POLYTECH DIJON

TRAVAUX DIRIGÉS

Développement d'applications Web

JAVA – JDBC & SQLite

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

Nouveau projet

➢ Créez un nouveau projet Java.

> **Note**
> Si vous utilisez Visual Studio Code, la méthode pour prendre en charge Java est expliquée en Annexe 1.

Driver SQLite

Java a besoin de connaître la procédure à suivre pour dialoguer correctement avec votre SGBD. Pour cela, nous allons lui donner le driver à utiliser.

➢ Connectez-vous sur le dépôt officiel du driver SQLite pour Java :

https://github.com/xerial/sqlite-jdbc

➢ Téléchargez le fichier sqlite-jdbc-[last-version].jar
➢ Ajoutez le fichier jar téléchargé précédemment comme bibliothèque externe de votre projet.

> **Note**
> Si vous utilisez Visual Studio Code, la méthode pour importer une bibliothèque externe est expliquée en Annexe 2.

BASE DE DONNEES

- Téléchargez l'archive db.zip.
- Et extrayez le fichier **db.sqlite** dans le dossier de votre projet (à la racine).

PREMIERS PAS AVEC JDBC

Connection

Java utilise JDBC (Java Database Connectivity) pour créer une liaison avec des bases de données de toutes sortes.

Le driver que vous avez téléchargé précédemment va permettre à JDBC de savoir comment parler à une base de données SQLite pour pouvoir effectuer des requêtes SQL.

Pour initier une connexion, on utilise la syntaxe suivante :

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

La chaine de connexion diffère d'un SGBD à un autre. Pour SQLite, elle est construite comme ceci :

```
jdbc:sqlite:path_to_database_file
```

- A partir des éléments précédents, modifiez la fonction main de votre programme pour ouvrir une connexion avec la base de données téléchargées précédemment.

> **Note**
> N'oubliez pas de prendre en charge les exceptions pouvant être levées.

- Exécutez votre programme et vérifiez qu'aucune exception ne soit levée.

Première requête

Statement

Pour effectuer une requête vers la base de données, JDBC 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();
```

Requête

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("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**.

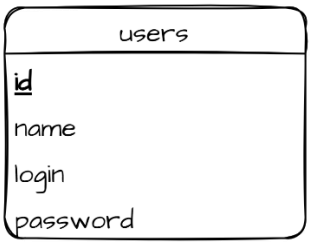
Une fois positionné sur un enregistrement, le **ResultSet** propose des méthodes permettant de récupérer la valeur d'une colonne pour l'enregistrement courant en fonction du type de la donnée :

```
//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 **users** et affiche pour chaque utilisateur son nom et son login.

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



- Testez le bon fonctionnement.

STRUCTURONS TOUT CELA

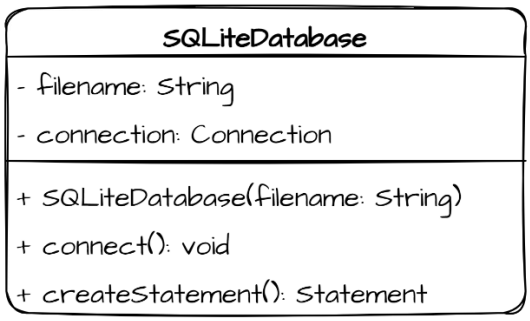
Voilà, vous savez à présent comment vous connecter à une base de données MySQL avec Java. Mais bon, le code que vous venez de faire passe dans le cadre d'un test rapide de faisabilité. Pour une application plus complexe (pensez au projet de Web qui arrive bientôt), autant dire que ce ne sera pas possible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

SQLiteDatabase

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



  - Le constructeur de la classe initialisera les attributs de cette dernière dont **`connection`** qui sera initialisée à **`null`**
  - 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`**).

> **Note**
> La classe **`SQLiteDatabase`** ne s'occupera pas du traitement des exceptions de type **`SQLException`**. Elle se contentera de les propager au niveau supérieur.

> **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.

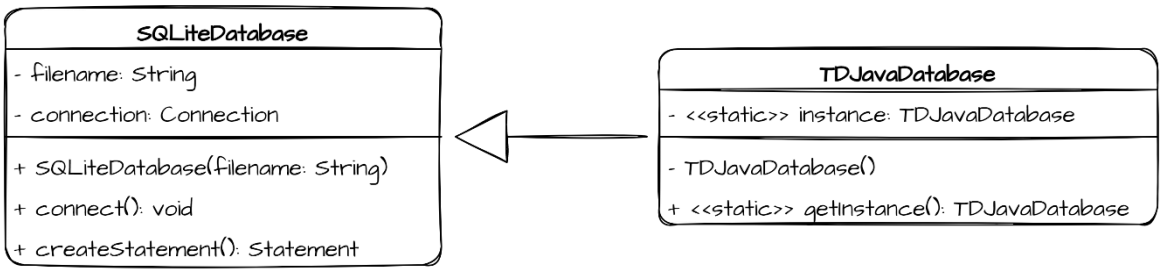
- Modifiez le code de la fonction **`main`** pour que la connexion à la base de données et la création du **`Statement`** soit pris en charge par une instance de la classe **`SQLiteDatabase`**.
- Testez le bon fonctionnement.

