Projet Java : Base de données relationnelle

Jianying Liu
Qi Wang

Plan de présentation

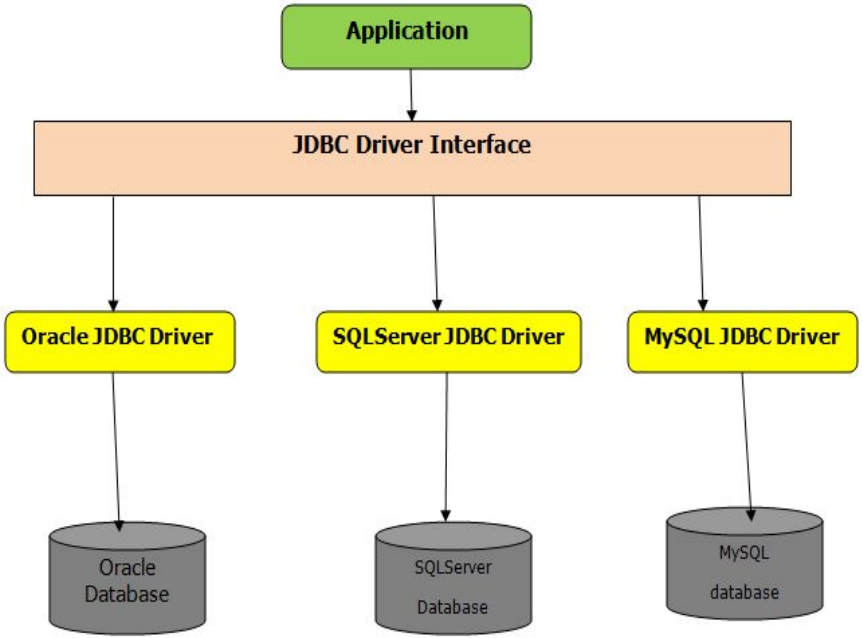
- Objectifs
- Présentation du JDBC
- Aperçu du projet
- Pratique des notions de base

Objectifs

- Proposition d’un programme Java utilisant une base de données relationnelles
- Rappel de notions de bases en Java

Présentation du JDBC

Fonctionnement de JDBC



- JDBC: Java Database Connectivity
- Elle désigne une API pour permettre un accès aux bases de données avec Java.
- Normalement, il s'agit d'une BD relationnelle.

Pour importer le JDBC Driver >>>

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

Installation : MySQL & MySQL Driver

1. Télécharger MySQL et créer une BD
   - SQL : https://dev.mysql.com/downloads/
   - Login : `mysql -u root -p;`
   - Créer la BD: `create database paristournage default character set utf8 collate utf8_bin;`

2. Télécharger SQL Driver et ajouter dans le ClassPath sous Eclipse
   - MySQL Driver : https://dev.mysql.com/downloads/connector/j/
   - Pour ajouter : Project > Properties > Java Build Path > Libraries > ClassPath > Add External JARS

Classes principales de JDBC

Librairie importée: `import java.sql.*;`

1. Charger le Driver

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

2. Établir la connexion

```
Connection connection = DriverManager.getConnection(DB_URL, USER, PASSWORD);
```

3.a Créer le Statement

```
Statement statement = connection.createStatement();
String sql = "SELECT FROM Cinema WHERE
titre=\"NomFilm\"";
// executeUpdate : modifier, ajouter, supprimer
statement.executeUpdate(sql);
// executeQuery : chercher
 resultSet = statement.executeQuery(sql);
```

3.b Créer le PreparedStatement

```
sql = "INSERT INTO Cinema (titre, anneeTournage, type,
realisateur, producteur) VALUES (?, ?, ?, ?, ?)";
preparedStatement = connection.preparedStatement(sql);
// 1 -> premier "?", ...
preparedStatement.setObject(1, arg1);
preparedStatement.setString(2, arg2);
// Opération : executeUpdate|executeQuery
preparedStatement.executeUpdate();
resultSet = preparedStatement.executeQuery();
```

Classes principales de JDBC

4. Afficher les résultats

```
// ResultSet
 resultSet.next()  //pour parcourir
 resultSet.getString(columnName)
 resultSet.getObject(String columnName)
 resultSet.getObject( int columnIndex)
 ResultSetMetaData metaData = resultSet.getMetaData();
 int nbColonne = metaData.getColumnCount();
 metaData.getColumnLabel(indice)
```

Connexion à la BD : exemple concret

Librairies importés :

```
import java.io.FileReader;
import java.io.IOException;
import java.net.URL;
import java.sql.*;
import java.util.Properties;
```

