TD JAVA

Mots clés : Bases de données, JDBC, architectures 3 couches

1.Le problème

On se propose d'écrire une application console permettant d'établir le bulletin de salaire des assistantes maternelles employées par la "Maison de la petite enfance" d'une commune.

1.1 La base de données

Les données statiques utiles pour construire la fiche de paie sont placées dans une base de données Firebird nommée **dbpam** (**pam**=Paie Assistante Maternelle). Cette base a un administrateur appelé **sysdba** ayant le mot de passe **masterkey.** Elle a trois tables, **EMPLOYES, COTISATIONS** et **INDEMNITES,** dont la structure est la suivante :

Table **EMPLOYES** : rassemble des informations sur les différentes assistantes maternelles

Structure :

FK	PK	Field Name	Field Type	Domain	Size
	1	SS	CHAR		15
		NOM	VARCHAR		30
		PRENOM	VARCHAR		30
		ADRESSE	VARCHAR		50
		VILLE	VARCHAR		30
		CODEPOSTAL	CHAR		5
F		INDICE	INTEGER		

SS	numéro de sécurité sociale de l'employé - clé primaire
NOM	nom de l'employé
PRENOM	son prénom
ADRESSE	son adresse
VILLE	sa ville
CODEPOSTAL	son code postal
INDICE	son indice de traitement - clé étrangère sur le champ [INDICE] de la table [INDEMNITES]

Son contenu pourrait être le suivant :

SS	NOM	PRENOM	ADRESSE	VILLE	CODEPOSTAL	INDICE
260124402111742	Laverti	Justine	la Brûlerie	St Marcel	49014	1
254104940426058	Jouveinal	Marie	5 rue des Oiseaux	St Corentin	49203	2

Table **COTISATIONS** : rassemble des pourcentages nécessaires au calcul des cotisations sociales

Structure :

FK	PK	Field Name	Field Type
		CSGRDS	DOUBLE PRECISION
		CSGD	DOUBLE PRECISION
		SECU	DOUBLE PRECISION
		RETRAITE	DOUBLE PRECISION

CSGRDS	pourcentage : contribution sociale généralisée + contribution au remboursement de la dette sociale
CSGD	pourcentage : contribution sociale généralisée déductible
SECU	pourcentage : contribution sociale généralisée déductible
RETRAITE	pourcentage : retraite complémentaire + assurance chômage

Son contenu pourrait être le suivant :

CSGRDS	CSGD	SECU	RETRAITE
3,490	6,150	9,390	7,880

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.

PK	Field Name	Field Type
1	INDICE	INTEGER
	BASEHEURE	DOUBLE PRECISION
	ENTRETIENJOUR	DOUBLE PRECISION
	REPASJOUR	DOUBLE PRECISION
	INDEMNITESCP	DOUBLE PRECISION

INDICE	indice de traitement - clé primaire
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 :

INDICE	BASEHEURE	ENTRETIENJOUR	REPASJOUR	INDEMNITESCP
1	1,930	2,000	3,000	12,000
2	2,100	2,100	3,100	15,000

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).

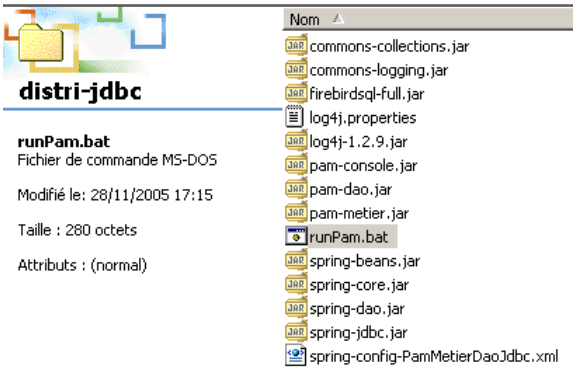
1.2 Mode de calcul du salaire d'une assistante maternelle

Nous présentons maintenant le mode de calcul du salaire mensuel d'une assistante maternelle. Il ne prétend pas être celui utilisé dans la réalité. Nous prenons pour exemple, le salaire de Mme Marie Jouveinal qui a travaillé 150 h sur 20 jours pendant le mois à payer.

Les éléments suivants sont pris en compte :	[TOTALHEURES]: total des heures travaillées dans le mois [TOTALJOURS]: total des jours travaillés dans le mois	[TOTALHEURES]=150 [TOTALJOURS]= 20
Le salaire de base de l'assistante maternelle est donné par la formule suivante :	[SALAIREBASE]=([TOTALHEURES]*[BASEHEURE])*(1+[INDEMNITESCP]/100)	[SALAIREBASE]=(150*[2.1])*(1+0.15) = 362,25
Un certain nombre de cotisations sociales doivent être prélevées sur ce salaire de base :	Contribution sociale généralisée et contribution au remboursement de la dette sociale : [SALAIREBASE]*[CSGRDS/100] Contribution sociale généralisée déductible : [SALAIREBASE]*[CSGD/100] Sécurité sociale, veuvage, vieillesse : [SALAIREBASE]*[SECU/100] Retraite Complémentaire + AGPF + Assurance Chômage : [SALAIREBASE]*[RETRAITE/100]	CSGRDS : 12,64 CSGD : 22,28 Sécurité sociale : 34,02 Retraite : 28,55
Total des cotisations sociales :	[COTISATIONSSOCIALES]=[SALAIREBASE]*(CSGRDS+CSGD+SECU+RETRAITE)/100	[COTISATIONSSOCIALES]=97,48
Par ailleurs, l'assistante maternelle a droit, chaque jour travaillé, à une indemnité d'entretien ainsi qu'à une indemnité de repas. A ce titre elle reçoit les indemnités suivantes :	[INDEMNITÉS]=[TOTALJOURS]*(ENTRETIENJOUR+REPASJOUR)	[INDEMNITES]=104
Au final, le salaire net à payer à l'assistante maternelle est le suivant :	[SALAIREBASE]-[COTISATIONSSOCIALES]+[INDEMNITÉS]	[salaire NET]=368,77

