TD JAVA

Mots clés : Bases de données, JDBC

1.1 Présentation

Considérons une application web Java permettant de gérer un groupe de personnes avec les quatre opérations suivantes :

- liste des personnes du groupe
- ajout d'une personne au groupe
- modification d'une personne du groupe
- suppression d'une personne du groupe

On reconnaîtra là les quatre opérations de base sur une table de base de données. Les copies d'écran qui suivent montrent les pages que l'application échange avec l'utilisateur.

MVC - Personnes

http://localhost:8080/personnes-01/do/list;jsessionid=6650A3BD8A69262A65C6BA288329F011

Liste des personnes

Id	Version	Prénom	Nom	Date de naissance	Marié	Nombre d'enfants		
1	1	Joachim	Major	13/11/1984	true	2	Modifier	Supprimer
2	1	Mélanie	Humbort	12/02/1985	false	1	Modifier	Supprimer
3	1	Charles	Lemarchand	01/03/1986	false	0	Modifier	Supprimer

Ajout

Une liste initiale de personnes est tout d'abord présentée à l'utilisateur. Il peut ajouter une personne ->

MVC - Personnes

http://localhost:8080/personnes-01/do/edit?id=-1

Ajout/Modification d'une personne

Id	-1
Version	0
Prénom	Sophie
Nom	Maxima
Date de naissance (JJ/MM/AAAA)	13/03/1946
Marié	Oui Non
Nombre d'enfants	4

Valider Annuler

L'utilisateur a créé une nouvelle personne qu'il valide avec le bouton [Valider] ->

MVC - Personnes

http://localhost:8080/personnes-01/do/list

Liste des personnes

Id	Version	Prénom	Nom	Date de naissance	Marié	Nombre d'enfants		
1	1	Joachim	Major	13/11/1984	true	2	Modifier	Supprimer
2	1	Mélanie	Humbort	12/02/1985	false	1	Modifier	Supprimer
3	1	Charles	Lemarchand	01/03/1986	false	0	Modifier	Supprimer
4	1	Sophie	Maxima	13/03/1946	true	4	Modifier	Supprimer

Ajout

La nouvelle personne a été ajoutée. On la modifie maintenant ->

MVC - Personnes

http://localhost:8080/personnes-01/do/edit?id=4

Ajout/Modification d'une personne

Id	4
Version	1
Prénom	Sophie
Nom	Maxima
Date de naissance (JJ/MM/AAAA)	13/03/1956
Marié	○ Oui ◉ Non
Nombre d'enfants	2

Valider Annuler

On modifie la date de naissance, l'état marital, le nombre d'enfants et on valide ->

MVC - Personnes

http://localhost:8080/personnes-01/do/list

Liste des personnes

Id	Version	Prénom	Nom	Date de naissance	Marié	Nombre d'enfants		
1	1	Joachim	Major	13/11/1984	true	2	Modifier	Supprimer
2	1	Mélanie	Humbort	12/02/1985	false	1	Modifier	Supprimer
3	1	Charles	Lemarchand	01/03/1986	false	0	Modifier	Supprimer
4	2	Sophie	Maxima	13/03/1956	false	2	Modifier	Supprimer

Ajout

On retouve la personne telle qu'elle a été modifiée. On la supprime maintenant ->

MVC - Personnes

http://localhost:8080/personnes-01/do/list

Liste des personnes

Id	Version	Prénom	Nom	Date de naissance	Marié	Nombre d'enfants		
1	1	Joachim	Major	13/11/1984	true	2	Modifier	Supprimer
2	1	Mélanie	Humbort	12/02/1985	false	1	Modifier	Supprimer
3	1	Charles	Lemarchand	01/03/1986	false	0	Modifier	Supprimer

Ajout

Elle n'est plus là. On ajoute une personne ->

MVC - Personnes

http://localhost:8080/personnes-01/do/edit?id=-1

Ajout/Modification d'une personne

Id	-1
Version	0
Prénom	
Nom	
Date de naissance (JJ/MM/AAAA)	
Marié	○ Oui ◉ Non
Nombre d'enfants	xx

Valider Annuler

Les erreurs de saisie sont signalées ->

MVC - Personnes

http://localhost:8080/personnes-01/do/validate

Ajout/Modification d'une personne

Id	-1	
Version	0	
Prénom		Le prénom est obligatoire
Nom		Le nom est obligatoire
Date de naissance (JJ/MM/AAAA)		Date incorrecte
Marié	○ Oui ◉ Non	
Nombre d'enfants	xx	Nombre d'enfants incorrect