! N'oubliez pas à les fermer après l'utilisation ! (page suivante)

```
try {
      // importer le driver
      Class.forName("com.mysql.cj.jdbc.Driver");
      // obtenir l'objet de connexion
      Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/pa
ristournage","root","ljy");
      // créer le statement de la connexion
      Statement statement = connection.createStatement();
      // exécuter la requête SQL et obtenir les résultats
      ResultSet resultSet = statement.executeQuery("SELECT *
FROM LIEUX");
      while (resultSet.next()) {
System.out.println(resultSet.getString("code_postal"));
      }
}
```

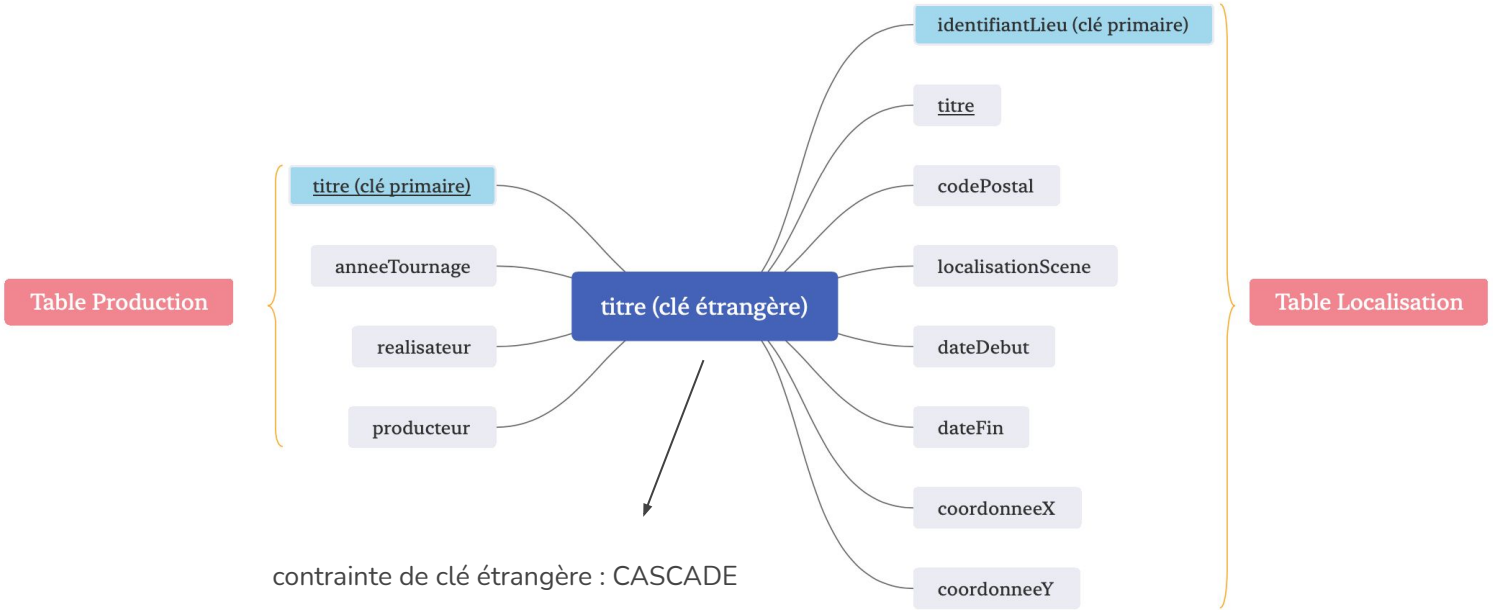
```
catch (SQLException e) {
      e.printStackTrace();
} catch (ClassNotFoundException e) {
      e.printStackTrace();
} finally {
      if (resultSet != null) {
            try {
                  resultSet.close();
            } catch (SQLException e) {
                  e.printStackTrace();
            }
      }
      if (statement != null) {
            try {
                  statement.close();
            } catch (SQLException a) {
                  a.printStackTrace();
            }
      }
      if (connection != null) {
            try {
                  connection.close();
            } catch (SQLException a) {
                  a.printStackTrace();
            }
      }
}
```