1.3 Fonctionnement de l'application à écrire

Voici un exemple de ce qui est attendu. L'ensemble des exécutables et des fichiers de configuration nécessaires à l'application sont rassemblés dans un unique répertoire :



Le fichier [runPam.bat] contient la commande lançant l'exécution du programme Java de l'application :

```
1.  @echo off
2.  %JAVA_HOME%\bin\java.exe -classpath pam-console.jar;pam-metier.jar;pam-dao.jar;commons-
    collections.jar;commons-logging.jar;firebirdsql-full.jar;log4j-1.2.9.jar;spring-beans.jar;spring-
    core.jar;spring-dao.jar;spring-jdbc.jar; istia.st.pam.ui.console.MainPamJdbc %1 %2 %3
```

Nous ne nous attarderons pas sur ce fichier de commandes. On notera simplement les points suivants :

- les fichiers .jar nécessaires à l'application sont indiqués derrière le paramètre [classpath]
- le programme exécuté est la classe [istia.st.pam.ui.console.MainPamJdbc]
- on lui passe trois paramètres notés ci-dessus %1 %2 %3

Voici un exemple d'exécution :

```
1.  dos>runPam.bat 254104940426058 150 20
2.
3.  Valeurs saisies :
4.  N° de sécurité sociale de l'employé : 254104940426058
5.  Nombre d'heures travaillées : 150
6.  Nombre de jours travaillés : 20
7.
8.  Informations Employé :
9.  Nom : Jouveinal
10. Prénom : Marie
11. Adresse : 5 rue des Oiseaux
12. Ville : St Corentin
13. Code Postal : 49203
14. Indice : 2
15.
16. Informations Cotisations :
17. CSGRDS : 3.49 %
18. CSGD : 6.15 %
19. Retraite : 7.88 %
20. Sécurité sociale : 9.39 %
21.
22. Informations Indemnités :
23. Salaire horaire : 2.1 euro
24. Entretien/jour : 2.1 euro
25. Repas/jour : 3.1 euro
26. Congés Payés : 15.0 %
27.
28. Informations Salaire :
29. Salaire de base : 362.25 euro
30. Cotisations sociales : 97.48 euro
31. Indemnités d'entretien : 42.0 euro
32. Indemnités de repas : 62.0 euro
33. Salaire net : 368.77 euro
```

On écrira un programme qui recevra les informations suivantes :

1. n° de sécurité sociale de l'assistante maternelle (254104940426058 dans l'exemple - ligne 1)
2. nombre total d'heures travaillées (150 dans l'exemple - ligne 1)
3. nombre total de jours travaillés (20 dans l'exemple - ligne 1)

On voit que :

- lignes 9-14 : affichent les informations concernant l'employé dont on a donné le n° de sécurité sociale
- lignes 17-20 : affichent les taux des différentes cotisations
- lignes 23-26 : affichent les indemnités associées à l'indice de traitement de l'employé (ici l'indice 2)

- lignes 29-33 : affichent les éléments constitutifs du salaire à payer

L'application signale les erreurs éventuelles.

Appel sans paramètres :

```
dos>runPam.bat
Syntaxe : pg num_securite_sociale nb_heures_travaillées nb_jours_travaillés
```

Appel avec des données erronées :

```
dos>runPam.bat 254104940426058 150x 20x
Le nombre d'heures travaillées [150x] est erroné
Le nombre de jours travaillés [20x] est erroné
```

Appel avec un n° de sécurité sociale erroné :

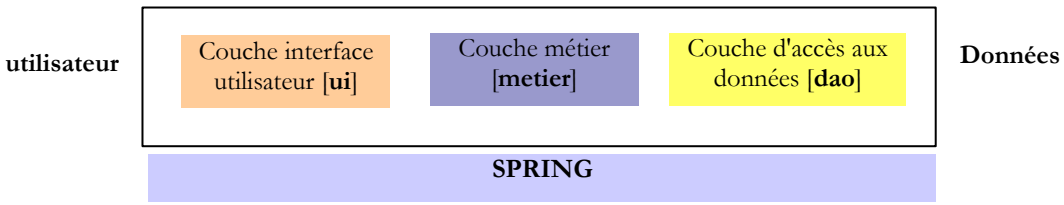
```
dos>runPam.bat xx 150 20
L'erreur suivante s'est produite : L'employé de n°[xx] est introuvable
```

1.4 Architecture de l'application Java

La solution précédente inclut trois phases classiques :

1. l'acquisition des données (en fait passées directement en paramètres au programme)
2. le calcul de la solution
3. l'affichage des résultats

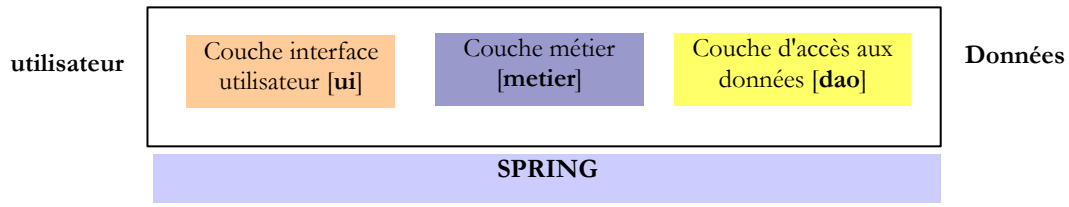
L'application sera découpée en trois couches ayant chacune un rôle bien défini :



