Sujet de TP 4
Framework JEE
EJB et persistance des données avec DAO

Chouki Tibermacine - chouki.tibermacine@umontpellier.fr

Objectifs du TP

- Rendre persistantes les données manipulées par les EJB
- Déployer une application JEE avec accès à source de données persistantes

Exercice

- L'objectif de cet exercice est de modifier l'application écrite dans le TP EJB précédent pour que les objets monnaies soient stockées dans une base de données
- Créer une nouvelle application JEE et y importer les fichiers du TP JEE MDB
- Modifier le bean session pour stocker dans une base de données pour chaque monnaie, son code, son nom complet, la liste des pays qui l'utilisent, le taux de change de l'euro vers cette monnaie et la date et l'heure du jour
- Côté bean session, nous allons juste ajouter une instruction dans la méthode `getAvailableCurrencies()` pour ordonner le stockage dans la base de chaque monnaie collectée
- Cette méthode sera désormais invoquée chaque matin à 6h par un client lourd, à définir aussi, qui est exécuté par une tâche CRON. Les autres méthodes du bean s'appuieront ensuite uniquement sur les données de la base
- Nous allons implémenter ici le patron DAO (Data Access Object) pour gérer la persistance des données. Ce patron préconise la définition d'une classe `DAOMonnaie`, qui implémente une interface avec des méthodes CRUD, qui centralisent le code technique lié à l'accès à la source de données persistantes (ici c'est une BdD relationnelle, donc les requêtes SQL, le code déclarant le driver JDBC utilisé, ... sont centralisés ici). Ces méthodes doivent avoir des signatures qui ne comportent aucun type technique lié au système de stockage utilisé (aucun type issu de `java.sql`). Un exemple de méthode CRUD est donné ici :

```
public Monnaie getMonnaieByCode(String code) {
  Monnaie m = null;
  if(code == null || code.length() == 0)
    return m;
  PreparedStatement statement = null;
```

```
  try {
    String sql = "SELECT * FROM monnaies WHERE code=?
      and date in (SELECT MAX(date) from monnaies GROUP BY code);";
    statement = connection.prepareStatement(sql);
    statement.setString(1,code);
    ResultSet results = statement.executeQuery(sql);
    results.next();
    if(results != null) {
      String name = results.getString("nom_complet");
      String pays = results.getString("pays"); // Simplification : un seul pays
      List<String> listePays = new ArrayList<>();
      listePays.add(pays);
      float taux = results.getFloat("taux");
      m = new Monnaie(listePays, name, code,taux);
    }
  } catch (SQLException e) {
    e.printStackTrace();
  }
  finally {
    try {
      statement.close();
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
  return m;
}
```

Il faudrait importer les classes `PreparedStatement`, `ResultSet`, ... du package `java.sql`.

- A côté de cette classe, on va définir une classe `DAOFactory` avec une méthode statique qui retourne une instance utilisée par `DAOMonnaie` et qui donne accès à la base (après avoir réalisé la connexion au serveur et l'authentification nécessaire).
  La classe `DAOFactory` est donnée sur Moodle. Elle permet l'accès à un serveur MySQL en utilisant le driver JDBC adéquat et les credentials indiqués dans le fichier `dao.properties` fourni sur Moodle aussi.
  Il faudrait ajouter au script de build du projet la dépendance au driver `mysql-connector-java` :
  Pour Maven :

```
<dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <version>8.0.22</version>
</dependency>
```

  Pour Gradle : `implementation group: 'mysql', name: 'mysql-connector-java', version: '8.0.22'`
- Pour vous connecter en ligne de commande au serveur de BdD MySQL (à installer sur vos machines au préalable), ouvrir un terminal et taper la commande :
  `mysql -u<user> -p`

Le mot de passe pour cet utilisateur est indiqué dans `dao.properties`
A partir de là, vous pouvez saisir des requêtes SQL pour tester votre code.
Vous pouvez créer une base de données `jee` et une table `monnaies` (utilisées dans les exemples fournis) à partir de ce shell MySQL ou bien à partir des classes `DAOMonnaie` ou `DAOFactory` dans leurs constructeurs en veillant à tester (avec `IF NOT EXISTS`) si elles n'existent pas déjà.

- Déployer l'application et la tester