment]

Le fragment [Vue2Fragment] gère la vue XML [vue2]. Son code est le suivant :

```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.exemples.activity.MainActivity;
5.  import android.support.v4.app.Fragment;
6.  import android.widget.Button;
7.  import android.widget.TextView;
8.  import org.androidannotations.annotations.AfterViews;
9.  import org.androidannotations.annotations.Click;
10. import org.androidannotations.annotations.EFragment;
11. import org.androidannotations.annotations.ViewById;
12.
13. @EFragment(R.layout.vue2)
14. public class Vue2Fragment extends Fragment {
15.
16.    // les champs de la vue
17.    private Button btnVue1;
18.   @ViewById(R.id.textViewResultats)
19.   TextView txtResultats;
20.
21.    // l'activité
22.    private MainActivity activité;
23.
24.    @AfterViews
```

```
25.     void initFragment(){
26.        // on récupère l'unique activité
27.        activité = (MainActivity) getActivity();
28.     }
29.
30.     @Override
31.     public void setMenuVisibility(final boolean visible) {
32.        super.setMenuVisibility(visible);
33.        if (visible) {
34.           // la vue est visible - on affiche les éléments de la liste qui ont été
35.           // sélectionnés
36.           StringBuilder texte = new StringBuilder("Eléments sélectionnés [");
37.           for (Data data : activité.getListe()) {
38.              if (data.isChecked()) {
39.                 texte.append(String.format("(%s)", data.getTexte()));
40.              }
41.           }
42.           texte.append("]");
43.           txtResultats.setText(texte);
44.        }
45.     }
46.
47.    @Click(R.id.button_vue1)
48.     void navigateToView1() {
49.        // on navigue vers la vue 1
50.        activité.navigateToView(0);
51.     }
52.
53. }
```

Le code important est dans la méthode [setMenuVisibility] de la ligne 31. Celle-ci est exécutée dès que la vue affichée par le fragment va devenir visible ou cachée à l'utilisateur.

- ligne 32 : on appelle la méthode de la classe parent. Obligatoire ;
- ligne 33 : si la vue va devenir visible, alors on calcule le texte à afficher dans le [TextView] n° 2 ;
- lignes 37-40 : on parcourt la liste des données affichée par le [ListView]. Elle est stockée dans l'activité ;
- ligne 39 : si la donnée n° i a été cochée, on ajoute le libellé associé dans un type [StringBuilder] ;
- ligne 43 : le [TextView] affiche le texte calculé ;

1.14.9 Exécution

Créez une configuration d'exécution pour ce projet et exécutez-la.

1.14.10 Amélioration

Dans l'exemple précédent nous avons utilisé une source de données List<Data> où la classe [Data] était la suivante :

```
1.  package android.exemples.fragments;
2.
3.  public class Data {
4.
5.     // données
6.     private String texte;
7.     private boolean isChecked;
8.
9.     // constructeur
10.    public Data(String texte, boolean isCkecked) {
11.       this.texte = texte;
12.       this.isChecked = isCkecked;
13.    }
14. ...
15.
16. }
```

Ligne 7, on avait utilisé un booléen pour gérer la case à cocher des éléments du [ListView]. Souvent, le [ListView] doit afficher des données qu'on peut sélectionner en cochant une case sans que pour autant l'élément de la source de données ait un champ booléen correspondant à cette case. On peut alors procéder de la façon suivante :

La classe [Data] devient la suivante :

```
1.  package android.exemples.fragments;
2.
3.  public class Data {
4.
5.     // données
6.     private String texte;
7.
8.     // constructeur
9.     public Data(String texte) {
10.       this.texte = texte;
11.    }
12.
13.    // getters et setters
14. ...
15. }
```

On crée une classe [CheckedData] dérivée de la précédente :

```
1.  package android.exemples.fragments;
2.
3.  public class CheckedData extends Data {
4.
5.     // élément coché
6.     private boolean isChecked;
7.
8.     // constructeur
9.     public CheckedData(String text, boolean isChecked) {
10.       // parent
11.       super(text);
12.       // local
13.       this.isChecked = isChecked;
14.    }
15.
16.    // getters et setters
17. ...
18. }
```

Il suffit ensuite de remplacer partout dans le code, le type [Data] par le type [CheckedData]. Par exemple dans [MainActivity] :

```
1.     // une liste de données à cocher
2.     private List<CheckedData> liste;
3.
4.     // constructeur
5.     public MainActivity() {
6.        // on crée une liste de données
7.        liste = new ArrayList<CheckedData>();
8.        for (int i = 0; i < 20; i++) {
9.           liste.add(new CheckedData("Texte n° " + i, false));
10.       }
11. }
```

Le projet de cette version vous est fourni sous le nom [exemple-13B].

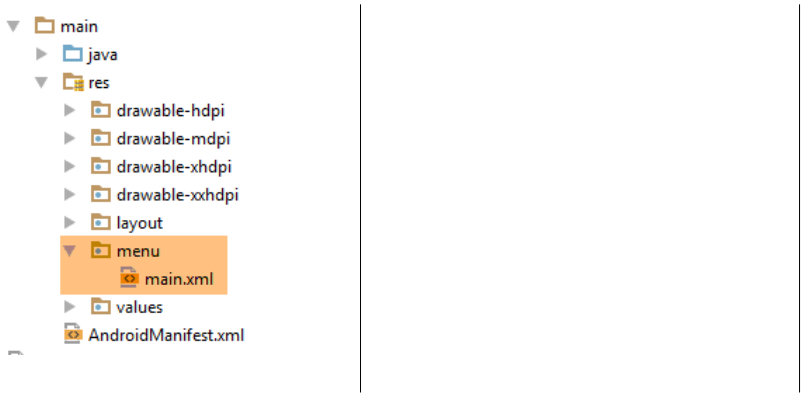
1.15 Exemple-14 : utiliser un menu

1.15.1 Création du projet

Nous dupliquons le projet [exemple-12] dans le projet [exemple-14] [1-2] :

Nous allons supprimer les boutons des vues 1 et 2 pour les remplacer par des options de menu [3-4].

1.15.2 La définition XML du menu



Le fichier [res / menu / main] définit un menu. Son contenu est le suivant :

```
1.  <menu xmlns:android="http://schemas.android.com/apk/res/android" >
```

```
2.
3.      <item
4.          android:id="@+id/menuActions"
5.          android:showAsAction="ifRoom"
6.          android:title="@string/menuActions">
7.          <menu>
8.              <item
9.                  android:id="@+id/actionValider"
10.                 android:title="@string/actionValider"/>
11.         </menu>
12.     </item>
13.     <item
14.         android:id="@+id/menuNavigation"
15.         android:showAsAction="ifRoom"
16.         android:title="@string/menuNavigation">
17.         <menu>
18.             <item
19.                 android:id="@+id/navigationVue1"
20.                 android:title="@string/navigationVue1"/>
21.             <item
22.                 android:id="@+id/navigationVue2"
23.                 android:title="@string/navigationVue2"/>
24.         </menu>
25.     </item>
26.
27. </menu>
```

Les éléments du menu sont définis par les informations suivantes :

- **android:id** : l'identifiant de l'élément ;
- **android:title** : le libellé de l'élément ;
- **android:showsAsAction** : indique si l'élément de menu peut être placé dans la barre d'actions de l'activité. [ifRoom] indique que l'élément doit être placé dans la barre d'actions s'il y a de la place pour lui ;

1.15.3 La gestion du menu dans le fragment [Vue1Fragment]

La classe [Vue1Fragment] devient la suivante :

```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.exemples.activity.MainActivity;
5.  import android.support.v4.app.Fragment;
6.  import android.view.Menu;
7.  import android.view.MenuInflater;
8.  import android.view.MenuItem;
9.  import android.widget.EditText;
10. import android.widget.Toast;
11. import org.androidannotations.annotations.*;
12.
13. // un fragment est une vue affichée par un conteneur de fragments
14. @EFragment(R.layout.vue1)
15. @OptionsMenu(R.menu.main)
16. public class Vue1Fragment extends Fragment {
17.
18.   // les champs de la vue affichée par le fragment
19.   @ViewById(R.id.editText_nom)
20.   EditText edtNom;
21.   // option de navigation vers vue 1
22.   @OptionsMenuItem(R.id.navigationVue1)
23.   MenuItem menuToVue1;
24.   // option de navigation vers vue 2
25.   @OptionsMenuItem(R.id.navigationVue2)
26.   MenuItem menuToVue2;
27.   // menu de validation
28.   @OptionsMenuItem(R.id.menuActions)
29.   MenuItem menuActions;
30.   // l'activité
31.   private MainActivity activité;
32.   // le menu
33.   private boolean menuDone = false;
34.
35.   @AfterViews
```

```
36.   void initFragment() {
37.     // on récupère l'unique activité
38.     activité = (MainActivity) getActivity();
39.   }
40.
41.   @Override
42.   public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
43.     // parent
44.     super.onCreateOptionsMenu(menu, inflater);
45.     // personnalisation menu
46.     menuToVue1.setVisible(false);
47.     menuToVue2.setVisible(true);
48.     menuActions.setVisible(true);
49.   }
50.
51.   @OptionsItem(R.id.navigationVue2)
52.   void navigateToView2() {
53.     // on mémorise le nom dans l'activité
54.     activité.setNom(edtNom.getText().toString());
55.     // on navigue vers la vue 2
56.     activité.navigateToView(1);
57.   }
58.
59.   @OptionsItem(R.id.actionValider)
60.   void doValider() {
61.     // on affiche le nom saisi
62.     Toast.makeText(getActivity(), String.format("Bonjour %s", edtNom.getText().toString()),
    Toast.LENGTH_LONG).show();
63.   }
64.
65.   }
```

- ligne 15 : le menu [menu / main.xml] est associé au fragment ;
- lignes 21-29 : les références sur les trois options du menu ;
- lignes 42-46 : la méthode [onCreateOptionsMenu] peut être utilisée pour préparer le menu que la vue va afficher ;
- lignes 46-48 : l'option [Vue 1] du sous-menu [Navigation] est cachée. On ne garde que les options [Actions / Valider] et [Navigation / Vue 2] ;
- lignes 51-52 : la méthode [navigateToView2] gère le clic sur l'option de menu [Vue n° 2] ;
- lignes 59-60 : la méthode [doValider] gère le clic sur l'option de menu [Valider] ;

1.15.4 La gestion du menu dans le fragment [Vue2Fragment]

On retrouve un code similaire dans le fragment de la vue n° 2 :

```
1.  package android.exemples.fragments;
2.
3.  import android.exemples.R;
4.  import android.exemples.activity.MainActivity;
5.  import android.support.v4.app.Fragment;
6.  import android.view.Menu;
7.  import android.view.MenuInflater;
8.  import android.view.MenuItem;
9.  import android.widget.TextView;
10. import org.androidannotations.annotations.*;
11.
12. // un fragment est une vue affichée par un conteneur de fragments
13. @EFragment(R.layout.vue2)
14. @OptionsMenu(R.menu.main)
15. public class Vue2Fragment extends Fragment {
16.
17.    // les champs de la vue
18.   @ViewById(R.id.textView_bonjour)
19.    TextView textViewBonjour;
20.    // l'activité
21.    private MainActivity activité;
22.   // option de navigation vers vue 1
23.   @OptionsMenuItem(R.id.navigationVue1)
24.   MenuItem menuToVue1;
25.   // option de navigation vers vue 2
26.   @OptionsMenuItem(R.id.navigationVue2)
27.   MenuItem menuToVue2;
28.   // menu de validation
```

```
29.    @OptionsMenuItem(R.id.menuActions)
30.    MenuItem menuActions;
31.
32.
33.     @AfterViews
34.     void initFragment() {
35.        // on récupère l'unique activité
36.        activité = (MainActivity) getActivity();
37.     }
38.
39.    @Override
40.    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
41.      // parent
42.      super.onCreateOptionsMenu(menu, inflater);
43.      // personnalisation menu
44.      menuToVue1.setVisible(true);
45.      menuToVue2.setVisible(false);
46.      menuActions.setVisible(false);
47.    }
48.
49.    @Override
50.     public void setMenuVisibility(final boolean visible) {
51.        super.setMenuVisibility(visible);
52.        if (visible) {
53.           // la vue est visible - on affiche le nom saisi dans la vue 1
54.           textViewBonjour.setText(String.format("Bonjour %s !", activité.getNom()));
55.        }
56.     }
57.
58.    @OptionsItem(R.id.navigationVue1)
59.     void navigateToView1() {
60.        // on navigue vers la vue 1
61.        activité.navigateToView(0);
62.     }
63.
64. }
```

- lignes 44-46 : on cache l'option [Actions] et l'option [Navigation / Vue 2]. L'option [Vue n° 1] est affichée ;
- lignes 58-59 : lors du clic sur l'option [Navigation / Vue1], on appelle la méthode [navigateToView1] ;

1.15.5 Exécution

Créez un contexte d'exécution pour ce projet et exécutez-le.

1.16 Annexes

1.16.1 La tablette Android

Les exemples ont été testés avec la tablette Samsung Galaxy Tab 2.

Pour tester les exemples avec une tablette, vous devez d'abord installer le driver de celle-ci sur votre machine de développement. Celui de la tablette Samsung Galaxy Tab 2 peut être trouvé à l'URL [http://www.samsung.com/fr/support/usefulsoftware/KIES/] :

Pour tester les exemples, vous aurez besoin de connecter la tablette à un réseau (wifi probablement) et de connaître son adresse IP sur ce réseau. Voici comment procéder (Samsung Glaxy Tab 2) :

- allumez votre tablette ;
- cherchez dans les applications disponibles sur la tablette (en haut à droite) celle qui s'appelle [paramètres] avec une icône de roue dentée ;
- dans la section à gauche, activez le wifi ;
- dans la section à droite, sélectionnez un réseau wifi ;
- une fois connecté au réseau, faites une frappe courte sur le réseau sélectionné. L'adresse IP de la tablette sera affichée. Notez-la. Vous allez en avoir besoin ;

Toujours dans l'application [Paramètres],

- sélectionnez à gauche l'option [Options de développement] (tout en bas des options) ;
- vérifiez qu'à droite l'option [Débogage USB] est cochée.

Pour revenir au menu, tapez dans la barre d'état en bas, l'icône du milieu, celle d'une maison. Toujours dans la barre d'état en bas,

- l'icône la plus à gauche est celle du retour en arrière : vous revenez à la vue précédente ;
- l'icône la plus à droite est celle de la gestion des tâches. Vous pouvez voir et gérer toutes les tâches exécutées à un moment donné par votre tablette ;

Reliez votre tablette à votre PC avec le câble USB qui l'accompagne.

Installez la clé wifi sur l'un des ports USB du PC puis connectez-vous sur le même réseau wifi que la tablette. Ceci fait, dans une fenêtre DOS, tapez la commande [ipconfig] :

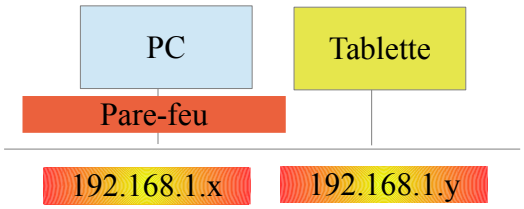
```
1. dos>ipconfig
2.
3. Configuration IP de Windows
```

```
4.
5.    Carte Ethernet Connexion au réseau local :
6.
7.       Suffixe DNS propre à la connexion. . . :
8.       Adresse IPv6 de liaison locale. . . . .: fe80::698b:455a:925:6b13%4
9.       Adresse IPv4. . . . . . . . . . . . . .: 192.168.2.1
10.      Masque de sous-réseau. . . . . . . . . : 255.255.255.0
11.      Passerelle par défaut. . . . . . . . . :
12.
13.   Carte réseau sans fil Wi-Fi :
14.
15.      Suffixe DNS propre à la connexion. . . :
16.      Adresse IPv6 de liaison locale. . . . .: fe80::39aa:47f6:7537:f8e1%2
17.      Adresse IPv4. . . . . . . . . . . . . .: 192.168.1.25
18.      Masque de sous-réseau. . . . . . . . . : 255.255.255.0
19.   Passerelle par défaut. . . . . . . . . : 192.168.1.1
```

Votre PC a deux cartes réseau et donc deux adresses IP :

- celle de la ligne 9 qui est celle du PC sur le réseau filaire ;
- celle de la ligne 17 qui est celle du PC sur le réseau wifi ;

Notez ces deux informations. Vous en aurez besoin. Vous êtes désormais dans la configuration suivante :



La tablette aura à se connecter à votre PC. Celui est normalement protégé par un pare-feu qui empêche tout élément extérieur d'ouvrir une connexion avec le PC. Il vous faut donc inhiber le pare-feu. Faites-le avec l'option [Panneau de configuration\Système et sécurité\Pare-feu Windows]. Parfois il faut de plus inhiber le pare-feu mis en place par l'antivirus. Cela dépend de votre antivirus.

Maintenant, sur votre PC, vérifiez la connexion réseau avec la tablette avec une commande [ping 192.168.1.y] où [192.168.1.y] est l'adresse IP de la tablette. Vous devez obtenir quelque chose qui ressemble à ceci :

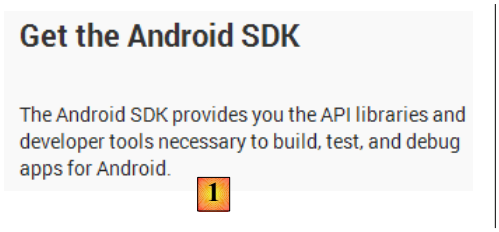
```
1.    dos>ping 192.168.1.26
2.
3.    Envoi d'une requête 'Ping'  192.168.1.26 avec 32 octets de données :
4.    Réponse de 192.168.1.26 : octets=32 temps=244 ms TTL=64
5.    Réponse de 192.168.1.26 : octets=32 temps=199 ms TTL=64
6.    Réponse de 192.168.1.26 : octets=32 temps=28 ms TTL=64
7.    Réponse de 192.168.1.26 : octets=32 temps=88 ms TTL=64
8.
9.    Statistiques Ping pour 192.168.1.26:
10.       Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
11.   Durée approximative des boucles en millisecondes :
12.   Minimum = 28ms, Maximum = 244ms, Moyenne = 139ms
```

Les lignes 4-7 indiquent que la tablette d'adresse IP [192.168.1.y] a répondu à la commande [ping].

1.16.2 Installation d'un JDK

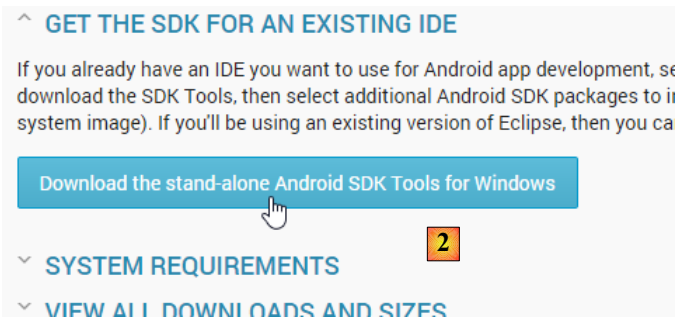
On trouvera à l'URL [http://www.oracle.com/technetwork/java/javase/downloads/index.html] (octobre 2014), le JDK le plus récent. On nommera par la suite <jdk-install> le dossier d'installation du JDK.

1.16.3 Installation du SDK Manager d'Android



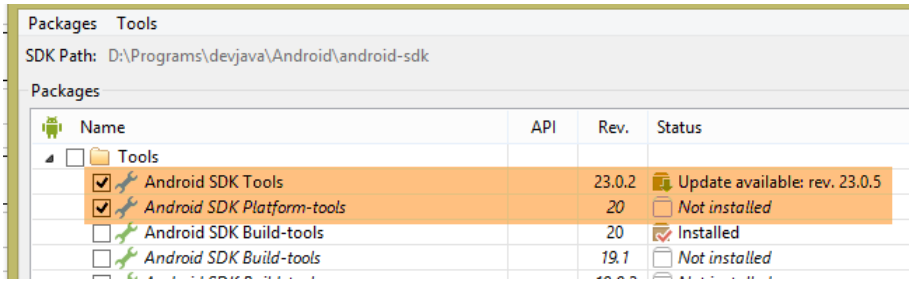
- en [1], pourquoi on a besoin du SDK d'Android ;

On trouvera le SDK Manager d'Android à l'adresse [http://developer.android.com/sdk/index.html] (octobre 2014).

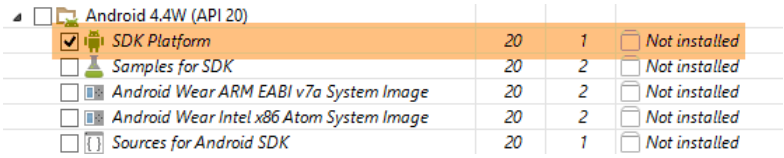


Faire l'installation du SDK Manager. Nous noterons par la suite <sdk-manager-install> son répertoire d'installation. Lancez-le.

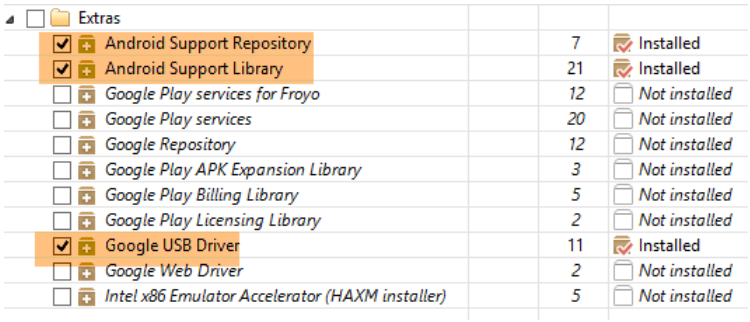
3

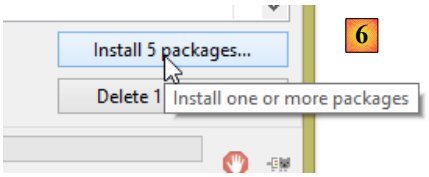


4



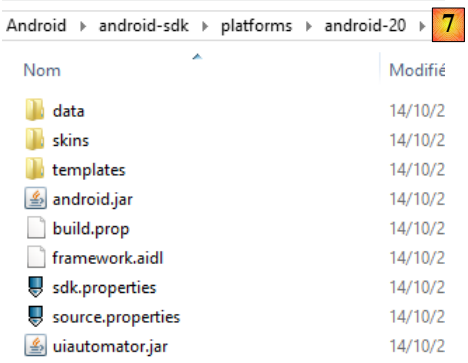
5





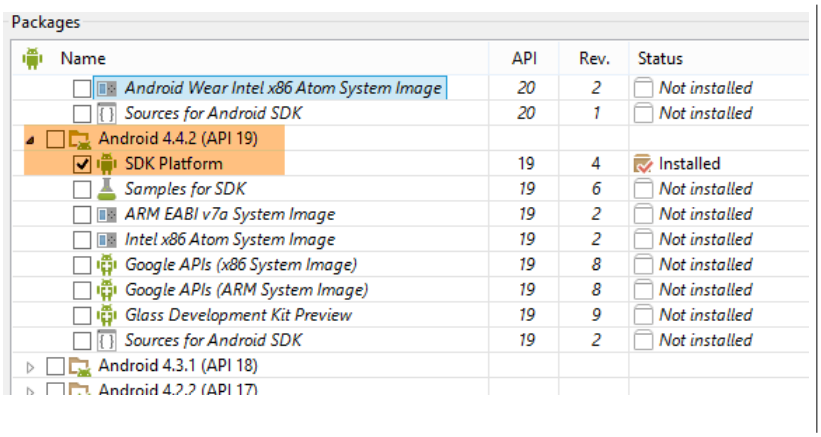
6

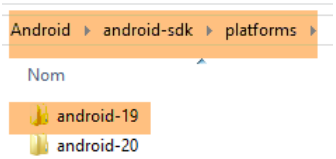
7



- en [3-6], téléchargez les packages sélectionnés par défaut ou comme dans les copies d'écran ci-dessus, choisissez ce qui est indispensable ;
- en [7], dans <sdk-manager-install>/platforms, l'API 20 d'Android a été ici installée ;

Téléchargez également l'API 19 :





Les exemples ont été construits avec les éléments définis précédemment :

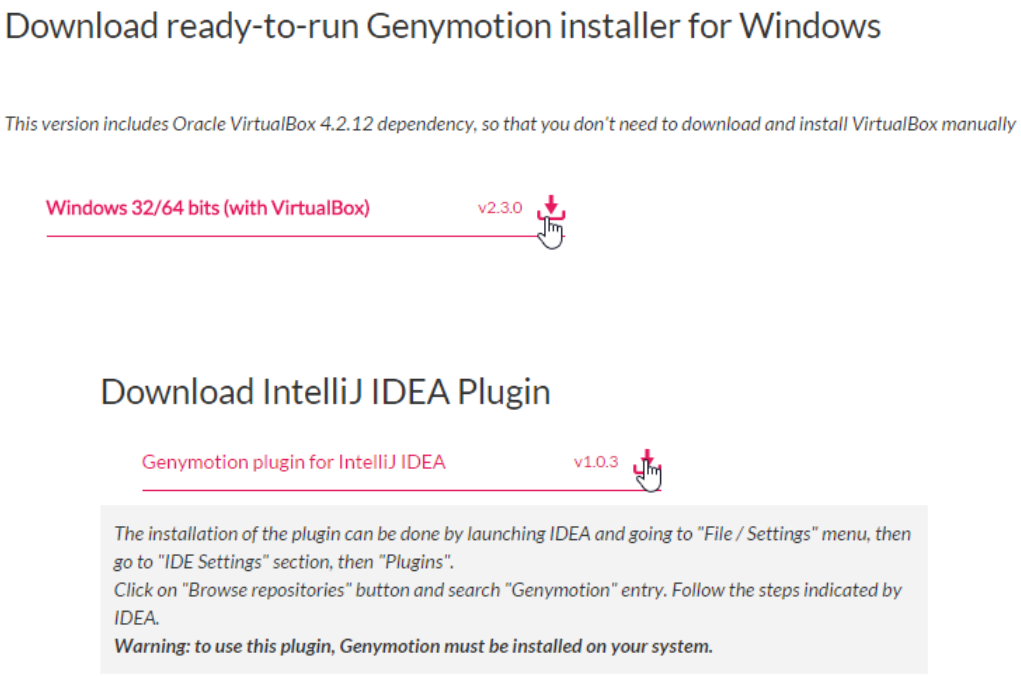
- JDK 1.8 ;
- Android SDK 20 pour l'exécution, SDK 19 pour l'affichage des vues dans l'IDE ;
- Android Build Tools version 20.0.0 ;
- Android Support Repository, version 6 ;
- Android Support Library, version 20 ;

Assurez-vous d'avoir téléchargé ces éléments.

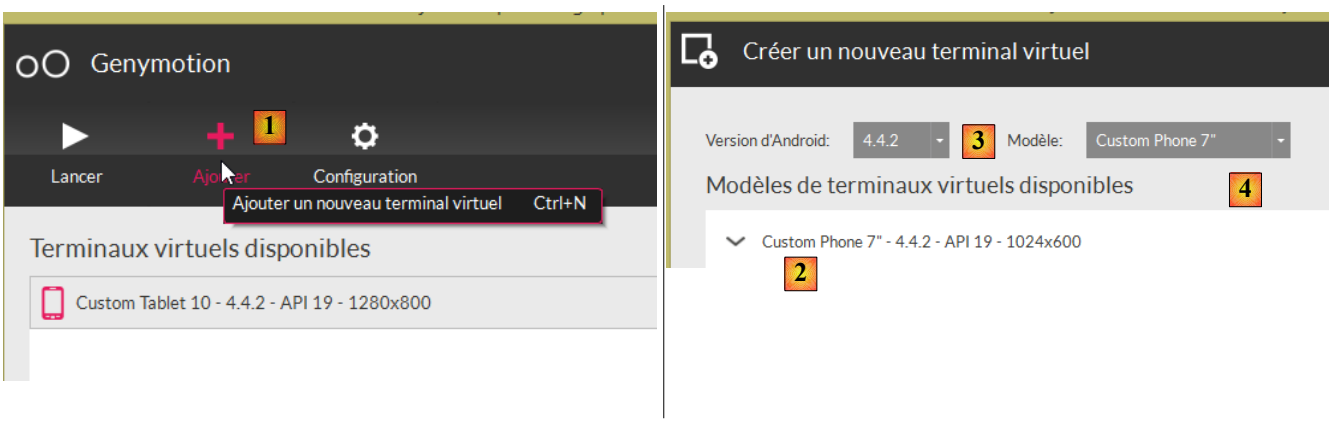
1.16.4 Installation du gestionnaire d'émulateurs Genymotion

Les émulateurs fournis avec le SDK d'Android sont lents ce qui décourage de les utiliser. L'entreprise [Genymotion] offre un émulateur performant. Celui-ci est disponible à l'URL [https://cloud.genymotion.com/page/launchpad/download/] (octobre 2014).

Vous aurez à vous enregistrer pour obtenir une version à usage personnel. Téléchargez le produit [Genymotion] avec la machine virtuelle VirtualBox ainsi que le plugin [Genymotion] pour l'IDE [IntellijIDEA] :



Nous appellerons par la suite <genymotion-install> le dossier d'installation de [Genymotion]. Lancez [Genymotion]. Téléchargez ensuite une image pour une tablette ou un téléphone :

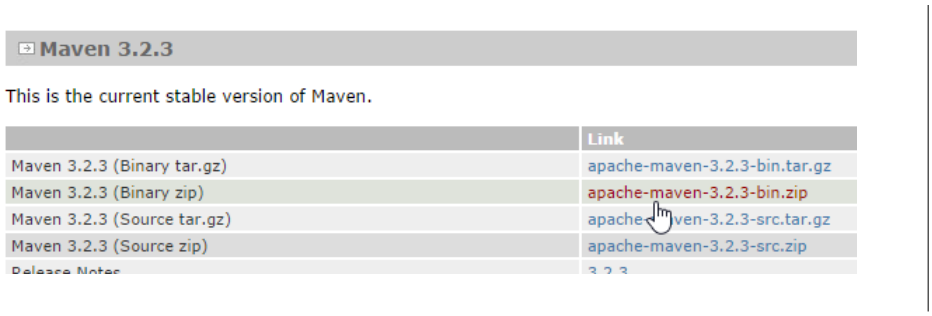


- en [1], ajoutez un terminal virtuel ;
- en [2], choisissez un ou plusieurs terminaux à installer. Vous pouvez affiner la liste affichée en précisant la version d'Android désirée [3] ainsi que le modèle de terminal [4] ;

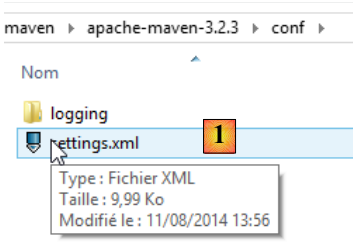
- une fois le téléchargement terminé, vous obtenez en [5] la liste des terminaux virtuels dont vous disposez pour tester vos applications Android ;

1.16.5 Installation de Maven

Maven est un outil de gestion des dépendances d'un projet Java et plus encore. Il est disponible à l'URL [http://maven.apache.org/download.cgi].



Téléchargez et dézippez l'archive. Nous appellerons <maven-install> le dossier d'installation de Maven.



- en [1], le fichier [conf / settings.xml] configure Maven ;

On y trouve les lignes suivantes :

```
1.    <!-- localRepository
2.     | The path to the local repository maven will use to store artifacts.
3.     |
4.     | Default: ${user.home}/.m2/repository
5.    <localRepository>/path/to/local/repo</localRepository>
6.  -->
```

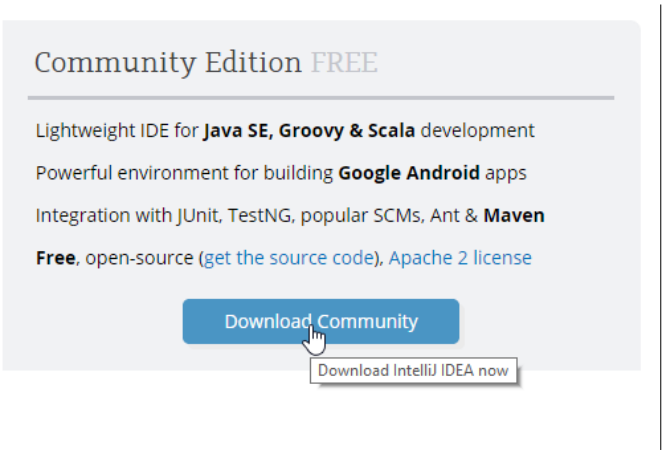
La valeur par défaut de la ligne 4, si comme moi votre {user.home} a un espace dans son chemin (par exemple [C:\Users\Serge Tahé]), peut poser problème à certains logiciels dont IntellijIDEA. On écrira alors quelque chose comme :

```
1.    <!-- localRepository
2.     | The path to the local repository maven will use to store artifacts.
3.     |
4.     | Default: ${user.home}/.m2/repository
5.    <localRepository>/path/to/local/repo</localRepository>
6.    -->
7.  <localRepository>D:\Programs\devjava\maven\.m2\repository</localRepository>
```

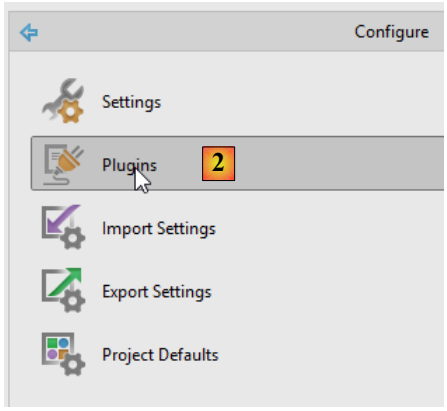
et on évitera, ligne 7, un chemin qui contient des espaces.

1.16.6 Installation de l'IDE IntellijIDEA Community Edition

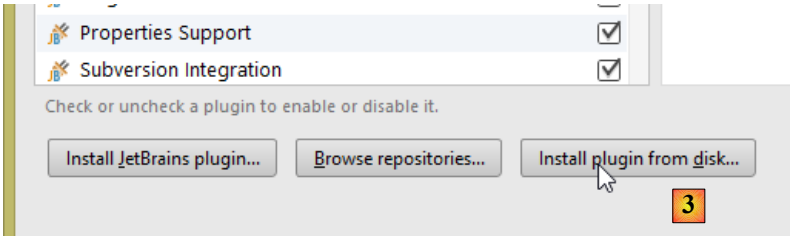
L'IDE IntellijIDEA Community Edition est disponible à l'URL [https://www.jetbrains.com/idea/download/] (octobre 2014) :

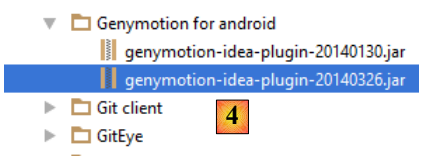


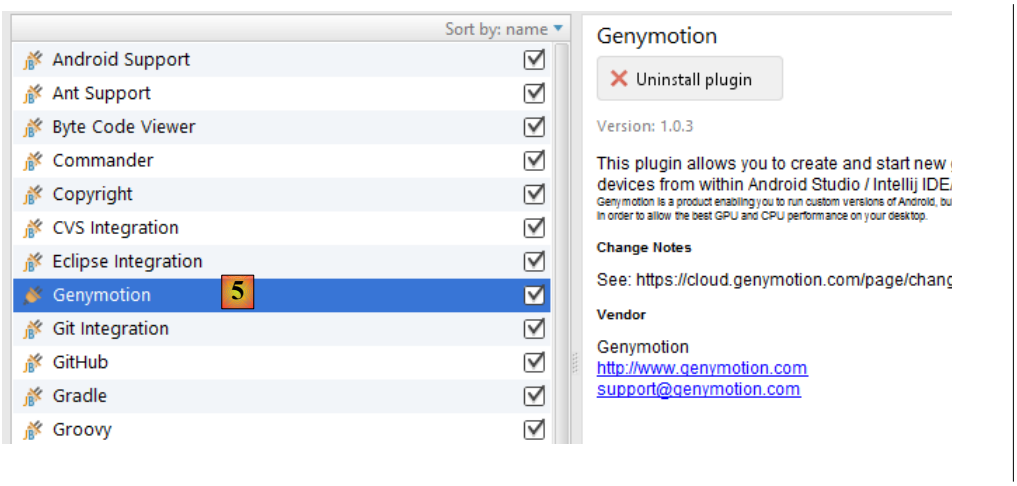
Installez l'IDE puis lancez-le.



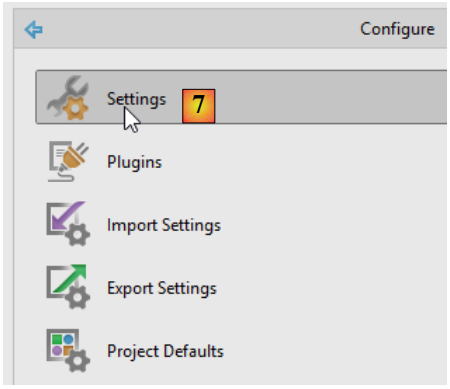
- en [1-2], configurez les plugins ;



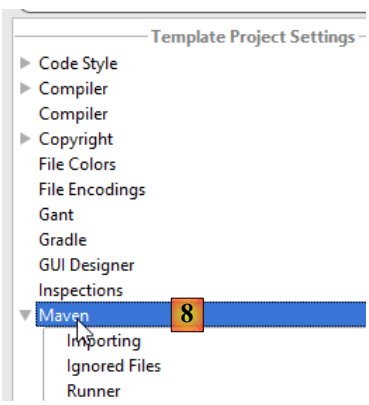


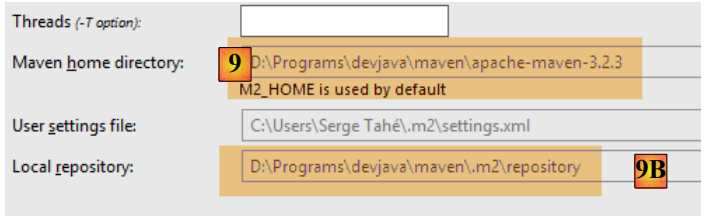


- en [3-5], ajoutez à l'IDE le plugin [Genymotion] téléchargé précédemment.

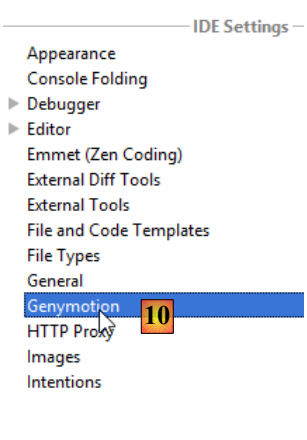


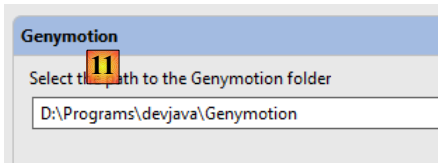
- en [6-7], configurez l'IDE ;



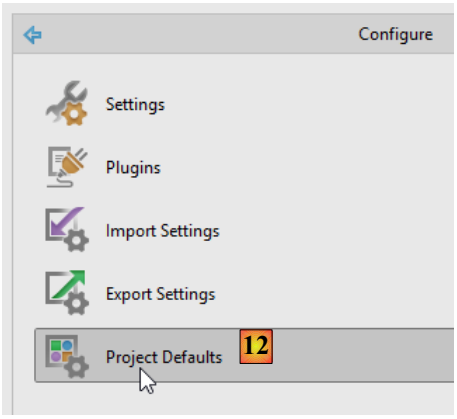


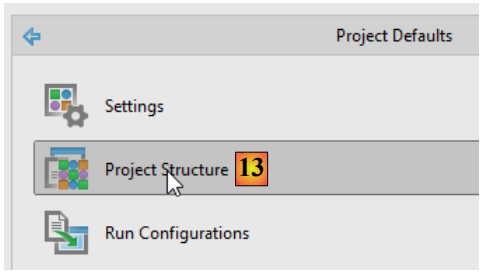
- en [8-9], indiquez le dossier d'installation de Maven ;
- en [9B], vérifiez bien qu'il n'y a pas d'espaces dans le chemin du dépôt local de Maven. La valeur [9B] prise par défaut provient du fichier [<maven-install> / conf / settings.xml] ;



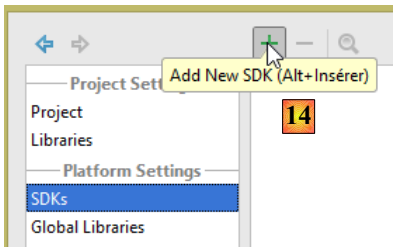


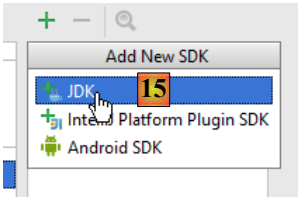
- en [10-11], indiquez le dossier d'installation du gestionnaire d'émulateurs [Genymotion] ;

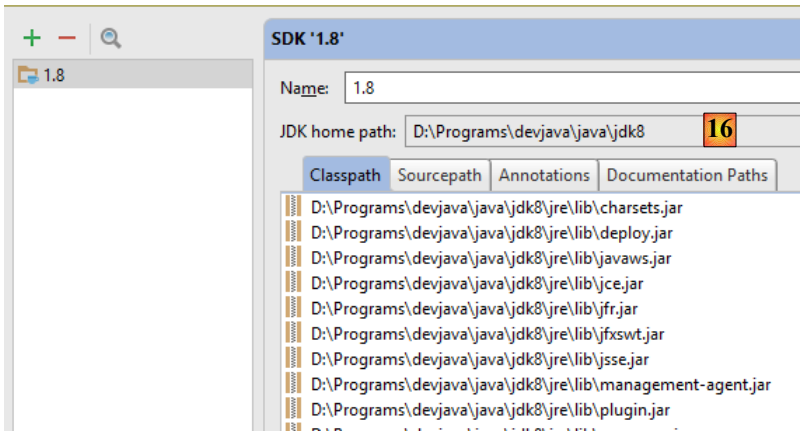




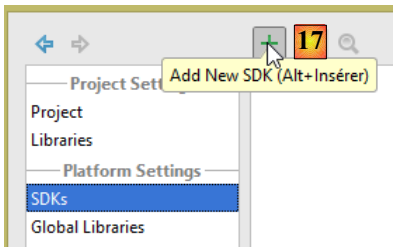
- en [12-13], on configure la nature par défaut des projets ;

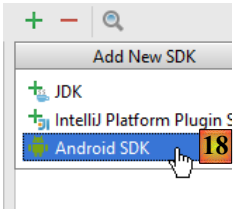


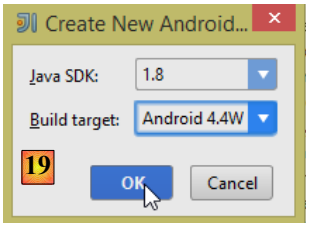


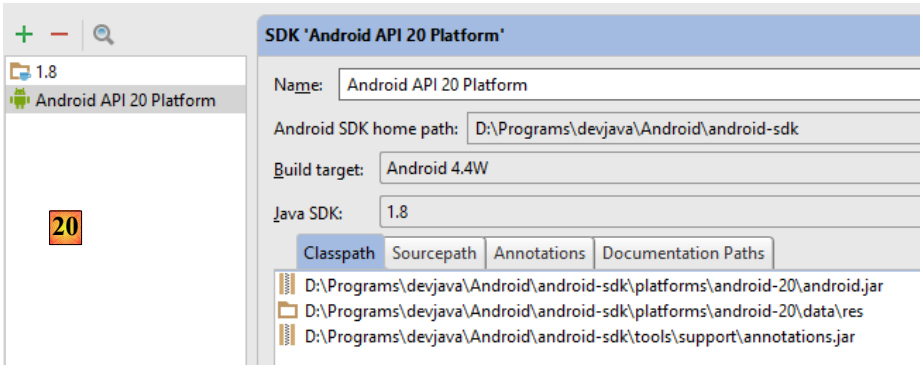


- en [14-16], on configure le JDK ;

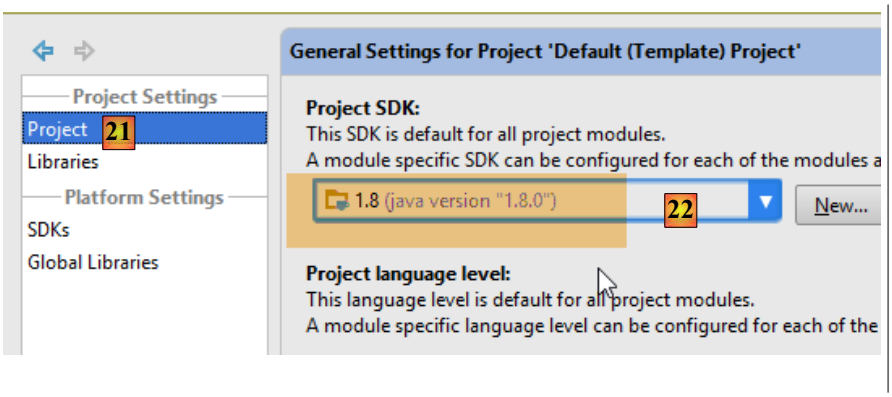




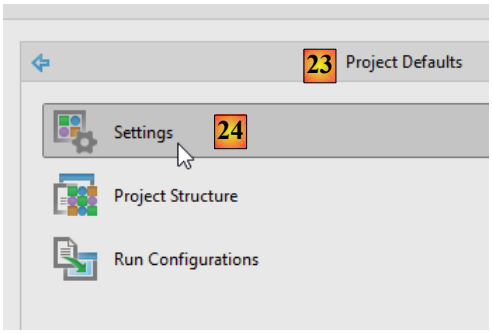


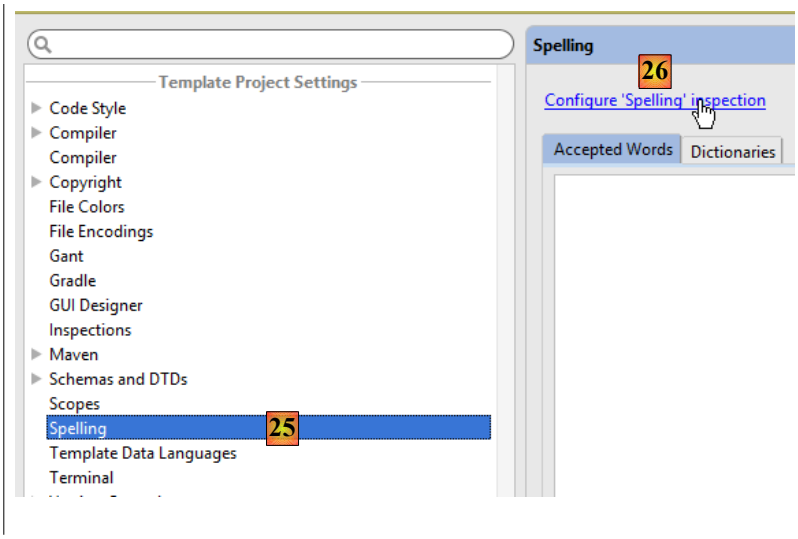


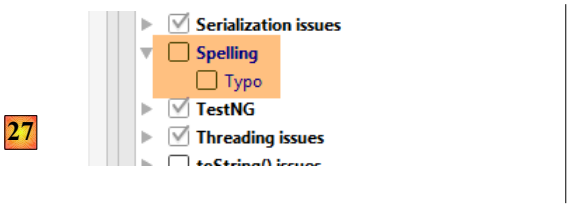
- en [17-20], on configure le SDK d'Android ;



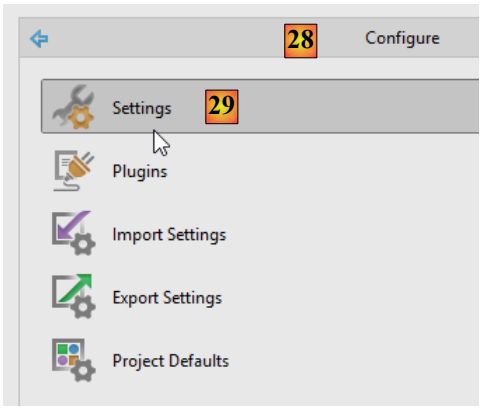
- en [21-22], on indique le JDK par défaut des projets ;

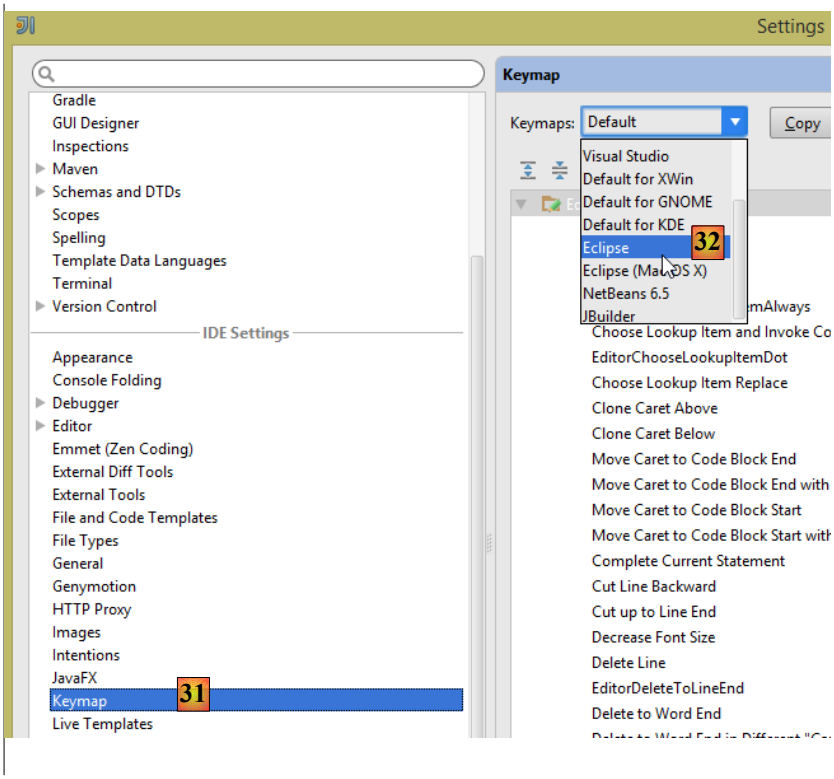




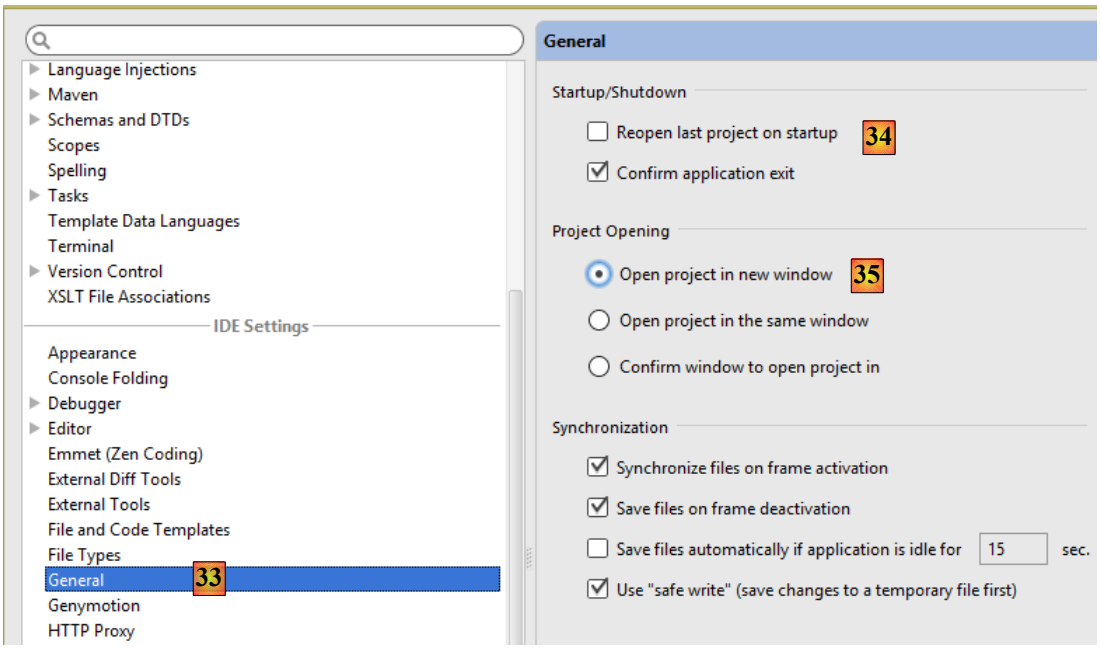


- en [23-27], on désactive la correction orthographique qui par défaut est pour la langue anglaise ;

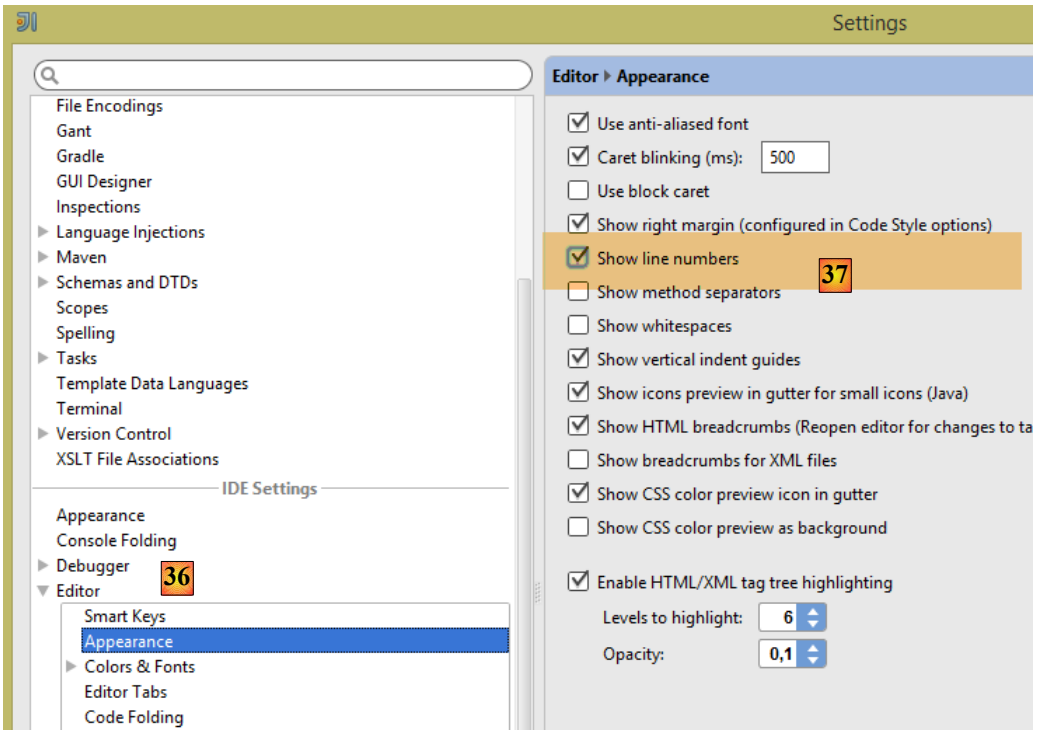




- en [29-32], choisissez le type de raccourcis clavier que vous souhaitez. Vous pouvez garder celui par défaut d'Intellij ou choisir celui d'un autre IDE auquel vous seriez plus habitués ;



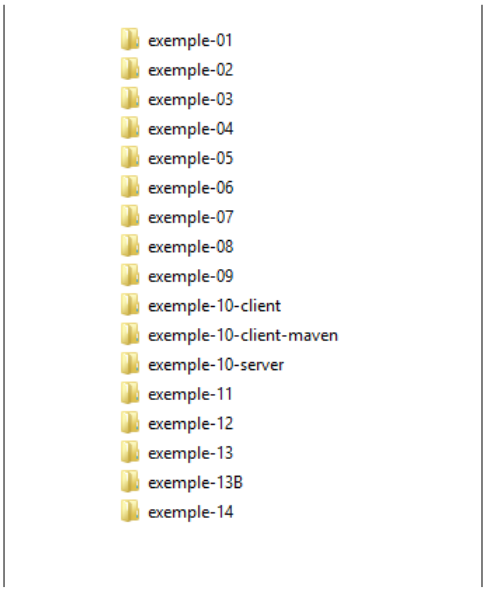
- en [33-35], configurez l'IDE vis à vis de multiples projets. Il peut gérer plusieurs projets dans la même fenêtre ou dans des fenêtre différentes ;



- en [36-37], par défaut numérotez les lignes. Cela vous permettra de retrouver rapidement la ligne qui a provoqué une exception ;

1.16.7 Utilisation des exemples

Les projets IntellijIDEA des exemples sont disponibles à l'URL [http://tahe.ftp-developpez.com/fichiers-archive/android-exemples-intellij-aa.zip]. Téléchargez-les.

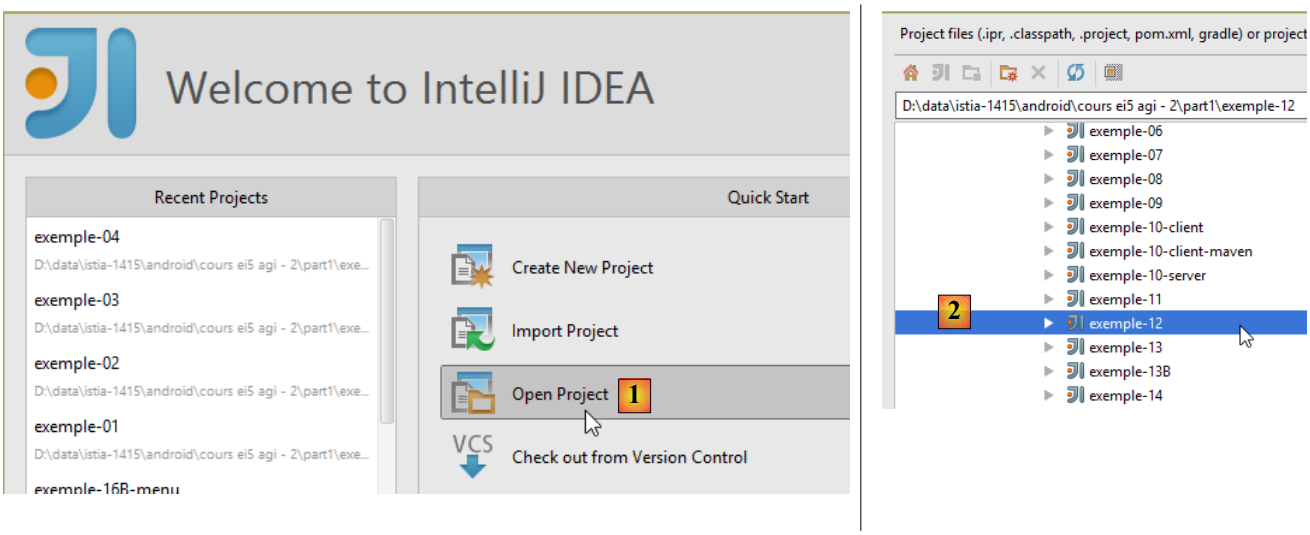


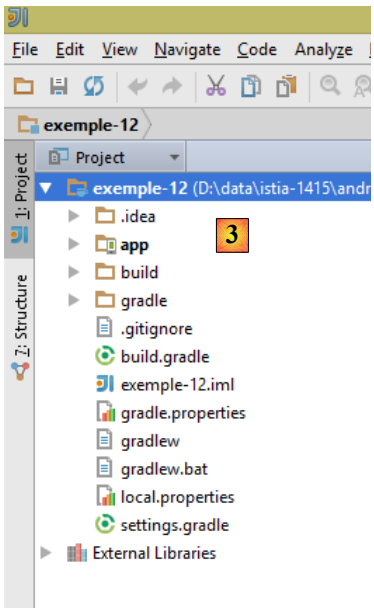
Les exemples ont été construits avec les éléments définis précédemment :

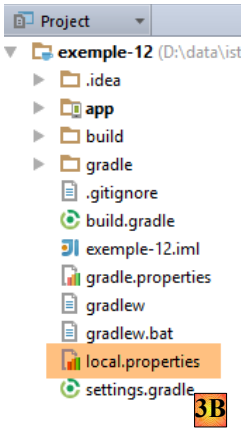
- JDK 1.8 ;
- Android SDK 20 pour l'exécution, SDK 19 pour l'affichage des vues dans l'IDE ;
- Android Build Tools version 20.0.0 ;
- Android Support Repository, version 6 ;
- Android Support Library, version 20 ;

Si votre environnement ne correspond pas au précédent, vous aurez à changer la configuration des projets. Cela peut être assez pénible. Dans un premier temps, il est probablement plus facile de reconstituer un environnement de travail analogue à celui décrit précédemment.