- la couche [metier] est celle qui contient les règles métier de l'application. Pour notre application, ce sont les règles de calcul du salaire exposées au paragraphe 1.2, page 2. Cette couche a besoin de données pour travailler. Dans le schéma ci-dessus, les données peuvent provenir de deux endroits :
    - la couche d'accès aux données ou [dao] (DAO = Data Access Object) pour les données déjà enregistrées dans des fichiers ou bases de données.
    - la couche d'interface avec l'utilisateur ou [ui] (UI = User Interface) pour les données saisies par l'utilisateur ou affichées à l'utilisateur. C'est le cas ici pour des informations telles que :
        - le n° de sécurité sociale de la personne à payer
        - son nombre d'heures travaillées
        - son nombre de jours travaillés
        - la feuille de salaire
- de façon générale, la couche [dao] s'occupe de l'accès aux données persistantes (fichiers, bases de données) ou non persistantes (réseau, ...).
- la couche [ui] elle, s'occupe des interactions avec l'utilisateur s'il y en a.
- les trois couches sont rendues indépendantes grâce à l'utilisation d'interfaces Java. Elles sont intégrées dans le schéma global de l'application à l'aide du framework Spring.

1.5 Les interfaces des couches [metier] et [dao]

Revenons à l'architecture 3tier de notre application :



Quelle interface doit offrir la couche [dao] à la couche [metier] et quelle interface doit offrir la couche [metier] à la couche [ui] ? Une première approche pour définir les interfaces des différentes couches est d'examiner les différents cas d'usage (use cases) de l'application. Ici nous n'en avons qu'un. Nous avons en effet dit que notre application serait utilisée de la façon suivante (cf paragraphe 1.3, page 3) :

```
1.  dos>runPam.bat 254104940426058 150 20
2.
3.  Valeurs saisies :
4.  N° de sécurité sociale de l'employé : 254104940426058
5.  Nombre d'heures travaillées : 150
6.  Nombre de jours travaillés : 20
7.
8.  Informations Employé :
9.  Nom : Jouveinal
10. ...
11.
12. Informations Cotisations :
13. CSGRDS : 3.49 %
14. ...
15.
16. Informations Indemnités :
17. ...
18.
19. Informations Salaire :
20. Salaire de base : 362.25 euro
21. Cotisations sociales : 97.48 euro
22. Indemnités d'entretien : 42.0 euro
23. Indemnités de repas : 62.0 euro
24. Salaire net : 368.77 euro
```

L'application reçoit trois informations de l'utilisateur (cf ligne 1 ci-dessus)

- le n° de sécurité sociale de l'assistante maternelle
- le nombre d'heures travaillées dans le mois
- le nombre de jours travaillés dans le mois

A partir de ces information et d'autres enregistrées dans des fichiers de configuration, l'application affiche les informations suivantes :

- lignes 4-6 : les valeurs saisies
- lignes 8-10 : les informations liées à l'employé dont on a donné le n° de sécurité sociale
- lignes 12-14 : les taux des différentes cotisations sociales
- lignes 16-17 : les différentes indemnités bénéficiant à l'assistante maternelle
- lignes 19-24 : les éléments de la feuille de salaire de l'assistante maternelle

Un certain nombre d'informations doivent être fournies par la couche [metier] à la couche [ui] :

1. les informations liées à une assistante maternelle identifiée par son n° de sécurité sociale. On trouve ces informations dans la table [EMPLOYES]. Cela permet d'afficher les lignes 8-10.
2. les montants des divers taux de cotisations sociales à prélever sur le salaire brut. On trouve ces informations dans la table [COTISATIONS]. Cela permet d'afficher les lignes 12-14.
3. les montants des diverses indemnités liées à la fonction d'assistante maternelle. On trouve ces informations dans la table [INDEMNITES]. Cela permet d'afficher les lignes 16-17.
4. les éléments constitutifs du salaire affichés lignes 19-24

De ceci, on pourrait décider d'une première écriture de l'interface [IPamMetier] présentée par la couche [metier] à la couche [ui] :

```
1.  package istia.st.pam.metier;
2.
3.  public interface IPamMetier {
4.          // obtenir la feuille de salaire
5.          FeuilleSalaire calculerFeuilleSalaire(String numEmploye, double nbHeuresTravaillées, int
    nbJoursTravaillés );
6.  }
```

- ligne 1 : les éléments de la couche [metier] sont mis dans le paquetage [istia.st.pam.metier]
- ligne 5 : la méthode [calculerFeuilleSalaire] prend pour paramètres les trois informations acquises par la couche [ui] et rend un objet de type [FeuilleSalaire] contenant les informations que la couche [ui] affichera sur la console. La classe [FeuilleSalaire] pourrait être la suivante :

```
1.  package istia.st.pam.metier;
2.
3.  import istia.st.pam.dao.Cotisations;
4.  import istia.st.pam.dao.Employe;
5.  import istia.st.pam.dao.Indemnites;
6.
7.  public class FeuilleSalaire {
8.          // champs privés
9.          private Employe employe;
10.         private Cotisations cotisations;
11.         private Indemnites indemnites;
12.         private ElementsSalaire elementsSalaire;
13.
14. ...
15. }
```

- ligne 9 : l'employé concerné par la feuille de salaire - information n° 1 affichée par la couche [ui]
- ligne 10 : les différents taux de cotisation - information n° 2 affichée par la couche [ui]
- ligne 11 : les différentes indemnités liées à l'indice de l'employé - information n° 3 affichée par la couche [ui]
- ligne 12 : les éléments constitutifs de son salaire - information n° 4 affichée par la couche [ui]