TDJavaDatabase

Vous allez à présent créer la classe **`TDJavaDatabase`** qui hérite de **`SQLiteDatabase`** et configurera automatiquement la connexion vers la base de données du TD. De plus, **`TDJavaDatabase`** sera un singleton, ce qui nous assurera que quel que 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. Comme toujours, c'est le projet qui détermine si une solution est adaptée ou non.
>
> Ici, nous allons simplement profiter de l'occasion pour manipuler un Design Pattern.

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



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

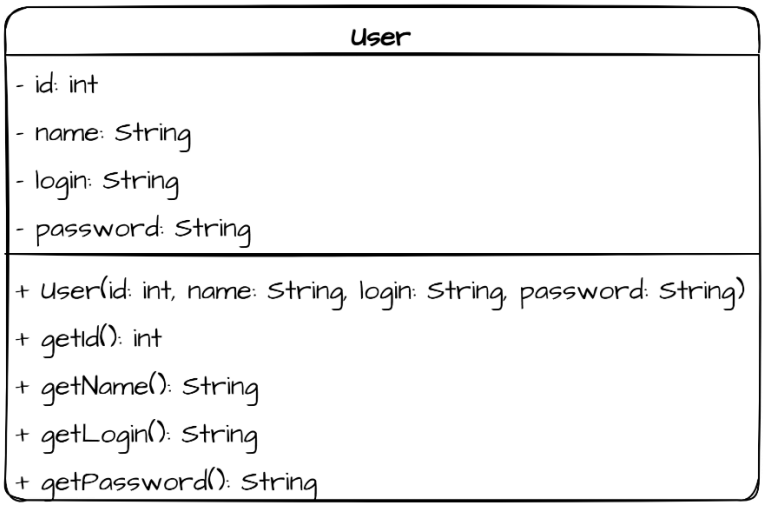
- Le constructeur de **`TDJavaDatabase`** (qui est privé) appellera le constructeur de la classe parent en lui transmettant les éléments de connexion propre à votre base de données.

➢ Remplacez l'utilisation de **`SQLiteDatabase`** dans la fonction main par **`TDJavaDatabase`**.

➢ Testez le bon fonctionnement.

User

➢ Implémentez la classe User ci-dessous :



> **Astuce**
> Votre IDE est en mesure de générer automatiquement les getters. Pensez-y.

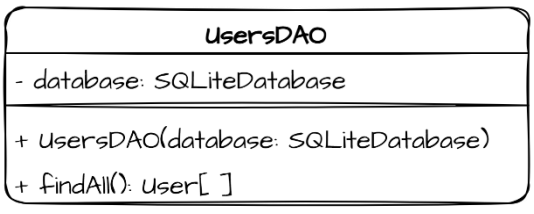
➢ Dans la fonction main, créez une instance de la classe **`User`** et affichez la valeur de ces attributs afin de tester le bon fonctionnement.

UsersDAO

Jusque-là vous avez d'un côté la classe **`TDJavaDatabase`** qui gère la connexion avec la base de données et la classe **`User`** qui représente un utilisateur. Vous allez à présent créer une classe qui fera le lien entre votre objet métier (**`User`**) 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 **`UsersDAO`** 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 utilisateurs trouvés dans la base de données ou un **ArrayList** vide si une erreur se produit.

➢ Modifiez le code de la fonction main pour remplacer le code restant utilisant un **Statement** par une instance de la classe **UsersDAO**

➢ Testez le bon fonctionnement.

findById

➢ Ajoutez à la classe **UsersDAO** une méthode **findById** qui prendra en paramètre l'id de l'utilisateur recherché et retournera l'utilisateur trouvé s'il existe.

> **Note**
> Le type **Optional** de Java est une bonne alternative à **null** lorsque l'on n'est pas certain de pouvoir retourné ce qui nous est demandé, et que l'on ne souhaite pas gérer une exception.

```
Optional<SomethingType> functionWhichMayReturnSomething()
{
    if(condition == true)
        return Optional.of(value);
    else
        return Optional.empty();
}

Optional<SomethingType> value = functionWhichMayReturnSomething() ;

if(value.isPresent())
    // Something has been returned
else
    // Nothing has been returned
```

➢ Modifiez la fonction **main** afin de tester la fonction **findById** de **UsersDAO** avec un id qui existe et un id qui n'existe pas.

