

Hibernate : La persistance des objets Java

David THIBAU – 2020
david.thibau@gmail.com

Agenda

- Introduction à la problématique de persistance
- Le projet Hibernate
- Premier pas avec Hibernate
- Hibernate et JPA
- Mapping basique
- Mapping de l'héritage
- Mapping des associations
- Cascading et composition
- Accès en lecture aux objets
- Gestion des transactions
- Annexes


Introduction au mapping Java - SGBDR

Plan

- Origines du problème
- L'impedance mismatch
- Architectures de la persistance
- Historique des solutions
- Présentation de l'exemple
- Rappel sur JDBC / SQL

Origines du problème

Constataction

- Les données des systèmes d'informations sont stockées majoritairement dans des SGBDR.
 - Les applications utilisent de plus en plus le paradigme objet.
- 
- Object / relational paradigm mismatch.
 - Importance et coûts souvent sous-estimé.

Origines du problème

La persistance des objets

- Définition

- La persistance est une propriété attachée à une donnée qui indique que celle-ci survit au programme qui l'a créée.

- Implications

- C'est une propriété des données des objets du métier (Compte Client, Livre, etc.)
- ... donc des objets les plus importants dans le logiciel.

Impedence mismatch

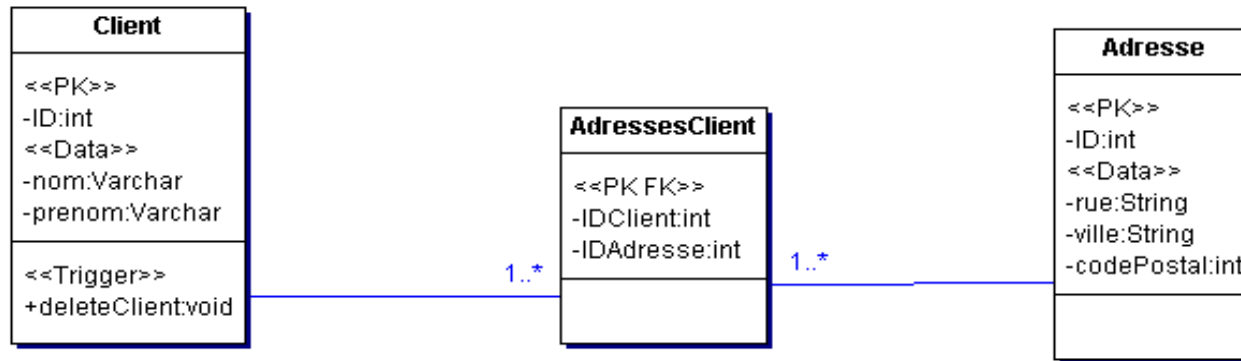
Problématique technique

- Modèle objet
 - Basé sur des principes de génie logiciel.
 - Modélise aussi bien les données que les traitements.
- Modèle relationnel
 - Basé sur des principes mathématiques.
 - Ne modélise que les données.
- Plus les modèles sont éloignés et plus il faudra écrire de code pour franchir le fossé et plus les performances seront affectées !