La couche [metier] ne peut initialiser les champs [employe, cotisations, indemnites] de l'objet [FeuilleSalaire] ci-dessus qu'en questionnant la couche [dao] car ces informations sont dans les tables de la base de données. Aussi une première définition de l'interface [IPamDao] de la couche [dao] pourrait être la suivante :

```
1.  package istia.st.pam.dao;
2.
3.  public interface IPamDao {
4.          // obtenir un salarié via son n° de sécurité sociale
5.          Employe getEmployeByNum(String num);
6.
7.          // obtenir la liste des cotisations
8.          Cotisations getCotisations();
9.
10.         // obtenir les indemnités d'un salarié via son n° de sécurité sociale
11.         Indemnites getIndemnitesByIndice(int indice);
12. }
```

- ligne 1 : toutes les classes et interfaces de la couche [dao] sont placées dans le paquetage [istia.st.pam.dao].
- ligne 5 : la méthode [getEmployeByNum] permet d'obtenir un enregistrement de la table [EMPLOYES] à partir de son n° de sécurité sociale.
- ligne 7 : la méthode [getCotisations] permet d'obtenir l'unique enregistrement de la table [COTISATIONS]
- ligne 11 : la méthode [getIndemnitesByIndice] permet d'obtenir un enregistrement de la table [INDEMNITES] à partir d'un indice de traitement.

Au final, on voit que les couches [metier] et [dao] ont besoin des classes suivantes :
- la classe [Employe] qui encapsule dans un objet les données d'une ligne de la table [EMPLOYES]
- la classe [Cotisations] qui encapsule dans un objet les données d'une ligne de la table [COTISATIONS]
- la classe [Indemnites] qui encapsule dans un objet les données d'une ligne de la table [INDEMNITES]
- la classe [FeuilleSalaire] qui encapsule dans un objet les éléments constitutifs de la feuille de salaire affichée à l'écran.

1.6 La classe [PamException]

La couche [dao] va travailler avec l'API JDBC de Java. Cette API lance des exceptions contrôlées de type [SQLException] qui présentent deux inconvénients :
- elles alourdissent le code qui doit obligatoirement gérer ces exceptions avec des try / catch.
- elles doivent être déclarées dans la signature des méthodes de l'interface [IPamDao] par un "throws SQLException". Ceci a pour conséquence d'empêcher l'implémentation de cette interface par des classes qui lanceraient une exception contrôlée d'un type différent de [SQLException].

Pour remédier à ce problème, la couche [dao] ne "remontera" que des exceptions non contrôlées de type [PamException]. Ceci a deux conséquences :
- la couche [metier] n'aura pas à gérer les exceptions de la couche [dao] avec des try / catch. Elle pourra simplement les laisser remonter jusqu'à la couche [ui].

- les méthodes de l'interface [IPamDao] n'ont pas à mettre dans leur signature la nature de l'exception [PamException], ce qui laisse la possiblité d'implémenter cette interface avec des classes qui lanceraient un autre type d'exception non contrôlée.

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

```
1.  package istia.st.pam.dao;
2.
3.  public class PamException extends RuntimeException {
4.
5.          public PamException() {
6.                  super();
7.          }
8.
9.          public PamException(String message) {
10.                 super(message);
11.         }
12.
13.         public PamException(String message, Throwable cause) {
14.                 super(message, cause);
15.         }
16.
17.         public PamException(Throwable cause) {
18.                 super(cause);
19.         }
20.
21. }
```

- ligne 3 : noter que la classe [PamException] dérive de la classe [RuntimeException] ce qui en fait une exception non contrôlée.

Le fonctionnement de l'application, du point de vue des exceptions, sera donc le suivant :
- les couches [dao] et [metier] encapsulent toute exception contrôlée qu'elles peuvent être amenées à gérer, dans une exception de type [PamException] et lancent cette dernière pour la couche supérieure.
- au final, c'est la couche [ui] qui intercepte toutes les exceptions non contrôlées qui remontent des couches [metier] et [dao]. Elle se contentera d'afficher l'exception sur la console et arrêtera l'application.

Examinons maintenant successivement l'implémentation des couches [dao] et [metier].

2 La couche [dao] de l'application [PAM]

2.1 Les classes de "mapping" relationnel / objets

A chacune des tables de la base de données [dbpam], nous allons faire correspondre une classe chargée de mémoriser une ligne de la table. Ces classes sont appelées [Cotisations, Employe, Indemnites]. Nous les décrivons maintenant :

2.1.1 Classe [Cotisations]

Son code est le suivant :

```
1.  package istia.st.pam.dao;
2.
3.  public class Cotisations {
4.          // champs privés
5.          private double csgrds;
6.          private double csgd;
7.          private double secu;
8.          private double retraite;
9.
10.         // constructeurs
11.         public Cotisations() {
12.
13.         }
14.
15.         public Cotisations(double csgrds, double csgd, double secu, double retraite) {
16.                 setCsgrds(csgrds);
17.                 setCsgd(csgd);
18.                 setSecu(secu);
19.                 setRetraite(retraite);
20.         }
21.
22.
23.         // toString
24.         public String toString(){
25.                 return "["+csgrds+","+csgd+","+secu+","+retraite+"]";
26.         }
```

```
27.
28.         // accesseurs
29.         public double getCsgrds() {
30.                 return csgrds;
31.         }
32.
33.         public void setCsgrds(double csgrds) {
34.                 this.csgrds = csgrds;
35.         }
36.
37. ...
38. }
```

- la classe [Cotisations] représente une ligne de la table [COTISATIONS]
- lignes 5-8 : les quatre champs correspondant aux quatre colonnes de la table [COTISATIONS]
- lignes 10-20 : les deux constructeurs de la classe
- lignes 24-26 : la méthode [toString] de la classe
- lignes 29 et au-delà : les accesseurs (get / set) associés aux champs privés des lignes 5-8