➢ Testez le bon fonctionnement.

findByCredentials

➢ Dans la même idée, ajoutez à la classe **UsersDAO** une méthode **findByCredentials** qui prendra en paramètre deux chaines de caractères (**login** et **password**) et qui retournera l'utilisateur correspondant aux identifiants fournis.

➢ Modifiez la fonction **main** afin de tester la fonction **findByCredentials** de **UsersDAO** avec le login **chris** et le mot de passe **wifi**.

➢ Testez le bon fonctionnement en affichant « Connexion OK » si un utilisateur est trouvé et « Utilisateur inconnu » dans le cas contraire.

➢ Modifiez à nouveau la fonction **main** afin que l'utilisateur puisse saisir le login et le mot de passe de l'utilisateur 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 les identifiants vus précédemment.

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 dans ce monde des utilisateurs qui ont souvent l'idée saugrenue de ne pas faire ce qu'on leurs demande et parfois bien pis encore.

- Exécutez de nouveau votre programme et saisissez la chaine de caractères suivante comme login et ce que vous voulez comme mot de passe :

```
' OR 1 --
```

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 **`findByCredentials`**, cela donnerait :

```
SELECT * FROM users WHERE login = ? AND password = ?;
```

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

```
// Associe une string au premier ?
myPreparedStatement.setString(1, stringData);

// Associe un int au second ?
myPreparedStatement.setInt(2, intData);
```

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 `\'`.

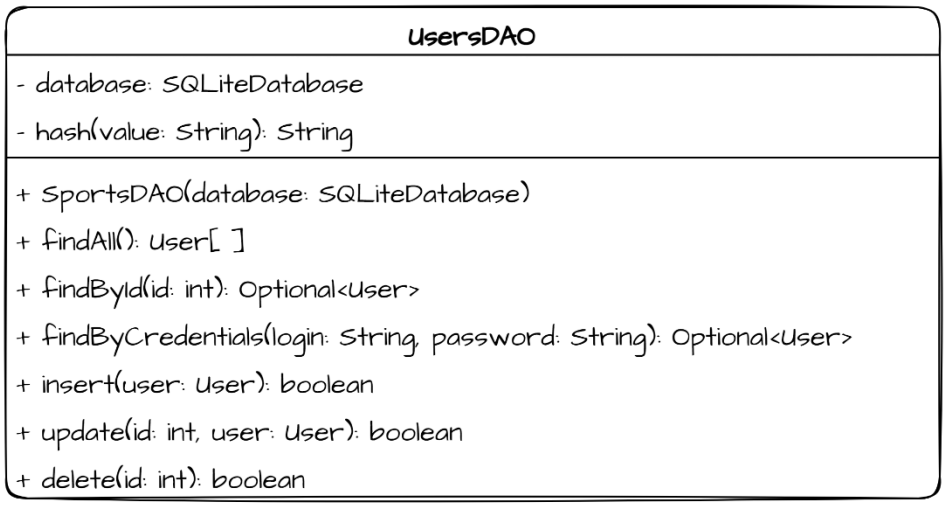
- Dans la classe **`SQLiteDatabase`**, ajoutez une méthode **`prepareStatement`** qui prendra en paramètre une chaine de caractères représentant la requête à exécuter et retournera un **`PreparedStatement`**.

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

- Modifiez les fonctions **`findById`** et **`findByCredentials`** de la classe **`UsersDAO`** afin d'utiliser des **`PreparedStatement`** à la place des **`Statement`**.
- Testez le bon fonctionnement et vérifiez que l'injection SQL n'est plus possible.

BONUS DU DEVELOPPEUR

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



  - **`hash`** générera le hachage d'une chaine de caractères. Idéal pour ne pas stocker les mots de passe en clair dans la base de données.
  - **`insert`** ajoutera un nouvel utilisateur à la base de données.
  - **`update`** mettra à jour les données de l'utilisateur dont l'id est fourni.
  - **`delete`** supprimera l'utilisateur dont l'id est fourni.

