
R2.06 EXPLOITATION D'UNE BASE DE DONNÉES
FEUILLE DE TP N°8 *L'utilisation de JDBC*

L'objectif de cette feuille de TP est de réaliser un classe qui gère les connexions à votre base de données puis une classe qui gère les mises à jour dans une table **JOUEUR**. La base de données contient une deuxième table **MESSAGE** qui sera utilisée pour le dernier exercice. Il s'agit des messages échangés entre les joueurs.

Dans l'archive `tp8.zip`, vous trouverez le script de création de la base de données `bd_joueur.sql` que vous devez exécuter dans un client `mysql`. Voici la structure de cette base de données.

```
CREATE TABLE JOUEUR ( numJoueur decimal(6,0), pseudo varchar(10),  
    motdepasse varchar(500), main char(1), abonne char(1),  
    niveau decimal(1,0),  
    PRIMARY KEY (numJoueur), UNIQUE KEY pseudo (pseudo)  
);  
CREATE TABLE MESSAGE ( idMsg decimal(6,0), dateMsg datetime,  
    contenuMsg text, luMsg char(1), idUt1 decimal(6,0),  
    idUt2 decimal(6,0), PRIMARY KEY (idMsg),  
    FOREIGN KEY (idUt1) REFERENCES JOUEUR(numJoueur),  
    FOREIGN KEY (idUt2) REFERENCES JOUEUR(numJoueur)  
);
```

La colonne `main` de la table `joueur` indique si le joueur est droitier 'D' ou gaucher 'G'.

Attention ! Les jointures entre ces tables ne sont pas *naturelles* mais se font par des tests entre `numJoueur` et `idUt1` et `idUt2`. Vous allez exécuter le script pour créer dans votre base de données.

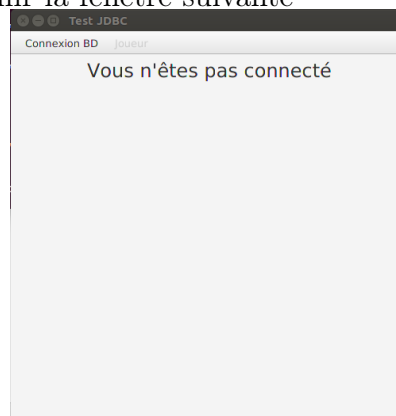
Exercice 1 *L'application*

1.1 Lancement de l'application

Normalement, si vous ouvrez le répertoire `tp8` dans `vscode`, tout est configuré pour lancer l'application dont l'interface graphique est écrite en JavaFX. La configuration est définie dans les fichiers `.vscode/settings.json` et `.vscode/launch.json`. Cependant, si vous avez des difficultés vous pouvez ouvrir un terminal et vous positionner dans le répertoire `tp8` puis taper les commandes suivantes :

```
javac --source-path ./src -d ./bin --module-path /usr/share/openjfx/lib/ --add-modules javafx.controls src/TestJDBC.java  
java -cp ./bin:/usr/share/java/mariadb-java-client.jar --module-path /usr/share/openjfx/lib/ --add-modules javafx.controls TestJDBC
```

Si tout va bien vous devriez obtenir la fenêtre suivante



1.2 Structure de l'application

L'application sur laquelle vous allez travailler est une application graphique dont la classe principale est `TestJDBC`. Vous n'aurez à intervenir ni sur les vues ni sur les contrôleurs. Seuls les fichiers concernant le modèle (et la base de données) seront à modifier. Il s'agit des fichiers suivants :

- `ConnexionMySQL.java` qui contient le squelette de la classe permettant de charger le driver MySQL et de se connecter à la base de données.
- `JoueurBD.java` qui contient le squelette de la classe permettant d'interagir avec la table `JOUEUR` de votre base de données.

Il vous est conseillé de regarder la classe `Joueur.java` qui contient la représentation java d'un joueur.



Attention

Pour faire ce TP, référez vous au cours JDBC disponible sur CELENE

Exercice 2 La classe `ConnexionMySQL`

Complétez la classe `ConnexionMySQL` qui contient les attributs `mysql` de type `Connection` et `connecte` qui est un booléen indiquant si la connexion est en cours ou non.

1. Le constructeur de cette classe va simplement charger le driver JDBC pour MySQL.
2. La méthode `connecter` prend en paramètres un nom de serveur, un nom de base de données, un login et le mot de passe de l'utilisateur et va créer la connexion.
3. La méthode `close` va simplement fermer la connexion.

Les exceptions éventuellement levées par ces méthodes seront traitées au niveau de la partie graphique de l'application.

Notez que cette classe contient des reprises des méthodes équivalentes de l'interface `Connection` afin de *simuler* un héritage.

