Compte rendu Java - Base de Données

Tom Dunand - 3A - TD2 - TP4

Objectifs

L'objectif de ce TD est de se connecter à une base de donnée avec Java, faire en sorte de pouvoir l'administrer et d'effectue des requêtes SQL enfin de sécuriser un minimum l'accès à la base de donnée.

Connexion

La connexion à la base de données va se faire via un driver JDBC qui prend en charge MySQL. Pour l'utiliser, on a crée une méthode `connect()` dans la classe `MYSQLDatabase` :

```
public void connect() {
        try {
            this.connection = DriverManager.getConnection(
                "jdbc:mysql://" + host + ":" + port + "/" + databaseName + "?
allowMultiQueries=true",
                user,
                password
```

```
            );
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
```

On se connecte à la BDD grâce à la méthode `getConnection()` du module `DriverManager` avec la connection_string "jdbc:mysql://hostname:portNumber/databaseName" fournie dans le sujet, dans laquelle on remplace les différentes valeurs par celles passées en paramètre du constructeur. La chaîne "?allowMultiQueries=true" nous authorizera à effectuer des requêtes multiples sur la DB.

Paramètres du contructeur : `public MYSQLDatabase(String host, int port, String databaseName, String user, String password).`

Requêtes

Statement

Pour pouvoir effectuer une requête SQL, on utilise le module `Java.sql` qui nus permet d'abour de créer un 'Statement' qui est un objet qui nous fourni la méthode `executeQuery()` qui permet quant à elle d'exécuter une requête.

```
public Statement createStatement() {
        try {
            return connection.createStatement();
        } catch (Exception sqlException) {
            System.err.println(sqlException.getMessage());
            return null;
        }
    }
```

Ici on crée uniquement un `Statement` sur la connexion ouvert précédemment.

Requêtes

La méthode simple pour faire des requêtes sur la DB à partir d'un `Statement` est la suivante :

```
statement = connection.createStatement()
ResultSet results = statement.executeQuery("SELECT * FROM sport;")
```

Nous avons par la suite implémenté cela dans des classes.

Structurons tout cela

PolySportsDatabase

```
public class PolySportsDatabase extends MYSQLDatabase{
    private static PolySportsDatabase instance = null;

    private PolySportsDatabase() {
        super("localhost", 3306, "Poly-sports", "root", "");
    }

    public static PolySportsDatabase getInstance() {
        if (instance == null) {
            instance = new PolySportsDatabase();
        }
        return instance;
    }
}
```

C'est une classe qui hérite de MYSQLDatabase et permet d'initier la connection à la DB. A savoir que c'est un singleton, donc on s'assure qu'une seule instance de la connexion peut être créee à chaque fois.

On initie la connexion avec la ligne suivante dans le main.

```
PolySportsDatabase myDatabase = PolySportsDatabase.getInstance();
```

Sport

```
public class Sport {
    private int id;
    private String name;
    private int requiredParticipants;
    public Sport(int id, String name, int requiredParticipants) {
        this.id = id;
        this.name = name;
```

```
        this.requiredParticipants = requiredParticipants;
    }
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
    public int getRequiredParticipants() {
        return requiredParticipants;
    }
}
```

Cette classe définit les attributs de chaque sport et nous permet de créer un nouveau Sport (que l'on pourra par la suite ajouter à la DB) avec tous les accesseurs nécéssaires aux attributs de chaque sport. C'est le code métier d'un objet Sport.

Exemple de création de sport et d'accès a ses attributs :

```
Sport foot = new Sport(4,"bad",145);
System.out.println(foot.getId() + " " + foot.getName() + " " +
foot.getRequiredParticipants());
```

Sortie :

```
4 bad 145
```

SportsDAO

```
public class SportsDAO {
    MYSQLDatabase database;
    public SportsDAO(MYSQLDatabase database) {
        this.database = database;
        this.database = PolySportsDatabase.getInstance();
        this.database.loadDriver();
        this.database.connect();
    }
    public ArrayList<Sport> findAll(){
        Statement statement = this.database.createStatement();
        try {
            ResultSet resultSet = statement.executeQuery("SELECT * FROM sport;");
            ArrayList<Sport> sports = new ArrayList<>();
            while (resultSet.next()) {
                sports.add(new Sport(resultSet.getInt("id"),
resultSet.getString("name"), resultSet.getInt("required_participants")));
```

```
            }
            return sports;
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return null;
    }
```

C'est la classe qui fait le lien entre un sport et la base de donnée. C'est à dire qu'elle définit la méthode `findAll` pour récupérer le contenu de la base de donnée et le formater.

Nous avons ensuite ajouté les méthodes `findById` et `findByName` qui nous permettent de récupérer un seul sport selon l'ID ou le nom de sport fourni :

```
    public Sport findById(int id){
        database.connect();
        try {
            PreparedStatement myStatement = database.prepareStatement("SELECT *
FROM `sport` WHERE `id` = ?");
            myStatement.setInt(1, id);

            ResultSet results = myStatement.executeQuery(myStatement.toString());
            while (results.next()) {
                return new
Sport(results.getInt("id"),results.getString("name"),results.getInt("required_parti
cipants"));
            }
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return null;
    }

    public Sport[] findByName(){
        List<Sport> sport = new ArrayList<Sport>();
        database.connect();
        System.out.println("Keyword to search :");
        try (Scanner myScanner = new Scanner(System.in)) {
            String name = myScanner.nextLine();
            try {
                String query = "SELECT * FROM `sport` WHERE `name` LIKE ?";
                PreparedStatement myStatement = database.prepareStatement(query);
                myStatement.setString(1, "%"+name+"%");
                ResultSet results = myStatement.executeQuery();
                while (results.next()) {
                    sport.add(new
Sport(results.getInt("id"),results.getString("name"),results.getInt("required_parti
cipants")));
                }
            }
```

```
            catch (Exception e) {
                System.out.println("ici");
                System.out.println(e.getMessage());
            }
        }
        Sport[] array_sport = new Sport[ sport.size() ];
        sport.toArray(array_sport);
        return array_sport;
    }
}
```

La seule chose qui change est la requête que l'on a modifié pour fournir l'ID ou nom passé en paramètre. Aussi, ici on utilise `prepareStatement()` qui est une méthode implémentée à la fin du sujet, que je présente dans la aprtie suivante, en réalité à cette étape, nous avons utilisé `executeQuery()` pour éxecuter une requête SQL sur la DB.

Exemple d'utilisation :

```
SportsDAO sportsDAO = new SportsDAO(db);
System.out.println(sportsDAO.findByName());
```

Injection SQL

Prepared Statement

On a ajouté la méthode suivante à la classe MYSQLDatabase car dans la partie précédente, on a remarqué que si on authorise les requêtes multiples, un utilisateur peut supprimmer toute notre DB en rentrant un certaine requête. C'est pourquoi, on utilise un méthode intégrée par JDBC, on remplace les données des requêtes (exemple : nom du sport) par "?" puis avant d'exécuter la requête SQL, on vérifie que la donnée entrée par l'utilisateur est bien du type que l'on veut, sinon elle convertira les caractères par des caractères qui n'ont pas d'effet sur la DB.

```
public PreparedStatement prepareStatement(String data) {
    try {
        return connection.prepareStatement(data);
    } catch (SQLException e) {

        throw new RuntimeException(e);
    }
}
```

On ne peut maintenant plsu effectuer d'injection SQL sur la base de données.