Lancez IntellijIDEA puis ouvrez le projet [exemple-12] par exemple :

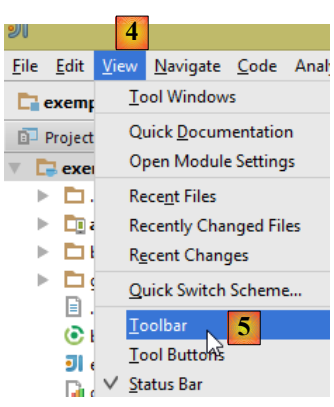


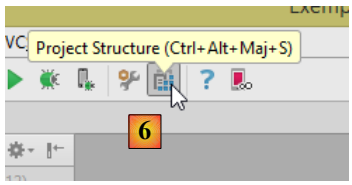




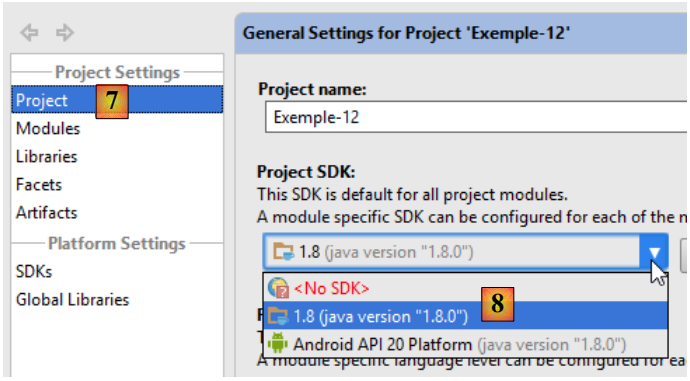
- en [1-3], on ouvre le projet [exemple-12] ;
- en [3B], on vérifie le fichier [local.properties] ;

- ligne 10 ci-dessus, mettez l'emplacement du SDK Manager d'Android <sdk-manager-install> (cf page 179) ;

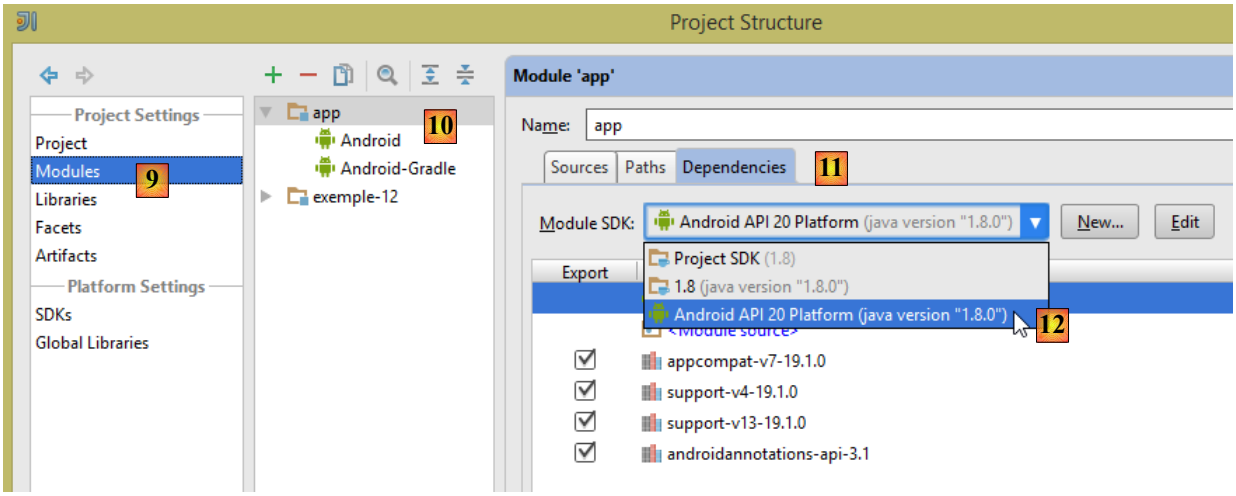




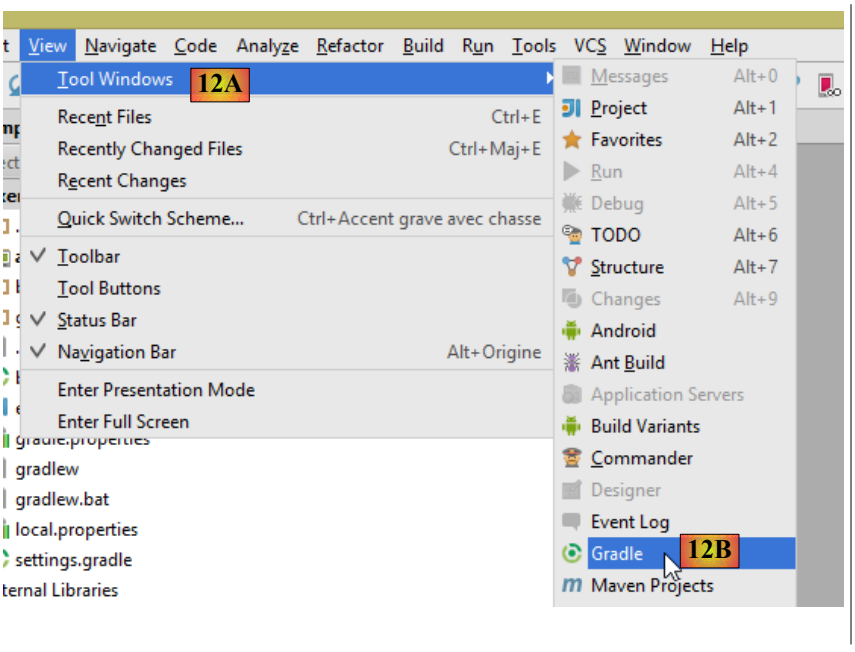
- en [4-5], on fait afficher la barre d'outils ;
- en [6], on accède à la structure du projet ;

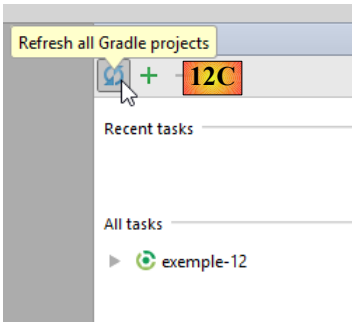


- en [7-8], on choisit un JDK (1.8 dans l'exemple) ;



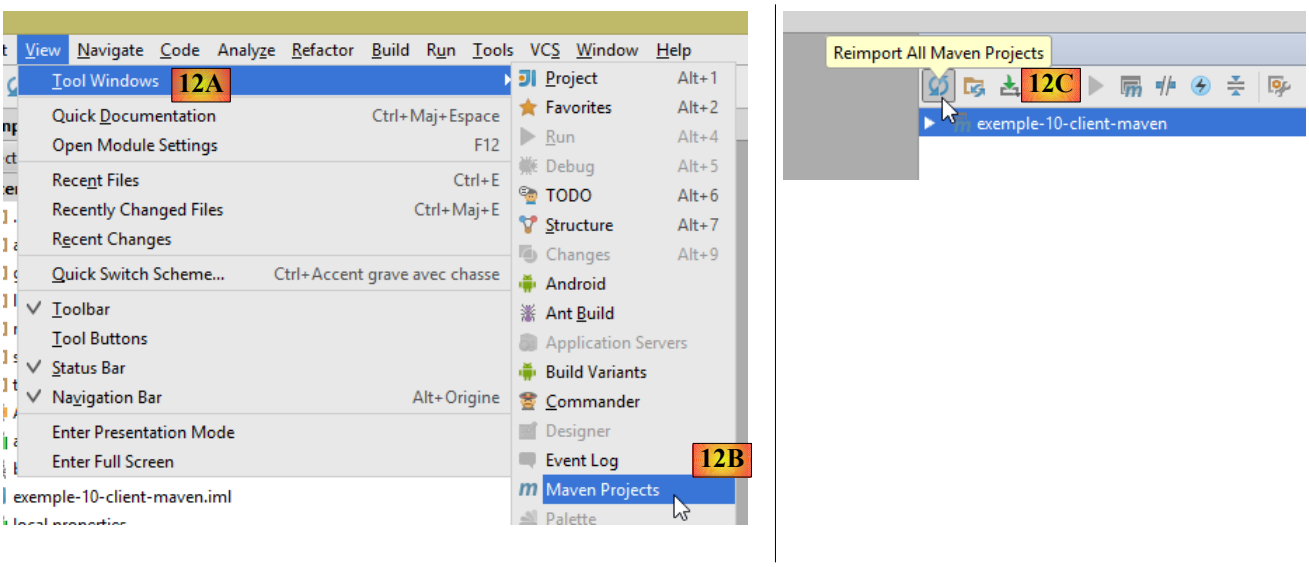
- en [9-12], on choisit un SDK d'Android (l'API 20 dans l'exemple) ;



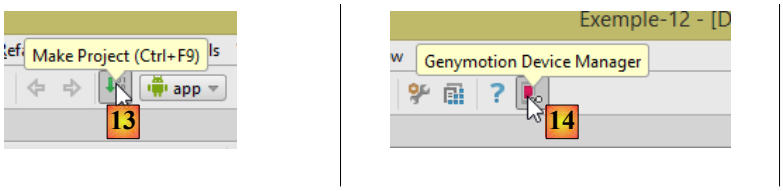


- en [12A-12C], on rafraîchit le projet Gradle [exemple-12]. Certains projets ne sont pas des projets Gradle et cette étape n'est alors pas à faire ;

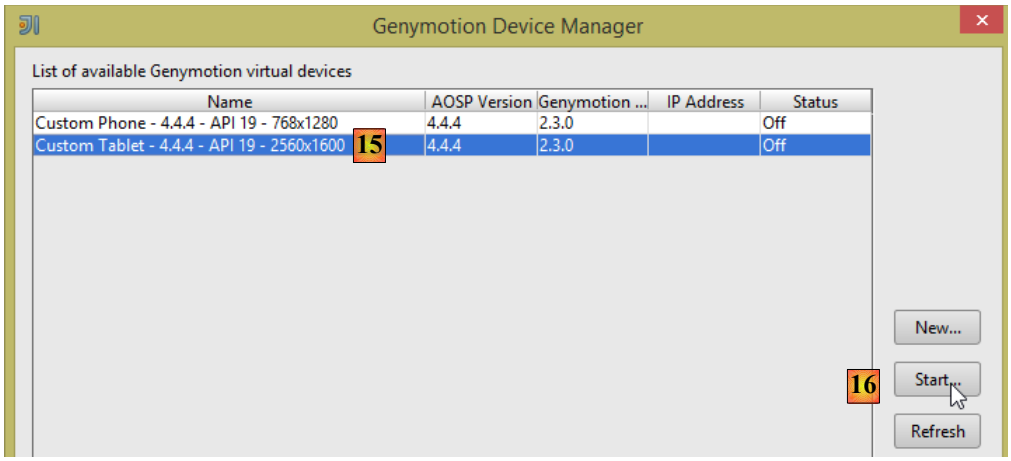
Pour un projet Maven, vous procéderez de la façon suivante :



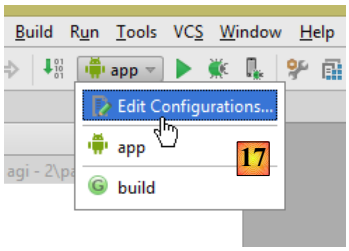
- en [12A-12C], on rafraîchit le projet Maven ;

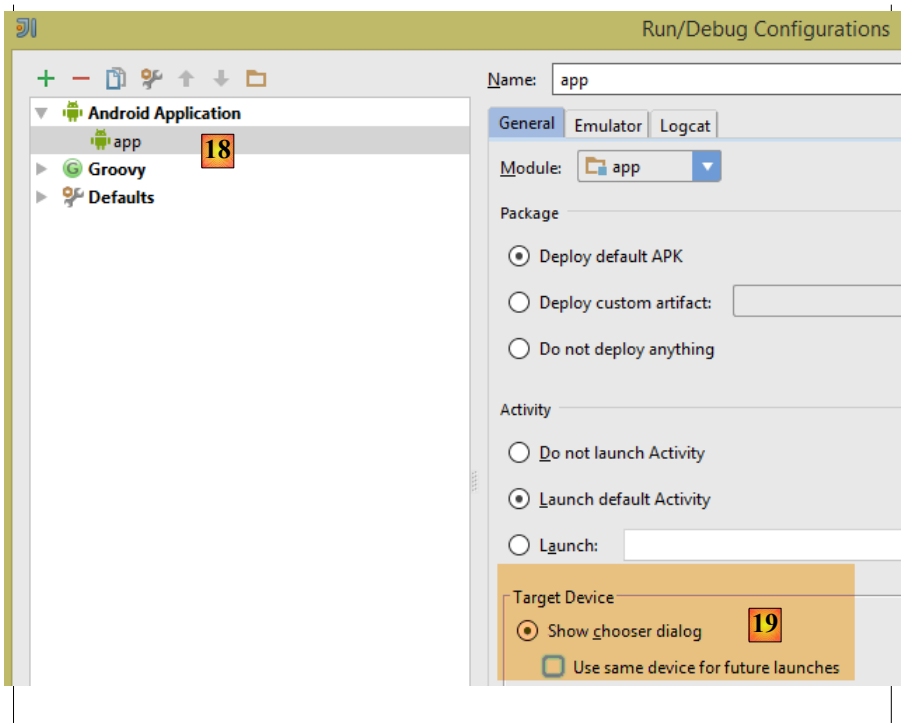


- en [13], on compile (Build, Make) le projet (souvent, il faut faire deux Make pour avoir une compilation sans erreurs) ;
- en [14], on lance le gestionnaire des émulateurs Android ;

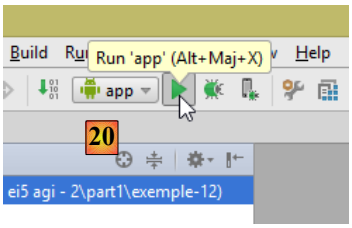


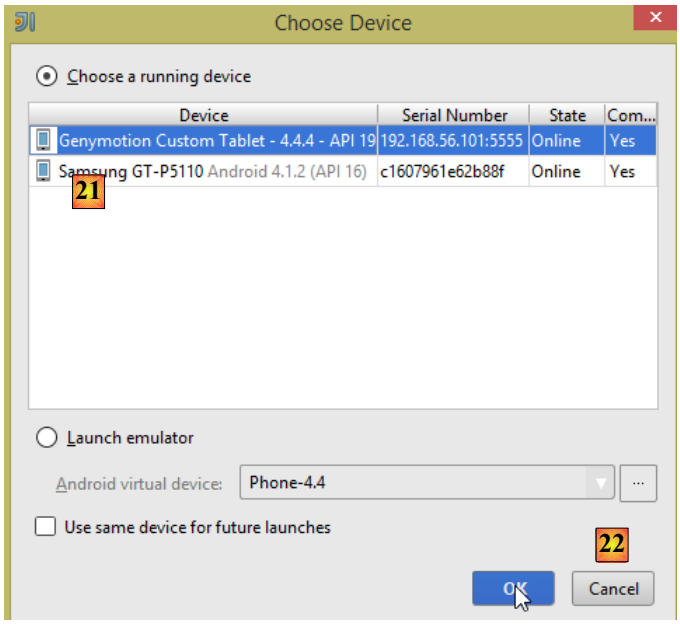
- en [15-16], on sélectionne un émulateur de tablette ;



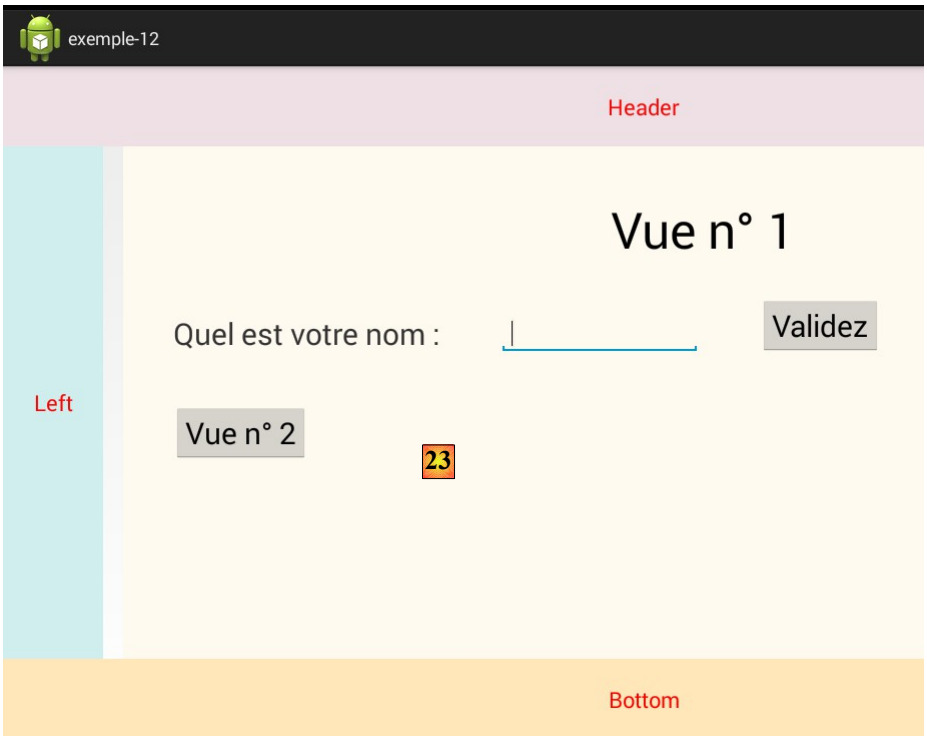


- en [17], on édite la configuration d'exécution ;
- en [18-19], on s'assure que dans la configuration d'exécution nommée [app], le périphérique d'exécution va être demandé à chaque fois (19). Par défaut ce n'est pas le cas ;



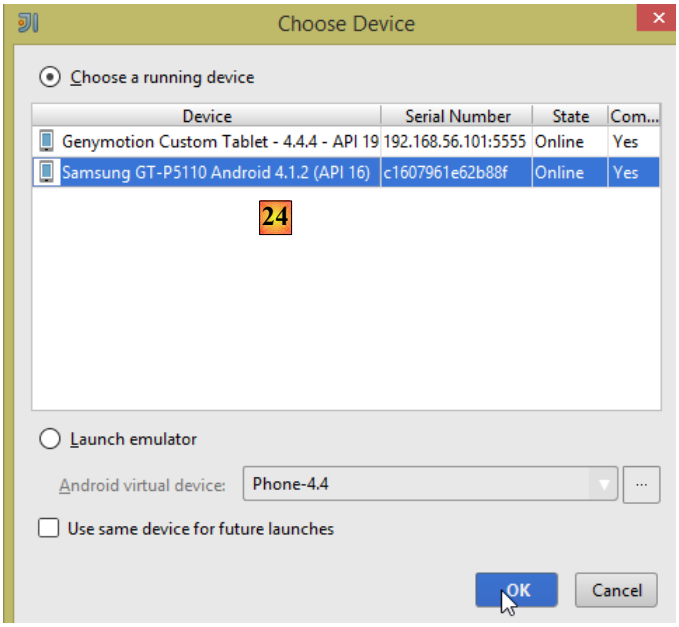


- en [20], on lance l'application ;
- en [21-22], on sélectionne l'émulateur de la tablette ;



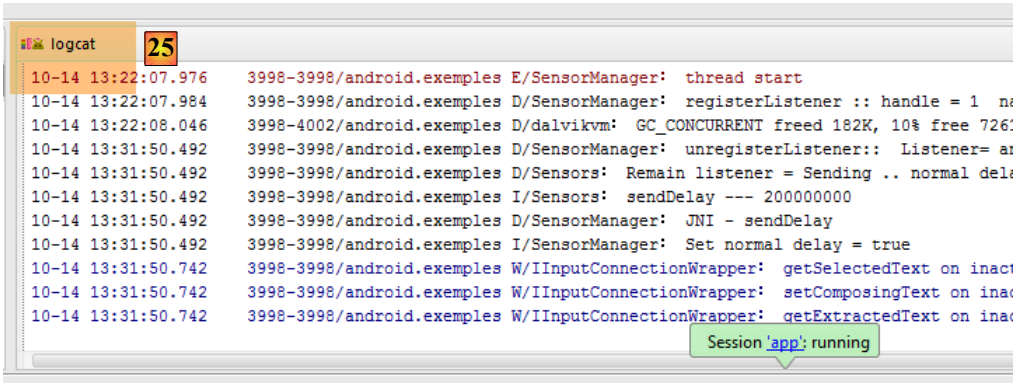
- en [23], la vue affichée par l'émulateur ;

Branchez maintenant une tablette Android sur un port USB du PC et exécutez l'application sur celle-ci :



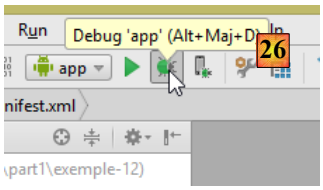
- en [24], sélectionnez la tablette Android et testez l'application.

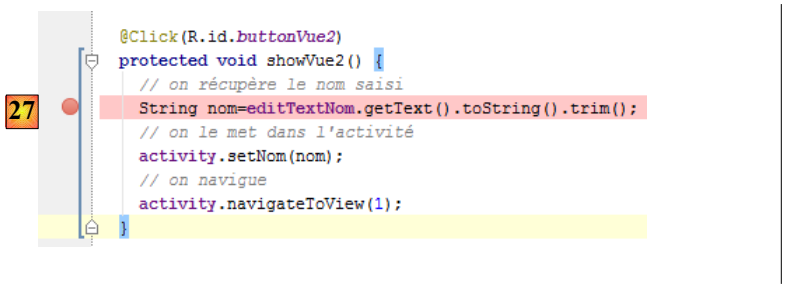
Dans chaque cas, des logs sont affichés dans la fenêtre appelée [Logcat] :



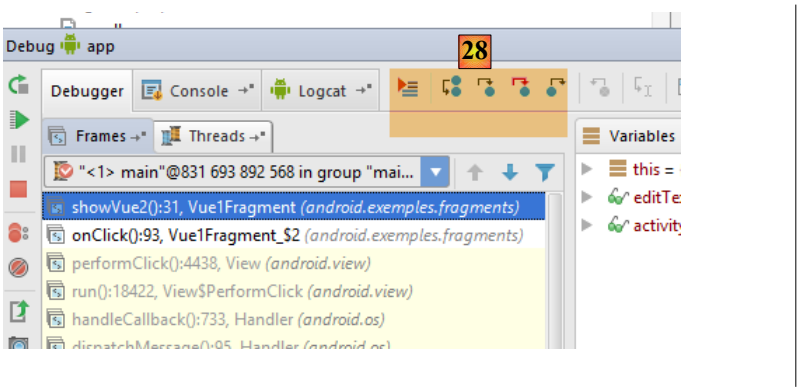
Consultez régulièrement ces logs. C'est là que seront signalées les exceptions qui feront planter votre programme.

Vous pouvez également déboguer votre programme [26] avec les outils habituels du débogage :





- en [27], on met un point d'arrêt en cliquant une fois sur la colonne à gauche de la ligne cible. Un nouveau clic annule le point d'arrêt ;

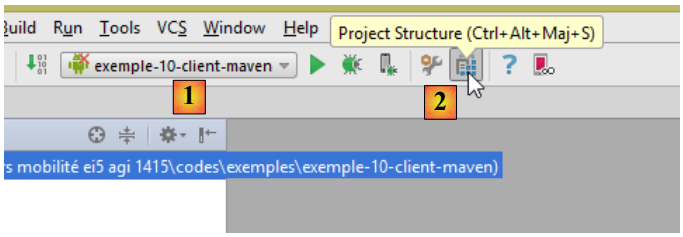


- en [28, au point d'arrêt faire :
    - [F6], pour exécuter la ligne sans entrer dans les méthodes si la ligne contient des appels de méthodes,
    - [F5], pour exécuter la ligne en entrant dans les méthodes si la ligne contient des appels de méthodes,
    - [F8], pour continuer jusqu'au prochain point d'arrêt ;
    - [Ctrl-F2] pour arrêter le débogage ;

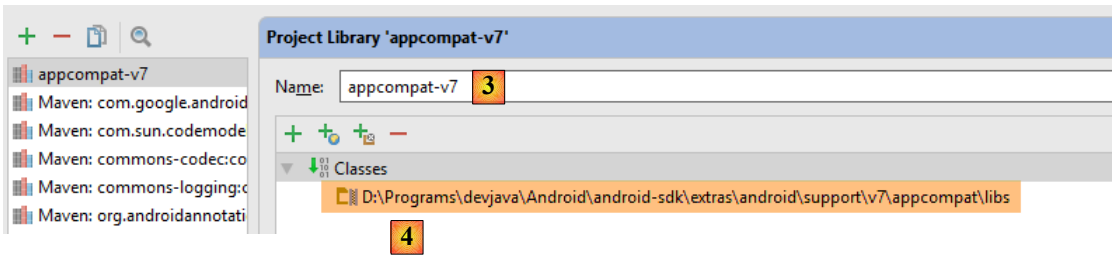
Vous reproduirez la démarche du projet [exemple-12] pour les exemples :

- Gradle 6-14 ;
- Maven 2-5 ;

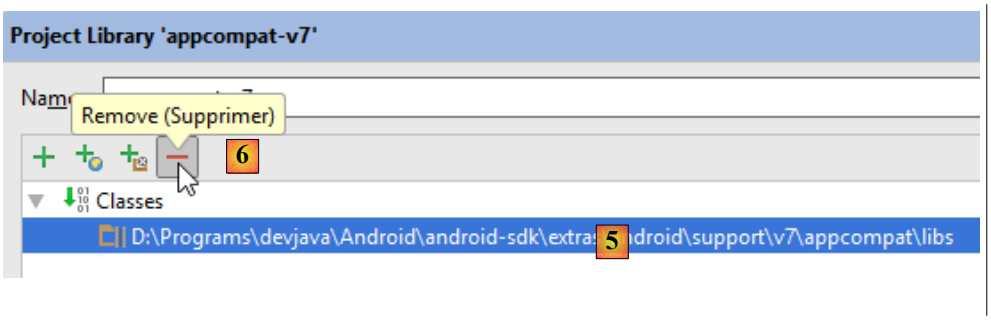
Pour le projet Maven [10-client-maven], il y a une démarche supplémentaire à faire :



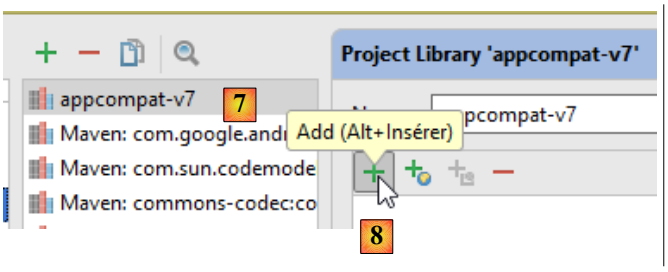
- en [1-2], accédez à la structure du projet ;



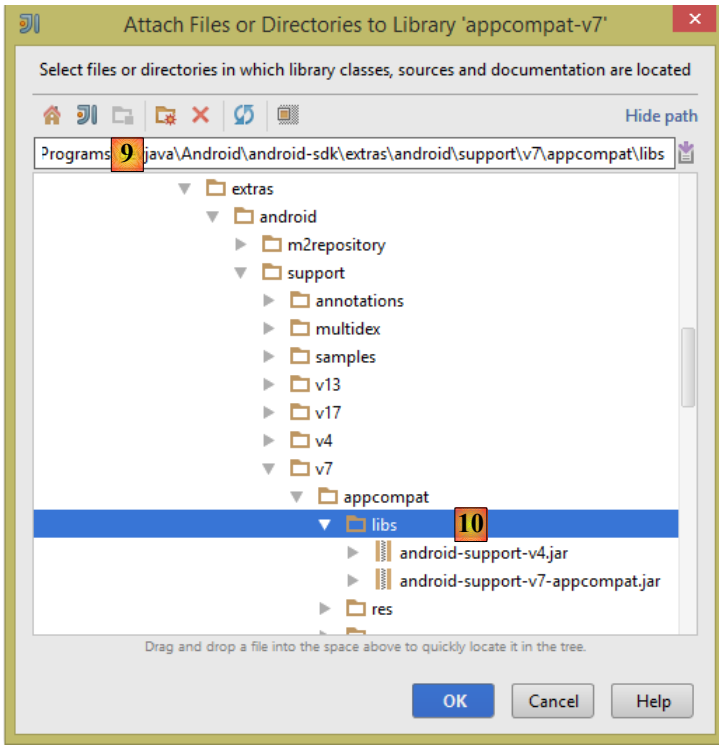
- en [3-4] est définie une bibliothèque [appcompat-v7] [3] constituée par les archives jar du dossier [4]. Ce dernier ne sera probablement pas le même sur votre poste. Procédez alors ainsi :



- en [5-6], supprimez le dossier erroné ;



- en [7-8], ajoutez une archive ;

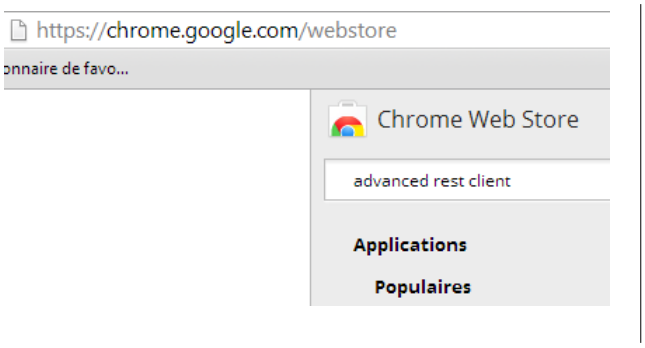


- en [9-10], désignez le dossier [<sdk-manager-install>/extras/android/support/v7/appcompat/libs] ;

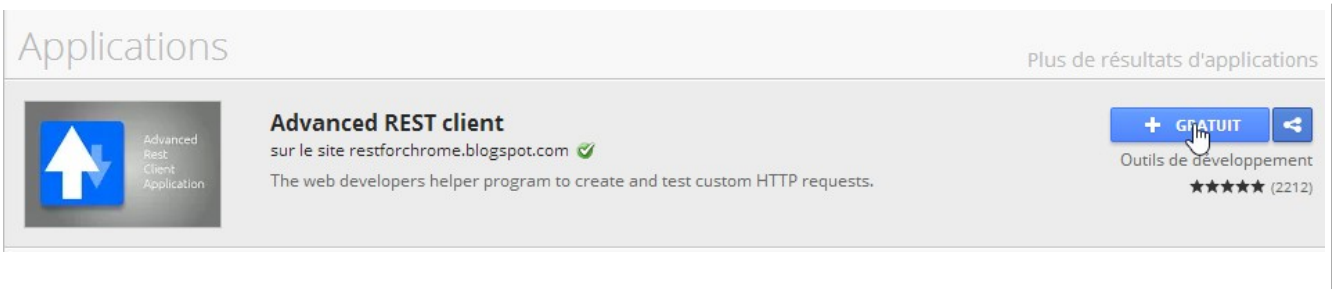
1.16.8 Installation du plugin Chrome [Advanced Rest Client]

Dans ce document, on utilise le navigateur **Chrome** de Google (http://www.google.fr/intl/fr/chrome/browser/). On lui ajoutera l'extension [Advanced Rest Client]. On pourra procéder ainsi :

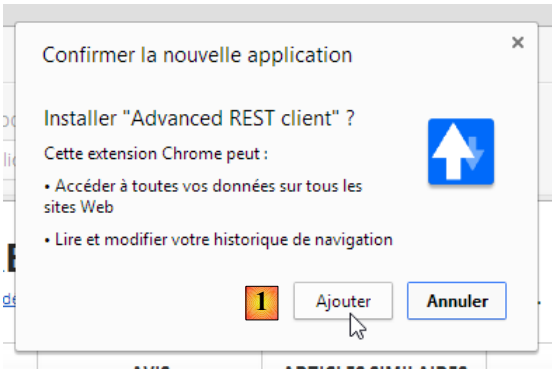
- aller sur le site de [Google Web store] (https://chrome.google.com/webstore) avec le navigateur Chrome ;
- chercher l'application [Advanced Rest Client] :

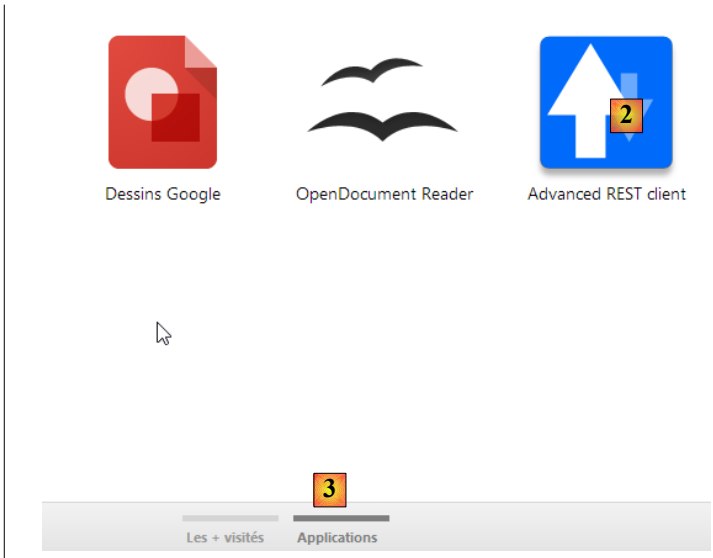


- l'application est alors disponible au téléchargement :



- pour l'obtenir, il vous faudra créer un compte Google. [Google Web Store] demande ensuite confirmation [1] :





- en [2], l'extension ajoutée est disponible dans l'option [Applications] [3]. Cette option est affichée sur chaque nouvel onglet que vous créez (CTRL-T) dans le navigateur.

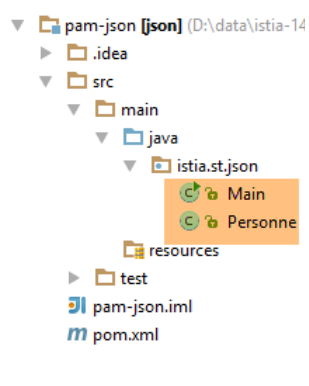
1.16.9 Gestion du jSON en Java

De façon transparente pour le développeur le framework [Spring MVC] utilise la bibliothèque jSON [Jackson]. Pour illustrer ce qu'est le jSON (JavaScript Object Notation), nous présentons ici un programme qui sérialise des objets en jSON et fait l'inverse en désérialisant les chaînes jSON produites pour recréer les objets initiaux.

La bibliothèque 'Jackson' permet de construire :

- la chaîne jSON d'un objet : *new ObjectMapper().writeValueAsString(object)* ;
- un objet à partir d'un chaîne jSON : *new ObjectMapper().readValue(jsonString, Object.class)*.

Les deux méthodes sont susceptibles de lancer une *IOException*. Voici un exemple.



Le projet ci-dessus est un projet Maven avec le fichier [pom.xml] suivant ;

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <project xmlns="http://maven.apache.org/POM/4.0.0"
3.           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
    4.0.0.xsd">
5.    <modelVersion>4.0.0</modelVersion>
6.
7.    <groupId>istia.st.pam</groupId>
8.    <artifactId>json</artifactId>
9.    <version>1.0-SNAPSHOT</version>
10.
11.   <dependencies>
12.     <dependency>
13.       <groupId>com.fasterxml.jackson.core</groupId>
14.       <artifactId>jackson-databind</artifactId>
15.       <version>2.3.3</version>
16.     </dependency>
17.   </dependencies>
18. </project>
```

- lignes 12-16 : la dépendance qui amène la bibliothèque 'Jackson' ;

La classe [Personne] est la suivante :

```
1.  package istia.st.json;
2.
3.  public class Personne {
4.     // data
5.     private String nom;
6.     private String prenom;
7.     private int age;
8.
9.     // constructeurs
10.    public Personne() {
11.
12.    }
13.
14.    public Personne(String nom, String prénom, int âge) {
15.       this.nom = nom;
16.       this.prenom = prénom;
17.       this.age = âge;
18.    }
19.
20.    // signature
21.    public String toString() {
22.       return String.format("Personne[%s, %s, %d]", nom, prenom, age);
23.    }
24.
25.    // getters et setters
26. ...
27. }
```

La classe [Main] est la suivante :

```
1.  package istia.st.json;
2.
3.  import com.fasterxml.jackson.databind.ObjectMapper;
4.
5.  import java.io.IOException;
6.  import java.util.HashMap;
7.  import java.util.Map;
8.
9.  public class Main {
10.   // l'outil de sérialisation / désérialisation
11.   static ObjectMapper mapper = new ObjectMapper();
12.
13.   public static void main(String[] args) throws IOException {
14.     // création d'une personne
15.     Personne paul = new Personne("Denis", "Paul", 40);
16.     // affichage Json
17.     String json = mapper.writeValueAsString(paul);
18.     System.out.println("Json=" + json);
19.     // instanciation Personne à partir du Json
20.     Personne p = mapper.readValue(json, Personne.class);
21.     // affichage personne
22.     System.out.println("Personne=" + p);
23.     // un tableau
24.     Personne virginie = new Personne("Radot", "Virginie", 20);
25.     Personne[] personnes = new Personne[]{paul, virginie};
26.     // affichage Json
27.     json = mapper.writeValueAsString(personnes);
28.     System.out.println("Json personnes=" + json);
29.     // dictionnaire
30.     Map<String, Personne> hpersonnes = new HashMap<String, Personne>();
31.     hpersonnes.put("1", paul);
32.     hpersonnes.put("2", virginie);
33.     // affichage Json
34.     json = mapper.writeValueAsString(hpersonnes);
35.     System.out.println("Json hpersonnes=" + json);
36.   }
37. }
```

L'exécution de cette classe produit l'affichage écran suivant :

```
1.  Json={"nom":"Denis","prenom":"Paul","age":40}
2.  Personne=Personne[Denis, Paul, 40]
3.  Json personnes=[{"nom":"Denis","prenom":"Paul","age":40},
    {"nom":"Radot","prenom":"Virginie","age":20}]
4.  Json hpersonnes={"2":{"nom":"Radot","prenom":"Virginie","age":20},"1":
    {"nom":"Denis","prenom":"Paul","age":40}}
```

De l'exemple on retiendra :

- l'objet [ObjectMapper] nécessaire aux transformations jSON / Object : ligne 11 ;
- la transformation [Personne] --> jSON : ligne 17 ;
- la transformation jSON --> [Personne] : ligne 20 ;
- l'exception [IOException] lancée par les deux méthodes : ligne 13.

Introduction à la programmation

de tablettes Android par l'exemple

-

Partie 2 - Étude de cas

Serge Tahé, IstiA - université d'Angers
novembre 2014

2 Étude de cas

2.1 Le projet

Dans le document [Tutoriel AngularJS / Spring 4], a été développée une application client / serveur pour gérer des rendez-vous de médecins. Par la suite nous référencerons ce document [**rdvmedecins-angular**]. L'application avait deux types de client :

- un client HTML / CSS / JS ;
- un client Android ;

Le client Android était obtenu de façon automatique à partir de la version HTML du client avec l'outil [Cordova]. Le projet sera ici de recréer ce client Android à la main en utilisant les connaissances acquises dans les chapitres précédents.

On notera une différence importante entre les deux solutions :

- celle que nous allons créer ne sera utilisable que sur les tablettes Android ;
- dans la version [**rdvmedecins-angular**], le client web mobile (HTML / CSS / JS) est utilisable sur n'importe quelle plate-forme (Android, IoS, Windows) ;

2.2 Les vues du client Android

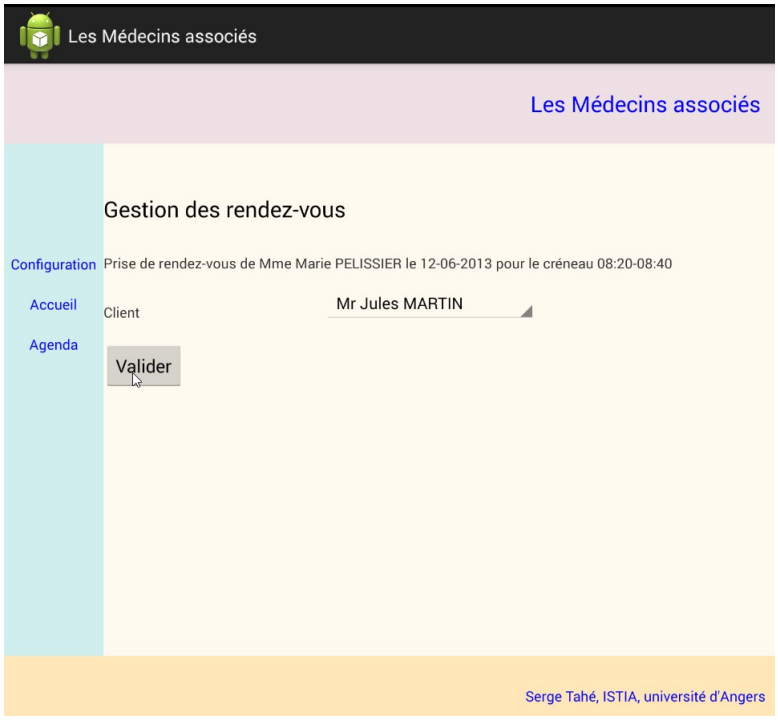
Il y a quatre vues.

Vue de configuration

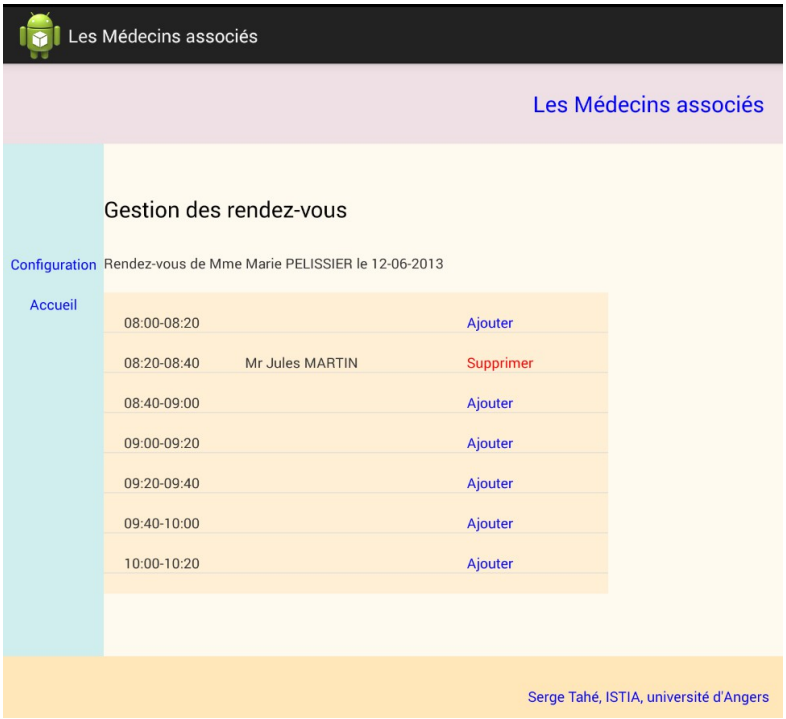
Vue de choix du médecin du rendez-vous

Vue de choix du créneau horaire du rendez-vous

Vue de choix du client du rendez-vous

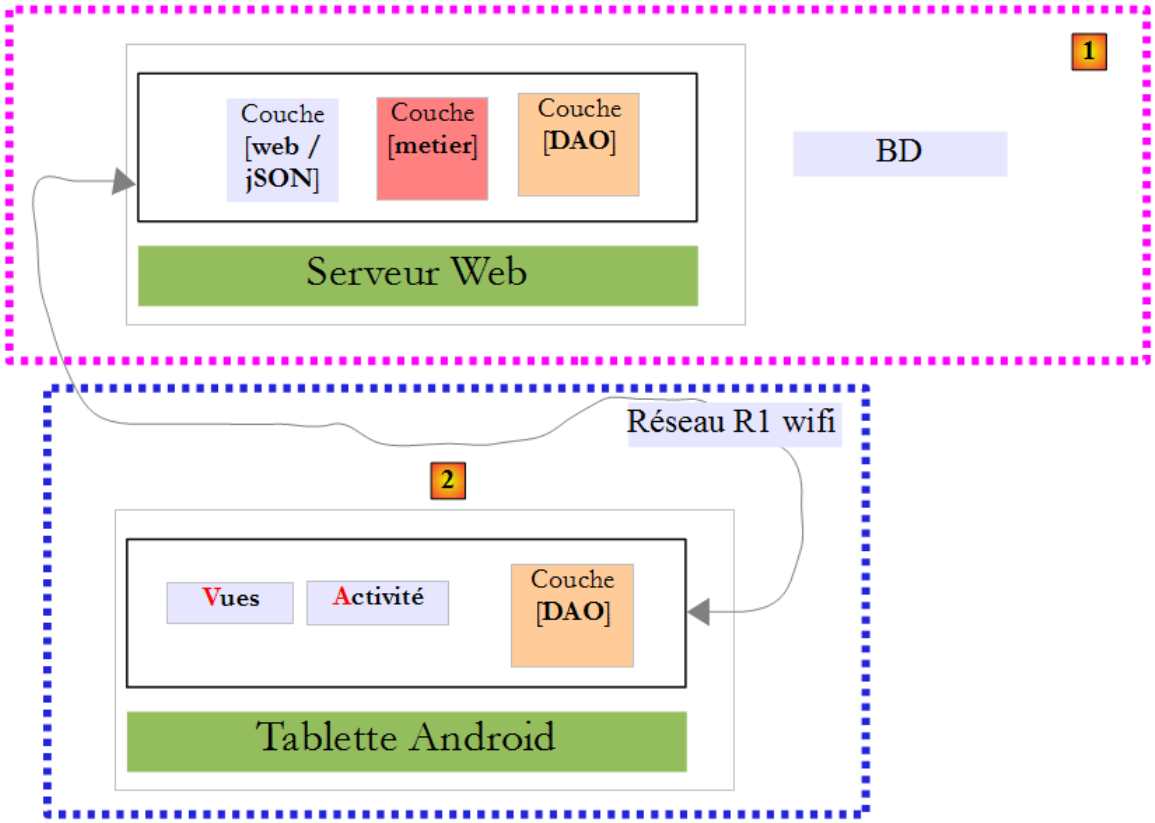


Après la prise de rendez-vous :



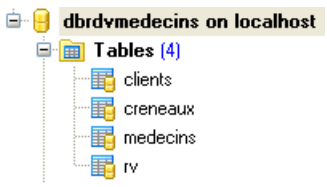
2.3 L'architecture du projet

On aura une architecture client / serveur analogue à celle de l'exemple [exemple-10] (cf paragraphe 1.11, page 103) de ce document :



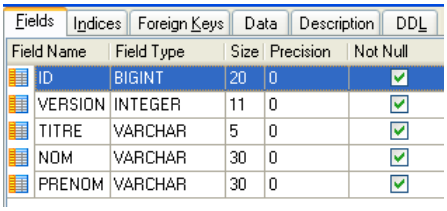
2.4 La base de données

Elle ne joue pas un rôle fondamental dans ce document. Nous la donnons à titre d'information. On l'appellera [dbrdvmedecins]. C'est une base de données MySQL5 avec quatre tables :

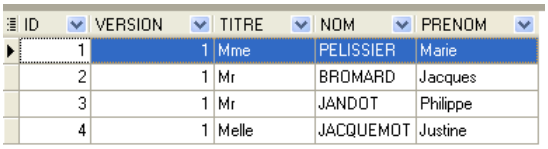


2.4.1 La table [MEDECINS]

Elle contient des informations sur les médecins gérés par l'application [RdvMedecins].



Field Name	Field Type	Size	Precision	Not Null
ID	BIGINT	20	0	☑
VERSION	INTEGER	11	0	☑
TITRE	VARCHAR	5	0	☑
NOM	VARCHAR	30	0	☑
PRENOM	VARCHAR	30	0	☑



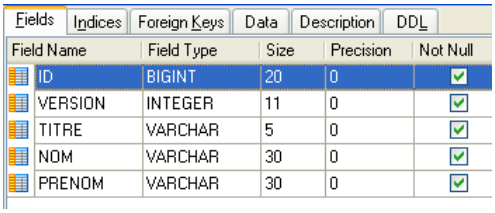
ID	VERSION	TITRE	NOM	PRENOM
1	1	Mme	PELISSIER	Marie
2	1	Mr	BROMARD	Jacques
3	1	Mr	JANDOT	Philippe
4	1	Melle	JACQUEMOT	Justine

- ID : n° identifiant le médecin - clé primaire de la table

- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du médecin
- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

2.4.2 La table [CLIENTS]

Les clients des différents médecins sont enregistrés dans la table [CLIENTS] :



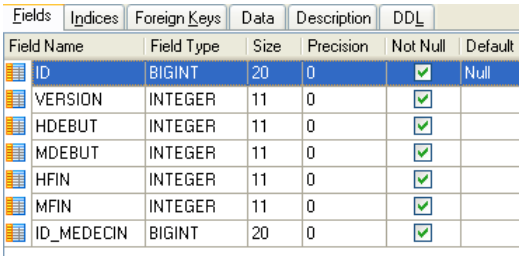
Field Name	Field Type	Size	Precision	Not Null
ID	BIGINT	20	0	☑
VERSION	INTEGER	11	0	☑
TITRE	VARCHAR	5	0	☑
NOM	VARCHAR	30	0	☑
PRENOM	VARCHAR	30	0	☑

ID	VERSION	TITRE	NOM	PRENOM
1	1	Mr	MARTIN	Jules
2	1	Mme	GERMAN	Christine
3	1	Mr	JACQUARD	Jules
4	1	Melle	BISTROU	Brigitte

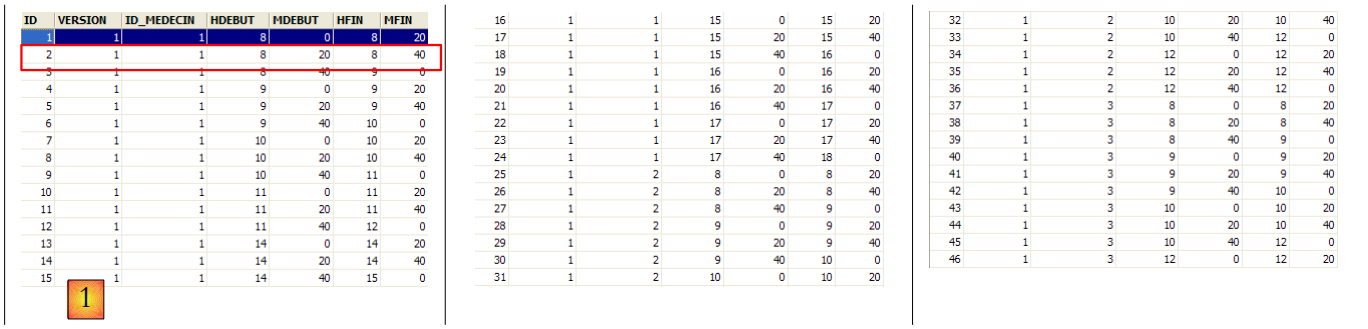
- ID : n° identifiant le client - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du client
- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

2.4.3 La table [CRENEAUX]

Elle liste les créneaux horaires où les RV sont possibles :



Field Name	Field Type	Size	Precision	Not Null	Default
ID	BIGINT	20	0	☑	Null
VERSION	INTEGER	11	0	☑	
HDEBUT	INTEGER	11	0	☑	
MDEBUT	INTEGER	11	0	☑	
HFIN	INTEGER	11	0	☑	
MFIN	INTEGER	11	0	☑	
ID_MEDECIN	BIGINT	20	0	☑	



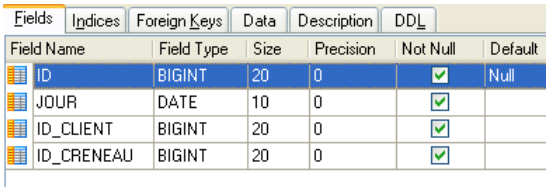
ID	VERSION	ID_MEDECIN	HDEBUT	MDEBUT	HFIN	MFIN
1	1	1	8	0	8	20
2	1	1	8	20	8	40
3	1	1	8	40	9	0
4	1	1	9	0	9	20
5	1	1	9	20	9	40
6	1	1	9	40	10	0
7	1	1	10	0	10	20
8	1	1	10	20	10	40
9	1	1	10	40	11	0
10	1	1	11	0	11	20
11	1	1	11	20	11	40
12	1	1	11	40	12	0
13	1	1	14	0	14	20
14	1	1	14	20	14	40
15	1	1	14	40	15	0
16	1	1	15	0	15	20
17	1	1	15	20	15	40
18	1	1	15	40	16	0
19	1	1	16	0	16	20
20	1	1	16	20	16	40
21	1	1	16	40	17	0
22	1	1	17	0	17	20
23	1	1	17	20	17	40
24	1	1	17	40	18	0
25	1	2	8	0	8	20
26	1	2	8	20	8	40
27	1	2	8	40	9	0
28	1	2	9	0	9	20
29	1	2	9	20	9	40
30	1	2	9	40	10	0
31	1	2	10	0	10	20
32	1	2	10	20	10	40
33	1	2	10	40	12	0
34	1	2	12	0	12	20
35	1	2	12	20	12	40
36	1	2	12	40	12	0
37	1	3	8	0	8	20
38	1	3	8	20	8	40
39	1	3	8	40	9	0
40	1	3	9	0	9	20
41	1	3	9	20	9	40
42	1	3	9	40	10	0
43	1	3	10	0	10	20
44	1	3	10	20	10	40
45	1	3	10	40	12	0
46	1	3	12	0	12	20

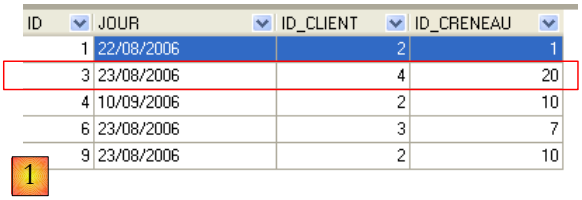
- ID : n° identifiant le créneau horaire - clé primaire de la table (ligne 8)
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- ID_MEDECIN : n° identifiant le médecin auquel appartient ce créneau – clé étrangère sur la colonne MEDECINS(ID).
- HDEBUT : heure début créneau
- MDEBUT : minutes début créneau
- HFIN : heure fin créneau
- MFIN : minutes fin créneau

La seconde ligne de la table [CRENEAUX] (cf [1] ci-dessus) indique, par exemple, que le créneau n° 2 commence à 8 h 20 et se termine à 8 h 40 et appartient au médecin n° 1 (Mme Marie PELISSIER).

2.4.4 La table [RV]

Elle liste les RV pris pour chaque médecin :





- ID : n° identifiant le RV de façon unique – clé primaire
- JOUR : jour du RV
- ID_CRENEAU : créneau horaire du RV - clé étrangère sur le champ [ID] de la table [CRENEAUX] – fixe à la fois le créneau horaire et le médecin concerné.
- ID_CLIENT : n° du client pour qui est faite la réservation – clé étrangère sur le champ [ID] de la table [CLIENTS]

Cette table a une contrainte d'unicité sur les valeurs des colonnes jointes (JOUR, ID_CRENEAU) :

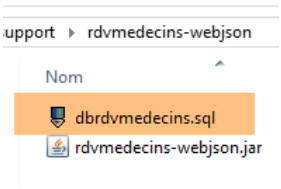
```
ALTER TABLE RV ADD CONSTRAINT UNQ1_RV UNIQUE (JOUR, ID_CRENEAU);
```

Si une ligne de la table[RV] a la valeur (JOUR1, ID_CRENEAU1) pour les colonnes (JOUR, ID_CRENEAU), cette valeur ne peut se retrouver nulle part ailleurs. Sinon, cela signifierait que deux RV ont été pris au même moment pour le même médecin. D'un point de vue programmation Java, le pilote JDBC de la base lance une *SQLException* lorsque ce cas se produit.

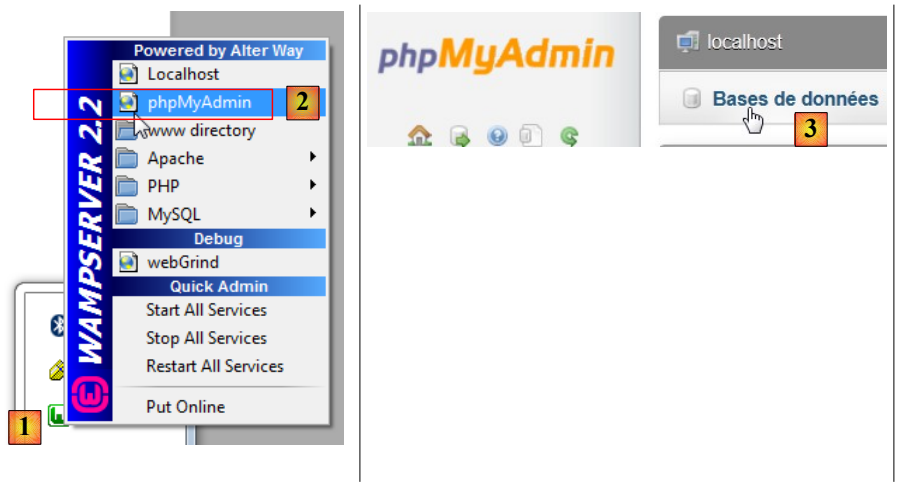
La ligne d'*id* égal à 3 (cf [1] ci-dessus) signifie qu'un RV a été pris pour le créneau n° 20 et le client n° 4 le 23/08/2006. La table [CRENEAUX] nous apprend que le créneau n° 20 correspond au créneau horaire 16 h 20 - 16 h 40 et appartient au médecin n° 1 (Mme Marie PELISSIER). La table [CLIENTS] nous apprend que le client n° 4 est Melle Brigitte BISTROU.

2.4.5 Génération de la base

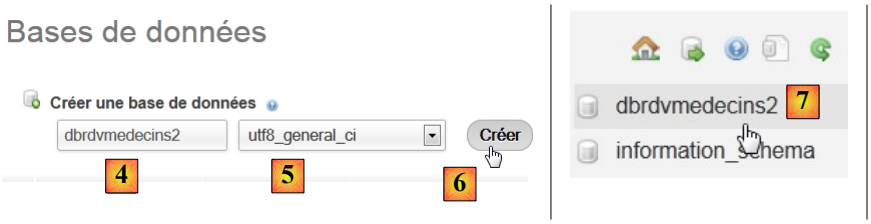
Pour créer les tables et les remplir on pourra utiliser le script [dbrdvmedecins.sql] qu'on trouvera sur le site des exemples.



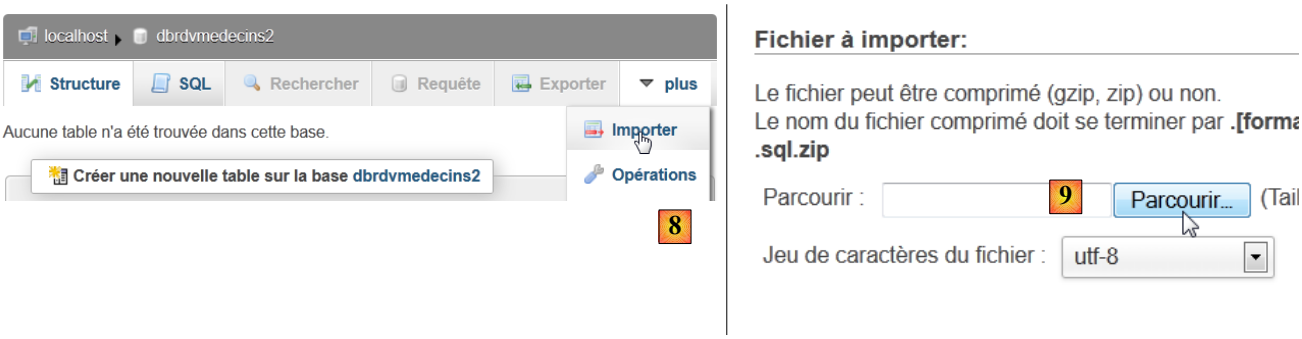
Avec [WampServer] (cf paragraphe 2.8.1, page 263), on pourra procéder comme suit :



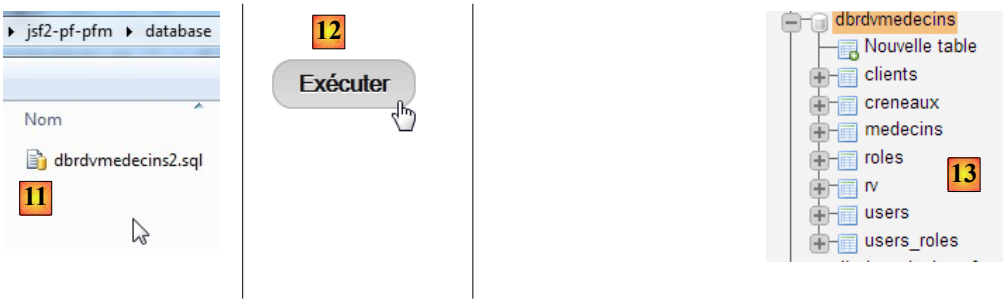
- en [1], on clique sur l'icône de [WampServer] et on choisit l'option [PhpMyAdmin] [2],
- en [3], dans la fenêtre qui s'est ouverte, on sélectionne le lien [Bases de données],



- en [6], on crée une base de données dont on a donné le nom [4] et l'encodage [5],
- en [7], la base a été créée. On clique sur son lien,



- en [8], on importe un fichier SQL,
- qu'on désigne dans le système de fichiers avec le bouton [9],



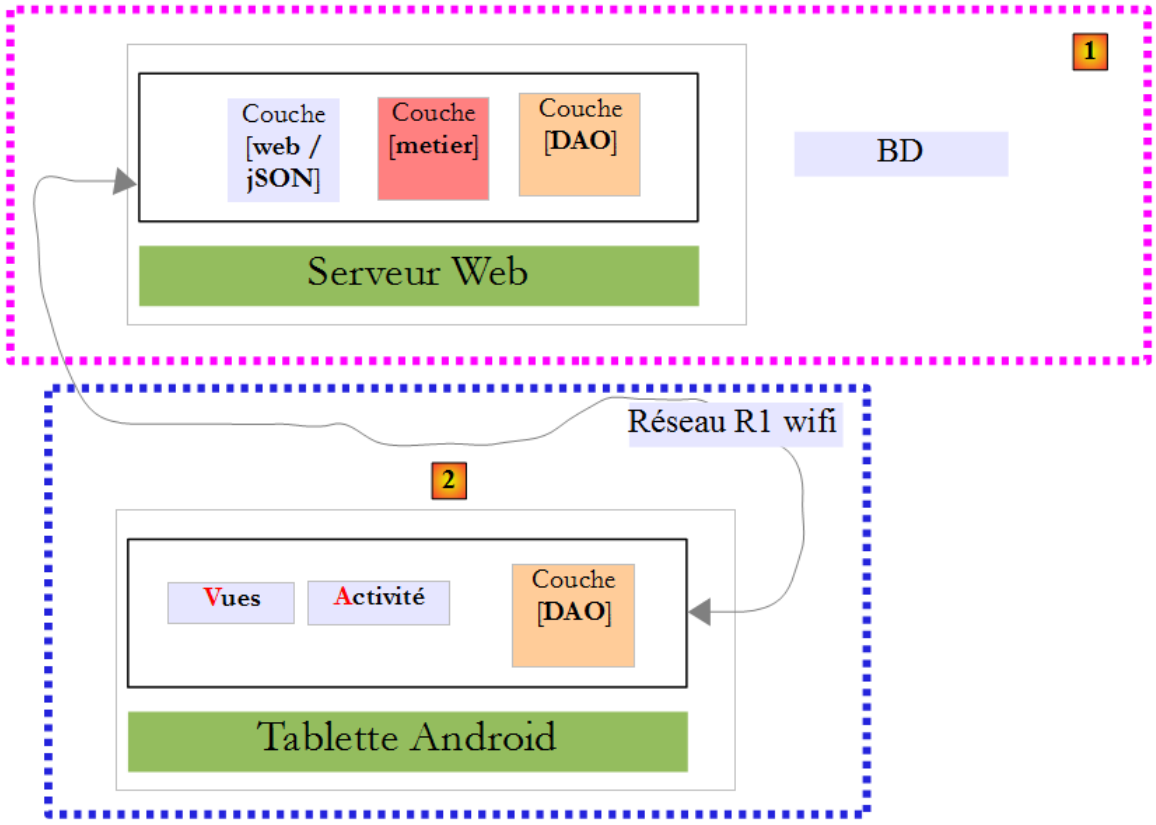
- en [11], on sélectionne le script SQL et en [12] on l'exécute,
- en [13], les tables de la base ont été créées. On suit l'un des liens,

←T→			ID	TITRE	NOM	VERSION	PRENOM
Modifier	Éditer en place	Copier Effacer	1	Mr	MARTIN	1	Jules
Modifier	Éditer en place	Copier Effacer	2	Mme	GERMAN	14 1	Christine
Modifier	Éditer en place	Copier Effacer	3	Mr	JACQUARD	1	Jules
Modifier	Éditer en place	Copier Effacer	4	Melle	BISTROU	1	Brigitte

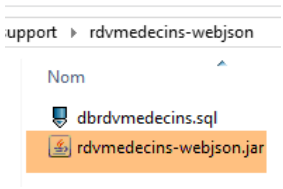
- en [14], le contenu de la table.

Par la suite, nous ne reviendrons plus sur cette base. Mais le lecteur est invité à suivre son évolution au fil des tests surtout lorsque ça ne marche pas.

2.5 Le serveur web / jSON



Nous nous intéressons ici au serveur [1]. Nous n'allons pas le développer. Celui-ci a été détaillé dans le document [Tutoriel AngularJS / Spring 4]. Le lecteur intéressé pourra s'y reporter. Le serveur sera disponible sous forme d'un binaire :



- [rdvmedecins-webjson.jar] est le binaire du serveur ;

2.5.1 Mise en oeuvre

Dans une fenêtre de commandes, on se place dans le dossier contenant le binaire du serveur :

```
.....\etudedecas\serveur>dir
 Le volume dans le lecteur D s'appelle Données
 Le numéro de série du volume est 8B35-77A0