Aperçu du projet

Données

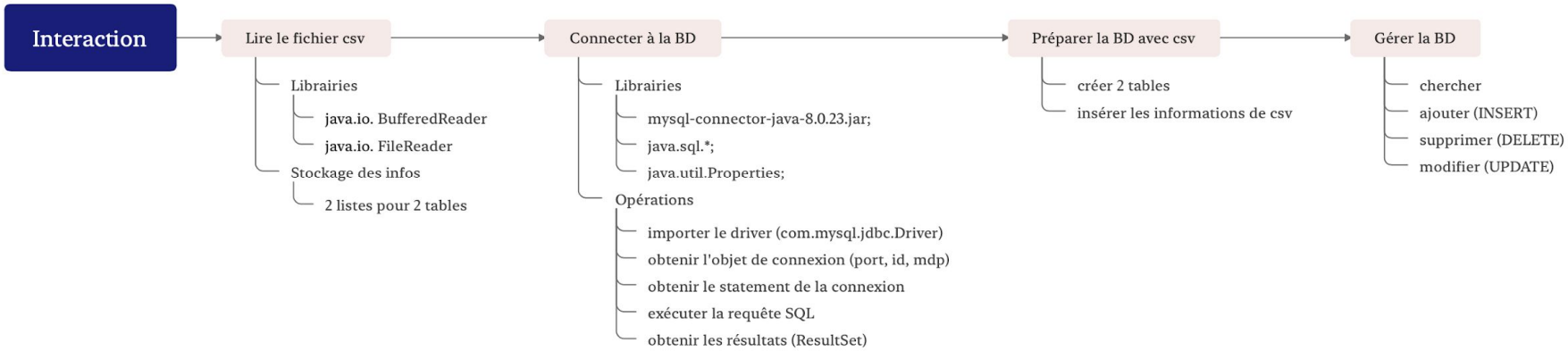
- Source : **Paris Data**
  - Un ensemble de jeu de données ***open source***.
- Sujet : Lieux de tournage à Paris (fichier csv)
  - https://opendata.paris.fr/explore/dataset/lieux-de-tournage-a-paris/information/
- Contenu :
  - Lieux de tournage de scène en extérieur à Paris ***depuis 2016***, les tournages désignent les ***longs métrages***, les ***séries*** et les ***téléfilms*****,** réalisés à l'extérieur.

