esprit
Se former autrement

Projet d'intégration GL JAVA WEB MOBILE

Connexion à la base de données - JDBC

ESPRIT - UP JAVA

Année universitaire 2019/2020

cdio
CONCEIVE DESIGN IMPLEMENT OPERATE

European Accreditation of Engineering Programmes EUR-ACE®

Délivré par Cti Commission des Titres d'Ingénieur

CONFÉRENCE DES GRANDES ÉCOLES

JDBC

Avant JDBC...

- Avant JDBC, il était difficile d'accéder à des bases de données SQL:
  - Utilisation de librairies C/C++
  - Utilisation d'API natives comme ODBC

- Problème majeur
  - dépendance totale avec le SGBD utilisé

SGBD?

- En informatique, un système de gestion de base de données (*abr. SGBD*) est un logiciel système servant à stocker, à manipuler ou gérer, et à partager des informations dans une base de données, en garantissant la qualité, la pérennité et la confidentialité des informations, tout en cachant la complexité des opérations.

- Exp: MySQL, Oracle DataBase, PostgreSQL,etc...

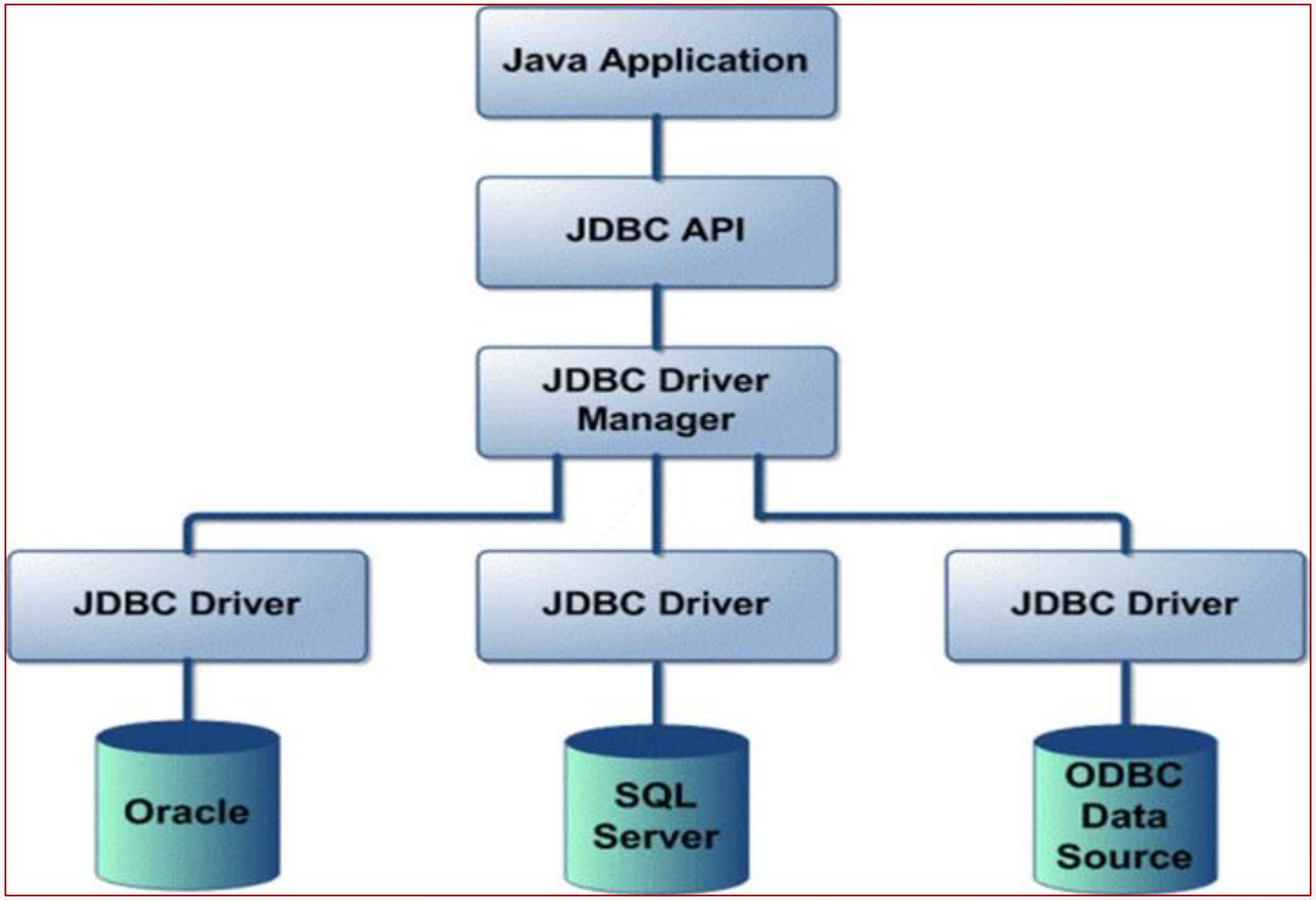
JDBC: Java DataBase Connectivity

- **API** d'interaction avec un **SGBD** contenant :
- un ensemble de classes et d'interfaces