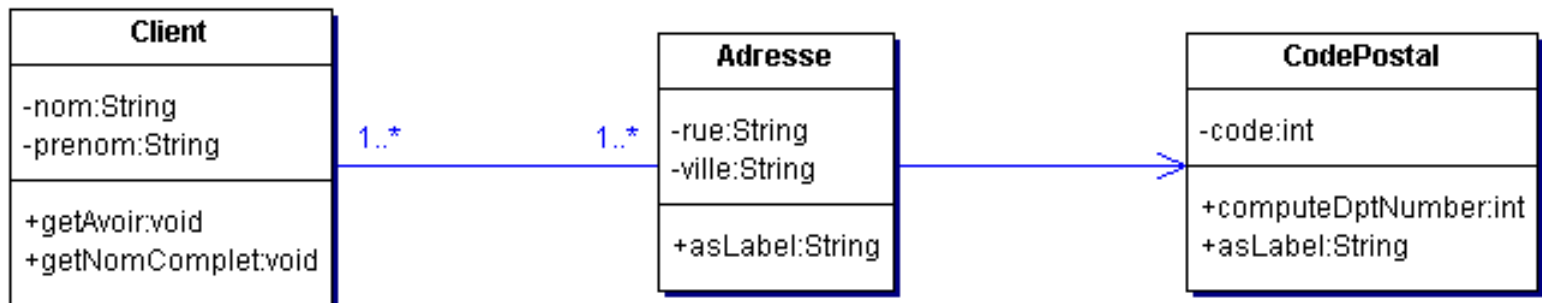
Impedence mismatch

Apparences trompeuses

Modèle de données



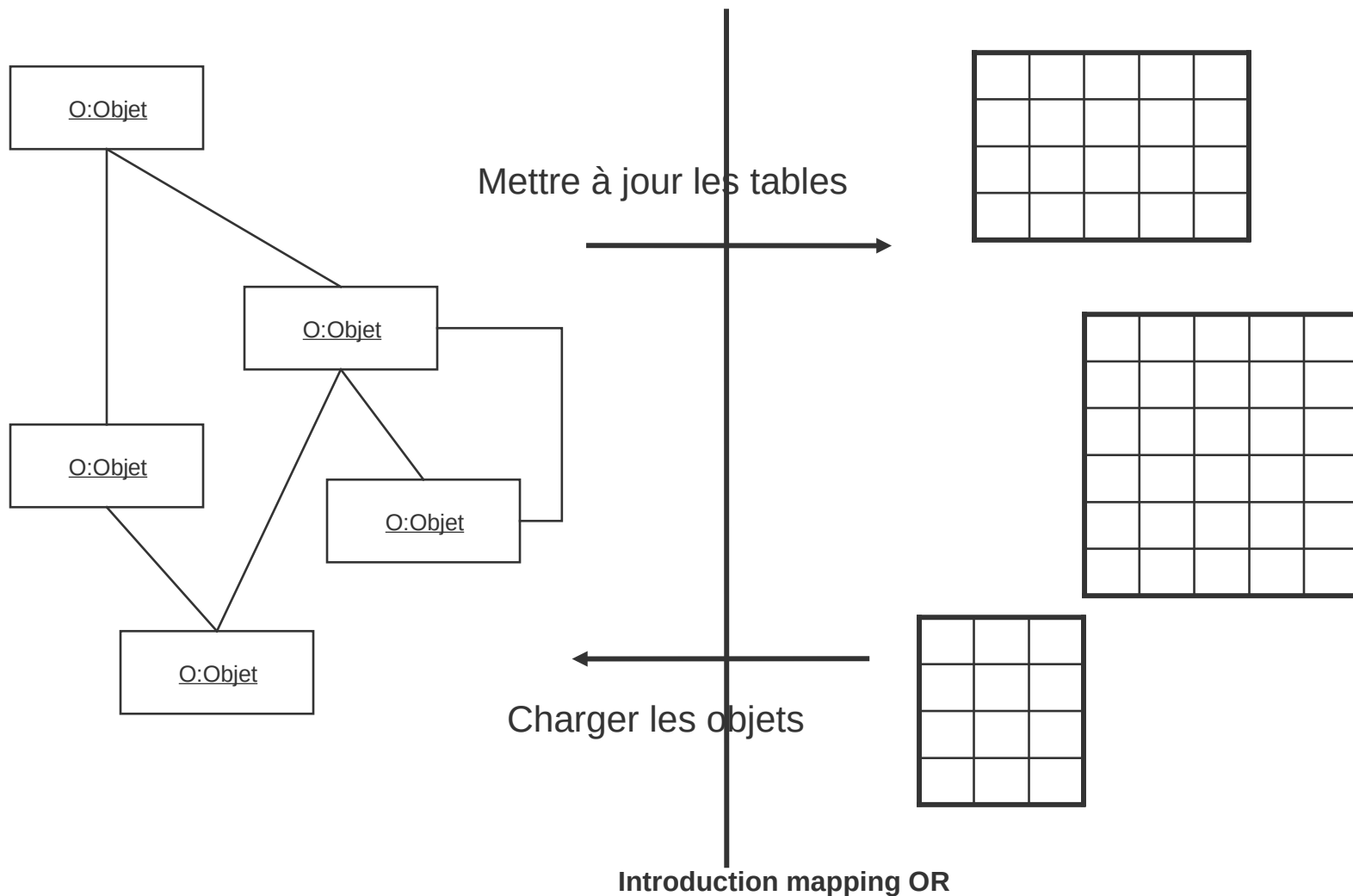
Modèle objet





Impedence mismatch

Instances persistantes vs Occurrences



Impedence mismatch

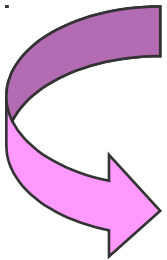
Problèmes

- Objet transient / objet persistent
- Identification et unicité d'un objet
- Persistance d'une grappe d'objet (sauvegarde, chargement et navigation)
- Représentation de l'héritage
- Représentation des associations
- Représentation des associations polymorphes
- Gestion des transactions utilisateurs (cache d'objets)