Données : construction d'une BD relationnelle

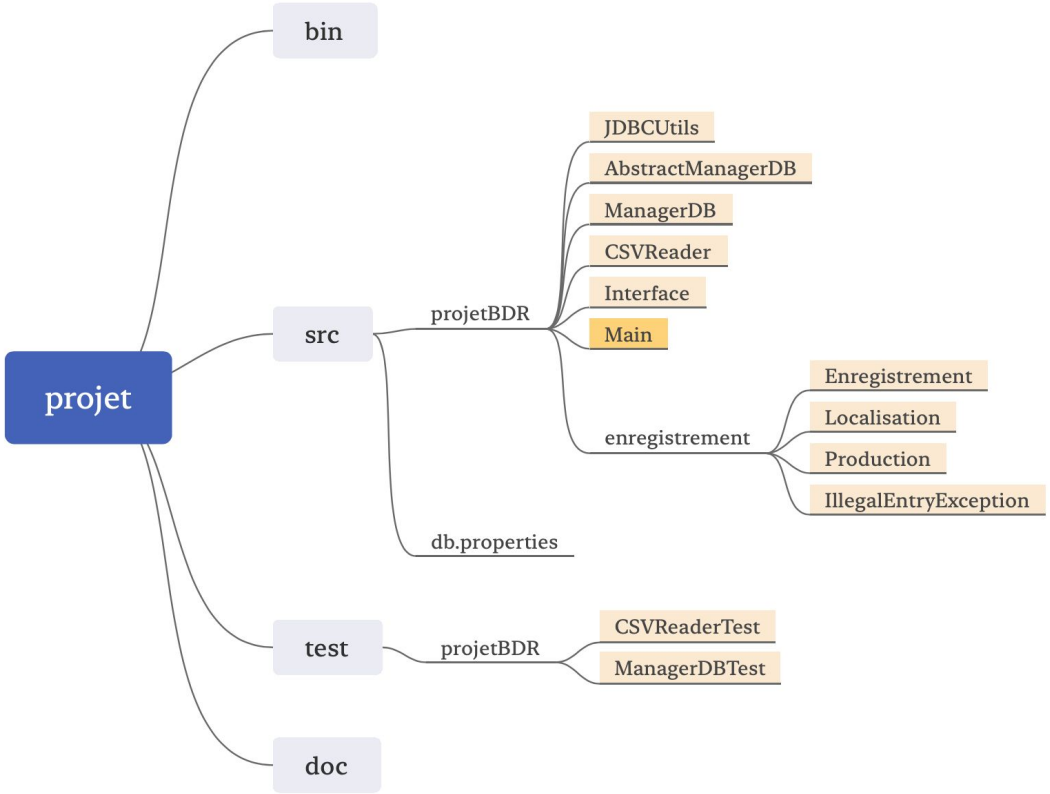


contrainte de clé étrangère : CASCADE

Interaction entre BD et Java



Hiérarchie du projet



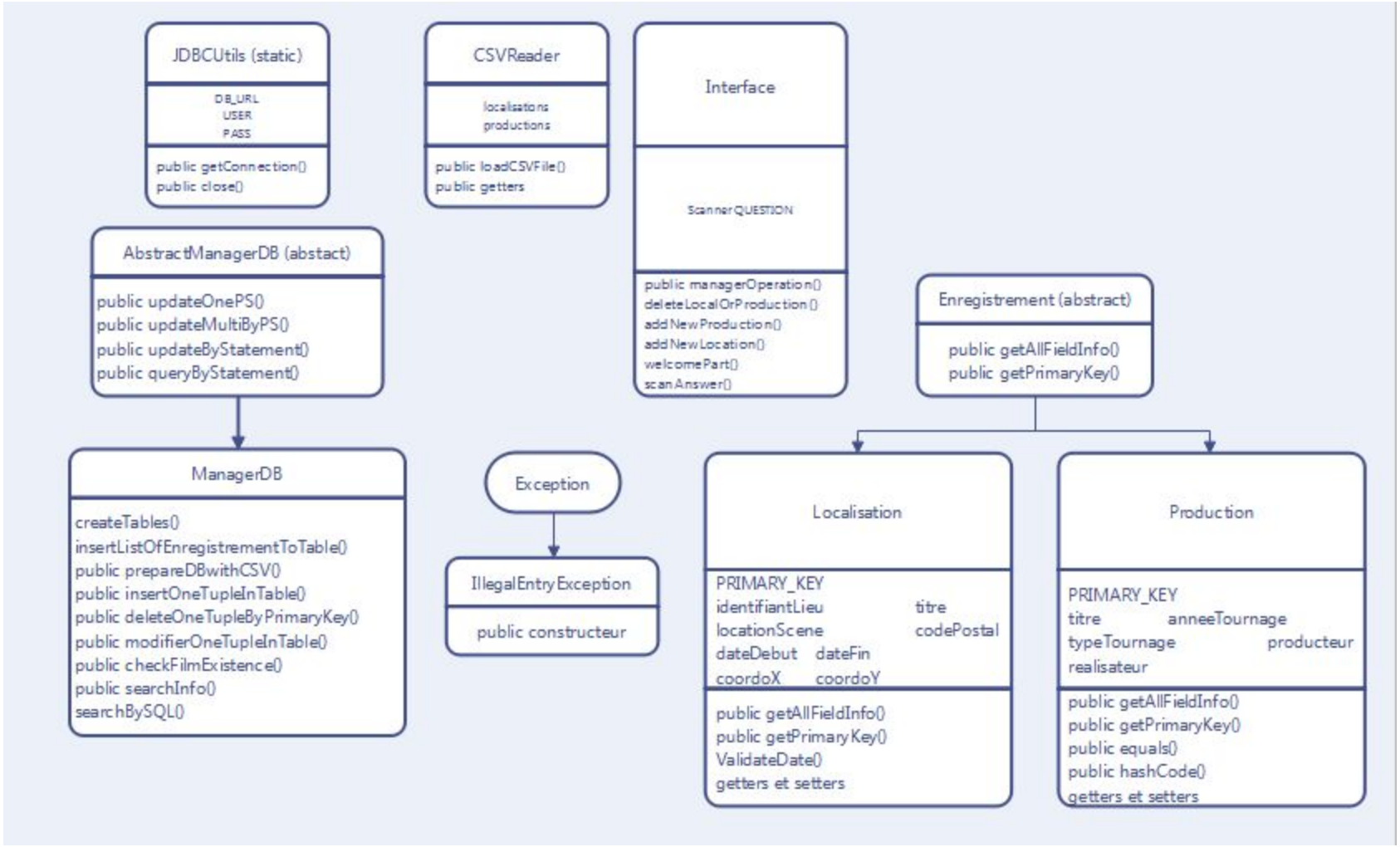
Pratique des notions de base

Classe, méthode, instance

- **Méthode**: correspond à un comportement donné commun à tous les objets de la classe
- **Classe**: décrit le modèle sur lequel les objets de cette classe vont être créés
- **Instance** : l'implémentation d'une classe

