Travaux pratiques avec JDBC

1 Base de données

1.1 Tester MySql

Un serveur MySql est accessible à partir des postes de travail (machine `dbs-perso.luminy.univmed.fr`) et vous pouvez demander la création d'une base de données personnelle [1]. Vous pouvez utiliser le client MySql (commande `mysql`), à partir de votre machine pour vous connecter sur le serveur de bases de données.

```
$ mysql -u utilisateur-bd -h dbs-perso.luminy.univmed.fr -p
password: mot-de-passe reçu par mail
mysql> use nom-de-la-base;
mysql> création d'une table
```

Vous pouvez maintenant créer une ou plusieurs tables simples, en ajoutant des données et en programmant quelques requêtes.

1.2 Utiliser HyperSQL

Si vous avez des difficultés avec la base personnelle MySQL, vous pouvez utiliser la base de données embarquée HyperSQL [2]. N'oubliez pas de lire la documentation HyperSQL [3].

1.3 Pilotes JDBC.

- Créez dans Eclipse JEE un projet Java standard.
- Ajoutez Maven à votre projet : **Sélectionnez votre projet** / **Bouton-droit** / **Configure** / **Convert to Maven Project**. Vous devez à cette étape donner une numéro de version à votre projet. Laissez les valeurs par défaut.
- Cherchez sur MVNRepository [4] le package `mysql-connector-java` [5] et incluez dans le fichier `pom.xml` de votre projet les dépendances.
- Profitez de cette étape pour ajouter les dépendances de la base de données embarquée HyperSQL [6].

2 Premiers programmes JDBC

Pour organiser notre travail, nous allons commencer pas créer deux classes :

- Une classe `JdbcTools` qui va contenir le code générique indépendant de notre domaine d'application.
- Une classe `Dao` (*Data Access Object*) qui va contenir le code JDBC qui a la charge de sauvegarder et de récupérer les données (javaBeans) de notre application. Cette classe étend la première.

Nous allons **immédiatement**, créer deux tests unitaires. Le premier qui étend la classe `JdbcTools` (`JdbcToolsTest`) et le second qui teste la classe `Dao` (`DaoTest`).

1. ref:dosicalu
2. http://hsqldb.org/
3. ref:doc-hsqldb
4. http://mvnrepository.com
6. http://hsqldb.org/

2.1 Préparer la couche des outils

L'objectif de cette classe est d'offrir une série d'outils afin de faciliter l'écriture des classes DAO.

- Commencez par créer dans la classe `JdbcTools` les propriétés, les setters et les getters pour représenter le nom du pilote, l'URL de connexion, l'identifiant et le mot de passe.
- Créez ensuite dans la classe `JdbcTools` les deux méthodes ci-dessous et prévoyez un test unitaire de ces méthodes (dans `JdbcToolsTest`).

```
public void init() { ... }
public void close() { ... }
```

- Créez ensuite dans la classe `JdbcTools` les deux méthodes de création/destruction des connexions. Prévoyez un test unitaire de ces méthodes (dans `JdbcToolsTest`).

```
public Connection newConnection() throws SQLException { ... }
public void quietClose(Connection c) { ... }
```

- Vous pouvez maintenant ajouter la méthode `executeUpdate` qui va vous offrir un moyen simple de détruire ou créer des tables et des lignes.

```
public int executeUpdate(String query) throws SQLException { ... }
```

  Cette méthode doit avoir la structure ci-dessous. Il est **très important** de bien capter les exceptions afin de libérer la connexion.

```
try {
   // créer une connexion
   // préparer l'instruction
   // exécuter l'instruction
} catch (SQLException e) {
   // renvoyer cette exception
}
} finally {
   // fermer la connexion
}
```

- Nous aurons souvent besoin de paramétrer l'exécution d'une mise à jour. Pour ce faire, ajoutez à la méthode précédente une série d'arguments. Cette nouvelle version est basée sur l'utilisation des `PreparedStatement`.

```
public int executeUpdate(String sql, Object... parameters)
   throws SQLException { ... }
```

  Attention : vous devrez, pour chaque paramètre, étudier sont type pour utliser la méthode `setType` la plus appropriée (contentez-vous d'étudier les cas numériques et chaînes de caractères).

2.2 Try-with-ressources de java 7

A partir de java 7 nous pouvons simplifier la gestion des connexions en utilisant le Try-with-ressources [7]. Utilisez ce principe pour simplifier la structure de la méthode `executeUpdate` précédente en supprimant la clause `finally`.

7. http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

```
try (Connection conn = newConnection()) {
    // Utilisation de la connexion
    ...

}
```

2.3 Lister des lignes

• Partons du principe que vous avez une table qui représente une personne (vous pouvez, bien entendu prendre n'importe quel autre exemple). Prenez soin de placer une clef primaire dans cette table (et dans le javaBean). Créez dans la classe `Dao` la méthode

```
public Collection<Person> findPersons() throws DaoException { ... }
```

Cette méthode doit avoir la structure ci-dessous. il est **très important** de bien capter les exceptions afin de libérer la connexion (sauf si vous utilisez le **try-with-ressources**).

```
try {
   // 1. créer une connexion
   // 2. préparer l'instruction
   // 3. exécuter la requête
   // 4. lire le résultat
} catch (SQLException e) {
   // 5. construire l'exception DAO
   // 6. renvoyer cette exception
}
} finally {
   // 7. fermer la connexion
}
// renvoyer le résultat
```

Écrivez le test unitaire de cette méthode (vous pouvez utiliser la méthode `executeUpdate` pour créer plusieurs personnes).