 Répertoire de D:\data\istia-1415\android\dvp mobilité 1415\etudedecas\serveur

13/11/2014  17:15    <DIR>          .
13/11/2014  17:15    <DIR>          ..
06/07/2014  15:36             7 631 dbrdvmedecins.sql
13/11/2014  17:10        27 313 378 rdvmedecins-webjson.jar
```

puis pour lancer le serveur on tape la commande suivante (le SGBD MySQL doit être déjà lancé) :

```
dos>java -jar rdvmedecins-webjson.jar


  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.1.1.RELEASE)

2014-11-14 08:58:16.937  INFO 5892 --- [           main] rdvmedecins.boot.Boot                    :
Starting Boot on Gportpers3 with PID 5892 (started by ST in D:\data\istia-1415\android\dvp mobilité
1415\etudedecas\serveur)
...
2014-11-14 08:58:21.046  INFO 5892 --- [           main] .t.TomcatEmbeddedServletContainerFactory : Server
initialized with port: 8080
2014-11-14 08:58:21.279  INFO 5892 --- [           main] o.apache.catalina.core.StandardService   :
Starting service Tomcat
2014-11-14 08:58:21.279  INFO 5892 --- [           main] org.apache.catalina.core.StandardEngine  :
Starting Servlet Engine: Apache Tomcat/7.0.54
2014-11-14 08:58:21.602  INFO 5892 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       :
Initializing Spring embedded WebApplicationContext
...
2014-11-14 08:58:29.080  INFO 5892 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean  :
Mapping filter: 'springSecurityFilterChain' to: [/*]
...
2014-11-14 08:58:30.322  INFO 5892 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"{[/getAgendaMedecinJour/{idMedecin}/
{jour}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
rdvmedecins.web.responses.AgendaMedecinJourResponse
rdvmedecins.web.controllers.RdvMedecinsController.getAgendaMedecinJour(long,java.lang.String,javax.servlet
.http.HttpServletResponse)
2014-11-14 08:58:30.323  INFO 5892 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"{[/getMedecinById/{id}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto
public rdvmedecins.web.responses.MedecinResponse
rdvmedecins.web.controllers.RdvMedecinsController.getMedecinById(long)
2014-11-14 08:58:30.323  INFO 5892 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"{[/getCreneauById/{id}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto
public rdvmedecins.web.responses.CreneauResponse
rdvmedecins.web.controllers.RdvMedecinsController.getCreneauById(long)
2014-11-14 08:58:30.323  INFO 5892 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"{[/supprimerRv],methods=[POST],params=[],headers=[],consumes=[application/json;charset=UTF-
8],produces=[],custom=[]}" onto public rdvmedecins.web.responses.RvResponse
rdvmedecins.web.controllers.RdvMedecinsController.supprimerRv(rdvmedecins.web.requests.PostSupprimerRv,jav
ax.servlet.http.HttpServletResponse)
2014-11-14 08:58:30.323  INFO 5892 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"{[/getAllCreneaux/{idMedecin}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}"
onto public rdvmedecins.web.responses.CreneauxForMedecinResponse
rdvmedecins.web.controllers.RdvMedecinsController.getAllCreneaux(long)
2014-11-14 08:58:30.324  INFO 5892 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
"{[/getClientById/{id}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
rdvmedecins.web.responses.ClientResponse
rdvmedecins.web.controllers.RdvMedecinsController.getClientById(long)
```

```
27. 2014-11-14 08:58:30.324  INFO 5892 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getAllClients],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    rdvmedecins.web.responses.ClientsResponse
    rdvmedecins.web.controllers.RdvMedecinsController.getAllClients(javax.servlet.http.HttpServletResponse)
28. 2014-11-14 08:58:30.324  INFO 5892 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/getRvById/{id}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    rdvmedecins.web.responses.RvResponse rdvmedecins.web.controllers.RdvMedecinsController.getRvById(long)
29. 2014-11-14 08:58:30.324  INFO 5892 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped
    "{[/ajouterRv],methods=[POST],params=[],headers=[],consumes=[application/json;charset=UTF-
    8],produces=[],custom=[]}" onto public rdvmedecins.web.responses.RvResponse
    rdvmedecins.web.controllers.RdvMedecinsController.ajouterRv(rdvmedecins.web.requests.PostAjouterRv,javax.s
    ervlet.http.HttpServletResponse)
30. ...
```

Le serveur affiche de nombreux logs. Nous n'avons retenu ci-dessus que ceux utiles à comprendre :

- lignes 14-16 : un serveur Tomcat embarqué est lancé sur le port 8080 de la machine. C'est ce serveur qui exécute l'application web de gestion des rendez-vous. Cette application est en fait un service web / jSON : elle est interrgogée via des URL et elle répond en envoyant une chaîne jSON ;
- ligne 19 : le service web est sécurisé avec le frameworl [Spring Security]. On accède aux URL du service web en s'authentifiant ;
- lignes 21-29 : les URL exposées par le service web ;

Nous allons détailler ces dernières.

2.5.2 Sécurisation du service web

Les URL exposées par le service web sont sécurisées. Le serveur attend dans la requête HTTP du client l'entête suivant :

```
Authorization: Basic code
```

Le code attendu est le codage en base64 [http://fr.wikipedia.org/wiki/Base64] de la chaîne 'utilisateur:motdepasse'. Le service web n'accepte dans son état initial qu'un utilisateur 'admin' avec le mot de passe 'admin'. L'entête ci-dessus devient pour cet utilisateur la ligne suivante :

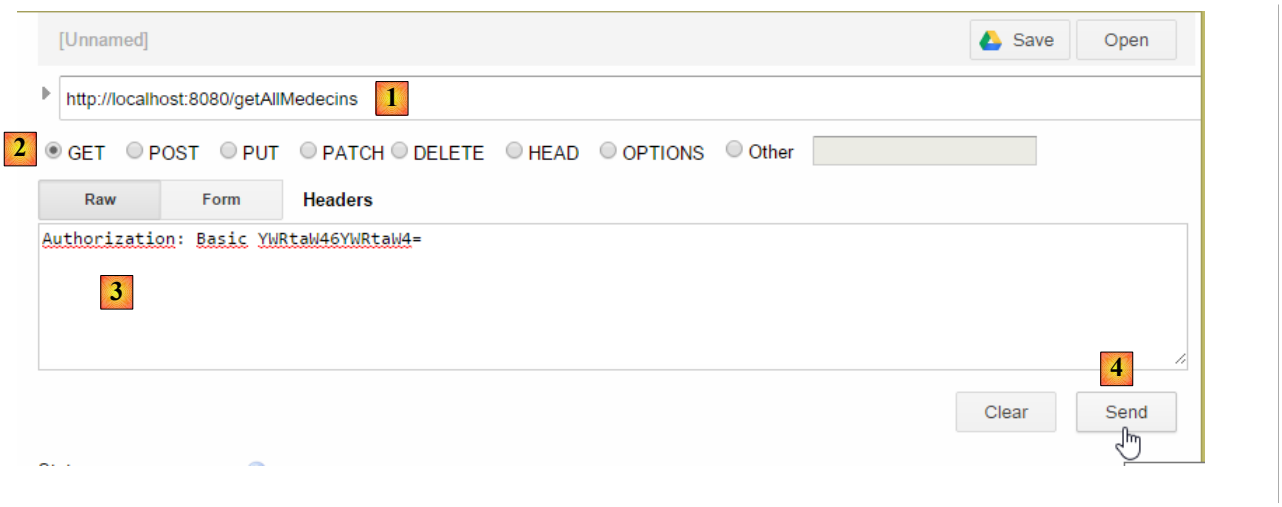
```
Authorization: Basic YWRtaW46YWRtaW4=
```

Afin de pouvoir envoyer cet entête HTTP, nous utilisons le client HTTP [Advanced Rest Client] qui est un plugin du navigateur Chrome (cf paragraphe 1.16.8, page 198). Nous allons tester à la main les différentes URL exposées par le service web afin de comprendre :

- les paramètres attendus par l'URL ;
- la nature exacte de sa réponse ;

2.5.3 Liste des médecins

L'URL [/getAllMedecins] permet d'obtenir la liste des médecins :



- en [1], l'URL interrogée ;
- en [2], la méthode HTTP utilisée pour cette interrogation ;
- en [3], l'entête HTTP de sécurité de l'utilisateur (admin, admin) ;
- en [4], on envoie la requête HTTP ;

La réponse du serveur est la suivante :

- en [5], la réponse jSON du serveur, mise en forme ;
- en [6], la même réponse à l'état brut ;

La forme [5] permet de mieux voir la structure de la réponse. Toutes les réponses du service web dérivent de la classe [AbstractResponse] suivante :

```
1.  public abstract  class AbstractResponse {
2.
3.      // ----------------- propriétés
4.      // statut de l'opération
5.      private int status;
6.      // les éventuels messages d'erreur
7.      private List<String> messages;
8.
9.      // constructeurs
10.     public AbstractResponse(){
11.
12.     }
13.
14.     public AbstractResponse(int status, List<String> messages){
15.        this.status=status;
16.        this.messages=messages;
17.     }
18.
19.     // getters et setters
20.     ....
21. }
```

- ligne 5 : le statut de la réponse. La valeur 0 veut dire qu'il n'y a pas eu d'erreur, sinon il y a eu erreur ;
- ligne 7 : une liste de messages d'erreur s'il y a eu erreur ;

La réponse à l'URL [/getAllMedecins] est la chaîne jSON d'un objet de type [MedecinsResponse] :

```
1.  package rdvmedecins.android.dao.responses;
2.
3.
4.  import rdvmedecins.android.dao.entities.Medecin;
5.
```

```
6.  import java.util.List;
7.
8.  public class MedecinsResponse extends AbstractResponse {
9.    // liste des médecins
10.   private List<Medecin> medecins;
11.
12.   // constructeurs
13.   public MedecinsResponse() {
14.
15.   }
16.
17.   public MedecinsResponse(int status, List<String> messages, List<Medecin> medecins) {
18.     super(status, messages);
19.     this.medecins = medecins;
20.   }
21.
22.   // getters et setters
23. ...
24. }
```

- ligne 8, la classe [MedecinsResponse] étend la classe [AbstractResponse] ;

Ligne 10, la classe [Medecin] est la suivante :

```
1.  package rdvmedecins.android.dao.entities;
2.
3.  public class Medecin extends Personne {
4.
5.     // constructeur par défaut
6.     public Medecin() {
7.     }
8.
9.     // constructeur avec paramètres
10.    public Medecin(String titre, String nom, String prenom) {
11.       super(titre, nom, prenom);
12.    }
13.
14.    public String toString() {
15.       return String.format("Medecin[%s]", super.toString());
16.    }
17.
18. }
```

Ligne 3, la classe [Medecin] étend la classe [Personne] suivante :

```
1.  package rdvmedecins.android.dao.entities;
2.
3.  public class Personne extends AbstractEntity {
4.     // attributs d'une personne
5.     private String titre;
6.     private String nom;
7.     private String prenom;
8.
9.     // constructeur par défaut
10.    public Personne() {
11.    }
12.
13.    // constructeur avec paramètres
14.    public Personne(String titre, String nom, String prenom) {
15.       this.titre = titre;
16.       this.nom = nom;
17.       this.prenom = prenom;
18.    }
19.
20.    // toString
21.    public String toString() {
22.       return String.format("Personne[%s, %s, %s, %s, %s]", id, version, titre, nom, prenom);
23.    }
24.
25.    // getters et setters
26.    ...
27.    }
```

Ligne 3, la classe [Personne] étend la classe [AbstractEntity] suivante :

```
1.  package rdvmedecins.android.dao.entities;
2.
3.  import java.io.Serializable;
4.
5.  public class AbstractEntity implements Serializable {
6.
7.     private static final long serialVersionUID = 1L;
8.     protected Long id;
9.     protected Long version;
10.
11.    @Override
12.    public int hashCode() {
13.       int hash = 0;
14.       hash += (id != null ? id.hashCode() : 0);
15.       return hash;
16.    }
17.
18.    // initialisation
19.    public AbstractEntity build(Long id, Long version) {
20.       this.id = id;
21.       this.version = version;
22.       return this;
23.    }
24.
25.    @Override
26.    public boolean equals(Object entity) {
27.       String class1 = this.getClass().getName();
28.       String class2 = entity.getClass().getName();
29.       if (!class2.equals(class1)) {
30.          return false;
31.       }
32.       AbstractEntity other = (AbstractEntity) entity;
33.       return this.id == other.id;
34.    }
35.
36.    // getters et setters
37.    ...
38. }
```

Au final, la structure d'un objet [Medecin] est la suivante :

```
[Long id; Long version; String titre; String nom; String prenom;]
```

et celle de [MedecinsResponse] la suivante :

```
[int status ; List<String> messages ; List<Medecin> medecins]
```

Par la suite, nous utiliserons ces définitions raccourcies pour caractériser la réponse du serveur. Par ailleurs, pendant un certain temps, nous ne montrerons plus de copies d'écran. Il suffit de répéter ce que nous venons de voir. Nous reviendrons aux copies d'écran lorsqu'il faudra faire une requête POST. Nous présenterons également un exemple d'exécution sous la forme suivante :

URL	/getAllMedecins
Réponse	{"status":0,"messages":null,"medecins": [{"id":1,"version":1,"titre":"Mme","nom":"PELISSIER","prenom":"Marie"}, {"id":2,"version":1,"titre":"Mr","nom":"BROMARD","prenom":"Jacques"}, {"id":3,"version":1,"titre":"Mr","nom":"JANDOT","prenom":"Philippe"}, {"id":4,"version":1,"titre":"Melle","nom":"JACQUEMOT","prenom":"Justine"}]}

2.5.4 Liste des clients

URL	/getAllClients
Réponse	ClientsResponse :[int status ; List<String> messages ; List<Client> clients] Client : [Long id; Long version; String titre; String nom; String prenom;]

Exemple :

URL	/getAllClients
Réponse	{"status":0,"messages":null,"clients": [{"id":1,"version":1,"titre":"Mr","nom":"MARTIN","prenom":"Jules"}, {"id":2,"version":1,"titre":"Mme","nom":"GERMAN","prenom":"Christine"}, {"id":3,"version":1,"titre":"Mr","nom":"JACQUARD","prenom":"Jules"}, {"id":4,"version":1,"titre":"Melle","nom":"BISTROU","prenom":"Brigitte"}]}

2.5.5 Liste des créneaux d'un médecin

URL	/getAllCreneaux/{idMedecin}
Réponse	CreneauxForMedecinResponse :[int status ; List<String> messages ; List<Creneau> creneaux] Creneau : [int hdebut ; int mdebut ; int hfin ; int mfin ;]

- [idMedecin] : identifiant du médecin dont on veut les créneaux horaires de consultation ;
- [hdebut] : heure de début de la consultation ;
- [mdebut] : minutes de début de la consultation ;
- [hfin] : heure de fin de la consultation ;
- [mfin] : minutes de fin de la consultation ;

Pour un créneau entre 10h20 et 10h40 on aura [hdebut, mdebut, hfin, mfin]=[10, 20, 10, 40].

Exemple :

URL	/getAllCreneaux/1
Réponse	{"status":0,"messages":null,"creneaux": [{"id":1,"version":1,"hdebut":8,"mdebut":0,"hfin":8,"mfin":20,"idMedecin":1}, {"id":2,"version":1,"hdebut":8,"mdebut":20,"hfin":8,"mfin":40,"idMedecin":1}, {"id":3,"version":1,"hdebut":8,"mdebut":40,"hfin":9,"mfin":0,"idMedecin":1}, {"id":4,"version":1,"hdebut":9,"mdebut":0,"hfin":9,"mfin":20,"idMedecin":1}, {"id":5,"version":1,"hdebut":9,"mdebut":20,"hfin":9,"mfin":40,"idMedecin":1}, {"id":6,"version":1,"hdebut":9,"mdebut":40,"hfin":10,"mfin":0,"idMedecin":1}, {"id":7,"version":1,"hdebut":10,"mdebut":0,"hfin":10,"mfin":20,"idMedecin":1}, {"id":8,"version":1,"hdebut":10,"mdebut":20,"hfin":10,"mfin":40,"idMedecin":1}, {"id":9,"version":1,"hdebut":10,"mdebut":40,"hfin":11,"mfin":0,"idMedecin":1}, {"id":10,"version":1,"hdebut":11,"mdebut":0,"hfin":11,"mfin":20,"idMedecin":1}, {"id":11,"version":1,"hdebut":11,"mdebut":20,"hfin":11,"mfin":40,"idMedecin":1}, {"id":12,"version":1,"hdebut":11,"mdebut":40,"hfin":12,"mfin":0,"idMedecin":1}, {"id":13,"version":1,"hdebut":14,"mdebut":0,"hfin":14,"mfin":20,"idMedecin":1}, {"id":14,"version":1,"hdebut":14,"mdebut":20,"hfin":14,"mfin":40,"idMedecin":1}, {"id":15,"version":1,"hdebut":14,"mdebut":40,"hfin":15,"mfin":0,"idMedecin":1}, {"id":16,"version":1,"hdebut":15,"mdebut":0,"hfin":15,"mfin":20,"idMedecin":1}, {"id":17,"version":1,"hdebut":15,"mdebut":20,"hfin":15,"mfin":40,"idMedecin":1}, {"id":18,"version":1,"hdebut":15,"mdebut":40,"hfin":16,"mfin":0,"idMedecin":1}, {"id":19,"version":1,"hdebut":16,"mdebut":0,"hfin":16,"mfin":20,"idMedecin":1}, {"id":20,"version":1,"hdebut":16,"mdebut":20,"hfin":16,"mfin":40,"idMedecin":1}, {"id":21,"version":1,"hdebut":16,"mdebut":40,"hfin":17,"mfin":0,"idMedecin":1}, {"id":22,"version":1,"hdebut":17,"mdebut":0,"hfin":17,"mfin":20,"idMedecin":1}, {"id":23,"version":1,"hdebut":17,"mdebut":20,"hfin":17,"mfin":40,"idMedecin":1}, {"id":24,"version":1,"hdebut":17,"mdebut":40,"hfin":18,"mfin":0,"idMedecin":1}]}

2.5.6 Liste des rendez-vous d'un médecin

URL	/getRvMedecinJour/{idMedecin}/{jour}
Réponse	RvMedecinJourResponse :[int status ; List<String> messages ; List<Rv> rvs] Rv : [Date jour ; Client client ; Creneau creneau ; long idClient ; long idCreneau]

- [idMedecin] : identifiant du médecin dont on veut les rendez-vous ;

- URL [jour] : jour des rendez-vous sous la forme 'aaaa-mm-jj' ;
- Réponse [jour] : idem mais sous la forme d'une date Java ;
- [client] : le client du rendez-vous. Sa structure a été décrite précédemment ;
- [idClient] : l'identifiant du client ;
- [creneau] : le créneau du rendez-vous. Sa structure a été décrite précédemment ;
- [idCreneau] : l'identifiant du créneau ;

Exemple :

URL	/getRvMedecinJour/1/2014-07-08
Réponse	{"status":0,"messages":null,"rvs":[{"id":45,"version":0,"jour":"2014-07-08","client":{"id":1,"version":1,"titre":"Mr","nom":"MARTIN","prenom":"Jules"},"creneau":{"id":1,"version":1,"hdebut":8,"mdebut":0,"hfin":8,"mfin":20,"idMedecin":1},"idClient":1,"idCreneau":1}]}

2.5.7 L'agenda d'un médecin

URL	/getAgendaMedecinJour/{idMedecin}/{jour}
Réponse	AgendaMedecinJourResponse :[int status ; List<String> messages ; AgendaMedecinJour agenda] AgendaMedecinJour : [Medecin medecin ;Date jour ; CreneauMedecinJour[] creneauxMedecinJour] CreneauMedecinJour : [Creneau creneau ; Rv rv]

- [idMedecin] : identifiant du médecin dont on veut les rendez-vous ;
- URL [jour] : jour des rendez-vous sous la forme 'aaaa-mm-jj' ;
- [agenda] : agenda du médecin ;
- [medecin] : le médecin concerné. Sa structure a été définie précédemment ;
- Réponse [jour] : le jour de l'agenda sous la forme d'une date Java ;
- [creneauxMedecinJour] : un tableau d'éléments de type [CreneauMedecinJour] ;
- [creneau] : un créneau. Sa structure a été décrite précédemment ;
- [rv] : un rendez-vous. Sa structure a été décrite précédemment ;

Exemple :

URL	/getAgendaMedecinJour/1/2014-07-08
Réponse	{"status":0,"messages":null,"agenda":{"medecin":{"id":1,"version":1,"titre":"Mme","nom":"PELISSIER","prenom":"Marie"},"jour":1404770400000,"creneauxMedecinJour":[{"creneau":{"id":1,"version":1,"hdebut":8,"mdebut":0,"hfin":8,"mfin":20,"idMedecin":1},"rv":{"id":45,"version":0,"jour":"2014-07-08","client":{"id":1,"version":1,"titre":"Mr","nom":"MARTIN","prenom":"Jules"},"creneau":{"id":1,"version":1,"hdebut":8,"mdebut":0,"hfin":8,"mfin":20,"idMedecin":1},"idClient":1,"idCreneau":1}},{"creneau":{"id":2,"version":1,"hdebut":8,"mdebut":20,"hfin":8,"mfin":40,"idMedecin":1},"rv":null},{"creneau":{"id":3,"version":1,"hdebut":8,"mdebut":40,"hfin":9,"mfin":0,"idMedecin":1},"rv":null},{"creneau":{"id":4,"version":1,"hdebut":9,"mdebut":0,"hfin":9,"mfin":20,"idMedecin":1},"rv":null},{"creneau":{"id":5,"version":1,"hdebut":9,"mdebut":20,"hfin":9,"mfin":40,"idMedecin":1},"rv":null},{"creneau":{"id":6,"version":1,"hdebut":9,"mdebut":40,"hfin":10,"mfin":0,"idMedecin":1},"rv":null},{"creneau":{"id":7,"version":1,"hdebut":10,"mdebut":0,"hfin":10,"mfin":20,"idMedecin":1},"rv":null},{"creneau":{"id":8,"version":1,"hdebut":10,"mdebut":20,"hfin":10,"mfin":40,"idMedecin":1},"rv":null},{"creneau":{"id":9,"version":1,"hdebut":10,"mdebut":40,"hfin":11,"mfin":0,"idMedecin":1},"rv":null},{"creneau":

```
{"id":10,"version":1,"hdebut":11,"mdebut":0,"hfin":11,"mfin":20,"idMedecin":1},"rv":nul
l},{"creneau":
{"id":11,"version":1,"hdebut":11,"mdebut":20,"hfin":11,"mfin":40,"idMedecin":1},"rv":nu
ll},{"creneau":
{"id":12,"version":1,"hdebut":11,"mdebut":40,"hfin":12,"mfin":0,"idMedecin":1},"rv":nul
l},{"creneau":
{"id":13,"version":1,"hdebut":14,"mdebut":0,"hfin":14,"mfin":20,"idMedecin":1},"rv":nul
l},{"creneau":
{"id":14,"version":1,"hdebut":14,"mdebut":20,"hfin":14,"mfin":40,"idMedecin":1},"rv":nu
ll},{"creneau":
{"id":15,"version":1,"hdebut":14,"mdebut":40,"hfin":15,"mfin":0,"idMedecin":1},"rv":nul
l},{"creneau":
{"id":16,"version":1,"hdebut":15,"mdebut":0,"hfin":15,"mfin":20,"idMedecin":1},"rv":nul
l},{"creneau":
{"id":17,"version":1,"hdebut":15,"mdebut":20,"hfin":15,"mfin":40,"idMedecin":1},"rv":nu
ll},{"creneau":
{"id":18,"version":1,"hdebut":15,"mdebut":40,"hfin":16,"mfin":0,"idMedecin":1},"rv":nul
l},{"creneau":
{"id":19,"version":1,"hdebut":16,"mdebut":0,"hfin":16,"mfin":20,"idMedecin":1},"rv":nul
l},{"creneau":
{"id":20,"version":1,"hdebut":16,"mdebut":20,"hfin":16,"mfin":40,"idMedecin":1},"rv":nu
ll},{"creneau":
{"id":21,"version":1,"hdebut":16,"mdebut":40,"hfin":17,"mfin":0,"idMedecin":1},"rv":nul
l},{"creneau":
{"id":22,"version":1,"hdebut":17,"mdebut":0,"hfin":17,"mfin":20,"idMedecin":1},"rv":nul
l},{"creneau":
{"id":23,"version":1,"hdebut":17,"mdebut":20,"hfin":17,"mfin":40,"idMedecin":1},"rv":nu
ll},{"creneau":
{"id":24,"version":1,"hdebut":17,"mdebut":40,"hfin":18,"mfin":0,"idMedecin":1},"rv":nul
l}]}}
```


On a mis en exergue le cas où il y a un rendez-vous dans le créneau et le cas où il n'y en a pas.

2.5.8 Obtenir un médecin par son identifiant

URL	/getMedecinById/{idMedecin}
Réponse	MedecinResponse :[int status ; List<String> messages ; Medecin medecin]

- [idMedecin] : l'identifiant du médecin ;

Exemple 1 :

URL	/getMedecinById/1
Réponse	{"status":0,"messages":null,"medecin": {"id":1,"version":1,"titre":"Mme","nom":"PELISSIER","prenom":"Marie"}}

Exemple 2 :

URL	/getMedecinById/100
Réponse	{"status":2,"messages":["Médecin [100] inexistant"],"medecin":null}

2.5.9 Obtenir un client par son identifiant

URL	/getClientById/{idClient}
Réponse	ClientResponse :[int status ; List<String> messages ; Client client]

- [idClient] : l'identifiant du client ;

Exemple 1 :

URL	/getClientById/1
Réponse	{"status":0,"messages":null,"client": {"id":1,"version":1,"titre":"Mr","nom":"MARTIN","prenom":"Jules"}}

Exemple 2 :

URL	/getClientById/100
Réponse	{"status":2,"messages":["Client [100] inexistant"],"client":null}

2.5.10 Obtenir un créneau par son identifiant

URL	/getCreneauById/{idCreneau}
Réponse	CreneauResponse :[int status ; List<String> messages ; Creneau creneau]

- [idCreneau] : l'identifiant du créneau ;

Exemple 1 :

URL	/getCreneauById/10
Réponse	{"status":0,"messages":null,"creneau": {"id":10,"version":1,"hdebut":11,"mdebut":0,"hfin":11,"mfin":20,"idMedecin":1}}

On remarquera que dans la réponse, il n'y a pas le médecin propriétaire du créneau mais seulement son identifiant.

Exemple 2 :

URL	/getCreneauById/100
Réponse	{"status":2,"messages":["Créneau [100] inexistant"],"creneau":null}

2.5.11 Obtenir un rendez-vous par son identifiant

URL	/getRvById/{idRv}
Réponse	RvResponse :[int status ; List<String> messages ; Rv rv]

- [idRv] : l'identifiant du rendez-vous ;

Exemple 1 :

URL	/getRvById/45
Réponse	{"status":0,"messages":null,"rv":{"id":45,"version":0,"jour":"2014-07- 08","idClient":1,"idCreneau":1}}

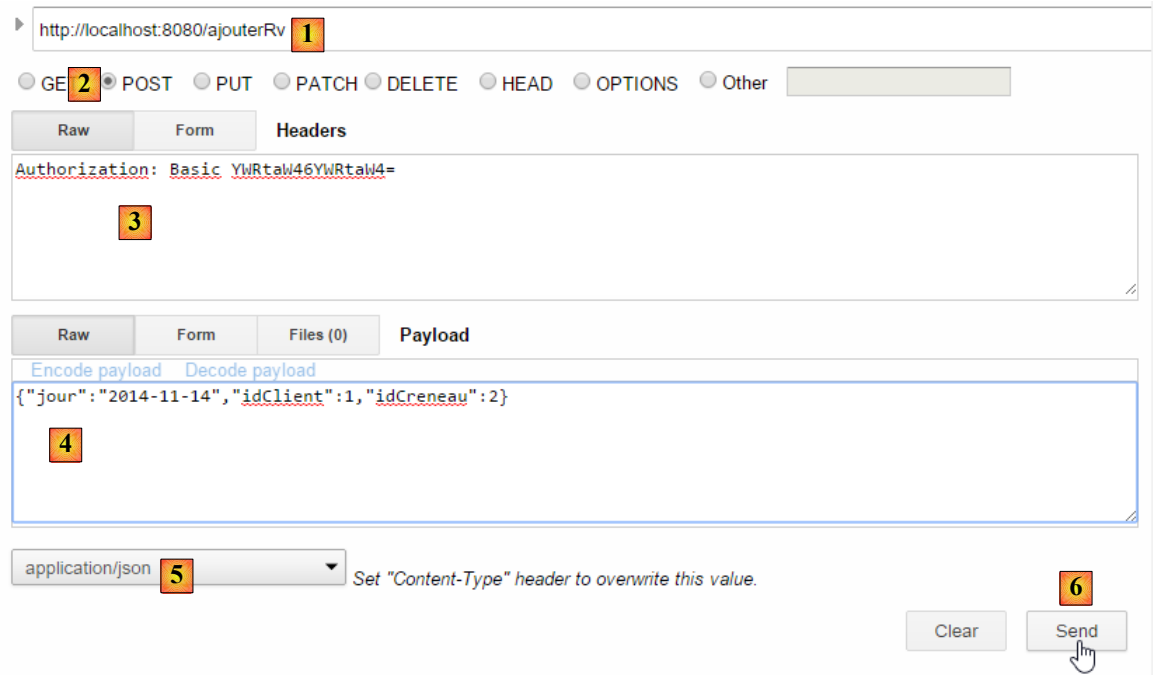
On remarquera que dans la réponse, il n'y a ni le client, ni le créneau du rendez-vous mais seulements leurs identifiants.

Exemple 2 :

URL	/getCreneauById/455
Réponse	{"status":2,"messages":["Rv [455] inexistant"],"rv":null}

2.5.12 Ajouter un rendez-vous

L'URL [/ajouterRv] permet d'ajouter un rendez-vous. Les informations nécessaires à cet ajout (le jour, le créneau et le client) sont transmises via une requête HTTP POST. Nous montrons comment réaliser cette requête avec l'outil [Advanced Rest Client].



- en [1], l'URL interrogée ;
- en [2], elle est interrogée par un POST ;
- en [3], l'entête HTTP de l'authentification ;
- en [4], les informations transmises par le POST. C'est une chaîne jSON contenant :
  - [jour] : le jour du rendez-vous sous la forme 'aaaa-mm-jj',
  - [idClient] : l'identifiant du client pour lequel le rendez-vous est pris,
  - [idCreneau] : l'identifiant du créneau horaire du rendez-vous. Comme un créneau horaire appartient à un médecin précis, on désigne par là également le médecin ;
- en [5], on précise au serveur que les valeurs qui lui sont postées le sont sous la forme d'une chaîne jSON ;
- en [6], on envoie la requête ;

La chaîne jSON qui est postée est celle de l'objet de type [PostAjouterRv] suivant :

```
1.  public class PostAjouterRv {
2.
3.    // données du post
4.    private String jour;
5.    private long idClient;
6.    private long idCreneau;
7.
8.    // constructeurs
9.    public PostAjouterRv() {
10.
11.   }
12.
13.   public PostAjouterRv(String jour, long idCreneau, long idClient) {
14.     this.jour = jour;
15.     this.idClient = idClient;
16.     this.idCreneau = idCreneau;
17.   }
18.
19.   // getters et setters
20.   ...
21. }
```

La réponse du serveur est de type [RvResponse] `[int status ; List<String> messages ; Rv rv]` où [rv] est le rendez-vous ajouté.

La réponse du serveur à la requête plus haut est la suivante :

On notera ci-dessus que certaines informations ne sont pas renseignées [idClient, idCreneau] mais on les trouve dans les champs [client] et [creneau]. L'information importante est l'identifiant du rendez-vous ajouté (65). Le service web aurait pu se contenter de renvoyer cette seule information.

2.5.13 Supprimer un rendez-vous

Cette opération se fait également par un POST :

URL	/supprimerRv
POST	{'idRv':idRv}
Réponse	RvResponse :[int status ; List<String> messages ; Rv rv]

La valeur postée est la chaîne jSON d'un objet de type [PostSupprimerRv] suivant :

```
1.  public class PostSupprimerRv {
2.
3.    // données du post
4.    private long idRv;
5.
6.    // constructeurs
7.    public PostSupprimerRv() {
8.
9.    }
10.
11.   public PostSupprimerRv(long idRv) {
12.     this.idRv = idRv;
13.   }
14.
15.   // getters et setters
16.   ...
17. }
```

- ligne 4, [idRv] est l'identifiant du rendez-vous à supprimer.

Exemple 1 :

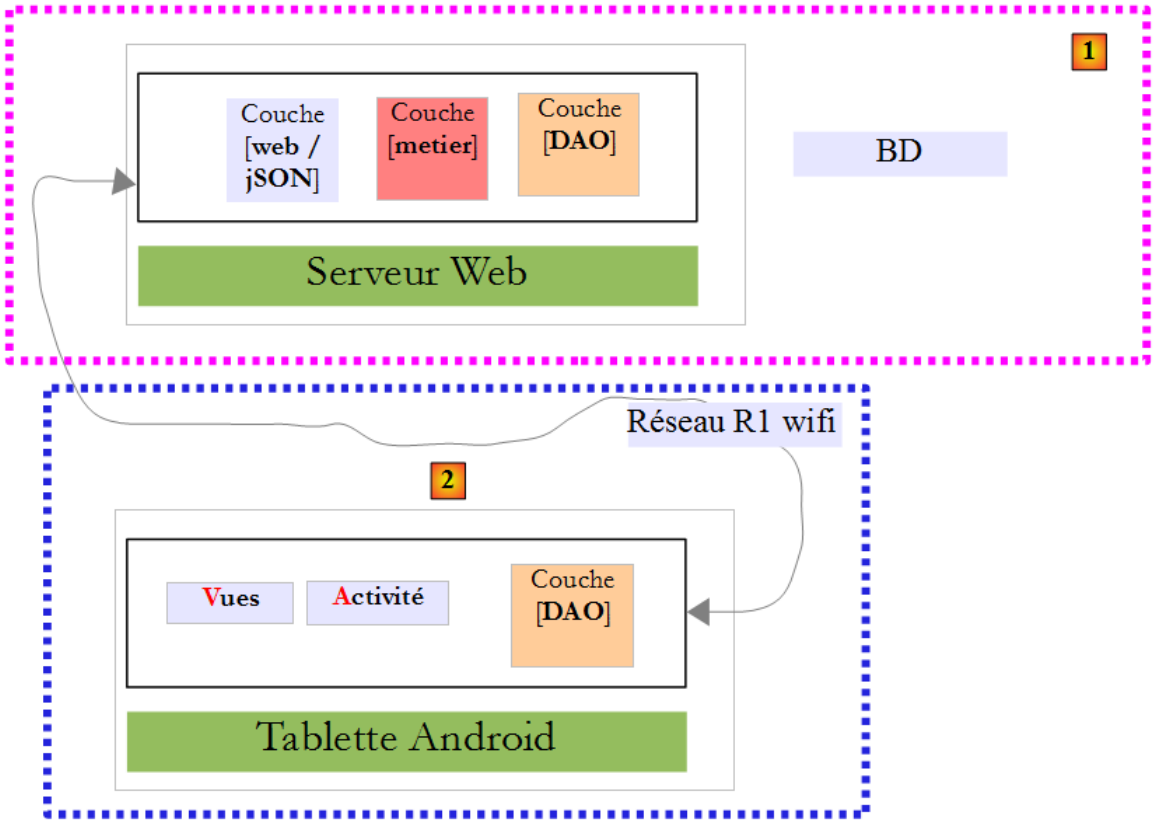
URL	/supprimerRv
POST	{"idRv":65}
Réponse	{"status":0,"messages":null,"rv":null}

Le rendez-vous de n° 65 a bien été supprimé car [status=0].

Exemple 2 :

URL	/supprimerRv
POST	{"idRv":650}
Réponse	{"status":2,"messages":["Rv [650] inexistant"],"rv":null}

2.6 Le client Android



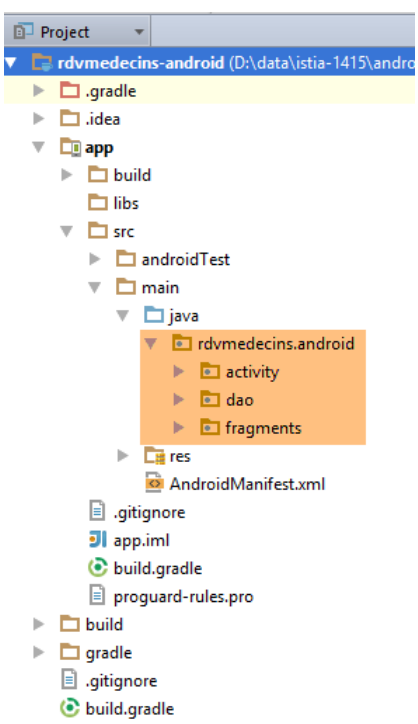
Maintenant que le serveur [1] a été détaillé et est opérationnel, nous allons étudier le client Android [2].

2.6.1 Architecture du projet IntellijIDEA

Dans l'architecture ci-dessus du client Android, on distingue trois blocs :

- la couche [DAO] chargée de la communication avec le service web ;
- les [vues] chargées de la communication avec l'utilisateur ;
- l'[activité] qui fait le lien entre les deux blocs précédents. Les vues n'ont pas connaissance de la couche [DAO]. Elles ne communiquent qu'avec l'activité.

Cette architecture est reflétée dans celle du projet IntellijIDEA du client Android :



- le projet est un projet Gradle ;
- le package [activity] implémente le bloc [activité] ;
- le package [dao] implémente la couche [DAO] ;
- le package [fragments] implémente les [vues] ;

2.6.2 Configuration Gradle

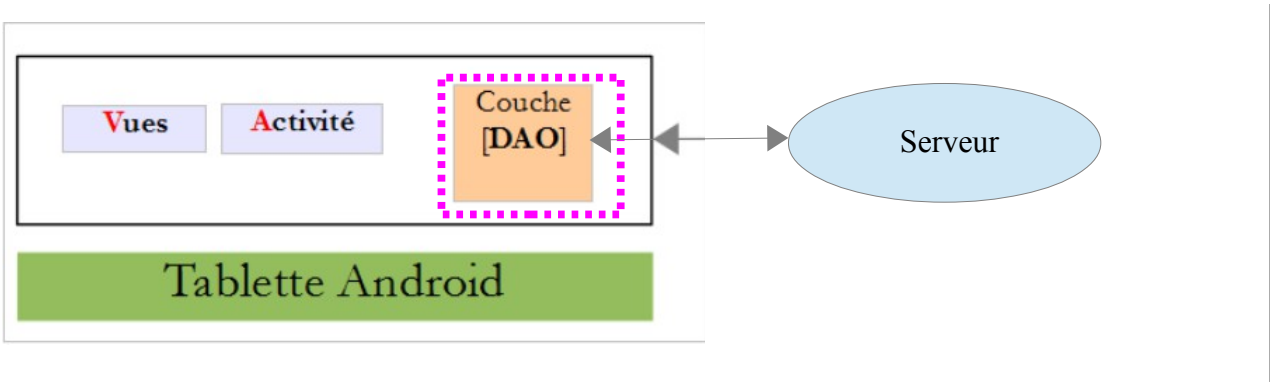
Le fichier [app / build.gradle] qui configure les dépendances du projet est le suivant :

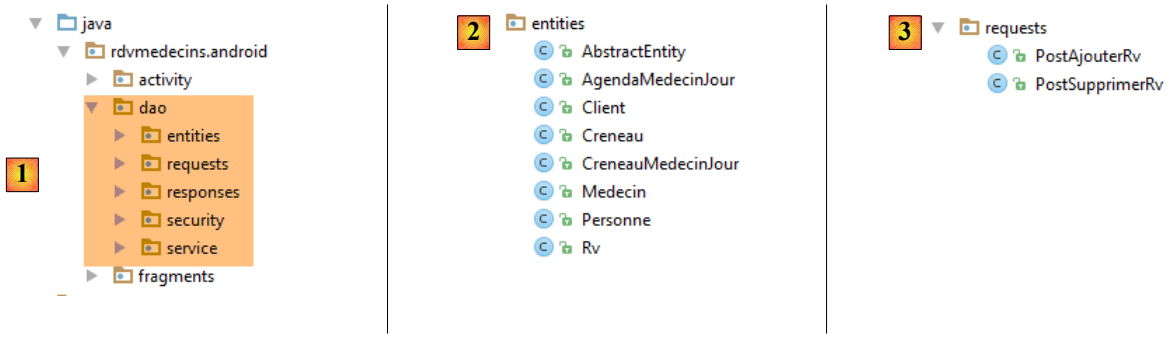
```
1.  buildscript {
2.    repositories {
3.      mavenCentral()
4.      mavenLocal()
5.    }
6.    dependencies {
7.      // replace with the current version of the Android plugin
8.      classpath 'com.android.tools.build:gradle:1.0.0'
9.      // Since Android's Gradle plugin 0.11, you have to use android-apt >= 1.3
10.     classpath 'com.neenbedankt.gradle.plugins:android-apt:1.4'
11.   }
12. }
13. apply plugin: 'com.android.application'
14. apply plugin: 'android-apt'
15. def AAVersion = '3.1'
16. dependencies {
17.   apt "org.androidannotations:androidannotations:$AAVersion"
18.   compile "org.androidannotations:androidannotations-api:$AAVersion"
19.   compile 'com.android.support:appcompat-v7:20.+'
20.   compile fileTree(dir: 'libs', include: ['*.jar'])
21.   compile 'org.springframework.android:spring-android-rest-template:1.0.1.RELEASE'
22.   compile 'org.codehaus.jackson:jackson-mapper-asl:1.9.9'
23. }
24. repositories {
25.   jcenter()
26. }
27. apt {
28.   arguments {
```

```
29.     androidManifestFile variant.outputs[0].processResources.manifestFile
30.     resourcePackageName android.defaultConfig.applicationId
31.   }
32. }
33. android {
34.   compileSdkVersion 20
35.   buildToolsVersion "20.0.0"
36.   packagingOptions {
37.     exclude 'META-INF/ASL2.0'
38.     exclude 'META-INF/NOTICE'
39.     exclude 'META-INF/LICENSE'
40.     exclude 'META-INF/notice.txt'
41.     exclude 'META-INF/license.txt'
42.   }
43.   defaultConfig {
44.     applicationId "rdvmedecins.android"
45.     minSdkVersion 11
46.     targetSdkVersion 20
47.     versionCode 1
48.     versionName "1.0"
49.   }
50.   compileOptions {
51.     sourceCompatibility JavaVersion.VERSION_1_6
52.     targetCompatibility JavaVersion.VERSION_1_6
53.   }
54.   buildTypes {
55.     release {
56.       minifyEnabled false
57.       proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
58.     }
59.   }
60. }
```

C'est celui utilisé dans un cas similaire, celui de l'exemple [exemple-10-client] (cf paragraphe 1.11.2, page 121) qui décrit le client Android d'un service web / jSON. Le lecteur est invité à revoir cet exemple et ses explications pour comprendre la suite.

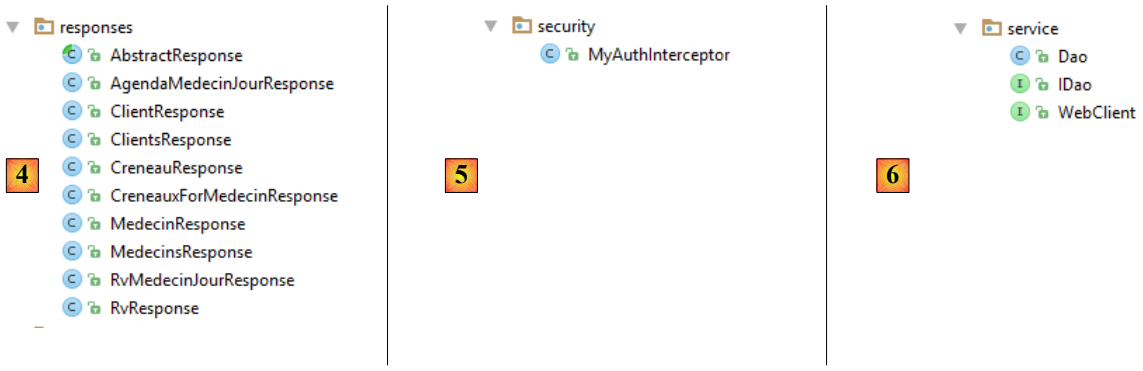
2.6.3 La couche [DAO]





- en [1], les différents packages de la couche [DAO] ;

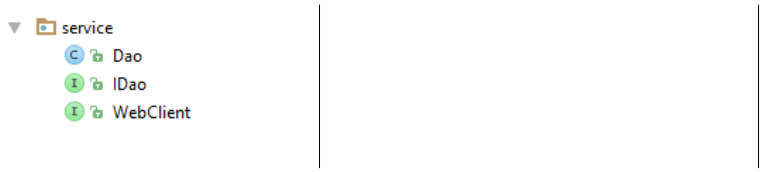
- en [2], les entités encapsulées dans les réponses du serveur. Elles ont été présentées au paragraphe 2.5, page 210 ;
- en [3], les valeurs postées pour l'ajout et la suppression d'un rendez-vous. Elles ont été présentées au paragraphe 2.5, page 210 ;



- en [4], les différentes réponses du serveur. Elles ont été présentées au paragraphe 2.5, page 210 ;
- en [5], implémentation de la sécurité côté client ;
- en [6], implémentation des échanges client / serveur ;

Nous n'allons pas revenir sur les éléments [2-4]. Ils ont déjà été présentés. Le lecteur est invité à revenir au paragraphe 2.5, page 210 si besoin est. Nous allons étudier l'implémentation du package [service]. Cela nous amènera à parler également de l'implémentation des échanges sécurisés entre le client et le serveur.

2.6.4 Implémentation des échanges client / serveur



La classe [WebClient] est un composant AA qui décrit :

- les URL exposées par le service web ;
- leurs paramètres ;
- leurs réponses ;

```
1.  package rdvmedecins.android.dao.service;
2.
3.  import org.androidannotations.annotations.rest.Get;
4.  import org.androidannotations.annotations.rest.Post;
5.  import org.androidannotations.annotations.rest.Rest;
6.  import org.androidannotations.api.rest.RestClientRootUrl;
7.  import org.androidannotations.api.rest.RestClientSupport;
8.  import org.springframework.http.converter.json.MappingJacksonHttpMessageConverter;
9.  import org.springframework.web.client.RestTemplate;
10. import rdvmedecins.android.dao.requests.PostAjouterRv;
11. import rdvmedecins.android.dao.requests.PostSupprimerRv;
12. import rdvmedecins.android.dao.responses.*;
13.
14. @Rest(converters = {MappingJacksonHttpMessageConverter.class})
15. public interface WebClient extends RestClientRootUrl, RestClientSupport {
16.
17.   // RestTemplate
18.   public void setRestTemplate(RestTemplate restTemplate);
19.
20.   // liste des médecins
21.   @Get("/getAllMedecins")
22.   public MedecinsResponse getAllMedecins();
23.
24.   // liste des clients
25.   @Get("/getAllClients")
26.   public ClientsResponse getAllClients();
27.
```

```
28.   // liste des créneaux d'un médecin
29.   @Get("/getAllCreneaux/{idMedecin}")
30.   public CreneauxForMedecinResponse getAllCreneaux(long idMedecin);
31.
32.   // liste des rendez-vous d'un médecin
33.   @Get("/getRvMedecinJour/{idMedecin}/{jour}")
34.   public RvMedecinJourResponse getRvMedecinJour(long idMedecin, String jour);
35.
36.   // Client
37.   @Get("/getClientById/{id}")
38.   public ClientResponse getClientById(long id);
39.
40.   // Médecin
41.   @Get("/getMedecinById/{id}")
42.   public MedecinResponse getMedecinById(long id);
43.
44.   // Rv
45.   @Get("/getRvById/{id}")
46.   public RvResponse getRvById(long id);
47.
48.   // Créneau
49.   @Get("/getCreneauById/{id}")
50.   public CreneauResponse getCreneauById(long id);
51.
52.   // ajouter un RV
53.   @Post("/ajouterRv")
54.   public RvResponse ajouterRv(PostAjouterRv post);
55.
56.   // supprimer un Rv
57.   @Post("/supprimerRv")
58.   public RvResponse supprimerRv(PostSupprimerRv post);
59.
60.   // obtenir l'agenda d'un médecin
61.   @Get(value = "/getAgendaMedecinJour/{idMedecin}/{jour}")
62.   public AgendaMedecinJourResponse getAgendaMedecinJour(long idMedecin, String jour);
63.
64. }
```

- lignes 20-62 : on retrouve toutes les URL étudiées au paragraphe 2.5, page 210 ;
- ligne 18 : le composant [RestTemplate] de [Spring Android] sur lequel repose la communication client / serveur ;

L'interface [IDao] de la couche [DAO] est la suivante :

```
1.  package rdvmedecins.android.dao.service;
2.
3.  import rdvmedecins.android.dao.responses.*;
4.
5.  public interface IDao {
6.    // Url du service web
7.    public void setUrlServiceWebJson(String url);
8.
9.    // utilisateur
10.   public void setUser(String user, String mdp);
11.
12.   // timeout du client
13.   public void setTimeout(int timeout);
14.
15.   // liste des clients
16.   public ClientsResponse getAllClients();
17.
18.   // liste des Médecins
19.   public MedecinsResponse getAllMedecins();
20.
21.   // liste des créneaux horaires d'un médecin
22.   public CreneauxForMedecinResponse getAllCreneaux(long idMedecin);
23.
24.   // liste des Rv d'un médecin, un jour donné
25.   public RvMedecinJourResponse getRvMedecinJour(long idMedecin, String jour);
26.
27.   // trouver un client identifié par son id
28.   public ClientResponse getClientById(long id);
29.
30.   // trouver un client identifié par son id
```

```
31.   public MedecinResponse getMedecinById(long id);
32.
33.   // trouver un Rv identifié par son id
34.   public RvResponse getRvById(long id);
35.
36.   // trouver un créneau horaire identifié par son id
37.   public CreneauResponse getCreneauById(long id);
38.
39.   // ajouter un RV
40.   public RvResponse ajouterRv(String jour, long idCreneau, long idClient);
41.
42.   // supprimer un RV
43.   public RvResponse supprimerRv(long idRv);
44.
45.   // metier
46.   public AgendaMedecinJourResponse getAgendaMedecinJour(long idMedecin, String jour);
47.
48. }
```

- ligne 7 : pour fixer l'URL du service web / jSON ;
- ligne 10 : pour fixer l'utilisateur de la communication client / serveur. [user] est l'identifiant de l'utilisateur, [mdp] son mot de passe ;
- ligne 13 : pour fixer un délai d'attente maximal de la réponse du serveur ;
- lignes 15-46 : à chaque URL exposée par le service web, correspond une méthode. Elles reprennent la signature des méthodes de mêmes noms du composant AA [WebClient] ;

L'implémentation [DAO] de l'interface [IDao] précédente est la suivante :

```
1.  package rdvmedecins.android.dao.service;
2.
3.  import org.androidannotations.annotations.AfterInject;
4.  import org.androidannotations.annotations.Bean;
5.  import org.androidannotations.annotations.EBean;
6.  import org.androidannotations.annotations.rest.RestService;
7.  import org.springframework.http.client.ClientHttpRequestFactory;
8.  import org.springframework.http.client.ClientHttpRequestInterceptor;
9.  import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;
10. import org.springframework.http.converter.json.MappingJacksonHttpMessageConverter;
11. import org.springframework.web.client.RestTemplate;
12. import rdvmedecins.android.dao.requests.PostAjouterRv;
13. import rdvmedecins.android.dao.requests.PostSupprimerRv;
14. import rdvmedecins.android.dao.responses.*;
15. import rdvmedecins.android.dao.security.MyAuthInterceptor;
16.
17. import java.util.ArrayList;
18. import java.util.List;
19.
20. @EBean(scope = EBean.Scope.Singleton)
21. public class Dao implements IDao {
22.
23.   // client du service web
24.   @RestService
25.   WebClient webClient;
26.   // sécurité
27.   @Bean
28.   MyAuthInterceptor authInterceptor;
29.
30.   // données locales
31.   private RestTemplate restTemplate;
32.   private HttpComponentsClientHttpRequestFactory factory;
33.
34.   // initialisation RestTemplate
35.   @AfterInject
36.   public void initDao() {
37.     // création du composant RestTemplate
38.     factory = new HttpComponentsClientHttpRequestFactory();
39.     restTemplate = new RestTemplate(factory);
40.     restTemplate.getMessageConverters().add(new MappingJacksonHttpMessageConverter());
41.     // intercepteur d'authentification
42.     List<ClientHttpRequestInterceptor> interceptors = new ArrayList<ClientHttpRequestInterceptor>();
43.     interceptors.add(authInterceptor);
44.     restTemplate.setInterceptors(interceptors);
45.     // on affecte le template au client web
```

```
46.     webClient.setRestTemplate(restTemplate);
47.   }
48.
49.   @Override
50.   public void setUrlServiceWebJson(String url) {
51.     // on fixe l'URL du service web
52.     webClient.setRootUrl(url);
53.   }
54.
55.   @Override
56.   public void setUser(String user, String mdp) {
57.     // on enregistre l'utilisateur dans l'intercepteur
58.     authInterceptor.setUser(user, mdp);
59.   }
60.
61.   @Override
62.   public void setTimeout(int timeout) {
63.     // on fixe le timeout des requêtes du client web
64.     factory.setReadTimeout(timeout);
65.     factory.setConnectTimeout(timeout);
66.   }
67.
68.   @Override
69.   public ClientsResponse getAllClients() {
70.     return webClient.getAllClients();
71.   }
72.
73.   @Override
74.   public MedecinsResponse getAllMedecins() {
75.     return webClient.getAllMedecins();
76.   }
77.
78.   @Override
79.   public CreneauxForMedecinResponse getAllCreneaux(long idMedecin) {
80.     return webClient.getAllCreneaux(idMedecin);
81.   }
82.
83.   @Override
84.   public RvMedecinJourResponse getRvMedecinJour(long idMedecin, String jour) {
85.     return webClient.getRvMedecinJour(idMedecin, jour);
86.   }
87.
88.   @Override
89.   public ClientResponse getClientById(long id) {
90.     return webClient.getClientById(id);
91.   }
92.
93.   @Override
94.   public MedecinResponse getMedecinById(long id) {
95.     return webClient.getMedecinById(id);
96.   }
97.
98.   @Override
99.   public RvResponse getRvById(long id) {
100.    return webClient.getRvById(id);
101.  }
102.
103.  @Override
104.  public CreneauResponse getCreneauById(long id) {
105.    return webClient.getCreneauById(id);
106.  }
107.
108.  @Override
109.  public RvResponse ajouterRv(String jour, long idCreneau, long idClient) {
110.    return webClient.ajouterRv(new PostAjouterRv(jour, idCreneau, idClient));
111.  }
112.
113.  @Override
114.  public RvResponse supprimerRv(long idRv) {
115.    return webClient.supprimerRv(new PostSupprimerRv(idRv));
116.  }
117.
118.  @Override
119.  public AgendaMedecinJourResponse getAgendaMedecinJour(long idMedecin, String jour) {
120.    return webClient.getAgendaMedecinJour(idMedecin, jour);
```

```
121.   }
122. }
```

- lignes 24-25 : le composant AA [WebClient] est injecté dans le champ [webClient] ;
- lignes 69-121 : les méthodes qui interrogent les URL exposées par le service web délèguent cette interrogation au composant AA [WebClient] ;
- lignes 36-47 : création du composant [RestTemplate] ;
- ligne 38 : le [HttpComponentsClientHttpRequestFactory] va nous permettre de fixer le [timeout] du client HTTP ;
- ligne 39 : le composant [RestTemplate] est créé avec le composant précédent ;
- ligne 40 : on indique que les échanges client / serveur doivent être traités par la librairie jSON Jackson ;
- lignes 42-44 : on ajoute un intercepteur au composant [RestTemplate]. Ceci va avoir pour effet que toute requête HTTP émise par le composant [RestTemplate] va être interceptée par la classe [MyAuthInterceptor] ;

La classe [MyAuthInterceptor] est la suivante :

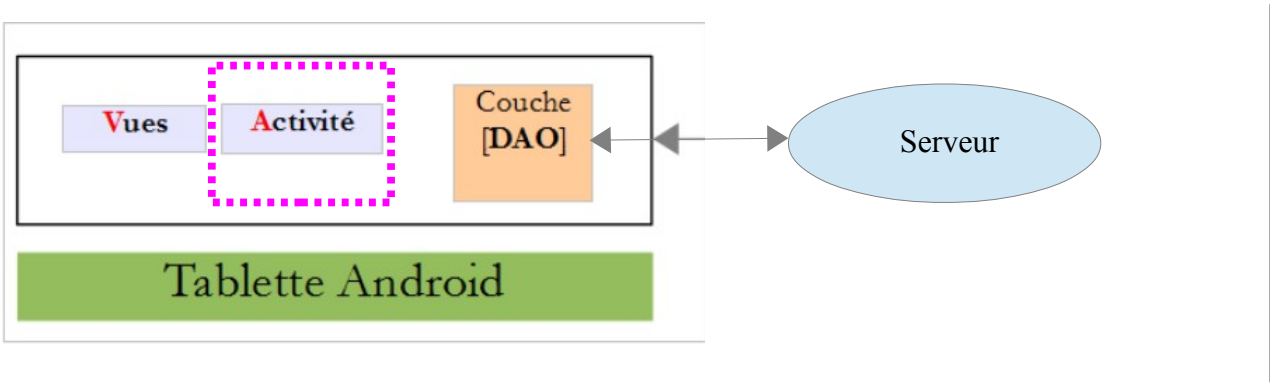
```
1.  package rdvmedecins.android.dao.security;
2.
3.  import org.androidannotations.annotations.Bean;
4.  import org.androidannotations.annotations.EBean;
5.  import org.springframework.http.HttpAuthentication;
6.  import org.springframework.http.HttpBasicAuthentication;
7.  import org.springframework.http.HttpHeaders;
8.  import org.springframework.http.HttpRequest;
9.  import org.springframework.http.client.ClientHttpRequestExecution;
10. import org.springframework.http.client.ClientHttpRequestInterceptor;
11. import org.springframework.http.client.ClientHttpResponse;
12.
13. import java.io.IOException;
14.
15. @EBean(scope = EBean.Scope.Singleton)
16. public class MyAuthInterceptor implements ClientHttpRequestInterceptor {
17.
18.   // utilisateur
19.   private String user;
20.   private String mdp;
21.
22.   public ClientHttpResponse intercept(HttpRequest request, byte[] body, ClientHttpRequestExecution
    execution) throws IOException {
23.     HttpHeaders headers = request.getHeaders();
24.     HttpAuthentication auth = new HttpBasicAuthentication(user, mdp);
25.     headers.setAuthorization(auth);
26.     return execution.execute(request, body);
27.   }
28.
29.   public void setUser(String user, String mdp) {
30.     this.user = user;
31.     this.mdp = mdp;
32.   }
33. }
```

- ligne 15 : la classe [MyAuthInterceptor] est un composant AA de type [singleton] ;
- ligne 16 : la classe [MyAuthInterceptor] étend l'interface Spring [ClientHttpRequestInterceptor]. Cette interace a une méthode, la méthode [intercept] de la ligne 22. On étend cette interface pour intercepter toute requête HTTP du client. La méthode [intercept] reçoit trois paramètres ;
  - [HtpRequest request] : la requête HTTP interceptée,
  - [byte[] body] : son corps si elle en a un (des valeurs postées par exemple),
  - [ClientHttpRequestExecution execution] : le composant Spring qui exécute la requête ;

  Nous interceptons toutes les requêtes HTTP du client Android pour lui ajouter l'entête HTTP d'authentification présenté au paragraphe 2.5, page 210.

- ligne 23 : nous récupérons les entêtes HTTP de la requête interceptée ;
- ligne 24 : nous créons l'entête HTTP d'authentification. Le mode d'authentification utilisé (codage base64 de la chaîne 'user:mdp') est fourni par la classe Spring [HttpBasicAuthentication] ;
- ligne 25 : l'entête d'authentification que nous venons de créer est ajouté aux entêtes actuels de la requête interceptée ;
- ligne 26 : on poursuit l'exécution de la requête interceptée. Si nous résumons, la requête interceptée a été enrichie de l'entête d'authentification ;

2.6.5 L'activité



La classe [MainActivity] est la suivante :

```
1.  package rdvmedecins.android.activity;
2.
3.  import android.os.Bundle;
4.  import android.support.v4.app.FragmentActivity;
5.  import android.support.v4.app.FragmentManager;
6.  import android.support.v4.app.FragmentPagerAdapter;
7.  import android.view.View;
8.  import android.view.Window;
9.  import android.widget.TextView;
10. import org.androidannotations.annotations.*;
11. import rdvmedecins.android.R;
12. import rdvmedecins.android.dao.responses.*;
13. import rdvmedecins.android.dao.service.Dao;
14. import rdvmedecins.android.dao.service.IDao;
15. import rdvmedecins.android.fragments.*;
16.
17. @EActivity(R.layout.main)
18. public class MainActivity extends FragmentActivity implements IDao {
19.
20.   // couche [DAO]
21.   @Bean(Dao.class)
22.   IDao dao;
23.   // la session
24.   @Bean(SessionModel.class)
25.   SessionModel session;
26.
27.   // le conteneur des fragments
28.   @ViewById(R.id.pager)
29.   MyPager mViewPager;
30.
31.   // les liens de navigation
32.   @ViewById(R.id.lnk_Config)
33.   TextView lnkConfig;
34.   @ViewById(R.id.lnk_Accueil)
35.   TextView lnkAccueil;
36.   @ViewById(R.id.lnk_Agenda)
37.   TextView lnkAgenda;
38.   // le tableau des liens
39.   TextView[] links;
```

```
40.
41.   // le gestionnaire de fragments ou sections
42.   private SectionsPagerAdapter mSectionsPagerAdapter;
43.
44.
45.   @Override
46.   protected void onCreate(Bundle savedInstanceState) {
47.     // classique
48.     super.onCreate(savedInstanceState);
49.     // il faut installer le sablier avant de créer la vue
50.     requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
51.   }
52.
53.   @AfterViews
54.   protected void initActivity() {
55.     // on inhibe le swipe entre fragments
56.     mViewPager.setSwipeEnabled(false);
57.     // instanciation du gestionnaire de fragments
58.     mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
59.     // qu'on associe au gestionnaire de fragments
60.     mViewPager.setAdapter(mSectionsPagerAdapter);
61.     // tableau des liens de navigation
62.     links = new TextView[]{lnkConfig, lnkAccueil, lnkAgenda};
63.     // configuration de la couche [DAO]
64.     dao.setTimeout(Constants.TIMEOUT);
65.     // on affiche la vue de configuration
66.     showView(Constants.VUE_CONFIG);
67.   }
68.
69. ...
70.
71.   // interface IDao -----------------------------------------------------
72.  ...
73.
74.   // gestionnaire de fragments
75.   public class SectionsPagerAdapter extends FragmentPagerAdapter {
76.
77.     // les fragments
78.     MyFragment[] fragments = {new ConfigFragment_(), new AccueilFragment_(), new AgendaFragment_(), new
    AjoutRvFragment_()};
79.
80.     // constructeur
81.     public SectionsPagerAdapter(FragmentManager fm) {
82.       super(fm);
83.     }
84.
85.     // doit rendre le fragment n° i avec ses éventuels arguments
86.     @Override
87.     public MyFragment getItem(int position) {
88.       // on rend le fragment
89.       return fragments[position];
90.     }
91.
92.     // rend le nombre de fragments à gérer
93.     @Override
94.     public int getCount() {
95.       return fragments.length;
96.     }
97.
98.     // rend le titre du fragment n° position
99.     @Override
100.    public CharSequence getPageTitle(int position) {
101.      return String.format("titre[%d]", position);
102.    }
103.  }
104.}
```

- ligne 17 : l'activité est associée au patron XML des vues [main.xml] ;
- ligne 18 : la classe [MainActivity] implémente l'interface [IDao] de la couche [DAO] qu'elle offre ainsi aux vues. L'intérêt est que celles-ci n'ont pas à connaître la couche [DAO] ;
- lignes 21-22 : injection de la couche [DAO] ;
- lignes 28-29 : référence sur la zone d'accueil des fragments à l'intérieur du patron [main.xml] ;
- lignes 32-37 : les références sur les trois liens de navigation arrière ;

- ligne 39 : le tableau de ces trois liens ;
- lignes 75-104 : le gestionnaire des fragments (vues) ;
- ligne 78 : il y a au total quatre vues ;
- lignes 54-67 : initialisation de l'activité ;
- ligne 62 : on mémorise les trois liens de navigation ;
- ligne 64 : on fixe le [timeout] de la couche [DAO] ;
- ligne 66 : on affiche la 1ère vue, celle qui saisit les informations de configuration ;

La gestion de la navigation est faite avec les méthodes suivantes :