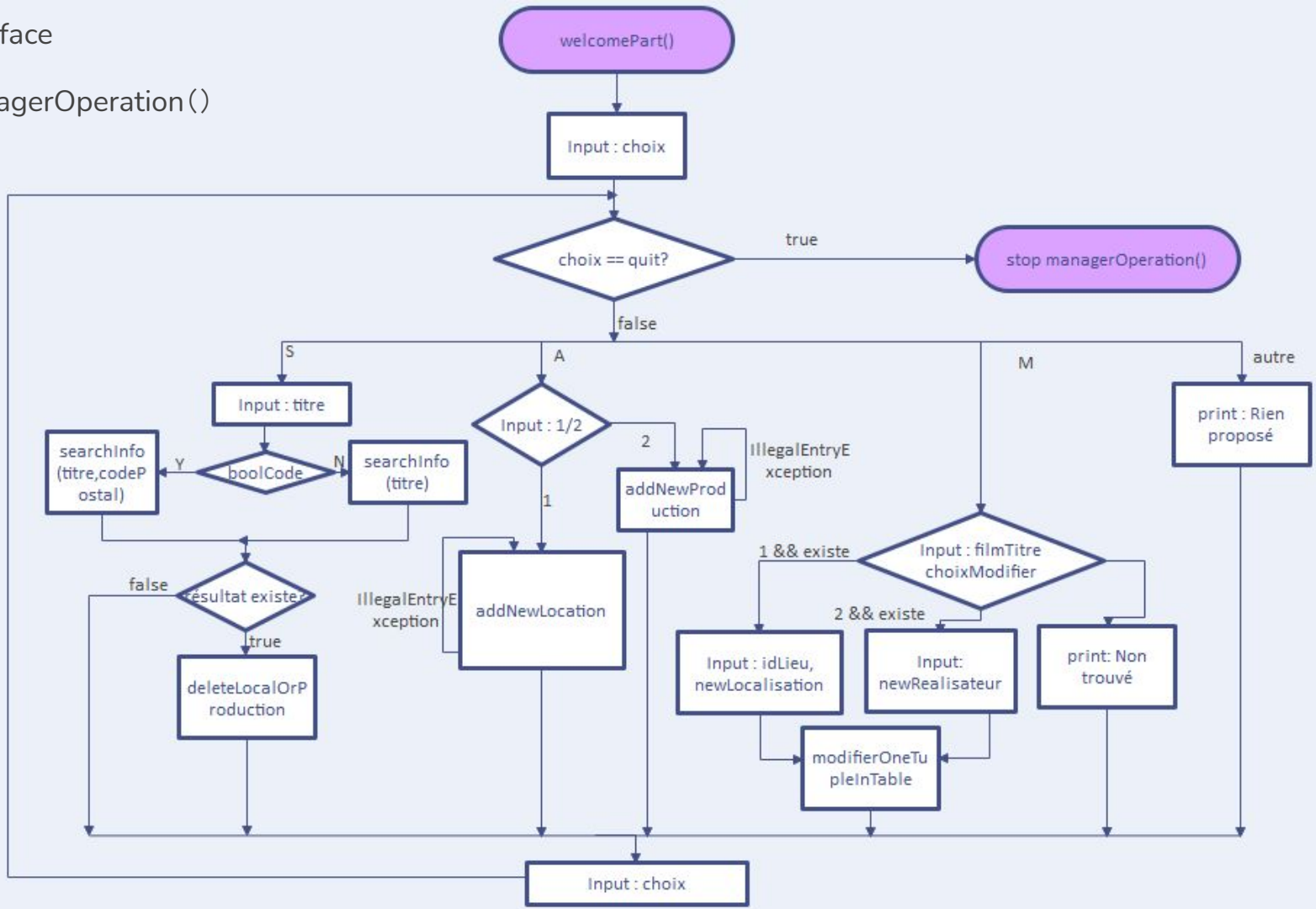
Encapsulation

- L’utilisation de l’encapsulation permet d'éviter de modifier directement les données d'un objet
- private vs public


JDBCUtils (static)
DB_URL
USER
PASS
public getConnection()
public close()
CSVReader
localisations
productions
public loadCSVFile()
public getters
Interface
ScannerQUESTION
public managerOperation()
deleteLocalOrProduction()
addNewProduction()
addNewLocation()
welcomePart()
scanAnswer()
AbstractManagerDB (abstact)
public updateOnePS()
public updateMultiByPS()
public updateByStatement()
public queryByStatement()
Enregistrement (abstract)
public getAllFieldInfo()
public getPrimaryKey()
ManagerDB
createTables()
insertListOfEnregistrementToTable()
public prepareDBwithCSV()
public insertOneTupleInTable()
public deleteOneTupleByPrimaryKey()
public modifierOneTupleInTable()
public checkFilmExistence()
public searchInfo()
searchBySQL()
Exception
IllegalEntryException
public constructeur
Localisation
PRIMARY_KEY
identifiantLieu titre
locationScene codePostal
dateDebut dateFin
coordoX coordoY
public getAllFieldInfo()
public getPrimaryKey()
ValidateDate()
getters et setters
Production
PRIMARY_KEY
titre anneeTournage
typeTournage producteur
realisateur
public getAllFieldInfo()
public getPrimaryKey()
public equals()
public hashCode()
getters et setters