2.1.2 Classe [Employe]

Son code est le suivant :

```
1.  package istia.st.pam.dao;
2.
3.  public class Employe {
4.          // champs privés
5.          private String num;
6.          private String nom;
7.          private String prenom;
8.          private String adresse;
9.          private String ville;
10.         private String codePostal;
11.         private int indice;
12.
13.         // constructeurs
14.         public Employe() {
15.
16.         }
17.
18.         public Employe(String num, String nom, String prenom, String adresse,
19.                         String ville, String codePostal, int indice) {
20.                 setNum(num);
21.                 setNom(nom);
22.                 setPrenom(prenom);
23.                 setAdresse(adresse);
24.                 setCodePostal(codePostal);
25.                 setVille(ville);
26.                 setIndice(indice);
27.         }
28.
29.         // toString
30.         public String toString() {
31.                 return "[" + num + "," + nom + "," + prenom + "," + adresse + ","
32.                                 + ville + "," + codePostal + "," + indice + "]";
33.         }
34.
35.         // accesseurs
36.         public String getAdresse() {
37.                 return adresse;
38.         }
39.
40.         public void setAdresse(String adresse) {
41.                 this.adresse = adresse;
42.         }
43. ...
44. }
```

- la classe [Employe] représente une ligne de la table [EMPLOYES]
- lignes 5-11 : les sept champs correspondant aux sept colonnes de la table [EMPLOYES]
- lignes 14-27 : les deux constructeurs de la classe
- lignes 30-33 : la méthode [toString] de la classe
- lignes 36 et au-delà : les accesseurs (get / set) associés aux champs privés des lignes 5-11

2.1.3 Classe [Indemnites]

Son code est le suivant :

```
1.  package istia.st.pam.dao;
2.
```

```
3.  public class Indemnites {
4.          // champs privés
5.          private int indice;
6.          private double baseHeure;
7.          private double entretienJour;
8.          private double repasJour;
9.          private double indemnitesCongesPayes;
10.
11.         // constructeurs
12.         public Indemnites() {
13.
14.         }
15.
16.         public Indemnites(int num, double baseHeure, double entretienJour,
17.                         double repasJour, double indemnitesCongesPayes) {
18.                 setIndice(num);
19.                 setBaseHeure(baseHeure);
20.                 setEntretienJour(entretienJour);
21.                 setRepasJour(repasJour);
22.                 setIndemnitesCongesPayes(indemnitesCongesPayes);
23.         }
24.
25.         // toString
26.         public String toString() {
27.                 return "[" + indice + "," + baseHeure + "," + entretienJour + ","
28.                                 + repasJour + "," + indemnitesCongesPayes + "]";
29.         }
30.
31.         // accesseurs
32.         public double getBaseHeure() {
33.                 return baseHeure;
34.         }
35.
36.         public void setBaseHeure(double baseHeure) {
37.                 this.baseHeure = baseHeure;
38.         }
39. ...
40. }
```

- la classe [Indemnites] représente une ligne de la table [INDEMNITES]
- lignes 5-9 : les cinq champs correspondant aux cinq colonnes de la table [INDEMNITES]
- lignes 12-23 : les deux constructeurs de la classe
- lignes 26-29 : la méthode [toString] de la classe
- lignes 31 et au-delà : les accesseurs (get / set) associés aux champs privés des lignes 5-11

2.2 La classe d'implémentation [PamDaoImplJdbc]

Une classe d'implémentation de l'interface [IPamDao] pourrait être la suivante :

```
1.  package istia.st.pam.dao;
2.
3.
4.  public class PamDaoImplJdbc implements IPamDao {
5.
6.          // champs privés
7.          private String driverClassName;
8.          private String jdbcUrl;
9.          private String userName;
10.         private String passwd;
11.
12.         // accesseurs
13.         public String getDriverClassName() {
14.                 return driverClassName;
15.         }
16.
17.         public void setDriverClassName(String driverClassName) {
18.                 this.driverClassName = driverClassName;
19.         }
20.
21. ...
22.
23.         // constructeurs
24.         public PamDaoImplJdbc() {
25.
26.         }
27.
28.         // constructeur avec paramètres
29.         public PamDaoImplJdbc(String driverClassName, String jdbcUrl, String userName,
30.                         String passwd) {
31.                 // on mémorise les paramètres
32. ...
33.                 // on charge le pilote du driver JDBC en mémoire
```

```
34. ...
35.         }
36.
37.         // obtention d'un employe
38.         public Employe getEmployeByNum(String num) {
39. ...
40.         }
41.
42.         // obtention des cotisations
43.         public Cotisations getCotisations() {
44. ...
45.         }
46.
47.         // obtentions des indemnités liées à un indice
48.         public Indemnites getIndemnitesByIndice(int indice) {
49. ...
50.         }
51. }
```

- ligne 4 : la classe [PamDaoImplJdbc] implémente l'interface [IPamDao]
- ligne 7 : le nom de la classe du pilote JDBC du SGBD utilisé
- ligne 8 : l'url de la base à exploiter
- ligne 9 : le nom de l'utilisateur qui va se connecter au SGBD
- ligne 10 : son mot de passe
- lignes 12-21 : les accesseurs des champs des lignes 7-10
- lignes 23-35 : les constructeurs de la classe
- lignes 38-48 : les méthodes implémentant l'interface [IPamDao].

Question 1 : écrire le code du constructeur des lignes 29-35. On se laissera guider par les commentaires.

Question 2 : écrire le code de la méthode [getEmployeByNum] :

- num est un n° de sécurité sociale de la table [EMPLOYES]
- la méthode doit rendre l'objet de type [Employe] correspondant à ce n° de sécurité sociale ou le pointeur [null] si aucune ligne de la table [EMPLOYES] ne correspond à ce n°. L'accès à la table [EMPLOYES] se fera à l'aide de l'API JDBC.