```
1.  // afichage vue n° [position]
2.    public void showView(int position) {
3.      // on mémorise le n° de la vue à afficher
4.      session.setNumVue(position);
5.      // on rafraîchit le fragment
6.      mSectionsPagerAdapter.getItem(position).onRefresh();
7.      // on affiche la vue demandée
8.      mViewPager.setCurrentItem(position);
9.      // maj des liens de navigation
10.     activateLinks(position);
11.   }
12.
13.   // navigation ---------------------
14.   @Click(R.id.lnk_Config)
15.   void showConfig() {
16.     showView(Constants.VUE_CONFIG);
17.   }
18.
19.   @Click(R.id.lnk_Accueil)
20.   void showAccueil() {
21.     showView(Constants.VUE_ACCUEIL);
22.   }
23.
24.   @Click(R.id.lnk_Agenda)
25.   void showAgenda() {
26.     showView(Constants.VUE_AGENDA);
27.   }
28.
29.   // gestion des liens
30.   private void activateLinks(int idLink) {
31.     // on active tous les liens qui précèdent idLink
32.     int i = 0;
33.     while (i < idLink) {
34.       links[i].setVisibility(View.VISIBLE);
35.       i++;
36.     }
37.     // on désactive le lien idLink et les suivants
38.     while (i < links.length) {
39.       links[i].setVisibility(View.INVISIBLE);
40.       i++;
41.     }
42.   }
```

- ligne 2 : la méthode [showView] affiche l'une des quatre vues via le n° de celle-ci dans [0,3] ;
- ligne 4 : ce n° est mémorisé en session. Cet objet [session] va nous servir de moyen de communication entre fragments. Il est instancié en un seul exemplaire et injecté dans l'activité et les fragments. Dans l'activité décrite précédemment, il est injecté aux lignes 24-25 ;
- ligne 6 : avant d'afficher le fragment, on prépare ces données. Tous les fragments implémentent une interface appelée [OnRefreshListener] qui n'a qu'une méthode [onRefresh] sans paramètres. Dans cette méthode, le fragment va chercher dans la session les éléments qu'il doit afficher. Le principe est le suivant. Un fragment F1 est affiché et un événement est traité. A l'issue de cet événement, le fragment F1 met dans la session les informations qui peuvent être utiles aux autres fragments. Puis le fragment F1 passe la main au fragment F2. Celui-ci avant de s'afficher va chercher en session ce que le fragment F1 a mis pour lui et met à jour ses composants avec ces informations ;
- ligne 8 : la vue est affichée ;
- ligne 10 : les liens de navigation arrière sont mis à jour. Le principe est le suivant (lignes 30-42) : lorsque la vue n° i est affichée, les liens de n°s [0, i-1] doivent être visibles, les autres non ;
- lignes 14-27 : les gestionnaires des liens de navigation arrière ;

Un certain nombre de constantes sont définies dans la classe [Constants] :

```
1.  package rdvmedecins.android.activity;
2.
3.  abstract public class Constants {
4.
5.    final static public int VUE_CONFIG = 0;
6.    final static public int VUE_ACCUEIL = 1;
7.    final static public int VUE_AGENDA = 2;
8.    final static public int VUE_AJOUT_RV = 3;
9.    final static public int DELAY = 0;
10.   final static public int TIMEOUT = 1000;
11. }
```

- lignes 5-8 : les n°s des vues ;
- ligne 9 : un temps d'attente en millisecondes avant le lancement d'une tâche de fond. Nous utiliserons cette constante pour mettre en lumière le mécanisme d'annulation d'une tâche longue. Pour l'instant, l'attente de 0 ms fait qu'il n'y a pas d'attente ;
- ligne 10 : le délai maximal d'attente de la réponse du serveur en millisecondes. Ce délai est ici d'une seconde.

Enfin, l'activité implémente l'interface [IDao] de la façon suivante :

```
1.  // interface IDao ----------------------------------------------------
2.    @Override
3.    public void setUrlServiceWebJson(String url) {
4.      dao.setUrlServiceWebJson(url);
5.    }
6.
7.    @Override
8.    public void setUser(String user, String mdp) {
9.      dao.setUser(user, mdp);
10.   }
11.
12.   @Override
13.   public void setTimeout(int timeout) {
14.     dao.setTimeout(timeout);
15.   }
16.
17.   @Override
18.   public ClientsResponse getAllClients() {
19.     return dao.getAllClients();
20.   }
21.
22.   @Override
23.   public MedecinsResponse getAllMedecins() {
24.     return dao.getAllMedecins();
25.   }
26.
27.   @Override
28.   public CreneauxForMedecinResponse getAllCreneaux(long idMedecin) {
29.     return dao.getAllCreneaux(idMedecin);
30.   }
31.
32.   @Override
33.   public RvMedecinJourResponse getRvMedecinJour(long idMedecin, String jour) {
34.     return dao.getRvMedecinJour(idMedecin, jour);
35.   }
36.
37.   @Override
38.   public ClientResponse getClientById(long id) {
39.     return dao.getClientById(id);
40.   }
41.
42.   @Override
43.   public MedecinResponse getMedecinById(long id) {
44.     return dao.getMedecinById(id);
45.   }
46.
47.   @Override
48.   public RvResponse getRvById(long id) {
49.     return dao.getRvById(id);
50.   }
51.
52.   @Override
```

```
53.   public CreneauResponse getCreneauById(long id) {
54.     return dao.getCreneauById(id);
55.   }
56.
57.   @Override
58.   public RvResponse ajouterRv(String jour, long idCreneau, long idClient) {
59.     return dao.ajouterRv(jour, idCreneau, idClient);
60.   }
61.
62.   @Override
63.   public RvResponse supprimerRv(long idRv) {
64.     return dao.supprimerRv(idRv);
65.   }
66.
67.   @Override
68.   public AgendaMedecinJourResponse getAgendaMedecinJour(long idMedecin, String jour) {
69.     return dao.getAgendaMedecinJour(idMedecin, jour);
70.   }
```

L'activité se contente de déléguer à la couche [DAO] qui lui a été injectée les appels qu'on lui fait.

2.6.6 La session

La classe [SessionModel] sert à mémoriser les informations qui doivent être transmises entre fragments. Elle est la suivante :

```
1.  package rdvmedecins.android.activity;
2.
3.  import org.androidannotations.annotations.EBean;
4.  import rdvmedecins.android.dao.entities.AgendaMedecinJour;
5.  import rdvmedecins.android.dao.entities.Client;
6.  import rdvmedecins.android.dao.entities.Medecin;
7.
8.  import java.io.Serializable;
9.  import java.util.List;
10.
11. @EBean(scope = EBean.Scope.Singleton)
12. public class SessionModel implements Serializable {
13.   // liste des médecins
14.   private List<Medecin> médecins;
15.   // liste des clients
16.   private List<Client> clients;
17.   // agenda
18.   private AgendaMedecinJour agenda;
19.   // position de l'élément cliqué dans l'agenda
20.   private int position;
21.   // jour du Rv en notation anglaise "yyyy-MM-dd"
22.   private String dayRv;
23.   // jour du Rv en notation française "dd-MM-yyyy"
24.   private String jourRv;
25.   // vue courante
26.   private int numVue;
27.
28.   // getters et setters
29.   ...
30. }
```

- ligne 11 : la classe [SessionModel] est un composant AA instancié en un unique exemplaire ;
- lignes 13-16 : on supposera dans cette étude de cas, que les listes de médecins et de clients ne changent pas. On les demandera au démarrage de l'application et on les stockera dans la session pour que les fragments puissent les utiliser ;
- lignes 21-24 : le jour souhaité pour un rendez-vous. Il est manipulé sous deux formes, en notation française (ligne 24) au sein du client Android, en notation anglaise (ligne 22) pour les échanges avec le serveur ;
- ligne 26 : le n° de la vue actuellement affichée ;
- ligne 20 : la position de l'élément cliqué (ajouter / supprimer) sur l'agenda. Lorsqu'on quitte puis revient sur l'agenda, on veut pouvoir se repositionner sur l'élément cliqué ;

2.6.7 Le patron des vues

La vue [main.xml] est un conteneur dans lequel viennent s'insérer les quatre vues de l'application. Son code est le suivant :

```
1.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.                xmlns:tools="http://schemas.android.com/tools"
3.                android:layout_width="match_parent"
4.                android:layout_height="match_parent"
5.                android:gravity="center"
6.                android:orientation="vertical">
7.
8.    <LinearLayout
9.      android:id="@+id/header"
10.     android:layout_width="match_parent"
11.     android:layout_height="100dp"
12.     android:layout_weight="0.1"
13.     android:background="@color/lavenderblushh2">
14.
15.     <TextView
16.       android:id="@+id/textViewHeader"
17.       android:layout_width="match_parent"
18.       android:layout_height="wrap_content"
19.       android:layout_gravity="center"
20.       android:gravity="center_horizontal"
21.       android:text="@string/txt_header"
22.       android:textAppearance="?android:attr/textAppearanceLarge"
23.       android:textColor="@color/blue"/>
24.   </LinearLayout>
25.
26.   <LinearLayout
27.     android:layout_width="match_parent"
28.     android:layout_height="fill_parent"
29.     android:layout_weight="0.8"
30.     android:orientation="horizontal">
31.
32.     <LinearLayout
33.       android:id="@+id/left"
34.       android:layout_width="100dp"
35.       android:layout_height="match_parent"
36.       android:background="@color/lightcyan2"
37.       android:orientation="vertical">
38.
39.       <TextView
40.         android:id="@+id/lnk_Config"
41.         android:layout_width="fill_parent"
42.         android:layout_height="40dp"
43.         android:layout_marginTop="100dp"
44.         android:gravity="center_vertical|center_horizontal"
45.         android:text="@string/lnk_config"
46.         android:textColor="@color/blue"/>
47.
48.       <TextView
49.         android:id="@+id/lnk_Accueil"
50.         android:layout_width="fill_parent"
51.         android:layout_height="40dp"
```

```
52.         android:gravity="center_vertical|center_horizontal"
53.         android:text="@string/lnk_accueil"
54.         android:textColor="@color/blue"/>
55.
56.       <TextView
57.         android:id="@+id/lnk_Agenda"
58.         android:layout_width="fill_parent"
59.         android:layout_height="40dp"
60.         android:gravity="center_vertical|center_horizontal"
61.         android:text="@string/lnk_agenda"
62.         android:textColor="@color/blue"/>
63.
64.     <rdvmedecins.android.activity.MyPager
65.       xmlns:android="http://schemas.android.com/apk/res/android"
66.       xmlns:tools="http://schemas.android.com/tools"
67.       android:id="@+id/pager"
68.       android:layout_width="match_parent"
69.       android:layout_height="match_parent"
70.       android:background="@color/floral_white"/>
71.   </LinearLayout>
72.
73.   <LinearLayout
74.     android:id="@+id/bottom"
75.     android:layout_width="match_parent"
76.     android:layout_height="100dp"
77.     android:layout_weight="0.1"
78.     android:background="@color/wheat1">
79.
80.     <TextView
81.       android:id="@+id/textViewBottom"
82.       android:layout_width="fill_parent"
83.       android:layout_height="fill_parent"
84.       android:gravity="center_vertical|center_horizontal"
85.       android:text="@string/txt_bottom"
86.       android:textColor="@color/blue"/>
87.   </LinearLayout>
88.
89. </LinearLayout>
```

- lignes 64-71 : les quatre vues viendront s'insérer dans la zone d'id [pager] (ligne 76) ;
- lignes 39-62 : trois liens de navigation arrière sont affichés dans le bandeau gauche ;

Voici ces deux zones lorsque la vue de configuration est affichée :

- en [1], le bandeau gauche des liens ;
- en [2], la vue insérée dans le patron ;

2.6.8 La classe mère des fragments

Les quatre fragments associés aux quatre vues partagent des informations et des comportements qui ont été factorisés dans la classe [MyFragment] suivante :

```
1.  package rdvmedecins.android.fragments;
2.
3.  import android.app.AlertDialog;
4.  import android.os.Bundle;
5.  import android.support.v4.app.Fragment;
6.  import android.view.View;
7.  import android.view.View.OnClickListener;
8.  import android.widget.TextView;
9.  import org.androidannotations.annotations.AfterViews;
10. import org.androidannotations.annotations.Bean;
11. import org.androidannotations.annotations.EFragment;
12. import org.androidannotations.annotations.ViewById;
13. import rdvmedecins.android.R;
14. import rdvmedecins.android.activity.Constants;
15. import rdvmedecins.android.activity.MainActivity;
16. import rdvmedecins.android.activity.SessionModel;
17.
18. import java.util.ArrayList;
19. import java.util.List;
20.
21. @EFragment
22. public abstract class MyFragment extends Fragment implements OnRefreshListener {
23.   // encapsule des méthodes et données communes aux classes filles
24.   // la session
25.   @Bean(SessionModel.class)
26.   SessionModel session;
27.   // l'activité unique
28.   protected MainActivity activité;
29.
30.   @Override
31.   public void onCreate(Bundle savedInstanceState) {
32.     // parent
33.     super.onActivityCreated(savedInstanceState);
34.     // on mémorise l'activité
35.     activité = (MainActivity) getActivity();
36.   }
37.
38.   // affichage exception
39.   protected void showMessages(List<String> messages) {
40.     // on construit le texte à afficher
41.     StringBuilder texte = new StringBuilder();
42.     for (String message : messages) {
43.       texte.append(String.format("%s\n", message));
44.     }
45.     // on l'affiche
46.     new AlertDialog.Builder(activité).setTitle("Des erreurs se sont
    produites").setMessage(texte).setNeutralButton("Fermer", null).show();
```

```
47.   }
48.
49.   // affichage d'une exception
50.   public List<String> getMessagesFromException(Exception exception) {
51.     // on récupère la liste des messages d'erreur de l'exception
52.     Throwable cause = exception;
53.     List<String> messages = new ArrayList<String>();
54.     while (cause != null) {
55.       messages.add(cause.getMessage());
56.       cause = cause.getCause();
57.     }
58.     return messages;
59.   }
60.
61.   // gestion du sablier
62.   protected void showHourGlass() {
63.     activité.setProgressBarIndeterminateVisibility(true);
64.   }
65.
66.   protected void hideHourGlass() {
67.     activité.setProgressBarIndeterminateVisibility(false);
68.   }
69.
70. }
```

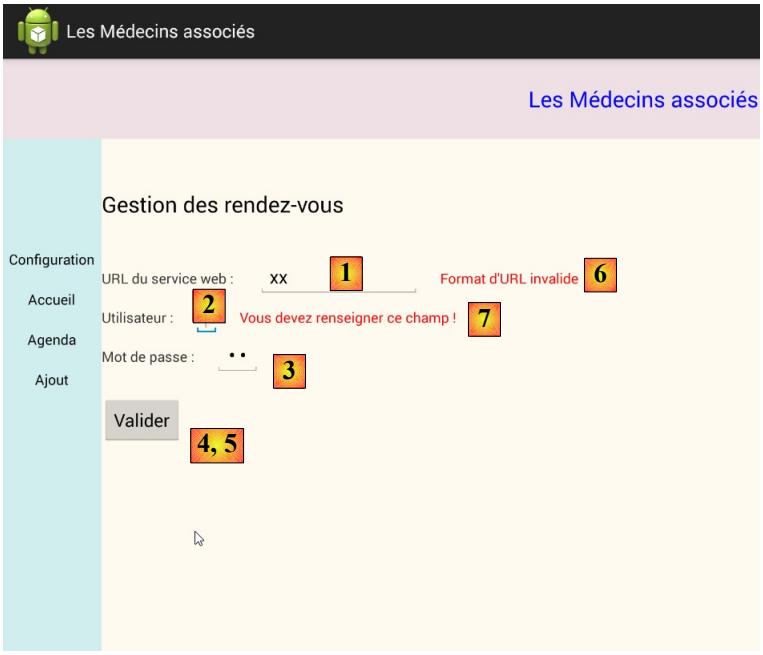
- ligne 21 : [MyFragment] est un composant AA ;
- ligne 22 : [MyFragment] implémente l'interface [OnRefreshListener] suivante :

```
1.  package rdvmedecins.android.fragments;
2.
3.  public interface OnRefreshListener {
4.     public void onRefresh();
5.  }
```

- parce que ce sont les classes filles qui implémentent cette interface, [MyFragment] est déclarée abstraite (ligne 22) ;
- lignes 25-28 : les informations nécessaires à tous les fragments ;
- ligne 26 : la session qui permet la communication inter-fragments ;
- ligne 28 : l'activité qui gère les changements de vue ;
- lignes 31-36 : la méthode exécutée lors de la création initiale de chaque fragment. Chaque fragment mémorise l'activité qui le gère ;
- lignes 50-59 : la méthode [getMessagesFromException] sert à obtenir la liste des messages d'erreur issues d'une pile d'exceptions ;
- lignes 39-47 : la méthode [showMessages] affiche dans une boîte de dialogue les messages d'erreur envoyées par le serveur dans sa réponse. On a vu au paragraphe 2.5, page 210 que toutes les réponses avaient les deux éléments [int status ; List<String> messages]. Il s'agit ici d'afficher le champ [messages] ;
- lignes 62-68 : gestion de l'indicateur d'attente appelé ici sablier (hourglass) ;

2.6.9 Gestion de la vue de configuration

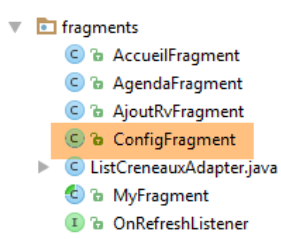
La vue de configuration est la vue affichée au démarrage de l'application :



Les éléments de l'interface visuelle sont les suivants :

n°	Type	Nom
1	EditText	edtUrlServiceRest
2	EditText	edtUtilisateur
3	EditText	edtMdp
4	Button	btnValider
5	Button	btnAnnuler
6	TextView	txtErrorUrlServiceRest
7	TextView	txtErrorUtilisateur

Les boutons [4] et [5] sont superposés. La vue de configuration est gérée par le fragment [ConfigFragment] suivant :




```
1.  package rdvmedecins.android.fragments;
2.
3.  import android.view.View;
4.  import android.widget.Button;
5.  import android.widget.EditText;
6.  import android.widget.TextView;
7.  import org.androidannotations.annotations.*;
8.  import org.androidannotations.api.BackgroundExecutor;
9.  import rdvmedecins.android.R;
10. import rdvmedecins.android.activity.Constants;
11. import rdvmedecins.android.dao.responses.ClientsResponse;
12. import rdvmedecins.android.dao.responses.MedecinsResponse;
```

```
13.
14. import java.net.URI;
15.
16. @EFragment(R.layout.config)
17. public class ConfigFragment extends MyFragment {
18.
19.   // les éléments de l'interface visuelle
20.   @ViewById(R.id.btn_Valider)
21.   Button btnValider;
22.   @ViewById(R.id.btn_Annuler)
23.   Button btnAnnuler;
24.   @ViewById(R.id.edt_urlServiceRest)
25.   EditText edtUrlServiceRest;
26.   @ViewById(R.id.txt_errorUrlServiceRest)
27.   TextView txtErrorUrlServiceRest;
28.   @ViewById(R.id.txt_errorUtilisateur)
29.   TextView txtErrorUtilisateur;
30.   @ViewById(R.id.edt_utilisateur)
31.   EditText edtUtilisateur;
32.   @ViewById(R.id.edt_mdp)
33.   EditText edtMdp;
34.
35.   // les saisies
36.   private String urlServiceRest;
37.   private String utilisateur;
38.   private String mdp;
39.
40.   // données locales
41.   private int nbRéponses;
42.   private boolean afterViewsDone = false;
43.   private boolean canceled;
44.
45.   // constructeur
46.   public ConfigFragment() {
47.     System.out.println("ConfigFragment constructeur");
48.   }
49.
50.   @AfterViews
51.   void afterViews() {
52.     afterViewsDone = true;
53.     System.out.println("Config afterViews");
54.     if (session.getNumVue() == Constants.VUE_CONFIG) {
55.       initFragment();
56.     }
57.   }
58.
59.   @Override
60.   public void onRefresh() {
61.     System.out.println("Config refresh");
62.     if (afterViewsDone) {
63.       initFragment();
64.     }
65.   }
66.
67.   private void initFragment() {
68.     System.out.println("Config init");
69.     // au départ pas de messages d'erreur
70.     txtErrorUrlServiceRest.setVisibility(View.INVISIBLE);
71.     txtErrorUtilisateur.setVisibility(View.INVISIBLE);
72.     // état des boutons
73.     btnValider.setVisibility(View.VISIBLE);
74.     btnAnnuler.setVisibility(View.INVISIBLE);
75.   }
```

```
76.
77.   // validation de la page
78.   @Click(R.id.btn_Valider)
79.   protected void doValider() {
80.   ...
81.   }
82.
83.   @Click(R.id.btn_Annuler)
84.   protected void doAnnuler() {
85.  ...
86.   }
87.
88.   @Background(id = "clients", delay = Constants.DELAY)
89.   void getClientsInBackground() {
90.    ...
91.   }
92.
93.   @UiThread
94.   void manageClientsResponse(ClientsResponse response) {
95.    ...
96.   }
97.
98.   @Background(id = "medecins", delay = Constants.DELAY)
99.   void getMedecinsInBackground() {
100.      ...
101.    }
102.
103.    @UiThread
104.    void manageMedecinsResponse(MedecinsResponse response) {
105.      ...
106.    }
107.
108.    private boolean isPageValid() {
109.      ...
110.    }
111.
112.    // début de l'attente
113.    private void beginWaiting() {
114.      // on met le sablier
115.      showHourGlass();
116.      // état des boutons
117.      btnValider.setVisibility(View.INVISIBLE);
118.      btnAnnuler.setVisibility(View.VISIBLE);
119.
120.    }
121.
122.    // fin de l'attente
123.    protected void cancelWaiting() {
124.      // on cache le sablier
125.      hideHourGlass();
126.      // état des boutons
127.      btnValider.setVisibility(View.VISIBLE);
128.      btnAnnuler.setVisibility(View.INVISIBLE);
129.    }
130.
131.  }
```

- lignes 20-33 : les éléments de l'interface visuelle ;
- lignes 36-38 : les trois saisies du formulaire ;
- lignes 46-48 : le constructeur du fragment. On fait un affichage pour que le lecteur puisse vérifier dans les logs que chaque fragment n'est instancié qu'une fois ;
- lignes 51-57 : la méthode exécutée après que les éléments de l'interface visuelle aient été initialisés (lignes 21-34). On a fait là également un affichage. Les logs montrent que cette méthode peut être exécutée plus d'une fois et à des moments

inattendus. Ainsi lorsque la vue de configuration est affichée, les logs montrent que la méthode [@AfterViews] du fragment [AccueilFragment] est appelée ;
- lignes 54-56 : on n'initialise le fragment que si c'est bien la vue de configuration qui va être affichée ;
- lignes 60-65 : on rappelle que la méthode [onRefresh] est appelée systématiquement lorsque la vue va être affichée. Elle doit initialiser l'interface visuelle qui va être affichée. Les logs montrent que cette méthode peut être appelée avant la méthode [afterViews]. On vérifie donc que cette dernière a bien été exécutée, avant de faire l'initialisation de l'interface visuelle ;
- lignes 67-75 : initialisation de l'interface visuelle ;

Le clic sur le lien [Valider] est géré par la méthode suivante :

```
1.    @Click(R.id.btn_Valider)
2.    protected void doValider() {
3.      // on cache les éventuels msg d'erreur précédents
4.      txtErrorUrlServiceRest.setVisibility(View.INVISIBLE);
5.      txtErrorUtilisateur.setVisibility(View.INVISIBLE);
6.      // on teste la validité des saisies
7.      if (!isPageValid()) {
8.        return;
9.      }
10.     // on renseigne l'URL du service web
11.     activité.setUrlServiceWebJson(urlServiceRest);
12.     // on renseigne l'utilisateur
13.     activité.setUser(utilisateur, mdp);
14.     // on demande la liste des médecins et des clients
15.     nbRéponses = 0;
16.     canceled=false;
17.     getMedecinsInBackground();
18.     getClientsInBackground();
19.     // début de l'attente
20.     beginWaiting();
21. }
```

- lignes 7-9 : la validité des trois saisies du formulaire est testée. Si le formulaire est invalide, on ne va pas plus loin ;
- lignes 11-13 : les trois saisies sont passées à l'activité. Elles vont servir à configurer la couche [DAO] ;
- lignes 15-18 : la liste des médecins et la liste des clients sont demandées en tâches de fond. Celles-ci vont être exécutées en parallèle ;
- ligne 20 : on commence l'attente ;

La validité des saisies est vérifiée avec la méthode suivante :

```
1.  private boolean isPageValid() {
2.      // on vérifie la validité des données saisies
3.      Boolean erreur = false;
4.      URI service = null;
5.      // validité de l'URL du service REST
6.      urlServiceRest = String.format("http://%s", edtUrlServiceRest.getText().toString().trim());
7.      try {
8.        service = new URI(urlServiceRest);
9.        erreur = service.getHost() == null || service.getPort() == -1;
10.     } catch (Exception ex) {
11.       // on note l'erreur
12.       erreur = true;
13.     }
14.     if (erreur) {
15.       // affichage erreur
16.       txtErrorUrlServiceRest.setVisibility(View.VISIBLE);
17.     }
18.     // utilisateur
19.     utilisateur = edtUtilisateur.getText().toString().trim();
20.     if (utilisateur.length() == 0) {
21.       // on affiche l'erreur
22.       txtErrorUtilisateur.setVisibility(View.VISIBLE);
23.       // on note l'erreur
24.       erreur = true;
25.     }
26.     // mot de passe
27.     mdp = edtMdp.getText().toString().trim();
28.     // retour
29.     return !erreur;
30. }
```

La liste des médecins est obtenue de la façon suivante :

```
1.  @Background(id = "medecins", delay = Constants.DELAY)
2.    void getMedecinsInBackground() {
3.      MedecinsResponse response;
4.      try {
5.        // on demande la liste des médecins
6.        response = activité.getAllMedecins();
7.      } catch (Exception e) {
8.        response = new MedecinsResponse();
9.        response.setStatus(1);
10.       response.setMessages(getMessagesFromException(e));
11.     }
12.     // on exploite la réponse
13.     manageMedecinsResponse(response);
14.   }
15.
16.   @UiThread
17.   void manageMedecinsResponse(MedecinsResponse response) {
18.     // tâches annulées ?
19.     if(canceled){
20.       return;
21.     }
22.     // une réponse de +
23.     nbRéponses++;
24.     if (nbRéponses == 2) {
25.       cancelWaiting();
26.     }
27.     // analyse de la réponse
28.     if (response.getStatus() == 0) {
29.       // on mémorise les médecins dans la session
30.       session.setMédecins(response.getMedecins());
31.     } else {
32.       // on affiche l'erreur
33.       showMessages(response.getMessages());
34.       // on arrête tout
35.       doAnnuler();
36.       return;
37.     }
38.     // chgt de vue ?
39.     if (nbRéponses == 2) {
40.       activité.showView(Constants.VUE_ACCUEIL);
41.     }
42.   }
```

- ligne 6 : la liste des médecins est demandée à l'activité. On se rappelle que celle-ci implémente l'interface [IDao] de la couche [DAO] ;
- lignes 8-10 : en cas d'exception, un objet [MedecinsResponse] est construit avec les messages d'erreur de l'exception ;
- ligne 13 : dans tous les cas, l'objet [MedecinsResponse] reçu ou construit est passé à la méthode [manageMedecinsResponse] ;
- lignes 19-21 : avant de traiter la réponse, on vérifie si l'utilisateur a annulé l'opération demandée. Si oui, on ne fait rien ;
- lignes 23-26 : on attendait deux réponses (médecins et clients). Si elles ont été reçues, on annule l'attente ;
- ligne 30 : les médecins sont mis dans la session. Dorénavant c'est là qu'ils seront cherchés ;
- lignes 33-36 : s'il y a eu erreur, alors celle-ci est affichée (ligne 33) et l'opération de configuration est annulée (ligne 35) ;
- lignes 39-41 : si on a reçu les deux réponses (clients et médecins), on passe à la vue d'accueil ;

Les clients sont traités de façon similaire :

```
1.  @Background(id = "clients", delay = Constants.DELAY)
2.    void getClientsInBackground() {
3.      ClientsResponse response;
4.      try {
5.        // on demande la liste des clients
6.        response = activité.getAllClients();
7.      } catch (Exception e) {
8.        response = new ClientsResponse();
9.        response.setStatus(1);
10.       response.setMessages(getMessagesFromException(e));
11.     }
12.     // on exploite la réponse
```

```
13.     manageClientsResponse(response);
14.   }
15.
16.   @UiThread
17.   void manageClientsResponse(ClientsResponse response) {
18.     // tâches annulées ?
19.     if(canceled){
20.       return;
21.     }
22.     // une réponse de +
23.     nbRéponses++;
24.     if (nbRéponses == 2) {
25.       cancelWaiting();
26.     }
27.     // analyse de la réponse
28.     if (response.getStatus() == 0) {
29.       // on mémorise les clients dans la session
30.       session.setClients(response.getClients());
31.     } else {
32.       // on affiche l'erreur
33.       showMessages(response.getMessages());
34.       // on arrête tout
35.       doAnnuler();
36.       return;
37.     }
38.     // chgt de vue ?
39.     if (nbRéponses == 2) {
40.       activité.showView(Constants.VUE_ACCUEIL);
41.     }
42.   }
```

La méthode d'annulation des tâches de fond est la suivante :

```
1.    @Click(R.id.btn_Annuler)
2.    protected void doAnnuler() {
3.      // on annule les tâches asynchrones
4.      BackgroundExecutor.cancelAll("clients", true);
5.      BackgroundExecutor.cancelAll("medecins", true);
6.      // on annule l'attente
7.      cancelWaiting();
8.      // on note qu'il y a eu annulation
9.      canceled=true;
10. }
```

2.6.10 Gestion de la vue d'accueil

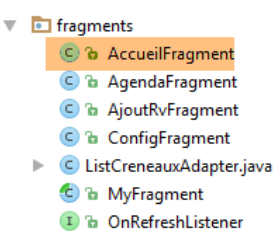
La vue d'accueil permet de choisir un médecin et un jour de rendez-vous :

Les éléments de l'interface visuelle sont les suivants :

n°	Type	Nom
1	Spinner	spinnerMedecins

2	EditText	edtJourRv
3	Button	btnValider
4	Button	btnAnnuler
5	TextView	txtErrorJourRv

La vue d'accueil est gérée par le fragment [AccueilFragment] suivant :



La classe [AccueilFragment] est la suivante :

```
1.  package rdvmedecins.android.fragments;
2.
3.  import android.view.View;
4.  import android.widget.*;
5.  import org.androidannotations.annotations.*;
6.  import org.androidannotations.api.BackgroundExecutor;
7.  import rdvmedecins.android.R;
8.  import rdvmedecins.android.activity.Constants;
9.  import rdvmedecins.android.dao.entities.Medecin;
10. import rdvmedecins.android.dao.responses.AgendaMedecinJourResponse;
11.
12. import java.text.ParseException;
13. import java.text.SimpleDateFormat;
14. import java.util.List;
15. import java.util.Locale;
16.
17. @EFragment(R.layout.accueil)
18. public class AccueilFragment extends MyFragment {
19.
20.   // les éléments de l'interface visuelle
21.   @ViewById(R.id.btn_Valider)
22.   Button btnValider;
23.   @ViewById(R.id.btn_Annuler)
24.   Button btnAnnuler;
25.   @ViewById(R.id.spinnerMedecins)
26.   Spinner spinnerMedecins;
27.   @ViewById(R.id.edt_JourRv)
28.   EditText edtJourRv;
29.   @ViewById(R.id.txt_errorJourRv)
30.   TextView txtErrorJourRv;
31.
32.   // les saisies
33.   private Long idMedecin;
34.   private String jourRv;
35.
36.   // données locales
37.   private List<Medecin> medecins;
38.   private boolean afterViewsDone = false;
39.   private boolean canceled;
40.
41.   // constructeur
```

```
42.   public AccueilFragment() {
43.     System.out.println("AccueilFragment constructeur");
44.   }
45.
46.   @AfterViews
47.   void afterViews() {
48.     afterViewsDone = true;
49.     System.out.println("Accueil afterViews");
50.     if (session.getNumVue() == Constants.VUE_ACCUEIL) {
51.       initFragment();
52.     }
53.   }
54.
55.   @Override
56.   public void onRefresh() {
57.     System.out.println("Accueil refresh");
58.     if (afterViewsDone) {
59.       // init fragment
60.       initFragment();
61.     }
62.   }
63.
64.   private void initFragment() {
65.     System.out.println("Accueil init");
66.     // état des boutons
67.     btnValider.setVisibility(View.VISIBLE);
68.     btnAnnuler.setVisibility(View.INVISIBLE);
69.     // au départ pas d'erreurs
70.     txtErrorJourRv.setVisibility(View.INVISIBLE);
71.     // on gère le spinner des médecins
72.     SpinnerAdapter spinnerAdapter = spinnerMedecins.getAdapter();
73.     if (spinnerAdapter == null) {
74.       // on récupère les médecins en session
75.       medecins = session.getMédecins();
76.       // on construit le tableau affiché par le spinner
77.       String[] arrayMedecins = new String[medecins.size()];
78.       int i = 0;
79.       for (Medecin medecin : medecins) {
80.         arrayMedecins[i] = String.format("%s %s %s", medecin.getTitre(),
    medecin.getPrenom(), medecin.getNom());
81.         i++;
82.       }
83.       // on associe les médecins au spinner
84.       ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(activité,
    android.R.layout.simple_spinner_item,
85.         arrayMedecins);
86.       dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
87.       spinnerMedecins.setAdapter(dataAdapter);
88.     }
89.   }
90.
91.   // validation de la page
92.   @Click(R.id.btn_Valider)
93.   protected void doValider() {
94.   ...
95.   }
96.
97.   @Click(R.id.btn_Annuler)
98.   protected void doAnnuler() {
99. ...
100.     }
101.   ...
102.   }
```

- lignes 21-30 : les composants de l'interface visuelle ;
- lignes 33-34 : les deux saisies du formulaire : l'identifiant du médecin sélectionné (ligne 33) et le jour du rendez-vous (ligne 34) ;
- les méthodes des lignes 47, 56 et 64 assurent l'initialisation du fragment à deux moments :
  - soit après que les composants de l'interface visuelle des lignes 21-30 ont été initialisés (afterViews) ;
  - soit lorsque la vue va être affichée (onRefresh) ;
- lignes 64-89 : initialisation de l'interface visuelle ;
- lignes 72-88 : initialisation de la liste déroulante des médecins ;
- ligne 75 : on récupère les médecins dans la session ;
- lignes 77-82 : on construit le tableau de [String] qui va être associé à la liste déroulante. Chaque élément du tableau sera un élément de la liste affichée ;
- lignes 84-87 : ce tableau est associé à la liste déroulante ;
- lignes 72-73 : parce que ce travail n'a besoin d'être fait qu'une fois, avant de le faire on vérifie si la liste a déjà été associée ou non à une source de données. Si c'est le cas, on ne refait pas le travail précédent ;

La méthode associée au clic sur le bouton [Valider] doit faire afficher l'agenda du médecin sélectionné pour le jour indiqué. La méthode [doValider] de la ligne 93 est la suivante :

```
1.  @Click(R.id.btn_Valider)
2.    protected void doValider() {
3.      // on teste la validité des saisies
4.      if (!isPageValid()) {
5.        return;
6.      }
7.      // on demande l'agenda du médecin
8.      canceled = false;
9.      getAgendaMedecinInBackground();
10.     // début de l'attente
11.     beginWaiting();
12.   }
13.
14.   @Background(id = "agenda", delay = Constants.DELAY)
15.   void getAgendaMedecinInBackground() {
16.     AgendaMedecinJourResponse response;
17.     try {
18.       // on demande l'agenda
19.       response = activité.getAgendaMedecinJour(idMedecin, session.getDayRv());
20.     } catch (Exception e) {
21.       response = new AgendaMedecinJourResponse();
22.       response.setStatus(1);
23.       response.setMessages(getMessagesFromException(e));
24.     }
25.     // gestion de la réponse
26.     manageAgendaMedecinJourResponse(response);
27.   }
28.
29.   private boolean isPageValid() {
30.   ...
31.   }
32.
33.   @UiThread
34.   public void manageAgendaMedecinJourResponse(AgendaMedecinJourResponse response) {
35.     if (!canceled) {
36.       // fin de l'attente
37.       cancelWaiting();
38.       // on gère le résultat
39.       if (response.getStatus() == 0) {
40.         // on met l'agenda dans la session
41.         session.setAgenda(response.getAgenda());
42.         // on affiche la vue de l'agenda
43.         activité.showView(Constants.VUE_AGENDA);
44.       } else {
45.         showMessages(response.getMessages());
46.       }
47.     }
48.   }
```

- lignes 4-6 : avant tout, on vérifie que les saisies sont valides, ici que le jour indiqué est une date valide. A noter, qu'on aurait pu utiliser un calendrier au lieu d'un [TextView] ce qui aurait évité ce test de validité ;

- lignes 8-9 : on demande en tâche de fond l'agenda du médecin sélectionné pour le jour indiqué ;
- ligne 11 : on met en route les indicateurs de l'attente (bouton [Annuler] et widget d'attente) ;
- ligne 19 : l'agenda est demandé à l'activité. Les deux paramètres de la méthode ont été calculés précédemment par la méthode [isPageValid]. Nous allons y revenir ;
- lignes 21-23 : si on rencontre une exception lors de la demande de l'agenda à l'activité, on construit un objet [AgendaMedecinJourResponse] avec la liste des messages d'erreur de l'exception ;
- ligne 26 : l'objet [AgendaMedecinJourResponse] est transmis à une méthode s'exécutant dans le thread de l'UI pour affichage ;
- ligne 35 : si la demande de l'agenda n'a pas déjà été annulée ;
- ligne 37 : l'attente est terminée ;
- ligne 41 : on met l'agenda dans la session. D'autres vues vont l'utiliser ;
- ligne 43 : on passe la main à la vue chargée d'afficher cet agenda ;
- ligne 45 : s'il y a eu erreur, on l'affiche ;

La méthode qui teste la validité des saisies est la suivante :

```
1.  private boolean isPageValid() {
2.      // on note l'id du médecin sélectionné
3.      idMedecin = medecins.get(spinnerMedecins.getSelectedItemPosition()).getId();
4.      // vérification format date
5.      jourRv = edtJourRv.getText().toString();
6.      if (!jourRv.matches("\\s*\\d{2}-\\d{2}-\\d{4}\\s*")) {
7.        // on affiche le msg d'erreur
8.        txtErrorJourRv.setVisibility(View.VISIBLE);
9.        return false;
10.     }
11.     // vérification validité de la date
12.     try {
13.       SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd-MM-yyyy", Locale.FRENCH);
14.       simpleDateFormat.setLenient(false);
15.       simpleDateFormat.parse(jourRv);
16.     } catch (ParseException ex) {
17.       // on l'affiche
18.       txtErrorJourRv.setVisibility(View.VISIBLE);
19.       // fin
20.       return false;
21.     }
22.     // on mémorise le jour dans la session
23.     session.setJourRv(jourRv);
24.     // on passe au format de date yyyy-MM-dd
25.     String[] éléments = jourRv.split("-");
26.     String dayRv = String.format("%s-%s-%s", éléments[2], éléments[1], éléments[0]);
27.     session.setDayRv(dayRv);
28.     // retour
29.     return true;
30.   }
```

- ligne 3 : on récupère l'identifiant du médecin ;
- ligne 5 : on récupère le jour saisi par l'utilisateur ;
- lignes 6-10 : on vérifie que la chaîne saisie a le format attendu ;
- lignes 12-21 : on vérifie que la chaîne saisie qui a un format valide est également une date valide ;
- ligne 23 : on mémorise dans la session le jour saisi au format français 'jj-mm-aaaa' ;
- lignes 25-27 : on transforme ce jour au format 'aaaa-m-jj' et on l'enregistre également dans la session. Ce format est celui utilisé dans les échanges avec le serveur ;

La méthode [doAnnuler] est la suivante :

```
1.    @Click(R.id.btn_Annuler)
2.    protected void doAnnuler() {
3.      // on annule la tâche asynchrone
4.      BackgroundExecutor.cancelAll("agenda", true);
5.      // on annule l'attente
6.      cancelWaiting();
7.      // on note l'annulation
8.      canceled = true;
9.  }
```

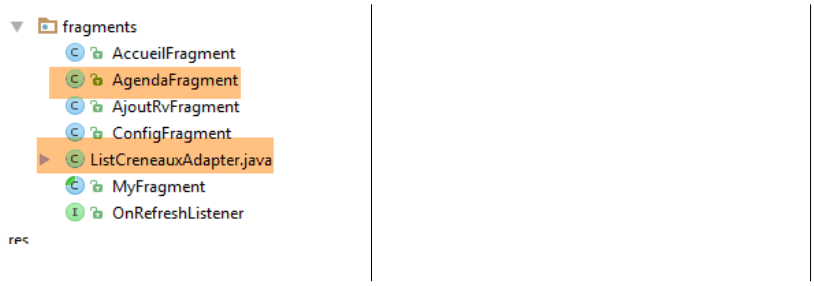
2.6.11 Gestion de la vue Agenda

La vue Agenda permet à l'utilisateur d'ajouter / supprimer un rendez-vous de l'agenda :

Les éléments de l'interface visuelle sont les suivants :

n°	Type	Nom
1	TextView	txtTitre2
2	ListView	lstCreneaux
3	Button	btnAnnuler (caché ici)

La vue Agenda est gérée par le fragment [AgendaFragment] suivant :



La classe [AgendaFragment] est la suivante :

```
1. package rdvmedecins.android.fragments;
2.
3. import android.view.View;
4. import android.widget.ArrayAdapter;
5. import android.widget.Button;
6. import android.widget.ListView;
7. import android.widget.TextView;
8. import org.androidannotations.annotations.*;
9. import org.androidannotations.api.BackgroundExecutor;
```

```
10. import rdvmedecins.android.R;
11. import rdvmedecins.android.activity.Constants;
12. import rdvmedecins.android.dao.entities.AgendaMedecinJour;
13. import rdvmedecins.android.dao.entities.CreneauMedecinJour;
14. import rdvmedecins.android.dao.entities.Medecin;
15. import rdvmedecins.android.dao.responses.AgendaMedecinJourResponse;
16. import rdvmedecins.android.dao.responses.RvResponse;
17.
18. @EFragment(R.layout.agenda)
19. public class AgendaFragment extends MyFragment {
20.
21.   // les éléments de l'interface visuelle
22.   @ViewById(R.id.txt_titre2_agenda)
23.   TextView txtTitre2;
24.   @ViewById(R.id.listViewAgenda)
25.   ListView lstCreneaux;
26.   @ViewById(R.id.btn_Annuler)
27.   Button btnAnnuler;
28.
29.   // données locales
30.   private AgendaMedecinJour agenda;
31.   private boolean canceled;
32.
33.   // mémoires
34.   private int firstPosition;
35.   private int top;
36.   private boolean afterViewsDone = false;
37.   private boolean rdvSupprimé;
38.   private int numCréneau;
39.
40.   // constructeur
41.   public AgendaFragment() {
42.     System.out.println("AgendaFragment constructeur");
43.   }
44.
45.   @AfterViews
46.   void afterViews() {
47.     afterViewsDone = true;
48.     if (session.getNumVue() == Constants.VUE_AGENDA) {
49.       initFragment();
50.     }
51.   }
52.
53.   @Override
54.   public void onRefresh() {
55.     System.out.println("Agenda refresh");
56.     if (afterViewsDone) {
57.       initFragment();
58.     }
59.   }
60.
61.   private void initFragment() {
62.     System.out.println("Agenda init");
63.     // on récupère l'agenda en session
64.     agenda = session.getAgenda();
65.     // on génère le titre de la page
66.     Medecin medecin = agenda.getMedecin();
67.     String text = String.format("Rendez-vous de %s %s %s le %s", medecin.getTitre(),
    medecin.getPrenom(),
68.       medecin.getNom(), session.getJourRv());
69.     txtTitre2.setText(text);
70.     // on régénère la liste des créneaux
71.     updateAgenda();
```

```
72.     // état du bouton [Annuler]
73.     btnAnnuler.setVisibility(View.INVISIBLE);
74.   }
75.
76.   // clic sur un lien [Ajouter / Supprimer]
77.   public void doValider(int position, String texte) {
78.     ...
79.   }
80.
81.   @Click(R.id.btn_Annuler)
82.   public void doAnnuler() {
83. ...
84.   }
85.
86.    // début de l'attente
87.   private void beginWaiting() {
88.     // on met le sablier
89.     showHourGlass();
90.     // état des boutons
91.     btnAnnuler.setVisibility(View.VISIBLE);
92.   }
93.
94.   // fin de l'attente
95.   protected void cancelWaiting() {
96.     // on cache le sablier
97.     hideHourGlass();
98.     // état du bouton Annuler
99.     btnAnnuler.setVisibility(View.INVISIBLE);
100.    }
101.
102.  }
```

- lignes 22-27 : les trois éléments de l'interface visuelle ;
- lignes 45-74 : les trois méthodes qui gèrent l'initialisation de l'interface visuelle soit après la méthode [afterViews] (ligne 46), soit après la méthode [onRefresh] (ligne 54) ;
- ligne 64 : on récupère l'agenda qui a été mis en session par la vue Accueil ;
- lignes 66-69 : on utilise les informations qui sont dedans pour initialiser l'élément [txtTitre2] de l'interface visuelle ;
- ligne 71 : grâce aux informations de l'agenda, on (ré)génère la liste des créneaux de l'agenda ;
- ligne 73 : on cache le bouton [Annuler] ;

La (ré)génération de la liste des créneaux de l'agenda est nécessaire à plusieurs endroits du code. Elle a été factorisée dans la méthode [updateAgenda] suivante :

```
1.    private void updateAgenda() {
2.      // (ré)génération des créneaux de l'agenda
3.      ArrayAdapter<CreneauMedecinJour> adapter = new ListCreneauxAdapter(activité,
    R.layout.creneau_medecin,agenda.getCreneauxMedecinJour(), this);
4.      lstCreneaux.setAdapter(adapter);
5.      // on se positionne au bon endroit du ListView
6.      lstCreneaux.setSelectionFromTop(firstPosition, top);
7.  }
```

- lignes 3-4 : on définit l'adaptateur du composant [ListView]. Cet adaptateur définit à la fois la source de données du [ListView] et le modèle d'affichage de chaque élément de celle-ci. Nous allons présenter cet adaptateur prochainement ;
- ligne 5 : on revient sur la position précédente de l'agenda. En effet, on ne voit qu'une partie des créneaux de la journée. Si on ajoute / supprime un rendez-vous dans le dernier créneau, le code va rafraîchir la page pour présenter le nouvel agenda. Ce rafraîchissement fait qu'on est alors positionné de nouveau sur le 1er créneau, ce qui n'est pas souhaitable. La ligne 6 remédie à ce problème. On trouvera la description de cette solution à l'URL [http://stackoverflow.com/questions/3014089/maintain-save-restore-scroll-position-when-returning-to-a-listview] ;

La classe [ListCreneauxAdapter] sert à définir une ligne du [ListView] :

On voit ci-dessus, que selon que le créneau a un rendez-vous ou non, l'affichage n'est pas le même. Le code de la classe [ListCreneauxAdapter] est le suivant :

```
1.  ...
2.
3.  public class ListCreneauxAdapter extends ArrayAdapter<CreneauMedecinJour> {
4.
5.      // le tableau des créneaux horaires
6.      private CreneauMedecinJour[] creneauxMedecinJour;
7.      // le contexte d'exécution
8.      private Context context;
9.      // l'id du layout d'affichage d'une ligne de la liste des créneaux
10.     private int layoutResourceId;
11.     // listener des clics
12.     private AgendaFragment vue;
13.
14.     // constructeur
15.     public ListCreneauxAdapter(Context context, int layoutResourceId, CreneauMedecinJour[]
    creneauxMedecinJour,
16.             AgendaFragment vue) {
17.         super(context, layoutResourceId, creneauxMedecinJour);
18.         // on mémorise les infos
19.         this.creneauxMedecinJour = creneauxMedecinJour;
20.         this.context = context;
21.         this.layoutResourceId = layoutResourceId;
22.         this.vue = vue;
23.         // on trie le tableau des créneaux dans l'ordre des horaires
24.         Arrays.sort(creneauxMedecinJour, new MyComparator());
25.     }
26.
27.     @Override
28.     public View getView(final int position, View convertView, ViewGroup parent) {
29.     ...
30. }
31.
32. // tri du tableau des créneaux
33. class MyComparator implements Comparator<CreneauMedecinJour> {
34. ...
35.     }
36. }
```

- ligne 3 : la classe [ListCreneauxAdapter] doit étendre un adaptateur prédéfini pour les [ListView], ici la classe [ArrayAdapter] qui comme son nom l'indique alimente le [ListView] avec un tableau d'objets, ici de type [CreneauMedecinJour]. Rappelons le code de cette entité :

```
1.  public class CreneauMedecinJour implements Serializable {
2.
```

```
3.      private static final long serialVersionUID = 1L;
4.      // champs
5.      private Creneau creneau;
6.      private Rv rv;
7.  ...
8.  }
```

- la classe [CreneauMedecinJour] contient un créneau horaire (ligne 5) et un éventuel rendez-vous (ligne 6) ou *null* si pas de rendez-vous ;

Retour au code de la classe [ListCreneauxAdapter] :

- ligne 15 : le constructeur reçoit quatre paramètres :
    1. l'activité Android en cours,
    2. le fichier XML définissant le contenu de chaque élément du [ListView],
    3. le tableau des créneaux horaires du médecin,
    4. la vue elle-même ;
- ligne 24 : le tableau des créneaux horaires est trié dans l'ordre croissant des horaires ;

La méthode [getView] est chargée de générer la vue correspondant à une ligne du [ListView]. Celle-ci comprend trois éléments :

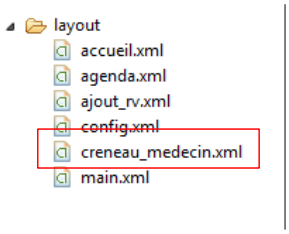
N°	Id	Type	Rôle
1	txtCreneau	TextView	créneau horaire
2	txtClient	TextView	le client
3	btnValider	TextView	lien pour ajouter / supprimer un rendez-vous

Le code de la méthode [getView] est le suivant :

```
1.  @Override
2.      public View getView(final int position, View convertView, ViewGroup parent) {
3.          // on se positionne sur le bon créneau
4.          CreneauMedecinJour creneauMedecin = creneauxMedecinJour[position];
5.          // on crée la ligne
6.          View row = ((Activity) context).getLayoutInflater().inflate(layoutResourceId, parent, false);
7.          // le créneau horaire
8.          TextView txtCreneau = (TextView) row.findViewById(R.id.txt_Creneau);
9.          txtCreneau.setText(String.format("%02d:%02d-%02d:%02d", creneauMedecin.getCreneau().getHdebut(),
    creneauMedecin
10.                 .getCreneau().getMdebut(), creneauMedecin.getCreneau().getHfin(),
    creneauMedecin.getCreneau().getMfin()));
11.         // le client
12.         TextView txtClient = (TextView) row.findViewById(R.id.txt_Client);
13.         String text;
14.         if (creneauMedecin.getRv() != null) {
15.             Client client = creneauMedecin.getRv().getClient();
16.             text = String.format("%s %s %s", client.getTitre(), client.getPrenom(), client.getNom());
17.         } else {
18.             text = "";
19.         }
20.         txtClient.setText(text);
21.         // le lien
22.         final TextView btnValider = (TextView) row.findViewById(R.id.btn_Valider);
23.         if (creneauMedecin.getRv() == null) {
24.             // ajouter
25.             btnValider.setText(R.string.btn_ajouter);
26.             btnValider.setTextColor(context.getResources().getColor(R.color.blue));
27.         } else {
28.             // supprimer
29.             btnValider.setText(R.string.btn_supprimer);
30.             btnValider.setTextColor(context.getResources().getColor(R.color.red));
31.         }
32.         // listener du lien
```

```
33.         btnValider.setOnClickListener(new OnClickListener() {
34.
35.             @Override
36.             public void onClick(View v) {
37.                 // on passe les infos à la vue de l'agenda
38.                 vue.doValider(position, btnValider.getText().toString());
39.             }
40.         });
41.         // on rend la ligne
42.         return row;
43.     }
```

- ligne 2 : **position** est le n° de ligne qu'on va générer dans le [ListView]. C'est également le n° du créneau dans le tableau [creneauxMedecinJour]. On ignore les deux autres paramètres ;
- ligne 4 : on récupère le créneau horaire à afficher dans la ligne du [ListView] ;
- ligne 6 : la ligne est construite à partir de sa définition XML



Le code de [creneau_medecin.xml] est le suivant :

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:id="@+id/RelativeLayout1"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     android:background="@color/wheat" >
7.
8.     <TextView
9.         android:id="@+id/txt_Creneau"
10.        android:layout_width="100dp"
11.        android:layout_height="wrap_content"
12.        android:layout_marginTop="20dp"
13.        android:layout_marginLeft="20dp"
14.        android:text="@string/txt_dummy" />
15.
16.    <TextView
17.        android:id="@+id/txt_Client"
18.        android:layout_width="200dp"
19.        android:layout_height="wrap_content"
20.        android:layout_alignBaseline="@+id/txt_Creneau"
21.        android:layout_marginLeft="20dp"
22.        android:layout_toRightOf="@+id/txt_Creneau"
23.        android:text="@string/txt_dummy" />
24.
25.    <TextView
26.        android:id="@+id/btn_Valider"
27.        android:layout_width="wrap_content"
28.        android:layout_height="wrap_content"
29.        android:layout_alignBaseline="@+id/txt_Client"
30.        android:layout_marginLeft="20dp"
31.        android:layout_toRightOf="@+id/txt_Client"
32.        android:text="@string/btn_valider"
33.        android:textColor="@color/blue" />
34.
35. </RelativeLayout>
```

- lignes 8-10 : le créneau horaire [1] est construit ;
- lignes 12-20 : l'identité du client [2] est construite ;
- ligne 23 : si le créneau n'a pas de rendez-vous ;
- lignes 25-26 : on construit le lien [Ajouter] de couleur bleue ;
- lignes 29-30 : sinon on construit le lien [Supprimer] de couleur rouge ;
- lignes 33-40 : quelque soit la nature lien [Ajouter / Supprimer] c'est la méthode [doValider] de la vue qui gèrera le clic sur le lien. La méthode recevra deux arguments :
    1. le n° du créneau qui a été cliqué,
    2. le libellé du lien qui a été cliqué ;
- ligne 42 : on rend la ligne qu'on vient de construire.

On notera que c'est la méthode [doValider] du fragment [AgendaFragment] qui gère les liens. Celle-ci est la suivante :

```
1.    public void doValider(int numCréneau, String texte) {
2.      // on note la position du scroll pour y revenir
3.      // lire [http://stackoverflow.com/questions/3014089/maintain-save-restore-scroll-position-when-returning-to-a-listview]
4.      // position du 1er élément visible complètement ou non
5.      firstPosition = lstCreneaux.getFirstVisiblePosition();
6.      // offset Y de cet élément par rapport au haut du ListView
7.      // mesure la hauteur de la partie éventuellement cachée
8.      View v = lstCreneaux.getChildAt(0);
9.      top = (v == null) ? 0 : v.getTop();
10.     // on note également le n° du créneau cliqué
11.     this.numCréneau = numCréneau;
12.     // selon le texte, on ne fait pas la même chose
13.     if (texte.equals(getResources().getString(R.string.btn_ajouter))) {
14.       doAjouter();
15.     } else {
16.       doSupprimer();
17.     }
18. }
```

- ligne 1 : on reçoit le n° du créneau sur lequel l'utilisateur a cliqué un lien ainsi que le libellé de ce lien ;
- lignes 2-9 : on note des informations sur la position du créneau cliqué afin de pouvoir réafficher ultérieurement celui-ci à l'endroit où il était lorsqu'il a été cliqué. On note deux informations (firstPosition, top). Ces deux informations sont utilisées dans la méthode [initFragment] :

```
      // on se positionne au bon endroit du ListView
      lstCreneaux.setSelectionFromTop(firstPosition, top);
```

  L'explication du mécanisme mis en oeuvre est assez complexe. Il faudrait faire un dessin. En ligne 3, on trouve l'URL qui donne cette explication ;
- ligne 11 : on mémorise le n° du créneau cliqué car la méthode [doAnnuler] en a besoin dans certains cas ;
- lignes 13-17 : selon le libellé du lien cliqué on ajoute (ligne 14) ou on supprime (ligne 16) un rendez-vous.

