



# Programmation Par Composants 2

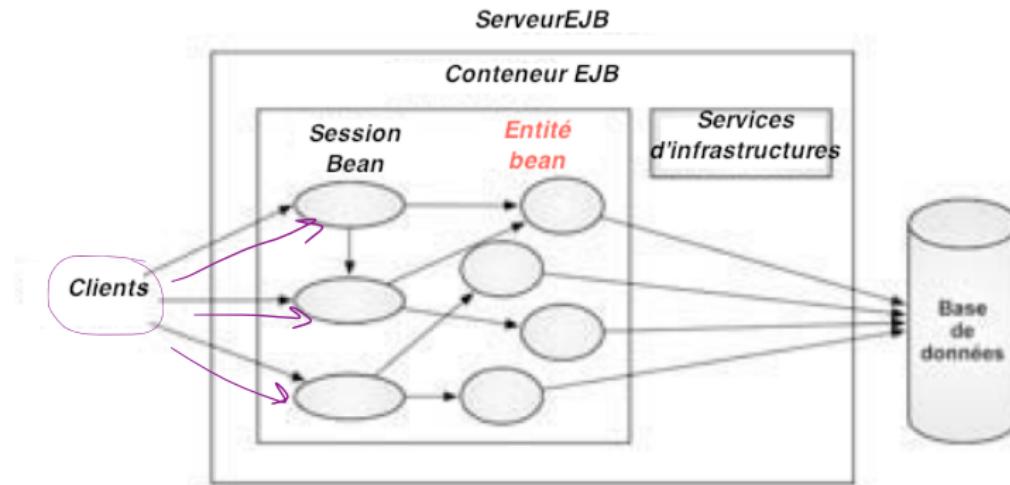
Yassamine Seladji

yassamine.seladji@gmail.com

1<sup>er</sup> février 2021

## Introduction

- ▶ Les Session Beans forment un pont entre le client et la logique métier.
- ▶ les Entity Beans forment la passerelle entre la logique applicative et les sources de données



## Entité bean

Entité bean = persistance des données

- ▶ simplifier la gestion des données au niveau d'une application.
- ▶ faciliter la sauvegarde en base de données.
- ▶ établir la relation entre votre application et vos bases de données

## Entité bean

Entité bean est une classe Java bean qui :

- ▶ implémente la classe **java.io.Serializable**.
- ▶ possède un constructeur sans argument.
- ▶ possède des attributs **private**.
- ▶ implémente les getters et setters.

## Entité bean

Chaque **objet métier** de l'application est représenté par un entité bean.

Les EJBs :

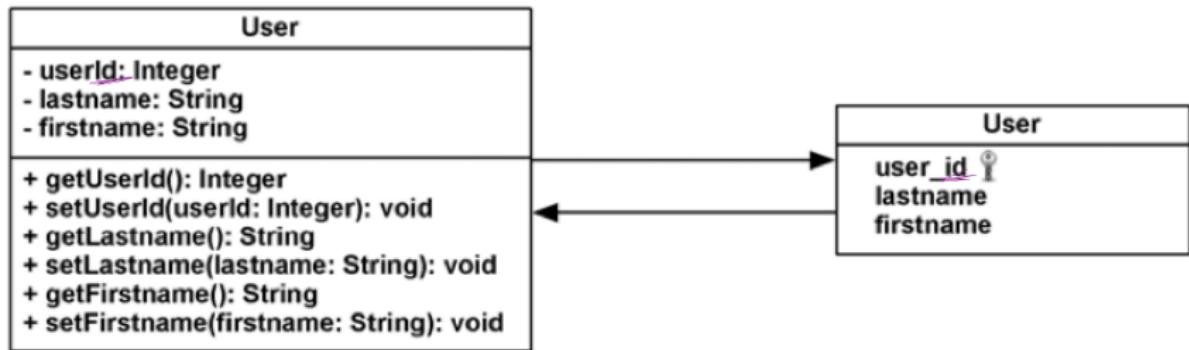
- ▶ Utilise le mapping objet/relationnel.
- ▶ Chaque **entité bean** est mappé à une **table** en base de données.

## Entité bean

Chaque **objet métier** de l'application est représenté par un **entité bean**.

Les EJBs :

- ▶ Utilise le mapping objet/relationnel.
- ▶ Chaque **entité bean** est mappé à une **table** en base de données.



## Entité bean

- ▶ **Entité Beans** permet de représenter une entité de l'application.
- ▶ **Session Beans** permet de représenter une fonctionnalité de l'application.

## Entité bean

- ▶ **Entité Beans** permet de représenter une **entité** de l'application.
- ▶ **Session Beans** permet de représenter une **fonctionnalité** de l'application.

