JAVA Avancé
JDBC

Introduction: Principe

- Les systèmes de base de données:

  - Assure le traitement des fichiers
  - Organisent les données de manière à faciliter l'obtention des résultats d'une requête complexe.
  -Les bases de données les plus courantes dans les ordinateurs qui utilisent Java sont les bases de données relationnelles.

- Java permet donc aux programmeurs d'écrire du code qui met en œuvre les requêtes SQL pour retrouver des renseignements dans des bases de données relationnelles, mais ce n'est pas tout. Le langage Java est un langage universel qui peut fonctionner sur différentes plates-formes.

Introduction

- JDBC : Java Data Base Connectivity.
- Framework permettant l'accès aux bases de données relationnelles dans un programme Java

  * Indépendamment du type de la base utilisée (MySQL, Oracle, Postgres ...) Seule la phase de connexion au SGBDR change.

  * Permet de faire tout type de requêtes :

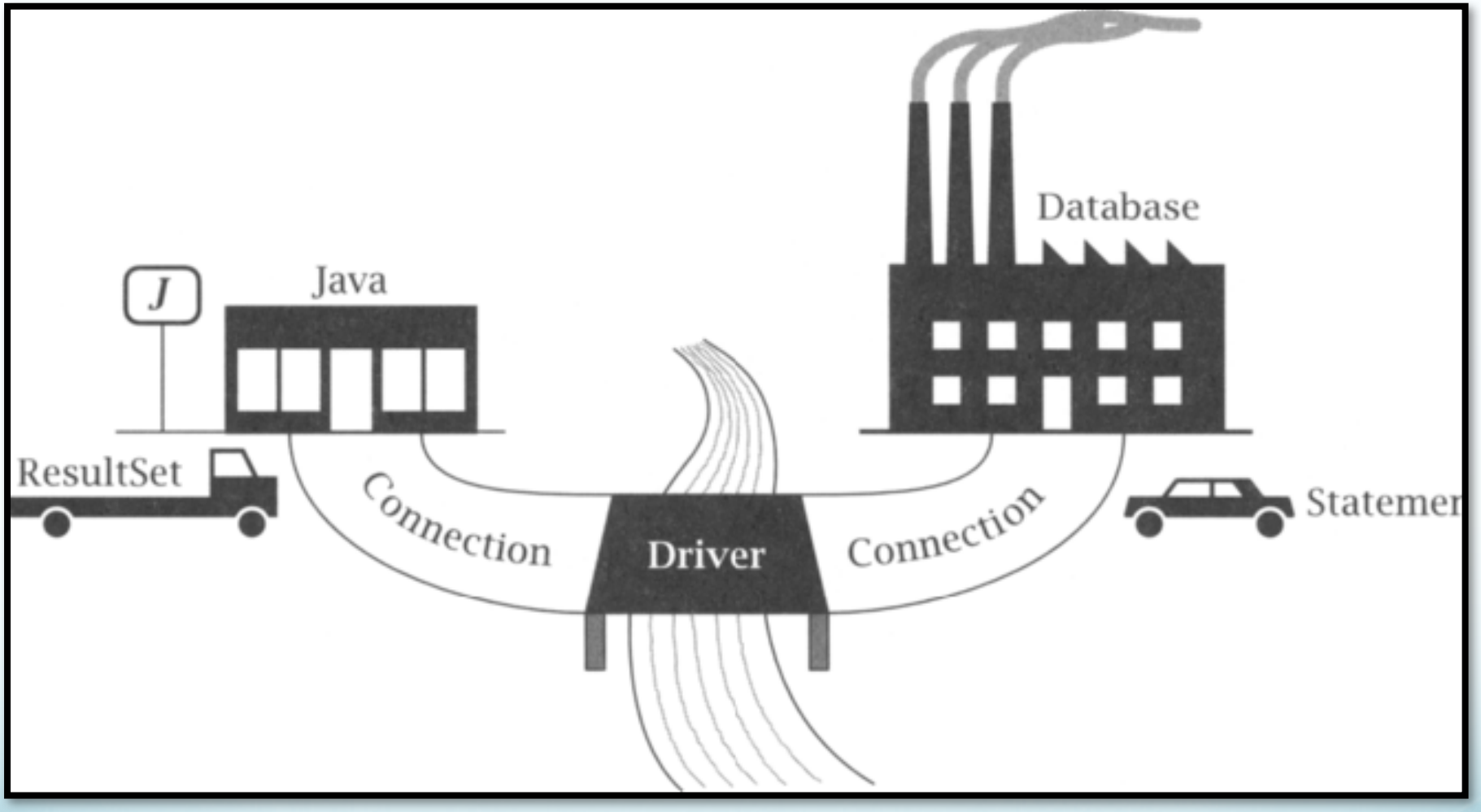
    - Sélection de données dans des tables

    - Création de tables et insertion d'éléments dans les tables

    - Gestion des transactions

- Packages : java.sql et javax.sql.