Valider Annuler

On notera que le formulaire a été renvoyé tel qu'il a été saisi (Nombre d'enfants). Le lien [Annuler] permet de revenir à la liste des personnes ->

MVC - Personnes

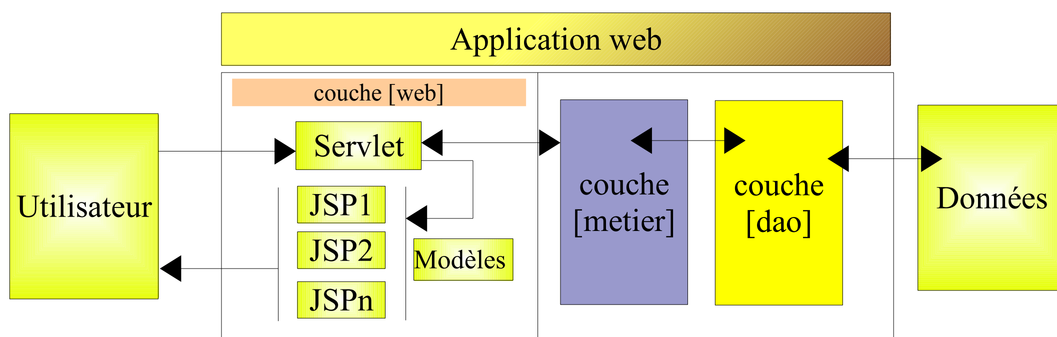
http://localhost:8080/personnes-01/do/list

Liste des personnes

Id	Version	Prénom	Nom	Date de naissance	Marié	Nombre d'enfants		
1	1	Joachim	Major	13/11/1984	true	2	Modifier	Supprimer
2	1	Mélanie	Humbort	12/02/1985	false	1	Modifier	Supprimer
3	1	Charles	Lemarchand	01/03/1986	false	0	Modifier	Supprimer

Ajout

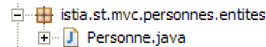
L'application web a l'architecture 3tier suivante :



Nous souhaitons écrire la couche [dao] qui interagit avec la base de données contenant les personnes. Il n'y a aucune notion de programmation web dans cette couche, seulement des accès JDBC aux données.

1.2 La représentation d'une personne

L'application gère un groupe de personnes. Les copies d'écran page 1 ont montré certaines des caractéristiques d'une personne. Formellement, celles-ci sont représentées par une classe [Personne] :



La classe [Personne] est la suivante :

```
1.  package istia.st.mvc.personnes.entites;
2.
3.  import java.text.SimpleDateFormat;
4.  import java.util.Date;
5.
6.  public class Personne {
7.
8.   // identifiant unique de la personne
9.   private int id;
10.  // la version actuelle
11.  private long version;
12.  // le nom
13.  private String nom;
14.  // le prénom
15.  private String prenom;
16.  // la date de naissance
17.  private Date dateNaissance;
18.  // l'état marital
19.  private boolean marie = false;
20.  // le nombre d'enfants
21.  private int nbEnfants;
22.
23.  // getters - setters
24. ...
25.
26.  // constructeur par défaut
27.  public Personne() {
28.
29.  }
30.
31.  // constructeur avec initialisation des champs de la personne
32.  public Personne(int id, long version, String prenom, String nom, Date dateNaissance,
33.      boolean marie, int nbEnfants) {
34.    setId(id);
35.    setVersion(version);
36.    setNom(nom);
37.    setPrenom(prenom);
38.    setDateNaissance(dateNaissance);
39.    setMarie(marie);
40.    setNbEnfants(nbEnfants);
41.  }
42.
43.  // constructeur d'une personne par recopie d'une autre personne
44.  public Personne(Personne p) {
45.    setId(p.getId());
46.    setVersion(p.getVersion());
47.    setNom(p.getNom());
48.    setPrenom(p.getPrenom());
49.    setDateNaissance(p.getDateNaissance());
50.    setMarie(p.getMarie());
51.    setNbEnfants(p.getNbEnfants());
52.  }
53.
54.
55.  // toString
56.  public String toString() {
57.    return "[" + id + "," + version + "," + prenom + "," + nom + ","
58.        + new SimpleDateFormat("dd/MM/yyyy").format(dateNaissance)
59.        + "," + marie + "," + nbEnfants + "]";
60.  }
61. }
```

