TD3 Maven JDBC

JDBC (Java Database Connectivity) est l'API Java pour interroger des bases de données relationnelles. Elle permet d'établir une connexion, de transmettre des requêtes et d'exploiter les résultats. (cf. synthèse de Jean-Luc Massat,Aix-Marseille)

Cet exemple introductif va exploiter une base de données MySQL en local (db_concours). Cette base stocke une table Candidats dont les enregistrements disposent des informations suivantes : id (clé entier) numero (entier) nom (chaîne) score (decimal) region (entier)

1) Créer et peupler la base de données

Lancer WampServer/XAMPP
phpMyAdmin (user='root' pwd=) pour gérer les bases MySQL
Exécuter un script pour créer une nouvelle base
Importer/Choisir fichier/db_concours.sql - exécuter le script

2) Créer et un projet Maven JAVA

Group ID : **pta.sagi**
Artifact ID : **maven-jdbc-concours**
Version : **0.1**

3) Ajouter une dépendance (fichier POM.xml) pour le connecteur MySQL

```
<dependency>
	<groupId>mysql</groupId>
	<artifactId>mysql-connector-java</artifactId>
	<version>5.1.34</version>
</dependency
```

4) **Ajouter une classe ClsMain (prog console).** Le code ci-dessous permet de charger le driver. Vérifier la bonne exécution .

```
public class ClsMain {

   public static void main(String[] args) {

         //chargement du Driver
         try{
               Class.forName("com.mysql.jdbc.Driver");
         }
         catch(ClassNotFoundException ex){
               System.out.println(ex.getMessage());
               System.exit(1); // driver non trouvé
         }
         System.out.println("Driver OK");
     }
}
```

5) Connexion et interrogation de la base

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class MainConsole {

	final static String url="jdbc:mysql://localhost:3306/db_concours";
	final static String user="root";
	final static String pwd="";

	public static void main(String[] args) {

		//chargement du Driver
		try{   Class.forName("com.mysql.jdbc.Driver");    }
		catch(ClassNotFoundException ex){
			System.out.println(ex.getMessage());
			System.exit(1); // driver non trouvé
		}

		// requête SQL
		String select = "SELECT NOM,SCORE,REGION FROM candidats WHERE REGION=?";

		// try-with-resources (cf Mémo p20)
		try(Connection con=DriverManager.getConnection(url,user,pwd);
			PreparedStatement ps = con.prepareStatement(select);)
		{
			ps.setInt(1,2);  //définit le paramètre n°1 (le seul) de la requête

			// try-with-resources
			try(ResultSet rs=ps.executeQuery();){
				while(rs.next()){
					System.out.printf("nom=%s score=%f region=%d \n",
						rs.getString(1),rs.getDouble(2),rs.getInt(3));
				}
			}
			catch(SQLException e1){
				System.out.println(e1.getMessage());
			}
		}
		catch(SQLException e2){
			System.out.println(e2.getMessage());
		}
	}
}
```

Classe `PreparedStatement` : pour construire une requête avec des parties variables marquées par des ' ?'. Dans l'exemple ci-dessus, le numéro de région est un paramètre.

Pour exécuter la requête, selon qu'il s'agit d'une lecture ou d'une modification de la base, il y a deux méthodes différentes :

`PreparedStatement.executeQuery()` : pour la consultation (requêtes select)
`PreparedStatement.executeUpdate()` : pour la modification (requêtes insert)

Classe `ResultSet` : contient les résultats. Le nombre de colonnes dépend de la requête.

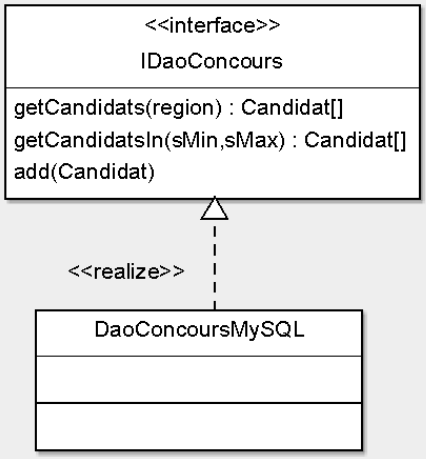
On parcourt les lignes de résultats (tant qu'il y en a). Pour chaque ligne, on obtient les données attendues par des appels rs.getXxx(indice). L'indice est le numéro (à partir de 1) de la donnée dans la requête. Dans l'exemple ci-dessus, le score est en position 2 d'une ligne de résultat.

6) Tester quelques requêtes (possible aussi sous phpMyAdmin)

a) Afficher tous les champs des candidats de la region 3
b) Afficher nom et score des candidats dont le score est entre 75 et 79
c) Afficher le score maximal (toutes régions)
d) Afficher le score max de la region 4
e) Insérer un candidat
f) Changer le score d'un candidat dont on connaît le numéro (ex : le dernier inséré)

7) Gérer les accès aux données via une interface/une implémentation

On peut donner accès aux candidats à travers un objet DAO. Au TD2, il s'agissait d'un accès à un fichier JSON. On propose ici une autre implémentation de l'interface IDaoConcours. Cette fois-ci, les données sont stockées dans la base MySQL.



Projet Maven :

```
Group ID = pta.sagi
Artifact ID= dao-concours-mysql
```

Packages :

pta.sagi.concours.dao = interface et classe IDao/Dao
pta.sagi.concours.entites = classe Candidat

```
public class DaoConcoursMySQL implements IDaoConcours
{
	final static String url="jdbc:mysql://localhost:3306/db_concours";
	final static String user="root";
	final static String pwd="";

	public DaoConcoursMySQL(){

		try{
			Class.forName("com.mysql.jdbc.Driver");
		}
		catch(ClassNotFoundException ex){
			System.out.println(ex.getMessage());
			System.exit(1); // driver non trouvé
		}
	}

	public Candidat[] getCandidats(int region)
	{
		... // requêtes pour obtenir les données
	}
}
```