2.3 Tests de la couche [dao]

Nous supposerons désormais que la classe [PamDaoImplJdbc] a été entièrement écrite. Un programme de test de la couche [dao] pourrait être le suivant :

```
1.  package istia.st.pam.dao.tests;
2.
3.  import org.springframework.beans.factory.xml.XmlBeanFactory;
4.  import org.springframework.core.io.ClassPathResource;
5.
6.  import istia.st.pam.dao.IPamDao;
7.
8.  public class MainTestPamDaoJdbc {
9.
10.         public static void main(String[] args) {
11.                 IPamDao pamDao = (IPamDao) (new XmlBeanFactory(new ClassPathResource(
12.                                 "spring-config-PamDaoJdbc.xml"))).getBean("pamDao");
13.                 System.out.println("Employé="
14.                                 + pamDao.getEmployeByNum("260124402111742"));
15.                 System.out.println("Cotisations=" + pamDao.getCotisations());
16.                 System.out.println("Indemnités=" + pamDao.getIndemnitesByIndice(1));
17.         }
18. }
```

- lignes 11-12 : un objet implémentant l'interface [IPamDao] est demandé à Spring.
- ligne 13 : affiche l'identité d'un employé identifié pas son n° de sécurité sociale
- ligne 15 : affiche la liste des taux de cotisations sociales
- ligne 16 : affiche la liste des indemnités liées à un certain indice de traitement

Les résultats obtenus sur la console sont les suivants :

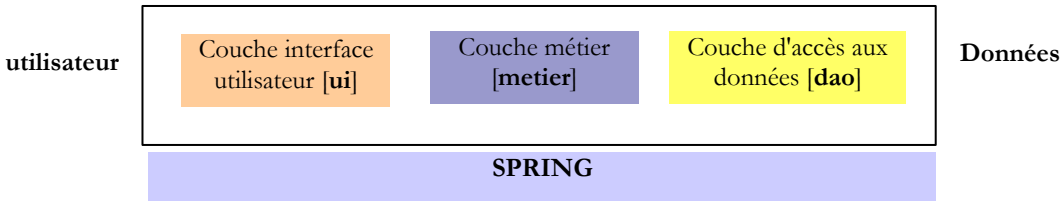
```
1.  Employé=[260124402111742,Laverti,Justine,la Brûlerie,St Marcel,49014,1]
2.  Cotisations=[3.49,6.15,9.39,7.88]
3.  Indemnités=[1,1.93,2.0,3.0,12.0]
```

Le lecteur est invité à vérifier l'exactitude de ces résultats en consultant le contenu des différentes tables de la base de données [dbpam] montré en début de document.

Question 3 : écrire le fichier de configuration Spring nommé [spring-config-PamDaoJdbc.xml] utilisé lignes 11-12 ci-dessus. Il doit permettre l'instanciation d'un objet de type [PamDaoImplJdbc].

3 La couche [metier] de l'application [PAM]

Maintenant que la couche [dao] a été écrite, nous passons à l'étude de la couche métier :



3.1 L'interface Java [IPamMetier]

Celle-ci a été décrite au paragraphe 1.5, page 5. Nous la rappelons ci-dessous :

```
1.  package istia.st.pam.metier;
2.
3.  public interface IPamMetier {
4.          // obtenir la feuille de salaire
5.          FeuilleSalaire calculerFeuilleSalaire(String numEmploye, double nbHeuresTravaillées, int
    nbJoursTravaillés );
6.  }
```

3.2 La classe [FeuilleSalaire]

La méthode [calculerFeuilleSalaire] rend un objet de type [FeuilleSalaire] qui représente les différents éléments d'une feuille de salaire. Sa définition est la suivante :

```
1.  package istia.st.pam.metier;
2.
3.  import istia.st.pam.dao.Cotisations;
4.  import istia.st.pam.dao.Employe;
5.  import istia.st.pam.dao.Indemnites;
6.
7.  public class FeuilleSalaire {
8.          // champs privés
9.          Employe employe;
10.         Cotisations cotisations;
11.         Indemnites indemnites;
12.         ElementsSalaire elementsSalaire;
13.
14.         // constructeurs
15.         public FeuilleSalaire() {
16.
17.         }
18.
19.         public FeuilleSalaire(Employe employe, Cotisations cotisations,
20.                         Indemnites indemnites, ElementsSalaire elementsSalaire) {
21.                 setEmploye(employe);
22.                 setCotisations(cotisations);
23.                 setElementsSalaire(elementsSalaire);
24.                 setIndemnites(indemnites);
25.         }
26.
27.         // toString
28.         public String toString() {
29.                 return "[" + employe + "," + cotisations + "," + indemnites + ","
30.                                 + elementsSalaire + "]";
31.         }
32.
33.         // accesseurs
34.         public Cotisations getCotisations() {
35.                 return cotisations;
36.         }
37.
38.         public void setCotisations(Cotisations cotisations) {
39.                 this.cotisations = cotisations;
```

```
40.         }
41.
42. ...
43. }
```

- ligne 9 : l'employé concerné par la feuille de salaire
- ligne 10 : les différents taux de cotisation
- ligne 11 : les différentes indemnités liées à l'indice de l'employé
- ligne 12 : les éléments constitutifs de son salaire
- lignes 14-25 : les deux constructeurs de la classe
- lignes 28-31 : méthode [toString] identifiant un objet [FeuilleSalaire] particulier
- lignes 34 et au-delà : les accesseurs publics aux champs privés de la classe