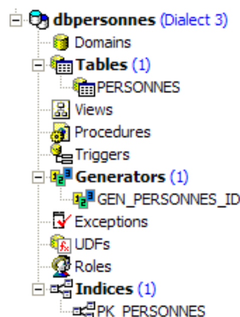
- une personne est identifiée par les informations suivantes :
  - **id** : un n° identifiant de façon unique une personne
  - **nom** : le nom de la personne
  - **prenom** : son prenom
  - **dateNaissance** : sa date de naissance
  - **marie** : son état marié ou non
  - **nbEnfants** : son nombre d'enfants

- **version** : n° de version de la personne. Lors de la création initiale de la personne, ce n° vaut 1. Il est incrémenté de 1 à chaque fois que la personne est modifiée.

- lignes 32-40 : un constructeur capable d'initialiser les champs d'une personne.
- lignes 43-51 : un constructeur qui crée une copie de la personne qu'on lui passe en paramètre. On a alors deux objets de contenu identique mais référencés par deux pointeurs différents.
- ligne 55 : la méthode [toString] est redéfinie pour rendre une chaîne de caractères représentant l'état de la personne

1.3 La base de données Firebird

La liste des personnes est dans une table nommée [PERSONNES] d'une base de données Firebird nommée [dbpersonnes.gdb].



La table [PERSONNES] contient la liste des personnes gérée par l'application web. Elle a été construite avec les ordres SQL suivants :

```
1.
2.  CREATE TABLE PERSONNES (
3.      ID              INTEGER NOT NULL,
4.      "VERSION"       INTEGER NOT NULL,
5.      NOM             VARCHAR(30) NOT NULL,
6.      PRENOM          VARCHAR(30) NOT NULL,
7.      DATENAISSANCE   DATE NOT NULL,
8.      MARIE           SMALLINT NOT NULL,
9.      NBENFANTS       SMALLINT NOT NULL
10. );
11.
12.
13. ALTER TABLE PERSONNES ADD CONSTRAINT CHK_PRENOM_PERSONNES check (PRENOM<>'');
14. ALTER TABLE PERSONNES ADD CONSTRAINT CHK_MARIE_PERSONNES check (MARIE=0 OR MARIE=1);
15. ALTER TABLE PERSONNES ADD CONSTRAINT CHK_NOM_PERSONNES check (NOM<>'');
16. ALTER TABLE PERSONNES ADD CONSTRAINT CHK_ENFANTS_PERSONNES check (NBENFANTS>=0);
17.
18.
19. ALTER TABLE PERSONNES ADD CONSTRAINT PK_PERSONNES PRIMARY KEY (ID);
```

- lignes 2-10 : la structure de la table [PERSONNES], destinée à sauvegarder des objets de type [Personne], reflète la structure de cet objet. Le type booléen n'existant pas dans Firebird, le champ [MARIE] (ligne 8) a été déclaré de type [SMALLINT], un entier. Sa valeur sera 0 (pas marié) ou 1 (marié).
- lignes 13-16 : des contraintes d'intégrité.
- ligne 19 : le champ ID est clé primaire de la table [PERSONNES]

La table [PERSONNES] pourrait avoir le contenu suivant :

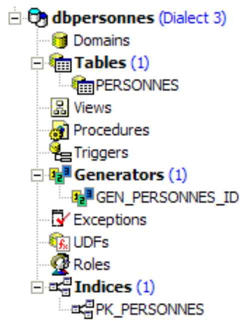
Table | Get record count | PERSONNES

Fields | Constraints | Indices | Dependencies | Triggers | Data | Description | DDL | Grants

Record: 3

ID	VERSION	NOM	PRENOM	DATENAISSANCE	MARIE	NBENFANTS
1	1	Major	Joachim	13.11.1984	1	2
2	1	Humbort	Mélanie	12.02.1985	0	1
3	1	Lemarchand	Charles	01.03.1986	0	0

La base [dbpersonnes.gdb] a, outre la table [PERSONNES], un objet appelé **générateur** et nommé [GEN_PERSONNES_ID]. Ce générateur délivre des nombres entiers successifs que nous utiliserons pour donner sa valeur, à la clé primaire [ID] de la classe [PERSONNES]. Prenons un exemple pour illustrer son fonctionnement :

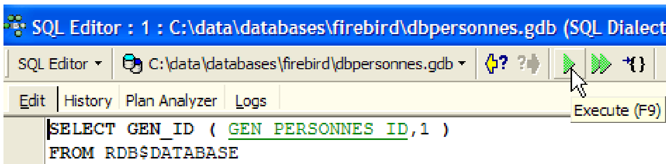


- les objets de la base [dbpersonne.gdb]