Impedence mismatch

Problématique culturelle

- **Objet**
 - « On ne devrait pas utiliser des bases relationnelles pour stocker des objets. »
- **Données**
 - « Le modèle objet devrait être dirigé par le schéma de la base de données. »



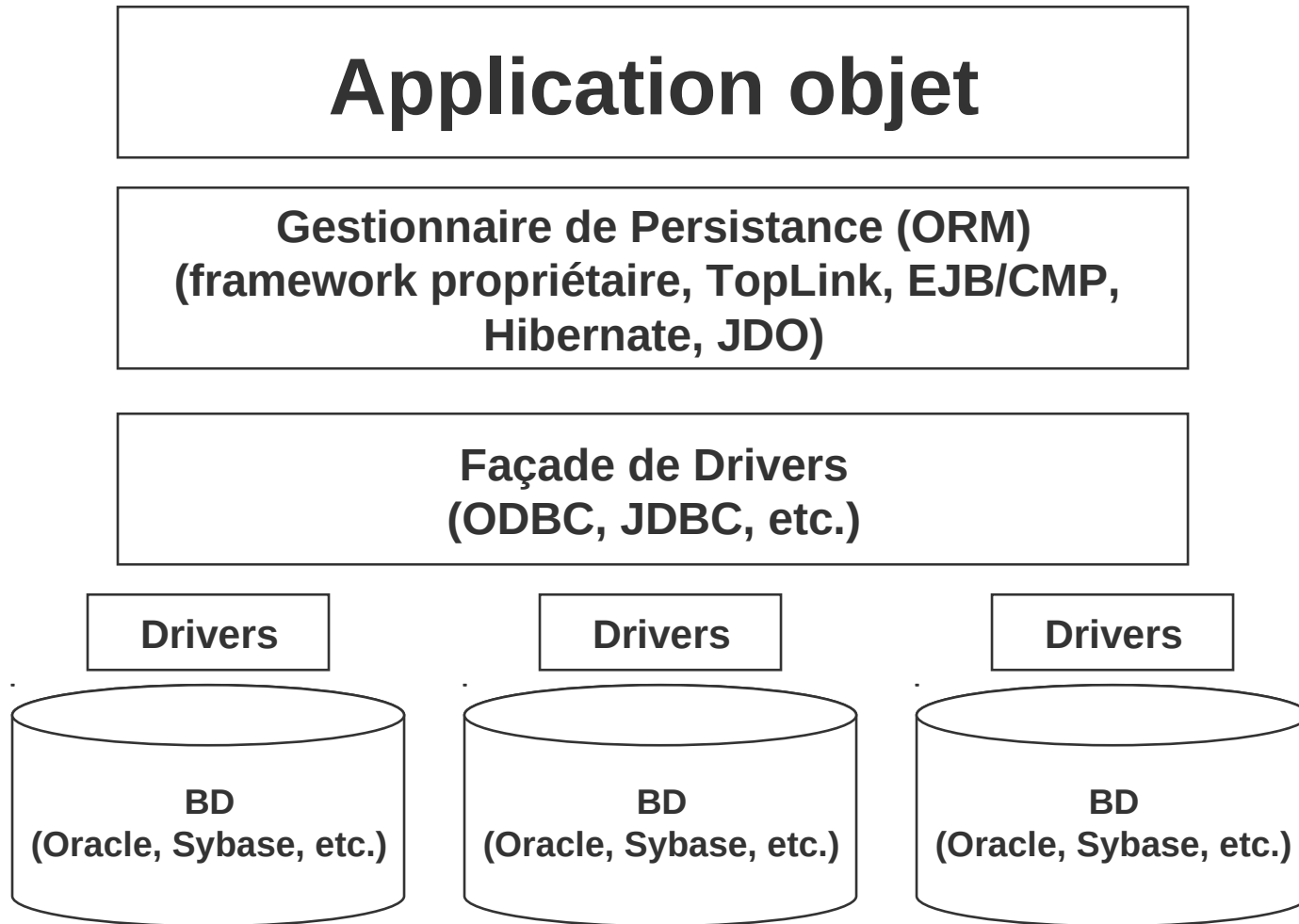
Risque important d'accroître le fossé !