Introduction



Principes généraux d'accès à une BDD

- Etape 1 : Charger le driver que l'on veut utiliser

  -Driver permet de gérer l'accès à un type particulier de SGBD.

- Etape 2 : Récupérer un objet « Connection » en s'identifiant auprès du SGBD et en précisant la base utilisée.

- Etape 3 : A partir de l' Objet Connection on créé le Statement(état) qui correspond à une requête.

- Etape 4 :Exécuter ce Statement au niveau du SGBD

- Etape 5 : close le Statement

- Etape 6 : Se déconnecter de la base en fermant la connexion.

Etape1: Driver

- Comment charger le driver?

```
Class.forName(String driver) throws ClassNotFoundException
```

signifie faire construire par la JVM un objet Class contenant toutes les interfaces du programme choisi

- Pour une base Mysql

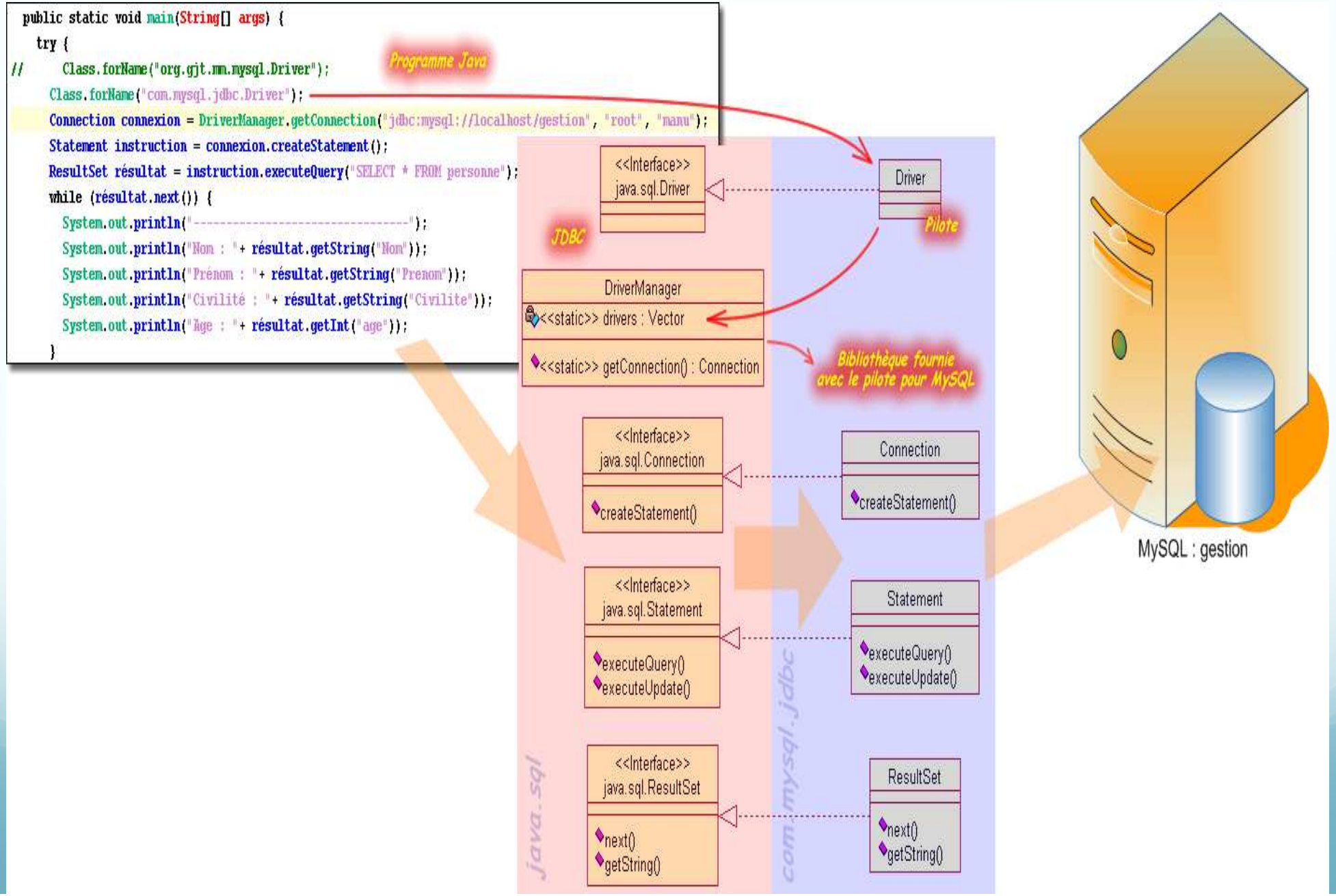
```
try {
    Class.forName("com.mysql.jdbc.Driver"). newInstance();
} catch(ClassNotFoundException ex) {
    System.out.println("Can not load the Driver");
}
```

