J2EE
TP 3

BRIZAI Olivier
THORAVAL Maxime

1 Utilisation de Spring

Dans la première partie du TP, nous avons appris à utiliser les bases de Spring. Ce dernier va nous permettre de dissocier, du code Java, les classes de service et les DAO. Ceci aura pour effet une plus grande facilité à changer de méthode récupération des données.
Par exemple : Nous pourrons décider de passer de l'utilisation d'une base de données à celle de fichiers juste en modifiant un fichier XML (en supposant que les classes nécessaires aient été créées au préalable).

1.1 DAO

Avant d'utiliser Spring, la récupération d'une classe DAO se faisait de cette manière :

```
1        DAOImpl dao = new DAOImpl();
2        dao.init();
```

Ici, nous avons instancié un objet de type *DAOImpl* et l'avons initialisé.
Le principal problème est lié au fait que nous utilisons une classe définie. Si l'on décide de changer le nom du DAO (nouveau DAO), il faudra modifier le code en conséquence. L'utilisation de Spring va ainsi eviter plusieurs recompilations en cas de changement du dao. La couche web et le couche service ne seront en effet pas à recompiler.

Voici le nouveau code lorsque l'on utilise Spring :

```
1        IDAO dao =(IDAO) (new XmlBeanFactory(new
            ClassPathResource("spring-config.xml"))).
            getBean("dao");
```

L'identifiant "dao" fait ici appel au bean définie dans le fichier spring-config.xml :

```
1 <bean id="dao" class="ensicaen.tb.mvc.eleves.dao.DAOImpl"
      destroy-method="destroy" init-method="init">
2 </bean>
```

Le XML va permettre d'instancier le bean, tandis que c'est le java qui l'utilisera. On remarque que l'on ignore dans le code java de quelle implémentation du IDAO

il s'agit. Ce choix est fait dans le XML et permet de diminuer les dépendances entre les couches.

On vient de mettre en place Spring pour la couche DAO et on fait de même pour la couche service grâce à un bean "seervice qui représentera une instance de notre classe de service.

Dans le fichier Application.java, on appellera alors l'instance de service ainsi :

```
1 service =(IService) (new XmlBeanFactory(new
     ClassPathResource("spring-config.xml"))).getBean("
     service");
```

On remarque que l'on fait appelle à un IService et non plus à une ServiceImpl, ce qui limite les dépendances (comme pour le DAO).

2 Utilisation d'iBatis

iBatis est une nouvelle couche qui vient s'insérer entre le DAO et la couche d'accès aux données (JDBC).

Le nouveau DAO appellé DAOImplCommon est définit ainsi :

```
1 public class DAOImplCommon extends SqlMapClientDaoSupport
      implements IDAO {
```

Il remplace la précédente classe DAOImpl. Il implémente naturellement l'interface IDAO, mais surtout il hérite de SqlMapClientSupport qui est une classe de la bibliothèque iBatis.

Dans le fichier spring-config.xml on définit la classe DAO ainsi :

```
1 <!-- La classe DAO -->
2         <bean id="dao" class="ensicaen.tb.mvc.eleves.dao.
             DAOImplCommon">
3               <property name="sqlMapClient" ref="
                   sqlMapClient" />
4         </bean>
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org//DTD SQL
       Map 2.0//EN"
4          "http://ibatis.apache.org/dtd/sql-map-2.dtd">
5
6  <sqlMap>
7          <!--  alias classe [Eleve] -->
8          <typeAlias alias="Eleve.classe" type = "ensicaen.
               tb.mvc.eleves.entities.Eleve"/>
9
10         <!--  mapping table [ELEVES] -objet [Eleve] -->
11         <resultMap id="Eleve.map" class="Eleve.classe">
12                 <result property="id" column="id"/>
```

```
13                <result property="version" column="
                     version"/>
14                <result property="nom" column="nom"/>
15                <result property="prenom" column="prenom"
                     />
16                <result property="dateNaissance" column="
                     datenaissance"/>
17                <result property="redoublant" column="
                     redoublant"/>
18                <result property="annee" column="annee"/>
19                <result property="filiere" column="
                     filiere"/>
20          </resultMap>
21
22          <!--  liste de tous les eleves -->
23          <select id="Eleve.getAll" resultMap="Eleve.map">
24                SELECT * FROM ELEVES
25          </select>
26
27          <!--  obtenir un eleve en particulier  -->
28          <select id="Eleve.getOne" parameterClass="int"
               resultMap="Eleve.map">
29                SELECT * FROM ELEVES WHERE id=#value#
30          </select>
31
32          <select id="Eleve.nbEleve" resultClass="int">
33                SELECT count(*) FROM ELEVES
34          </select>
35
36          <!-- ajouter un eleve -->
37          <insert id="Eleve.insertionOne" parameterClass="
               Eleve.classe">
38                <selectKey keyProperty="id">
39                      SELECT nextval('SEQ_ELEVES') as
                           value
40                </selectKey>
41                INSERT INTO ELEVES values (#id#, 1, #nom#
                     , #prenom#, #dateNaissance#, #
                     redoublant#, #annee#, #filiere#)
42          </insert>
43
44          <!-- mettre    jour un   lve   -->
45          <update id="Eleve.updateOne" parameterClass="
               Eleve.classe">
46                UPDATE ELEVES SET version = #version#,
                     nom = #nom#, prenom = #prenom#,
                     dateNaissance = #dateNaissance#,
47                 annee = #annee#, redoublant = #
                      redoublant#, filiere = #filiere#
                      WHERE id = #id#
```

```
48          </update>
49
50          <!-- supprimer un    lve    -->
51          <delete id="Eleve.deleteOne" parameterClass="int"
               >
52                  DELETE FROM ELEVES WHERE id = #id#
53          </delete>
54
55  </sqlMap>
```