Impedence mismatch

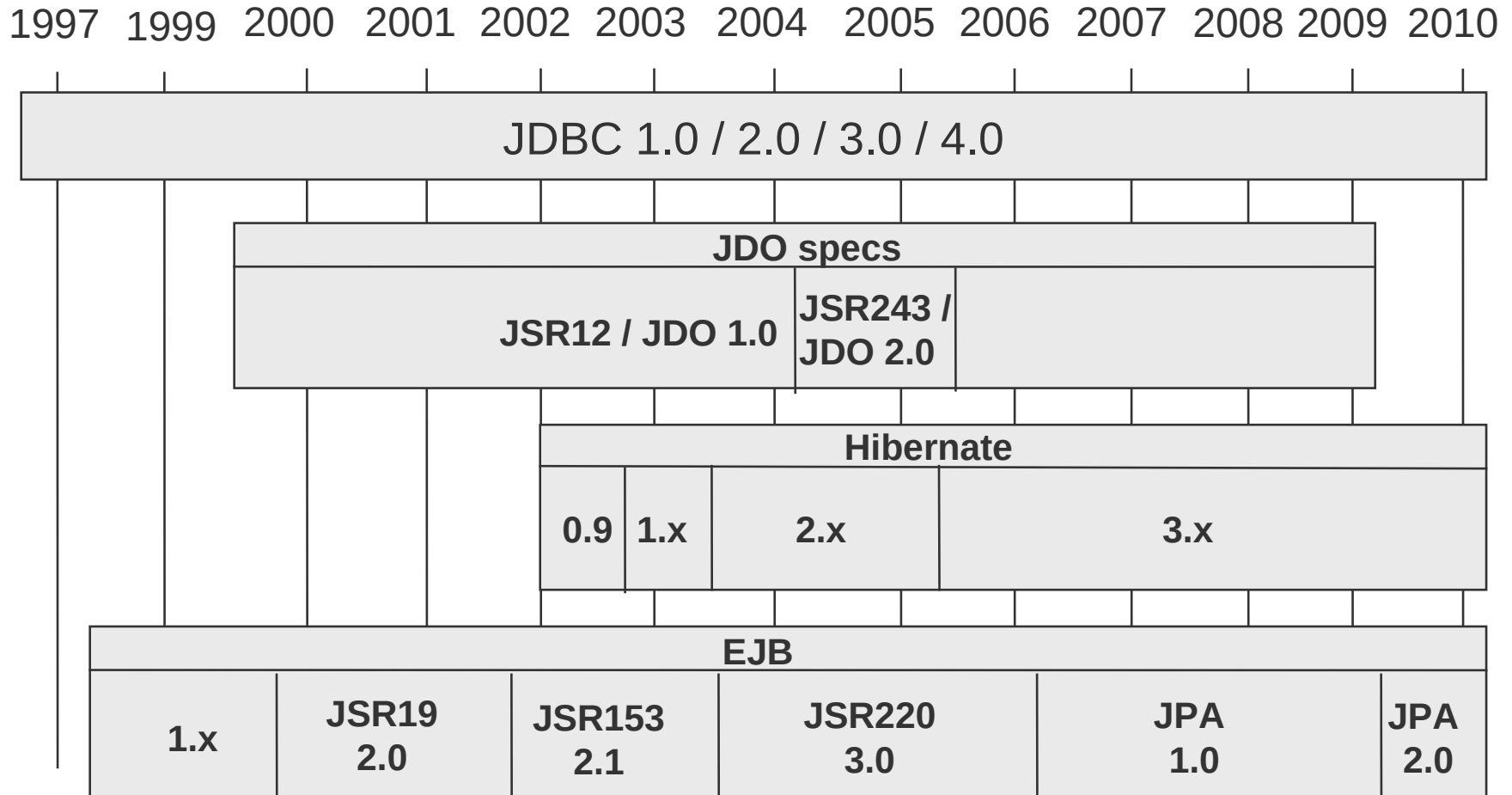
Coûts

- Code dédié au mapping (JDBC/SQL) coûte de 30% à 40%.
- Pas souvent satisfaisant
 - Les problèmes sont à la charge du développeur.
 - On a contraint le modèle objet par le modèle relationnel → perte des avantages objet.
 - Code sujet à erreur (mélange SQL / Java).
 - Une évolution du modèle relationnel a un impact fort.
- Effort important sur un service technique et non pas métier.