L'ajout d'un rendez-vous se fait avec la méthode [doAjouter] suivante :

```
1.    private void doAjouter() {
2.      // on met le n° du créneau cliqué dans la session
3.      session.setPosition(numCréneau);
4.      // on passe à la vue d'ajout
5.      activité.showView(Constants.VUE_AJOUT_RV);
6.  }
```

- ligne 3 : on met le n° du créneau cliqué dans la session car la vue d'ajout de rendez-vous va en avoir besoin ;
- ligne 5 : on affiche la vue d'ajout de rendez-vous ;

La suppression d'un rendez-vous se fait elle sans changement de vue :

```
1.    private void doSupprimer() {
2.      // on supprime le Rdv
3.      canceled = false;
4.      rdvSupprimé = false;
5.      deleteRvInBackground();
6.      // début de l'attente
7.      beginWaiting();
8.    }
```

```
9.
10.   @Background(id = "delete", delay = Constants.DELAY)
11.   void deleteRvInBackground() {
12.     RvResponse response;
13.     try {
14.       // on supprime le Rv
15.       long idRv = agenda.getCreneauxMedecinJour()[numCréneau].getRv().getId();
16.       response = activité.supprimerRv(idRv);
17.     } catch (Exception e) {
18.       response = new RvResponse();
19.       response.setStatus(1);
20.       response.setMessages(getMessagesFromException(e));
21.     }
22.     // on exploite la réponse
23.     manageRvResponse(response);
24. }
```

- lignes 3-5 : la suppression se fait en tâche de fond ;
- ligne 15 : on récupère l'identifiant du rendez-vous à supprimer ;
- ligne 16 : on demande à l'activité de supprimer le rendez-vous ;
- lignes 17-21 : si cette demande provoque une exception, on construit un objet [RvResponse] avec les messages d'erreur de l'exception ;
- ligne 23 : la réponse est affichée ;

La méthode [manageRvResponse] est la suivante :

```
1.  @UiThread
2.    void manageRvResponse(RvResponse response) {
3.      if (!canceled) {
4.        // on gère le cas d'erreur
5.        if (response.getStatus() != 0) {
6.          // on affiche l'erreur
7.          showMessages(response.getMessages());
8.        } else {
9.          // on note que le rdv a été supprimé
10.         rdvSupprimé = true;
11.       }
12.       // dans tous les cas, on demande l'agenda le plus récent
13.       getAgendaMedecinInBackground();
14.     }
15.   }
```

- ligne 3 : si l'utilisateur a annulé l'opération, rien n'est affiché ;
- lignes 5-8 : s'il y a eu erreur, on affiche le message d'erreur et on s'arrête là ;
- ligne 10 : on note que le rendez-vous a été supprimé. La méthode [doAnnuler] utilise cette information ;
- ligne 13 : erreur ou pas, on demande le nouvel agenda du médecin. C'est un choix. Si l'erreur est dûe à un problème réseau, on aura une seconde erreur sur la nouvelle demande au serveur. Mais si l'erreur est dûe au fait que le rendez-vous à supprimer a été supprimé au même moment (mais quand même avant) par quelqu'un d'autre, le nouvel agenda demandé va le montrer et ça c'est intéressant pour l'utilisateur. De façon générale, l'intérêt de redemander au serveur l'agenda du médecin est que le système étant multi-utilisateurs, on a intérêt à avoir la version la plus récente de l'agenda pour voir les ajouts / suppressions des autres utilisateurs ;

La méthode [getAgendaMedecinInBackground] est analogue à sa version dans [AccueilFragment]. On aurait pu la factoriser dans la classe mère [MyFragment] :

```
1.    @Background(id = "agenda", delay = Constants.DELAY)
2.    void getAgendaMedecinInBackground() {
3.      AgendaMedecinJourResponse response;
4.      try {
5.        // on demande le nouvel agenda
6.        response = activité.getAgendaMedecinJour(agenda.getMedecin().getId(), session.getDayRv());
7.      } catch (Exception e) {
8.        response = new AgendaMedecinJourResponse();
9.        response.setStatus(1);
10.       response.setMessages(getMessagesFromException(e));
11.     }
12.     // gestion de la réponse
13.     manageAgendaMedecinJourResponse(response);
14.   }
15.
```

```
16.   @UiThread
17.   public void manageAgendaMedecinJourResponse(AgendaMedecinJourResponse response) {
18.     if (!canceled) {
19.       // fin de l'attente
20.       cancelWaiting();
21.       // on gère le cas d'erreur
22.       if (response.getStatus() != 0) {
23.         // on affiche l'erreur
24.         showMessages(response.getMessages());
25.         // retour à l'UI
26.         return;
27.       }
28.       // on met le nouvel agenda dans la session
29.       agenda = response.getAgenda();
30.       session.setAgenda(agenda);
31.       // on régénère la liste des créneaux
32.       updateAgenda();
33.     }
34. }
```

- lignes 29-30 : on notera que le nouvel agenda est mis en session afin d'être disponible aux autres fragments ;

Enfin, la méthode d'annulation est la suivante :

```
1.    @Click(R.id.btn_Annuler)
2.    public void doAnnuler() {
3.      // on annule les tâches asynchrones
4.      BackgroundExecutor.cancelAll("delete", true);
5.      BackgroundExecutor.cancelAll("agenda", true);
6.      // si le rdv a été supprimé il faut mettre à jour l'agenda local
7.      if (rdvSupprimé) {
8.        // on supprime le rendez-vous dans l'agenda local
9.        agenda.getCreneauxMedecinJour()[numCréneau].setRv(null);
10.       // on met à jour l'interface visuelle
11.      updateAgenda();
12.     }
13.     // on annule l'attente
14.     cancelWaiting();
15.     // on note l'annulation
16.     canceled = true;
17. }
```

- lignes 4-5 : les deux tâches asynchrones sont annulées. A noter que l'annulation d'une tâche qui n'a pas encore été lancée ne provoque pas d'erreur ;
- lignes 7-12 : ces lignes traitent le cas où l'annulation a lieu après que la suppression du rendez-vous ait été réussie. L'annulation concerne alors la demande du nouvel agenda au serveur. Afin que l'utilisateur n'ait pas devant lui un agenda erroné, on met à jour l'agenda local et on régénère l'interface visuelle avec ;

2.6.12 La vue d'ajout d'un rendez-vous

La vue Ajout permet à l'utilisateur d'ajouter un rendez-vous à l'agenda :

Les éléments de l'interface visuelle sont les suivants :

n°	Type	Nom
1	TextView	txtTitre2
2	Spinner	spinnerClients
3	Button	btnValider
4	Button	btnAnnuler

La vue Ajout est gérée par le fragment [AjoutRvFragment] suivant :

La classe [AjoutRvFragment] est la suivante :

```
1.  package rdvmedecins.android.fragments;
2.
3.  import android.view.View;
4.  import android.widget.*;
5.  import org.androidannotations.annotations.*;
6.  import org.androidannotations.api.BackgroundExecutor;
7.  import rdvmedecins.android.R;
8.  import rdvmedecins.android.activity.Constants;
9.  import rdvmedecins.android.dao.entities.*;
10. import rdvmedecins.android.dao.responses.AgendaMedecinJourResponse;
11. import rdvmedecins.android.dao.responses.RvResponse;
12.
13. import java.util.List;
```

```
14. import java.util.Locale;
15.
16. @EFragment(R.layout.ajout_rv)
17. public class AjoutRvFragment extends MyFragment {
18.
19.   // les éléments de l'interface visuelle
20.   @ViewById(R.id.btn_Valider)
21.   Button btnValider;
22.   @ViewById(R.id.btn_Annuler)
23.   Button btnAnnuler;
24.   @ViewById(R.id.spinnerClients)
25.   Spinner spinnerClients;
26.   @ViewById(R.id.txt_titre2_ajoutRv)
27.   TextView txtTitre2;
28.
29.   // les clients
30.   private List<Client> clients;
31.
32.   // données locales
33.   private Creneau creneau;
34.   private boolean afterViewsDone;
35.   private boolean canceled;
36.   private boolean rdvAjouté;
37.   private Rv rv;
38.
39.   // constructeur
40.   public AjoutRvFragment() {
41.     System.out.println("AjoutRvFragment constructeur");
42.   }
43.
44.   @AfterViews
45.   void afterViews() {
46.     afterViewsDone = true;
47.     System.out.println("Ajout afterViews");
48.     if (session.getNumVue() == Constants.VUE_AJOUT_RV) {
49.       initFragment();
50.     }
51.   }
52.
53.
54.   @Override
55.   public void onRefresh() {
56.     System.out.println("Ajout refresh");
57.     if (afterViewsDone) {
58.       initFragment();
59.     }
60.   }
61.
62.   private void initFragment() {
63.     System.out.println("Ajout init");
64.     // état des boutons
65.     btnValider.setVisibility(View.VISIBLE);
66.     btnAnnuler.setVisibility(View.INVISIBLE);
67.     // on gère le spinner des clients
68.     SpinnerAdapter spinnerAdapter = spinnerClients.getAdapter();
69.     if (spinnerAdapter == null) {
70.       // on récupère les clients en session
71.       clients = session.getClients();
72.       // on construit le tableau affiché par le spinner
73.       String[] arrayClients = new String[clients.size()];
74.       int i = 0;
75.       for (Client client : clients) {
76.         arrayClients[i] = String.format("%s %s %s", client.getTitre(), client.getPrenom(),
    client.getNom());
77.         i++;
78.       }
79.       // on associe les clients au spinner
80.       ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(activité,
    android.R.layout.simple_spinner_item,
81.         arrayClients);
82.       dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
83.       spinnerClients.setAdapter(dataAdapter);
84.     }
85.     // on construit le titre 2 de la page
86.     // on récupère le n° du créneau à réserver en session
```

```
87.     int position = session.getPosition();
88.     // on récupère l'agenda du médecin dans la session
89.     AgendaMedecinJour agenda = session.getAgenda();
90.     Medecin medecin = agenda.getMedecin();
91.     creneau = agenda.getCreneauxMedecinJour()[position].getCreneau();
92.     String jour = session.getJourRv();
93.     String titre2 = String.format(Locale.FRANCE,
94.       "Prise de rendez-vous de %s %s %s le %s pour le créneau %02d:%02d-%02d:%02d", medecin.getTitre(),
95.       medecin.getPrenom(), medecin.getNom(), jour, creneau.getHdebut(), creneau.getMdebut(),
    creneau.getHfin(),
96.       creneau.getMfin());
97.     txtTitre2.setText(titre2);
98.   }
99.
100.  // validation de la page
101.  @Click(R.id.btn_Valider)
102.  protected void doValider() {
103.   ...
104.  }
105.
106.  // annulation
107.  @Click(R.id.btn_Annuler)
108.  protected void doAnnuler() {
109....
110.  }
111.
112.// début de l'attente
113.  private void beginWaiting() {
114.    // on met le sablier
115.    showHourGlass();
116.    // état des boutons
117.    btnValider.setVisibility(View.INVISIBLE);
118.    btnAnnuler.setVisibility(View.VISIBLE);
119.
120.  }
121.
122.  // fin de l'attente
123.  protected void cancelWaiting() {
124.    // on cache le sablier
125.    hideHourGlass();
126.    // état des boutons
127.    btnValider.setVisibility(View.VISIBLE);
128.    btnAnnuler.setVisibility(View.INVISIBLE);
129.  }
130.
131.}
```

- lignes 20-27 : les éléments de l'interface visuelle ;
- lignes 44-98 : les trois méthodes d'initialisation de l'interface visuelle ;
- ligne 62 : la méthode qui factorise cette initialisation ;
- lignes 68-78 : initialisation de la liste déroulante des clients ;
- ligne 69 : on regarde d'abord si cette liste n'a pas été déjà été initialisée auquel cas il est inutile de le refaire. Lorsque la liste a été initialisée elle a un adaptateur non *null* ;
- lignes 70-78 : on crée le tableau de [String] qui va alimenter la liste déroulante ;
- lignes 80-83 : ce tableau est associé à la liste déroulante via un adaptateur ;
- lignes 85-97 : on construit le message 'Prise de rendez-vous de Mme Marie PELISSIER pour le créneau 08:00-08:20' affiché en haut de la vue ;
- ligne 71 : la liste des clients est enregistrée localement dans le champ de la ligne 30 ;
- lignes 91 : le créneau du rendez-vous est enregistré localement dans le champ de la ligne 33 ;

La méthode de validation du rendez-vous est la suivante :

```
1.    @Click(R.id.btn_Valider)
2.    protected void doValider() {
3.      // on récupère le client choisi
4.      Client client = clients.get(spinnerClients.getSelectedItemPosition());
5.      // on ajoute le RV
6.      canceled = false;
7.      rdvAjouté=false;
8.      addRvInBackground(client);
9.      // début de l'attente
10.     beginWaiting();
```

```
11.   }
12.
13.   @Background(id = "addRv", delay = Constants.DELAY)
14.   void addRvInBackground(Client client) {
15.     RvResponse response = null;
16.     try {
17.       // on ajoute le Rv
18.       response = activité.ajouterRv(session.getDayRv(), creneau.getId(), client.getId());
19.     } catch (Exception e) {
20.       response = new RvResponse();
21.       response.setStatus(1);
22.       response.setMessages(getMessagesFromException(e));
23.     }
24.     // on exploite la réponse
25.     manageRvResponse(response);
26.   }
27.
28.   @UiThread
29.   void manageRvResponse(RvResponse response) {
30.     if (!canceled) {
31.       // on gère le cas d'erreur
32.       if (response.getStatus() != 0) {
33.         // on affiche l'erreur
34.         showMessages(response.getMessages());
35.       } else {
36.         // on note que le rdv a été ajouté
37.         rdvAjouté = true;
38.         // on mémorise le rdv
39.         rv = response.getRv();
40.       }
41.     // on demande le nouvel agenda
42.     getAgendaMedecinInBackground();
43.   }
44. }
```

- ligne 4 : le client sélectionné dans la liste est récupéré ;
- ligne 8 : l'ajout du rendez-vous est fait en tâche de fond ;
- ligne 18 : c'est l'activité qui fait cet ajout ;
- lignes 20-22 : si cet ajout provoque une exception, on crée un objet [RvResponse] mémorisant les messages d'erreur de l'exception ;
- ligne 25 : on demande l'affichage de la réponse ;
- ligne 30 : si l'opération d'ajout a été annulée, on n'affiche rien ;
- lignes 32-35 : s'il y a eu erreur, on l'affiche ;
- lignes 36-39 : on note que le rendez-vous a bien été ajouté et on enregistre localement les caractéristiques de celui-ci ;
- ligne 42 : le nouvel agenda est demandé. On va obtenir l'agenda le plus récent avec le rendez-vous ajouté ainsi que les modifications faites par les autres utilisateurs de l'agenda. S'il y a eu erreur lors de l'ajout de l'agenda, on demande quand même le nouvel agenda car cette erreur peut être dûe au fait qu'au même moment (mais un peu avant) un autre utilisateur a réservé ce même créneau ;

L'obtention du nouvel agenda est obtenu avec les méthodes suivantes :

```
1.  @Background(id = "agenda", delay = Constants.DELAY)
2.    void getAgendaMedecinInBackground() {
3.      AgendaMedecinJourResponse response;
4.      try {
5.        // on demande l'agenda
6.        AgendaMedecinJour agenda = session.getAgenda();
7.        response = activité.getAgendaMedecinJour(agenda.getMedecin().getId(), session.getDayRv());
8.      } catch (Exception e) {
9.        response = new AgendaMedecinJourResponse();
10.       response.setStatus(1);
11.       response.setMessages(getMessagesFromException(e));
12.     }
13.     // gestion de la réponse
14.     manageAgendaMedecinJourResponse(response);
15.   }
16.
17.   @UiThread
18.   public void manageAgendaMedecinJourResponse(AgendaMedecinJourResponse response) {
19.     if (!canceled) {
20.       // fin de l'attente
```

```
21.       cancelWaiting();
22.       // on gère le cas d'erreur
23.       if (response.getStatus() != 0) {
24.         // on affiche l'erreur
25.         showMessages(response.getMessages());
26.       } else {
27.         // on met l'agenda dans la session
28.         session.setAgenda(response.getAgenda());
29.         // on l'affiche
30.         activité.showView(Constants.VUE_AGENDA);
31.       }
32.     }
33.   }
```

- ligne 28 : le nouvel agenda obtenu est mis en session ;
- ligne 30 : et la vue chargée de l'afficher est rendue visible ;

Enfin, la méthode d'annulation des tâches de fond est la suivante :

```
1.    @Click(R.id.btn_Annuler)
2.    protected void doAnnuler() {
3.      // on annule les deux tâches
4.      BackgroundExecutor.cancelAll("addRv", true);
5.      BackgroundExecutor.cancelAll("agenda", true);
6.      // et l'attente
7.      cancelWaiting();
8.      // on note l'annulation
9.      canceled = true;
10.     // rdv déjà ajouté ?
11.     if (rdvAjouté) {
12.       // on modifie l'agenda local
13.       AgendaMedecinJour agenda = session.getAgenda();
14.       agenda.getCreneauxMedecinJour()[session.getPosition()].setRv(rv);
15.       // on affiche l'agenda
16.       activité.showView(Constants.VUE_AGENDA);
17.     }
18. }
```

- lignes 4-5 : les deux tâches de fond sont annulées ;
- lignes 11-17 : si l'annulation est faite après que l'ajout du rendez-vous ait été réussi, on n'a pas le nouvel agenda mais on a encore l'agenda local qui est en session. On met alors à jour celui-ci avec le nouveau rendez-vous puis on le fait afficher. Si l'annulation est faite avant que l'ajout du rendez-vous ait été réussi, on reste sur la même vue ;

2.7 Conclusion

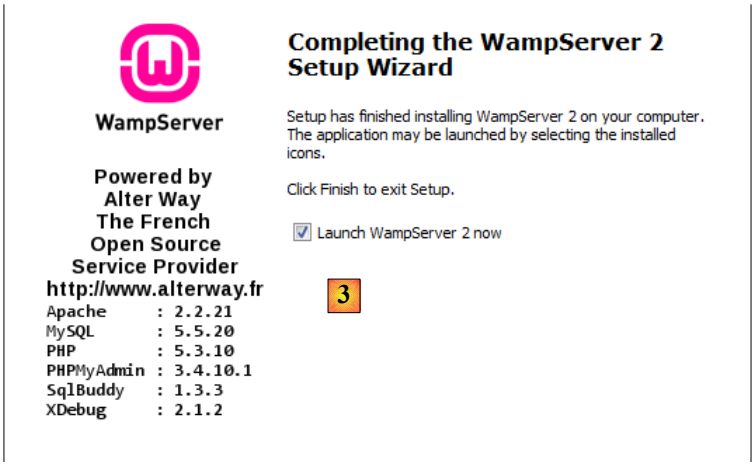
Les pages qui précèdent donnent une méthodologie solide et reproductibe pour créer des applications Android pour tablettes. Nous proposons maintenant au lecteur deux TP afin de mettre en oeuvre les connaissances acquises précédemment :

- TP1 : gestion basique d'une fiche de paie, page 266 ;
- TP2 : commande de cartes Arduino, page 280 ;

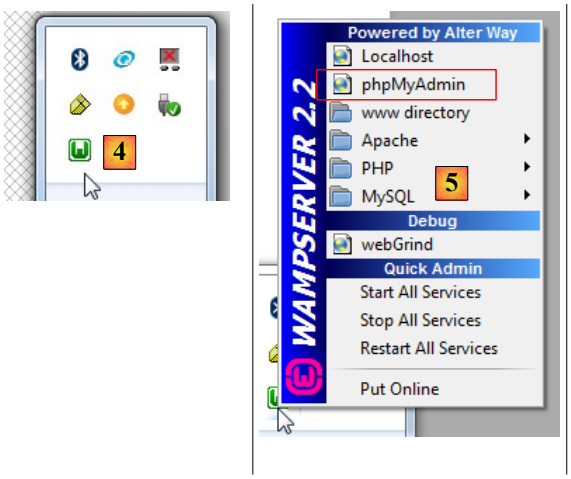
2.8 Annexes

2.8.1 Installation de [WampServer]

[WampServer] est un ensemble de logiciels pour développer en PHP / MySQL / Apache sur une machine Windows. Nous l'utiliserons uniquement pour le SGBD MySQL.



- sur le site de [WampServer] [1], choisir la version qui convient [2],
- l'exécutable téléchargé est un installateur. Diverses informations sont demandées au cours de l'installation. Elles ne concernent pas MySQL. On peut donc les ignorer. La fenêtre [3] s'affiche à la fin de l'installation. On lance [WampServer],



- en [4], l'icône de [WampServer] s'installe dans la barre des tâches en bas et à droite de l'écran [4],
- lorsqu'on clique dessus, le menu [5] s'affiche. Il permet de gérer le serveur Apache et le SGBD MySQL. Pour gérer celui-ci, on utiliser l'option [PhpPmyAdmin],
- on obtient alors la fenêtre ci-dessous,

Nous donnerons peu de détails sur l'utilisation de [PhpMyAdmin]. Nous montrerons dans le document comment l'utiliser.

Introduction à la programmation

de tablettes Android par l'exemple

-

TP1 - Gestion basique d'une fiche de paie

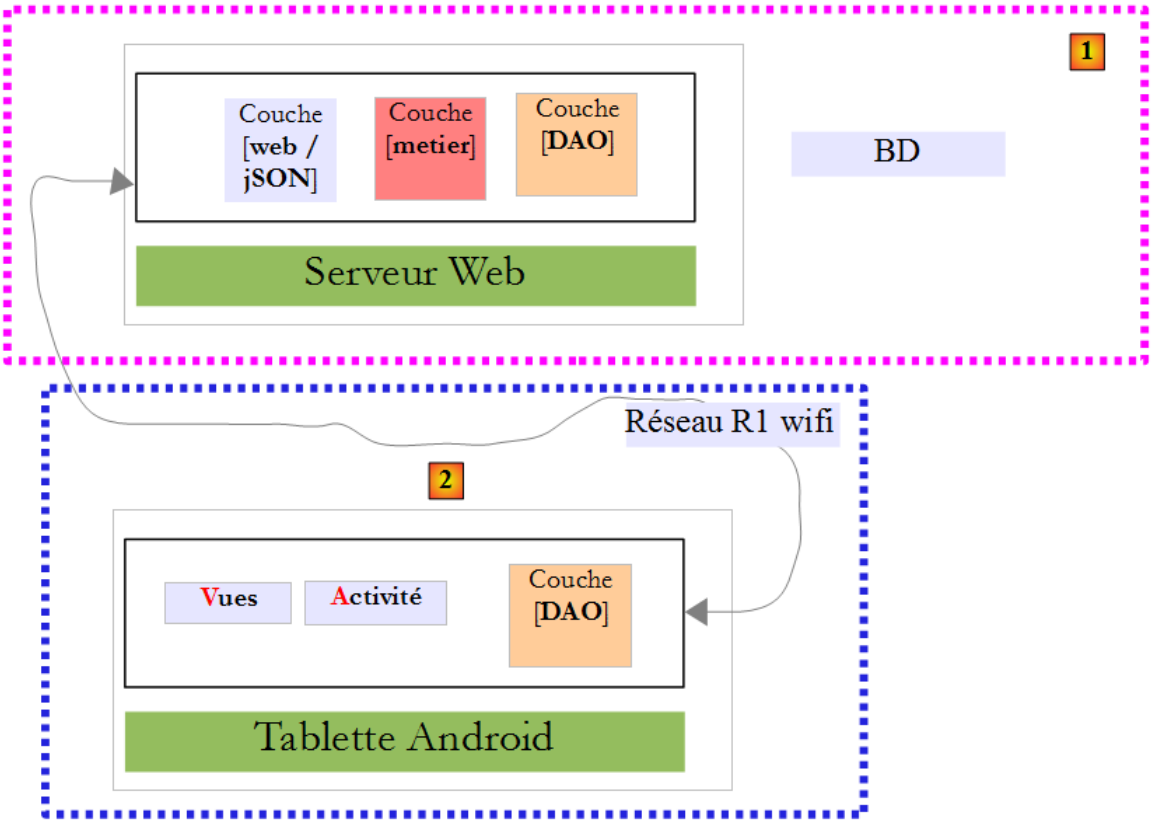
Serge Tahé, IstiA - université d'Angers
novembre 2014

3 TP 1 : Gestion basique d'une fiche de paie

3.1 Introduction

Pour appliquer ce qui a été vu précédemment, nous proposons maintenant un travail consistant à écrire un client Android pour tablette, permettant de simuler des calculs de feuille de salaire des employés d'une association.

L'application aura une architecture client / serveur :



- le serveur [1] est founi ;
- il faut construire le client Android [2].

3.2 La base de données

3.2.1 Définition

Les données statiques utiles pour construire la fiche de paie seront placées dans une base de données que nous désignerons par la suite **dbpam**. Cette base de données a les tables suivantes :

Table **EMPLOYES** : rassemble des informations sur les différentes assistantes maternelles

Structure :

`ID`	clé primaire
`VERSION`	n° de version – augmente à chaque modification de la ligne
`SS`	numéro de sécurité sociale de l'employé - unique
`NOM`	nom de l'employé
`PRENOM`	son prénom

ADRESSE	son adresse
VILLE	sa ville
CODEPOSTAL	son code postal
INDEMNITE_ID	clé étrangère sur le champ [ID] de la table [INDEMNITES]

Son contenu pourrait être le suivant :

ID	PRENOM	SS	ADRESSE	CP	VILLE	NOM	VERSION	INDEMNITE_ID
5	Marie	254104940426058	5 rue des oiseaux	49203	St Corentin	Jouveinal	1	8
6	Justine	260124402111742	La Brûlerie	49014	St Marcel	Laverti	1	7

Table **COTISATIONS** : rassemble des pourcentages nécessaires au calcul des cotisations sociales

Structure :

ID	clé primaire
VERSION	n° de version – augmente à chaque modification de la ligne
CSGRDS	pourcentage : contribution sociale généralisée + contribution au remboursement de la dette sociale
CSGD	pourcentage : contribution sociale généralisée déductible
SECU	pourcentage : sécurité sociale, veuvage, vieillesse
RETRAITE	pourcentage : retraite complémentaire + assurance chômage

Son contenu pourrait être le suivant :

ID	SECU	RETRAITE	CSGD	CSGRDS	VERSION
3	9.39	7.88	6.15	3.49	1

Les taux des cotisations sociales sont indépendants du salarié. La table précédente n'a qu'une ligne.

Table **INDEMNITES** : rassemble les éléments permettant le calcul du salaire à payer.

ID	clé primaire
VERSION	n° de version – augmente à chaque modification de la ligne
INDICE	indice de traitement - unique
BASEHEURE	prix net en euro d'une heure de garde
ENTRETIENJOUR	indemnité d'entretien en euro par jour de garde
REPASJOUR	indemnité de repas en euro par jour de garde
INDEMNITESCP	indemnité de congés payés. C'est un pourcentage à appliquer au salaire de base.

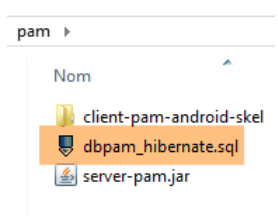
Son contenu pourrait être le suivant :

ID	ENTRETIEN_JOUR	REPAS_JOUR	INDICE	INDEMNITES_CP	BASE_HEURE	VERSION
7	2	3	1	12	1.93	1
8	2.1	3.1	2	15	2.1	1

On notera que les indemnités peuvent varier d'une assistante maternelle à une autre. Elles sont en effet associées à une assistante maternelle précise via l'indice de traitement de celle-ci. Ainsi Mme Marie Jouveinal qui a un indice de traitement de 2 (table EMPLOYES) a un salaire horaire de 2,1 euro (table INDEMNITES).

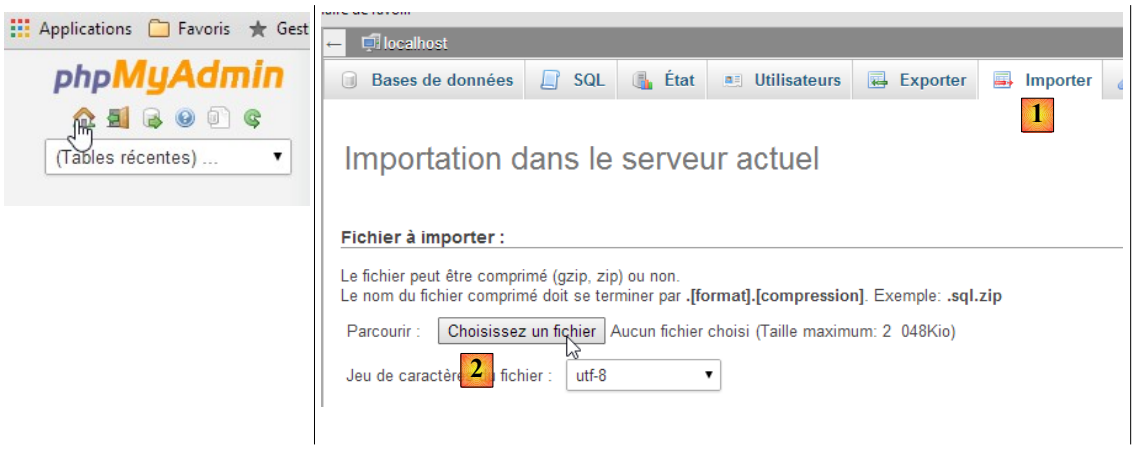
3.2.2 Génération

Le script [dbpam_hibernate.sql] de génération de la base de données est fourni :



Créez la base de données [dbpam_hibernate] (c'est le nom de la BD que le serveur web / jSON exploite) et faites en sorte que le login *root* sans mot de passe puisse y accéder. Vous pouvez procéder ainsi :

Lancez MySQL puis [PhpMyAdmin] :



- [1-2] : importez le script [dbpam_hibernate.sql] puis exécutez-le ;

3.2.3 Modélisation Java de la base

Les éléments des tables [EMPLOYES], [INDEMNITES] et [COTISATIONS] sont modélisés par les classes suivantes :

[Employe]

```
1.  package pam.entities;
2.
3.  import java.io.Serializable;
4.
5.  public class Employe implements Serializable {
6.
7.    private static final long serialVersionUID = 1L;
8.    private Long id;
9.    private int version;
10.   private String SS;
11.   private String nom;
12.   private String prenom;
13.   private String adresse;
14.   private String ville;
15.   private String codePostal;
16.   private int idIndemnite;
17.   private Indemnite indemnite;
18.
19.   public Employe() {
20.   }
21.
22.   public Employe(String SS, String nom, String prenom, String adresse, String ville, String codePostal,
    Indemnite indemnite) {
23.     ...
24.   }
25.   // getters et setters
26. ....
```

```
27. }
```

- lignes 8-15 : ces champs correspondent aux colonnes de la table [EMPLOYES] ;
- ligne 16 : le champ [indemniteId] correspond à la colonne [INDEMNITE_ID] qui est la clé étrangère de la table [EMPLOYES] ;
- ligne 17 : l'indemnité de l'employé. Ce champ n'est pas toujours renseigné :
  - il ne l'est pas lorsqu'on demande l'URL [/employes],
  - il l'est lorsqu'on demande l'URL [/salaire] ;

[**Indemnite**]

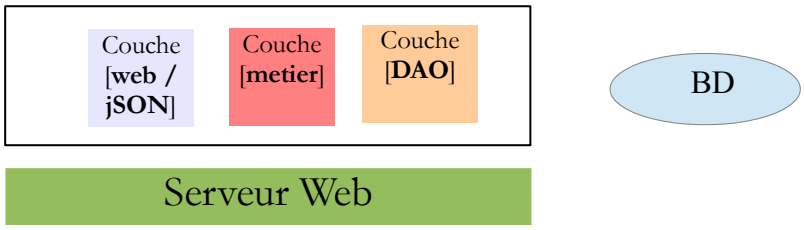
```
1.  package pam.entities;
2.
3.  import java.io.Serializable;
4.
5.  public class Indemnite implements Serializable {
6.
7.      private static final long serialVersionUID = 1L;
8.      private Long id;
9.      private int version;
10.     private int indice;
11.     private double baseHeure;
12.     private double entretienJour;
13.     private double repasJour;
14.     private double indemnitesCp;
15.
16.     public Indemnite() {
17.     }
18.
19.     public Indemnite(int indice, double baseHeure, double entretienJour, double repasJour, double
    indemnitesCP) {
20.         ...
21.     }
22.
23.     // getters et setters
24.     ....
25. }
```

- lignes 8-14 : les champs correspondent aux colonnes de la table [INDEMNITES] ;

[**Cotisation**]

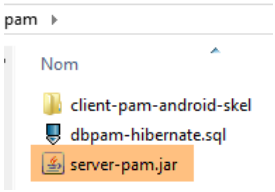
```
1.  package pam.entities;
2.
3.  import java.io.Serializable;
4.
5.  public class Cotisation implements Serializable {
6.
7.      private static final long serialVersionUID = 1L;
8.      private Long id;
9.      private int version;
10.     private double csgrds;
11.     private double csgd;
12.     private double secu;
13.     private double retraite;
14.
15.     public Cotisation() {
16.     }
17.
18.     public Cotisation(double csgrds, double csgd, double secu, double retraite) {
19.         ...
20.     }
21.     // getters et setters
22.     ...
23. }
```

3.3 Installation du serveur web / jSON



3.3.1 Installation

Le binaire Java du serveur web / jSON est fourni :



Pour lancer le serveur web / jSON, procédez de la façon suivante :

- lancez le SBD MySQL ;
- assurez-vous que la BD [dbpam_hibernate] existe ;
- ouvrez une fenêtre DOS ;
- placez vous dans le dossier du jar ;
- tapez la commande :

```
java -jar server-pam.jar
```

Cela suppose que le binaire [java.exe] est dans le PATH de votre machine. Si ce n'est pas le cas, tapez le chemin complet de [java.exe], par exemple :

```
D:\Programs\devjava\java\jdk1.8\bin\java -jar server-pam.jar
```

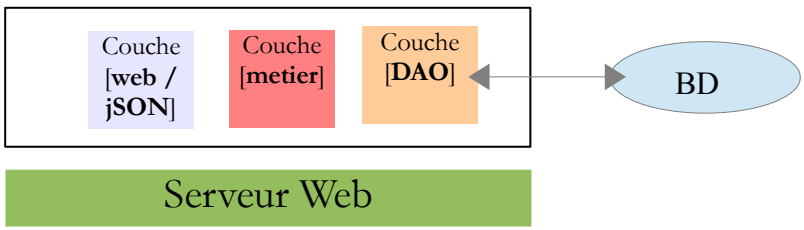
Une fenêtre DOS va s'ouvrir et afficher des logs :

```
1.  .   ____          _            __ _ _
2.   /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
3.  ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
4.   \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
5.    '  |____| .__|_| |_|_| |_\__, | / / / /
6.   =========|_|==============|___/=/_/_/_/
7.   :: Spring Boot ::        (v1.1.1.RELEASE)
8.
9.  2014-10-22 16:45:23.347  INFO 1868 --- [           main] pam.boot.BootWeb                         :
    Starting BootWeb on Gportpers3 with PID 1868 (D:\Temp\14-10-22\pam\server-pam.jar started by ST in
    D:\Temp\14-10-22\pam)
10. 2014-10-22 16:45:23.414  INFO 1868 --- [           main] ationConfigEmbeddedWebApplicationContext :
    Refreshing
    org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@689ab9e2:
    startup date [Wed Oct 22 16:45:23 CEST 2014]; root of context hierarchy
11. ...
12. ...
13. 2014-10-22 16:45:31.147  INFO 1868 --- [           main] org.hibernate.dialect.Dialect            :
    HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
14. 2014-10-22 16:45:31.484  INFO 1868 --- [           main] o.h.h.i.ast.ASTQueryTranslatorFactory    :
    HHH000397: Using ASTQueryTranslatorFactory
15. 2014-10-22 16:45:33.564  INFO 1868 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  :
    Mapped URL path [/**/favicon.ico] onto handler of type [class
    org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
```

```
16. 2014-10-22 16:45:33.804  INFO 1868 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/salaire/{SS}/{ht}/
    {jt}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    pam.restapi.FeuilleSalaireResponse
    pam.restapi.PamController.getFeuilleSalaire(java.lang.String,double,int)
17. 2014-10-22 16:45:33.805  INFO 1868 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/employes],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    pam.restapi.EmployesResponse pam.restapi.PamController.getEmployes()
18. 2014-10-22 16:45:33.807  INFO 1868 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/error],methods=[],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>>
    org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequ
    est)
19. 2014-10-22 16:45:33.807  INFO 1868 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/error],methods=[],params=[],headers=[],consumes=[],produces=[text/html],custom=[]}" onto
    public org.springframework.web.servlet.ModelAndView
    org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServlet
    Request)
20. 2014-10-22 16:45:33.839  INFO 1868 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  :
    Mapped URL path [/webjars/**] onto handler of type [class
    org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
21. 2014-10-22 16:45:33.839  INFO 1868 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  :
    Mapped URL path [/**] onto handler of type [class
    org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
22. 2014-10-22 16:45:34.384  INFO 1868 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        :
    Registering beans for JMX exposure on startup
23. 2014-10-22 16:45:34.535  INFO 1868 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer :
    Tomcat started on port(s): 8080/http
24. 2014-10-22 16:45:34.538  INFO 1868 --- [           main] pam.boot.BootWeb                         :
    Started BootWeb in 11.916 seconds (JVM running for 12.725)
25. 2014-10-22 16:45:39.329  INFO 1868 --- [       Thread-2] ationConfigEmbeddedWebApplicationContext :
    Closing
    org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@689ab9e2:
    startup date [Wed Oct 22 16:45:23 CEST 2014]; root of context hierarchy
26. 2014-10-22 16:45:39.331  INFO 1868 --- [       Thread-2] o.s.j.e.a.AnnotationMBeanExporter        :
    Unregistering JMX-exposed beans on shutdown
27. 2014-10-22 16:45:39.333  INFO 1868 --- [       Thread-2] j.LocalContainerEntityManagerFactoryBean :
    Closing JPA EntityManagerFactory for persistence unit 'default'
```

- ligne 16 : l'URL [/salaire/{SS}/{ht}/{jt}] est découverte ;
- ligne 17 : l'URL [/employes] est découverte ;

3.3.2 Les URL du service web/jSON



Le service web / jSON est implémenté par Spring MVC et expose deux URL :

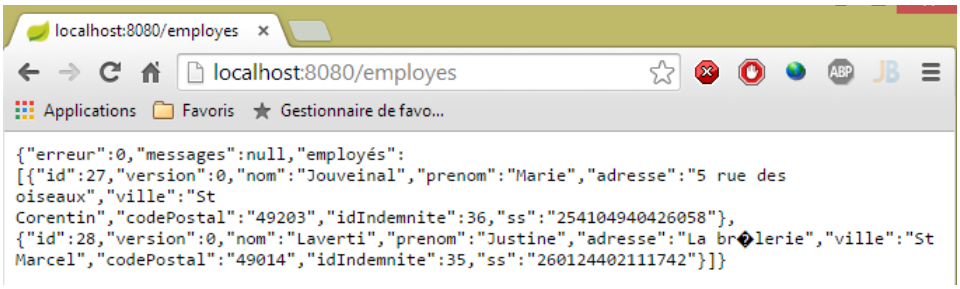
```
1.    @RequestMapping(value = "/employes", method = RequestMethod.GET)
2.  public EmployesResponse getEmployes() {
3.  ...
4.    @RequestMapping(value = "/salaire/{SS}/{ht}/{jt}", method = RequestMethod.GET)
5.  public FeuilleSalaireResponse getFeuilleSalaire(@PathVariable("SS") String SS, @PathVariable("ht") double
    ht, @PathVariable("jt") int jt) {
```

Le service web accepte les deux URL suivantes :

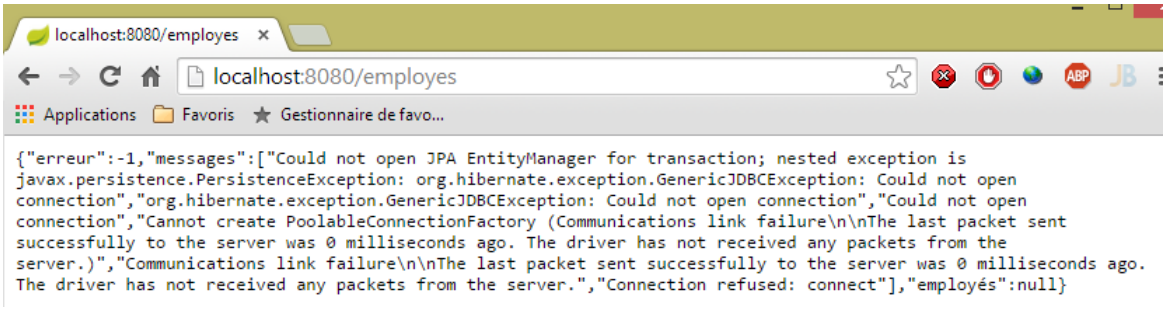
- **/employes** : pour avoir la liste des employés ;
- **/salaire/SS/ht/jt** : pour avoir la feuille de salaire de l'employé de n° [SS] ayant travaillé [ht] heures pendant [jt] jours ;

Voici des copies d'écran montrant cela.

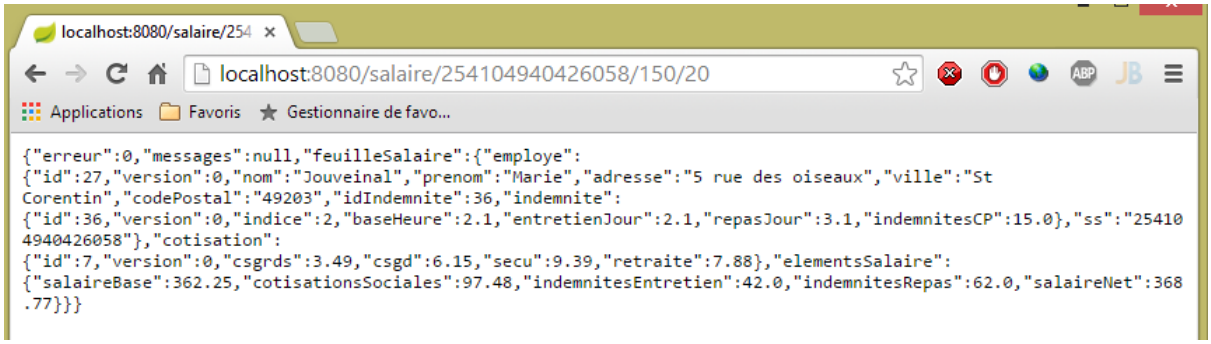
On demande les employés :



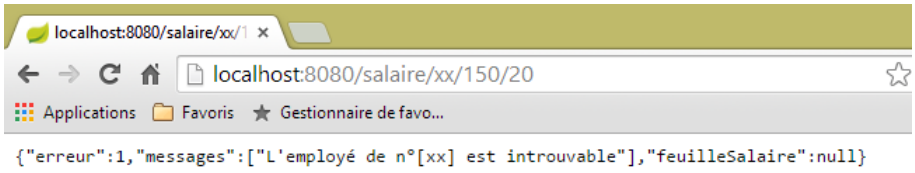
On coupe la base, on relance le serveur et on demande les employés :



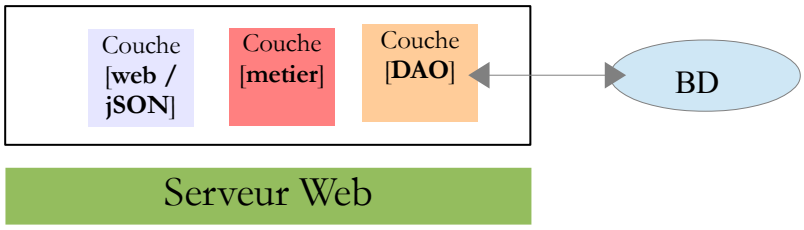
On demande un salaire :

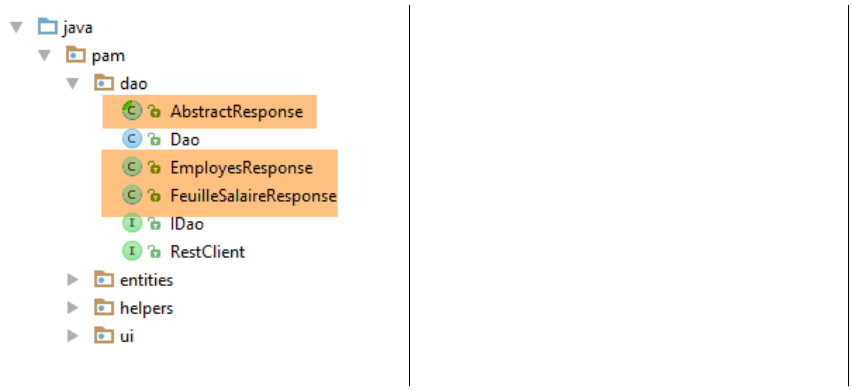


On demande le salaire d'une personne inexistante :



3.3.3 Les réponses jSON du service web/jSON





Il y a deux types de réponse jSON selon l'URL demandée. Lorsque l'URL [**/employes**] est demandée, la réponse a trois champs :

```
1.  int erreur ; // un code d'erreur - 0 si pas d'erreur
2.  List<String> messages ; // une liste de messages d'erreur - null si pas d'erreur
3.  List<Employe> employes ; // la liste des employés - null si erreur
```

Lorsque l'URL [**/salaire**] est demandée, la réponse a également trois champs :

```
1.  int erreur ; // un code d'erreur - 0 si pas d'erreur
2.  List<String> messages ; // une liste de messages d'erreur - null si pas d'erreur
3.  FeuilleSalaire feuilleSalaire ; // la feuille de salaire - null si erreur
```

On factorise les champs [erreur, messages] dans la classe abstraite [AbstractResponse] :

```
1.  package pam.dao;
2.
3.  import java.util.List;
4.
5.  public abstract class AbstractResponse {
6.
7.    // erreur
8.    private int erreur;
9.    // message d'erreur
10.   private List<String> messages;
11.
12.   // getters et setters
13. ...
14. }
```

La classe [EmployesResponse] est la réponse de l'URL [**/employes**] :

```
1.  package pam.dao;
2.
3.  import pam.entities.Employe;
4.
5.  import java.util.List;
6.
7.  public class EmployesResponse extends AbstractResponse {
8.
```

```
9.    // la liste des employés
10.   private List<Employe> employés;
11.
12.   // getters et setters
13. ...
14. }
```

- ligne 10 : la classe [Employe] a été présentée page 268 ;

La classe [SalaireResponse] est la réponse de l'URL [**/salaire**] :

```
1.  package pam.dao;
2.
3.  import pam.metier.FeuilleSalaire;
4.
5.  public class FeuilleSalaireResponse extends AbstractResponse {
6.
7.    // la feuille de salaire
8.    private FeuilleSalaire feuilleSalaire;
9.
10.   // getters et setters
11. ...
12. }
```

La classe [FeuilleSalaire] est la suivante :

```
1.  package pam.entities;
2.
3.  import java.io.Serializable;
4.
5.  public class FeuilleSalaire implements Serializable {
6.
7.     private static final long serialVersionUID = 1L;
8.     // champs privés
9.     private Employe employe;
10.    private Cotisation cotisation;
11.    private ElementsSalaire elementsSalaire;
12.
13.    // constructeurs
14.    public FeuilleSalaire() {
15.    }
16.
17.    public FeuilleSalaire(Employe employe, Cotisation cotisation, ElementsSalaire elementsSalaire) {
18.       ...
19.    }
20.
21.    // getters et setters
22.    ...
23. }
```

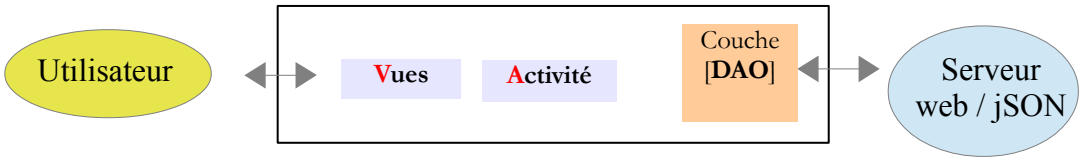
- ligne 10 : la classe [Cotisation] a été présentée page 269 ;

La classe [ElementsSalaire] (ligne 11) est la suivante :

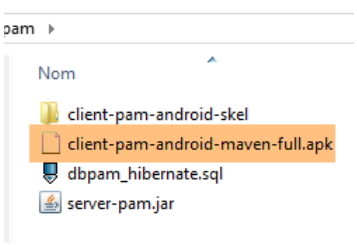
```
1.  package pam.entities;
2.
3.  import java.io.Serializable;
4.
5.  public class ElementsSalaire implements Serializable {
6.
7.     private static final long serialVersionUID = 1L;
8.     // champs privés
9.     private double salaireBase;
10.    private double cotisationsSociales;
11.    private double indemnitesEntretien;
12.    private double indemnitesRepas;
13.    private double salaireNet;
14.
15.    // constructeurs
16.    public ElementsSalaire() {
17.
18.    }
```

```
19.
20.     public ElementsSalaire(double salaireBase, double cotisationsSociales, double indemnitesEntretien,
    double indemnitesRepas, double salaireNet) {
21.         ...
22.     }
23.
24.     // getters et setters
25.     ...
26. }
```

3.4 Tests du client Android

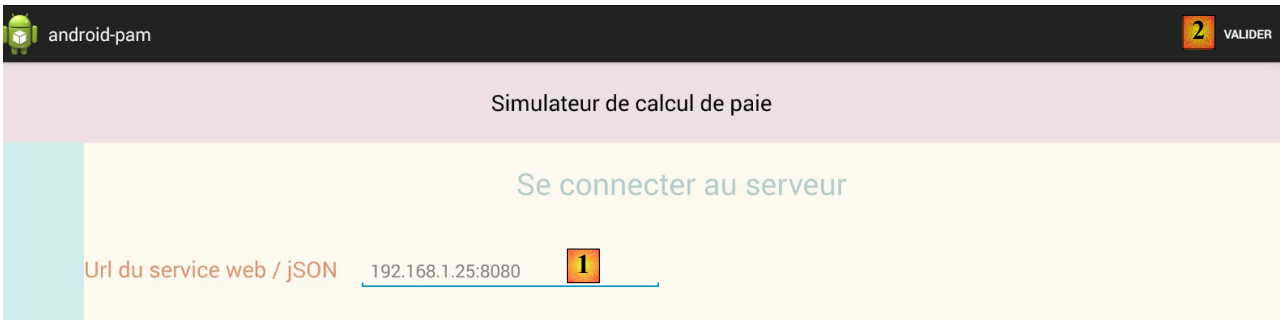


Le binaire exécutable du client Android terminé vous est donné :



Avec la souris déposez le binaire [client-pam-android-maven-full.apk] ci-dessus sur un émulateur de tablette [GenyMotion]. Il va alors être enregistré puis exécuté. Lancez également le serveur web / jSON si ce n'est déja fait. Le client Android a pour objet de récupérer les informations renvoyées par le serveur web / jSON et de les mettre en forme. Les différentes vues du client Android sont les suivantes :

Il faut tout d'abord se connecter au service web / jSON :

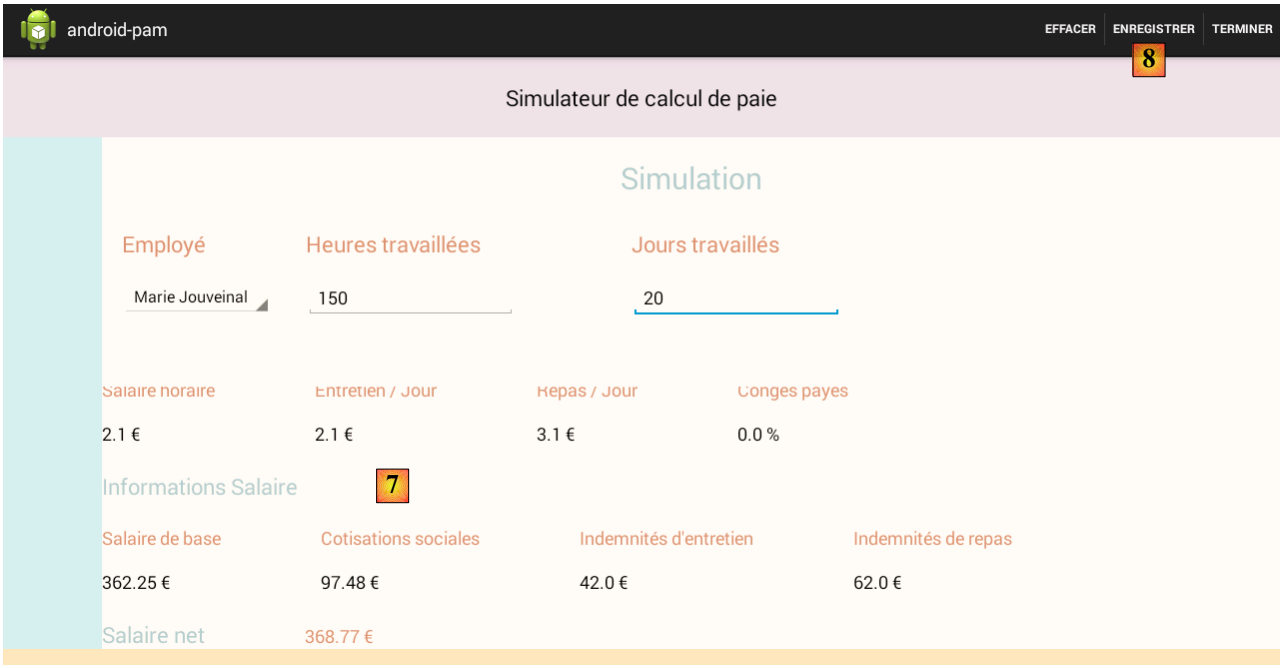


- en [1], on donne l'URL du service web / jSON. Avec l'émulateur, mettez l'une des adresses IP du PC (mais pas 127.0.0.1). Avec une tablette, mettez l'adresse wifi de la machine du serveur web / jSON ;
- en [2], on se connecte ;

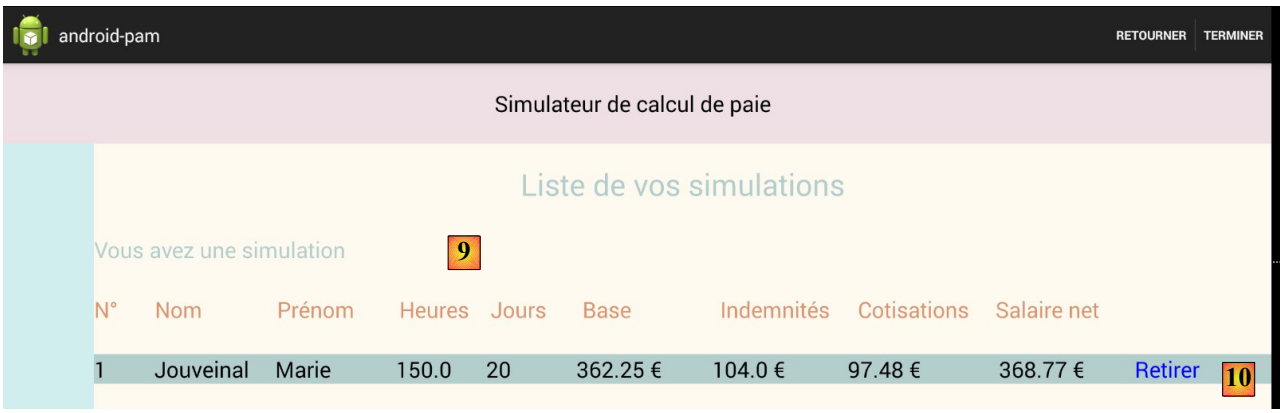
On arrive alors à la page de simulation :

- en [3], on choisit un employé ;
- en [4], on indique un nombre d'heures ;
- en [5], on indique un nombre de jours ;
- en [6], on demande la simulation ;

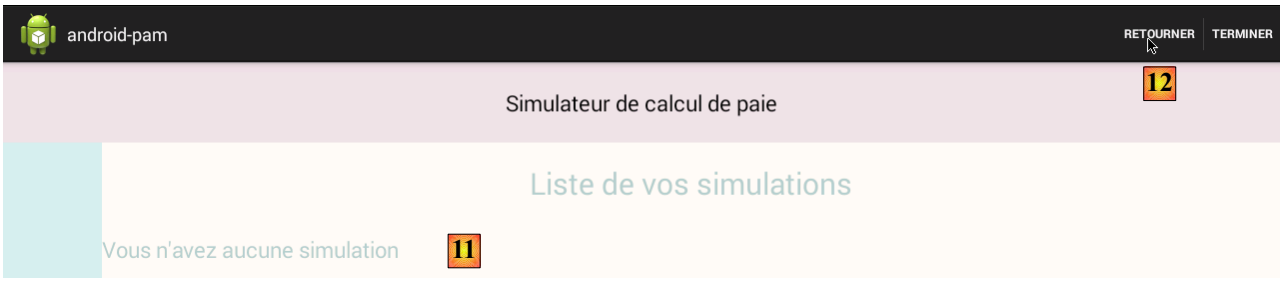
La page de simulation obtenue est la suivante :



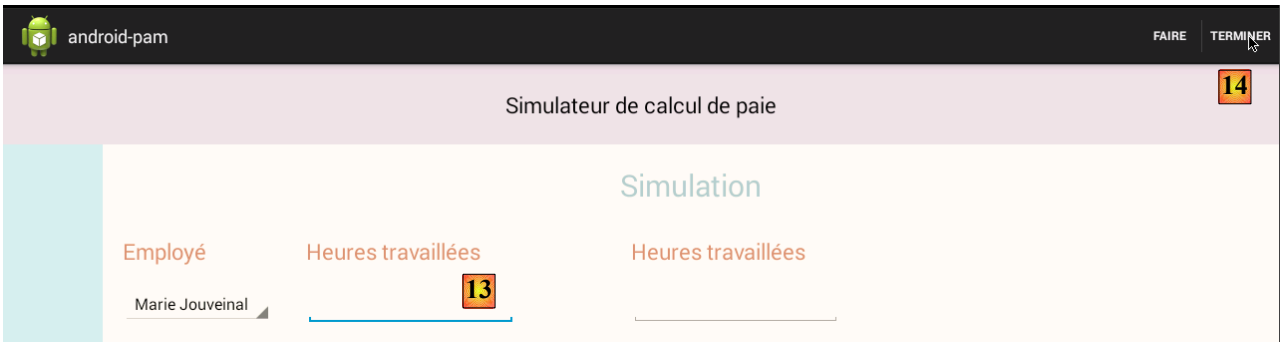
- en [7], la simulation obtenue ;
- en [8], on l'enregistre ;



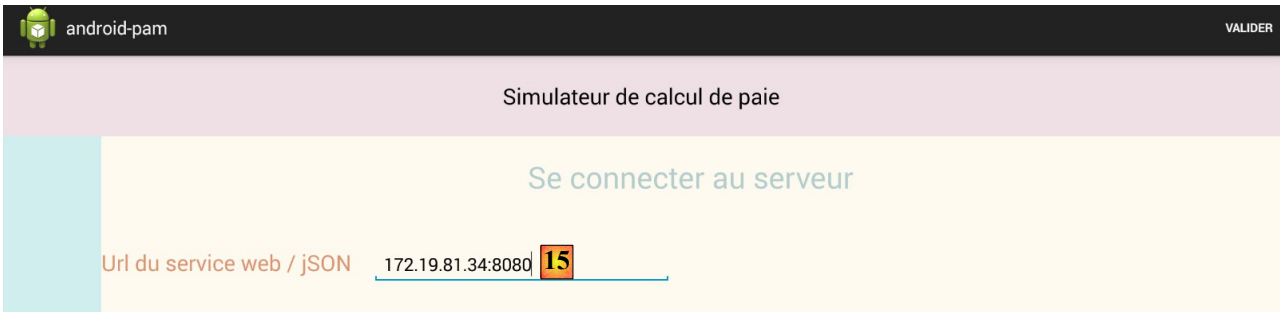
- en [9], la liste des simulations ;
- en [10], on retire une simulation ;



- en [11], il n'y a plus de simulations ;
- en [12], on retourne au formulaire de simulation ;



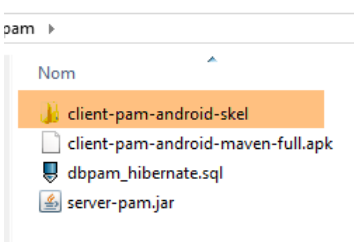
- en [13], on retrouve un formulaire vide ;
- en [14], on termine la session de simulations ;



- en [15], on retrouve le formulaire de connexion initial.

3.5 Travail à faire

Le squelette du client Android présenté précédemment vous est donné.



Le projet est exécutable et a déjà les vues nécessaires. Il y a simplement du code à rajouter pour que l'application fasse ce qu'elle a à faire. La procédure à suivre est la suivante :

- exécutez la version complète pour appréhender le travail à faire ;
- exécutez la version allégée et étudiez le code de celle-ci. Il respecte les méthodes de conception utilisées dans les pages précédentes ;
- ajoutez le code manquant ;

Introduction à la programmation

de tablettes Android par l'exemple

-

TP 2 - Pilotage d'une carte ARDUINO

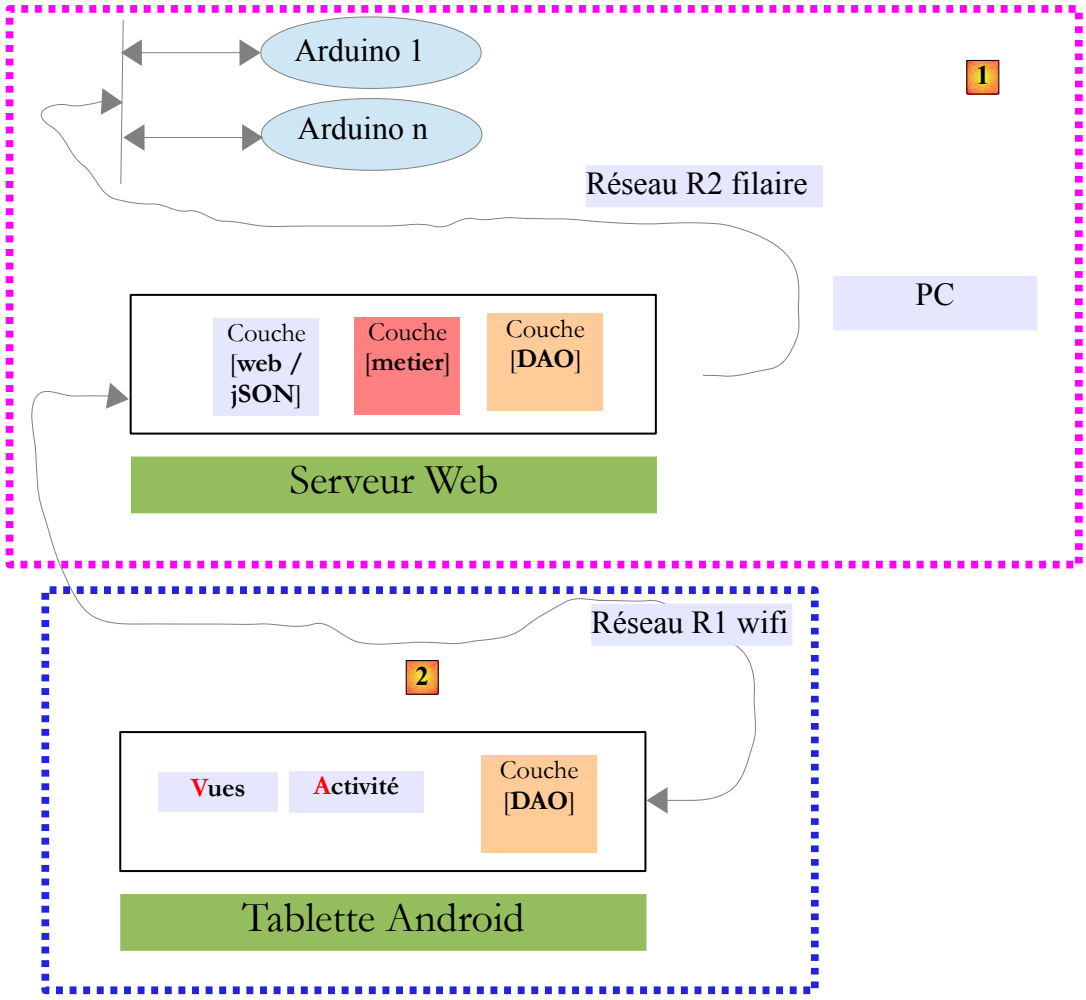
Serge Tahé, IstiA - université d'Angers
novembre 2014

4 TP 2 - Piloter des Arduinos avec une tablette Android

Nous allons maintenant apprendre à piloter une carte Arduino avec une tablette. L'exemple à suivre est celui du projet [exemple-10-client] du cours (cf paragraphe 1.11, page 103).

4.1 Architecture du projet

L'ensemble du projet aura l'architecture suivante :



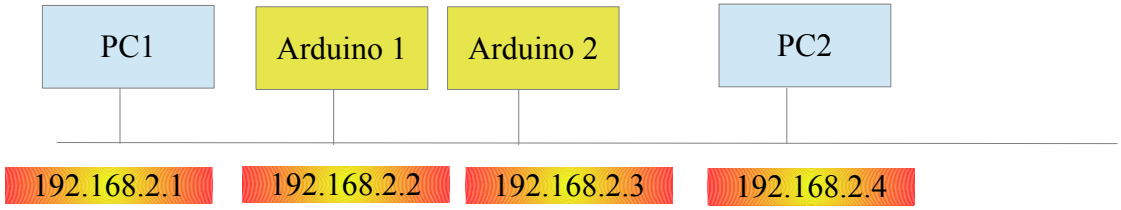
- le bloc [1], serveur web / jSON et Arduinos vous sera donné ;
- vous aurez à construire le bloc [2], la programmation de la tablette Android pour dialoguer avec le serveur web / jSON.

4.2 Le matériel

Vous avez à votre disposition les éléments suivants :

- un Arduino avec une extension Ethernet, une led et un capteur de température ;
- un miniHub à partager avec un autre étudiant ;
- un câble USB pour alimenter l'Arduino ;
- deux câbles réseau pour connecter l'Arduino et le PC sur un même réseau privé ;
- une tablette Android ;

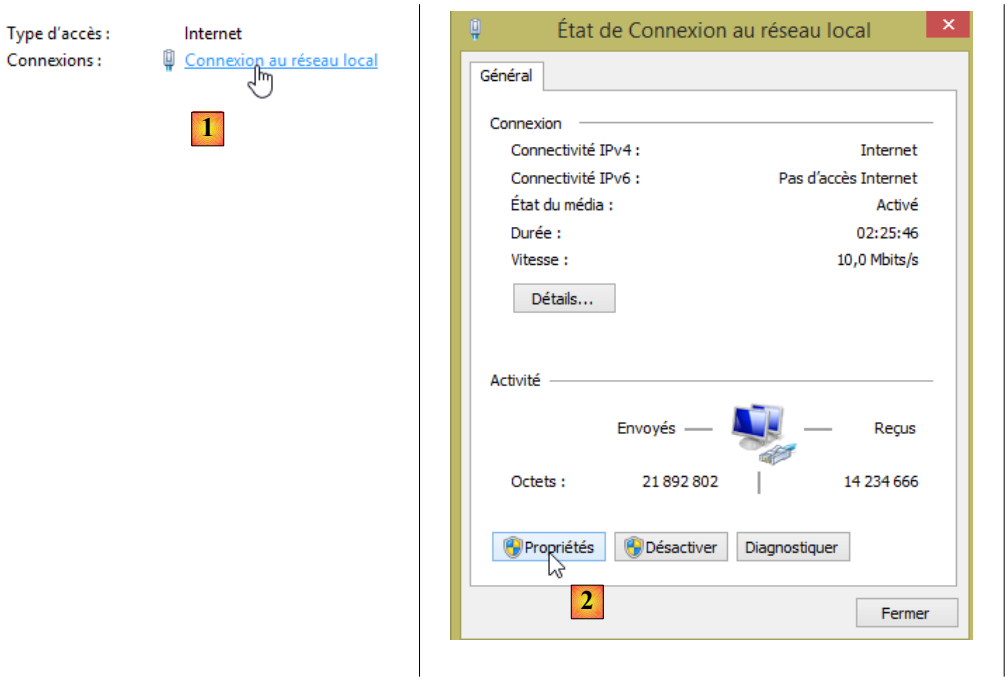
4.2.1 L'Arduino



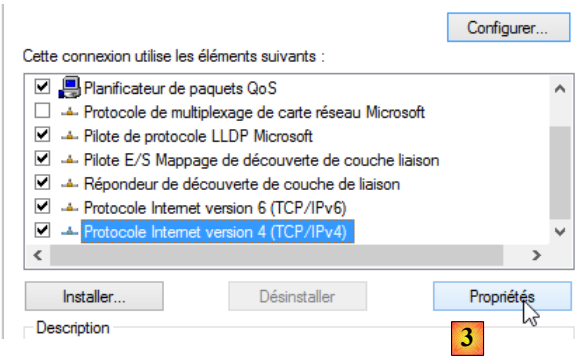
Voici comment procéder pour connecter les différents éléments ensemble :

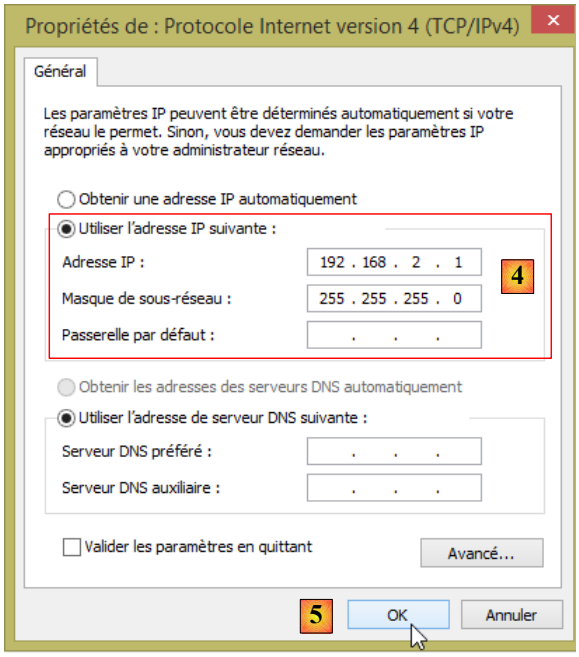
- retirez le câble réseau de votre PC ;
- joignez votre PC et l'Arduino par un câble réseau ;
- l'Arduino dont vous disposerez aura déjà été programmé. Son adresse IP sera [192.168.2.2]. Pour que votre PC voit l'Arduino, il faut lui donner une adresse IP sur le réseau [192.168.2]. Les Arduinos ont été programmés pour dialoguer avec un PC ayant l'adresse IP [192.168.2.1]. Voici comment procéder :

Allez sur [Panneau de configuration\Réseau et Internet\Centre Réseau et partage] :



- en [1], cliquez sur le lien [réseau local] ;
- en [2], cliquez le bouton [Propriétés] du réseau local ;





- en [3], cliquez sur les propriétés [IPv4] de la carte [réseau local] ;
- en [4], donnez à cette carte l'adresse IP [192.168.2.1] et le masque de sous-réseau [255.255.255.0] ;
- en [5], cliquez sur [OK] autant de fois que nécessaire pour sortir de l'assistant.

