OBJECTIFS

A travers cet exercice, vous allez :

- Utiliser Java pour vous connecter à une base de données,
- Créer une architecture combinant objets métiers et DAO,

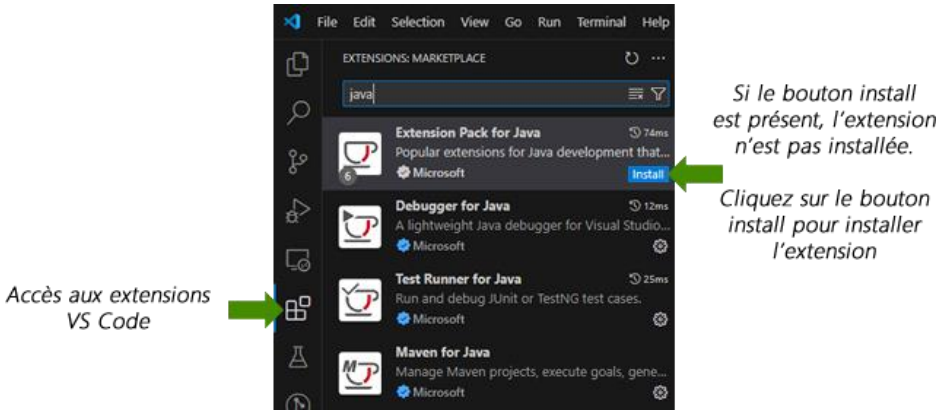
PREPARATION

MYSQL

- Recherchez XAMPP sur votre ordinateur. Cet outil intègre Apache, MySQL et PHPMyAdmin et est généralement installé sur les ordinateurs de l'école.
- S'il n'est pas présent, installez-le (https://www.apachefriends.org/fr/index.html).
- Utilisez le XAMPP Panel Control pour démarrer les services Apache et MySQL.
- Accéder à PHPMyAdmin.
- Créez une nouvelle base de données **`poly_sports`** et importer le script SQL fourni pour y intégrer le jeu de données.

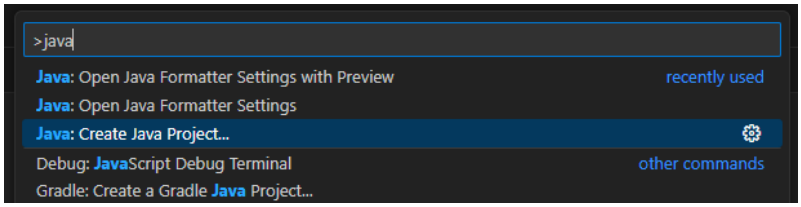
PRISE EN CHARGE DE JAVA AVEC VSCODE

- Démarrez Visual Studio Code.
- Si ce n'est pas déjà fait, installez l'extension **`Extension Pack for Java`**.

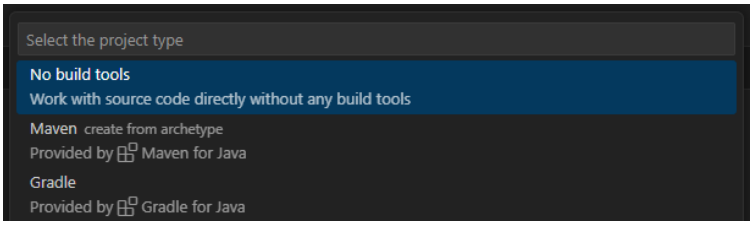


NOUVEAU PROJET

- Appuyez sur la touche **`F1`** de votre clavier.
- Saisissez **`Java`** dans l'invite qui apparaît en haut de la fenêtre VS Code.
- Cliquez sur **`Java: Create Java Project...`**.

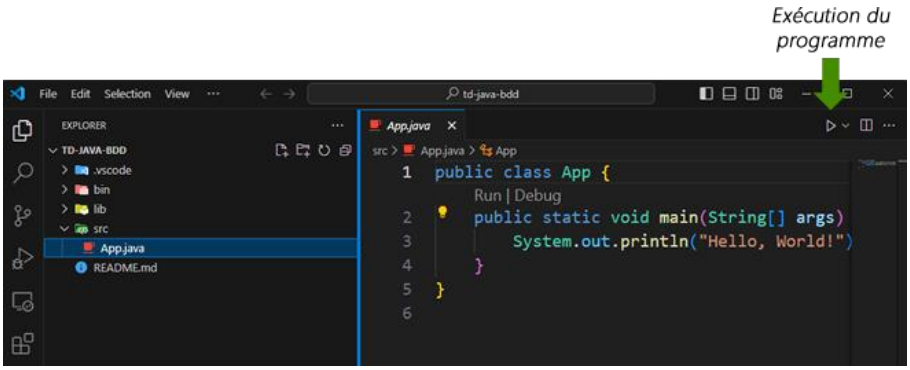


- Sélectionnez **`No build tools`**



- Sélectionnez le dossier dans lequel vous souhaitez créer votre projet à l'aide de l'explorateur de dossiers qui apparaît.
- Saisissez le nom de votre projet (ex : td-java-bdd).