Architectures de la persistance



Historique des solutions Mapping SGBDR - Java



Historique des solutions (après 2010)

- Hibernate s'est imposé comme la solution de mapping la plus utilisée
- Les lead developers ont été embauché par Jboss (désormais RedHat) en 2003
- Hibernate et JPA continuent d'évoluer (... plus doucement)
 - 2011 Hib 4.0 : Multi-tenancy, Internationalisation
 - 2013 Hib 4.3.0, JPA 2.1
 - 2018 Hib 5.3, JPA 2.2
- D'autres projets connexes à l'ORM sont identifiés :
 - Hibernate Search : Intégration avec le moteur full-text Lucene
 - Hibernate Validator : Contraintes sur le modèle du domaine
 - Hibernate OGM : Application au NoSQL

Historique des solutions JDBC

- API indépendante de la base de données
- Permet d'envoyer des requêtes SQL et de récupérer le résultat dans une structure Java : le ResultSet.
- Le code SQL est mélangé avec le code Java.
- Le travail de mapping O/R est à la charge du développeur.

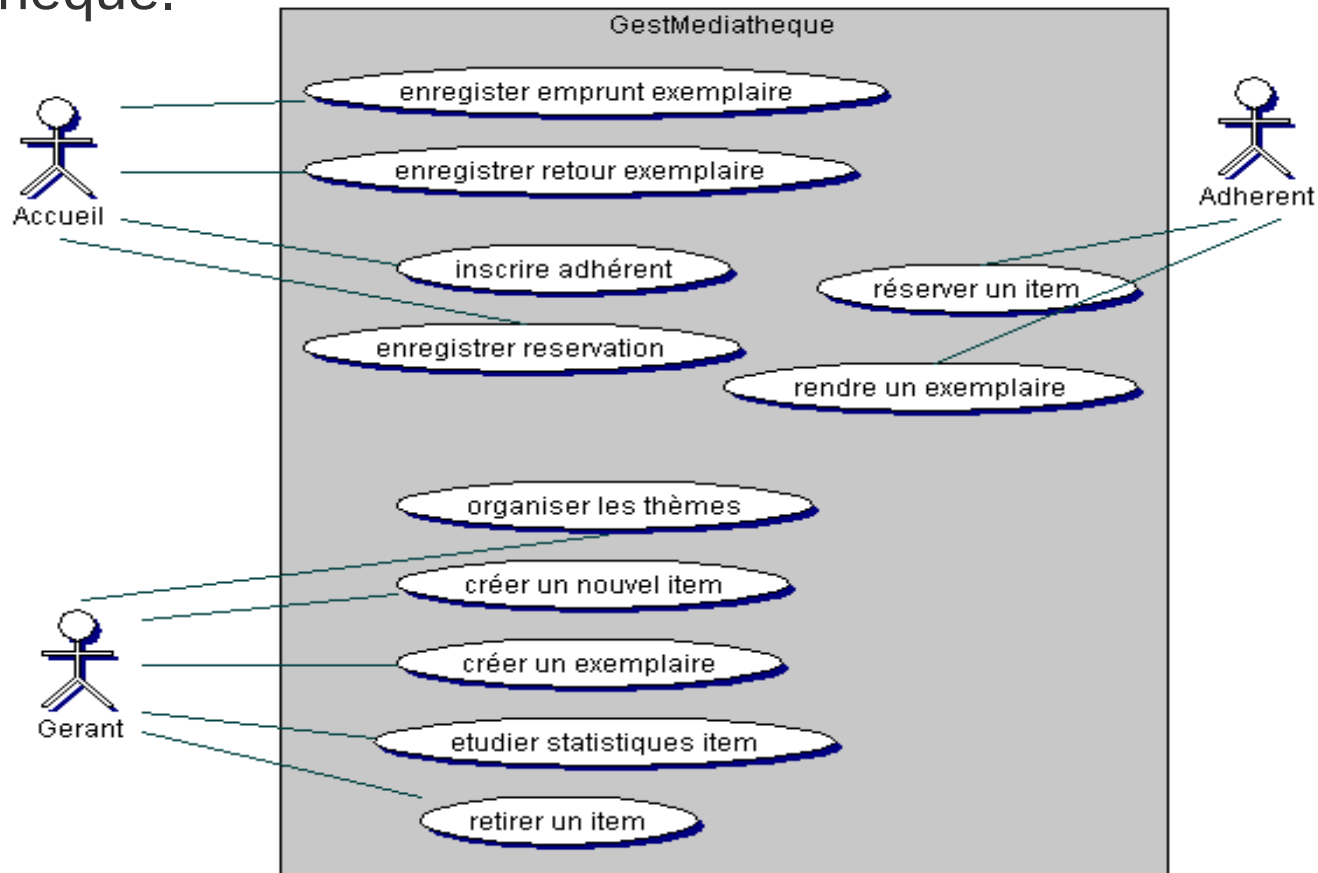
Historique des solutions Hibernate

- Développé par Gavin King pour pallier aux EJB.
- Open source connaissant un grand succès.
- Persiste des POJO (Plain Old Java Object).
- Traite l'ensemble de la problématique de persistance Java.
- Non-intrusif
- Dédié principalement aux bases de données relationnelles
- A servi de base pour JPA et les EJB 3.0.