Interface

managerOperation()



Gestion des exceptions

- Détecter et traiter les Exceptions : **try, catch, finally**
- Lever les Exceptions : **throw, throws**

Gestion des exceptions

1) **try, catch et finally**

- try : code qui est susceptible de produire des erreurs ou des exceptions
- catch : capturer les exceptions et les traiter
- finally : facultatif, toujours exécuté qu'une exception soit levée ou non

```
try {
    opération_risquée1;
    opération_risquée2;
} catch(ExceptionAppelante e){
    traitements
} catch(ExcpetionPossible e){
    traitements
} finally {
    traitement_toujours_exécuté;
}
```

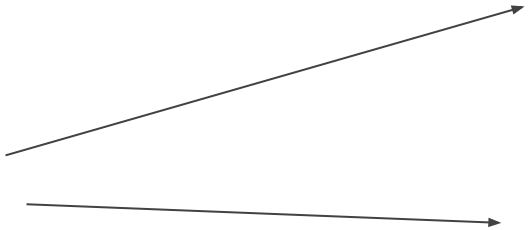
Exemple 1

Gestion des exceptions

```
try (BufferedReader br = new BufferedReader(new
FileReader(filePath))) {

      while ((line = br.readLine()) != null) {
     // Traitements Omis
            }

} catch (IOException e) {
     e.printStackTrace();
}
```



IOException :
la classe de base des exceptions levées lors de l'accès aux informations **à l'aide de flux, de fichiers et de répertoires**

méthode printStackTrace() :
pour faire **afficher les informations plus détaillées** à propos de l'exception

Exemple 2 : Class CSVReader

Gestion des exceptions

```
public List<Map<String, Object>> queryByStatement(String sql){
    Connection connection = null;
    Statement statement = null;
    ResultSet resultSet = null;
    List<Map<String, Object>> resultList = new ArrayList<>();
    try {
        connection = JDBCUtils.getConnection();
        statement = connection.createStatement();
        resultSet = statement.executeQuery(sql);
        ResultSetMetaData metaData = resultSet.getMetaData();
        int nbColonne = metaData.getColumnCount();
        while (resultSet.next()){
            Map<String,Object> row = new HashMap<>();
            for (int i=1; i <= nbColonne; i++){
                row.put(metaData.getColumnLabel(i), resultSet.getObject(i));
            }
            resultList.add(row);
        }
    } catch (SQLException e) {
        System.out.println("La requête a échouée ! Exception: ");
        System.out.println(e.getMessage());
    } finally {
        JDBCUtils.close(resultSet,statement);
        JDBCUtils.close(connection);
    }
    return resultList;
}
```

AbstractManagerDB

Gestion des exceptions

2) throw et throws

- throw : utilisé dans la méthode, pour lever une exception.

- throws : utilisé dans la signature d'une méthode pour déclarer une méthode dont on peut anticiper l'échec.

Exemples

```
private void setAnneeTournage(int anneeTournage) throws IllegalEntryException {
    if ( anneeTournage > 2050 || anneeTournage < 1895 ){
        throw new IllegalEntryException("L'année de tournage illégale !");
    }
    this.anneeTournage = anneeTournage;
}

public Production(String titre, int anneeTournage, String typeTournage,
        String producteur, String realisateur) throws IllegalEntryException {
    this.titre = titre;
    setAnneeTournage(anneeTournage);
    this.typeTournage = typeTournage;
    this.producteur = producteur;
    this.realisateur = realisateur;
}
```

AddNewProduction:

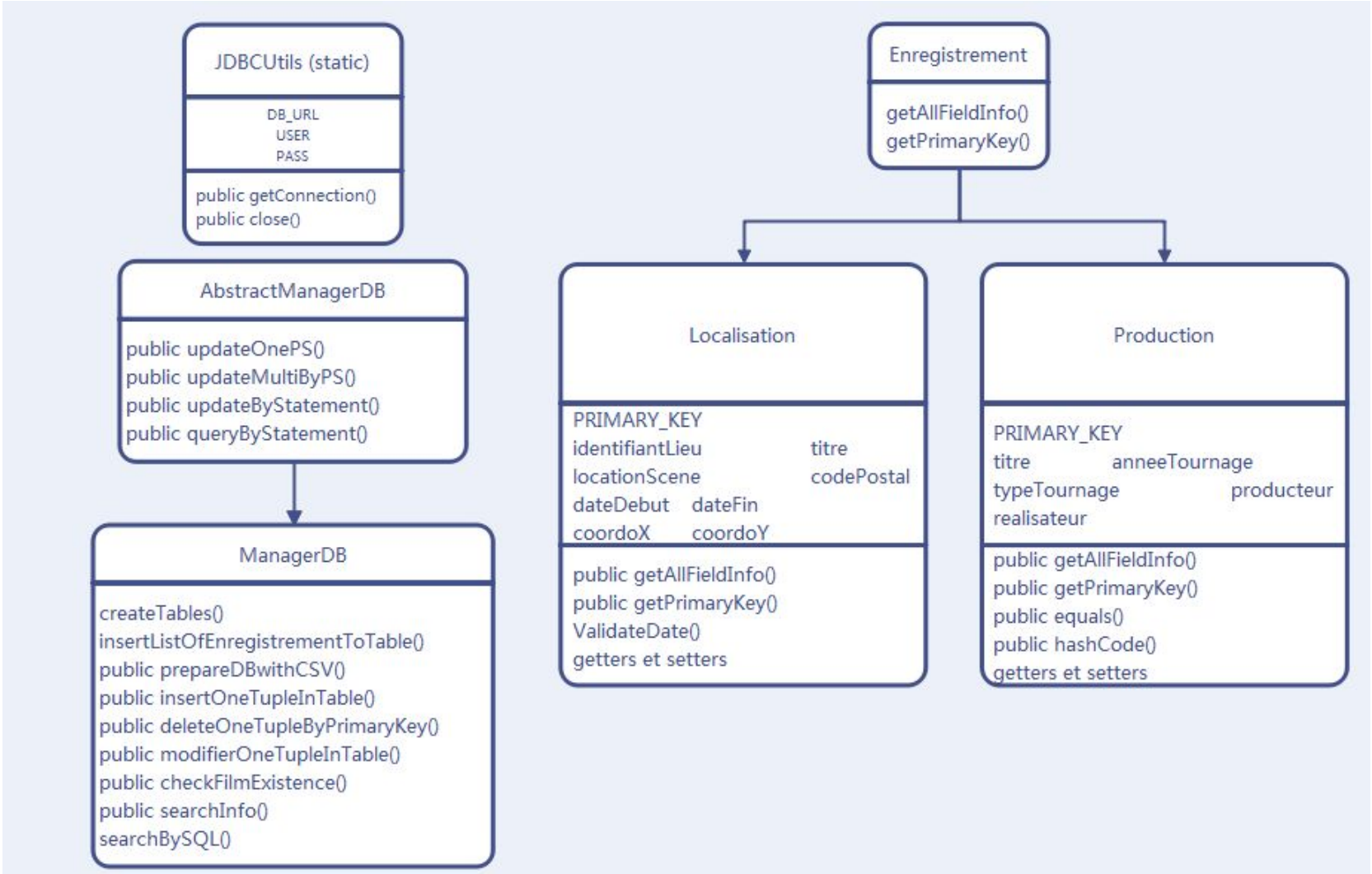
```
} catch (IllegalEntryException e){
    System.out.println("Entrée illégale:" + e.getMessage());
    System.out.println("Recommencez : ");
    addNewProduction(managerDB);
}
```

Héritage

- Il permet de définir une classe par extension des caractéristiques d'une autre classe et de définir une hiérarchie de classes. (`ClassFils extends ClassPere`)
- La classe Objet est la racine de l'héritage des classes en java.

```java
public abstract class Enregistrement {
    // déclaration des méthodes abstraires, à redéfinir dans ses classes enfant
    public abstract Object[] getAllFieldInfo();
    public abstract String getPrimaryKey();
}
```

```java
public class Localisation extends Enregistrement{
    private final String PRIMARY_KEY = "identifiantLieu";
    private String identifiantLieu;
    private String titre;
    private String locationScene;
    private int codePostal;
    private String dateDebut;
    private String dateFin;
    private double coordoX;
    private double coordoY;
```