- Permet de:
    1. Établir une connexion avec un SGBD
    2. Envoyer des requêtes SQL
    3. Récupérer des résultats de requêtes

⇒ Permettre aux programmeurs Java d'écrire un code indépendant de la base de données et du moyen de connexion utilisé

JDBC: Java DataBase Connectivity



JDBC: Java DataBase Connectivity

- Drivers

- chaque SGBD utilise un pilote (driver) qui lui est propre et qui permet de convertir les requêtes JDBC dans le langage natif du SGBD

- le driver est un ensemble de classes qui implantent les interfaces de JDBC

- les drivers sont le lien entre le programme Java et le SGBD

JDBC: Java DataBase Connectivity

- API JDBC

- Interface uniforme permettant un accès homogène aux SGBD

- Simple à mettre en œuvre

- Indépendant du SGBD support

- Supportant les fonctionnalités de base du langage SQL

Mise en place

JDBC: Mise en place

- Importer le package en **java.sql.***
- Etablir la connexion au SGBD
- Créer une requête (ou instruction SQL)
- Exécuter la requête

NB: Vous devez télécharger la dernière version de l'IDE IntelliJ

JDBC: Mise en place

- Etape 1 :Base de données MySQL

- Créer une base mysql ‘esprit’ et une table personne (avec 3 champs: id, nom et prenom)

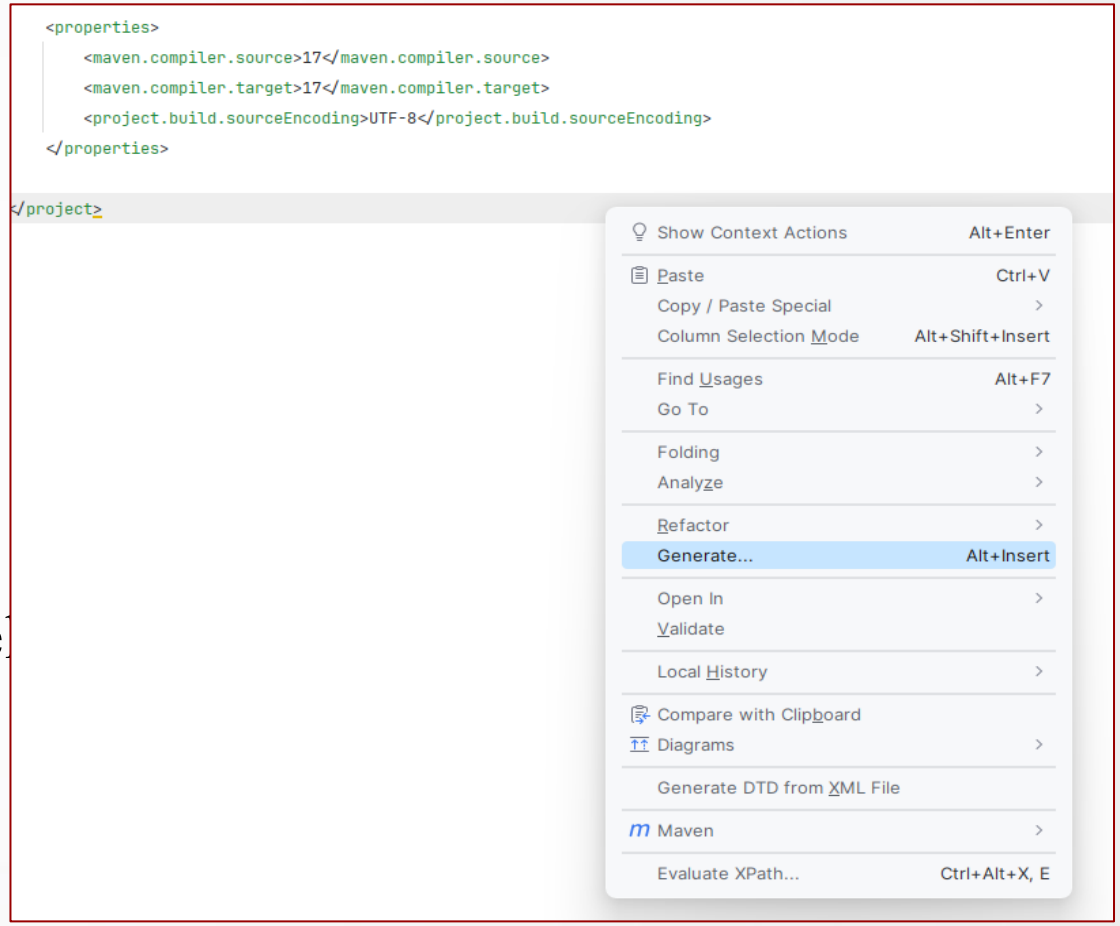
	Champ	Type	Interclassement	Attributs	Null	Défaut	Extra
	id	int(11)			Non	*Aucun*	auto_increment
	nom	varchar(20)	latin1_swedish_ci		Non	*Aucun*	
	prenom	varchar(20)	latin1_swedish_ci		Non	*Aucun*	