Etape2:Connexion au SGBD

- Classe **java.sql.DriverManager** : Gestion du contrôle et de la connexion au SGBD.
- La méthode **static Connection getConnection(String URL, String user, String password) :** Crée une connexion permettant d'utiliser une base
- **URL :** Identification de la base considérée sur le SGBD
  le Format de l'URL est dépendant du SGGB utilise.
  - **jdbc:mysql://<hostname>/<databaseName>**
  - Exemple: **jdbc:mysql://localhost/SunGamer**
- **user** : nom de l'utilisateur qui se connecte à la base.
- **password** : mot de passe de l'utilisateur.

Connexion au SGBD

```java
public static void main(String[] args) {
  try {
//      Class.forName("org.gjt.mm.mysql.Driver");
    Class.forName("com.mysql.jdbc.Driver");
    Connection connexion = DriverManager.getConnection("jdbc:mysql://localhost/gestion", "root", "manu");
    Statement instruction = connexion.createStatement();
    ResultSet résultat = instruction.executeQuery("SELECT * FROM personne");
    while (résultat.next()) {
      System.out.println("------------------------------");
      System.out.println("Nom : "+ résultat.getString("Nom"));
      System.out.println("Prénom : "+ résultat.getString("Prenom"));
      System.out.println("Civilité : "+ résultat.getString("Civilite"));
      System.out.println("Age : "+ résultat.getInt("age"));
    }
```



Etape3:Gestion des connexions

- **Interface java.sql.Connection**

  Préparation de l'exécution d'instructions sur la base, 2 types :

1. Instruction simple : classe **Statement**
   On exécute directement et une fois l'action sur la base
2. Instruction paramétrée : classe **PreparedStatement**

- L'instruction est générique, des champs sont non remplis
- Permet une pré-compilation de l'instruction optimisant les performances
- Pour chaque exécution, on précise les champs manquants

Gestion des connexions

- *La méthode* « **Statement createStatement() »** Retourne un état permettant de réaliser une instruction simple.
- La méthode **PreparedStatement prepareStatement(String ordre)**

1. Retourne un état permettant de réaliser une instruction paramétrée et pré-compilée pour un ordre

2. Dans l'ordre, les champs libres (au nombre quelconque) sont précisés par des « ? ». Ex : **"select nom from clients where ville=?' '**

- *En fin, La méthode* **void close() :** Ferme la connexion avec le SGBD.

Instruction simple

- Classe **Statement**
- La méthode **ResultSet executeQuery(String ordre) :** Exécute un ordre de type SELECT sur la base et Retourne un objet de type **ResultSet** contenant tous les résultats de la requête.
- La méthode **int executeUpdate(String ordre)** : Exécute un ordre de type INSERT, UPDATE, ou DELETE.
- La méthode **void close() :** Fermeture de l'état.

```
Statement stmt = myConnection.createStatement();
ResutSet rs = stmt.executeQuery("SELECT * FROM Peronne");
```

Instruction paramétrée

- Classe PreparedStatement.