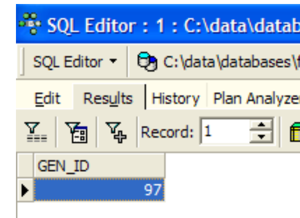
Generators | Dependencies | DDL | Scripts

Name	Value
GEN_PERSONNES_ID	96

- le générateur a actuellement la valeur 96



- émettons l'ordre SQL ci-dessus (F12) ->



- la valeur obtenue est l'ancienne valeur du générateur +1

On peut constater que la valeur du générateur [GEN_PERSONNES_ID] a changé :

Generators | Dependencies | DDL | Scripts

Name	Value
GEN_PERSONNES_ID	97

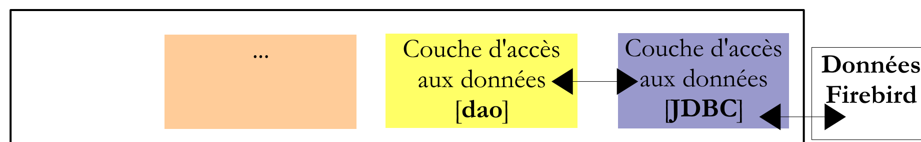
L'ordre SQL

```
SELECT GEN_ID ( GEN_PERSONNES_ID,1 ) FROM RDB$DATABASE
```

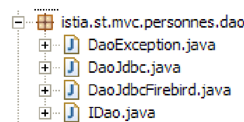
permet donc d'avoir la valeur suivante du générateur [GEN_PERSONNES_ID]. GEN_ID est une fonction interne de Firebird et [RDB$DATABASE], une table système de ce SGBD.

1.4 La couche [dao]

La couche [dao] est la couche qui dialogue avec la base de données via le pilote JDBC du SGBD Firebird :



Elle est constituée des classes et interfaces suivantes :



- [IDao] est l'interface présentée par la couche [dao]
- [DaoJdbc] est une implémentation de celle-ci pour gérer le groupe de personnes lorsqu'il se trouve dans une base de données.
- [DaoJdbcFirebird] est une implémentation particulière à Firebird. Cette classe sera ici ignorée. Seule la classe [DaoJdbc] sera écrite et on considèrera qu'elle est valide pour tout type de SGBD.

- [DaoException] est un type d'exceptions non contrôlées (unchecked), lancées par la couche [dao]

L'interface [IDao] est la suivante :

```
1.  package istia.st.mvc.personnes.dao;
2.
3.  import istia.st.mvc.personnes.entites.Personne;
4.
5.  import java.util.Collection;
6.
7.  public interface IDao {
8.    // liste de toutes les personnes
9.    Collection<Personne> getAll();
10.   // obtenir une personne particulière
11.   Personne getOne(int id);
12.   // ajouter/modifier une personne
13.   void saveOne(Personne personne);
14.   // supprimer une personne
15.   void deleteOne(int id);
16. }
```

- l'interface a quatre méthodes pour les quatre opérations que l'on souhaite faire sur le groupe de personnes :
  - **getAll** : pour obtenir une collection de personnes
  - **getOne** : pour obtenir une personne ayant un *id* précis
  - **saveOne** : pour ajouter une personne (*id*=-1) ou modifier une personne existante (*id* <> -1)
  - **deleteOne** : pour supprimer une personne ayant un *id* précis

La couche [dao] est susceptible de lancer des exceptions. Celles-ci seront de type [DaoException] :

```
1.  package istia.st.mvc.personnes.dao;
2.
3.  public class DaoException extends RuntimeException {
4.
5.    // code erreur
6.    private int code;
7.
8.    public int getCode() {
9.      return code;
10.   }
11.
12.   // constructeur
13.   public DaoException(String message,int code) {
14.     super(message);
15.     this.code=code;
16.   }
17. }
```

- ligne 3 : la classe [DaoException] dérivant de [RuntimeException] est un type d'exception non contrôlée : le compilateur ne nous oblige pas à :
  - gérer ce type d'exceptions avec un try / catch lorsqu'on appelle une méthode pouvant la lancer
  - mettre le marqueur " throws DaoException " dans la signature d'une méthode susceptible de lancer l'exception

  Cette technique nous évite d'avoir à signer les méthodes de l'interface [IDao] avec des exceptions d'un type particulier. Toute implémentation lançant des exceptions non contrôlées sera alors acceptable amenant ainsi de la souplesse dans l'architecture.