La classe [ElementsSalaire] référencée ligne 12 de la classe [FeuilleSalaire] ci-dessus, rassemble les éléments constituant une fiche de paie. Sa définition est la suivante :

```
1.  package istia.st.pam.metier;
2.
3.  public class ElementsSalaire {
4.
5.          // champs privés
6.          private double salaireBase;
7.          private double cotisationsSociales;
8.          private double indemnitesEntretien;
9.          private double indemnitesRepas;
10.         private double salaireNet;
11.
12.         // constructeurs
13.         public ElementsSalaire() {
14.
15.         }
16.
17.         public ElementsSalaire(double salaireBase, double cotisationsSociales,
18.                         double indemnitesEntretien, double indemnitesRepas,
19.                         double salaireNet) {
20.                 setSalaireBase(salaireBase);
21.                 setCotisationsSociales(cotisationsSociales);
22.                 setIndemnitesEntretien(indemnitesEntretien);
23.                 setIndemnitesRepas(indemnitesRepas);
24.         }
25.
26.         // toString
27.         public String toString() {
28.                 return "[" + salaireBase + "," + cotisationsSociales + ","
29.                                 + indemnitesEntretien + "," + indemnitesRepas + ","
30.                                 + salaireNet + "]";
31.         }
32.
33.         // accesseurs publics
34.         public double getCotisationsSociales() {
35.                 return cotisationsSociales;
36.         }
37.
38.         public void setCotisationsSociales(double cotisationsSociales) {
39.                 this.cotisationsSociales = cotisationsSociales;
40.         }
41. ...
42. }
```

- ligne 6 : le salaire de base
- ligne 7 : les cotisations sociales payées sur ce salaire de base
- ligne 8 : les indemnités journalières d'entretien de l'enfant
- ligne 9 : les indemnités journalières de repas de l'enfant
- ligne 10 : le salaire net à payer à l'assistante maternelle
- lignes 12-24 : les constructeurs de la classe
- lignes 27-31 : méthode [toString] identifiant un objet [ElementsSalaire] particulier
- lignes 34 et au-delà : les accesseurs publics aux champs privés de la classe

3.3 La classe d'implémentation [PamMetierImpl] de la couche [metier]

La classe d'implémentation [PamMetierImpl] de la couche [metier] pourrait être la suivante :

```
1.  package istia.st.pam.metier;
2.
3.  import istia.st.pam.dao.Cotisations;
4.  import istia.st.pam.dao.Employe;
5.  import istia.st.pam.dao.IPamDao;
```

```
6.  import istia.st.pam.dao.Indemnites;
7.  import istia.st.pam.dao.PamException;
8.
9.  public class PamMetierImpl implements IPamMetier {
10.
11.         // référence sur la couche [dao]
12.         private IPamDao pamDao = null;
13.
14.         public IPamDao getPamDao() {
15.                 return pamDao;
16.         }
17.
18.         public void setPamDao(IPamDao pamDao) {
19.                 this.pamDao = pamDao;
20.         }
21.
22.         // obtenir la feuille de salaire
23.         public FeuilleSalaire calculerFeuilleSalaire(String numEmploye,
24.                         double nbHeuresTravaillées, int nbJoursTravaillés) {
25. ....}
26. }
```

- ligne 12 : une référence sur un objet implémentant la couche [dao]
- lignes 14-20 : les accesseurs publics au champ [pamDao]
- lignes 23-25 : la méthode [calculerFeuilleSalaire]

Question 4 : écrire le code de la méthode [calculerFeuilleSalaire]. On notera les points suivants :

- lorsque cette méthode s'exécute, le champ [pamDao] a déjà été initialisé. Il pointe donc sur un objet donnant accès au contenu de la base de données [dbpam].
- le mode de calcul du salaire a été expliqué au paragraphe 1.2, page 2.
- si le paramètre [numEmploye] ne correspond à aucun employé (la couche [dao] a renvoyé un pointeur *null*), la méthode lancera une exception de type [PamException] avec un message d'erreur approprié

3.4 Tests de la couche [metier]

Un programme de tests de la couche métier pourrait être le suivant :

```
1.  package istia.st.pam.metier.tests;
2.
3.  import org.springframework.beans.factory.xml.XmlBeanFactory;
4.  import org.springframework.core.io.ClassPathResource;
5.
6.  import istia.st.pam.dao.PamException;
7.  import istia.st.pam.metier.IPamMetier;
8.
9.  public class MainTestMetierDaoJdbc {
10.
11.         public static void main(String[] args) {
12.                 // instanciation couche [metier]
13.                 IPamMetier pamMetier = (IPamMetier) (new XmlBeanFactory(
14.                                 new ClassPathResource(
15.                                                 "spring-config-PamMetierDaoJdbc.xml")))
16.                                 .getBean("pamMetier");
17.                 // calcul de feuilles de salaire
18.                 System.out.println(pamMetier.calculerFeuilleSalaire("260124402111742",
19.                                 30, 5));
20.                 System.out.println(pamMetier.calculerFeuilleSalaire("254104940426058",
21.                                 150, 20));
22.                 try {
23.                         System.out.println(pamMetier.calculerFeuilleSalaire(
24.                                         "xx", 150, 20));
25.                 } catch (PamException ex) {
26.                         System.err.println(ex.toString());
27.                 }
28.         }
29. }
```

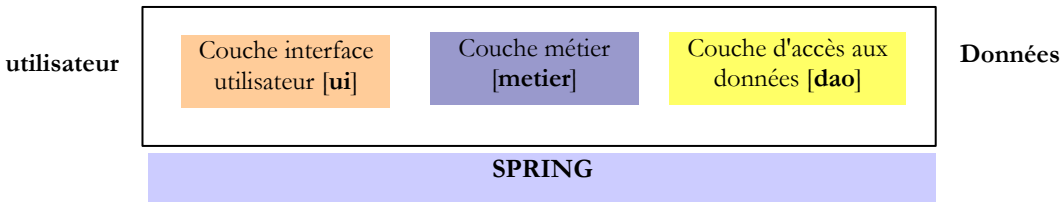
Les résultats obtenus sont les suivants :

```
1.  [[260124402111742,Laverti,Justine,la Brûlerie,St
    Marcel,49014,1],[3.49,6.15,9.39,7.88],[1,1.93,2.0,3.0,12.0],[64.85,17.45,10.0,15.0,72.4]]
2.  [[254104940426058,Jouveinal,Marie,5 rue des Oiseaux,St
    Corentin,49203,2],[3.49,6.15,9.39,7.88],[2,2.1,2.1,3.1,15.0],[362.25,97.48,42.0,62.0,368.77]]
3.  istia.st.pam.dao.PamException: L'employé de n°[xx] est introuvable
```

Question 5 : écrire le fichier de configuration Spring [spring-config-PamMetierDaoJdbc.xml] utilisé lignes 13-16 ci-dessus et qui instancie un objet de type [PamMetierImpl].