  Avant d'exécuter l'ordre, on remplit les champs avec

1. **void set[Type](int index, [Type] val)**
2. Remplit le champ en ième position définie par index avec la valeur val de type [Type]
   -[Type] peut être : String, int, float, long

```
PreparedStatement pstmt =
myConnection.prepareStatement("SELECT * FROM Personne
WERE lastname = ? AND firstname = ?");
pstmt.setString(1, "GEORGES");
pstmt.setString(2, "Ron");
```

Lecture des résultats

- Classe ResultSet : Contient les résultats d'une requête SELECT

1. Plusieurs lignes contenant plusieurs colonnes
2. On y accède ligne par ligne puis valeur par valeur dans la ligne.

- La méthode **boolean next()** : Se place à la ligne suivante s'il y en a une (Iterateur).
- La méthode **boolean previous() :** Se place à la ligne précédente s'il y en a une
- La methode **boolean absolute(int index)** Se place à la ligne numérotée index.

Lecture des résultats

- Accès aux colonnes/données dans une ligne.
- La méthode **[type] get[type](int col)** : Retourne le contenu de la colonne col.
- Exemple : **String getString(int col)**
- La méthode **void close():** Fermeture du ResultSet.

ResultSet

- Exemple

```
// ...
ResultSet rs = stmt.executeQuery("SELECT * FROM Personne");

while(rs.next()) {
    int id = rs.getInt(1);
    String name = rs.getString("name");
    // ...
}
// ...
```

Stockage d'objet type(BLOB)

```
 */
public class Personne implements Serializable {
   private String nom;
   private String prénom;
   private int age;

   /**
    * @param nom
    * @param prenom
    * @param age
    */
   public Personne(String nom, String prénom, int age) {
      this.nom = nom;
      this.prénom = prénom;
      this.age = age;
   }
   /**
    * @return Renvoie age.
    */
   public int getAge() {
      return age;
   }
   /**
    * @return Renvoie nom.
    */
   public String getNom() {
      return nom;
   }
   /**
    * @return Renvoie prenom.
    */
   public String getPrénom() {
      return prénom;
   }
}
```

Stockage d'objet (suite)

```
public class BaseDeDonnées {

   public static void main(String[] args) {
      try {
         Class.forName("com.mysql.jdbc.Driver");
         Connection connexion = DriverManager.getConnection("jdbc:mysql://localhost/gestion", "root", "manu");

         //Création de l'objet et stockage dans la base de données
         Personne personne = new Personne("REMY", "Emmanuel", 45);

         String sql = "INSERT INTO messagerie (personne, message) VALUES (?,?)";
         PreparedStatement statement = connexion.prepareStatement(sql);

         //insertion de l'objet
         statement.setObject(1, personne);
         statement.setString(2, "Bienvenue à tous");
         statement.executeUpdate();

      }
      catch (ClassNotFoundException e) {
         System.err.println("Le driver n'a pas été chargé");
      }
      catch (SQLException e) {
         System.err.println("La requête n'a pas aboutie");
      }
```

Stockage d'objet (suite)

```
public class BaseDeDonnées {

   public static void main(String[] args) {
      try {
         Class.forName("com.mysql.jdbc.Driver");
         Connection connexion = DriverManager.getConnection("jdbc:mysql://localhost/gestion", "root", "manu");

         // Récupération de l'objet stocké dans la base de données
         Statement instruction = connexion.createStatement();
         ResultSet résultat = instruction.executeQuery("SELECT * FROM messagerie");
         résultat.next();
         InputStream stream = résultat.getBlob("personne").getBinaryStream();
         ObjectInputStream objet = new ObjectInputStream(stream);
         Personne personne = (Personne) objet.readObject();
         System.out.println("Nom : "+ personne.getNom());
         System.out.println("Prénom : "+ personne.getPrénom());
         System.out.println("Age : "+ personne.getAge());
         System.out.println("Message : "+ résultat.getString("message"));
      }
      catch (ClassNotFoundException e) {
         System.err.println("Le driver n'a pas été chargé");
      }
      catch (SQLException e) {
         System.err.println("La requête n'a pas aboutie");
      }
      catch (IOException e) {
         System.err.println("Problème de flux");
      }
   }
}
```

Gestion des Transactions

Transaction

- Par défaut, après chaque instruction exécution SQL, les modifications sont automatiquement effectuées à la BD.
- Pour gérer soi-même les transactions

```
connection.setAutoCommit(false);
```