Une nouvelle fenêtre VSCode apparaît avec une arborescence de projet prête à l'emploi :

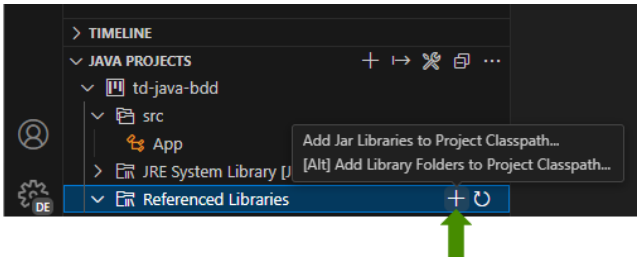


- Sélectionnez le fichier **`App.java`** présent dans le dossier **`src`** et cliquez sur le bouton permettant de démarrer le programme.
- Tout fonctionne ? Passons au dernier préparatif.

DRIVER MYSQL

- Connectez-vous au site suivant : https://dev.mysql.com/downloads/connector/j/.
- Sélectionnez **`Plateform Independent`**.
- Téléchargez l'archive Zip.
- Décompressez l'archive ans le dossier de votre choix.

- Dans VSCode, déroulez la section **`Java Projects`** située en bas à gauche de votre écran et ajoutez une nouvelle référence vers une bibliothèque externe :



- Sélectionnez le fichier mysql-connector-j-8.x.x.jar qui était contenu dans l'archive précédemment décompressée.

PREMIERS PAS AVEC JDBC

CONNECTION

La connexion à la base de données va se faire via un driver JDBC qui prend en charge MySQL. C'est ce que contient le fichier mysql-connector-j-8.x.x.jar que vous avez lié précédemment au projet.

Pour pouvoir utiliser le driver JDBC pour MySQL, il est nécessaire de charger dynamiquement ce dernier :

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

Une fois que le driver a été chargé, nous pouvons initier une connexion avec la base de données :

```
Connection myConnection = DriverManager.getConnection(
    "connection_string",
    user,
    password
);
```

La chaine de connexion utilisée dépend du driver. Dans le cas de MySQL, la chaine de connexion est construite comme ceci :

```
jdbc:mysql://hostname:portNumber/databaseName
```

- A partir des éléments précédents, modifiez la fonction main de votre programme pour :
  - Charger dynamiquement le driver JDBC pour MySQL.
  - Ouvrir une connexion avec la base de données créée plus tôt dans le sujet.
  - Prendre en charge les exceptions pouvant se produire lors des étapes précédentes, en les capturant et en affichant dans la console un texte mentionnant le problème.
- Exécutez votre programme et vérifiez qu'aucune exception ne soit levée.

PREMIERE REQUETE

STATEMENT

Pour effectuer une requête vers la base de données, Java utilise des objets de type **`Statement`** (java.sql). La classe **`Connection`** dispose d'une fonction **`createStatement`** qui retourne un tel objet.

```
Statement myStatement = myConnection.createStatement();
```

REQUETE

Une fois le statement entre vos mains, vous pouvez appeler sa méthode **executeQuery** pour exécuter une requête SQL de type **SELECT** :

```
ResultSet results = myStatement.executeQuery("MY_SQL_QUERY");
```

Les enregistrements trouvés dans la base de données seront stockés dans un objet de type **ResultSet** (java.sql).

Le parcours des enregistrements contenus dans le **ResultSet** se fait ainsi :

```
while(results.next())
{
    // Do something with the current record.
}
```

Chaque appel à **next** positionne le **ResultSet** sur l'enregistrement suivant, sachant que le premier appel à **next** positionne le **ResultSet** sur le premier enregistrement disponible.

Si aucun enregistrement n'est disponible, **next** renvoie **null**.

