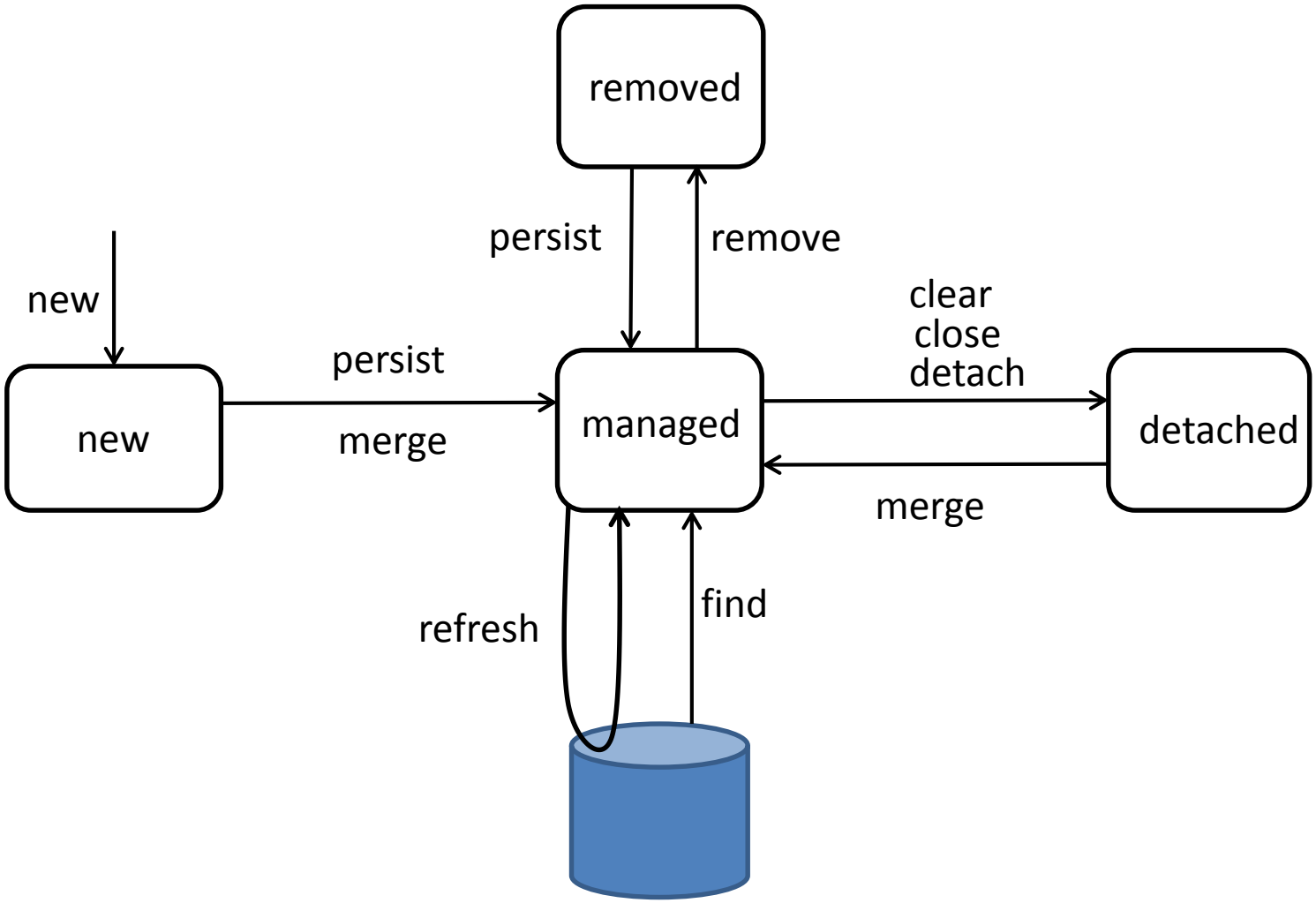
Module 2 : JPA

JPA ?