- Pour valider

```
Connection.commit();
```

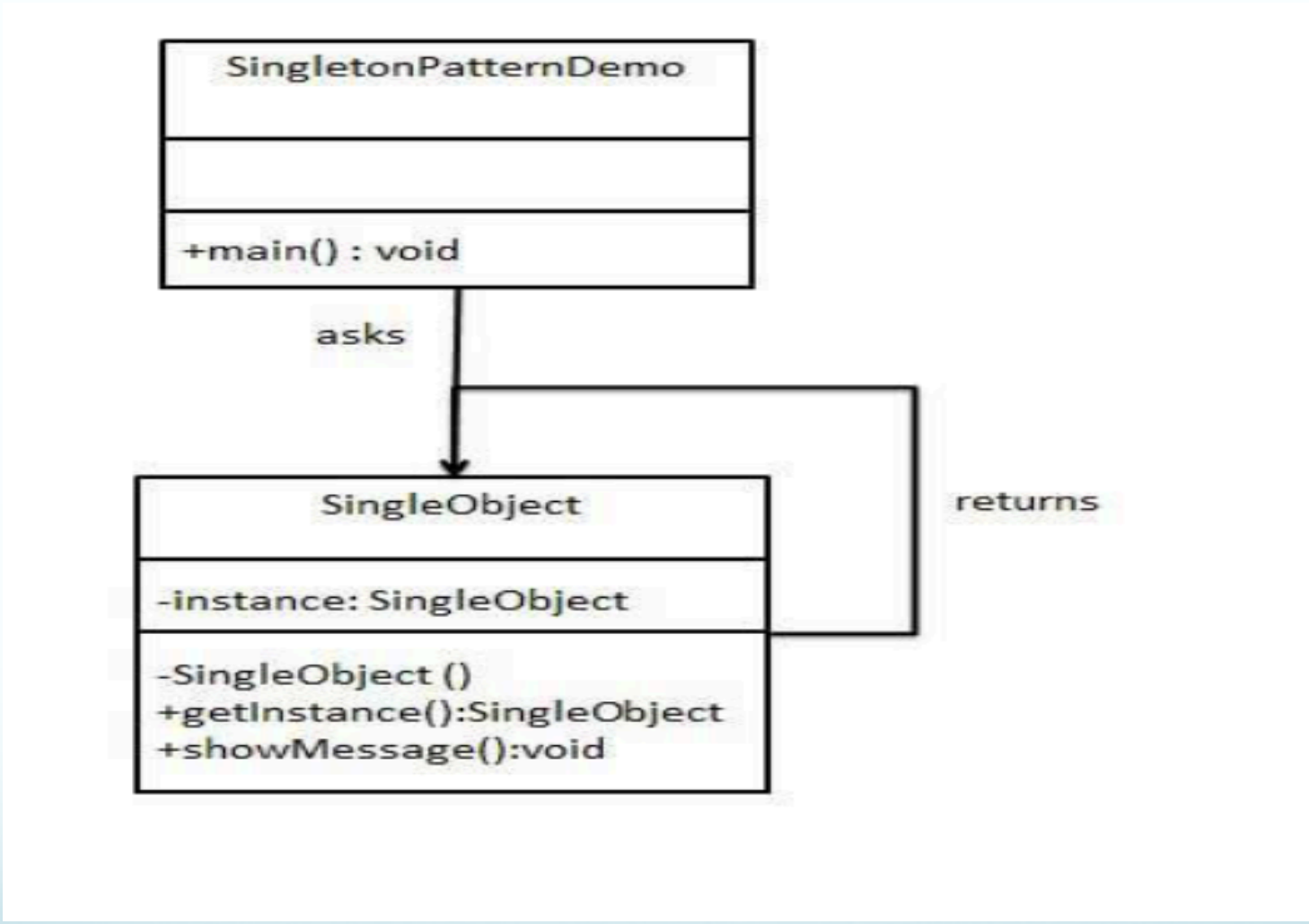
- Pour annuler

```
connection.rollback();
```

Exemple

```
Connection connection =
DriverManager.getConnection(url, userProperties);
connection.setAutoCommit(false);
try {
    statement.executeUpdate(...);
    statement.executeUpdate(...);
    ...
    connection.commit();
    } catch (Exception e) {
        try {
            connection.rollback();
              } catch (SQLException sqle) {
              // report problem
                   }
     }
finally {
         try {
              connection.close();
             } catch (SQLException sqle) { }
   }
```

Design Pattern: Singleton



Singleton (suite)

SingleObject.java

```
public class SingleObject {

   //create an object of SingleObject
   private static SingleObject instance = new SingleObject();

   //make the constructor private so that this class cannot be
   //instantiated
   private SingleObject(){}

   //Get the only object available
   public static SingleObject getInstance(){
      return instance;
   }

   public void showMessage(){
      System.out.println("Hello World!");
   }
}
```

Step 2

Get the only object from the singleton class.