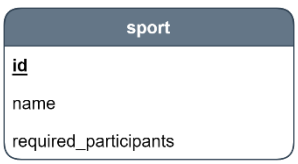
Une fois positionné sur un enregistrement, le **ResultSet** propose des méthodes permettant de récupérer la valeur d'une colonne pour l'enregistrement courant en fonction du type de la donnée :

```
//Récupère la valeur de la colonne name au format String
final String name = results.getString("name");

//Récupère la valeur de la colonne age au format int
final int age = results.getInt("age");
```

- Complétez la fonction main afin de réaliser une requête qui sélectionne tous les enregistrements de la table **sport** et affiche pour chaque sport son nom et le nombre de joueurs requis.

  La modélisation ci-dessous représente la structure de la table **sport** importée précédemment :



- Testez le bon fonctionnement.

STRUCTURONS TOUT CELA

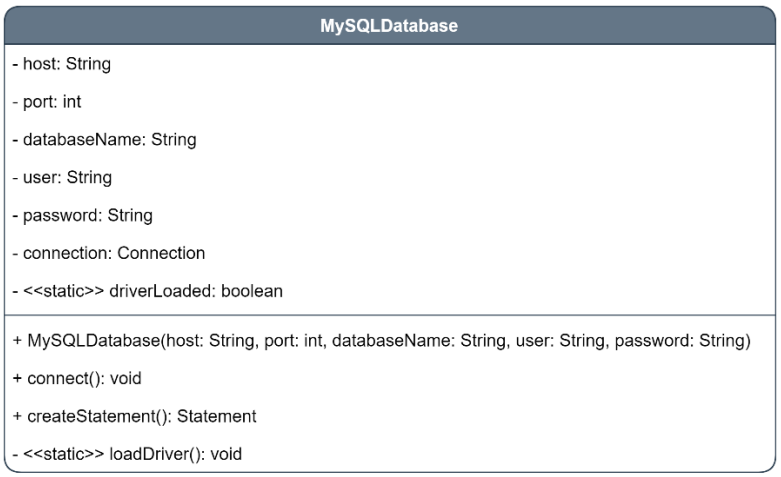
Voilà, vous savez à présent comment vous connecter à une base de données MySQL avec Java. Mais bon, le code que vous venez de faire passe dans le cadre d'un test rapide de faisabilité. Pour une application plus complexe (pensez au projet de Web qui arrive bientôt), autant dire que ce ne sera pas possible de tout mettre dans la fonction **`main`**.

Dans les faits, la fonction **`main`** n'est là que comme point d'entrée de votre programme et ne sert qu'à mettre en route les éléments de base de votre application. Ainsi, une fonction **`main`** ne devrait contenir que quelques lignes de code servant à initialiser les classes qui, elles, gèreront le reste de votre application.

Vous allez donc refactoriser votre code afin d'obtenir quelque chose de plus élégant et surtout de réutilisable (vous ai-je parlé d'un projet en Web ?).

MYSQLDATABASE

- Faites un clic-droit sur le dossier **`src`** de votre projet et sélectionnez **`New Java File > Class`**.
- Saisissez **`MYSQLDatabase`** comme nom de classe.
- Implémentez la classe **`MYSQLDatabase`** à partir de la modélisation suivante :



- Le constructeur de la classe initialisera les attributs de cette dernière. **`connection`** sera initialisée à **`null`** et **`driverLoaded`** à **`false`**.
- La méthode statique **`loadDriver`** chargera dynamiquement le driver JDBC pour MySQL seulement si la variable statique **`driverLoaded`** est à false. Si la méthode **`loadDriver`** parvient à charger le driver, elle passera la variable **`driverLoaded`** à **`true`**. Ainsi, quelque soit le nombre d'instances de **`MySQLDatabase`** créés, le driver ne sera chargé qu'une seule fois.
- La méthode **`connect`** se chargera d'ouvrir une connexion avec la base de données dont les paramètres sont enregistrés dans les attributs de la classe. En cas de succès, la méthode **`connect`** conservera la connexion avec la base de données dans l'attribut **`connection`**.

- La méthode **`createStatement`** retournera un **`Statement`** créé à partir de la connexion en cours (si une connexion a pu être établie, sinon elle lèvera une exception de type **`SQLException`**).

> Bonne pratique : lorsqu'une donnée n'est pas censée évoluer dans le temps, il est recommandé de la stocker dans une constante plutôt qu'une variable. En Java, vous utiliserez le mot clé final pour cela :
>
> ```
> private final type constantAttribute;
> ```
>
> La valeur d'un attribut constant ne pourra être affecté qu'une seule fois : lors de la construction de la classe.

