Introduction à la persistance de données avec JPA

Java EE

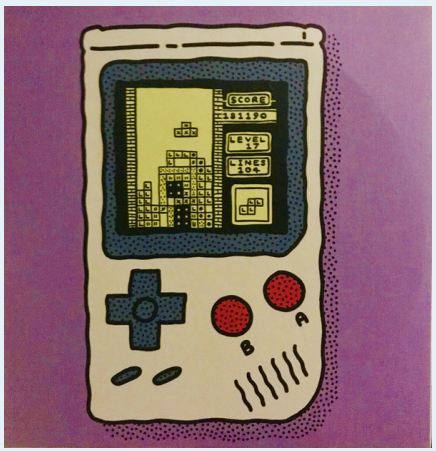
Nicolas Delanoue

Université d'Angers - Polytech Angers

Définition

En programmation, la gestion de la persistance des données réfère au mécanisme responsable de la sauvegarde et de la restauration des données.

Exemple



Pas de persistance sur Game boy.

Exemple

Persistance sur Mario. . . .

Base de données versus fichier

Pourquoi stocker des données dans une base de données alors qu'on peut les stocker dans un fichier ?

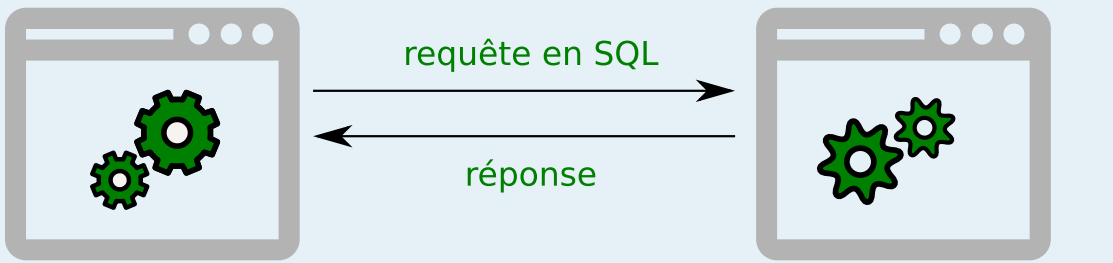
Base de données versus fichier

Pourquoi stocker des données dans une base de données alors qu'on peut les stocker dans un fichier ?

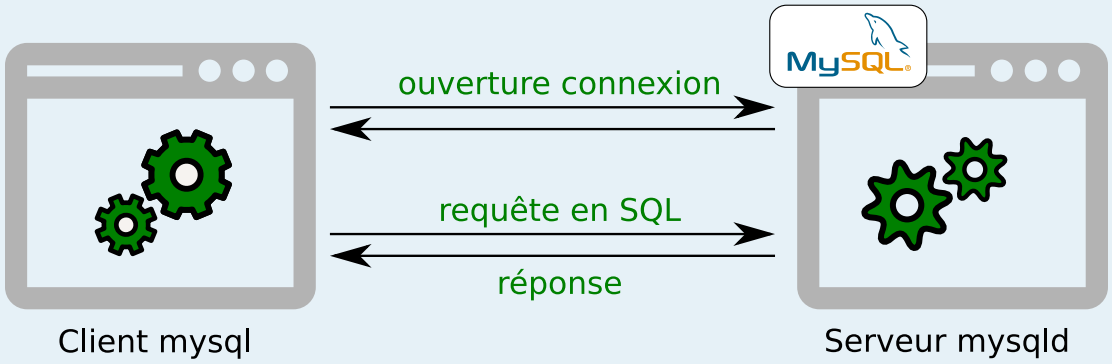
Mécanismes

- donnée : rechercher, ajouter ou supprimer des enregistrements (Create, Read, Update, Delete abrégé CRUD),
- sécurité : protection via comptes, mots de passe, ...
- concurrence : transactions, ...