Si votre classe est bien implémentée vous devriez pouvoir vous connecter à la base de données en utilisant l'option *Connexion* du menu *Connexion BD*.

Exercice 3 La classe `JoueurBD`

Cette classe va servir à gérer les mises à jour dans la table `JOUEUR` à partir d'un programme JAVA. Cette classe utilise la classe `Joueur` qui permet de représenter les joueurs en mémoire.

La classe contient un attribut de type `ConnexionMySQL` qui stocke la connexion avec laquelle on travaille. Par ailleurs cette classe devra implémenter les méthodes ci-dessous. **Attention!** Testez bien vos méthodes dans la classe `TestJDBC` au fur et à mesure que vous les codez ! La méthode `rechercherJoueurParNum` doit être implémentée avant les méthodes de mise à jour de la table `JOUEUR`.

- un constructeur prenant en paramètre un objet de la classe `ConnexionMySQL`.
- `int maxNumJoueur()` qui interroge la table `JOUEUR` pour connaître le plus grand numéro de joueur déjà attribué. Si la table est vide la méthode doit retourner 0.
- `Joueur rechercherJoueurParNum(int num)` qui retourne le joueur identifié par `num` ou jette une exception SQL si aucun joueur ne correspond :
`throw new SQLException("Joueur non trouvé");`

- `int insererJoueur(Joueur j)` qui insère dans la table `JOUEUR` le joueur passé en paramètres. Cette méthode ne prend pas en compte le numéro du joueur mais elle en attribue automatiquement au joueur à l'aide de la méthode `maxNumJoueur()`.
- `void effacerJoueur(int num)` qui supprime de la table `JOUEUR` le joueur identifié par `num`.
- `void majJoueur(Joueur j)` qui effectue la mise à jour du joueur passé en paramètre. Attention seul le numéro du joueur ne pourra pas être modifié.
- `ArrayList<Joueur> listeDesJoueurs()` qui retourne la liste de tous les joueurs de la base de données.
- `String rapportMessage()` qui va indiquer pour chaque joueur et chaque jour le nombre de message qu'il a envoyés comme ci-dessous.

Résultat

```

donkey 2024-04-09 2
mario 2024-04-11 2
mario 2024-04-12 3
peach 2024-04-10 2
peach 2024-04-11 2
wario 2024-04-09 1
wario 2024-04-11 1
wario 2024-04-12 1
yoshi 2024-04-09 1
yoshi 2024-04-10 1
yoshi 2024-04-11 1

```

- `String rapportMessageCompleet()` qui va indiquer pour chaque joueur la liste des messages qu'il a reçus classés par ordre chronologique comme ci-dessous.

Résultat

```

Messages reçus par donkey
2024-04-11 10:10:00.0: de peach: bla bla13
2024-04-11 10:14:00.0: de wario: bla bla17
total messages: 2

Messages reçus par mario
2024-04-09 10:25:05.0: de yoshi: bla bla8
2024-04-09 11:12:00.0: de wario: bla bla5
2024-04-11 12:12:10.0: de peach: bla bla12
2024-04-12 10:12:18.0: de wario: bla bla4
total messages: 4

Messages reçus par wario
2024-04-09 10:24:00.0: de donkey: bla bla6
2024-04-09 10:24:00.0: de donkey: bla bla7
2024-04-10 10:13:06.0: de yoshi: bla bla9
2024-04-11 10:15:00.0: de yoshi: bla bla16
2024-04-12 10:12:00.0: de mario: bla bla1
2024-04-12 10:12:07.0: de mario: bla bla3
2024-04-12 10:12:12.0: de mario: bla bla2
total messages: 7

Messages reçus par yoshi
2024-04-10 10:12:07.0: de peach: bla bla10
2024-04-10 10:12:08.0: de peach: bla bla11
2024-04-11 10:12:00.0: de mario: bla bla14
2024-04-11 10:13:00.0: de mario: bla bla15
total messages: 4

```



Important

Pour les deux points suivant n'hésitez pas à regarder les classes `GraphiqueMessages` et `GraphiqueMain`. Cela peut vous aider pour la SAé finale.

- `ArrayList<Map.Entry<String, Integer>> nbMsgParJour()` qui va permettre de récupérer les données servant à construire le graphique suivant. Le type de retour de la fonction est un `ArrayList` contenant des `Map.Entry` qui sont des couple clé,valeur. La ici sera la date du jour sous forme du chaine de caractères et la valeur sera le nombre de messges envoyés ce jour là.



- `ArrayList<Map.Entry<String, Integer>> nbMain()` qui va permettre de retrouver les données pour construire le graphique suivant donnant la répartition gaucher/droitier parmi les joueurs.