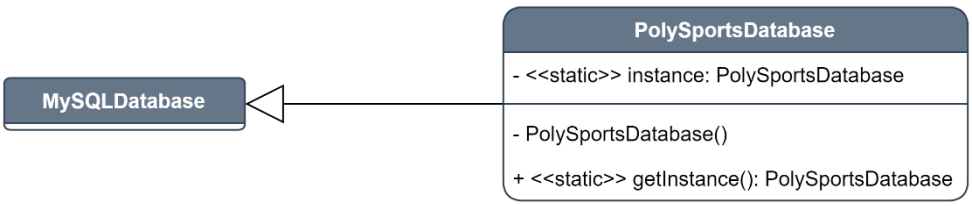
- Modifiez le code de la fonction **`main`** pour que le chargement du driver JDBC, la connexion à la base de données et la création du **`Statement`** soit pris en charge par une instance de la classe **`MySQLDatabase`**.
- Testez le bon fonctionnement.

POLYSPORTSDATABASE

Vous allez à présent créer la classe **`PolySportsDatabase`** qui hérite de **`MySQLDatabase`** et configurera automatiquement la connexion vers la base de données. De plus, **`PolySportsDatabase`** sera un singleton, ce qui nous assurera que quelque soit le nombre de requêtes que nous ferons dans notre application, seule une connexion vers la base de données sera ouverte.

> Attention : utiliser une seule connexion à une base de données peut sérieusement diminuer les performances d'une application dans le cas de nombreux échanges avec le SGBD. Mais comment ne pas profiter de l'occasion présente pour vous faire manipuler un Design Pattern ?

Voici la modélisation de la classe **`PolySportsDatabase`**:



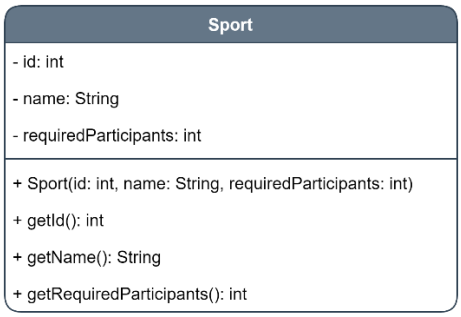
- Implémentez la classe **`PolySportsDatabase`**.
  - L'attribut statique **`instance`** sera initialisé à **`null`**.
  - La méthode statique **`getInstance`** créera une nouvelle instance de **`PolySportsDatabase`** qui sera stockée dans **`instance`** si cette dernière est **`null`**. Puis **`getInstance`** retournera la valeur de **`instance`**.

- Le constructeur de **`PolySportsDatabase`** (qui est privé) appellera le constructeur de la classe parent en lui transmettant les éléments de connexion propre à votre base de données.

➢ Remplacez l'utilisation de **`MySQLDatabase`** dans la fonction main par **`PolySportsDatabase`**.
➢ Testez le bon fonctionnement.

SPORT

➢ Implémentez la classe Sport ci-dessous :



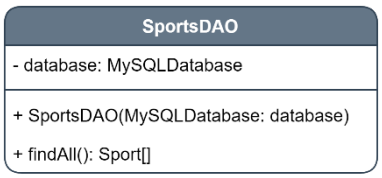
➢ Dans la fonction main, créez une instance de la classe Sport et affichez la valeur de ces attributs afin de tester le bon fonctionnement.

SPORTSDAO

Jusque-là vous avez d'un côté la classe **`PolySportsDatabase`** qui gère la connexion avec la base de données et la classe **`Sport`** qui représente un sport. Vous allez à présent créer une classe qui fera le lien entre votre objet métier (**`Sport`**) et la base de données : un **`Data Access Object`**, plus communément appelé **`DAO`** (à ne pas confondre avec Dessin Assisté par Ordinateur).

Commençons simplement :

➢ Implémentez la classe **`SportsDAO`** suivante :



- La base de données à utiliser sera passer en paramètre au constructeur de la classe.
- La méthode **`findAll`** renverra un **`ArrayList`** contenant tous les sports trouvés dans la base de données.

➢ Modifiez le code de la fonction main pour remplacer le code restant utilisant un **`Statement`** par une instance de la classe **`SportsDAO`**
➢ Testez le bon fonctionnement.

FINDBYID

- Ajoutez à la classe **SportsDAO** une méthode **findById** qui prendra en paramètre l'id du sport recherché et retournera le sport trouvé s'il existe ou **null** dans le cas contraire.
- Modifiez la fonction main afin de tester la fonction **findById** de **SportsDAO** avec un id qui existe et un id qui n'existe pas.
- Testez le bon fonctionnement.

FINDBYNAME

- Dans la même idée, ajoutez à la classe **SportsDAO** une méthode **findByName** qui prendra en paramètre une chaine de caractère et qui retournera un ArrayList des sports dont le nom contient la chaine donnée (LIKE).
- Modifiez la fonction main afin de tester la fonction **findByName** de **SportsDAO** avec le mot "bad".
- Testez le bon fonctionnement.