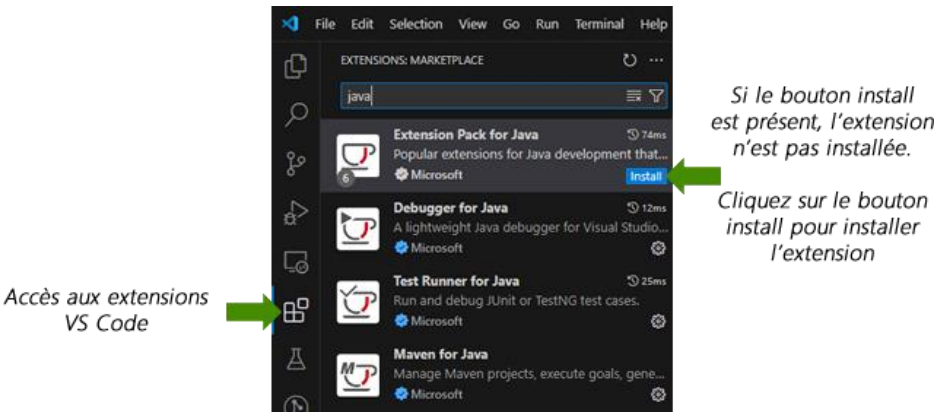
- Testez le bon fonctionnement des méthodes **`insert`**, **`update`** et **`delete`**.

ANNEXES

Annexe 1 – Prise en charge de Java avec Visual Studio Code

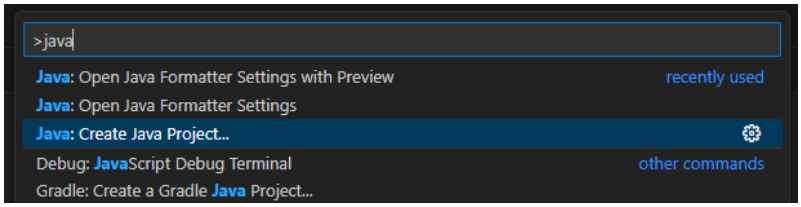
Prise en charge de Java AVEC VSCode

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

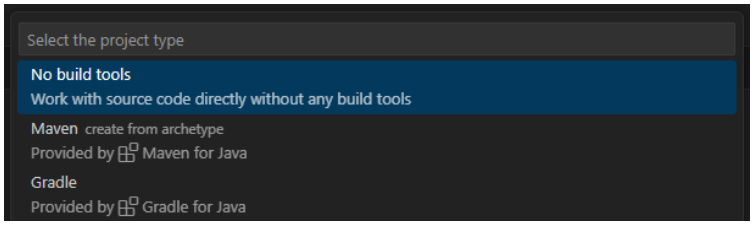


Nouveau projet

- 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...`**.

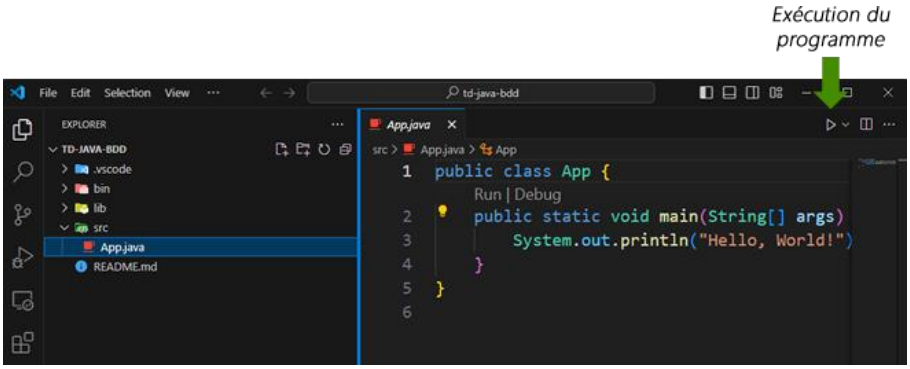


- Sélectionnez **`No build tools`**



- 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).

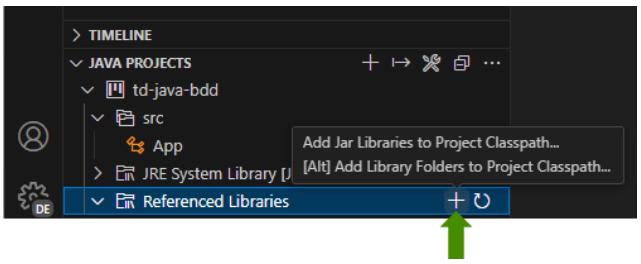
Une nouvelle fenêtre VSCode apparaît avec une arborescence de projet prête à l'emploi :



- Sélectionnez le fichier **App.java** présent dans le dossier **src** et cliquez sur le bouton permettant de démarrer le programme.

Annexe 2 – Importer une bibliothèque externe

- Sélectionnez le fichier **src/App.java** de votre projet. Cela active l'extension Java de VS Code.
- 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 jar correspondant à la bibliothèque que vous souhaitez importer.