Architecture standard



Exemple



Ouverture de connexion

```
nico@PC:~$ mysql -u test -h localhost -P 3306 -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ;.
Your MySQL connection id is 19
Server version: 8.0.23-0ubuntu0.20.04.1 (Ubuntu)
Type 'help;' or '\h' for help.
Type '\c' to clear the current input statement.
mysql>
```

Requêtes SQL :

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| UE_DB              |
| information_schema |
| mysql              |
+--------------------+
3 rows in set (0.01 sec)

mysql> USE UE_DB;
Database changed

mysql> SHOW TABLES;
+-----------------+
| Tables_in_UE_DB |
+-----------------+
| etats           |
+-----------------+
1 row in set (0.01 sec)
```

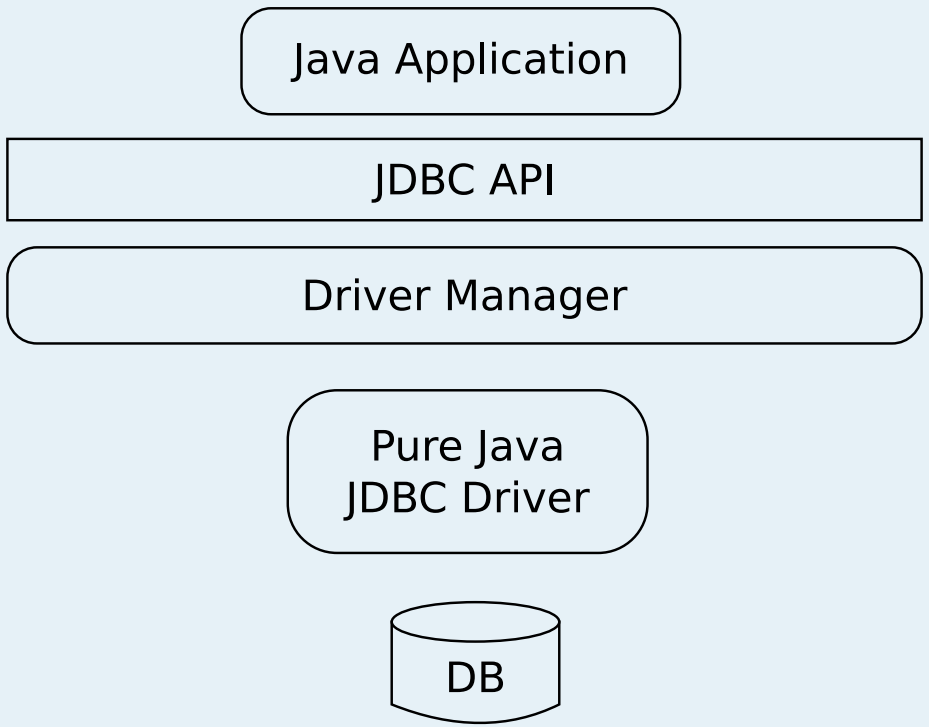
Requêtes SQL :

```
mysql> select COLUMN_NAME
 from INFORMATION_SCHEMA.COLUMNS
 where table_name='etats';
+-------------+
| COLUMN_NAME |
+-------------+
| id          |
| nom         |
| population  |
+-------------+
3 rows in set (0.00 sec)
```

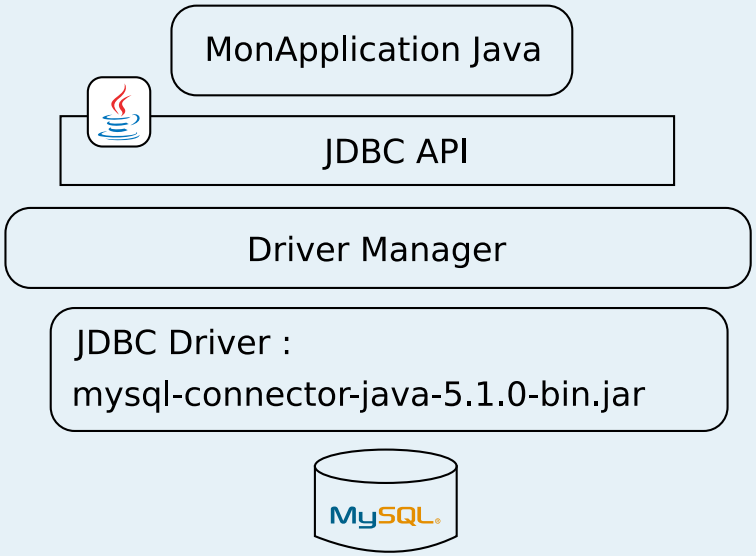
Java Database Connectivity

JDBC est interface de programmation permettant aux applications Java d'accéder par le biais d'une interface commune à des sources de données pour lesquelles il existe des pilotes JDBC.

Principe



Exemple



Le driver

- implémente JDBC,
- communique avec la base de données,
- est souvent fourni par le propriétaire de la base de données.

Idée similaire à la notion de driver pour un système d'exploitation.

Choix de la base de données et du driver depuis le code java

On utilise une url de la forme :

`jdbc:driver:adresse`

Exemples

- `jdbc:mysql://www.sante.fr/vaccins`
- `jdbc:oracle:userDB@www.sante.fr:vaccins`
- `jdbc:mariadb://localhost:3306/vaccins`

Exemple d'utilisation