Présentation de l'exemple

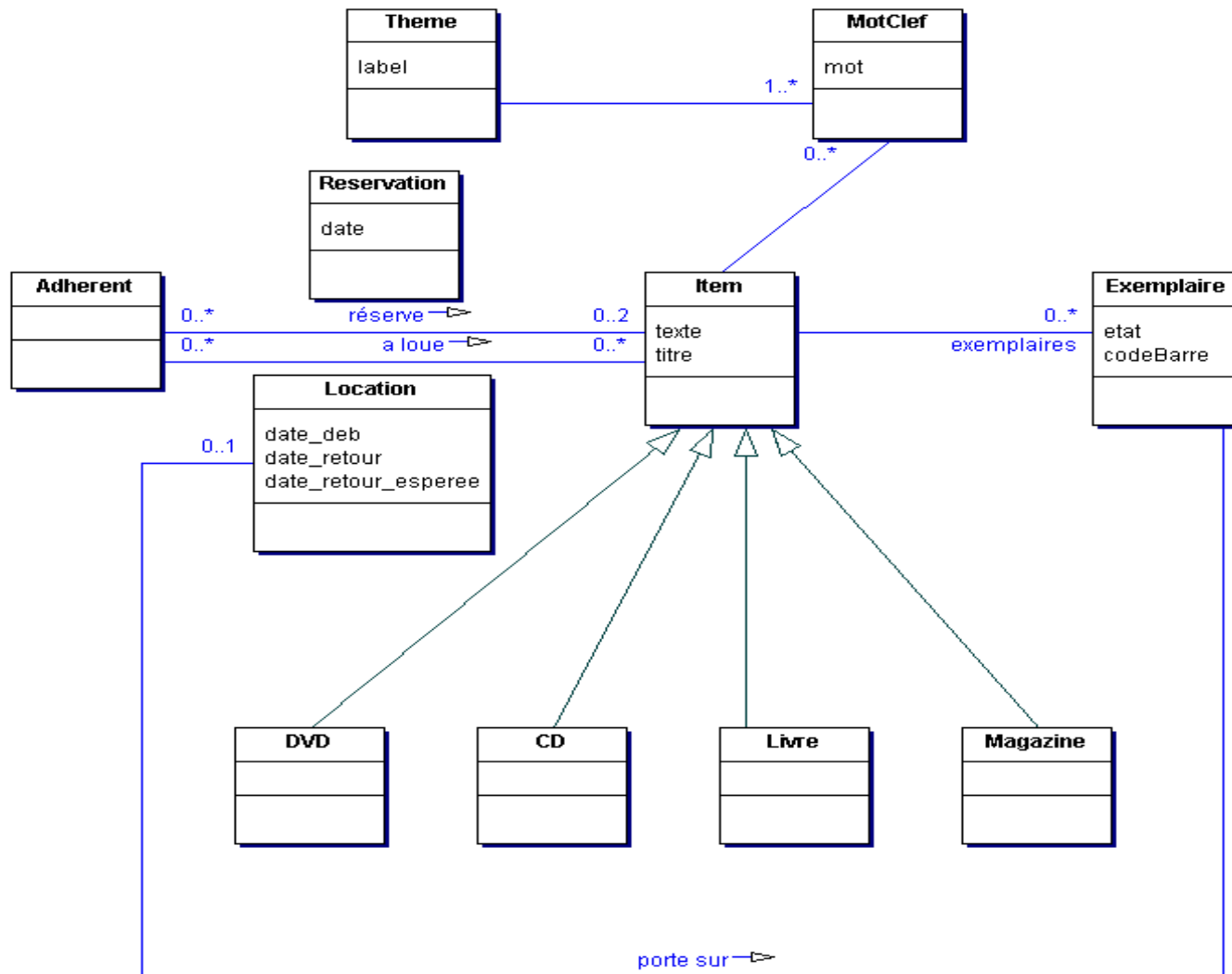
Expression du besoin

- On se propose de mettre en place un logiciel de gestion d'une médiathèque.