4 La couche [ui] de l'application [PAM]

Maintenant que la couche [metier] a été écrite, il ne nous reste plus qu'à écrire la couche [ui] :



La couche [ui] présentée à l'utilisateur est ici un programme console dont le fonctionnement a été décrit au paragraphe 1.3, page 3. Son squelette pourrait être le suivant :

```
1.  package istia.st.pam.ui.console;
2.
3.  import istia.st.pam.dao.PamException;
4.  import istia.st.pam.metier.FeuilleSalaire;
5.  import istia.st.pam.metier.IPamMetier;
6.
7.  import java.util.ArrayList;
8.
9.  import org.springframework.beans.factory.xml.XmlBeanFactory;
10. import org.springframework.core.io.ClassPathResource;
11.
12. public class MainPamWithJdbc {
13.
14.         public static void main(String[] args) {
15.                 // données locales
16.                 final String syntaxe = "pg num_securite_sociale nb_heures_travaillées
    nb_jours_travaillés";
17.                 // on vérifie le nombre de paramètres du tableau args
18. // à compléter ...
19.                 // le second paramètre args[1] doit être un nombre réel >0
20. // à compléter ...
21.                 // le troisième paramètre args[2] doit être un nombre entier >0
22. // à compléter ...
23.
24.                 // instanciation couche [metier]
25.                 IPamMetier pamMetier = (IPamMetier) (new XmlBeanFactory(
26.                                 new ClassPathResource(
27.                                                 "spring-config-PamMetierDaoJdbc.xml")))
28.                                 .getBean("pamMetier");
29.                 // calcul de la feuille de salaire
30. // à compléter ...
31.                 // affichage détaillé
32.                 String output = "Valeurs saisies :\n";
33.                 output += ajouteInfo("N° de sécurité sociale de l'employé", args[0]);
34.                 output += ajouteInfo("Nombre d'heures travaillées", args[1]);
35.                 output += ajouteInfo("Nombre de jours travaillés", args[2]);
36.                 output += ajouteInfo("\nInformations Employé", "");
37.                 output += ajouteInfo("Nom", feuilleSalaire.getEmploye().getNom());
38.                 output += ajouteInfo("Prénom", feuilleSalaire.getEmploye().getPrenom());
39.                 output += ajouteInfo("Adresse", feuilleSalaire.getEmploye()
40.                                 .getAdresse());
41.                 output += ajouteInfo("Ville", feuilleSalaire.getEmploye().getVille());
42.                 output += ajouteInfo("Code Postal", feuilleSalaire.getEmploye()
43.                                 .getCodePostal());
44.                 output += ajouteInfo("Indice", ""
45.                                 + feuilleSalaire.getEmploye().getIndice());
46.                 output += ajouteInfo("\nInformations Cotisations", "");
47.                 output += ajouteInfo("CSGRDS", ""
48.                                 + feuilleSalaire.getCotisations().getCsgrds() + " %");
49.                 output += ajouteInfo("CSGD", ""
50.                                 + feuilleSalaire.getCotisations().getCsgd() + " %");
51.                 output += ajouteInfo("Retraite", ""
52.                                 + feuilleSalaire.getCotisations().getRetraite() + " %");
53.                 output += ajouteInfo("Sécurité sociale", ""
54.                                 + feuilleSalaire.getCotisations().getSecu() + " %");
55.                 output += ajouteInfo("\nInformations Indemnités", "");
56.                 output += ajouteInfo("Salaire horaire", ""
57.                                 + feuilleSalaire.getIndemnites().getBaseHeure() + " euro");
58.                 output += ajouteInfo("Entretien/jour", ""
59.                                 + feuilleSalaire.getIndemnites().getEntretienJour() + " euro");
60.                 output += ajouteInfo("Repas/jour", ""
61.                                 + feuilleSalaire.getIndemnites().getRepasJour() + " euro");
```

```
62.                output += ajouteInfo("Congés Payés", ""
63.                              + feuilleSalaire.getIndemnites().getIndemnitesCongesPayes()
64.                              + " %");
65.                output += ajouteInfo("\nInformations Salaire", "");
66.                output += ajouteInfo("Salaire de base", ""
67.                              + feuilleSalaire.getElementsSalaire().getSalaireBase()
68.                              + " euro");
69.                output += ajouteInfo("Cotisations sociales", ""
70.                              + feuilleSalaire.getElementsSalaire().getCotisationsSociales()
71.                              + " euro");
72.                output += ajouteInfo("Indemnités d'entretien", ""
73.                              + feuilleSalaire.getElementsSalaire().getIndemnitesEntretien()
74.                              + " euro");
75.                output += ajouteInfo("Indemnités de repas", ""
76.                              + feuilleSalaire.getElementsSalaire().getIndemnitesRepas()
77.                              + " euro");
78.                output += ajouteInfo("Salaire net", ""
79.                              + feuilleSalaire.getElementsSalaire().getSalaireNet() + " euro");
80.
81.                System.out.println(output);
82.        }
83.
84.        static String ajouteInfo(String message, String valeur) {
85.                return message + " : " + valeur + "\n";
86.        }
87. }
```

Question 6 : compléter le code ci-dessus (lignes 18, 20, 22, 30)

Table des matières