4.2.2 La tablette

- à l'aide de votre clé wifi, connectez-votre poste au réseau wifi qu'on vous indiquera. Faites de même avec votre tablette ;
- vérifiez l'adresse IP wifi de votre PC en faisant [ipconfig] dans une fenêtre DOS. Vous allez trouver une adresse du genre [192.168.x.y] ;

```
dos>ipconfig

Configuration IP de Windows

Carte réseau sans fil Wi-Fi :

   Suffixe DNS propre à la connexion. . . :
   Adresse IPv6 de liaison locale. . . . .: fe80::39aa:47f6:7537:f8e1%2
   Adresse IPv4. . . . . . . . . . . . . .: 192.168.1.25
   Masque de sous-réseau. . . . . . . . . : 255.255.255.0
   Passerelle par défaut. . . . . . . . . : 192.168.1.1
```

- vérifiez l'adresse IP wifi de votre tablette. Demandez à votre encadrant comment faire si vous ne savez pas. Vous allez trouver une adresse du genre [192.168.x.z] ;
- inhibez le pare-feu de votre PC s'il est actif [Panneau de configuration\Système et sécurité\Pare-feu Windows] ;
- dans une fenêtre Dos, vérifiez que le PC et la tablette peuvent communiquer en tapant la commande [ping 192.168.x.z] où [192.168.x.z] est l'adresse IP de votre tablette. La tablette doit alors répondre :

```
dos>ping 192.168.1.26

Envoi d'une requête 'Ping'  192.168.1.26 avec 32 octets de données :
Réponse de 192.168.1.26 : octets=32 temps=102 ms TTL=64
Réponse de 192.168.1.26 : octets=32 temps=134 ms TTL=64
Réponse de 192.168.1.26 : octets=32 temps=168 ms TTL=64
Réponse de 192.168.1.26 : octets=32 temps=208 ms TTL=64

Statistiques Ping pour 192.168.1.26:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
Durée approximative des boucles en millisecondes :
    Minimum = 102ms, Maximum = 208ms, Moyenne = 153ms
```

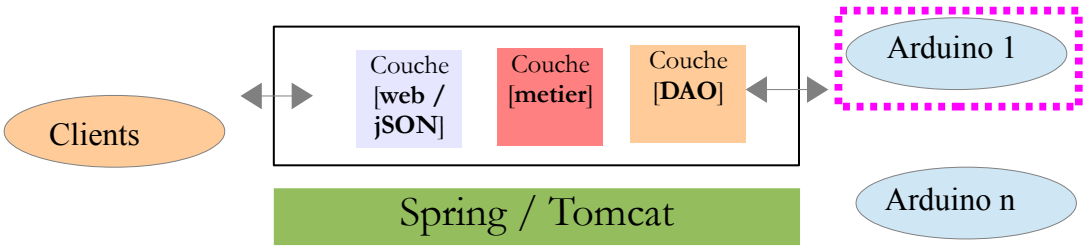
La configuration réseau de votre système est désormais prête.

4.2.3 L'émulateur [Genymotion]

L'émulateur [Genymotion] (cf paragraphe 1.16.4, page 181) remplace avantageusement la tablette. Il est quasiment aussi rapide et ne nécessite pas de réseau wifi. C'est cette méthode qu'il est conseillé d'utiliser. Vous pourrez utiliser la tablette pour la vérification finale de votre application.

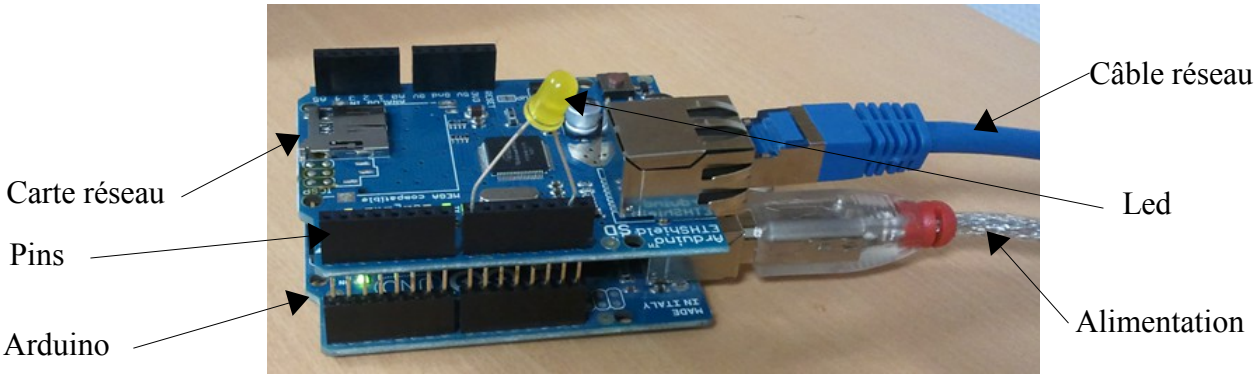
4.3 Programmation des Arduinos

Nous nous intéressons ici à l'écriture du code C des Arduinos :



A lire

- installation de l'IDE de développement Arduino (cf paragraphe 4.8.1, page 333) ;
- utilisation de bibliothèques jSON (Annexes, paragraphe 4.8.6) ;
- dans l'IDE de l'Arduino, tester l'exemple d'un serveur TCP (le serveur web par exemple) et celui d'un client TCP (le client Telnet par exemple) ;
- les annexes sur l'environnement de programmation des Arduinos au paragraphe 4.8, page 333.



Un Arduino est un ensemble de pins reliées à du matériel. Ces pins sont des entrées ou des sorties. Leur valeur est binaire ou analogique. Pour commander l'Arduino, il y aura deux opérations de base :

- **écrire une valeur** binaire / analogique sur une pin désignée par son numéro ;
- **lire une valeur** binaire / analogique sur une pin désignée par son numéro ;

A ces deux opérations de base, nous en ajouterons une troisième :

- **faire clignoter une led** pendant une certaine durée et avec une certaine fréquence. Cette opération peut être réalisée en appelant de façon répétée les deux opérations de base précédentes. Mais nous verrons aux tests que les échanges de la couche [DAO] avec un Arduino sont de l'ordre de la seconde. Il n'est alors pas possible de faire clignoter une led toutes les 100 millisecondes par exemple. Aussi implanterons-nous sur l'Arduino lui-même cette fonction de clignotement.

Le fonctionnement de l'Arduino sera le suivant :

- les communications entre la couche [DAO] et un Arduino se font via un réseau TCP-IP par échanges de lignes de texte au format jSON (**J**ava**S**cript **O**bject **N**otation) ;

- au démarrage, l'Arduino vient se connecter au port 100 d'un serveur d'enregistrement présent dans la couche [DAO]. Il envoie au serveur une unique ligne de texte :

```
{"id":"192.168.2.3","desc":"duemilanove","mac":"90:A2:DA:00:1D:A7","port":102}
```

  C'est une chaîne jSON caractérisant l'Arduino qui se connecte :
  - **id** : un identifiant de l'Arduino, ici son adresse IP ;
  - **desc** : une description de ce que sait faire l'Arduino. Ici on a simplement mis le type de l'Arduino ;
  - **mac** : adresse Mac de l'Arduino ;
  - **port** : le numéro du port sur lequel l'Arduino va attendre les commandes de la couche [DAO].

  Toutes ces informations sont de type chaînes de caractères sauf le port qui est un nombre entier.
- une fois que l'Arduino s'est inscrit auprès du serveur d'enregistrement, il se met à l'écoute sur le port qu'il a indiqué au serveur. Il attend des commandes jSON de la forme suivante :

```
{"id":"identifiant","ac":"une_action","pa":{"param1":"valeur1","param2":"valeur2",...}}
```

  C'est une chaîne jSON avec les éléments suivants :
  - **id** : un identifiant de la commande. Peut être quelconque ;
  - **ac** : une action. Il y en a trois :
    - **pw** (**pin write**) pour écrire une valeur sur une pin,
    - **pr** (**pin read**) pour lire la valeur d'une pin,
    - **cl** (**clignoter**) pour faire clignoter une led ;
  - **pa** : les paramètres de l'action. Ils dépendent de l'action.

- l'Arduino renvoie **systématiquement** une réponse à son client. Celle-ci est une chaîne jSON de la forme suivante :

```
{"id":"1","er":"0","et":{"pinx":"valx"}}
```

  où
  - **id** : l'identifiant de la commande à laquelle on répond ;
  - **er (erreur)** : un code d'erreur s'il y a eu une erreur, 0 sinon ;
  - **et (état)** : un dictionnaire toujours vide sauf pour la commande de lecture **pr**. Le dictionnaire contient alors la valeur de la pin n° **x** demandée.

Voici des exemples destinés à clarifier les spécifications précédentes :

Faire clignoter la led n° 8 10 fois avec une période de 100 millisecondes :

Commande	`{"id":"1","ac":"cl","pa":{"pin":"8","dur":"100","nb":"10"}}`
Réponse	`{"id":"1","er":"0","et":{}}`

Les paramètres **pa** de la commande **cl** sont : la durée **dur** en millisecondes d'un clignotement, le nombre **nb** de clignotements, le n° **pin** de la pin de la led.

Ecrire la valeur binaire 1 sur la pin n° 7 :

Commande	`{"id":"2","ac":"pw","pa":{"pin":"7","mod":"b","val":"1"}}`
Réponse	`{"id":"2","er":"0","et":{}}`

Les paramètres **pa** de la commande **pw** sont : le mode **mod** ***b*** (binaire) ou ***a*** (analogique) de l'écriture, la valeur **val** à écrire, le n° **pin** de la pin. Pour une écriture binaire, **val** est 0 ou 1. Pour une écriture analogique, **val** est dans l'intervalle [0,255].

Ecrire la valeur analogique 120 sur la pin n° 2 :

Commande	`{"id":"3","ac":"pw","pa":{"pin":"2","mod":"a","val":"120"}}`
Réponse	`{"id":"3","er":"0","et":{}}`

Lire la valeur analogique de la pin 0 :

Commande	`{"id":"4","ac":"pr","pa":{"pin":"0","mod":"a"}}`
Réponse	`{"id":"4","er":"0","et":{"pin0":"1023"}}`

Les paramètres **pa** de la commande **pr** sont : le mode **mod b** (binaire) ou **a** (analogique) de la lecture, le n° **pin** de la pin. S'il n'y a pas d'erreur, l'Arduino met dans le dictionnaire **"et"** de sa réponse, la valeur de la pin demandée. Ici **pin0** indique que c'est la valeur de la pin n° 0 qui a été demandée et **1023** est cette valeur. En lecture, une valeur analogique sera dans l'intervalle [0, 1024].

Nous avons présenté les trois commandes **cl**, **pw** et **pr**. On peut se demander pourquoi on n'a pas utilisé des champs plus explicites dans les chaînes jSON, **action** au lieu de **ac**, **pinwrite** au lieu de **pw**, **parametres** au lieu de **pa**, ... Un Arduino a une mémoire très réduite. Or les chaînes jSON échangées avec l'Arduino participent à l'occupation mémoire. On a donc choisi de raccourcir celles-ci au maximum.

Voyons maintenant quelques cas d'erreur :

Commande	`xx`
Réponse	`{"id":"","er":"100","et":{}}`

On a envoyé une commande qui n'est pas au format jSON. L'Arduino a renvoyé le code d'erreur 100.

Commande	`{"id":"4","ac":"pr","pa":{"mod":"a"}}`
Réponse	`{"id":"4","er":"302","et":{}}`

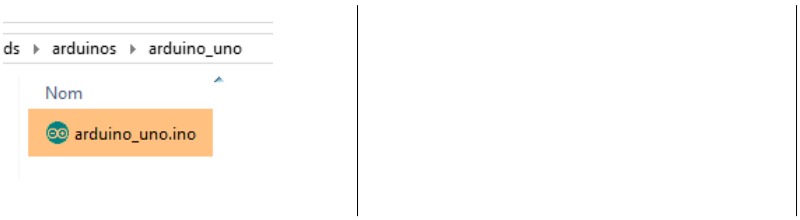
On a envoyé une commande **pr** en oubliant le paramètre **pin**. L'Arduino a renvoyé le code d'erreur 302.

Commande	`{"id":"4","ac":"pinread","pa":{"pin":"0","mod":"a"}}`
Réponse	`{"id":"4","er":"104","et":{}}`

On a envoyé une commande **pinread** inconnue (c'est pr). L'Arduino a renvoyé le code d'erreur 104.

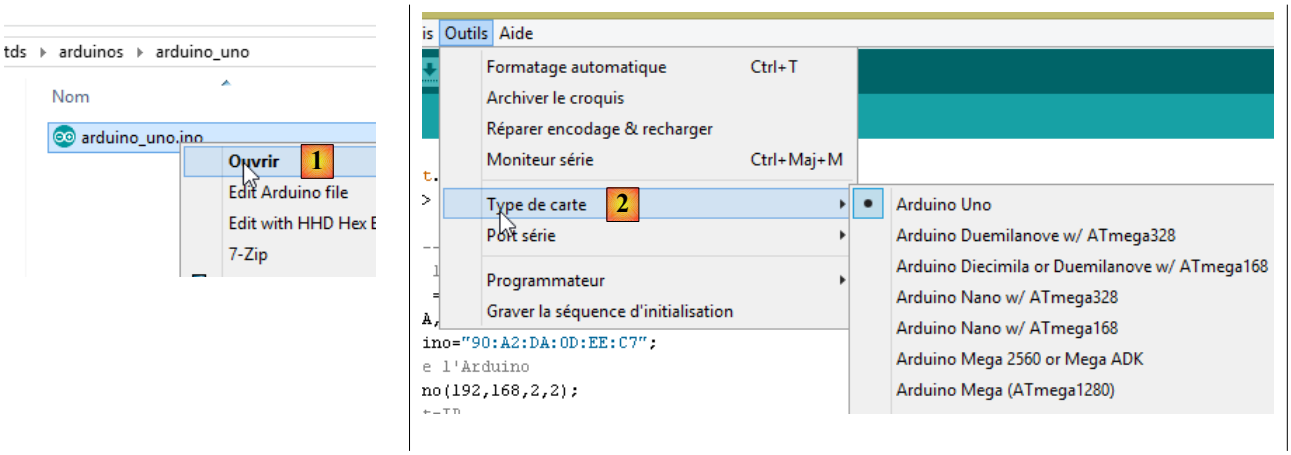
On ne continuera pas les exemples. La règle est simple. L'Arduino **ne doit pas planter**, quelque soit la commande qu'on lui envoie. Avant d'exécuter une commande jSON, il s'assure que celle-ci est correcte. Dès qu'une erreur apparaît, l'Arduino arrête l'exécution de la commande et renvoie à son client la chaîne jSON d'erreur. Là encore, parce qu'on est contraint en espace mémoire, on renvoie un code d'erreur plutôt qu'un message complet.

Le code du programme exécuté sur l'Arduino vous est fourni dans les exemples de ce document :

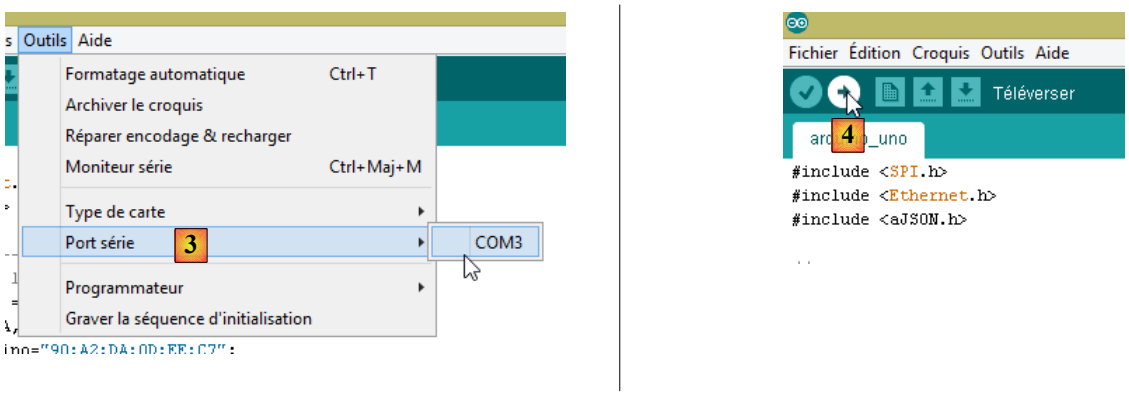


Pour le transférer sur l'Arduino :

- connectez celui-ci à votre PC ;



- en [1], ouvrez le fichier [arduino_uno.ino]. L'IDE Arduino va se lancer et charger le fichier ;
- en [2], indiquez le type d'Arduino utilisé ;



- en [3], indiquez sur quel port série du PC il se trouve ;
- en [4], téléversez (=chargez) le programme [arduino_uno] sur l'Arduino ;

Le code du programme est très commenté. Le lecteur intéressé pourra s'y référer. Nous signalons simplement les lignes du code qui permettent de configurer la communication bidirectionnelle client / serveur entre l'Arduino et le PC :

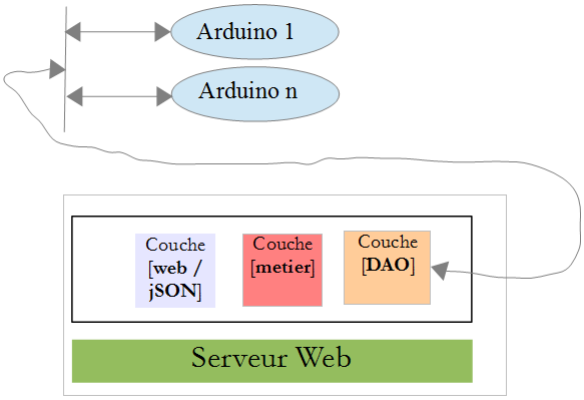
```
1.  #include <SPI.h>
2.  #include <Ethernet.h>
3.  #include <ajSON.h>
4.
5.  // --------------------------------- CONFIGURATION DE L'ARDUINO UNO
6.  // adresse MAC de l'Arduino UNO
7.  byte macArduino[] = {
8.    0x90, 0xA2, 0xDA, 0x0D, 0xEE, 0xC7 };
9.  char * strMacArduino="90:A2:DA:0D:EE:C7";
10. // l'adresse IP de l'Arduino
11. IPAddress ipArduino(192,168,2,2);
12. // son identifiant
13. char * idArduino="cuisine";
14. // port du serveur Arduino
15. int portArduino=102;
16. // description de l'Arduino
17. char * descriptionArduino="contrôle domotique";
18. // le serveur Arduino travaillera sur le port 102
19. EthernetServer server(portArduino);
20. // IP du serveur d'enregistrement
21. IPAddress ipServeurEnregistrement(192,168,2,1);
22. // port du serveur d'enregistrement
23. int portServeurEnregistrement=100;
24. // le client Arduino du serveur d'enregistrement
25. EthernetClient clientArduino;
26. // la commande du client
```

```
27. char commande[100];
28. // la réponse de l'Arduino
29. char message[100];
30.
31. // initialisation
32. void setup() {
33.   // Le moniteur série permettra de suivre les échanges
34.   Serial.begin(9600);
35.   // démarrage de la connection Ethernet
36.   Ethernet.begin(macArduino,ipArduino);
37.   // mémoire disponible
38.   Serial.print(F("Memoire disponible : "));
39.   Serial.println(freeRam());
40. }
41.
42. // boucle infinie
43. void loop()
44. {
45.   ...
46. }
```

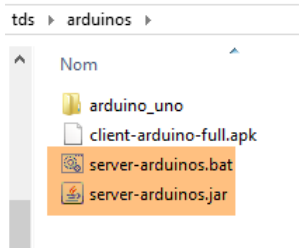
- ligne 8 : l'adresse Mac de l'arduino. Elle n'a pas beaucoup d'importance ici car l'Arduino va être sur un réseau privé où il y a un PC et un ou plusieurs Arduinos. Il faut simplement que l'adresse Mac soit unique sur ce réseau privé. Normalement, la carte réseau de l'Arduino a un sticker où est indiquée l'adresse Mac de la carte. Si ce sticker est absent et si vous ne connaissez pas l'adresse Mac de la carte, vous pouvez mettre ce que vous voulez en ligne 8 tant que la règle d'unicité de l'adresse Mac sur le réseau privé est respectée ;
- ligne 11 : l'adresse IP de la carte. De nouveau, on met ce qu'on veut du type [192.168.2.x] et on fait varier x pour les différents Arduinos du réseau privé ;
- ligne 13 : identifiant de l'Arduino. Doit être unique parmi les identifiants des Arduinos d'un même réseau privé ;
- ligne 15 : le port de service de l'Arduino. On peut mettre ce qu'on veut ;
- ligne 17 : la description de la fonction de l'Arduino. On peut mettre ce qu'on veut. Attention aux longues chaînes à cause de la mémoire restreinte de l'Arduino ;
- ligne 21 : adresse IP du serveur d'enregistrement de l'Arduino sur le PC. **Ne doit pas être modifié** ;
- ligne 23 : port de ce service d'enregistrement. **Ne doit pas être modifié** ;

4.4 Le serveur web / jSON Java

4.4.1 Installation



Le binaire Java du serveur web / jSON vous est donné :



Placez les deux fichiers [server-arduinos.*] sur le bureau et exécutez le fichier [server-arduinos.bat] en double-cliquant dessus. Une autre façon de faire est d'ouvrir une fenêtre de commandes et de taper la commande qui est dans le fichier [server-arduinos.bat] :

```
dos>java -jar server-arduinos.jar
```

Si [java.exe] n'est pas dans le PATH de la fenêtre de commandes, il sera nécessaire de taper le chemin complet de [java.exe] (en général c:\Program Files\java\...).

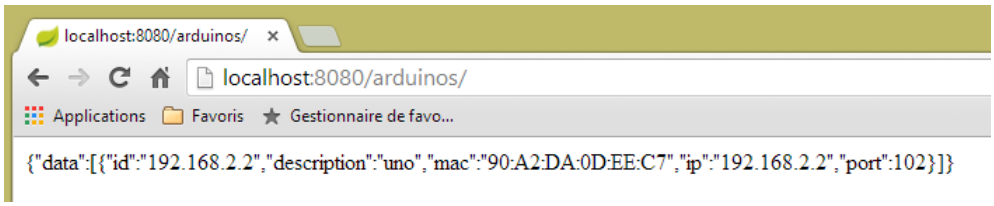
Une fenêtre DOS va s'ouvrir et afficher des logs :

```
1.  .   ____          _            __ _ _
2.   /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
3.  ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
4.   \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
5.    '  |____| .__|_| |_|_| |_\__, | / / / /
6.   =========|_|==============|___/=/_/_/_/
7.   :: Spring Boot ::             (v0.5.0.M6)
8.
9.  2014-01-06 11:11:35.550  INFO 8408 --- [           main] arduino.rest.metier.Application          :
    Starting Application on Gportpers3 with PID 8408 (C:\Users\SergeTahÚ\Desktop\part2\server.jar started by
    ST)
10. 2014-01-06 11:11:35.587  INFO 8408 --- [           main] ationConfigEmbeddedWebApplicationContext :
    Refreshing
    org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@6a4ba620:
    startup date [Mon Jan 06 11:11:35 CET 2014]; root of context hierarchy
11. 2014-01-06 11:11:36.765  INFO 8408 --- [           main] o.apache.catalina.core.StandardService   :
    Starting service Tomcat
12. 2014-01-06 11:11:36.766  INFO 8408 --- [           main] org.apache.catalina.core.StandardEngine  :
    Starting Servlet Engine: Apache Tomcat/7.0.42
13. 2014-01-06 11:11:36.876  INFO 8408 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       :
    Initializing Spring embedded WebApplicationContext
14. 2014-01-06 11:11:36.877  INFO 8408 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root
    WebApplicationContext: initialization completed in 1293 ms
15. 2014-01-06 11:11:37.084  INFO 8408 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       :
    Initializing Spring FrameworkServlet 'dispatcherServlet'
16. 2014-01-06 11:11:37.084  INFO 8408 --- [ost-startStop-1] o.s.web.servlet.DispatcherServlet        :
    FrameworkServlet 'dispatcherServlet': initialization started
17. 2014-01-06 11:11:37.184  INFO 8408 --- [ost-startStop-1] o.s.w.s.handler.SimpleUrlHandlerMapping  :
    Mapped URL path [/**/favicon.ico] onto handler of type [class
    org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
18. 2014-01-06 11:11:37.386  INFO 8408 --- [ost-startStop-1] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/arduinos/blink/{idCommande}/{idArduino}/{pin}/{duree}/
    {nombre}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    java.lang.String
    arduino.rest.metier.RestMetier.faireClignoterLed(java.lang.String,java.lang.String,java.lang.String,java
    .lang.String,java.lang.String,javax.servlet.http.HttpServletResponse)
19. 2014-01-06 11:11:37.388  INFO 8408 --- [ost-startStop-1] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/arduinos/commands/
    {idArduino}],methods=[POST],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    java.lang.String
    arduino.rest.metier.RestMetier.sendCommandesJson(java.lang.String,java.lang.String,javax.servlet.http.Ht
    tpServletResponse)
20. 2014-01-06 11:11:37.388  INFO 8408 --- [ost-startStop-1] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/arduinos/],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    java.lang.String arduino.rest.metier.RestMetier.getArduinos(javax.servlet.http.HttpServletResponse)
21. 2014-01-06 11:11:37.389  INFO 8408 --- [ost-startStop-1] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/arduinos/pinRead/{idCommande}/{idArduino}/{pin}/
    {mode}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    java.lang.String
```

```
    arduino.rest.metier.RestMetier.pinRead(java.lang.String,java.lang.String,java.lang.String,java.lang.Stri
    ng,javax.servlet.http.HttpServletResponse)
22. 2014-01-06 11:11:37.390  INFO 8408 --- [ost-startStop-1] s.w.s.m.m.a.RequestMappingHandlerMapping :
    Mapped "{[/arduinos/pinWrite/{idCommande}/{idArduino}/{pin}/{mode}/
    {valeur}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]}" onto public
    java.lang.String
    arduino.rest.metier.RestMetier.pinWrite(java.lang.String,java.lang.String,java.lang.String,java.lang.Str
    ing,java.lang.String,javax.servlet.http.HttpServletResponse)
23. 2014-01-06 11:11:37.463  INFO 8408 --- [ost-startStop-1] o.s.w.s.handler.SimpleUrlHandlerMapping  :
    Mapped URL path [/**] onto handler of type [class
    org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
24. 2014-01-06 11:11:37.464  INFO 8408 --- [ost-startStop-1] o.s.w.s.handler.SimpleUrlHandlerMapping  :
    Mapped URL path [/webjars/**] onto handler of type [class
    org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
25. 2014-01-06 11:11:37.881  INFO 8408 --- [ost-startStop-1] o.s.web.servlet.DispatcherServlet        :
    FrameworkServlet 'dispatcherServlet': initialization completed in 796 ms
26. Serveur d'enregistrement lancÚ sur 192.168.2.1:100
27. 2014-01-06 11:11:38.101  INFO 8408 --- [       Thread-4] arduino.dao.Recorder                    :
    Recorder : [11:11:38:101] : [Serveur d'enregistrement : attente d'un client]
28. 2014-01-06 11:11:38.142  INFO 8408 --- [           main] arduino.rest.metier.Application : Started
    Application in 3.257 seconds
```

- ligne 11 : un serveur Tomcat embarqué est lancé ;
- ligne 15 : la servlet [dispatcherServlet] de Spring MVC est chargée et exécutée ;
- ligne 18 : l'URL Rest [/arduinos/blink/{idCommande}/{idArduino}/{pin}/{duree}/{nombre}] est détectée ;
- ligne 19 : l'URL Rest [/arduinos/commands/{idArduino}] est détectée ;
- ligne 20 : l'URL Rest [/arduinos/] est détectée ;
- ligne 21 : l'URL Rest [/arduinos/pinRead/{idCommande}/{idArduino}/{pin}/{mode}] est détectée ;
- ligne 22 : l'URL Rest [/arduinos/pinWrite/{idCommande}/{idArduino}/{pin}/{mode}/{valeur}] est détectée ;
- ligne 26 : le serveur d'enregistrement des Arduinos est lancé ;

Connectez votre Arduino au PC si ce n'est déja fait. **Le pare-feu du PC doit être désactivé**. Puis avec un navigateur demandez l'URL [http://localhost:8080/arduinos] :



Vous devez voir apparaître l'identifiant de l'Arduino connecté. Si vous n'avez rien, pensez à resetter l'Arduino. Il a un bouton poussoir pour cela.

Le serveur web / jSON est désormais installé.

4.4.2 Les URL exposées par le service web / jSON

A lire : projet [exemple-10-server] (cf paragraphe 1.11.1, page 103) ;

Le service web / jSON a été implémenté avec Spring MVC et expose les URL suivantes :

```
1.    // liste des arduinos
2.    @RequestMapping(value = "/arduinos", method = RequestMethod.GET)
3.    public ArduinosResponse getArduinos() {
4.  ...
5.    }
```

```
6.    // clignotement
7.    @RequestMapping(value = "/arduinos/blink/{idCommande}/{idArduino}/{pin}/{duree}/
    {nombre}", method = RequestMethod.GET)
8.    public GenericResponse faireClignoterLed(@PathVariable("idCommande") String idCommande,
    @PathVariable("idArduino") String idArduino,
```

```
    @PathVariable("pin") int pin, @PathVariable("duree") int duree, @PathVariable("nombre")
    int nombre) {
9.  ...
10.   }

11.   // envoi de commandes jSON
12.   @RequestMapping(value = "/arduinos/commands/{idArduino}", method = RequestMethod.POST,
    consumes = "application/json")
13.   public CommandsResponse sendCommandesJson(@PathVariable("idArduino") String idArduino,
    @RequestBody List<ArduinoCommand> commandes) {
14.    ...
15.   }

16.   // lecture pin
17.   @RequestMapping(value = "/arduinos/pinRead/{idCommande}/{idArduino}/{pin}/{mode}",
    method = RequestMethod.GET)
18.   public GenericResponse pinRead(@PathVariable("idCommande") String idCommande,
    @PathVariable("idArduino") String idArduino, @PathVariable("pin") int pin,
    @PathVariable("mode") String mode) {
19. ...
20.   }

21.   // écriture pin
22.   @RequestMapping(value = "/arduinos/pinWrite/{idCommande}/{idArduino}/{pin}/{mode}/
    {valeur}", method = RequestMethod.GET)
23.   public GenericResponse pinWrite(@PathVariable("idCommande") String idCommande,
    @PathVariable("idArduino") String idArduino, @PathVariable("pin") int pin,
    @PathVariable("mode") String mode,@PathVariable("valeur") int valeur) {
24. ...
25. }
```

Les réponses envoyées par le serveur sont des représentations jSON des classes suivantes :

[**AbstractResponse**] est la classe parente de toutes les réponses :

```
1.  package istia.st.arduinos.server.web;
2.
3.  import java.util.List;
4.
5.  public abstract class AbstractResponse {
6.
7.    // erreur
8.    private int erreur;
9.    // message d'erreur
10.   private List<String> messages;
11.
12.   // getters et setters
13. ...
14. }
```

[**ArduinosResponse**] contient la liste des Arduinos connectés. Elle est envoyée en réponse à l'URL [**/arduinos**] :

```
1.  package istia.st.arduinos.server.web;
2.
3.  import istia.st.arduinos.server.entities.Arduino;
4.
5.  import java.util.List;
6.
7.  public class ArduinosResponse extends AbstractResponse {
8.
9.    // liste des Arduinos
10.   private List<Arduino> arduinos;
11.
12.   // getters et setters
13.   ...
14. }
```

La classe [**Arduino**] est la suivante :

```
1.  package android.arduinos.entities;
2.
3.  import java.io.Serializable;
4.
5.  public class Arduino implements Serializable {
6.    // données
7.    private String id;
8.    private String description;
9.    private String mac;
10.   private String ip;
11.   private int port;
12.
13.  // getters et setters
14. ...
15. }
```

- ligne 7 : [id] est l'identifiant de l'arduino ;
- ligne 8 : sa description ;
- ligne 9 : son adresse MAC ;
- ligne 10 : son adresse IP ;
- ligne 11 : le port sur lequel il attend des commandes ;

[**GenericResponse**] est une réponse qui encapsule une réponse de l'Arduino. Elle est envoyée en réponse aux URL suivantes :

- [**/arduinos/blink/{idCommande}/{idArduino}/{pin}/{duree}/{nombre}**] ;
- [**/arduinos/pinRead/{idCommande}/{idArduino}/{pin}/{mode}**] ;
- [**/arduinos/pinWrite/{idCommande}/{idArduino}/{pin}/{mode}/{valeur}**] ;

```
1.  package istia.st.arduinos.server.web;
2.
3.  import istia.st.arduinos.server.entities.ArduinoResponse;
4.
5.  public class GenericResponse extends AbstractResponse {
6.
7.    // réponse de l'Arduino
8.    private ArduinoResponse response;
9.
10.   // getters et setters
11.   ...
12. }
```

[**ArduinoResponse**] représente la réponse standard d'un Arduino :

```
13. public class ArduinoResponse implements Serializable {
14.
15.   private String json;
16.   private String id;
17.   private String erreur;
18.   private Map<String, Object> etat;
19.
20.   // getters et setters
21. ...
22. }
```

- [json] : la chaîne jSON envoyée par un Arduino et qui n'a pu être décodée (cas d'erreur) ;
- [id] : l'identifiant de la commande à laquelle l'Arduino répond ;
- [erreur] : un code d'erreur, 0 si OK, autre chose sinon ;
- [etat] : un dictionnaire contenant la réponse spécifique à la commande. Il est le plus souvent vide sauf si la commande demandait la lecture d'une valeur de l'Arduino auquel cas celle-ci sera placée dans ce dictionnaire ;

[**CommandsResponse**] est la réponse du serveur lorsqu'on lui envoie une liste de commandes jSON à exécuter sur les Arduinos. Cette réponse contient la liste des réponses de l'arduino à chacune des commandes exécutées. Elle est envoyée en réponse à l'URL [**/arduinos/commands/{idArduino}**] :

```
1.  public class CommandsResponse extends AbstractResponse {
2.
3.    // réponses des Arduinos
4.    private List<ArduinoResponse> responses;
```

```
5.
6.    // getters et setters
7.    ...
8.  }
```

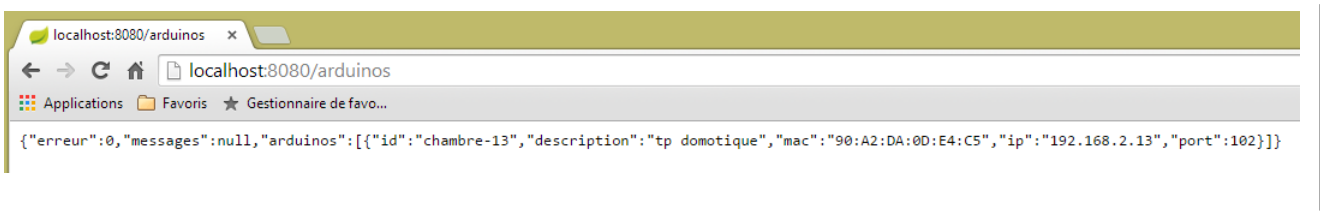
4.4.3 Les tests du service web / jSON

Habituez-vous au serveur web / jSON en testant les URL suivantes :

URL	rôle
http://localhost:8080/arduinos/	rend la liste des Arduinos connectés
http://localhost:8080/arduinos/blink/1/192.168.2.2/8/100/20/	fait clignoter la led de la pin n° 8 de l'Arduino identifié par 192.168.2.2, 20 fois toutes les 100 ms.
http://localhost:8080/arduinos/pinRead/1/192.168.2.2/0/a/	lecture analogique de la pin n° 0 de l'Arduino identifié par 192.168.2.2
http://localhost:8080/arduinos/pinRead/1/192.168.2.2/5/b/	lecture binaire de la pin n° 5 de l'Arduino identifié par 192.168.2.2
http://localhost:8080/arduinos/pinWrite/1/192.168.2.2/8/b/1/	écriture binaire de la valeur 1 sur la pin n° 8 de l'Arduino identifié par 192.168.2.2
http://localhost:8080/arduinos/pinWrite/1/192.168.2.2/4/a/100/	écriture analogique de la valeur 100 sur la pin n° 4 de l'Arduino identifié par 192.168.2.2

Voici quelques copies d'écran de ce que vous devez obtenir :

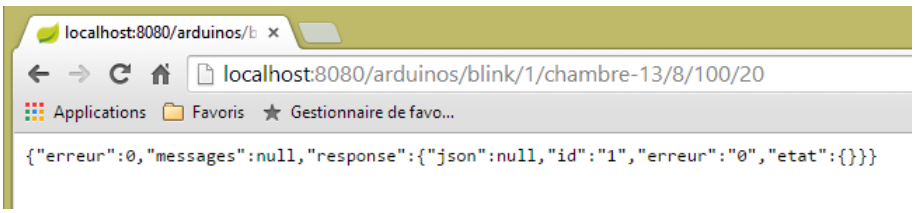
Obtenir la liste des Arduinos connectés :



La chaîne jSON reçue du serveur web / jSON est un objet avec les champs suivants :

- [erreur] : à 0 indique qu'il n'y a pas eu d'erreur - sinon il y a eu erreur ;
- [messages] : une liste de messages expliquant l'erreur s'il y a eu erreur :
- [arduinos] : la liste des Arduinos s'il n'y a pas eu d'erreur. Chaque Arduino est alors décrit par un objet avec les champs suivants :
  - [id] : identifiant de l'Arduino. Deux Arduinos ne peuvent avoir le même identifiant ;
  - [description] : courte description de la fonctionnalité de l'Arduino ;
  - [mac] : adresse Mac de l'Arduino ;
  - [ip]: adresse IP de l'Arduino ;
  - [port] : port sur lequel il attend des commandes ;

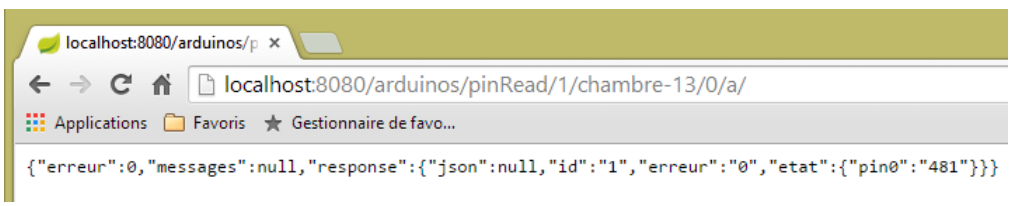
Faire clignoter la led de la pin n° 8 de l'Arduino identifié par [chambre-13], 20 fois toutes les 100 ms :



La chaîne jSON reçue du serveur web / jSON est un objet avec les champs suivants :

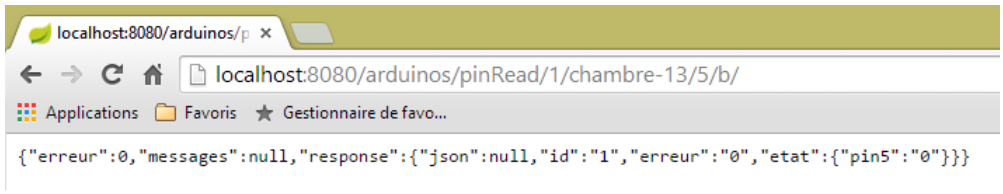
- [erreur] : à 0 indique qu'il n'y a pas eu d'erreur - sinon il y a eu erreur ;
- [messages] : une liste de messages expliquant l'erreur s'il y a eu erreur :
- [response] : la réponse de l'Arduino s'il n'y a pas eu erreur :
  - [id] : identifiant de la commande. Cet identifiant est le 1 dans [/blink/1]. L'Arduino reprend cet identifiant de commande dans sa réponse ;
  - [erreur] : un n° d'erreur. Une valeur différente de 0 signale une erreur ;
  - [etat] : n'est utilisé que pour la lecture d'une pin. A alors pour valeur, la valeur de la pin ;
  - [json] : n'est utilisé qu'en cas d'erreur jSON entre le client et le serveur. A alors pour valeur, la chaîne jSON erronée envoyée par l'Arduino ;

Lecture analogique de la pin n° 0 de l'Arduino identifié par [chambre-13] :



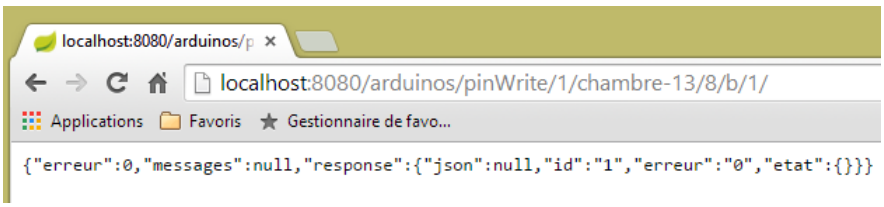
La chaîne jSON reçue du serveur web / jSON est analogue à la précédente à la différence près du champ [etat] qui représente la valeur de la pin n° 0.

Lecture binaire de la pin n° 5 de l'Arduino identifié par [chambre-13] :



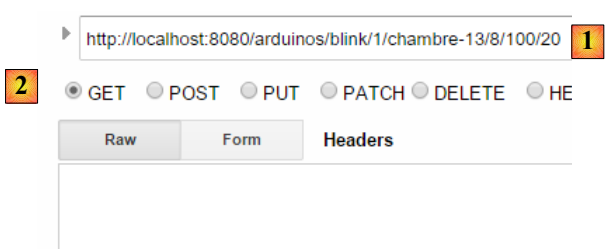
La chaîne jSON reçue du serveur web / jSON est analogue à la précédente.

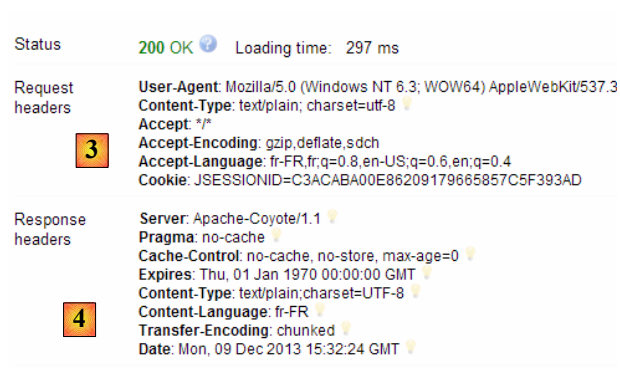
Ecriture binaire de la valeur 1 sur la pin n° 8 de l'Arduino identifié par [chambre-13] :



La chaîne jSON reçue du serveur web / jSON est analogue à la précédente.

Le test de l'URL [http://localhost:8080/arduinos/commands/chambre-13] est plus délicat. La méthode du serveur web / jSON qui traite cette URL attend une requête POST qu'on ne peut pas simuler simplement avec un navigateur. Pour tester cette URL, on pourra utiliser un navigateur Chrome avec l'extension [Advanced REST Client] (cf paragraphe 1.16.8, page 198) :

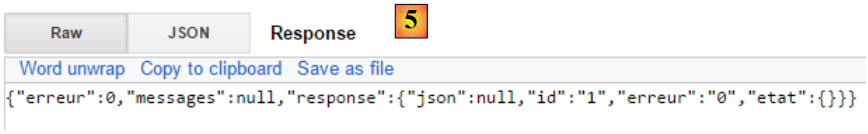




- en [1], l'URL de la méthode web / jSON à tester ;
- en [2], la méthode GET ou POST pour envoyer la requête ;

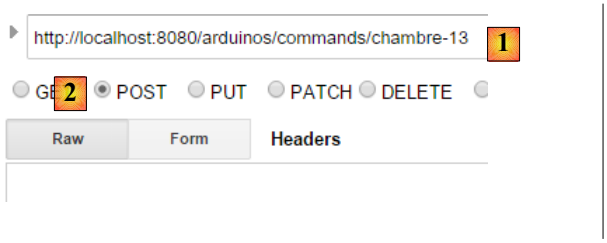
La requête est envoyée en la validant ou via le bouton [Send].

- en [3], les entêtes HTTP de la demande ;
- en [4], les entêtes HTTP de la réponse ;

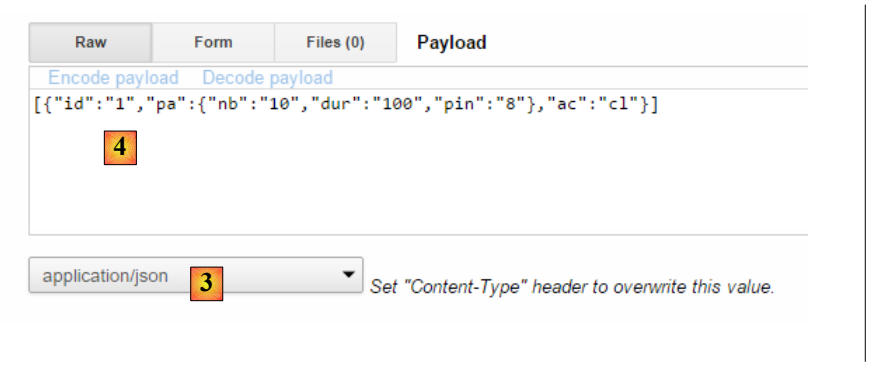


- en [5], la réponse jSON renvoyée par le serveur web / jSON ;

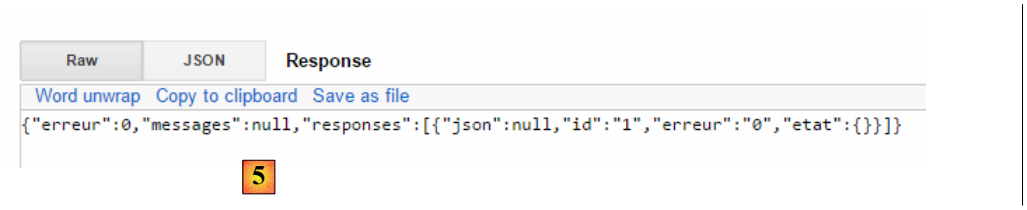
Toutes les méthodes GET du service web / jSON peuvent être testées ainsi. Pour la méthode POST [sendCommandesJson], on procèdera ainsi :



- en [1], l'URL de la méthode [sendCommandsJson] ;
- en [2], la requête se fait via la commande HTTP POST ;



- en [3], on indique que l'objet envoyé par le POST le sera sous la forme d'une chaîne jSON ;
- en [4], la liste des commandes jSON. On notera bien les crochets qui commencent et terminent la liste. Ici, dans la liste il n'y a qu'une commande jSON qui fait clignoter la pin n° 8, 10 fois toutes les 100 ms.

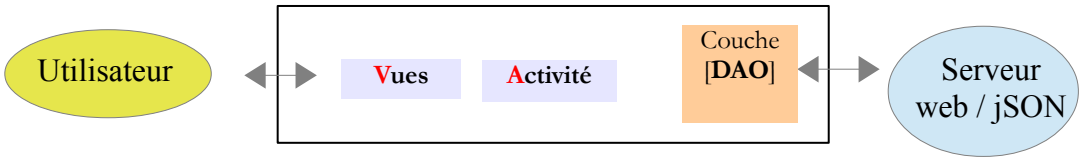


- en [5], la réponse jSON envoyée par le serveur. L'objet a reçu un objet avec les deux champs habituels [erreur, messages] et un champ [responses] dont la valeur est la liste des réponses de l'Arduino à chacune des commandes jSON envoyées.

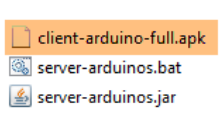
Voyons ce qui se passe lorsqu'on envoie une commande jSON syntaxiquement incorrecte pour l'Arduino :

On voit que dans la réponse de l'Arduino, le n° d'erreur est [104] indiquant par là que la commande [xx] n'a pas été reconnue.

4.5 Tests du client Android



Le binaire exécutable du client Android terminé vous est donné :

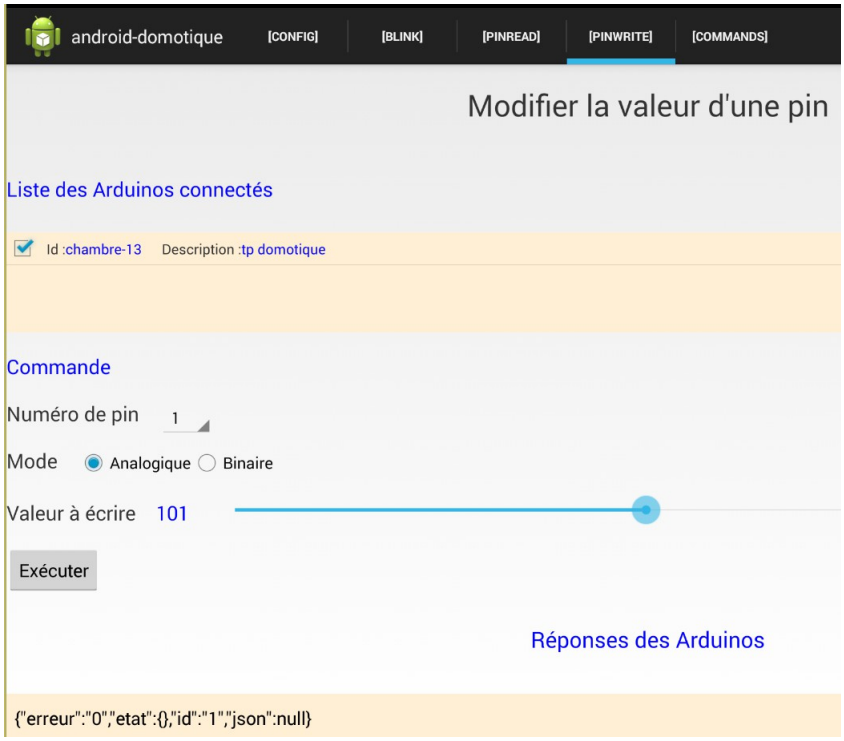


Avec la souris déposez le binaire [client-arduino-full.apk] ci-dessus sur un émulateur de tablette [GenyMotion]. Il va alors être enregistré puis exécuté. Lancez également le serveur web / jSON si ce n'est déja fait. Connectez l'Arduino au PC avec une led dessus. Le client Android permet de gérer les Arduinos à distance. Il présente à l'utilisateur les écrans suivants.

L'onglet [CONFIG] permet de se connecter au serveur et de récupérer la liste des Arduinos connectés :

- en [1], connectez-vous à l'URL [192.168.2.1:8080]. C'est l'adresse IP donnée à votre PC (cf paragraphe 4.2, page 280).

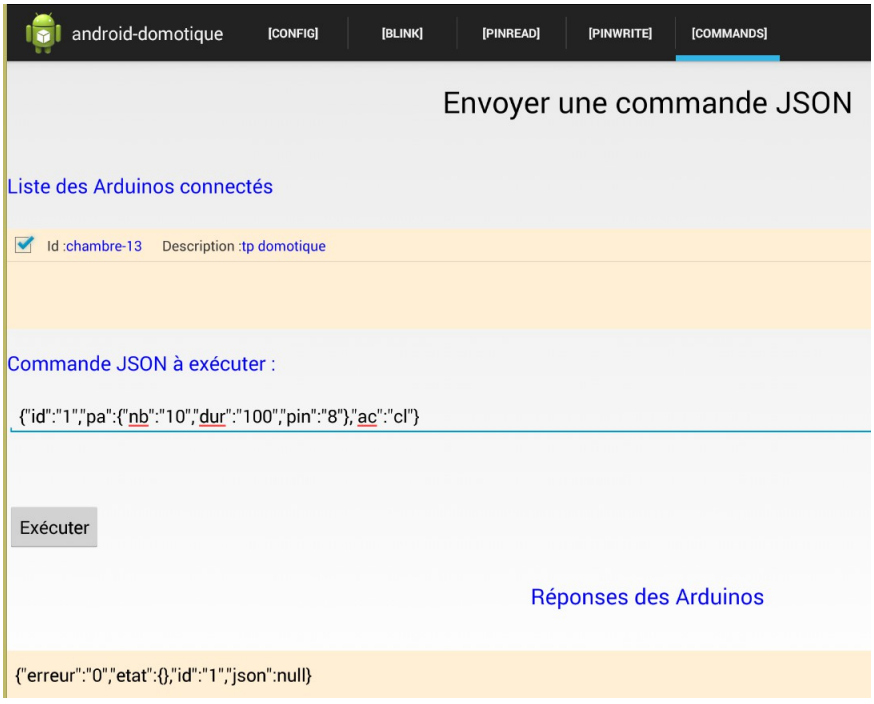
L'onglet [PINWRITE] permet d'écrire une valeur sur une pin d'un Arduino :



L'onglet [PINREAD] permet de lire la valeur d'une pin d'un Arduino :

L'onglet [BLINK] permet de faire clignoter une led d'un Arduino :

L'onglet [COMMAND] permet d'envoyer une commande jSON à un Arduino :

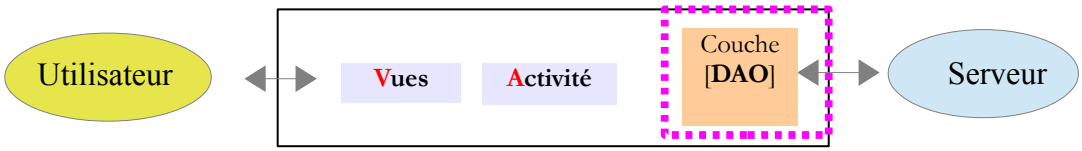


4.6 Le client Android du service web / jSON

Nous abordons maintenant l'écriture du client Android.

4.6.1 L'architecture du client

L'architecture du client Android sera celle du projet [exemple-10-client] de la partie 1 du cours :

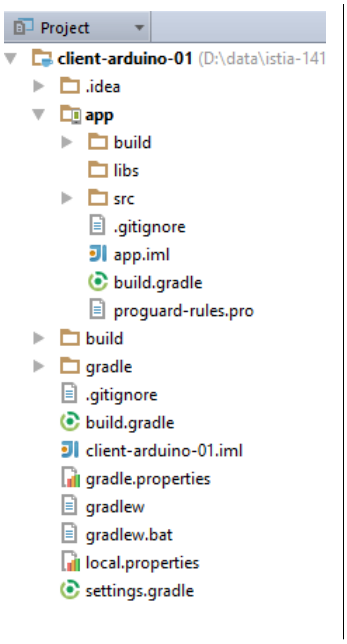


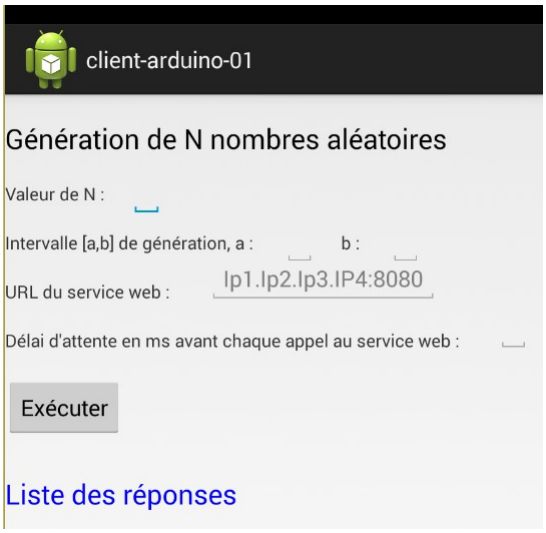
- la couche [DAO] communique avec le serveur web / jSON ;

Le client Android doit pouvoir commander plusieurs Arduinos simultanément. Par exemple, on veut pouvoir faire clignoter deux leds placées sur deux Arduinos, en même temps et non pas l'une après l'autre. Aussi notre client Android utilisera-t-il une tâche asynchrone par Arduino et ces tâches s'exécuteront en parallèle.

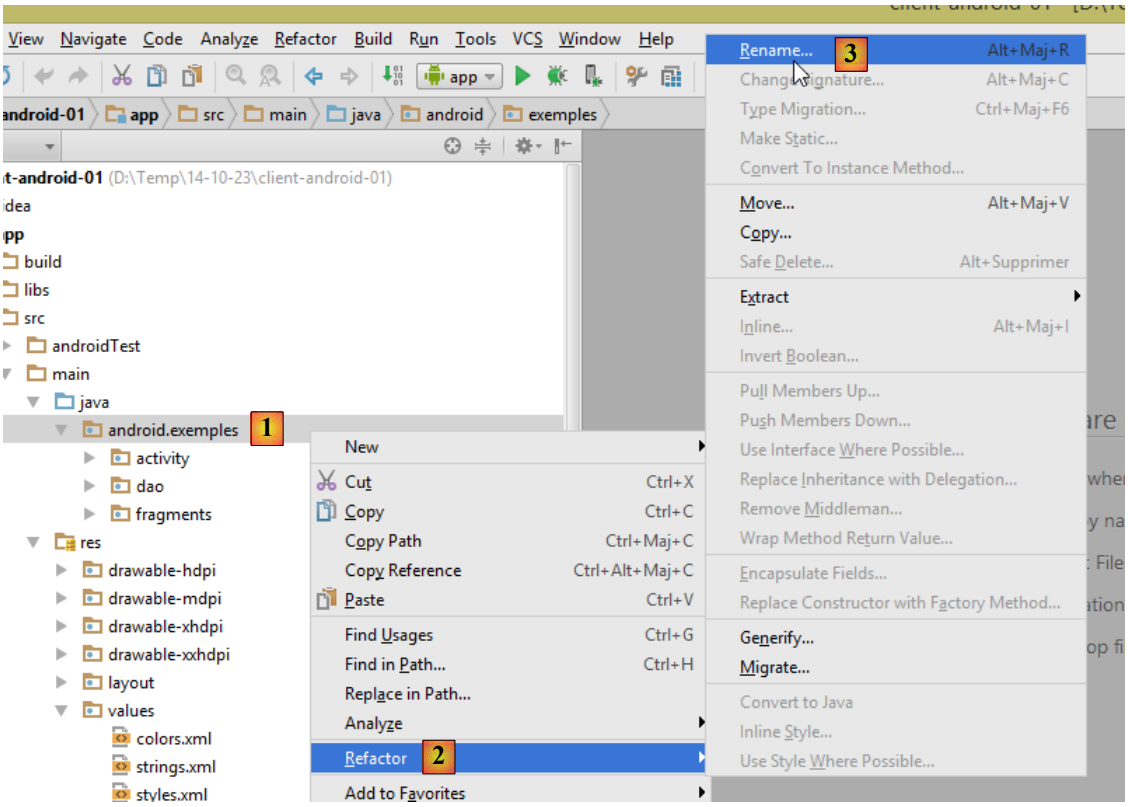
4.6.2 Le projet IntellijIDEA du client

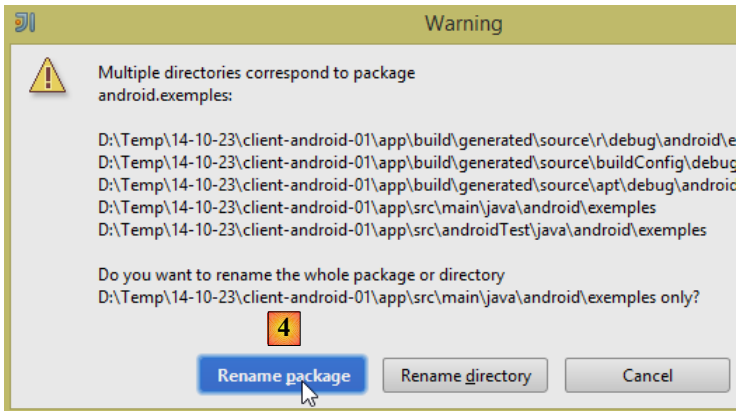
Dupliquez le projet [exemple-10-client] dans le projet [client-arduino-01] (si besoin est, revoyez comment dupliquer un projet Gradle au paragraphe 1.10, page 91) :

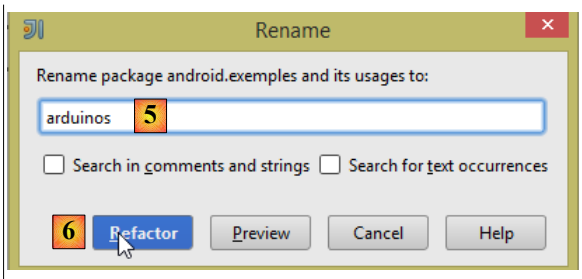


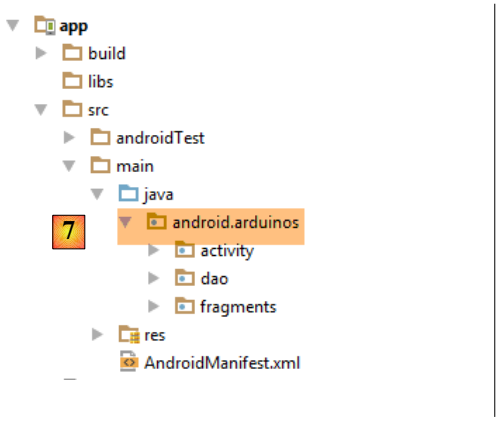


Renommez le package [android.exemples] en [android.arduinos] [1-7] :


View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
android-01 app src main java android exemples
t-android-01 (D:\Temp\14-10-23\client-android-01)
idea
pp
build
libs
src
androidTest
main
java
android.exemples
activity
dao
fragments
res
drawable-hdpi
drawable-mdpi
drawable-xhdpi
drawable-xxhdpi
layout
values
colors.xml
strings.xml
styles.xml
1
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Maj+C
Copy Reference Ctrl+Alt+Maj+C
Paste Ctrl+V
Find Usages Ctrl+G
Find in Path... Ctrl+H
Replace in Path...
Analyze
Refactor
2
Add to Favorites
Rename... 3 Alt+Maj+R
Change Signature... Alt+Maj+C
Type Migration... Ctrl+Maj+F6
Make Static...
Convert To Instance Method...
Move... Alt+Maj+V
Copy...
Safe Delete... Alt+Supprimer
Extract
Inline... Alt+Maj+I
Invert Boolean...
Pull Members Up...
Push Members Down...
Use Interface Where Possible...
Replace Inheritance with Delegation...
Remove Middleman...
Wrap Method Return Value...
Encapsulate Fields...
Replace Constructor with Factory Method...
Generify...
Migrate...
Convert to Java
Inline Style...
Use Style Where Possible...