- ligne 6 : un code d'erreur. La couche [dao] lancera diverses exceptions qui seront identifiées par des codes d'erreur différents. Cela permettra à la couche qui décidera de gérer l'exception de connaître l'origine exacte de l'erreur et de prendre ainsi les mesures appropriées. Il y a d'autres façons d'arriver au même résultat. L'une d'elles est de créer un type d'exception pour chaque type d'erreur possible, par exemple NomManquantException, PrenomManquantException, AgeIncorrectException, ...

- lignes 13-16 : le constructeur qui permettra de créer une exception identifiée par un code d'erreur ainsi qu'un message d'erreur.

- lignes 8-10 : la méthode qui permettra au code de gestion d'une exception d'en récupérer le code d'erreur.

La classe [DaoJdbc] implémente l'interface [IDao] lorsque les personnes sont dans la table [PERSONNES] décrite au paragraphe 1.3, page 5.

1.5 Tests de la couche [dao]

Une application de test JUnit de la classe [DaoJdbc] est la suivante :

```
1.  package istia.st.mvc.personnes.tests;
2.
3.  import java.text.ParseException;
4.  import java.text.SimpleDateFormat;
5.  import java.util.Collection;
6.  import java.util.Iterator;
7.
8.  import org.springframework.beans.factory.xml.XmlBeanFactory;
9.  import org.springframework.core.io.ClassPathResource;
10.
11. import istia.st.mvc.personnes.dao.DaoException;
12. import istia.st.mvc.personnes.dao.IDao;
13. import istia.st.mvc.personnes.entites.Personne;
14. import junit.framework.TestCase;
15.
16. public class TestDao extends TestCase {
17.
18.   // couche [dao]
19.   private IDao dao;
20.
21.   public IDao getDao() {
22.     return dao;
23.   }
24.
25.   public void setDao(IDao dao) {
26.     this.dao = dao;
27.   }
28.
29.   // constructeur
30.   public void setUp() {
31.     dao = (IDao) (new XmlBeanFactory(new ClassPathResource(
32.                 "spring-config.xml"))).getBean("dao");
33.   }
34.
35.   // liste des personnes
36.   private void doListe(Collection<Personne> personnes) {
37.     for(Personne p : personnes){
38.       System.out.println(p);
39.     }
40.   }
41.
42.   // test1
43.   public void test1() throws ParseException {
44.     // liste actuelle
45.     Collection personnes = dao.getAll();
46.     int nbPersonnes = personnes.size();
47.     // affichage
48.     doListe(personnes);
49.     // ajout d'une personne
50.     Personne p1 = new Personne(-1, "X", "X", new SimpleDateFormat(
51.                 "dd/MM/yyyy").parse("01/02/2006"), true, 1);
52.     dao.saveOne(p1);
53.     int id1 = p1.getId();
54.     // vérification - on aura un plantage si la personne n'est pas trouvée
55.     p1 = dao.getOne(id1);
56.     assertEquals("X", p1.getNom());
57.     // modification
58.     p1.setNom("Y");
59.     dao.saveOne(p1);
60.     // vérification - on aura un plantage si la personne n'est pas trouvée
61.     p1 = dao.getOne(id1);
62.     assertEquals("Y", p1.getNom());
63.     // suppression
64.     dao.deleteOne(id1);
65.     // vérification
66.     int codeErreur = 0;
67.     boolean erreur = false;
68.     try {
69.         p1 = dao.getOne(id1);
70.     } catch (DaoException ex) {
71.         erreur = true;
72.         codeErreur = ex.getCode();
73.     }
74.     // on doit avoir une erreur de code 2
75.     assertTrue(erreur);
76.     assertEquals(2, codeErreur);
77.     // liste des personnes
78.     personnes = dao.getAll();
79.     assertEquals(nbPersonnes, personnes.size());
80.   }
```

```
81.
82.  // modification-suppression d'un élément inexistant
83.  public void test2() throws ParseException {
84.    // d'abord ajout
85.    Personne p1 = new Personne(-1, "X", "X", new SimpleDateFormat(
86.        "dd/MM/yyyy").parse("01/02/2006"), true, 1);
87.    dao.saveOne(p1);
88.    // récupération copie de la personne
89.    Personne p2 = dao.getOne(p1.getId());
90.    // modification copie avec un id inexistant
91.    p2.setId(p1.getId() + 1);
92.    // modification nom de la copie
93.    p2.setNom("Y");
94.    // sauvegarde copie - on doit avoir une DaoException de code 2
95.    boolean erreur = false;
96.    int codeErreur = 0;
97.    try {
98.      dao.saveOne(p2);
99.    } catch (DaoException ex) {
100.     erreur = true;
101.     codeErreur = ex.getCode();
102.   }
103.   // vérification - on doit avoir une erreur de code 2
104.   assertTrue(erreur);
105.   assertEquals(2, codeErreur);
106.   // suppression élément inexistant - on doit avoir une DaoException de
107.   // code 2
108.   erreur = false;
109.   codeErreur = 0;
110.   try {
111.     dao.deleteOne(-2);
112.   } catch (DaoException ex) {
113.     erreur = true;
114.     codeErreur = ex.getCode();
115.   }
116.   // vérification - on doit avoir une erreur de code 2
117.   assertTrue(erreur);
118.   assertEquals(2, codeErreur);
119.   // suppression personne p1
120.   dao.deleteOne(p1.getId());
121.   // vérification
122.   try {
123.     p1 = dao.getOne(p1.getId());
124.   } catch (DaoException ex) {
125.     erreur = true;
126.     codeErreur = ex.getCode();
127.   }
128.   // on doit avoir une erreur de code 2
129.   assertTrue(erreur);
130.   assertEquals(2, codeErreur);
131. }
132.
133. // gestion des versions de personne
134. public void test3() throws ParseException, InterruptedException {
135.   // d'abord ajout
136.   Personne p1 = new Personne(-1, "X", "X", new SimpleDateFormat(
137.       "dd/MM/yyyy").parse("01/02/2006"), true, 1);
138.   dao.saveOne(p1);
139.   // récupération copie p2 de la personne p1
140.   Personne p2 = dao.getOne(p1.getId());
141.   // récupération copie p3 de la personne p1
142.   Personne p3 = dao.getOne(p1.getId());
143.   // on vérifie qu'on a bien la même version
144.   assertEquals(p2.getVersion(), p3.getVersion());
145.   // attente 10 ms
146.   Thread.sleep(10);
147.   // sauvegarde copie p2 - la version de p1 va changer
148.   dao.saveOne(p2);
149.   // sauvegarde copie p3 - on doit avoir une DaoException de code 2
150.   boolean erreur = false;
151.   int codeErreur = 0;
152.   try {
153.     dao.saveOne(p3);
154.   } catch (DaoException ex) {
155.     erreur = true;
156.     codeErreur = ex.getCode();
157.   }
158.   // vérification - on doit avoir une erreur de code 2
159.   assertTrue(erreur);
160.   assertEquals(codeErreur, 2);
161.   // suppression personne p1
162.   dao.deleteOne(p1.getId());
163.   // vérification
164.   try {
165.     p1 = dao.getOne(p1.getId());
166.   } catch (DaoException ex) {
167.     erreur = true;
```

```
168.     codeErreur = ex.getCode();
169.   }
170.   // on doit avoir une erreur de code 2
171.   assertTrue(erreur);
172.   assertEquals(2, codeErreur);
173. }
174....
175.}
```