```
1  package jdbcNico;
2  import java.sql.*;
3  public class test_jdbc {
4   public static void main(String[] args) throws Exception {
5   //Class.forName("com.mysql.jdbc.Driver");  //chargement Driver
6   String url = "jdbc:mysql://localhost:3306/UE_DB";
7   Connection con=DriverManager.getConnection(url,"login","pass");
8   String select = "SELECT * FROM etats";
9   PreparedStatement ps = con.prepareStatement(select);
10  ResultSet rs=ps.executeQuery();
11  while(rs.next())
12   {
13   String chaine = rs.getString("nom") + rs.getString("pop");
14   System.out.println( chaine );
15    }
16  rs.close();
17  ps.close();
18  con.close();
19  }
20 }
```

Exemple

Imaginons une application qui offre des possibilités CRUD sur différentes tables via différents évènements.

Remarque

Avec uniquement JDBC, le code est difficile à maintenir.

Fonctionnalités de JDBC

- Curseur multi-directionnel,
- Requête précompilée,
- Transaction,
- Traitement par lot,
- Pool de connexions,
- Points de reprise dans les transactions,
- Mode déconnecté (Row set),
- ...

Définition

Une requête précompilée est requête dont certaines valeurs doivent encore être renseignées.

Exemple

```
PreparedStatement ps;
String select = "SELECT * FROM etats where population > ?";
PreparedStatement ps = con.prepareStatement(select);
ps.setInt(1 , 1000);
ResultSet rs = ps.executeQuery();
```

Avantages

- Exécution plus rapide de la requête,
- Sécurité supérieure à l'injection SQL.

Définition

Une transaction est un groupe de requêtes devant être exécutées de façon indivisible.

Intérêts

Consistance des données, annulation d'une transaction, ...

Mise en oeuvre

- début de transaction (`begin`),
- engagement de la transaction (`commit`),
- annulation de la transaction (`roolback`).

```
CREATE TABLE Etats(
    id  PRIMARY KEY NOT NULL,
    nom VARCHAR(30),
    population INT, CHECK(population >=0) );
```

Exemple

```
int deces = 12;
con.setAutoCommin(false);
try {
 Statement st = cx.createStatement();
 String r1 = "UPDATE continent SET population=population-" +
                              deces + "WHERE nom='Europe'" );
 st.executeUpdate(r1);
 String r2 = "UPDATE etat SET population=population-" +
                              deces + "WHERE nom='Laponie'" );
 st.executeUpdate(r2);
 con.commit();
catch (SQLException e)
{
 con.rollback();
}
```

Définition

Un Data Access Object est un patron de conception (sorte de bonne pratique de programmation) qui permet de séparer les opérations d'accès de leur mise en oeuvre.

Exemple

```
interface EtatDAO {
   public Etat recherche(String nom);
   public List<Etat> populationSuperieureA( int pop);
}
...
class EtatDAOImplJDBC implements EtatDAO {
   public Etat recherche(String nom) {
     // Code JDBC qui renvoie le bon état
   }
   public List<Etat> populationSuperieureA( int pop) {
     // Code JDBC qui renvoie les états vérifiant etat.population > pop
   }
}
```

Exemple d'utilisation :

```
EtatDAOImplJDBC etatDAO;
Etat f = etatDAO.recherche("france");
```

Définition - ORM

Un mapping objet-relationnel (ORM) est un type de programme informatique qui se place en interface entre un programme applicatif et une base de données relationnelle pour simuler une base de données orientée objet.

Principe de mise en correspondance

Programmation	**Base de données**
Classe	Table
Attribut d'une classe	Champ dans une table
Référence entre classes	Relation entre tables
Un objet	Un enregistrement dans une table

Une classe java

```
public class Etat {
 private long id;
 private String nom;
 private long population;
}
```

Une table sql