JDBC: Mise en place

- Etape 2 : MySQL JDBC Driver

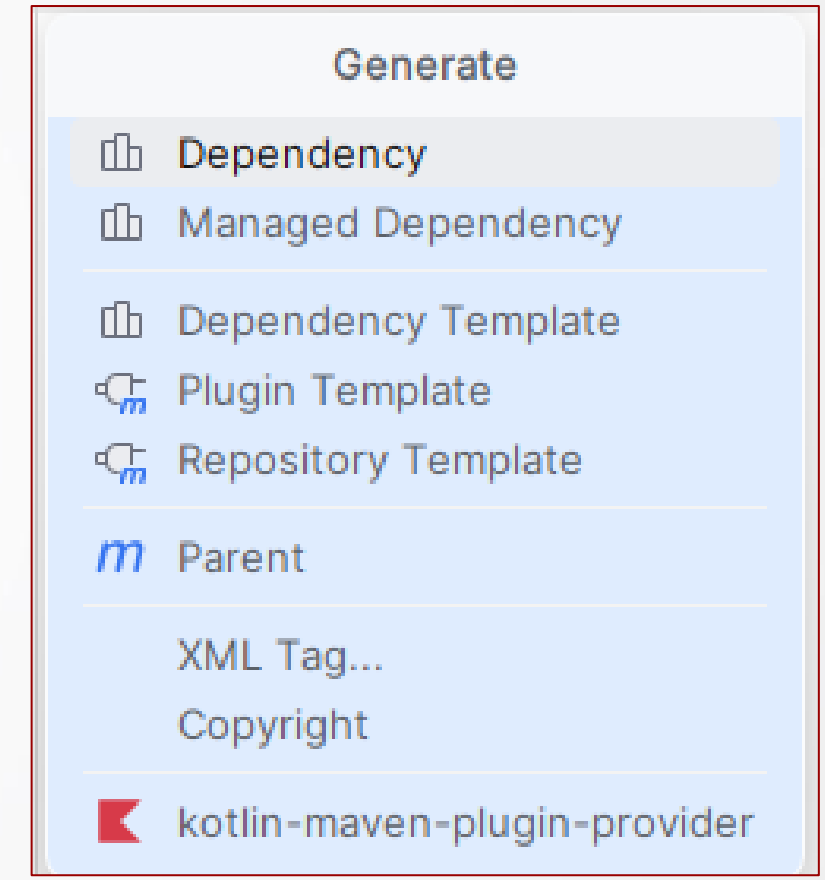
- Ouvrez le fichier « **pom.xml** » de votre projet.

- Faites un clic droit sur le fichier et sélectionnez l'option « **Generate …**» dans le menu contextuel



JDBC: Mise en place

- Choisissez « **Dependency** » dans la liste des options. Cela ouvrira une fenêtre intitulée « **Maven Artifact Search** ».
- Dans l'onglet « **Search For Artifact** », saisissez «**mysql:mysql-connector-java** » et choisissez la dernière version disponible.
- Appuyez sur le bouton « **Load Maven Changes** » pour télécharger et installer automatiquement la bibliothèque dans votre projet.



JDBC: Mise en place

- Etape 3 : Etablir une connexion

  - Création d'un objet de type Connection : conn
  - conn = DriverManager.getConnection(url, user, pwd);

- Etape 4 : Traiter les exceptions

  - SQLException

JDBC: Mise en place

- Etape 5 : Les attributs de connexion

  - String url = "jdbc:mysql://localhost:3306/esprit";
  - String user = "root";
  - String pwd = "";
  - Connection conn;
  - Statement ste;

JDBC: Création d'un statement

- L'interface **Statement** possède les méthodes nécessaires pour réaliser les requêtes sur la base

  ⇒ exécuter des instruction SQL

- **2 types de Statement :**
  - Statement ⇒ requêtes statiques simples
  - PreparedStatement ⇒ requêtes dynamiques précompilées

JDBC: Statement Vs. Prepared Statement

- Lors de l'envoi d'une requête pour exécution 4 étapes doivent être faites :
    - analyse de la requête
    - compilation de la requête
    - optimisation de la requête
    - exécution de la requête

⇒ et ceci même si cette requête est la même que la précédente !! Or les 3 premières étapes ont déjà été effectuées dans ce cas.