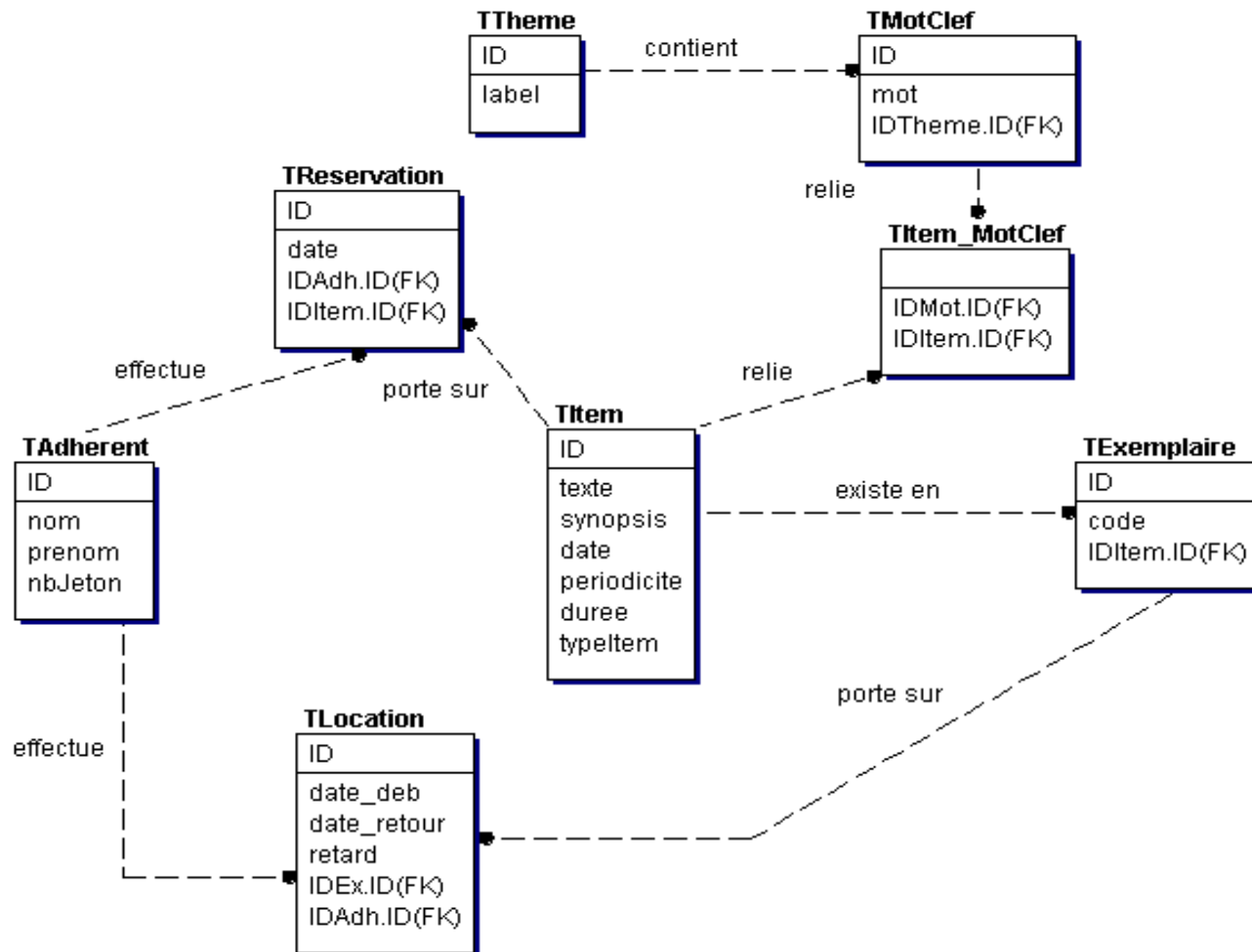


Présentation de l'exemple

Modèle du domaine



Présentation de l'exemple MCD implémenté en base



Rappel JDBC / SQL

Mécanique

- On utilise JDBC comme suit
 - Chargement du Driver
 - Obtention d'une connexion
 - Fabrication d'un statement et de sa chaîne SQL
 - Exécution du statement (`executeQuery` / `executeUpdate`)
 - Lecture du résultat (`ResultSet`)
 - Fermeture de la connexion

Rappel JDBC / SQL

Exemple – chargement driver

- Une classe Helper pour l'obtention de la connexion

```
public class DBHelper {
    static {
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("problème de chargement de driver");
        }
    }

    public static Connection getConnection() throws SQLException {
        Connection connection = DriverManager.getConnection(
            "jdbc:mysql://172.16.102.42/Mediatheque",
            "hve", "pwd");
        return connection;
    }
}
```