```
mysql> select COLUMN_NAME
 from INFORMATION_SCHEMA.COLUMNS
 where table_name='etat';
+-------------+
| COLUMN_NAME |
+-------------+
| id          |
| nom         |
| population  |
+-------------+
3 rows in set (0.00 sec)
```

Définition

Une classe pour laquelle il existe une correspondance en base de donnée est appelée une *entité*.

Définition

Java Persistence API (JPA) est une spécification d'interface de programmation pour la gestion de données relationnelles.

Possibilités offertes

Manipulation de données et de leurs relations directement en Java.

Différente implémentations de la spécification :

- Hibernate,
- TopLink,
- EclipseLink,
- Apache OpenJPA,
- ...

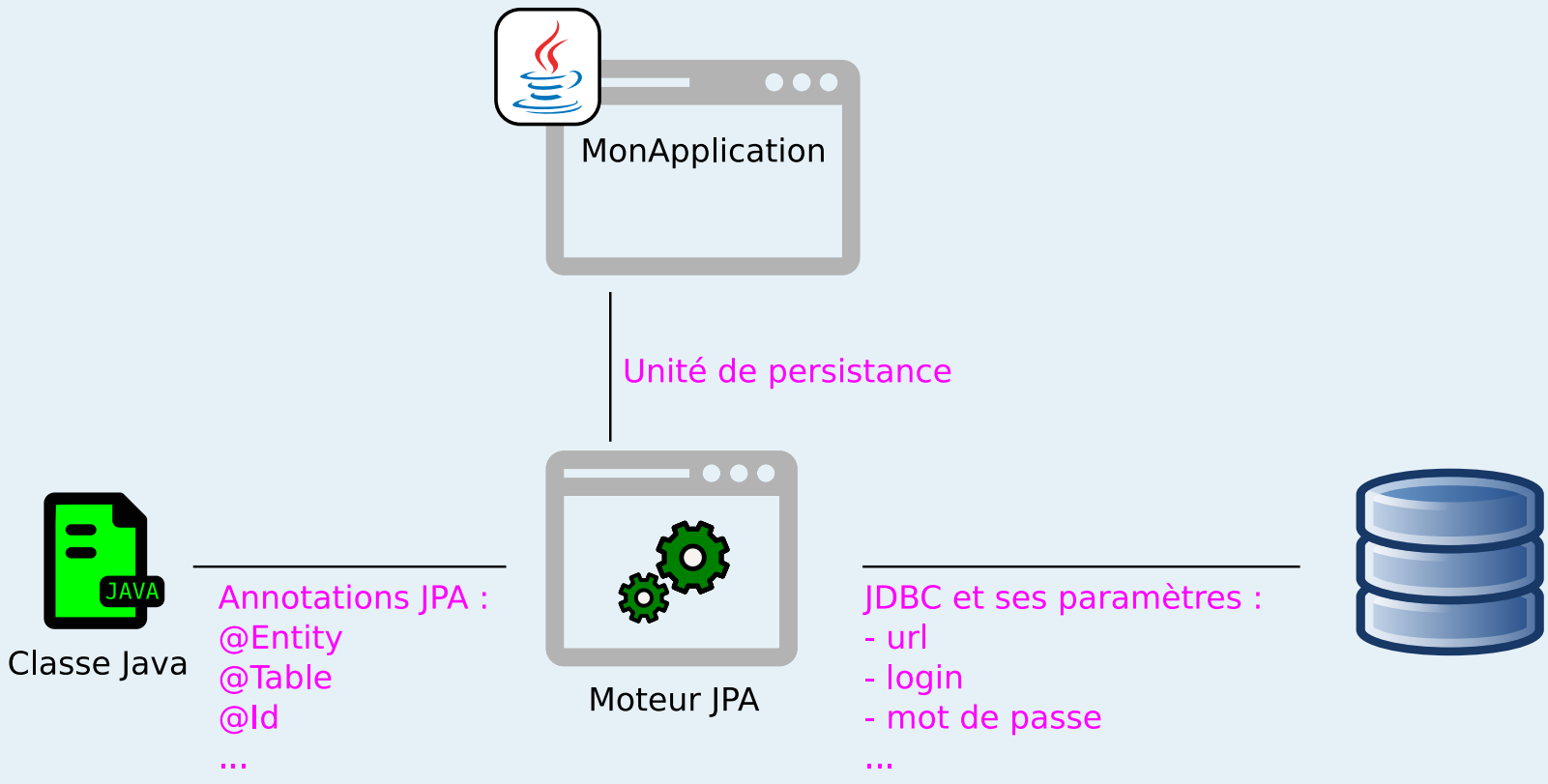
Concrètement

- Les correspondances se font via des annotations JPA dans les classes,
- une implémentation fournit une API pour manipuler les données,
- JPA s'appuie sur JDBC pour communiquer avec une base de données.

Exemple de classe annotée, i.e. une entité