Nous n'avons reproduit que trois méthodes de test. Elles doivent vous permettre de déterminer

- comment appeler les méthodes de l'interface [IDao]
- quels sont les résultats attendus de chacune d'elles
- quelles exceptions chacune d'elles est susceptible de lancer

La méthode [setUp] des lignes 30-33 utilisent un fichier Spring pour créer l'instance de la classe [DaoJdbc] qui va être testée. Ce n'est pas une obligation. La méthode [setUp] aurait pu créer cette instance en faisant appel au constructeur de la classe [DaoJdbc].

Travail à faire

Écrire le code Java de la classe [DaoJdbc] implémentant l'interface [IDao] :

- définir les champs privés et le constructeur
- définir chacune des méthodes implémentant l'interface [IDao]. Pour chacune d'elles vous vous aiderez des tests présentés au paragraphe 1.5, page 8, qui doivent vous permettre de déterminer comment appeler ces méthodes et quels résultats en attendre.

L'essentiel de la classe [DaoJdbc] consiste à utiliser l'API JDBC pour accéder au contenu d'une base de données.

Table des matières

1.1 Présentation..........1
1.2 La représentation d'une personne..........3
1.3 La base de données Firebird..........4
1.4 La couche [dao]..........6
1.5 Tests de la couche [dao]..........7