- Modifiez à nouveau la fonction main afin que l'utilisateur puisse saisir le nom du sport à rechercher.

  Le code suivant permet de saisir une chaine de caractères au clavier :

```
Scanner myScanner = new Scanner(System.in);
String input = myScanner.nextLine();
```

- Testez le bon fonctionnement en saisissant "bad".

INJECTION SQL

Avec la dernière question, vous venez de quitter le monde rassurant des Bisounours pour sauter à pieds joints dans le monde réel. Ce dernier est empli de bonnes choses comme les cerises, les films des frères Cohen ou encore l'hymne national Suisse (la version de David Castello Lopes bien évidemment). Mais on trouve également de ce monde des utilisateurs qui ont souvent l'idée saugrenue de ne pas faire ce qu'on leurs demande et parfois bien pis encore.

ALLOWMULTIQUERIES

Pour la suite de l'exercice nous allons devoir activer une fonctionnalité du driver MySQL permettant d'exécuter plusieurs requêtes en une commande.

- Dans la classe **MySQLDatabase**, modifiez la chaine de connexion pour qu'elle est la forme suivante (seule la fin, en vert, diffère de votre version actuelle) :

```
jdbc:mysql://hostname:portNumber/databaseName?allowMultiQuerie=true
```

- Exécutez de nouveau votre programme et saisissez la chaine de caractères suivantes (attention aux espace ␣) :

```
'; DELETE FROM sport WHERE 1; -- 
```

- Retournez sur PHPMyAdmin et vérifiez le contenu de la table sport.

Oups ! Vous venez de faire une injection SQL. C'est à dire que vous avez injecté du code SQL dans une requête afin d'en changer le comportement.
Ceci a été possible à cause de la manière dont vous faites votre requête et plus précisément de la manière avec laquelle vous incorporez des données extérieures : la concaténation. Cette dernière n'offre aucune protection contre les injections SQL.

PREPAREDSTATEMENT

La solution est de laisser JDBC intégrer les données à votre requête. Pour cela, vous allez utiliser des **PreparedStatement** qui prennent en paramètre la requête à exécuter dans laquelle toutes les données ont été remplacées par des point d'interrogation. Dans le cas de **findByName**, cela donnerait :

```
SELECT * FROM sport WHERE name LIKE ? ORDER BY name;
```

Puis, vous allez indiquer au **PreparedStatement** quelle donnée doit être associée à chaque ? et dans quel format :

```
myPreparedStatement.setString(1, stringData); // Associe une string au premier ?
myPreparedStatement.setInt(2, intData);       // Associe un int au second ?
```

L'intérêt de cette technique est que si la donnée à intégrer contient des caractères interprétables par SQL (comme **'** dans l'injection SQL précédente), **PrepraredStatement** va automatiquement les convertir en caractères inoffensifs : **'** devient **\'**.

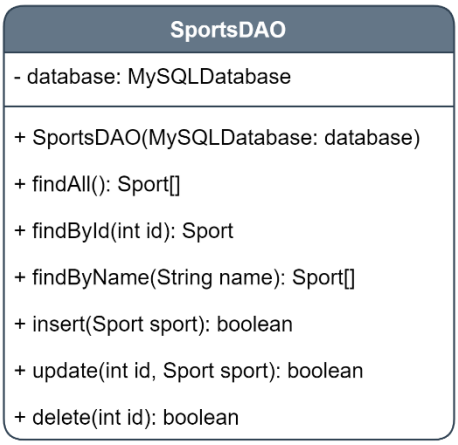
- Dans la classe **MySQLDatabase**, ajoutez une méthode **prepareStatement** qui prendra en paramètre une chaine de caractère représentant la requête à exécuter et retournera un **PreparedStatement**.

  L'objet **Connection** possède une méthode **prepareStatement** qui retourne un **PreparedStatement**.

- Modifiez les fonctions **findById** et **findByName** de la classe **SportsDAO** afin d'utiliser des **PreparedStatement** à la place des **Statement**.
- Utilisez PHPMyAdmin pour ajouter de nouvelles données dans la table **sport** puis testez le bon fonctionnement et vérifiez que l'injection SQL n'est plus possible.

BONUS DU DEVELOPPEUR

- Complétez la classe **`SportsDAO`** à l'aide de la modélisation suivante :



- Testez le bon fonctionnement des méthodes **`insert`**, **`update`** et **`delete`**.