JDBCUtils (static)
DB_URL
USER
PASS
public getConnection()
public close()
AbstractManagerDB
public updateOnePS()
public updateMultiByPS()
public updateByStatement()
public queryByStatement()
ManagerDB
createTables()
insertListOfEnregistrementToTable()
public prepareDBwithCSV()
public insertOneTupleInTable()
public deleteOneTupleByPrimaryKey()
public modifierOneTupleInTable()
public checkFilmExistence()
public searchInfo()
searchBySQL()
Enregistrement
getAllFieldInfo()
getPrimaryKey()
Localisation
PRIMARY_KEY
identifiantLieu titre
locationScene codePostal
dateDebut dateFin
coordoX coordoY
public getAllFieldInfo()
public getPrimaryKey()
ValidateDate()
getters et setters
Production
PRIMARY_KEY
titre anneeTournage
typeTournage producteur
realisateur
public getAllFieldInfo()
public getPrimaryKey()
public equals()
public hashCode()
getters et setters

Polymorphisme

Désigne le fait que la même méthode peut avoir un comportement différent selon les situations.
La gestion du polymorphisme est assurée par la machine virtuelle dynamiquement à l'exécution.

Surcharge (Overload)

- La surcharge d'une méthode permet de définir plusieurs fois une même méthode avec des paramètres différents dans la même classe.
- Le compilateur choisi la méthode qui doit être appelée en fonction du nombre et du type des arguments.

```java
public boolean searchInfo(String titre){
    String sql = "SELECT * FROM Cinema, LieuxTournage WHERE Cinema.titre = LieuxTournage.titre AND LieuxTournage.titre = \"" + titre + "\";";
    return searchBySQL(sql, titre);
}

public boolean searchInfo(String titre, String codePostal){
    String sql = "SELECT * FROM Cinema, LieuxTournage WHERE cinema.titre = LieuxTournage.titre AND LieuxTournage.titre = \"" + titre + "\" AND codePostal =" + codePostal + ";";
    return searchBySQL(sql, titre);
}
```

`searchInfo(String titre)` & `searchInfo(String titre, String codePostal)`

Redéfinition (Override)

- En redéfinissant une méthode dans une sous-classe, on peut spécialiser le comportement d'une méthode.
- La redéfinition d'une méthode héritée conserve la déclaration de la méthode parente (type et nombre de paramètres, la valeur de retour et les exceptions propagées doivent être identiques).

Méthode Parente →

```java
public abstract class Enregistrement {
    // déclaration des méthodes abstraires, à redéfinir dans ses classes enfant
    public abstract Object[] getAllFieldInfo();
    public abstract String getPrimaryKey();
}
```

Méthode héritée →

```java
/**
 * Obtenir et renvoyer un objet contenant les informations de tous les champs
 * redéfinition de getAllFieldInfo() dans classe parent Enregistrement
 * @return champsInfo ordonnées selon l'ordre dans la BD
 */
@Override
public Object[] getAllFieldInfo() {
    Object[] champsInfo = {identifiantLieu,titre,locationScene,codePostal,dateDebut,dateFin,coordoX,coordoY};
    return champsInfo;
}

/**
 * renvoie le nom de la clé primaire de localisation
 * @return
 */
@Override
public String getPrimaryKey() {
    return PRIMARY_KEY;
}
```

Polymorphisme

```
 */
public boolean insertOneTupleInTable(Enregistrement enregistrement){
    String key = enregistrement.getPrimaryKey();
    String sql;
    if (key.equals("identifiantLieu")){
        sql = "INSERT INTO LieuxTournage ( identifiantLieu, titre, localisationScene, codePostal, dateDebut, date
    } else {
        sql = "INSERT INTO Cinema ( titre, anneeTournage, type, realisateur, producteur) VALUES (?, ?, ?, ?, ?)";
    }
    Object[] args = enregistrement.getAllFieldInfo();
    return updateOnePS(sql, args);
}
```

Tests Unitaires

- Le test unitaire est une procédure permettant de vérifier le bon fonctionnement d'une portion d'un programme.

```
@Test
void testDeleteOneTupleByPrimaryKey() throws SQLException {
    //GIVEN
    Production prod = new Production("Daft Punk Unchained", 2015, "Film" ,"Patrice Gellé, Jean-Louis Blot", "Hervé Martin-Delpierre");
    String nomCle = "titre";
    String valeurCle = "Daft Punk Unchained";
    String nomTable = "Cinema";

    //WHEN
    mDB.insertOneTupleInTable(prod);
    mDB.deleteOneTupleByPrimaryKey(nomCle, valeurCle, nomTable);

    //THEN
    assertFalse(mDB.checkFilmExistence("Daft Punk Unchained"));
}
```

Merci !