```
1  package MonPackage;
2
3  import javax.persistence.Entity;
4  import javax.persistence.Table;
5  import javax.persistence.Id ;
6
7  @Entity @Table(name ="etats")
8  public class Etat {
9          @Id
10         private long id;
11         private String nom;
12         private long population;
13
14         public Etat() {
15                 super();
16                 this.id = 0;
17                 this.nom = "unknown";
18                 this.population = 0;
19         }
```

Architecture d'une application s'appuyant sur JPA



Remarque

L'unité de persistance est décrite dans le fichier de configuration : `src/META-INF/persistence.xml`

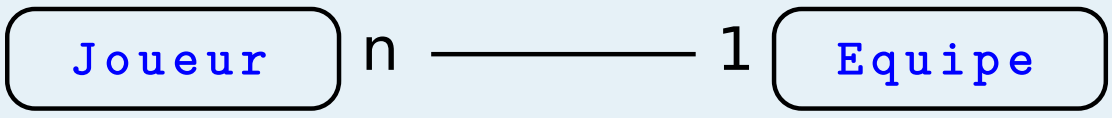
Persistence.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <persistence version="2.2"
3     xmlns="http://xmlns.jcp.org/xml/ns/persistence"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
6     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
7   <persistence-unit name="UniteUE">
8    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
9    <class>MonPackage.Etat</class>
10   <properties>
11    <property name="javax.persistence.jdbc.driver"
12              value="com.mysql.jdbc.Driver" />
13    <property name="javax.persistence.jdbc.url"
14              value="jdbc:mysql://localhost/UE_BD" />
15    <property name="javax.persistence.jdbc.user" value="login" />
16    <property name="javax.persistence.jdbc.password" value="pass" />
17   </properties>
18  </persistence-unit>
19 </persistence>
```

Gestion des cardinalités

Cardinalités	**Annotations JPA**
1-n	@OneToMany
n-1	@ManyToOne
1-1	@OneToOne
n-n	@ManyToMany

Exemple de relation n-1



Une classe java

```
@Entity
public class Joueur {
 @Id
 private long id;
 private String nom;
 private String prenom;
 @ManyToOne
 private Equipe equipe;
}
```

Une classe java

```
@Entity
public class Equipe {
 @Id
 private long id;
 private String ville;
 private long CouleurMaillot;
 @OneToMany
 List<Joueur> joueurs;
}
```

Définition

Un `EntityManager` permet de réaliser les opérations CRUD sur une base de données via une unité de persistance particulière.

Exemple

```
1  public static void main(String[] args)  throws Exception {
2    EntityManagerFactory emf ;
3    emf = Persistence.createEntityManagerFactory("UniteUE");
4    EntityManager entityManager = emf.createEntityManager();
5    Query q = entityManager.createQuery( "from Etat", Etat.class );
6    List<Etat> etats ;
7    etats = q.getResultList();
8    for (Etat etat : etats) {
9     System.out.println( etat.toString() );
10    }
11   // Ajout d'un nouvel enregistrement
12   Etat Laponie = new Etat("Laponie", 2);
13   em.persist(Laponie);
14 }
```

Méthodes de la classe EntityManager

find()

```
Etat e = find (Etat.class , 1);
```

createQuery()

```
TypeQuery <Etat> q;
String reqJPQL = "SELECT e FROM Etat e WHERE population >= :v";
q = em.createQuery(reqJPQL);
q.setParameter("v", 1000);
List <Etat> etats = q.getResultList();
```

merge()

```
Laponie.setPopulation(10);
em.merge(Laponie);
```

remove()

```
em.remove(Laponie);
```

Conclusion

- Interface de programmation pour la gestion de base de données relationnelles,
- Manipulation des données d'un SGBD comme des objets,
- Sauvegarde transparente des informations,
- Abstraction de SQL.
- Différentes implémentations ;
  - Hibernate (en TP),
  - EclipseLink (implémentation de références),
  - ...

Avantages

- Pool de connexions sur le serveurs,
- Mécanisme de transaction en JPA,
- Sécurité accrue.

Inconvénients

Les avantages ont un prix : plus lents que d'autres mécanismes.