- Java Persistence API
- Standardisation ORM
- JSR 317
- Construit à partir des standards de facto
  - Hibernate
  - TopLink/Eclipse Link
  - JDO


Interface uniquement

Cycle de vie

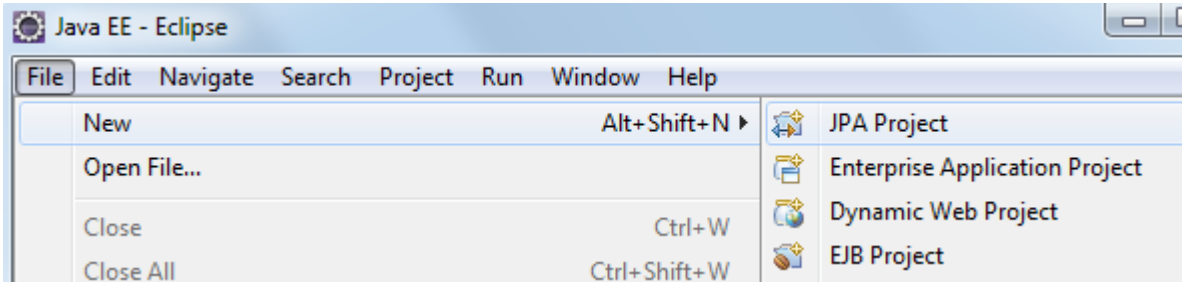

removed
persist
remove
new
clear
close
detach
persist
new
merge
managed
detached
merge
refresh
find

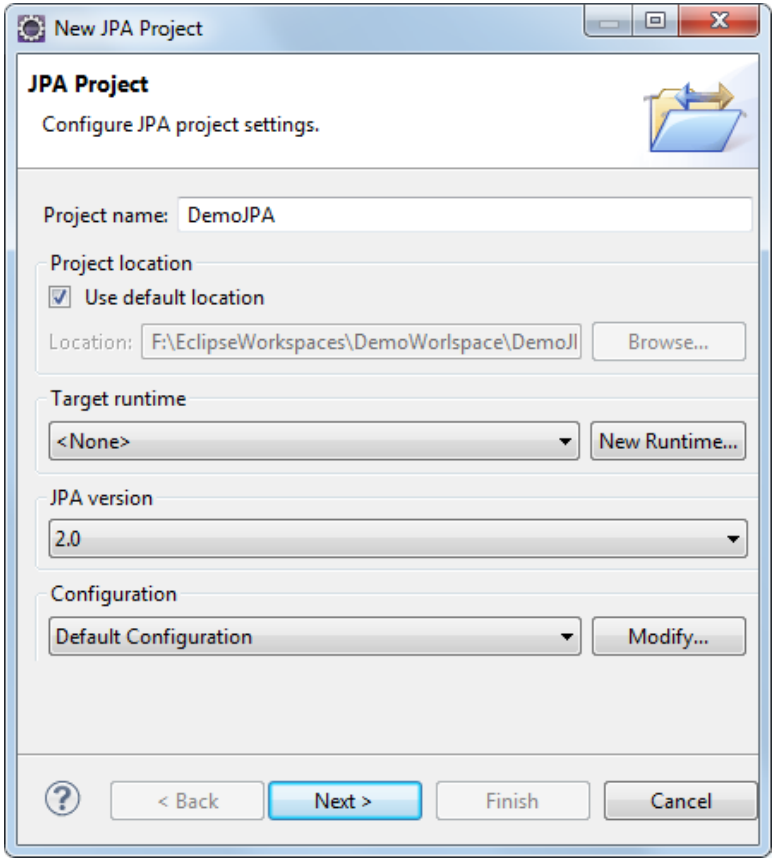
Mise en place

Mise en place

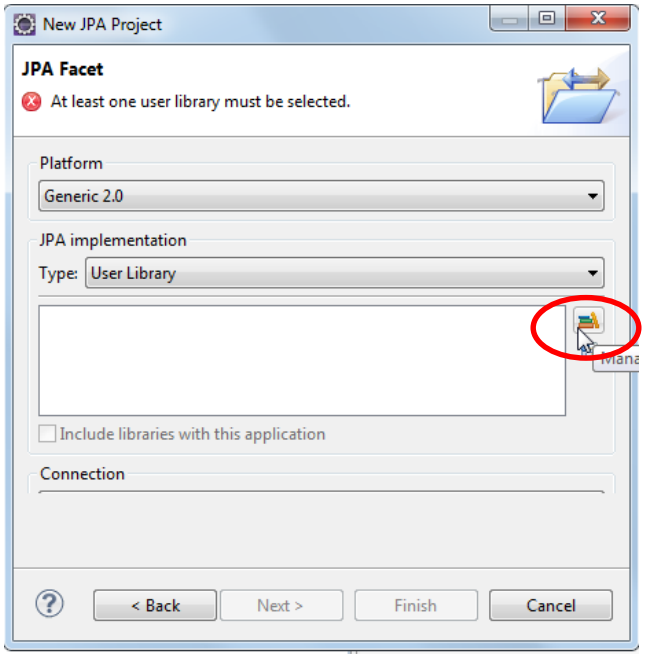
1. Installer l'environnement de développement
2. Créer le projet
3. Intégrer une implémentation de JPA
4. Intégrer le pilote JDBC
5. Configurer JPA
6. Décrire la correspondance modèle objet <--> modèle relationnel