Rappel JDBC / SQL

Exemple – requête en lecture (1)

- Une classe appliquant le pattern DAO

```
public class ThemeDAO {
    public List<Theme> getAllThemes(){
        List<Theme> ret = new ArrayList<Theme>();
        try {
            Connection connection = DBHelper.getConnection();
            String stmt ="Select * from TTheme";
            PreparedStatement ps = connection.prepareStatement(stmt);
            ResultSet rs = ps.executeQuery();
            while(rs.next()){
                Theme current = new Theme();
                current.setId(rs.getInt("ID"));
                current.setLabel(rs.getString("label"));
                ret.add(current);
            }
            connection.close();
        } catch (SQLException e) {e.printStackTrace(); }
        return ret;
    }
}
```

Rappel JDBC / SQL

Exemple – requête en lecture (2)

- On rapatrie un thème et ses mot-clefs

```
public Theme getThemeCompleet(String label){
    Theme ret = new Theme();
    try {
        Connection connection = DBHelper.getConnection();
        String stmt = "Select * from TTheme where label = ?";
        PreparedStatement ps = connection.prepareStatement(stmt);
        ps.setString(1,label);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            ret.setId(rs.getInt("ID"));
            ret.setLabel(rs.getString("label"));
        }
        stmt = "Select * from tmotclef where IdTheme =" + ret.getId();
        ps = connection.prepareStatement(stmt);
        rs = ps.executeQuery();
        while (rs.next()) {
            MotClef mot = new MotClef(rs.getString("mot_clef"));
            mot.setId(rs.getInt("ID"));
            ret.addMotClef(mot);
        }
    }
}
```

....

Rappel JDBC / SQL

Exemple – requête en écriture (1/2)

- On crée un thème et on récupère son ID.

```
public void createTheme(Theme t) {
    try {
        Connection connection = DBHelper.getConnection();
        // crée le nouveau thème
        String stmt = "insert into TTheme(label) values(?)";
        PreparedStatement ps = connection.prepareStatement(stmt);
        ps.setString(1,t.getLabel());
        ps.executeUpdate();

        // récupère l'id calculé par la base
        stmt = "select * from TTheme where label = '" + t.getLabel() + "'";
        ps = connection.prepareStatement(stmt);
        ResultSet rs = ps.executeQuery();
        int id = 0;
        if (rs.next()){
            id = rs.getInt("ID");
        }
    }
}
```

Rappel JDBC / SQL

Exemple – requête en écriture (2/2)

- Il faut maintenant créer les mot-clefs en positionnant la clé étrangère pour établir la relation Theme - MotClef

```
// insère les motclefs avec la bonne clé étrangère
Iterator<MotClef> it = t.getMotClefs();
while(it.hasNext()){
    stmt = "insert into tmotclef(mot_clef,IDTheme) values (?,?)";
    ps = connection.prepareStatement(stmt);
    ps.setString(1,it.next().getMot());
    ps.setInt(2,id);
    ps.executeUpdate();
}
```

Rappel JDBC / SQL

Exemple – requête de suppression

- Supprimer le thème et ses mot-clefs

```
public void deleteTheme(Theme t) {
    try {
        Connection connection = DBHelper.getConnection();
        // supprime l'enregistrement
        String stmt = "delete from TTheme where ID = ?";
        PreparedStatement ps = connection.prepareStatement(stmt);
        ps.setInt(1, t.getId());
        ps.executeUpdate();

        // supprime aussi les mots clefs attachés
        Iterator<MotClef> it = t.getMotClefs();
        while(it.hasNext()){
            stmt = "delete from tmotclef where ID = ?";
            ps = connection.prepareStatement(stmt);
            ps.setInt(1, it.next().getId());
            ps.executeUpdate();
        }
        ...
    }
}
```

Rappel JDBC / SQL

Exemple – requête de mise à jour

- Met à jour le label d'un thème

```
public void updateTheme(Theme t) {  
    try {  
        Connection connection = DBHelper.getConnection();  
        // met à jour l'enregistrement  
        String stmt = "update TTheme set label = ? where ID = ?";  
        PreparedStatement ps = connection.prepareStatement(stmt);  
        ps.setString(1,t.getLabel());  
        ps.setInt(2,(int)t.getId().longValue());  
        ps.executeUpdate();  
        connection.close();  
        ...  
    }  
}
```

Exercice

- Exercice 1 Rappel JDBC