SingletonPatternDemo.java

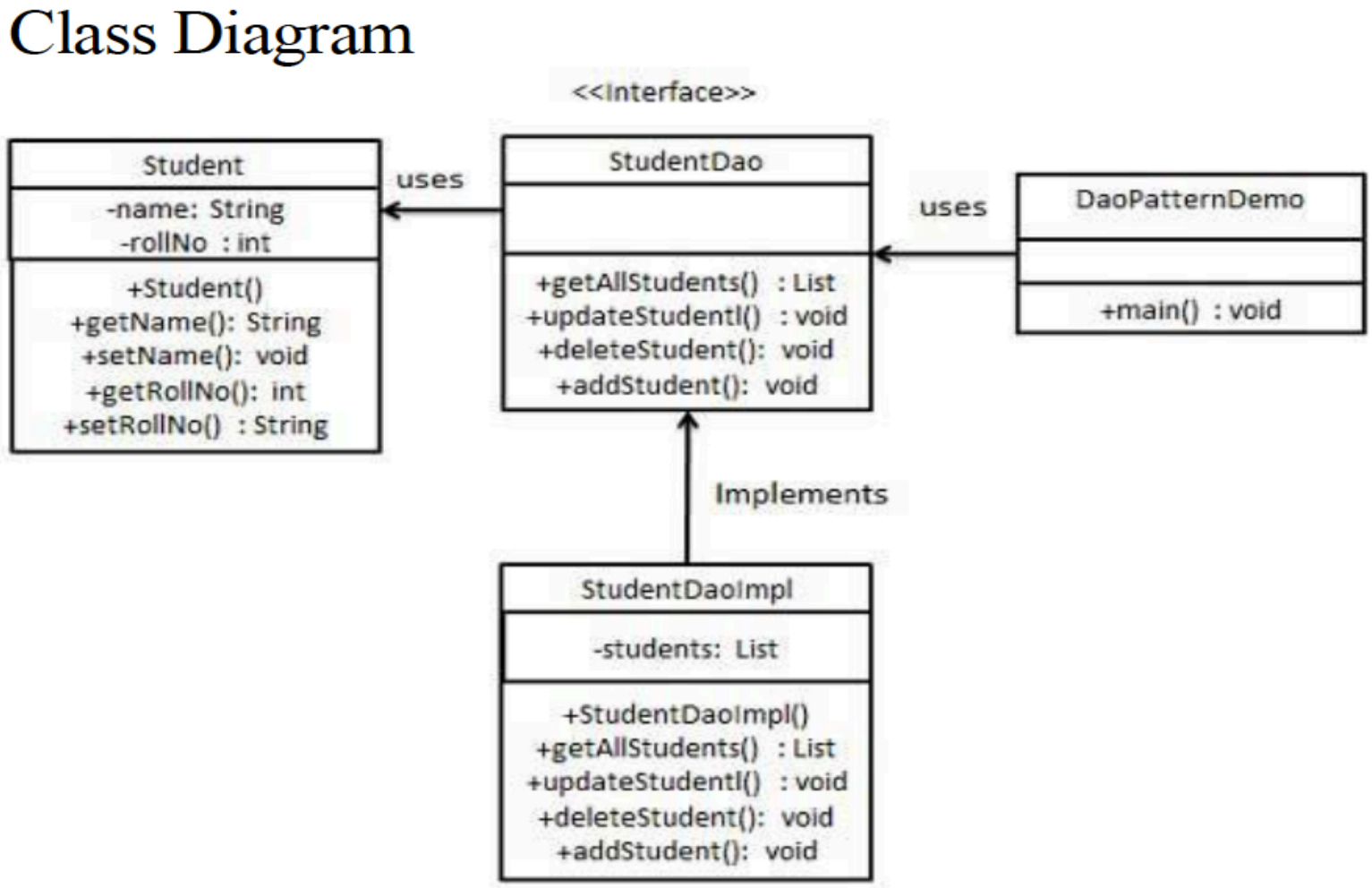
```
public class SingletonPatternDemo {
   public static void main(String[] args) {

      //illegal construct
      //Compile Time Error: The constructor SingleObject() is not
visible
      //SingleObject object = new SingleObject();

      //Get the only object available
      SingleObject object = SingleObject.getInstance();

      //show the message
      object.showMessage();
   }
}
```