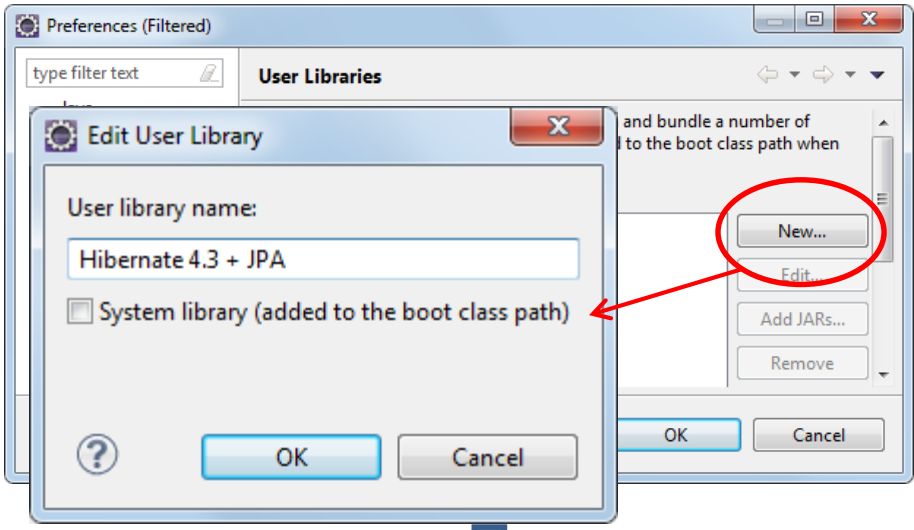
Mise en place (2) - Projet

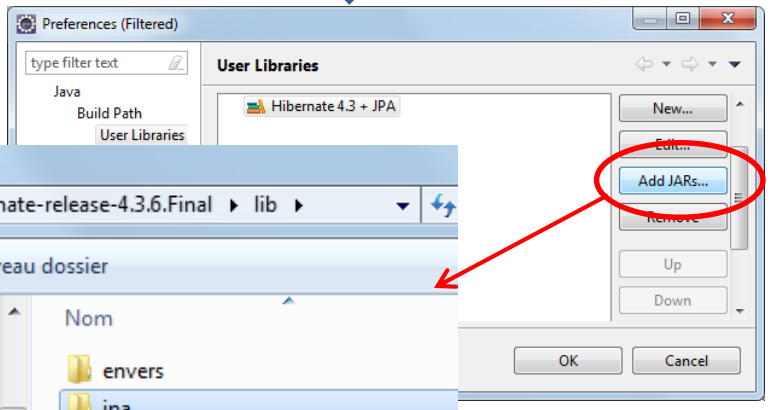


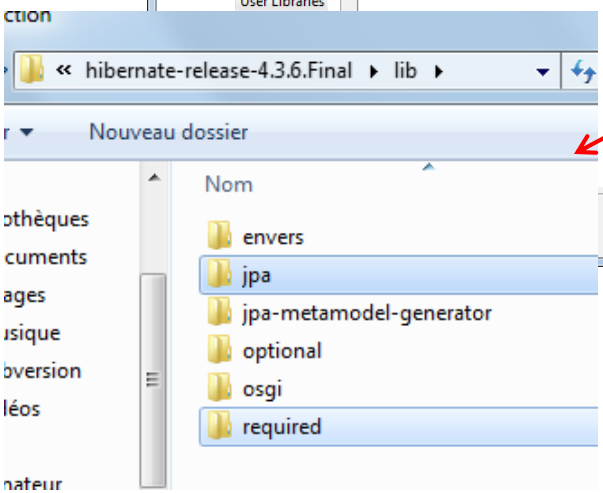


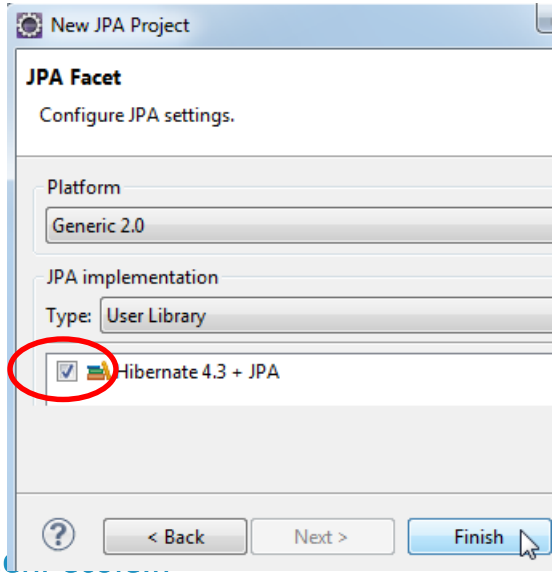
Mise en place (3) - Hibernate



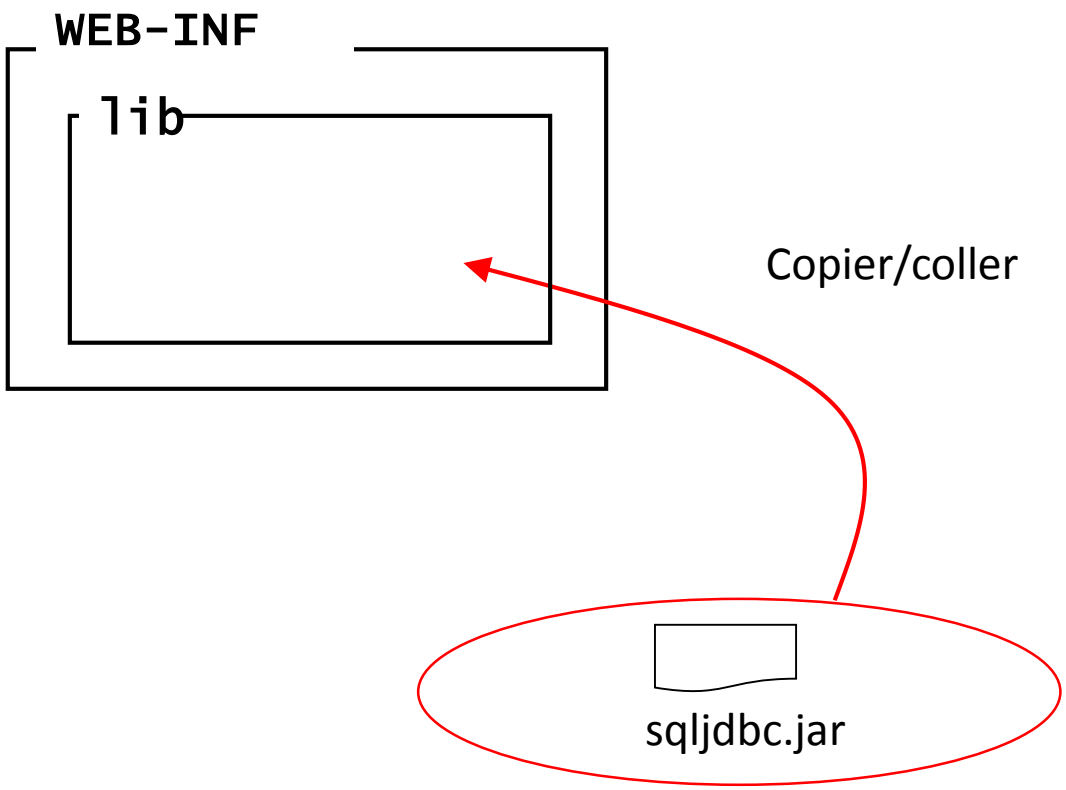






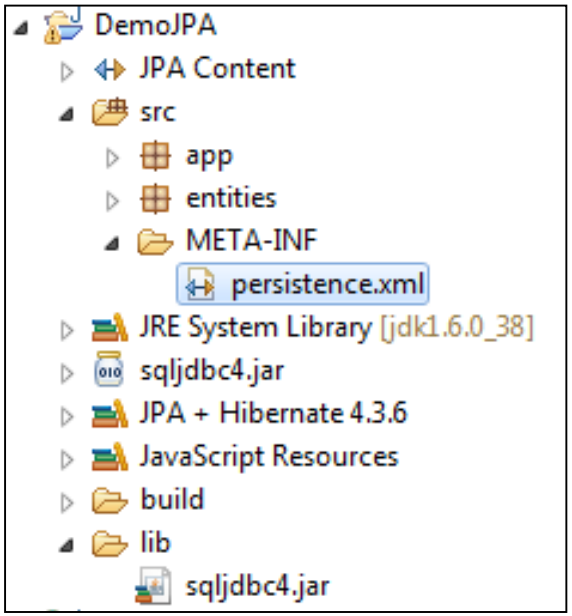


Mise en place (4) - Pilote JDBC



Mise en place (5) - Configuration

- Fichier persistence.xml




```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/
                ns/persistence http://java.sun.com/
              xml/ns/persistence/persistence_2_0.xsd">

  <persistence-unit name="..." transaction-type="...">

     <provider>...</provider>

     <properties>
        ...
     </properties>

        </persistence-unit>

</persistence>
```

- Packagé dans META-INF

Mise en place (5) – Principaux éléments

- Identité
  - `<persistence-unit name="...">`
- Implémentation JPA
  - `<provider>`
- Classes mappées
  - `<class>`
- Gestionnaire de transaction