JDBC: Statement Vs. Prepared Statement

- Les bases de données définissent la notion de requête préparée, requête où les 3 premières étapes ne sont effectuées qu'une seule fois.

- JDBC propose l'interface **PreparedStatement** pour modéliser cette notion

JDBC: Création d'un statement

- Créer un STATEMENT

- ste = conn.createStatement();

- Requête SQL d'ajout

- String req = *"Insert into personne values(12,'Tounsi','Wael')";*

- Exécuter la Requête

    - ste.executeUpdate(req) ⇒ (insert, update, delete)

    - ste.executeQuery(req) ⇒ select

Créer une classe Personne

- Créer une classe

  Personne.java

```
public class Personne {
    public int id;
    public String nom;
    public String prenom;

    public Personne(int id, String nom, String prenom) {
        this.id = id;
        this.nom = nom;
        this.prenom = prenom;
    }
    public String toString() {
        return "id=" + id + ", nom=" + nom + ""
                + ", prenom=" + prenom + '}';
    }
}
```

JDBC: Méthodes CRUD

- Créer une classe **ServicePersonne** qui contient :
  - public void **ajouter**(Personne per) throws SQLException{ }
  - public void **updatePrenom**(Personne per) throws SQLException{ }
  - public void **delete**(int id) throws SQLException{ }
  - public Arraylist<Personne> **afficherAll**() throws SQLException{ }

⇒ Utiliser le **ResultSet** pour récupérer le résultat de executeQuery

JDBC: Fermeture des connexions

- Pour terminer proprement un traitement, il faut fermer les différents espaces ouverts

  - **resultset.close();**
  - **statement.close();**
  - **connection.close();**

Correspondance Java / BD

Type JDBC/SQL	Méthode Java
CHAR, VARCHAR	getString()
BINARY, VARBINARY	getBytes()
BIT	getBoolean()
INTEGER	getInt()
BIGINT	getLong()
SMALLINT	getShort()
REAL	getFloat()
DOUBLE, FLOAT	getDouble()
DATE	getDate()
TIME	getTime()
TIME STAMP	getTimeStamp()

Requêtes précompilées : PreparedStatemt

- **PreparedStatemt** : envoie une requête sans paramètres à la base de données:

  - plus rapide qu'un Statement classique
  - le SGBD n'analyse qu'une seule fois la requête

⇒ requêtes dynamiques précompilées (avec paramètres d'entrée/sortie)

- La méthode **prepareStatement()** de l'objet Connection crée un PreparedStatement

Requêtes précompilées : PreparedStatemt

```
PreparedStatement ps = c.prepareStatement("SELECT *
   FROM Table-name "+ "WHERE att = ? ");
```

- les paramètres sont spécifiés par un " ? "
  - ils sont ensuite instanciés par les méthodes setInt(), setString(), setDate()…
  - ces méthodes nécessitent 2 arguments (setInt(n, valeur))
    - n(int) : le numéro relatif de l'argument dans la requête
    - Valeur : la valeur à positionner

Insertion

```
ste = connexion.createStatement();
String req = "INSERT INTO `esprit`.`personne` (`id` ,`nom`
                                                ,`prenom`)
                VALUES (NULL , 'Test', 'Test');";

 ste.executeUpdate(req);
```

Insertion

```
PreparedStatement stm = connexion.prepareStatement
        ("insert into personne (nom,prenom) values (?,?)");

            stm.setString(1,"Esprit");
            stm.setString(2,"Esprit");
            stm.executeUpdate();
```

Sélection

```
String requête = "SELECT * FROM `personne`";
        ResultSet res = ste.executeQuery(requete);
         while (res.next()) {
             System.out.println(res.getInt (1));
             System.out.println(res.getString(2));
         }
```

Modification

```
String rq = "UPDATE `esprit`.`personne` SET `nom` = '"
                    +varNom+"' WHERE `personne`.`id`
                ='"+varId+"';";


ste.executeUpdate(rq);
```

Suppression

```
String rq = "DELETE FROM `esprit`.`personne` WHERE
                                `personne`.`id` ='"+id+"';";

ste.executeUpdate(rq);
```

Merci pour votre attention