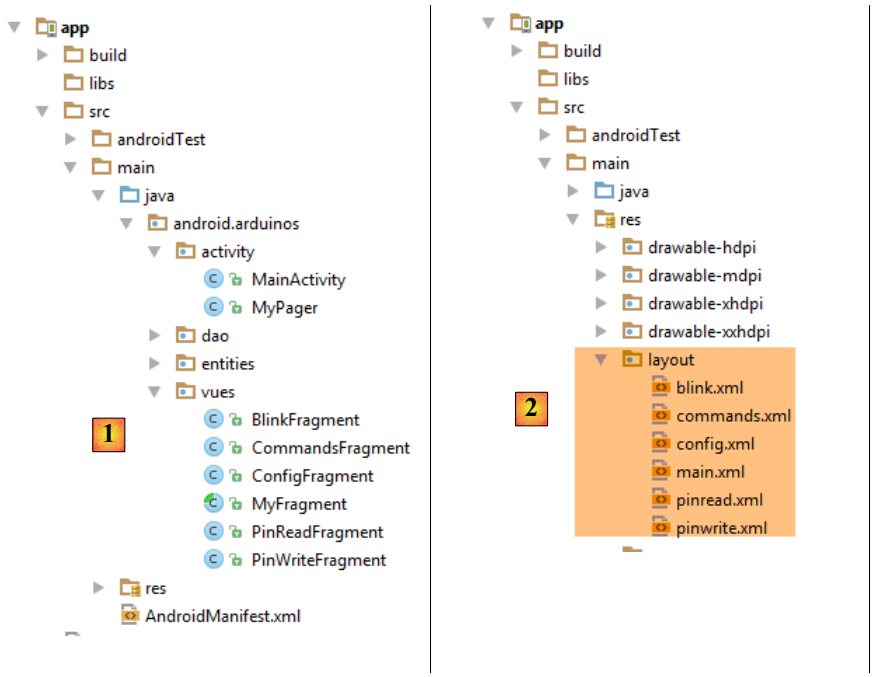

Warning
Multiple directories correspond to package android.exemples:
D:\Temp\14-10-23\client-android-01\app\build\generated\source\r\debug\android\e
D:\Temp\14-10-23\client-android-01\app\build\generated\source\buildConfig\debug
D:\Temp\14-10-23\client-android-01\app\build\generated\source\apt\debug\androic
D:\Temp\14-10-23\client-android-01\app\src\main\java\android\exemples
D:\Temp\14-10-23\client-android-01\app\src\androidTest\java\android\exemples
Do you want to rename the whole package or directory
D:\Temp\14-10-23\client-android-01\app\src\main\java\android\exemples only?
4
Rename package
Rename directory
Cancel


Rename
Rename package android.exemples and its usages to:
arduinos
5
Search in comments and strings
Search for text occurrences
6
Refactor
Preview
Cancel
Help


app
build
libs
src
androidTest
main
java
7
android.arduinos
activity
dao
fragments
res
AndroidManifest.xml

Recompilez le projet puis exécutez-le.

Le projet final ressemblera à ceci :



- en [1], les codes source du projet :
    - dans le package [activity], l'activité principale et le gestionnaire de vues,
    - dans le package [dao], la couche [DAO] ;
    - dans le package [entities], des objets utilisés par l'application,
    - dans le package [vues], les fragments associés aux vues XML que l'on voit en [2] ;

4.6.3 Les cinq vues XML

Il y a cinq vues XML :

- [blink] : pour faire clignoter une led d'un Arduino. Elle est associée au fragment [BlinkFragment] ;
- [commands] : pour envoyer une commande jSON à un Arduino. Elle est associée au fragment [CommandsFragment] ;
- [config] : pour configurer l'URL du service web / jSON et obtenir la liste initiale des Arduinos connectés. Elle est associée au fragment [ConfigFragment] ;
- [pinread] : pour lire la valeur binaire ou analogique d'une pin d'un Arduino. Elle est associée au fragment [PinReadFragment] ;
- [pinwrite] : pour écrire une valeur binaire ou analogique sur une pin d'un Arduino. Elle est associée au fragment [PinWriteFragment] ;

Pour le moment, ces cinq vues XML auront toutes le même contenu vide :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
3.              android:id="@+id/scrollView1"
4.              android:layout_width="wrap_content"
5.              android:layout_height="wrap_content">
6.
7.    <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
8.                    android:layout_width="match_parent"
9.                    android:layout_height="match_parent">
10.   </RelativeLayout>
11. </ScrollView>
```

- la vue est dans un conteneur [RelativeLayout] (lignes 7-10) lui-même inclus dans un conteneur [ScrollView] (lignes 2-11). Cela nous assure de pouvoir 'scroller' la vue si celle-ci dépasse la taille d'un écran de tablette ;

Travail : créez les vues XML après avoir supprimé celles du projet [exemple-10-client].

4.6.4 Les cinq fragments

Le fragment [ConfigFragment] associé à la vue XML [config] sera le suivant :

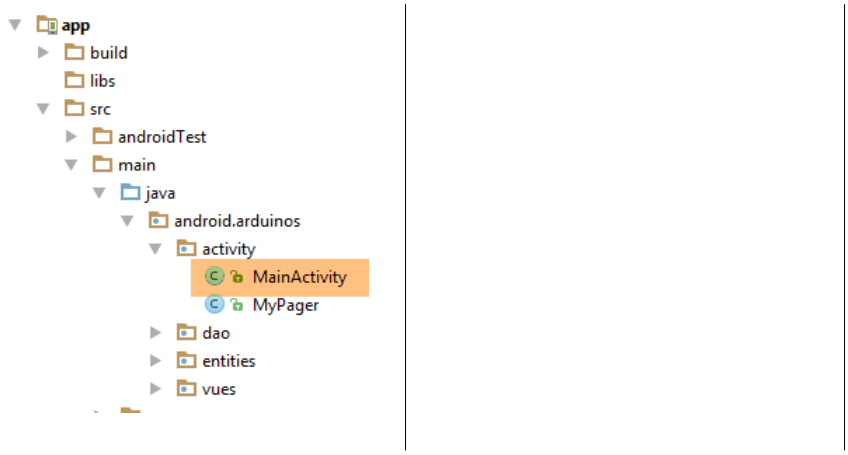
```
1.  package android.arduinos.vues;
2.
3.  import android.arduinos.R;
4.  import android.arduinos.activity.MainActivity;
5.  import org.androidannotations.annotations.*;
6.
7.  @EFragment(R.layout.config)
8.  public class ConfigFragment extends Fragment {
9.
10.   // l'activité
11.   private MainActivity activité;
12.
13.   @AfterViews
14.   public void initFragment() {
15.     // on note l'activité
16.     activité = (MainActivity) getActivity();
17.   }
18. }
```

On retrouve ici du code déjà rencontré dans divers projets. Les cinq fragments sont tous identiques sauf à la ligne 7, où la vue associée au fragment diffère :

Fragment	Vue
ConfigFragment	`R.layout.config`
PinReadFragment	`R.layout.pinread`
PinWriteFragment	`R.layout.pinwrite`
CommandsFragment	`R.layout.commands`
BlinkFragment	`R.layout.blink`

Travail : créez les cinq fragments précédents après avoir supprimé ceux du projet [exemple-10-client].

4.6.5 La classe [MainActivity]



Le code de la classe [MainActivity] est le suivant :

```
1.  package android.arduinos.activity;
2.
3.  import android.app.ActionBar;
4.  import android.app.ActionBar.Tab;
5.  import android.app.FragmentTransaction;
6.  import android.arduinos.R;
7.  import android.os.Bundle;
8.  import android.support.v4.app.Fragment;
```

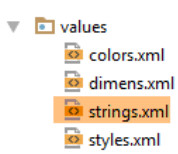
```
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.view.Window;
import android.arduinos.vues.*;
import org.androidannotations.annotations.EActivity;

import java.util.Locale;

@EActivity
public class MainActivity extends FragmentActivity implements ActionBar.TabListener {

  // le gestionnaire de fragments ou sections
  SectionsPagerAdapter mSectionsPagerAdapter;

  // le conteneur des fragments
  MyPager mViewPager;

  // les onglets
  private Tab[] tabs;

  // les constantes
  // ------------------------------------------------------
  // pause avant exécution d'une tâche asynchrone
  public final static int PAUSE = 0;
  // délai en ms d'attente maximale de la réponse du serveur
  private final static int TIMEOUT = 1000;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    // classique
    super.onCreate(savedInstanceState);
    // il faut installer le sablier avant de créer la vue
    requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
    // la vue
    setContentView(R.layout.main);

    // la barre d'onglets
    final ActionBar actionBar = getActionBar();
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

    // instanciation de notre gestionnaire de fragments
    mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());

    // on récupère la référence du conteneur de fragments
    mViewPager = (MyPager) findViewById(R.id.pager);
    // il est associé à notre gestionnaire de fragments
    mViewPager.setAdapter(mSectionsPagerAdapter);
    // on inhibe le swipe
    mViewPager.setSwipeEnabled(false);

    // on crée autant d'onglets qu'il y a de fragments affichés par le conteneur
    tabs = new Tab[mSectionsPagerAdapter.getCount()];
    for (int i = 0; i < tabs.length; i++) {
      // actionBar est la barre d'onglets
      // actionBar.newTab() crée un nouvel onglet
      // actionBar.newTab().setText() donne un titre à cet onglet
      // actionBar.newTab().setText().setTabListener(this) indique que cette
      // classe (le fragment) gère les évts des onglets
      tabs[i] = actionBar.newTab().setText(mSectionsPagerAdapter.getPageTitle(i)).setTabListener(this);
      actionBar.addTab(tabs[i]);
    }
  }

  public void onTabSelected(Tab tab, FragmentTransaction fragmentTransaction) {
    // un onglet a été sélectionné - on change le fragment affiché par le
    // conteneur de fragments
    int position = tab.getPosition();
    // on l'affiche
    mViewPager.setCurrentItem(position);
  }

  public void onTabUnselected(Tab tab, FragmentTransaction fragmentTransaction) {
  }

```

```
84.   public void onTabReselected(Tab tab, FragmentTransaction fragmentTransaction) {
85.   }
86.
87.   // notre gestionnaire de fragments
88.   // à redéfinir pour chaque application
89.   // doit définir les méthodes suivantes
90.   // getItem, getCount, getPageTitle
91.   public class SectionsPagerAdapter extends FragmentPagerAdapter {
92. ...
93.   }
94.
95. }
```

- ligne 18 : notre activité est de type [FragmentActivity]. Les cinq vues seront dans cinq onglets. Aussi l'activité implémente-elle l'interface [ActionBar.TabListener] (ligne 19). Ceci est fait aux lignes 73, 81 et 84. Reportez-vous au projet [exemple-06] de la partie 1 si vous avez oublié comment gérer des onglets ;
- ligne 91 : notre gestionnaire de fragments doit gérer cinq fragments. Son code est le suivant :

```
1.  public class SectionsPagerAdapter extends FragmentPagerAdapter {
2.
3.      // les fragments
4.      Fragment[] fragments = {new ConfigFragment_(), new BlinkFragment_(), new PinReadFragment_(), new
    PinWriteFragment_(), new CommandsFragment_()};
5.
6.      // constructeur
7.      public SectionsPagerAdapter(FragmentManager fm) {
8.        super(fm);
9.      }
10.
11.     // doit rendre le fragment n° i avec ses éventuels arguments
12.     @Override
13.     public Fragment getItem(int position) {
14.       // on rend le fragment
15.       return fragments[position];
16.     }
17.
18.     // rend le nombre de fragments à gérer
19.     @Override
20.     public int getCount() {
21.       return fragments.length;
22.     }
23.
24.     // rend le titre du fragment n° position
25.     @Override
26.     public CharSequence getPageTitle(int position) {
27.       Locale l = Locale.getDefault();
28.       switch (position) {
29.         case 0:
30.           return getString(R.string.config_titre).toUpperCase(l);
31.         case 1:
32.           return getString(R.string.blink_titre).toUpperCase(l);
33.         case 2:
34.           return getString(R.string.pinread_titre).toUpperCase(l);
35.         case 3:
36.           return getString(R.string.pinwrite_titre).toUpperCase(l);
37.         case 4:
38.           return getString(R.string.commands_titre).toUpperCase(l);
39.       }
40.       return null;
41.     }
42.   }
```

- les cinq fragments sont instanciés ligne 4 et leurs références mises dans un tableau. A cause des annotations AA, les classes des fragments sont celles présentées précédemment suffixées avec un underscore ;
- ligne 21 : il y a 5 fragments ;
- lignes 26-41 : on définit un titre pour chacun des fragments. Ces titres seront cherchés dans le fichier [res / values / strings.xml]



Le contenu de [strings.xml] est le suivant :

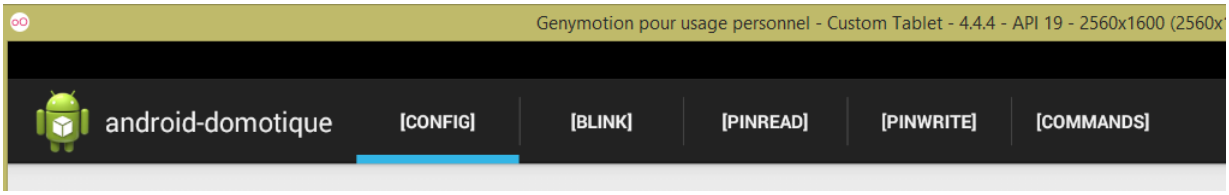
```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <!-- général -->
    <string name="app_name">android-domotique</string>

    <!-- Fragments et onglets -->
    <string name="config_titre">[Config]</string>
    <string name="blink_titre">[Blink]</string>
    <string name="pinread_titre">[PinRead]</string>
    <string name="pinwrite_titre">[PinWrite]</string>
    <string name="commands_titre">[Commands]</string>

</resources>
```

Ces titres de fragments seront également les titres des onglets grâce au code suivant de la méthode [onCreate] de la classe [MainActivity] :

```
        // on crée autant d'onglets qu'il y a de fragments affichés par le conteneur
        for (int i = 0; i < mSectionsPagerAdapter.getCount(); i++) {

    actionBar.addTab(actionBar.newTab().setText(mSectionsPagerAdapter.getPageTitle(i)).setTabListener(this));
    }
```

Travail : créez les éléments précédents.

Exécutez le projet. Vous devez obtenir la vue suivante sur la tablette :



4.6.6 La vue XML [config]

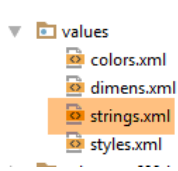
La vue XML [config] sera la suivante :

La vue ci-dessus est obtenue avec le code XML suivant :

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/RelativeLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/txt_TitreConfig"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:text="@string/txt_TitreConfig"
        android:textSize="@dimen/titre" />

    <TextView
        android:id="@+id/txt_UrlServiceRest"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/txt_TitreConfig"
        android:layout_marginTop="50dp"
        android:text="@string/txt_UrlServiceRest"
        android:textSize="20sp" />

    <EditText
        android:id="@+id/edt_UrlServiceRest"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/txt_UrlServiceRest"
        android:layout_alignBottom="@+id/txt_UrlServiceRest"
        android:layout_marginLeft="20dp"
        android:layout_toRightOf="@+id/txt_UrlServiceRest"
        android:ems="10"
        android:hint="@string/hint_UrlServiceRest"
        android:inputType="textUri" >

        <requestFocus />
    </EditText>

    <TextView
        android:id="@+id/txt_MsgErreurIpPort"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
47.         android:layout_alignParentLeft="true"
48.         android:layout_below="@+id/txt_UrlServiceRest"
49.         android:layout_marginTop="20dp"
50.         android:text="@string/txt_MsgErreurUrlServiceRest"
51.         android:textColor="@color/red"
52.         android:textSize="20sp" />
53.
54.     <TextView
55.         android:id="@+id/txt_arduinos"
56.         android:layout_width="wrap_content"
57.         android:layout_height="wrap_content"
58.         android:layout_alignParentLeft="true"
59.         android:layout_below="@+id/txt_MsgErreurIpPort"
60.         android:layout_marginTop="40dp"
61.         android:text="@string/titre_list_arduinos"
62.         android:textColor="@color/blue"
63.         android:textSize="20sp" />
64.
65.     <Button
66.         android:id="@+id/btn_Rafraichir"
67.         android:layout_width="wrap_content"
68.         android:layout_height="wrap_content"
69.         android:layout_alignBaseline="@+id/txt_arduinos"
70.         android:layout_alignBottom="@+id/txt_arduinos"
71.         android:layout_marginLeft="20dp"
72.         android:layout_toRightOf="@+id/txt_arduinos"
73.         android:text="@string/btn_rafraichir" />
74.
75.     <Button
76.         android:id="@+id/btn_Annuler"
77.         android:layout_width="wrap_content"
78.         android:layout_height="wrap_content"
79.         android:layout_alignBaseline="@+id/txt_arduinos"
80.         android:layout_alignBottom="@+id/txt_arduinos"
81.         android:layout_marginLeft="20dp"
82.         android:layout_toRightOf="@+id/txt_arduinos"
83.         android:text="@string/btn_annuler"
84.         android:visibility="invisible" />
85.
86.     <ListView
87.         android:id="@+id/ListViewArduinos"
88.         android:layout_width="match_parent"
89.         android:layout_height="200dp"
90.         android:layout_alignParentLeft="true"
91.         android:layout_below="@+id/txt_arduinos"
92.         android:layout_marginTop="30dp"
93.         android:background="@color/wheat">
94.     </ListView>
95.
96. </RelativeLayout>
```

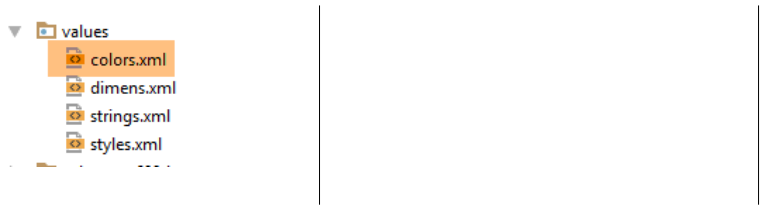
La vue utilise des chaînes de caractères (*android:text* aux lignes 15, 25, 37, 50, 61, 73) qui sont définies dans le fichier [res / values / strings] :




```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.
4.      <string name="app_name">android-domotique</string>
5.
6.      <!-- Fragments et onglets -->
7.      <string name="config_titre">[Config]</string>
8.      <string name="blink_titre">[Blink]</string>
9.      <string name="pinread_titre">[PinRead]</string>
```

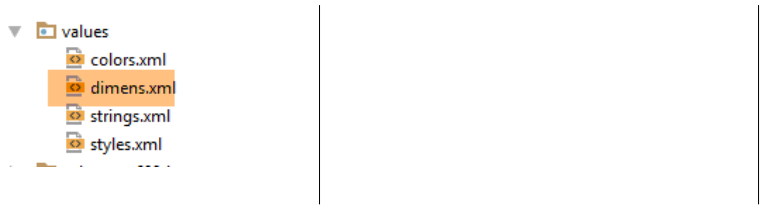
```
10.     <string name="pinwrite_titre">[PinWrite]</string>
11.     <string name="commands_titre">[Commands]</string>
12.
13.     <!-- Config -->
14.     <string name="txt_TitreConfig">Se connecter au serveur</string>
15.     <string name="txt_UrlServiceRest">Url du service web / jSON</string>
16.     <string name="txt_MsgErreurUrlServiceRest">L\'Url du service doit être entrée sous la forme
    Ip1.Ip2.Ip3.IP4:Port/contexte</string>
17.     <string name="hint_UrlServiceRest">ex (192.168.1.120:8080/rest)</string>
18.     <string name="btn_annuler">Annuler</string>
19.     <string name="btn_rafraichir">Rafraîchir</string>
20.     <string name="titre_list_arduinos">Liste des Arduinos connectés</string>
21.
22. </resources>
```

La vue utilise des couleurs (android:textColor aux lignes 51 et 62] définies dans le fichier [res / values / colors] :




```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.
4.      <color name="red">#FF0000</color>
5.      <color name="blue">#0000FF</color>
6.      <color name="wheat">#FFEFD5</color>
7.      <color name="floral_white">#FFFAF0</color>
8.
9.  </resources>
```

La vue utilise des dimensions (*android:textSize* à la ligne 16) qui sont définies dans le fichier [res / values / dimens] :

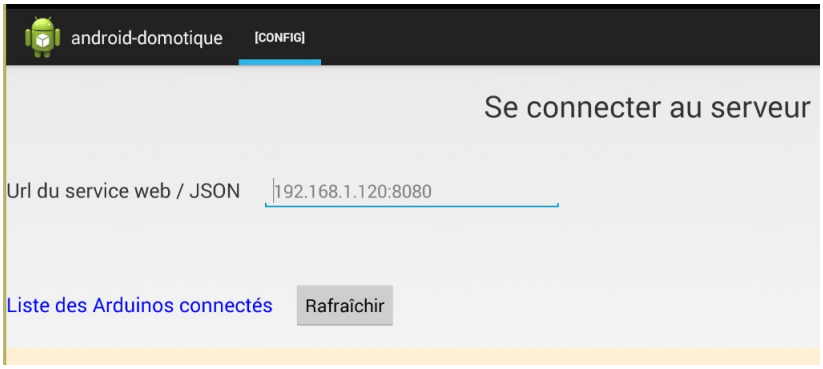



```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.      <dimen name="titre">30dp</dimen>
4.  </resources>
```

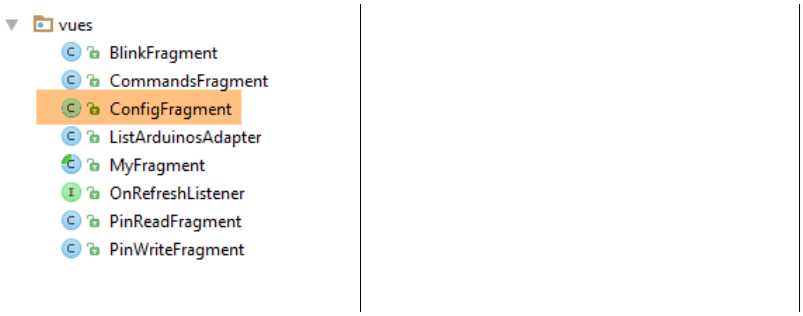
Cette technique n'a pas été utilisée pour toutes les dimensions. C'est cependant celle qui est conseillée. Elle permet de changer des dimensions en un seul endroit.

Travail : créez les éléments précédents.

Exécutez de nouveau votre projet. Vous devez obtenir la vue suivante :



4.6.7 Le fragment [ConfigFragment]



Pour gérer la nouvelle vue [config], le code du fragment [ConfigFragment] évolue de la façon suivante :

```
1.  package android.arduinos.vues;
2.
3.  import android.arduinos.R;
4.  import android.support.v4.app.Fragment;
5.  import android.view.View;
6.  import android.widget.Button;
7.  import android.widget.EditText;
8.  import android.widget.ListView;
9.  import android.widget.TextView;
10. import android.arduinos.activity.MainActivity;
11. import android.arduinos.dao.ArduinosResponse;
12. import android.arduinos.entities.Arduino;
13. import android.arduinos.entities.CheckedArduino;
14. import org.androidannotations.annotations.*;
15. import org.androidannotations.api.BackgroundExecutor;
16.
17. import java.net.URI;
18. import java.util.ArrayList;
19. import java.util.List;
20.
21. @EFragment(R.layout.config)
22. public class ConfigFragment extends Fragment {
23.
24.   // les éléments de l'interface visuelle
25.   @ViewById(R.id.btn_Rafraichir)
26.   Button btnRafraichir;
27.   @ViewById(R.id.btn_Annuler)
28.   Button btnAnnuler;
29.   @ViewById(R.id.edt_UrlServiceRest)
30.   EditText edtUrlServiceRest;
31.   @ViewById(R.id.txt_MsgErreurIpPort)
32.   TextView txtMsgErreurUrlServiceRest;
33.   @ViewById(R.id.ListViewArduinos)
34.   ListView listArduinos;
35.
36.   // l'activité
37.   private MainActivity activité;
```

```
38.   // les valeurs saisies
39.   private String urlServiceRest;
40.   // le nombre d'informations reçues
41.   private int nbInfosAttendues;
42.
43.   @AfterViews
44.   public void initFragment() {
45.     // on note l'activité
46.     activité = (MainActivity) getActivity();
47.     // visibilité boutons
48.     btnRafraichir.setVisibility(View.VISIBLE);
49.     btnAnnuler.setVisibility(View.INVISIBLE);
50.     // le msg d'erreur éventuel
51.     txtMsgErreurUrlServiceRest.setText("");
52.   }
53.
54.   @Click(R.id.btn_Rafraichir)
55.   protected void doRafraichir() {
56.     // on commence l'attente
57.     beginWaiting();
58.   }
59.
60.   @Click(R.id.btn_Annuler)
61.   void doAnnuler() {
62.     // on annule l'attente
63.     cancelWaiting();
64.   }
65.
66.   protected void cancelWaiting() {
67.     // le bouton [Exécuter] remplace le bouton [Annuler]
68.     btnAnnuler.setVisibility(View.INVISIBLE);
69.     btnRafraichir.setVisibility(View.VISIBLE);
70.     // on enlève le sablier
71.     activité.setProgressBarIndeterminateVisibility(false);
72.     // plus de tâches à attendre
73.     nbInfosAttendues = 0;
74.   }
75.
76.   private void beginWaiting() {
77.     // le bouton [Annuler] remplace le bouton [Exécuter]
78.     btnRafraichir.setVisibility(View.INVISIBLE);
79.     btnAnnuler.setVisibility(View.VISIBLE);
80.     // on met le sablier
81.     activité.setProgressBarIndeterminateVisibility(true);
82.   }
83. }
```

- lignes 25-34 : les éléments de l'interface visuelle ;
- ligne 37 : une référence sur l'activité unique du projet ;
- ligne 39 : le formulaire ne fait qu'une saisie, celle de l'URL du service web / jSON ;
- ligne 41 : comptage des informations reçues par le fragment ;
- ligne 44 : la méthode [initFragment] initialise le fragment après que les références sur les composants de l'interface visuelle aient été obtenues [@AfterViews] ;
- lignes 54-55 : la méthode exécutée lors d'un 'clic' sur le bouton [Rafraîchir] ;
- lignes 60-61 : la méthode exécutée lors d'un 'clic' sur le bouton [Annuler] ;
- lignes 66-74 : la méthode d'annulation de l'affichage du widget d'attente ;
- lignes 76-82 : la méthode d'affichage du widget d'attente ;

Travail : créez les éléments précédents.

Exécutez cette nouvelle version. Vérifiez que le bouton [Rafraîchir] lance l'attente et que le bouton [Annuler] l'arrête.

4.6.8 Vérification des saisies

Dans la version précédente, nous ne vérifions pas la validité de l'URL saisie. Pour la vérifier, nous ajoutons le code suivant dans [ConfigFragment] :

```
1.  package android.arduinos.vues;
2.
3.  import android.arduinos.R;
4.  ...
```

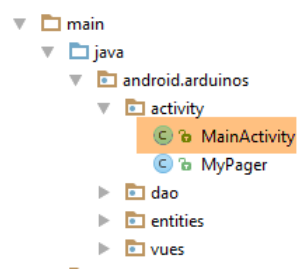
```
5.
6.  @EFragment(R.layout.config)
7.  public class ConfigFragment extends Fragment {
8.
9.  ...
10.   // les valeurs saisies
11.   private String urlServiceRest;
12.   // le nombre d'informations reçues
13.   private int nbInfosAttendues;
14.
15.   @Click(R.id.btn_Rafraichir)
16.   protected void doRafraichir() {
17.     // on vérifie les saisies
18.     if (!pageValid()) {
19.       return;
20.     }
21.     // on commence l'attente
22.     beginWaiting();
23.   }
24.
25.
26.   // vérification des saisies
27.   private boolean pageValid() {
28.     // on récupère l'Ip et le port du serveur
29.     urlServiceRest = String.format("http://%s", edtUrlServiceRest.getText().toString().trim());
30.     // on vérifie sa validité
31.     try {
32.       URI uri = new URI(urlServiceRest);
33.       String host = uri.getHost();
34.       int port = uri.getPort();
35.       if (host == null || port == -1) {
36.         throw new Exception();
37.       }
38.     } catch (Exception ex) {
39.       // affichage msg d'erreur
40.       txtMsgErreurUrlServiceRest.setText(getResources().getString(R.string.txt_MsgErreurUrlServiceRest));
41.       // retour à l'UI
42.       return false;
43.     }
44.     // c'est bon
45.     txtMsgErreurUrlServiceRest.setText("");
46.     return true;
47.   }
48.
49. }
```

- ligne 11 : l'URL saisie ;
- lignes 18-20 : avant de faire quoique ce soit, on vérifie la validité des saisies ;
- ligne 29 : on récupère l'URL saisie et on lui ajoute le préfixe [http://] ;
- ligne 32 : on essaie de construire un objet URI (**U**niform **R**esource **I**dentifier) avec. Si l'URL saisie est syntaxiquement incorrecte, on aura une exception ;
- lignes 33-37 : on crée une exception si l'URI est correcte mais qu'on a cependant [host==null] et [port==-1]. C'est un cas possible ;
- ligne 40 : on a eu une exception. On affiche un message d'erreur. Celui-ci est pris dans le fichier [res / values / strings]. La méthode [*getResources*] donne accès aux fichiers XML du dossier [res]. *[getResources].getString* donne accès au fichier XML [res / values / strings]. Son paramètre est l'id de la chaîne désirée ;
- ligne 42 : on retourne [false] pour indiquer que la page est invalide ;
- ligne 45 : on n'a pas eu d'erreur. On efface le message d'erreur qui peut être affiché depuis une précédente erreur ;
- ligne 46 : on retourne [true] pour indiquer que la page est valide ;

Travail : créez les éléments précédents.

Testez cette nouvelle version et vérifiez que les URL invalides sont bien signalées.

4.6.9 Gestion des onglets



Pour l'instant, les cinq onglets sont visibles. Nous introduisons une méthode dans la classe [MainActivity] pour les gérer :

```
1.  // les onglets
2.    private Tab[] tabs;
3.  ...
4.
5.    @Override
6.    protected void onCreate(Bundle savedInstanceState) {
7.      // classique
8.      super.onCreate(savedInstanceState);
9.      // il faut installer le sablier avant de créer la vue
10.     requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
11.     // la vue
12.     setContentView(R.layout.main);
13.
14.     // la barre d'onglets
15.     final ActionBar actionBar = getActionBar();
16.     actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
17.
18.     // instanciation de notre gestionnaire de fragments
19.     mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
20.
21.     // on récupère la référence du conteneur de fragments
22.     mViewPager = (MyPager) findViewById(R.id.pager);
23.     // il est associé à notre gestionnaire de fragments
24.     mViewPager.setAdapter(mSectionsPagerAdapter);
25.     // on inhibe le swipe
26.     mViewPager.setSwipeEnabled(false);
27.
28.     // on crée autant d'onglets qu'il y a de fragments affichés par le conteneur
29.     tabs = new Tab[mSectionsPagerAdapter.getCount()];
30.     for (int i = 0; i < tabs.length; i++) {
31.       tabs[i] = actionBar.newTab().setText(mSectionsPagerAdapter.getPageTitle(i)).setTabListener(this);
32.       actionBar.addTab(tabs[i]);
33.     }
34.     // on n'affiche que le 1er onglet
35.     showTabs(new Boolean[]{true, false, false, false, false});
36.   }
37.
38.
39. }
```

- lignes 29-33 : les cinq onglets sont créés et mémorisés dans le tableau de la ligne 2 ;
- ligne 35 : la méthode [showTabs] est utilisée pour afficher (true) ou cacher (false) les cinq onglets.

La méthode [showTabs] est la suivante :

```
1.  // gestion des onglets
2.    public void showTabs(Boolean[] show) {
3.      // si show[i] est vrai, affiche l'onglet n° i
4.      final ActionBar actionBar = getActionBar();
5.      // on passe tous les onglets en revue
6.      // en commençant par la fin
7.      for (int i = 0; i < show.length; i++) {
8.        // onglet n° i
9.        Tab tab = tabs[i];
10.       int position = tab.getPosition();
11.       if (show[i]) {
12.         // la vue doit être affichée si elle ne l'est pas déjà
```

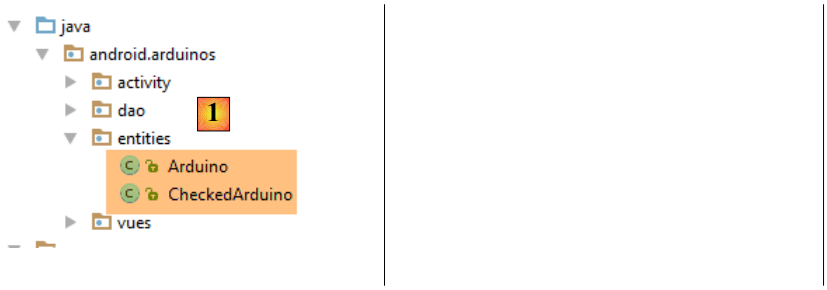
```
13.         if (position == Tab.INVALID_POSITION) {
14.           actionBar.addTab(tab);
15.         }
16.       } else {
17.         // la vue doit être enlevée si elle ne l'est pas déjà
18.         if (position != Tab.INVALID_POSITION) {
19.           actionBar.removeTab(tab);
20.         }
21.       }
22.     }
23.   }
```

Notez les points suivants :

- ligne 4 : on récupère la barre d'actions dans laquelle se trouvent les cinq onglets ;
- ligne 10 : pour récupérer la position d'un onglet. On récupère [Tab.*INVALID_POSITION*] si l'onglet n'est pas affiché ;
- lignes 11-15 : si l'onglet n'est pas affiché (position) et qu'il doit l'être (show[i]) alors on l'ajoute (ligne 14) à la barre d'onglets ;
- lignes 18-20 : si l'onglet est affiché (position) et qu'il ne doit pas l'être (show[i]) alors on l'enlève (ligne 19) de la barre d'onglets ;

Faites ces modifications et testez de nouveau votre application. Vérifiez que seul l'onglet [Config] est désormais affiché lors du démarrage de l'application.

4.6.10 Affichage de la liste des Arduinos



Les différentes vues vont avoir besoin d'afficher la liste des Arduinos connectés. Pour cela, nous allons définir différentes classes et une vue XML :

- un Arduino sera représenté par la classe [Arduino] [1] ;
- la classe [CheckedArduino] [1] est héritée de la classe [Arduino] à laquelle on a ajouté un booléen pour savoir si l'Arduino a été sélectionné ou non dans une liste ;

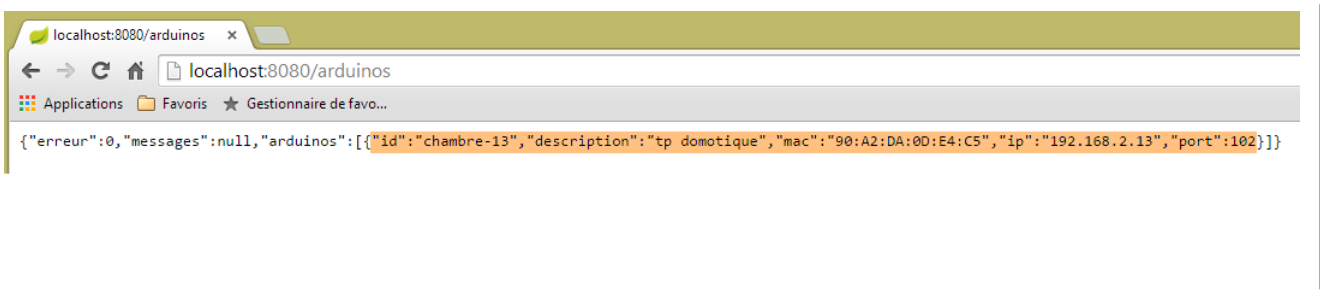
La classe [Arduino] est celle déjà utilisée par le serveur et présentée page 291. C'est la suivante :

```
16. package android.arduinos.entities;
17.
18. import java.io.Serializable;
19.
20. public class Arduino implements Serializable {
21.   // données
22.   private String id;
23.   private String description;
24.   private String mac;
25.   private String ip;
26.   private int port;
27.
28.  // getters et setters
29. ...
30. }
```

- ligne 7 : [id] est l'identifiant de l'arduino ;
- ligne 8 : sa description ;
- ligne 9 : son adresse MAC ;
- ligne 10 : son adresse IP ;

- ligne 11 : le port sur lequel il attend des commandes ;

Cette classe correspond à la chaîne jSON reçue du serveur lorsqu'on lui demande la liste des Arduinos connectés :



La classe [CheckedArduino] hérite de la classe [Arduino] :

```
1.  package android.arduinos.entities;
2.
3.  public class CheckedArduino extends Arduino {
4.     private static final long serialVersionUID = 1L;
5.     // un Arduino peut être sélectionné
6.     private boolean isChecked;
7.
8.     // constructeur
9.     public CheckedArduino(Arduino arduino, boolean isChecked) {
10.       // parent
11.       super(arduino.getId(), arduino.getDescription(), arduino.getMac(), arduino.getIp(),
    arduino.getPort());
12.       // local
13.       this.isChecked = isChecked;
14.    }
15.
16.    // getters et setters
17.    public boolean isChecked() {
18.       return isChecked;
19.    }
20.
21.    public void setChecked(boolean isChecked) {
22.       this.isChecked = isChecked;
23.    }
24.
25. }
```

- ligne 3 : la classe [CheckedArduino] hérite de la classe [Arduino] ;
- ligne 6 : on lui ajoute un booléen qui nous servira à savoir si dans la liste des Arduinos affichée, un Arduino a été sélectionné ou non ;

Dans [ConfigFragment], nous allons simuler l'obtention de la liste des Arduinos connectés.


```
1.    @Click(R.id.btn_Rafraichir)
2.    protected void doRafraichir() {
3.      // on efface la liste actuelle des Arduinos
4.      clearArduinos();
5.      // on vérifie les saisies
6.      if (!pageValid()) {
7.        return;
```

```
8.      }
9.      // on demande la liste des Arduinos
10.     nbInfosAttendues = 1;
11.     getArduinosInBackground();
12.     // on commence l'attente
13.     beginWaiting();
14.   }
15.
16.   @Background(id = "getArduinos", delay = MainActivity.PAUSE)
17.   void getArduinosInBackground() {
18.   ...
19.   }
20.
21.   // affichage réponse
22.   @UiThread
23.   void showResponse(ArduinosResponse response) {
24.   ....
25.   }
26.
27.   // raz liste des Arduinos
28.   private void clearArduinos() {
29.     // on crée une liste vide
30.     List<String> strings = new ArrayList<String>();
31.     // on l'affiche
32.     listArduinos.setAdapter(new ArrayAdapter<String>(activité, android.R.layout.simple_list_item_1,
    android.R.id.text1, strings));
33. }
```

- ligne 2 : la méthode qui demande la liste des Arduinos connectés ;
- ligne 4 : on efface la liste des Arduinos actuellement affichée ;
- ligne 11 : on demande en tâche de fond la liste des Arduinos connectés ;
- ligne 10 : on attend une unique réponse jSON de la part du serveur ;
- ligne 28 : la méthode qui efface la liste des Arduinos actuellement affichée ;

La méthode [getArduinosInBackground] est la suivante :

```
1.    @Background(id = "getArduinos", delay = MainActivity.PAUSE)
2.    void getArduinosInBackground() {
3.      // on crée une liste d'arduinos fictive
4.      List<Arduino> arduinos = new ArrayList<Arduino>();
5.      for (int i = 0; i < 20; i++) {
6.        arduinos.add(new Arduino("id" + i, "desc" + i, "mac" + i, "ip" + i, i));
7.      }
8.      ArduinosResponse response = new ArduinosResponse();
9.      response.setArduinos(arduinos);
10.     response.setErreur(0);
11.     showResponse(response);
12. }
```

- ligne 1 : l'annotation [@Background] indique que la méthode [getArduinosInBackground] s'exécute dans un thread différent de celui de l'UI ;
- ligne 1 : l'attribut [id = "getArduinos"] va nous permettre d'annuler la tâche via son [id] ;
- ligne 1 : l'attribut [delay = MainActivity.PAUSE] crée une pause avant le lancement de la tâche. Cela va nous permettre d'observer le bouton [Annuler]. La constante [PAUSE] est définie dans la classe [MainActivity] :

```
// pause avant exécution d'une tâche asynchrone
public final static int PAUSE = 5000;
```

- lignes 4-7 : on crée une liste de 20 Arduinos ;
- lignes 8-10 : on construit la réponse de type [ArduinosResponse] (défini page 290) qui va encapsuler la liste des Arduinos créée ;
- ligne 11 : on affiche cette réponse ;

La méthode [showResponse] est la suivante :

```
1.    @UiThread
2.    void showResponse(ArduinosResponse response) {
3.      // on teste la nature de l'information reçue
4.      int erreur = response.getErreur();
5.      if (erreur == 0) {
```

```
6.        // on récupère la liste d'Arduinos
7.        List<Arduino> arduinos = response.getArduinos();
8.        // on en fait une liste de [CheckedArduino]
9.        List<CheckedArduino> checkedArduinos = new ArrayList<CheckedArduino>();
10.       for (Arduino arduino : arduinos) {
11.         checkedArduinos.add(new CheckedArduino(arduino, false));
12.       }
13.       // on affiche les Arduinos
14.       showArduinos(checkedArduinos);
15.       // on affiche tous les onglets
16.       activité.showTabs(new Boolean[]{true, true, true, true, true});
17.     }
18.     // erreur ?
19.     if (erreur != 0) {
20.       // on affiche les msg d'erreur
21.       showMessages(activité, response.getMessages());
22.       // on n'affiche que l'onglet [Config]
23.       activité.showTabs(new Boolean[]{true, false, false, false, false});
24.     }
25.     // attente finie ?
26.     nbInfosAttendues--;
27.     if (nbInfosAttendues == 0) {
28.       cancelWaiting();
29.     }
30.   }
31.
```

- ligne 1 : l'affichage de la réponse doit se faire dans le thread de l'UI. C'est obligatoire ;
- lignes 5-17 : s'il n'y a pas eu d'erreur, on affiche la liste des Arduinos reçue ;

La méthode [showArduinos] est la suivante :

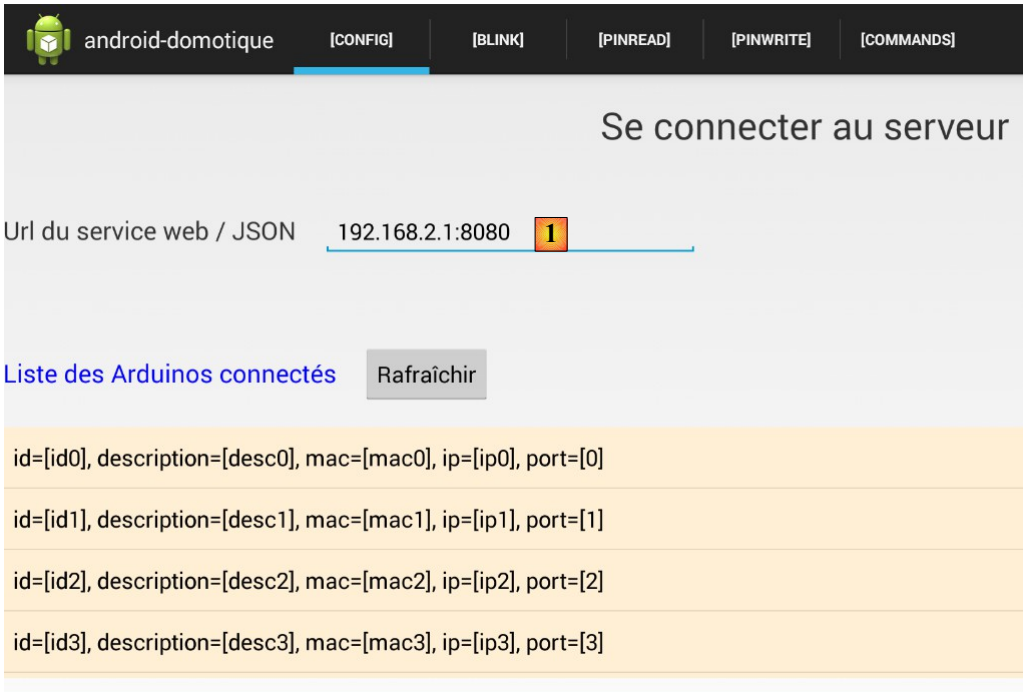
```
1.    private void showArduinos(List<CheckedArduino> checkedArduinos) {
2.      // on crée une liste de String à partir la liste des Arduinos
3.      List<String> strings = new ArrayList<String>();
4.      for (CheckedArduino checkedArduino : checkedArduinos) {
5.        strings.add(checkedArduino.toString());
6.      }
7.      // on l'affiche
8.      listArduinos.setAdapter(new ArrayAdapter<String>(activité, android.R.layout.simple_list_item_1,
    android.R.id.text1, strings));
9.  }
```

En cas d'erreur (ligne 19), on affiche les messages d'erreur contenus dans la réponse reçue du serveur avec la méthode [showMessages] suivante :

```
1.    protected void showMessages(MainActivity activité, List<String> messages) {
2.      // on construit le texte à afficher
3.      StringBuilder texte = new StringBuilder();
4.      for (String message : messages) {
5.        texte.append(String.format("%s\n", message));
6.      }
7.      // on l'affiche
8.      new AlertDialog.Builder(activité).setTitle("De erreurs se sont
    produites").setMessage(texte).setNeutralButton("Fermer", null).show();
9.  }
```

Travail : faites les modifications précédentes et exécutez votre projet.

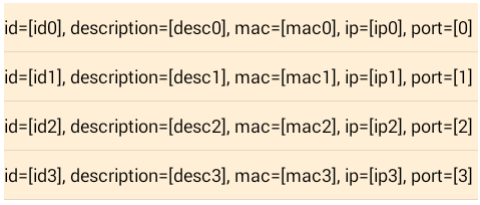
Vous devez obtenir la vue suivante :



La saisie en [1] n'est pas utilisée. Vous pouvez donc mettre n'importe quoi tant que ça respecte le format attendu.

4.6.11 Un patron pour afficher un Arduino

Pour l'instant, les Arduinos connectés sont affichés dans la vue [Config] de la façon suivante :



On veut désormais les afficher de la façon suivante :

- en [1], une case à cocher qui permettra de sélectionner un Arduino. Cette case à cocher sera cachée lorsqu'on voudra présenter une liste d'Arduinos non sélectionnables ;
- en [2], l'id de l'Arduino ;
- en [3], sa description ;

Ce qui suit reprend des concepts développés dans les projets [exemple-13] et [exemple-13B] de la partie 1. Revoyez-les si besoin est.

Nous créons tout d'abord la vue qui va afficher un élément de la liste des Arduinos :

Le code de la vue [listarduinos_item] ci-dessus est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:id="@+id/RelativeLayout1"
4.      android:layout_width="match_parent"
5.      android:layout_height="match_parent"
6.      android:background="@color/wheat"
7.      android:orientation="vertical" >
8.
9.      <CheckBox
10.         android:id="@+id/checkBoxArduino"
11.         android:layout_width="wrap_content"
12.         android:layout_height="wrap_content"
13.         android:layout_alignParentLeft="true"
14.         android:layout_alignParentTop="true"
15.         android:layout_toRightOf="@+id/txt_arduino_description" />
16.
17.     <TextView
18.         android:id="@+id/TextView1"
19.         android:layout_width="wrap_content"
20.         android:layout_height="wrap_content"
21.         android:layout_alignBaseline="@+id/checkBoxArduino"
22.         android:layout_marginLeft="40dp"
23.         android:text="@string/txt_arduino_id" />
24.
25.     <TextView
26.         android:id="@+id/txt_arduino_id"
27.         android:layout_width="wrap_content"
28.         android:layout_height="wrap_content"
29.         android:layout_alignBaseline="@+id/checkBoxArduino"
30.         android:layout_alignParentTop="true"
31.         android:layout_toRightOf="@+id/TextView1"
32.         android:text="@string/dummy"
33.         android:textColor="@color/blue" />
34.
35.     <TextView
36.         android:id="@+id/TextView2"
37.         android:layout_width="wrap_content"
38.         android:layout_height="wrap_content"
39.         android:layout_alignBaseline="@+id/checkBoxArduino"
40.         android:layout_alignParentTop="true"
41.         android:layout_marginLeft="20dp"
42.         android:layout_toRightOf="@+id/txt_arduino_id"
43.         android:text="@string/txt_arduino_description" />
44.
45.     <TextView
46.         android:id="@+id/txt_arduino_description"
47.         android:layout_width="wrap_content"
48.         android:layout_height="wrap_content"
49.         android:layout_alignBaseline="@+id/checkBoxArduino"
50.         android:layout_alignTop="@+id/TextView2"
51.         android:layout_toRightOf="@+id/TextView2"
52.         android:text="@string/dummy"
53.         android:textColor="@color/blue" />
54.
55. </RelativeLayout>
```

- lignes 9-15 : la case à cocher ;
- lignes 17-23 : le texte [Id :] ;

- lignes 25-33 : l'id de l'Arduino sera inscrit ici ;
- lignes 35-43 : le texte [Description :] ;
- lignes 45-53 : la description de l'Arduino sera inscrite ici ;

Cette vue utilise des textes (lignes 23, 32, 43) définis dans [res / values / strings.xml] :

```
1.      <string name="dummy">XXXXX</string>
2.
3.      <!--  listarduinos_item -->
4.      <string name="txt_arduino_id">Id : </string>
5.  <string name="txt_arduino_description">Description : </string>
```

La vue utilise également une couleur (lignes 33, 53) définie dans [res / values / colors.xml] :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <resources>
3.
4.      <color name="red">#FF0000</color>
5.      <color name="blue">#0000FF</color>
6.      <color name="wheat">#FFEFD5</color>
7.      <color name="floral_white">#FFFAF0</color>
8.
9.  </resources>
```

Le gestionnaire d'affichage d'un élément de la liste des Arduinos

La classe [ListArduinosAdapter] est la classe appelée par le [ListView] pour afficher chacun des éléments de la liste des Arduinos. Son code est le suivant :

```
1.  package istia.st.android.vues;
2.
3.  import istia.st.android.R;
4.  ...
5.
6.  public class ListArduinosAdapter extends ArrayAdapter<CheckedArduino> {
7.
8.      // le tableau des arduinos
9.      private List<CheckedArduino> arduinos;
10.     // le contexte d'exécution
11.     private Context context;
12.     // l'id du layout d'affichage d'une ligne de la liste des arduinos
13.     private int layoutResourceId;
14.     // la ligne comporte ou non un checkbox
15.     private Boolean selectable;
16.
17.     // constructeur
18.     public ListArduinosAdapter(Context context, int layoutResourceId, List<CheckedArduino> arduinos,
    Boolean selectable) {
19.         // parent
20.         super(context, layoutResourceId, arduinos);
21.         // on mémorise les infos
22.         this.arduinos = arduinos;
23.         this.context = context;
24.         this.layoutResourceId = layoutResourceId;
25.         this.selectable = selectable;
26.     }
27.
28.     @Override
29.     public View getView(final int position, View convertView, ViewGroup parent) {
```

```
30. ...
31.     }
32. }
```

- ligne 18 : le constructeur de la classe admet quatre paramètres : l'activité en cours d'exécution, l'identifiant de la vue à afficher pour chaque élément de la source de données, la source de données qui alimente la liste, un booléen qui indique si la case à cocher associée à chaque Arduino doit être affichée ou non ;
- lignes 8-15 : ces quatre informations sont mémorisées localement ;

Ligne 29, la méthode [getView] est chargée de générer la vue n° [position] dans le [ListView] et d'en gérer les événements. Son code est le suivant :

```
1.  @Override
2.     public View getView(int position, View convertView, ViewGroup parent) {
3.        // l'arduino courant
4.        final CheckedArduino arduino = arduinos.get(position);
5.        // on crée la ligne courante
6.        View row = ((Activity) context).getLayoutInflater().inflate(layoutResourceId, parent, false);
7.        // on récupère les références sur les [TextView]
8.        TextView txtArduinoId = (TextView) row.findViewById(R.id.txt_arduino_id);
9.        TextView txtArduinoDesc = (TextView) row.findViewById(R.id.txt_arduino_description);
10.       // on remplit la ligne
11.       txtArduinoId.setText(arduino.getId());
12.       txtArduinoDesc.setText(arduino.getDescription());
13.       // la CheckBox n'est pas toujours visible
14.       CheckBox ck = (CheckBox) row.findViewById(R.id.checkBoxArduino);
15.       ck.setVisibility(selectable ? View.VISIBLE : View.INVISIBLE);
16.       if (selectable) {
17.          // on lui affecte sa valeur
18.          ck.setChecked(arduino.isChecked());
19.          // on gère le clic
20.          ck.setOnCheckedChangeListener(new OnCheckedChangeListener() {
21.
22.             public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
23.                arduino.setChecked(isChecked);
24.             }
25.          });
26.       }
27.       // on rend la ligne
28.       return row;
29.    }
```

- ligne 2 : le 1er paramètre est la position dans le [ListView] de la ligne à créer. C'est également la position dans la liste des Arduinos mémorisée localement ;
- ligne 4 : on récupère une référence sur l'Arduino qui va être associé à la ligne construite ;
- ligne 6 : la ligne courante est construite à partir de la vue [listarduinos_item.xml] ;
- lignes 8-9 : les références sur les deux [TextView] sont récupérées ;
- lignes 11-12 : les deux [TextView] reçoivent leur valeur ;
- ligne 14 : on récupère une référence sur la case à cocher ;
- ligne 15 : on la rend visible ou non, selon la valeur [selectable] passée initialement au constructeur ;
- ligne 16 : si la case à cocher est présente ;
- ligne 18 : on lui affecte la valeur [isChecked] de l'Arduino courant ;
- lignes 20-26 : on gère le clic sur la case à cocher ;
- ligne 23 : la valeur de la case à cocher est mémorisée dans l'Arduino courant ;

Gestion de la liste des Arduinos

L'affichage de la liste des Arduinos est pour le moment géré par deux méthodes de la classe [ConfigFragment] :

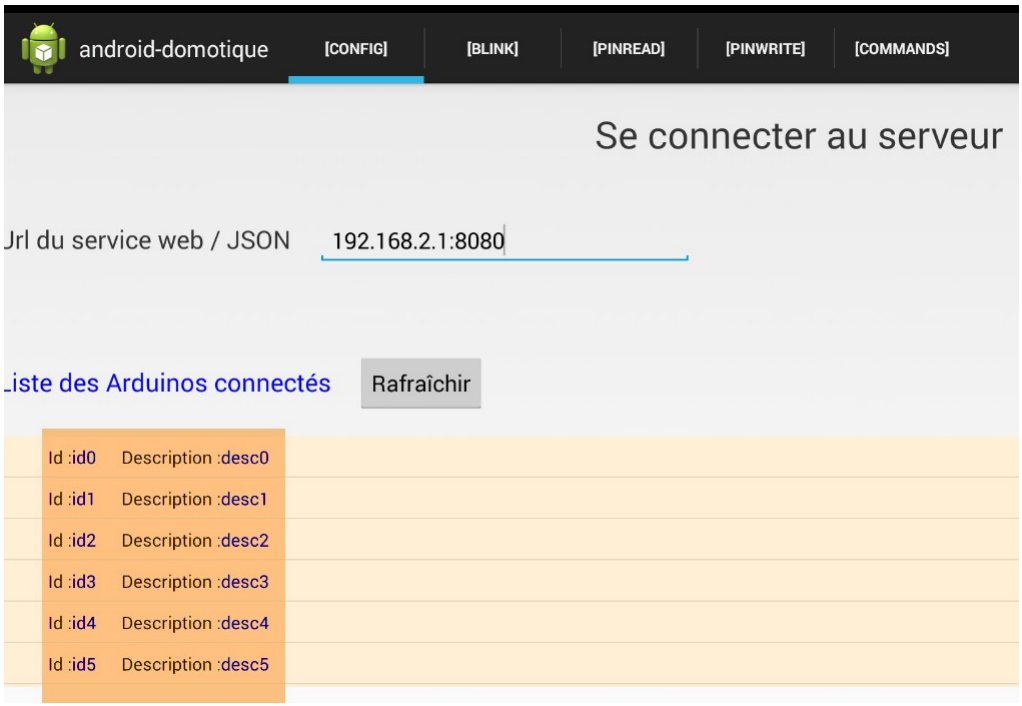
- [clearArduinos] : qui affiche une liste vide ;
- [showArduinos] : qui affiche la liste renvoyée par le serveur ;

Ces deux méthodes évoluent de la façon suivante :

```
1.    // raz liste des Arduinos
2.    private void clearArduinos() {
3.      // on affiche une liste vide
```

```
4.      ListArduinosAdapter adapter = new ListArduinosAdapter(getActivity(), R.layout.listarduinos_item, new
    ArrayList<CheckedArduino>(), false);
5.      listArduinos.setAdapter(adapter);
6.    }
7.
8.    // affichage liste d'Arduinos
9.    private void showArduinos(List<CheckedArduino> checkedArduinos) {
10.     // on affiche les Arduinos
11.     ListArduinosAdapter adapter = new ListArduinosAdapter(getActivity(), R.layout.listarduinos_item,
    checkedArduinos, false);
12.     listArduinos.setAdapter(adapter);
13. }
```

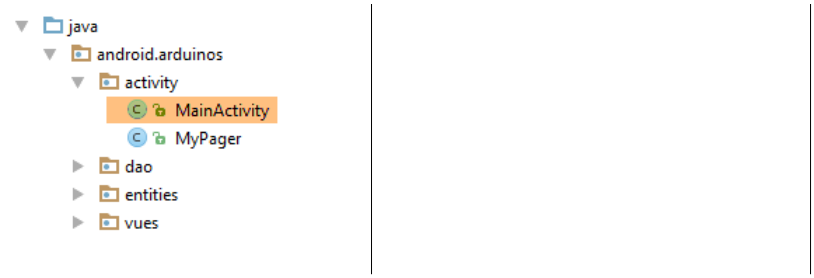
Travail : Faites ces modifications et testez la nouvelle application.



4.6.12 La communication entre vues

Nous avons vu à plusieurs reprises que l'activité unique d'une application Android pouvait être un bon endroit pour stocker les informations qui devaient être partagées entre les vues. Ici il y en a au moins deux à partager : la liste des Arduinos connectés obtenue par la vue [Config] et l'URL du service web / jSON. Nous allons faire afficher la liste des Arduinos par tous les onglets.