  - `<persistence-unit transaction-type="...">`
- Source de données
  - `<jta-data-source>`
  - `javax.persistence.jdbc.driver`
  - `javax.persistence.jdbc.url`
  - `javax.persistence.jdbc.user`
  - `javax.persistence.jdbc.password`

Mise en place (5) - Exemple

```
<persistence-unit name="DemoPU" transaction-type="RESOURCE_LOCAL">

  <provider>org.hibernate.ejb.HibernatePersistence</provider>

  <class>entite.Produit</class>

  <properties>
    <property name="javax.persistence.jdbc.url"
              value="jdbc:sqlserver://localhost:1433
                                        ;databaseName=JavaEE_Av" />
    <property name="javax.persistence.jdbc.driver"
              value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
    <property name="javax.persistence.jdbc.user"
              value="java" />
    <property name="javax.persistence.jdbc.password"
              value="Secret007" />

    <property name="hibernate.show_sql" value="true"/>

  </properties>

</persistence-unit>
```

Mise en place (6) - Mapping

- Entité
  - `@Entity`
  - `@Table(name="nomTable")`
- Identifiant
  - `@Id`
  - `@GeneratedValue(strategy=GenerationType.IDENTITY)`
- Colonne
  - `@Column(name="nomColonne")`
- Comportement particulier
  - `@Transient`
  - `@Basic(fetch=LAZY)`

Mise en place (6) – Exemple

```
@Entity
@Table(name = "PRODUITS")
public class Produit implements Serializable {
private static final long serialVersionUID = 1L;

@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private int id;

@Column(name="REF")
private String reference;

// Mappé par défaut
private String marque;
private float prix;
```

Gestion de la persistance

Obtenir une unité de travail

```
EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("DemoPU");

EntityManager em = emf.createEntityManager();

// em.close();
// emf.close();
```

ou

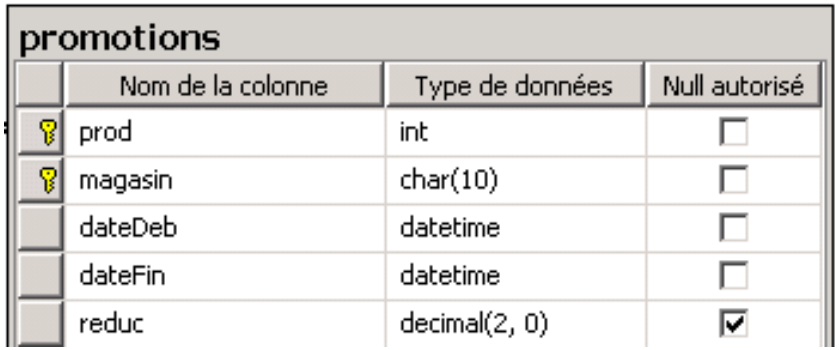
```
@PersistenceContext(name="DemoPU")
EntityManager em;
```