Design Pattern: ***DAO***



DAO (Exemple)

Student.java

```
public class Student {
   private String name;
   private int rollNo;

   Student(String name, int rollNo){
      this.name = name;
      this.rollNo = rollNo;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public int getRollNo() {
      return rollNo;
   }

   public void setRollNo(int rollNo) {
      this.rollNo = rollNo;
   }
}
```

StudentDao.java

```
import java.util.List;

public interface StudentDao {
   public List<Student> getAllStudents();
   public Student getStudent(int rollNo);
   public void updateStudent(Student student);
   public void deleteStudent(Student student);
}
```

Exemple: suite 1

StudentDaoImpl.java

```
import java.util.ArrayList;
import java.util.List;

public class StudentDaoImpl implements StudentDao {

   //list is working as a database
   List<Student> students;

   public StudentDaoImpl(){
      students = new ArrayList<Student>();
      Student student1 = new Student("Robert",0);
      Student student2 = new Student("John",1);
      students.add(student1);
      students.add(student2);
   }
   @Override
   public void deleteStudent(Student student) {
      students.remove(student.getRollNo());
      System.out.println("Student: Roll No " + student.getRollNo()
         +", deleted from database");
   }

   //retrive list of students from the database
   @Override
   public List<Student> getAllStudents() {
      return students;
   }

   @Override
   public Student getStudent(int rollNo) {
      return students.get(rollNo);
   }

   @Override
   public void updateStudent(Student student) {
      students.get(student.getRollNo()).setName(student.getName());
      System.out.println("Student: Roll No " + student.getRollNo()
         +", updated in the database");
   }
}
```

Exemple: suite 2

```
public class DaoPatternDemo {
   public static void main(String[] args) {
      StudentDao studentDao = new StudentDaoImpl();

      //print all students
      for (Student student : studentDao.getAllStudents()) {
         System.out.println("Student: [RollNo : "
            +student.getRollNo()+", Name : "+student.getName()+" ]");
      }


      //update student
      Student student =studentDao.getAllStudents().get(0);
      student.setName("Michael");
      studentDao.updateStudent(student);

      //get the student
      studentDao.getStudent(0);
      System.out.println("Student: [RollNo : "
         +student.getRollNo()+", Name : "+student.getName()+" ]");
   }
}
```

JNDI

JNDI et BD dans le Web

- Idée : au lieu de tout programmer, on utilise un nom pour obtenir une connexion à partir d'une source de données.
- Avantage :

1. On change la source sans modifier le code java.
2. disponible pour les application pour des BD en réseaux.

- Code java :

```
Context context = new InitialContext();
DataSource dataSource = (DataSource)context.lookup
("java:comp/env/jdbc/myDatabase");
Connection connection = dataSource.getConnection();
```

JNDI et BD dans le Web

- Web.xml:

```
<resource-ref>
   <description>DB Connection</description>
   <res-ref-name>jdbc/myDatabase</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Container</res-auth>
</resource-ref>
```

- context.xml

```
<Context>
<Resource
name="jdbc/myDatabase"
driverClassName="org.apache.derby.jdbc.ClientDriver"
url="jdbc:derby:myDatabase"
type="javax.sql.DataSource"
username="someuser"
password="somepassword"
auth="Container"
maxActive="8" />
</Context>
```

Exercice(Gestion cours)

Et, si nous suivons la logique des relations entre nos tables, nos classes sont liées suivant le diagramme de classes correspondant à la figure suivante.