La classe [MainActivity] évolue comme suit :

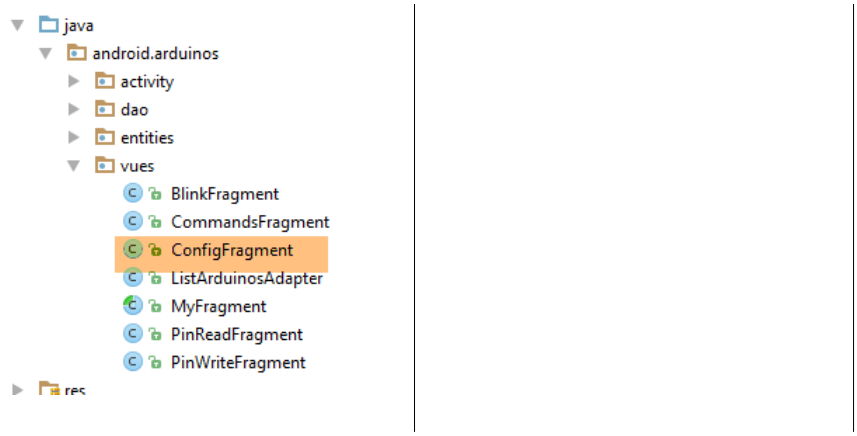



```
1.    // la liste des Arduinos
2.    private List<CheckedArduino> checkedArduinos;
3.  ...
```

```
4.      // getters et setters
5.      public List<CheckedArduino> getCheckedArduinos() {
6.         return checkedArduinos;
7.      }
8.
9.      public void setCheckedArduinos(List<CheckedArduino> checkedArduinos) {
10.        this.checkedArduinos = checkedArduinos;
11.     }
12.
13. }
```

- ligne 2 : la liste d'Arduinos à partager entre les vues ;

La classe [ConfigFragment] évolue comme suit :

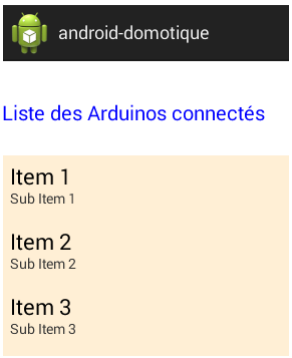


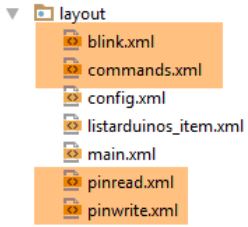
La méthode [showResponse] devient la suivante :

```
1.  @UiThread
2.    void showResponse(ArduinosResponse response) {
3.      // on teste la nature de l'information reçue
4.      int erreur = response.getErreur();
5.      if (erreur == 0) {
6.  ...
7.        // on affiche les Arduinos
8.        showArduinos(checkedArduinos);
9.        // on affiche tous les onglets
10.       activité.showTabs(new Boolean[]{true, true, true, true, true});
11.       // on mémorise les Arduinos dans l'activité
12.       activité.setCheckedArduinos(checkedArduinos);
13.     }
14. ...
15.   }
```

- ligne 12 : mémorise dans l'activité la liste d'objets [CheckedArduino] affichée par la ligne 8 ;

Pour vérifier la communication entre vues, on va faire afficher par toutes les autres vues la liste des Arduinos obtenue par la vue [Config]. Commençons par la vue [blink.xml]. Alors qu'elle n'affichait rien, elle va désormais afficher la liste des Arduinos connectés :

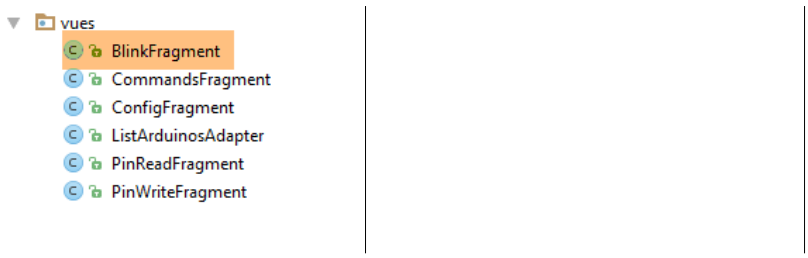




Le code XML de la vue [blink.xml] est le suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/RelativeLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/txt_arduinos"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_marginTop="40dp"
        android:text="@string/titre_list_arduinos"
        android:textColor="@color/blue"
        android:textSize="20sp" />

    <ListView
        android:id="@+id/ListViewArduinos"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/txt_arduinos"
        android:layout_marginTop="30dp"
        android:background="@color/wheat">
    </ListView>

</RelativeLayout>
```

Ce code a été repris directement de la vue [config.xml]. Dupliquez ce code dans les vues [commands.xml, pinread.xml, pinwrite.xml]. Le code du fragment [BlinkFragment] associé à la vue [blink.xml] évolue lui aussi :




```
@EFragment(R.layout.blink)
public class BlinkFragment extends Fragment {

  // interface visuelle
  @ViewById(R.id.ListViewArduinos)
  ListView listArduinos;
  // l'activité
  private MainActivity activité;

```

```
10.   @AfterViews
11.   public void initFragment() {
12.     // on note l'activité
13.     activité = (MainActivity) getActivity();
14.   }
15.
16.   @Override
17.   public void setMenuVisibility(final boolean visible) {
18.     super.setMenuVisibility(visible);
19.     if (visible && activité != null) {
20.       // on affiche les Arduinos
21.       List<CheckedArduino> arduinos = activité.getCheckedArduinos();
22.       if (arduinos != null) {
23.         listArduinos.setAdapter(new ListArduinosAdapter(getActivity(), R.layout.listarduinos_item,
    activité.getCheckedArduinos(), true));
24.       }
25.     }
26.   }
```

- lignes 5-6 : on récupère une référence sur le [ListView] qui affiche les Arduinos ;
- ligne 17 : à chaque fois que la vue devient visible ;
- lignes 19-24 : on rafraîchit le [ListView] des Arduinos avec la liste des Arduinos présente dans l'activité. Ligne 34, on notera le dernier paramètre [true] de l'appel du constructeur de [ListArduinosAdapter] : la case à cocher sera affichée ;
- ligne 19 : on met la condition [activité != **null**] car il n'est pas complètement clair si la méthode [setMenuVisibility] peut être appelée avant que le champ de la ligne 8 ait été initialisé par la méthode de la ligne 11 ;

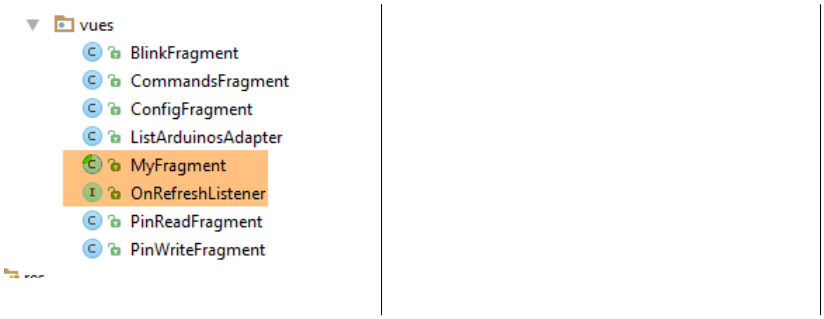
Copiez la méthode [setMenuVisibility] dans chacun des fragments [ConfigFragment, CommandsFragment, PinReadFragment, PinWriteFragment]. Pour le fragment [ConfigFragment], le dernier paramètre de l'appel de la ligne 23 doit être [false] (la case à cocher ne sera pas affichée).

Lorsque vous n'avez plus d'erreurs de compilation, exécutez votre projet. Notez que maintenant toutes les vues partagent la même liste d'Arduinos. Lorsque vous cochez l'un des éléments de la liste dans une vue, vous le retrouvez coché dans la vue suivante.

En fait ça ne marche pas vraiment. Lorsqu'on passe d'un onglet à l'onglet adjacent ça marche. Mais si l'on passe à un onglet non adjacent cela ne marche plus : la liste des Arduinos n'est plus affichée. On la retrouve dès que l'on passe à l'onglet adjacent. Pourquoi ? Mystère...

Nous allons donc explorer une autre solution. Lorsque l'utilisateur clique sur un onglet, on peut en être averti. On en profitera alors pour rafraîchir le fragment qui va être affiché.

Nous introduisons une nouvelle interface [OnRefreshListener] et une nouvelle classe [MyFragment] :



La classe [MyFragment] est la suivante :

```
1.  package istia.st.android.vues;
2.
3.  import android.support.v4.app.Fragment;
4.
5.  public abstract class MyFragment extends Fragment implements OnRefreshListener {
6.  }
```

- ligne 5 : la classe étend la classe [Fragment] que nous utilisions jusqu'à maintenant et elle implémente une interface [OnRefreshListener] que nous allons créer. Elle est déclarée abstraite [abstract] parce qu'elle n'implémente pas elle-même cette interface. Ce sont ses classes filles qui vont le faire ;

L'interface [OnRefreshListener] est la suivante :

```
1.  package istia.st.android.vues;
2.
3.  public interface OnRefreshListener {
4.     public void onRefresh();
5.  }
```

- l'interface n'a qu'une méthode [OnRefresh].

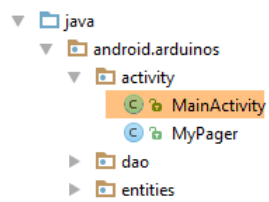
On va faire dériver tous les fragments de la classe [MyFragment] et implémenter dedans la méthode [onRefresh]. C'est dans cette méthode que le [ListView] des arduinos va être régénéré. Pour la classe [BlinkFragment], cela donne le code suivant :

```
1.  package android.arduinos.vues;
2.
3.  import android.arduinos.R;
4.  import android.arduinos.activity.MainActivity;
5.  import android.arduinos.entities.CheckedArduino;
6.  import android.support.v4.app.Fragment;
7.  import android.widget.ListView;
8.  import org.androidannotations.annotations.*;
9.
10. import java.util.List;
11.
12. @EFragment(R.layout.blink)
13. public class BlinkFragment extends MyFragment {
14.
15.   // interface visuelle
16.   @ViewById(R.id.ListViewArduinos)
17.   ListView listArduinos;
18.   // l'activité
19.   private MainActivity activité;
20.
21.   @AfterViews
22.   public void initFragment() {
23.     // on note l'activité
24.     activité = (MainActivity) getActivity();
25.   }
26.
27.   public void onRefresh() {
28.     if (activité != null) {
29.       // on rafraîchit les Arduinos
30.       List<CheckedArduino> arduinos = activité.getCheckedArduinos();
31.       if (arduinos != null) {
32.         listArduinos.setAdapter(new ListArduinosAdapter(activité, R.layout.listarduinos_item, arduinos,
    true));
33.       }
34.     }
35.   }
36. }
```

- ligne 13 : la classe [BlinkFragment] étend la classe [MyFragment] ;
- lignes 27-34 : la méthode [onRefresh] dont le rôle est de régénérer le [ListView] des Arduinos ;
- ligne 28 : au démarrage, la méthode peut être appelée alors même que [activité] n'a pas encore été initialisée. On évite ce cas ;
- ligne 30 : on récupère la liste des Arduinos dans l'activité ;
- lignes 31-33 : si cette liste existe, alors elle est associée au [ListView] ;

Copiez la méthode [onRefresh] dans chacun des fragments [ConfigFragment, CommandsFragment, PinReadFragment, PinWriteFragment] et faites hériter chacun d'eux de la classe [MyFragment]. Pour le fragment [ConfigFragment], le dernier paramètre de l'appel de la ligne 32 doit être [false].

Maintenant, il faut que les méthodes [onRefresh] des fragments soient appelées. Cela se passe dans l'activité [MainActivity] :



La première modification se fait dans notre gestionnaire de fragments. Au lieu de gérer des types [Fragment], il gère désormais des types [MyFragment] :

```
1.  public class SectionsPagerAdapter extends FragmentPagerAdapter {
2.
3.      // les fragments
4.      MyFragment[] fragments = {new ConfigFragment_(), new BlinkFragment_(), new PinReadFragment_(), new
    PinWriteFragment_(), new CommandsFragment_()};
5.
6.      // constructeur
7.      public SectionsPagerAdapter(FragmentManager fm) {
8.        super(fm);
9.      }
10.
11.     // doit rendre le fragment n° i avec ses éventuels arguments
12.     @Override
13.     public MyFragment getItem(int position) {
14.       // on rend le fragment
15.       return fragments[position];
16.     }
17.
18.     // rend le nombre de fragments à gérer
19.     @Override
20.     public int getCount() {
21.       return fragments.length;
22.     }
23.
24.     // rend le titre du fragment n° position
25.     @Override
26.     public CharSequence getPageTitle(int position) {
27.       ...
28.     }
29.   }
```

Les modifications sont aux lignes 4 et 13. L'autre modification se fait dans la méthode qui gère l'événement [changement d'onglet] :

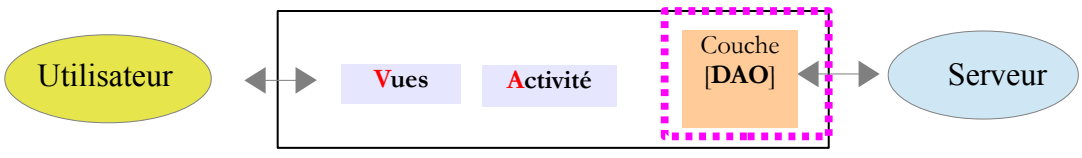
```
1.    public void onTabSelected(Tab tab, FragmentTransaction fragmentTransaction) {
2.      // un onglet a été sélectionné - on change le fragment affiché par le conteneur de fragments
3.      int position = tab.getPosition();
4.      // on l'affiche
5.      mViewPager.setCurrentItem(position);
6.      // on rafraîchit le fragment affiché
7.      mSectionsPagerAdapter.getItem(position).onRefresh();
8.  }
```

- ligne 7 : on demande au fragment n° [position] de se rafraîchir.

Note : l'ordre des instructions des lignes 5 et 7 **importe**. Si vous les échangez, ça ne marche plus. Je ne sais pas expliquer pourquoi.

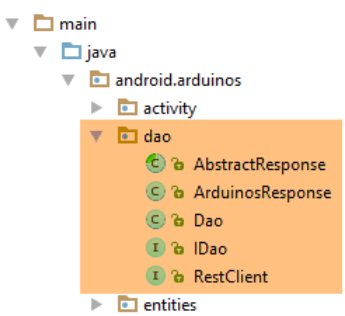
Travail : exécutez cette nouvelle mouture et constatez que le dysfonctionnement noté précédemment a disparu.

4.6.13 La couche [DAO]



Note : pour cette partie, revoyez l'implémentation de la couche [DAO] dans le projet [exemple-10-client] (cf paragraphe 1.11, page 103).

Pour l'instant, nous avons généré la liste des Arduinos connectés. Nous allons maintenant la demander au serveur web / jSON. Pour cela, nous allons construire une couche [DAO] :



4.6.13.1 Implémentation

La couche [DAO] présente l'interface [IDao] suivante :

```
1.  package android.arduinos.dao;
2.
3.  import java.util.List;
4.
5.  public interface IDao {
6.    // liste des arduinos
7.    public ArduinosResponse getArduinos();
8.
9.    // URL du service web
10.   public void setUrlServiceRest(String url);
11.
12.   // timeout du service web
13.   public void setTimeout(int timeout);
14. }
```

- ligne 7 : méthode qui permet d'obtenir la liste des Arduinos dans un objet de type [ArduinosResponse] (cf page 290) ;
- ligne 10 : méthode qui permet de fixer l'URL du service web / jSON ;
- ligne 13 : méthode qui permet de fixer la durée maximale d'attente en millisecondes de la réponse du serveur ;

Rappelons la structure de l'objet [ArduinosResponse] :

```
1.  package android.arduinos.dao;
2.
3.  import java.util.List;
4.
5.  public abstract class AbstractResponse {
6.
7.    // erreur
8.    private int erreur;
9.    // message d'erreur
10.   private List<String> messages;
11.
12.   // getters et setters
13. ...
14. }
```

```
1.  package android.arduinos.dao;
2.
3.
4.  import android.arduinos.entities.Arduino;
5.
6.  import java.util.List;
```

```
7.
8.  public class ArduinosResponse extends AbstractResponse {
9.
10.   // liste des Arduinos
11.   private List<Arduino> arduinos;
12.
13.   // getters et setters
14. ...
15. }
```

Le dialogue client / serveur est réalisé par une implémentation de l'interface [RestClient] suivante :

```
1.  package android.arduinos.dao;
2.
3.  import org.androidannotations.annotations.rest.Get;
4.  import org.androidannotations.annotations.rest.Post;
5.  import org.androidannotations.annotations.rest.Rest;
6.  import org.androidannotations.api.rest.RestClientRootUrl;
7.  import org.androidannotations.api.rest.RestClientSupport;
8.  import org.springframework.http.converter.json.MappingJacksonHttpMessageConverter;
9.  import org.springframework.web.client.RestTemplate;
10.
11. import java.util.List;
12.
13. @Rest(converters = {MappingJacksonHttpMessageConverter.class})
14. public interface RestClient extends RestClientRootUrl, RestClientSupport {
15.
16.   // liste des Arduinos
17.   @Get("/arduinos")
18.   public ArduinosResponse getArduinos();
19.
20.   // RestTemplate
21.   public void setRestTemplate(RestTemplate restTemplate);
22. }
```

L'interface [RestClient] reproduit ce qui a été vu et expliqué dans le projet [exemple-10-client] de la partie 1 du cours.

- lignes 17-18 : la méthode [getArduinos] va demander au serveur web l'URL [/arduinos] avec la méthode HTTP GET ;

L'interface [IDao] est implémentée par la classe [Dao] suivante :

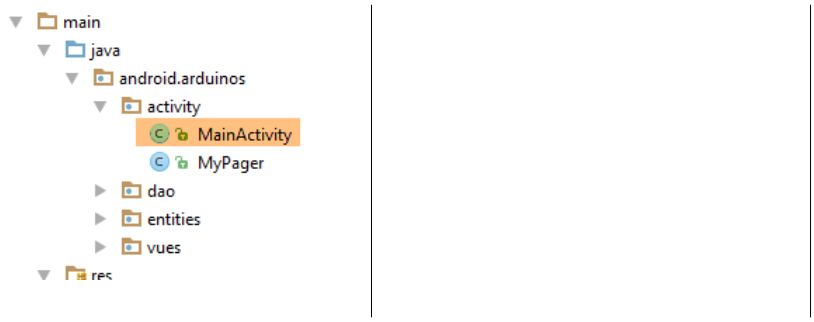
```
1.  package android.arduinos.dao;
2.
3.  import org.androidannotations.annotations.EBean;
4.  import org.androidannotations.annotations.rest.RestService;
5.  import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;
6.  import org.springframework.http.converter.json.MappingJacksonHttpMessageConverter;
7.  import org.springframework.web.client.RestTemplate;
8.
9.  import java.util.List;
10.
11. @EBean(scope = EBean.Scope.Singleton)
12. public class Dao implements IDao {
13.
14.   // client du service REST
15.   @RestService
16.   RestClient restClient;
17.
18.   @Override
19.   public ArduinosResponse getArduinos() {
20.     return restClient.getArduinos();
21.   }
22.
23.   @Override
24.   public void setUrlServiceRest(String urlServiceRest) {
25.     // on fixe l'URL du service REST
26.     restClient.setRootUrl(urlServiceRest);
27.   }
28.
29.   @Override
30.   public void setTimeout(int timeout) {
31.     // on fixe le timeout des requêtes du client REST
32.     HttpComponentsClientHttpRequestFactory factory = new HttpComponentsClientHttpRequestFactory();
```

```
33.     factory.setReadTimeout(timeout);
34.     factory.setConnectTimeout(timeout);
35.     RestTemplate restTemplate = new RestTemplate(factory);
36.     restTemplate.getMessageConverters().add(new MappingJacksonHttpMessageConverter());
37.     restClient.setRestTemplate(restTemplate);
38.   }
39. }
```

De nouveau, on rappelle que ce code a été vu et expliqué dans le projet [exemple-10-client] (cf paragraphe 1.11, page 103).

4.6.13.2 Mise en oeuvre

La couche [DAO] ne sera utilisée que dans un seul endroit, l'activité de l'application :



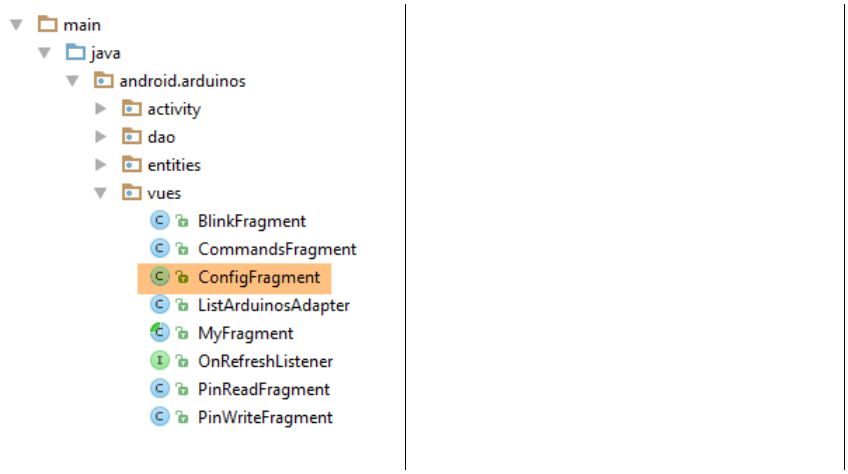
Les vues ne vont pas communiquer directement avec la couche [DAO] mais avec l'activité. Pour cela, l'activité va implémenter elle aussi l'interface [IDao] vue précédemment. La classe [Mainactivity] évolue de la façon suivante :

```
1.  package android.arduinos.activity;
2.
3.  import android.app.ActionBar;
4.  ...
5.
6.  @EActivity
7.  public class MainActivity extends FragmentActivity implements ActionBar.TabListener, IDao {
8.
9.    // couche [DAO]
10.   @Bean(Dao.class)
11.   IDao dao;
12.
13.
14.
15.   // les constantes
16.   // ------------------------------------------------------
17.   // pause avant exécution d'une tâche asynchrone
18.   public final static int PAUSE = 0;
19.   // délai en ms d'attente maximale de la réponse du serveur
20.   private final static int TIMEOUT = 1000;
21.
22.
23.   @AfterInject
24.   public void initActivity() {
25.     // configuration de la couche [DAO]
26.     dao.setTimeout(TIMEOUT);
27.   }
28.
29.   @Override
30.   public ArduinosResponse getArduinos() {
31.     return dao.getArduinos();
32.   }
```

```
33.
34.   @Override
35.   public void setUrlServiceRest(String urlServiceRest) {
36.     dao.setUrlServiceRest(urlServiceRest);
37.   }
38.
39.   @Override
40.   public void setTimeout(int timeout) {
41.     dao.setTimeout(timeout);
42.   }
43. }
```

- ligne 7 : l'activité implémente l'interface [IDao] ;
- lignes 10-11 : une référence de la couche [DAO] est injectée ligne 11 ;
- lignes 23-24 : la méthode [initActivity] est exécutée après cette injection (annotation @AfterInject). On en profite pour fixer le [timeout] de la couche [DAO] ;
- lignes 29-42 : implémentation de l'interface [IDao] ;
- lignes 30-32 : la liste des Arduinos est demandée à la couche [DAO] ;
- lignes 35-37 : la méthode [setUrlServiceRest] servira à fixer l'URL du service web / jSON de la couche [DAO] ;
- lignes 40-42 : idem pour la méthode [setTimeout] ;

Enfin la classe [ConfigFragment] évolue de la façon suivante :




```
1.    @Background(id = "getArduinos", delay = MainActivity.PAUSE)
2.    void getArduinosInBackground() {
3.      ArduinosResponse response = null;
4.      try {
5.        response = activité.getArduinos();
6.      } catch (Exception e) {
7.        response = new ArduinosResponse();
8.        response.setErreur(2);
9.        response.setMessages(getMessagesFromException(e));
10.     }
11.     showResponse(response);
12. }
13.
14.   protected List<String> getMessagesFromException(Exception e) {
15.     // liste des messages d'une exception
16.     Throwable th = e;
17.     List<String> messages = new ArrayList<String>();
18.     while (th != null) {
19.       messages.add(th.getMessage());
20.       th = th.getCause();
21.     }
22.     return messages;
23. }
```

- lignes 2-12 : la méthode qui demande la liste des Arduinos en tâche de fond ;
- ligne 5 : la liste n'est plus générée en dur mais demandée à l'activité qui implémente l'interface [IDao] ;
- lignes 6-10 : en cas d'exception, on construit un objet [ArduinosResponse] avec :
  - un code d'erreur 2,

- une liste de messages qui est celle de la pile d'exceptions (lignes 16-22) ;

Le clic sur le bouton [Rafraîchir] est géré par la méthode suivante :

```
1.    @Click(R.id.btn_Rafraichir)
2.    protected void doRafraichir() {
3.      // on efface la liste actuelle des Arduinos
4.      clearArduinos();
5.      // on vérifie les saisies
6.      if (!pageValid()) {
7.        return;
8.      }
9.      // on mémorise la saisie
10.     activité.setUrlServiceRest(urlServiceRest);
11.     // on demande la liste des Arduinos
12.     nbInfosAttendues = 1;
13.     getArduinosInBackground();
14.     // on commence l'attente
15.     beginWaiting();
16. }
```

- ligne 10 : on mémorise dans l'activité l'URL du service web / jSON. On a vu (cf paragraphe 4.6.13.2, page 329) que l'activité va la transmettre à la couche [DAO] ;

4.6.13.3 Exécution du projet

Travail : faites les modifications précédentes et exécutez votre application.

Note : pour installer le serveur, suivez le paragraphe 4.4, page 287.

4.6.13.4 Refactorisation de la classe [MyFragment]

Dans la construction des autres fragments, vous allez avoir besoin de deux méthodes déjà utilisées dans [ConfigFragment] :

```
1.    protected void showMessages(MainActivity activité, List<String> messages) {
2.      // on construit le texte à afficher
3.      StringBuilder texte = new StringBuilder();
4.      for (String message : messages) {
5.        texte.append(String.format("%s\n", message));
6.      }
7.      // on l'affiche
8.      new AlertDialog.Builder(activité).setTitle("De erreurs se sont
    produites").setMessage(texte).setNeutralButton("Fermer", null).show();
9.  }
```

```
1.    protected List<String> getMessagesFromException(Exception e) {
2.      // liste des messages d'une exception
3.      Throwable th = e;
4.      List<String> messages = new ArrayList<String>();
5.      while (th != null) {
6.        messages.add(th.getMessage());
7.        th = th.getCause();
8.      }
9.      return messages;
10. }
```

Ces deux méthodes visent à afficher des messages d'erreur. Plutôt que de les dupliquer dans chaque fragment, on va les factoriser dans la classe [MyFragment] qui est la classe parente de tous les fragments :

```
1.  package android.arduinos.vues;
2.
3.  import android.app.AlertDialog;
4.  import android.arduinos.activity.MainActivity;
5.  import android.support.v4.app.Fragment;
6.
7.  import java.util.ArrayList;
8.  import java.util.List;
9.
10. public abstract class MyFragment extends Fragment implements OnRefreshListener {
11.
```

```
12.   protected void showMessages(MainActivity activité, List<String> messages) {
13.     ...
14.   }
15.
16.   protected List<String> getMessagesFromException(Exception e) {
17.  ...
18.   }
19. }
```

Ceci fait, supprimez ces deux méthodes de la classe [ConfigFragment] et vérifiez que votre application fonctionne toujours.

4.7 Travail à faire

En procédant comme il a été fait pour la vue [Config], réalisez les quatre autres vues de l'application : [Blink], [PinRead], [PinWrite] et [Commands].

Pour chaque vue, il faut :

- dessiner la vue XML (cf paragraphe 4.6.6, page 305) ;
- construire le fragment associé (cf paragraphe 4.6.7, page 309) ;
- ajouter une méthode à l'interface [RestClient] (cf paragraphe 4.6.13.1, page 327) ;
- ajouter une méthode à l'interface [IDao] (cf paragraphe 4.6.13.1, page 327) ;
- ajouter une méthode à la classe [Dao](cf paragraphe 4.6.13.1, page 327) ;
- ajouter une méthode à l'activité [MainActivity] (cf paragraphe 4.6.13.2, page 329) ;
- tester ;

Note 1 : L'exemple à suivre est le projet [exemple-10-client] du cours.

Note 2 :

La classe [CommandsFragment] envoie une liste contenant une unique commande à exécuter par un ou plusieurs Arduinos. Cette commande sera encapsulée dans la classe [ArduinoCommand] suivante :

```
1.  package android.arduinos.dao;
2.
3.  import java.util.Map;
4.
5.  public class ArduinoCommand {
6.
7.    // data
8.    private String id;
9.    private String ac;
10.   private Map<String, Object> pa;
11.
12.   // constructeurs
13.   public ArduinoCommand() {
14.
15.   }
16.
17.   public ArduinoCommand(String id, String ac, Map<String, Object> pa) {
18.     this.id = id;
19.     this.ac = ac;
20.     this.pa = pa;
21.   }
22.
23.   // getters et setters
24. ...
25. }
```

Dans l'interface [RestClient], la méthode pour exécuter cette liste d'une commande sera la suivante :

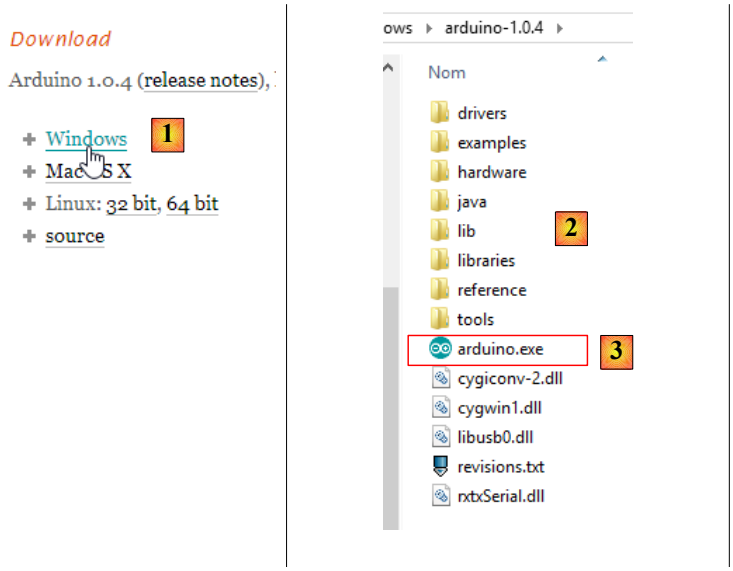
```
1.    // envoi de commandes jSON
2.    @Post("/arduinos/commands/{idArduino}")
3.  public CommandsResponse sendCommands(List<ArduinoCommand> commands, String idArduino);
```

- ligne 2 : l'URL est demandée avec un ordre HTTP POST ;
- ligne 3 : la valeur postée doit être le 1er argument de la méthode ;

4.8 Annexes

4.8.1 Intallation de l'IDE Arduino

Le site officiel de l'Arduino est [http://www.arduino.cc/]. C'est là qu'on trouvera l'IDE de développement pour les Arduinos [http://arduino.cc/en/Main/Software] :



Le fichier téléchargé [1] est un zip qui une fois décompressé donne l'arborescence [2]. On pourra lancer l'IDE en double-cliquant sur l'exécutable [3].

4.8.2 Intallation du pilote (driver) de l'Arduino

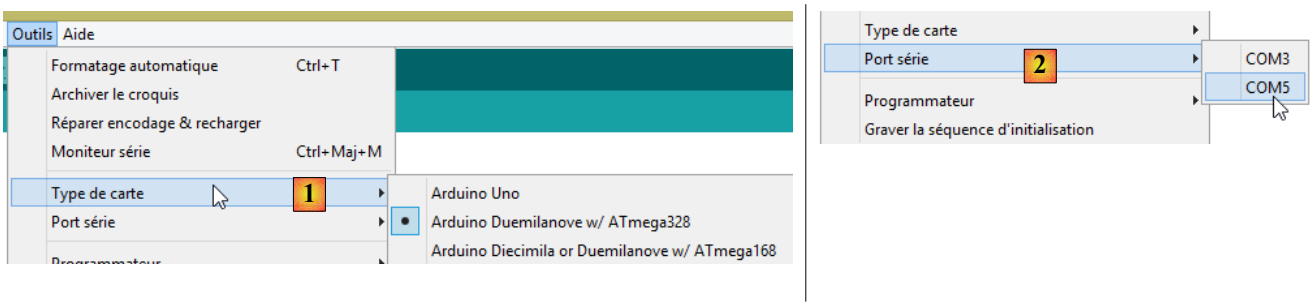
Afin que l'IDE puisse communiquer avec un Arduino, il faut que ce dernier soit reconnu par le PC hôte. Pour cela, on pourra procéder de la façon suivante :

- brancher l'Arduino sur un port USB de l'ordinateur hôte ;
- le système va alors tenter de trouver le pilote du nouveau périphérique USB. Il ne va pas le trouver. Indiquez alors que le pilote est sur le disque à l'endroit **<arduino>/drivers** où <arduino> est le dossier d'installation de l'IDE Arduino.

4.8.3 Tests de l'IDE

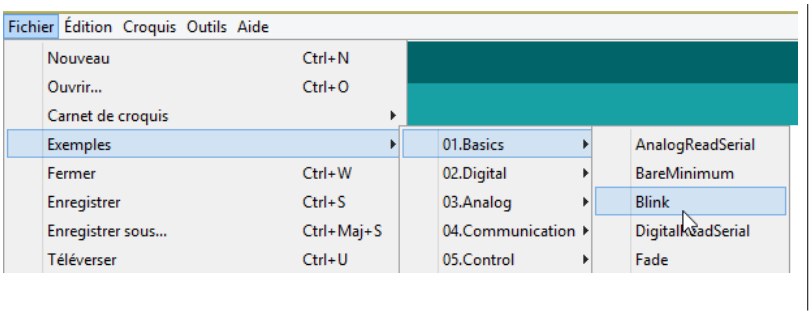
Pour tester l'IDE, on pourra procéder de la façon suivante :

- lancer l'IDE ;
- connecter l'Arduino au PC via son câble USB. Pour l'instant, ne pas inclure la carte réseau ;

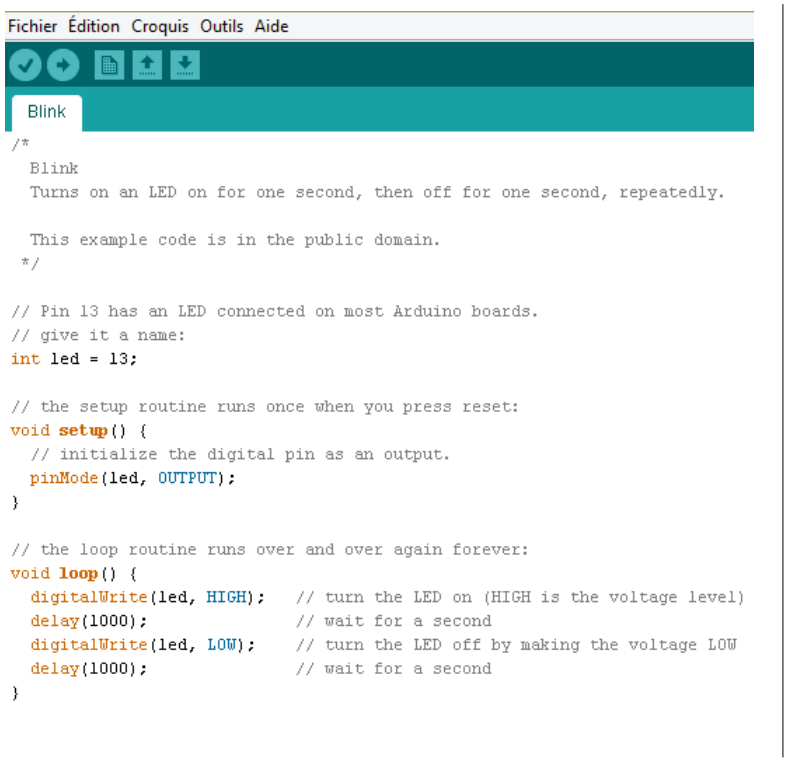


- en [1], choisissez le type de la carte Arduino connectée au port USB ;
- en [2], indiquez le port USB sur lequel l'Arduino est connectée. Pour le savoir, débranchez l'Arduino et notez les ports. Rebranchez-le et vérifiez de nouveau les ports : celui qui a été rajouté est celui de l'Arduino ;

Exécutez certains des exemples inclus dans l'IDE :



L'exemple est chargé et affiché :



Les exemples qui accompagnent l'IDE sont très didactiques et très bien commentés. Un code Arduino est écrit en langage C et se compose de deux parties bien distinctes :

- une fonction [**setup**] qui s'exécute une seule fois au démarrage de l'application, soit lorsque celle-ci est " téléversée " du PC hôte sur l'Arduino, soit lorsque l'application est déjà présente sur l'Arduino et qu'on appuie sur le bouton [Reset]. C'est là qu'on met le code d'initialisation de l'application ;
- une fonction [**loop**] qui s'exécute continuellement (boucle infinie). C'est là qu'on met le coeur de l'application.

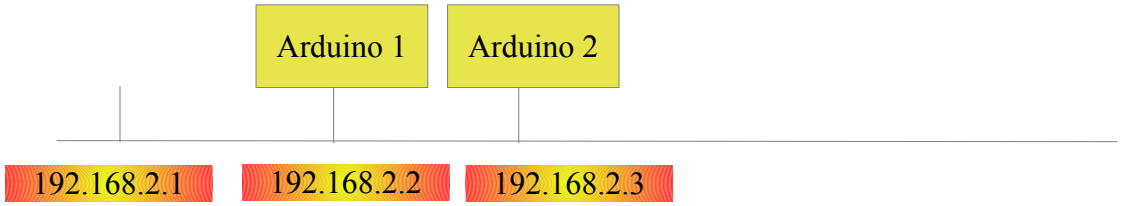
Ici,

- la fonction [setup] configure la pin n° 13 en sortie ;
- la fonction [loop] l'allume et l'éteint de façon répétée : la led s'allume et s'éteint toutes les secondes.

Le programme affiché est transféré (téléversé) sur l'Arduino avec le bouton [1]. Une fois transféré, il s'exécute et la led n° 13 se met à clignoter indéfiniment.

4.8.4 Connexion réseau de l'Arduino

L'Arduino ou les Arduinos et le PC hôte doivent être sur un même réseau privé :

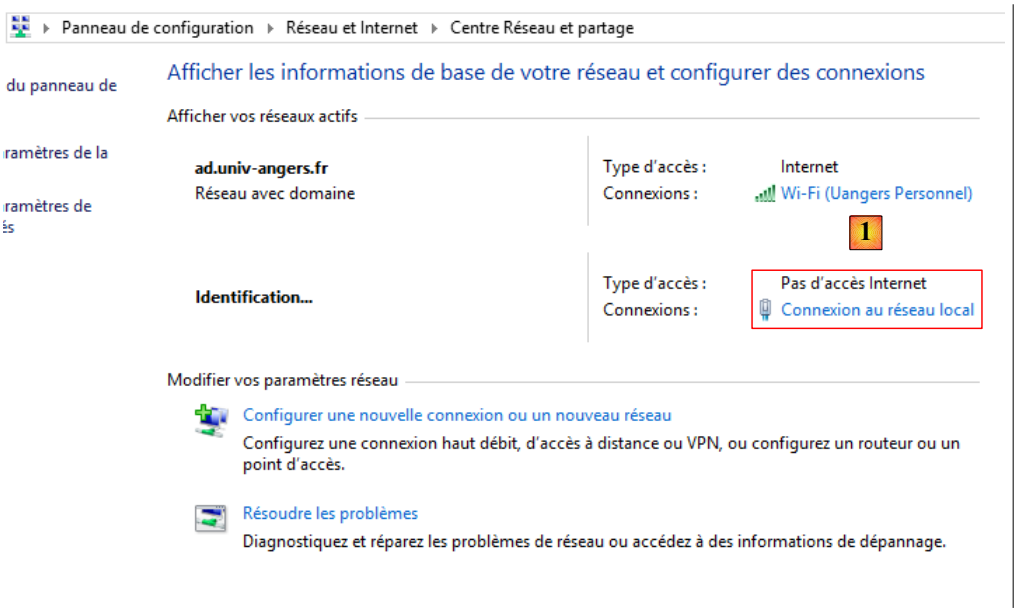


S'il n'y a qu'un Arduino, on pourra le relier au PC hôte par un simple câble RJ 45. S'il y en a plus d'un, le PC hôte et les Arduinos seront mis sur le même réseau par un mini-hub.

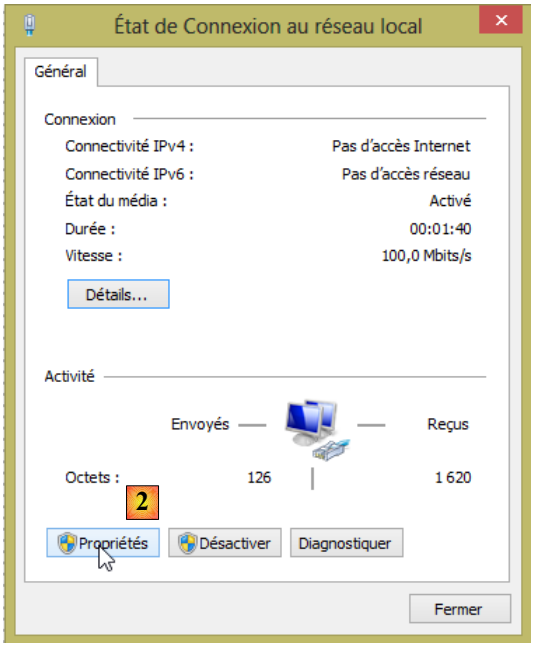
On mettra le PC hôte et les Arduinos sur le réseau privé **192.168.2.x**.

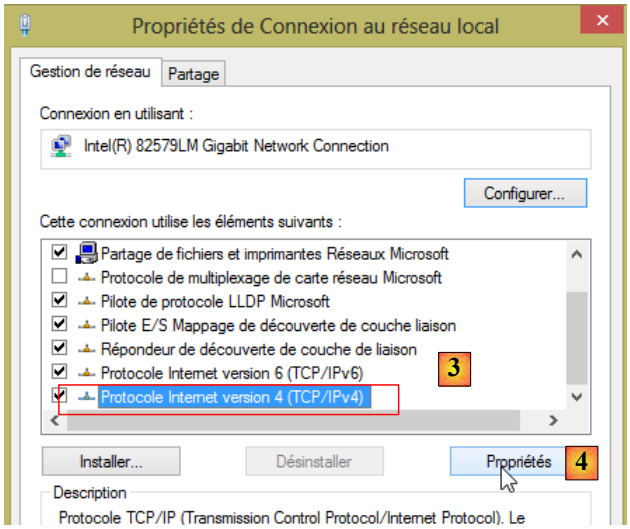
- l'adresse IP des Arduinos est fixée par le code source. Nous verrons comment ;
- l'adresse IP de l'ordinateur hôte pourra être fixée comme suit :

- prendre l'option [Panneau de configuration\Réseau et Internet\Centre Réseau et partage] :

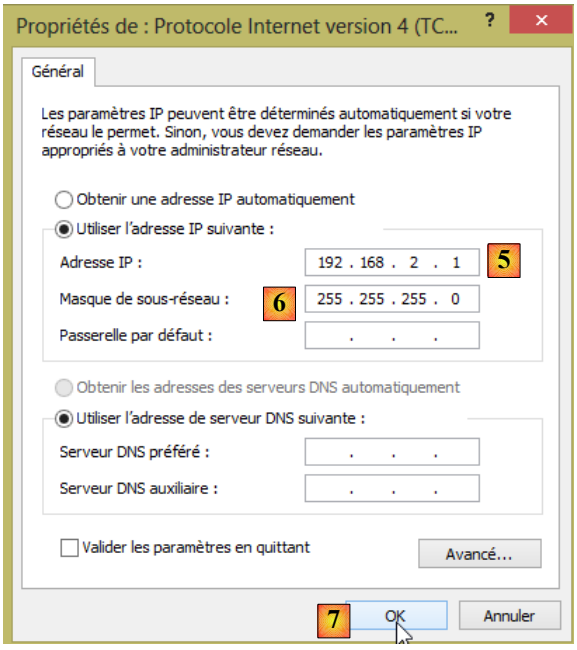


- en [1], suivre le lien [Connexion au réseau local]





- en [2], visualiser les propriétés de la connexion ;
- en [4], visualiser les propriétés IP v4 [3] de la connexion ;



- en [5], donner l'adresse IP [192.168.2.1] à l'ordinateur hôte ;
- en [6], donner le masque [255.255.255.0] au réseau ;
- valider le tout en [7].

4.8.5 Test d'une application réseau

Avec l'IDE, chargez l'exemple [Exemples / Ethernet / WebServer] :

```
1.  /*
2.    Web Server
3.
```

```
4.   A simple web server that shows the value of the analog input pins.
5.   using an Arduino Wiznet Ethernet shield.
6.
7.   Circuit:
8.   * Ethernet shield attached to pins 10, 11, 12, 13
9.   * Analog inputs attached to pins A0 through A5 (optional)
10.
11.  created 18 Dec 2009
12.  by David A. Mellis
13.  modified 9 Apr 2012
14.  by Tom Igoe
15.
16.  */
17.
18. #include <SPI.h>
19. #include <Ethernet.h>
20.
21. // Enter a MAC address and IP address for your controller below.
22. // The IP address will be dependent on your local network:
23. byte mac[] = {
24.   0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
25. IPAddress ip(192,168,1, 177);
26.
27. // Initialize the Ethernet server library
28. // with the IP address and port you want to use
29. // (port 80 is default for HTTP):
30. EthernetServer server(80);
31.
32. void setup() {
33.  // Open serial communications and wait for port to open:
34.   Serial.begin(9600);
35.    while (!Serial) {
36.     ; // wait for serial port to connect. Needed for Leonardo only
37.   }
38.
39.
40.   // start the Ethernet connection and the server:
41.   Ethernet.begin(mac, ip);
42.   server.begin();
43.   Serial.print("server is at ");
44.   Serial.println(Ethernet.localIP());
45. }
46.
47.
48. void loop() {
49.   // listen for incoming clients
50.   EthernetClient client = server.available();
51.   if (client) {
52.     Serial.println("new client");
53.     // an http request ends with a blank line
54.     boolean currentLineIsBlank = true;
55.     while (client.connected()) {
56.       if (client.available()) {
57.         char c = client.read();
58.         Serial.write(c);
59.         // if you've gotten to the end of the line (received a newline
60.         // character) and the line is blank, the http request has ended,
61.         // so you can send a reply
62.         if (c == '\n' && currentLineIsBlank) {
63.           // send a standard http response header
64.           client.println("HTTP/1.1 200 OK");
65.           client.println("Content-Type: text/html");
66.           client.println("Connection: close");
67.           client.println();
68.           client.println("<!DOCTYPE HTML>");
69.           client.println("<html>");
70.                     // add a meta refresh tag, so the browser pulls again every 5 seconds:
71.           client.println("<meta http-equiv=\"refresh\" content=\"5\">");
72.           // output the value of each analog input pin
73.           for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
74.             int sensorReading = analogRead(analogChannel);
75.             client.print("analog input ");
76.             client.print(analogChannel);
77.             client.print(" is ");
78.             client.print(sensorReading);
```

```
79.             client.println("<br />");
80.           }
81.           client.println("</html>");
82.           break;
83.         }
84.         if (c == '\n') {
85.           // you're starting a new line
86.           currentLineIsBlank = true;
87.         }
88.         else if (c != '\r') {
89.           // you've gotten a character on the current line
90.           currentLineIsBlank = false;
91.         }
92.       }
93.     }
94.     // give the web browser time to receive the data
95.     delay(1);
96.     // close the connection:
97.     client.stop();
98.     Serial.println("client disconnected");
99.   }
100.}
```

Cette application crée un serveur web sur le port 80 (ligne 30) à l'adresse IP de la ligne 25. L'adresse MAC de la ligne 23 est l'adresse MAC indiquée sur la carte réseau de l'Arduino.

La fonction [setup] initialise le serveur web :

- ligne 34 : initialise le port série sur lequel l'application va faire des logs. Nous allons suivre ceux-ci ;
- ligne 41 : le noeud TCP-IP (IP, port) est initialisé ;
- ligne 42 : le serveur de la ligne 30 est lancé sur ce noeud réseau ;
- lignes 43-44 : on logue l'adresse IP du serveur web ;

La fonction [loop] implémente le serveur web :

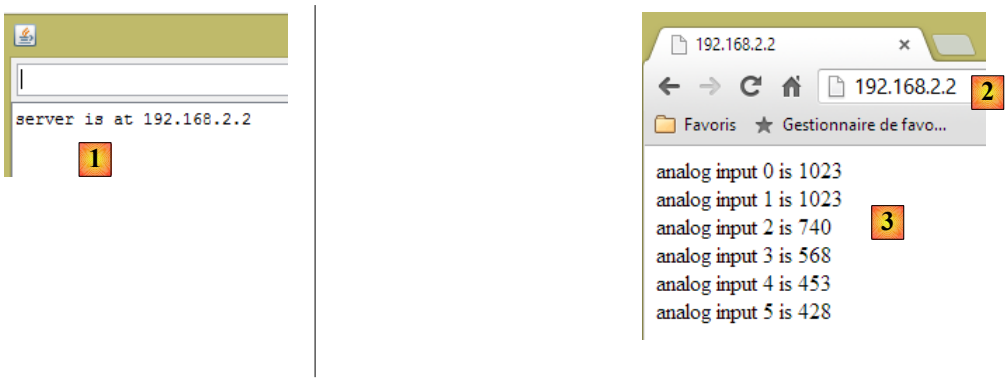
- ligne 50 : si un client se connecte au serveur web *[server].available* rend ce client, sinon rend *null* ;
- ligne 51 : si le client n'est pas *null* ;
- ligne 55 : tant que le client est connecté ;
- ligne 56 : *[client].available* est vrai si le client a envoyé des caractères. Ceux-ci sont stockés dans un buffer. *[client].available* rend vrai tant que ce buffer n'est pas vide ;
- ligne 57 : on lit un caractère envoyé par le client ;
- ligne 58 : ce caractère est affiché en écho sur la console de logs ;
- ligne 62 : dans le protocole HTTP, le client et le serveur échangent des lignes de texte.
    - le client envoie une requête HTTP au serveur web en lui envoyant une série de lignes de texte terminées par une ligne vide,
    - le serveur répond alors au client en lui envoyant une réponse et en fermant la connexion ;

  Ligne 62, le serveur ne fait rien avec les entêtes HTTP qu'il reçoit du client. Il attend simplement la ligne vide : une ligne qui contient le seul caractère **\n** ;
- lignes 64-67 : le serveur envoie au client les lignes de texte standard du protocole HTTP. Elles se terminent par une ligne vide (ligne 67) ;
- à partir de la ligne 68, le serveur envoie un document à son client. Ce document est généré dynamiquement par le serveur et est au format HTML (lignes 68-69) ;
- ligne 71 : une ligne HTML particulière qui demande au navigateur client de rafraîchir la page toutes les 5 secondes. Ainsi le navigateur va demander la même page toutes les 5 secondes ;
- lignes 73-80 : le serveur envoie au navigateur client, les valeurs des 6 entrées analogiques de l'Arduino ;
- ligne 81 : le document HTML est fermé ;
- ligne 82 : on sort de la boucle *while* de la ligne 55 ;
- ligne 97 : la connexion avec le client est fermée ;
- ligne 98 : on logue l'événement dans la console de logs ;
- ligne 100 : on reboucle sur le début de la fonction **loop** : le serveur va se remettre à l'écoute des clients. Nous avons dit que le navigateur client allait redemander la même page toutes les 5 secondes. Le serveur sera là pour lui répondre de nouveau.

Modifiez le code en ligne 25 pour y mettre l'adresse IP de votre Arduino, par exemple :

```
IPAddress ip(192,168,2,2);
```

Téléversez le programme sur l'Arduino. Lancez la console de logs (Ctrl-M) (M majuscule) :



- en [1], la console de logs. Le serveur a été lancé ;
- en [2], avec un navigateur, on demande l'adresse IP de l'Arduino. Ici [192.168.2.2] est traduit par défaut comme [http://198.162.2.2:80];
- en [3], les informations envoyées par le serveur web. Si on visualise le code source de la page du navigateur, on obtient :

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <meta http-equiv="refresh" content="5">
4.  analog input 0 is 1023<br />
5.  analog input 1 is 1023<br />
6.  analog input 2 is 727<br />
7.  analog input 3 is 543<br />
8.  analog input 4 is 395<br />
9.  analog input 5 is 310<br />
10. </html>
```

On reconnaît là, les lignes de texte envoyées par le serveur. Du côté Arduino, la console de logs affiche ce que le client lui envoie :

```
1.  new client
2.  GET /favicon.ico HTTP/1.1
3.  Host: 192.168.2.2
4.  Connection: keep-alive
5.  Accept: */*
6.  User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.22 (KHTML, like Gecko)
    Chrome/25.0.1364.172 Safari/537.22
7.  Accept-Encoding: gzip,deflate,sdch
8.  Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
9.  Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
10.
11. client disconnected
```

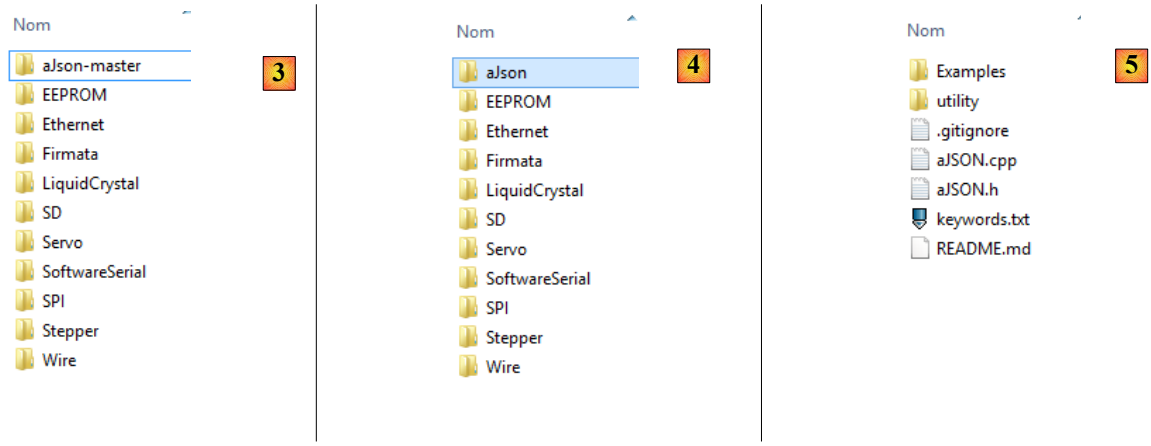
- lignes 2-9 : des entêtes HTTP standard ;
- ligne 10 : la ligne vide qui les termine.

Etudiez bien cet exemple. Il vous aidera à comprendre la programmation de l'Arduino utilisé dans ce TP.

4.8.6 La bibliothèque aJson

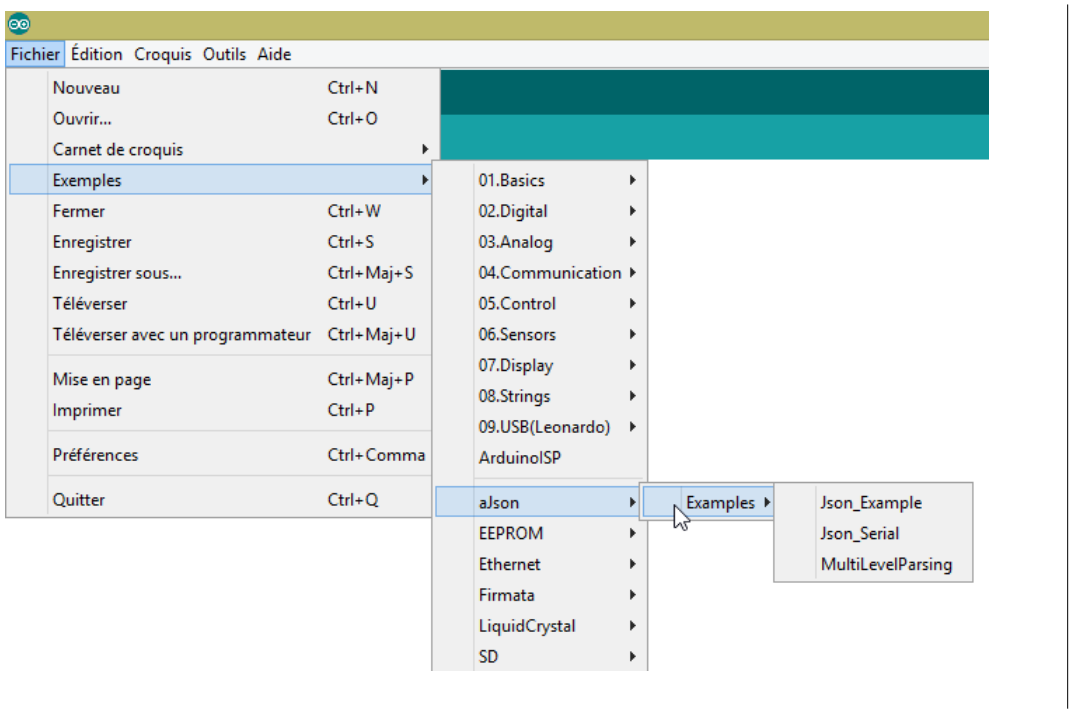
Dans le TP à écrire, les Arduinos échangent des lignes de texte au format jSON avec leurs clients. Par défaut, l'IDE Arduino n'inclut pas de bibliothèque pour gérer le jSON. Nous allons installer la bibliothèque **aJson** disponible à l'URL [https://github.com/interactive-matter/aJson].

- en [1], téléchargez la version zippée du dépôt Github ;
- décompressez le dossier et copiez le dossier [aJson-master] [3] dans le dossier [<arduino>/libraries] [3] où <arduino> est le dossier d'installation de l'IDE Arduino ;



- en [4], renommez ce dossier [aJson] ;
- en [5], son contenu.

Maintenant, lancez l'IDE Arduino :



Vérifiez que dans les exemples, vous avez désormais des exemples pour la bibliothèque **aJson**. Exécutez et étudiez ces exemples.

Table des matières

1 APPRENTISSAGE DE LA PROGRAMMATION ANDROID 6
1.1 Introduction 6
1.1.1 Contenu 6
1.1.2 Pré-requis 7
1.1.3 Les outils utilisés 7
1.2 Exemple-01 : un projet Android basique 8
1.2.1 Création du projet 8
1.2.2 Le manifeste de l'application 11
1.2.3 L'activité principale 13
1.2.4 Exécution de l'application 15
1.3 Exemple-02 : une application Maven / Android basique 19
1.4 Exemple-03 : une application Maven [Android Annotations] basique 27
1.5 Exemple-04 : Vues et événements 34
1.5.1 Création du projet 34
1.5.2 Construire une vue 34
1.5.3 Gestion des événements 41
1.6 Exemple-05 : navigation entre vues 44
1.6.1 Création du projet 44
1.6.2 Ajout d'une seconde activité 44
1.6.3 Navigation de la vue n° 1 à la vue n° 2 46
1.6.4 Construction de la vue n° 2 47
1.6.5 L'activité [SecondActivity] 48
1.6.6 Navigation de la vue n° 2 vers la vue n° 1 51
1.7 Exemple-06 : navigation par onglets 53
1.7.1 Création du projet 53
1.7.2 L'activité 55
1.7.3 Personnalisation des fragments 60
1.8 Exemple-07 : utilisation de la bibliothèque [Android Annotations] avec Gradle 61
1.8.1 Création du projet 61
1.8.2 Configuration du projet 63
1.8.3 Ajout d'une première annotation 66
1.8.4 Refactoring des fragments 68
1.8.5 Un nouveau fragment 71
1.8.6 Désactiver le Swipe ou Balayage 74
1.9 Exemple-08 : La navigation entre vues revisitée 78
1.9.1 Création du projet 78
1.9.2 Configuration Gradle du projet 79
1.9.3 Création des fragment et des vues associées 81
1.9.4 Gestion des fragments par l'activité 84
1.9.5 Mise en place de la navigation 87
1.9.6 Ecriture finale des fragments 88
1.9.7 Conclusion 90
1.10 Exemple-09 : une architecture à deux couches 91
1.10.1 Création du projet 91
1.10.2 La vue [vue1] 92
1.10.3 Le fragment [Vue1Fragment] 95
1.10.4 L'activité [MainActivity] 96
1.10.5 La couche [métier] 97
1.10.6 L'activité [MainActivity] revisitée 99
1.10.7 Le fragment [Vue1Fragment] revisité 100
1.10.8 Exécution 102
1.11 Exemple-10 : architecture client / serveur 103
1.11.1 Le serveur [web / jSON] 103
1.11.1.1 Création du projet 103
1.11.1.2 La couche [métier] 107
1.11.1.3 Le service web / jSON 109
1.11.1.4 Configuration du projet Spring 113
1.11.1.5 Exécution du serveur web 114
1.11.1.6 Génération du jar exécutable du projet 118
1.11.2 Le client Android du serveur web / jSON 121
1.11.2.1 Création du projet 121

1.11.2.2 La couche [DAO]....122
1.11.2.2.1 L'interface [IDao] de la couche [DAO]....122
1.11.2.2.2 La classe [AleaResponse]....123
1.11.2.2.3 La classe [WebClient]....123
1.11.2.2.4 Implémentation de la couche [DAO]....124
1.11.2.3 Les dépendances Gradle....126
1.11.2.4 Le manifeste de l'application Android....127
1.11.2.5 L'activité [MainActivity]....128
1.11.2.6 La vue [vue1.xml]....130
1.11.2.7 Le fragment [Vue1Fragment]....133
1.11.2.8 Exécution du projet....137
1.11.3 Conversion du projet [Gradle] en projet [Maven]....140
1.12 Exemple-11 : composants de saisie de données....146
1.12.1 Création du projet....146
1.12.2 La vue XML du formulaire....147
1.12.3 Les chaînes de caractères du formulaire....151
1.12.4 Le fragment du formulaire....152
1.12.5 Exécution du projet....155
1.13 Exemple-12 : utilisation d'un patron de vues....156
1.13.1 Création du projet....156
1.13.2 Le patron des vues....156
1.14 Exemple-13 : le composant [ListView]....161
1.14.1 Création du projet....161
1.14.2 La vue [Vue1] initiale....162
1.14.3 La vue répétée par le [ListView]....164
1.14.4 Le fragment [Vue1Fragment]....165
1.14.5 L'adaptateur [ListAdapter] du [ListView]....167
1.14.6 Retirer un élément de la liste....168
1.14.7 La vue XML [Vue2]....169
1.14.8 Le fragment [Vue2Fragment]....170
1.14.9 Exécution....171
1.14.10 Amélioration....171
1.15 Exemple-14 : utiliser un menu....173
1.15.1 Création du projet....173
1.15.2 La définition XML du menu....173
1.15.3 La gestion du menu dans le fragment [Vue1Fragment]....174
1.15.4 La gestion du menu dans le fragment [Vue2Fragment]....175
1.15.5 Exécution....176
1.16 Annexes....177
1.16.1 La tablette Android....177
1.16.2 Installation d'un JDK....178
1.16.3 Installation du SDK Manager d'Android....179
1.16.4 Installation du gestionnaire d'émulateurs Genymotion....181
1.16.5 Installation de Maven....182
1.16.6 Installation de l'IDE IntellijIDEA Community Edition....183
1.16.7 Utilisation des exemples....189
1.16.8 Installation du plugin Chrome [Advanced Rest Client]....198
1.16.9 Gestion du jSON en Java....199
2 ÉTUDE DE CAS....203
2.1 Le projet....203
2.2 Les vues du client Android....203
2.3 L'architecture du projet....205
2.4 La base de données....206
2.4.1 La table [MEDECINS]....206
2.4.2 La table [CLIENTS]....207
2.4.3 La table [CRENEAUX]....207
2.4.4 La table [RV]....208
2.4.5 Génération de la base....208
2.5 Le serveur web / jSON....210
2.5.1 Mise en oeuvre....211
2.5.2 Sécurisation du service web....212
2.5.3 Liste des médecins....212
2.5.4 Liste des clients....215

2.5.5 Liste des créneaux d'un médecin....216
2.5.6 Liste des rendez-vous d'un médecin....216
2.5.7 L'agenda d'un médecin....217
2.5.8 Obtenir un médecin par son identifiant....218
2.5.9 Obtenir un client par son identifiant....218
2.5.10 Obtenir un créneau par son identifiant....219
2.5.11 Obtenir un rendez-vous par son identifiant....219
2.5.12 Ajouter un rendez-vous....219
2.5.13 Supprimer un rendez-vous....221
2.6 Le client Android....222
2.6.1 Architecture du projet IntellijIDEA....222
2.6.2 Configuration Gradle....223
2.6.3 La couche [DAO]....224
2.6.4 Implémentation des échanges client / serveur....225
2.6.5 L'activité....230
2.6.6 La session....234
2.6.7 Le patron des vues....235
2.6.8 La classe mère des fragments....237
2.6.9 Gestion de la vue de configuration....238
2.6.10 Gestion de la vue d'accueil....244
2.6.11 Gestion de la vue Agenda....249
2.6.12 La vue d'ajout d'un rendez-vous....257
2.7 Conclusion....262
2.8 Annexes....263
2.8.1 Installation de [WampServer]....263
3 TP 1 : GESTION BASIQUE D'UNE FICHE DE PAIE....266
3.1 Introduction....266
3.2 La base de données....266
3.2.1 Définition....266
3.2.2 Génération....267
3.2.3 Modélisation Java de la base....268
3.3 Installation du serveur web / jSON....269
3.3.1 Installation....270
3.3.2 Les URL du service web/jSON....271
3.3.3 Les réponses jSON du service web/jSON....273
3.4 Tests du client Android....275
3.5 Travail à faire....277
4 TP 2 - PILOTER DES ARDUINOS AVEC UNE TABLETTE ANDROID....280
4.1 Architecture du projet....280
4.2 Le matériel....280
4.2.1 L'Arduino....280
4.2.2 La tablette....282
4.2.3 L'émulateur [Genymotion]....283
4.3 Programmation des Arduinos....283
4.4 Le serveur web / jSON Java....287
4.4.1 Installation....287
4.4.2 Les URL exposées par le service web / jSON....289
4.4.3 Les tests du service web / jSON....292
4.5 Tests du client Android....295
4.6 Le client Android du service web / jSON....298
4.6.1 L'architecture du client....298
4.6.2 Le projet IntellijIDEA du client....299
4.6.3 Les cinq vues XML....301
4.6.4 Les cinq fragments....302
4.6.5 La classe [MainActivity]....302
4.6.6 La vue XML [config]....305
4.6.7 Le fragment [ConfigFragment]....309
4.6.8 Vérification des saisies....310
4.6.9 Gestion des onglets....311
4.6.10 Affichage de la liste des Arduinos....313
4.6.11 Un patron pour afficher un Arduino....317
4.6.12 La communication entre vues....321
4.6.13 La couche [DAO]....326

4.6.13.1 Implémentation....327
4.6.13.2 Mise en oeuvre....329
4.6.13.3 Exécution du projet....331
4.6.13.4 Refactorisation de la classe [MyFragment]....331
4.7 Travail à faire....332
4.8 Annexes....333
4.8.1 Intallation de l'IDE Arduino....333
4.8.2 Intallation du pilote (driver) de l'Arduino....333
4.8.3 Tests de l'IDE....333
4.8.4 Connexion réseau de l'Arduino....335
4.8.5 Test d'une application réseau....336
4.8.6 La bibliothèque aJson....339