Utilisation – CRUD

```
EntityTransaction t =  em.getTransaction();
t.begin();

// Create
Client c1 = new Client("AHO" , "Alfred");
em.persist(c1);

// Retreive
Client c2 = em.find(Client.class, 2);

// Update
c2.setPrenom("Bernard");

// Delete
em.remove(c2);

t.commit();
```

Utilisation – Clef composite (1)

- Classe entité + classe clé
- Contraintes classe clé
  - constructeur par défaut
  - Serializable
  - redéfinir equals() et hashcode()



promotions

	Nom de la colonne	Type de données	Null autorisé
🔑	prod	int	☐
🔑	magasin	char(10)	☐
	dateDeb	datetime	☐
	dateFin	datetime	☐
	reduc	decimal(2, 0)	☑

```
@Entity
@Table(name="PROMOTIONS")
@IdClass(value=PromotionPK.class)
public class Promotion {

@Id
private int idProd;
@Id
private String idMagasin;
```

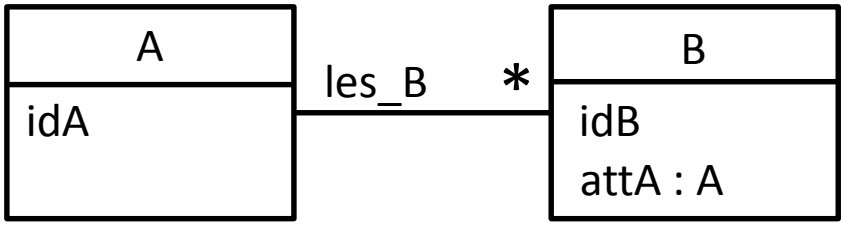
Utilisation – Clef composite (2)

```
public class PromotionPK implements Serializable {

private int idProd;
private String idMagasin;

public PromotionPK() { super(); }

@Override
public boolean equals(Object obj) {
    if (!(obj instanceof PromotionPK)) { return false; }

    PromotionPK objCompare = (PromotionPK)obj;

    return ( objCompare.getIdMagasin().equals(getIdMagasin())
        && objCompare.getIdProd()==getIdProd());
}

@Override
public int hashCode() {
    return getIdProd() + getIdMagasin().hashCode();
}
```

Utilisation – Relations

- 4 types
  - @OneToOne
  - @OneToMany
  - @ManyToOne
  - @ManyToMany
- 2 navigations
  - Unidirectionnelle
  - Bidirectionnelle

Utilisation – Relation 1-* bidirectionnelle




```
@Entity
public class A {
    @Id
    int idA;

    @OneToMany(mappedBy="attA")
    Set<B> les_B;
```

```
@Entity
public class B {
    @Id
    int idB;

    @ManyToOne
    A attA;
```

Utilisation - Propagation

- Cascade
  - ALL
  - MERGE
  - PERSIST
  - REFRESH
  - REMOVE

```
@OneToMany(cascade={CascadeType.PERSIST,CascadeType.MERGE})
```

Propagation même si entité source non-modifiée

JPQL

Quoi ?

- Java Persistence Query Language

Pour quoi faire ?

- Requêter des entités et leur état persistant

- Types
  - SELECT
  - UPDATE
  - DELETE

JPQL - Utilisation

- Syntaxe

```
Query reqJPQL = em.createQuery("SELECT p FROM Produit p");
List<Produit> listeProd = reqJPQL.getResultList();
```

- Paramètre nommé

```
Query reqJPQL = em.createQuery("SELECT p FROM Produit p
                    WHERE p.prix > :prixMax");
reqJPQL.setParameter("prixMax", 12.0f);
List<Produit> listeProd = reqJPQL.getResultList();
```

- MEMBER OF

```
SELECT c FROM Commande
        WHERE 'Dupont' MEMBER OF c.client.nom
```

- TYPE

```
SELECT p FROM Produit WHERE TYPE(p) IN (Ramette, Stylo)
```

Requête nommée

- Déclaration

```
@Entity
@NamedQueries (
     { @NamedQuery(    name="Client.findAll",
                       query="SELECT c FROM Client c"),
       @NamedQuery( name="Client.findByName",
                    query="SELECT c FROM Client c WHERE c.nom = :name")
      }  )
 public class Client { ... }
```

- Utilisation

```
TypedQuery<Client> query = em.createNamedQuery( "Client.findAll" ,
                                                 Client.class);
 List<Client> results = query.getResultList();
```

Questions ?