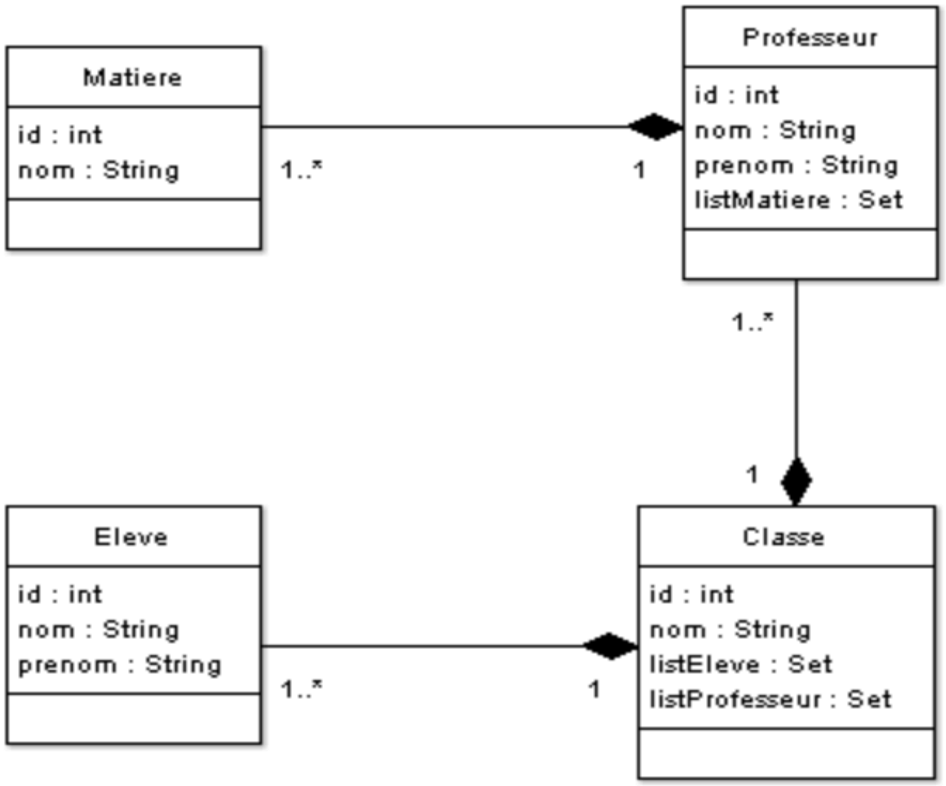


Diagramme de classe de notre BDD

Grâce à ce diagramme, nous voyons les liens entre les objets : une classe est composée de plusieurs élèves et de plusieurs professeurs, et un professeur peut exercer plusieurs matières. Les tables de jointures de la base sont symbolisées par la composition dans nos objets.

Lien

voir: https://openclassrooms.com/courses/apprenez-a-programmer-en-java/lier-ses-tables-avec-des-objets-java-le-pattern-dao

Compléments de cours

Exemple connexion Oracle

- Création de la connexion à la base

```
Connection con;
// chargement du driver Oracle
DriverManager.registerDriver(
    new oracle.jdbc.driver.OracleDriver());
// création de la connexion
con = DriverManager.getConnection(
    ''jdbc :oracle :thin :@ladybird :1521
    :test, ''etudiant'', ''mdpetud'');
```

Exemple Statement

```
Statement req;
ResultSet res;
String libelle;
int code;
 req = con.createStatement();

 res = req.executeQuery(
    ''select codcat, libellecat from categorie'');

 while(res.next()) {
     code = getInt(1);
     libelle = getString(2);
     System.out.println(
             '' produit : ''+code +'','' + libelle);
 }
 req.close();
```

Exemple Update

```
Statement req;
int nb;

req = con.createStatement();

nb = req.executeUpdate(''
 insert into categories values (5, 'cereales')'');

System.out.println(
     '' nombre de lignes modifiées : ''+nb);

req.close();
```

Exemple PrepareStatement

```
PreparedStatement req;
ResultSet res;
String nom;
 req = con.prepareStatement(''select codprod, nomprod
    from categorie c, produit p where c.codcat=p.codcat
    and libellecat = ?'');

 req.setString(1, ''cereales'');

 res = req.executeQuery();

 while(res.next()) {
      code = getInt(1);
      libelle = getString(2);
       System.out.println(
              '' produit : ''+code +'',''+ libelle); }
 req.close();
```

Exemple Update

```
PreparedStatement req;
int nb;

req = con.prepareStatement(
        ''insert into categories values (?,?)'');

req.setInt(1, 12);
req.setString(2, ''fruits'');
nb = req.executeUpdate();

req.setInt(1, 13);
req.setString(2, ''légumes'');
nb = req.executeUpdate();

req.close();
```