• Vous pouvez remarquer que dans cette méthode, seule la ligne 4 est spécifique à notre application. Nous pouvons généraliser cette méthode en créant (dans `JdbcTools`) la version suivante :

```
public <T> Collection<T> findBeans(String sql, ResultSetToBean<T> mapper)
   throws SQLException { ... }
```

dans laquelle l'interface `ResultSetToBean` est définie par

```
// construire un bean de type T à partir d'un ResultSet
interface ResultSetToBean<T> {
   T toBean(java.sql.ResultSet rs) throws SQLException;
}
```

La méthode `findBeans` ainsi définie permet d'interroger n'importe quelle table pour construire le JavaBean équivalent. Utilisez la méthode `findBeans` pour simplifier `findPersons`. Prenez soin d'utiliser les lambda expressions [8] de Java 8 pour instancier l'interface `ResultSetToBean`.

• Dans un deuxième temps vous pouvez ajouter à la méthode `findBeans` une série de paramètres.

8. https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

2.4 Insérer une ligne

Vous pouvez maintenant enrichir votre classe `Dao` pour offrir une méthode d'ajout d'une personne. Prévoyez un test unitaire de cette méthode ainsi qu'un enrichissement du test de la méthode précédente (vous pouvez maintenant ajouter des lignes).

```
public void addPerson(Person p) throws DaoException { ... }
```

Testez notamment les cas d'erreur dus à la contrainte de clef primaire.

Conseil : Utilisez la version `UPDATABLE` de la classe `java.sql.ResultSet`. C'est plus simple.

2.5 Détruire ou modifier des lignes

- Écrivez dans la classe `Dao` la méthode d'effacement d'une personne :

```
public void removePerson(int id) throws DaoException { ... }
```

- Toujours dans la classe `Dao`, écrivez la méthode de mise à jour d'une personne. Utilisez la version `UPDATABLE` de la classe `java.sql.ResultSet`.

```
public void updatePerson(Person p) throws DaoException { ... }
```

Ralentissez votre méthode (avec un `sleep` entre la requête et un éventuel `commit`), et testez les mises à jour concurrentes avec l'interface ligne de commande de mysql. **Attention** : dans sa version de base, MySql ne gère ni les transactions, ni la pose de verrous. Pour avoir accès à ces fonctionnalités, vous devez utiliser un codage particulier (appelé `InnoDB`) pour vos tables. Pour ce faire, utilisez la clause ci-dessous :

```
ALTER TABLE votre_table TYPE=InnoDB ;
```

- Dans un deuxième temps, vous pouvez concevoir dans la classe `JdbcTools` une version générale de la méthode `updatePerson` :

```
public <T> void updateBean(String sql,
   BeanToResultSet<T> mapper,
   T theBean,
   Object... parameters)
   throws SQLException { ... }
```

3 Utiliser les DataSource

Nous allons maintenant passer par une `DataSource` pour obtenir nos connexions :

- Ajoutez les dépendances pour Apache Commons DBCP [9]. Cela a du ajouter les librairies *Commons Pool* et *Common Logging*.
- Dans la méthode `init` de `JdbcTools` préparez une instance

9. http://commons.apache.org/dbcp/

```
BasicDataSource bds = new BasicDataSource();
bds.setUrl(...);
bds.setUsername(...);
bds.setPassword(...);
bds.setInitialSize(5); // ouvrir cinq connexions
bds.setMaxTotal(5);    // pas plus de cinq connexions
```

- Utilisez la `DataSource` pour créer vos connexions (méthode `getConnection`). Profitez-en pour afficher le nombre de connexions actives et inactives (méthodes `getNumActive` et `getNumIdle`).
- Préparez un test qui va créer une dizaine de threads pour tester l'allocation des connexions :

```
public void testWorks() throws Exception {
    long debut = System.currentTimeMillis();

    // définition d'un travail long (fictif)
    Runnable longWork = () -> {
        try (Connection c = newConnection()) {
            // travail fictif
            Thread.sleep(5000);
        } catch (Exception e) {
        }
    };

    // exécution des threads
    ExecutorService executor = Executors.newFixedThreadPool(10);
    for (int i = 1; (i < 10); i++) {
        executor.execute(longWork);
    }

    // attente de la fin des threads
    executor.shutdown();
    executor.awaitTermination(10, TimeUnit.HOURS);

    // calcul du temps de réponse
    long fin = System.currentTimeMillis();
    System.out.println("duree = " + (fin - debut) + "ms");
}
```

- Faites varier les deux derniers paramètres de la `DataSource` et étudiez l'impact sur le temps de réponse.

4 Utiliser les métas informations

Écrire (dans la classe `JdbcTools`) une méthode générale qui prend en paramètre une requête et qui imprime une page HTML contenant la requête et le résultat de la requête.

```
public void makeTable(String sql, PrintWriter stdout)
   throws SQLException { ... }
```

5 Utiliser les RowSet

Vous pouvez utiliser les `RowSet` pour traiter cette question. Pour ce faire, vous avez besoin d'une implantation des interfaces RowSet. Depuis la version 1.5, ces implantations sont disponibles dans la JVM. Pour les trouver, vous pouvez essayer :

```
import javax.sql.rowset.RowSetProvider;
import javax.sql.rowset.RowSetFactory;
import javax.sql.rowset.RowSet;

...

RowSetFactory factory = RowSetProvider.newFactory();
RowSet rs = factory.createCachedRowSet();
```

Il est **fortement conseillé** de s'inspirer de ce guide [10].

10. http://download.oracle.com/javase/tutorial/jdbc/basics/jdbcrowset.html