Dans une application bancaire :

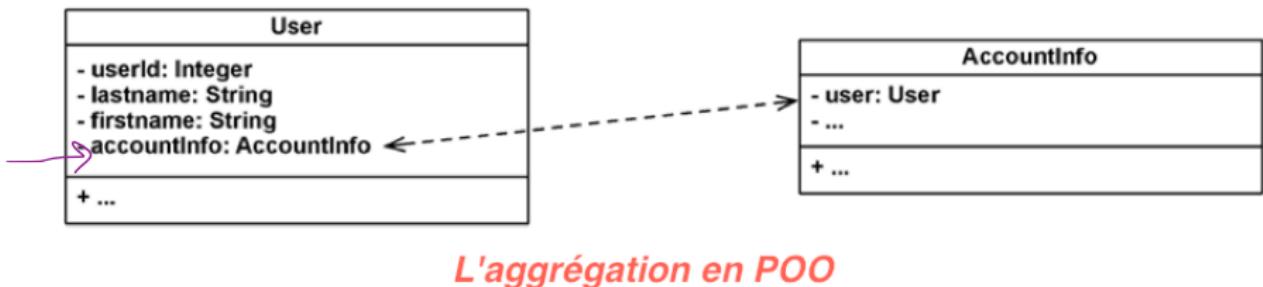
- ▶ Un compte : objet persistant  $\implies$  entité bean
- ▶ Ajouter un compte : un service  $\implies$  session bean

## Entité bean

Les propriétés des entités beans :

- ▶ chaque entité bean correspond à une table.
- ▶ Les Entités Beans doivent tous posséder un identifiant unique (clé primaire)
- ▶ Les attributs d'un entité bean sont mappées aux champs de la table associée.
- ▶ un Entité Bean possède des champs relationnels :
  - ▶ un attribut dont le type est un autre Entité Bean.
  - ▶ une clé étrangère en base de données.

## Entité bean : Exemple



## Les avantages des entités beans

- ▶ un mécanisme simple pour l'accès et la modification des données.
- ▶ entité bean a un code clair et plus facilement réutilisable.
- ▶ entité bean hérite des services tels que la gestion des transactions, de la sécurité ...

## L'écriture des entités beans

La classe de l'entité :

- ▶ peut être abstraite (`abstract`) ou concrète.

## L'écriture des entités beans

La classe de l'entité :

- ▶ peut être abstraite (`abstract`) ou concrète.
- ▶ peut aussi bien hériter d'une classe entité que d'une classe non entité, et inversement.

## L'écriture des entités beans

La classe de l'entité :

- ▶ peut être abstraite (`abstract`) ou concrète.
- ▶ peut aussi bien hériter d'une classe entité que d'une classe non entité, et inversement.
- ▶ Les méthodes, les attributs de la classe et la classe elle-même ne doivent pas être finales.

## L'écriture des entités beans

La classe de l'entité :

- ▶ peut être abstraite (`abstract`) ou concrète.
- ▶ peut aussi bien hériter d'une classe entité que d'une classe non entité, et inversement.
- ▶ Les méthodes, les attributs de la classe et la classe elle-même ne doivent pas être finales.
- ▶ doit être précédé par l'annotation `@Entity`.

```
@Entity  
public class User {  
//...  
}
```

## L'écriture des entités beans

### L'annotation `@Entity`

- ▶ Le nom de l'entité bean doit être unique dans une application.
- ▶ le nom de l'entité bean = le nom de la table
- ▶ Possibilité de définir un autre nom à la table :
  - ▶ en utilisant l'attribut **name**

## L'écriture des entités beans

### L'annotation `@Entity`

- ▶ Le nom de l'entité bean doit être unique dans une application.
- ▶ le nom de l'entité bean = le nom de la table
- ▶ Possibilité de définir un autre nom à la table :
  - ▶ en utilisant l'attribut `name`

```
@Entity(name = "MyUser")
public class User {
//...
}
```

## L'écriture des entités beans

### L'annotation `@Table`

- ▶ mapper un entité bean à une table qui ne possède pas le même nom.
- ▶ utiliser l'attribut **name** : définit le nom de la table à utiliser pour le mapping.

## L'écriture des entités beans

### L'annotation `@Table`

- ▶ mapper un entité bean à une table qui ne possède pas le même nom.
- ▶ utiliser l'attribut **name** : définit le nom de la table à utiliser pour le mapping.

```
@Entity  
@Table(name = "User")  
public class User {  
//...  
}
```

## L'écriture des entités beans

### Les champs persistants

- ▶ Les attributs d'un entité bean sont persistant par défaut.
- ▶ Il y a deux types d'annotations liés au mapping objet/relationnel :
  - ▶ les annotations liées aux propriétés.
  - ▶ les annotations liées aux colonnes.
- ▶ Les annotations peuvent se placer :
  - ▶ directement sur les champs.
  - ▶ sur les .

## L'écriture des entités beans

### Annotations liées aux propriétés simples

- ▶ **@Basic** : l'annotation par défaut pour les attributs persistants.
- ▶ **@Lob** (Large Binary Object) : préciser que l'attribut peut avoir une grande taille.

```
// utilisé pour les longs textes
@Lob
public String getArticleContent() {
    return articleContent;
}
```

## L'écriture des entités beans

### Annotations liées aux propriétés simples

- ▶ **@Temporal** : définir des propriétés dites " temporelles "
- ▶ il prend en paramètre un **TemporalType** dont les valeurs sont les suivantes :
  - ▶ **DATE** : utilisé pour la date (*java.sql.Date*)
  - ▶ **TIME** : utilisé pour l'heure (*java.sql.Time*)
  - ▶ **TIMESTAMP** : utilisé pour les temps précis (*java.sql.Timestamp*)

## L'écriture des entités beans

### Annotations liées aux propriétés simples

- ▶ **@Temporal** : définir des propriétés dites " temporelles "
- ▶ il prend en paramètre un **TemporalType** dont les valeurs sont les suivantes :
  - ▶ **DATE** : utilisé pour la date (*java.sql.Date*)
  - ▶ **TIME** : utilisé pour l'heure (*java.sql.Time*)
  - ▶ **TIMESTAMP** : utilisé pour les temps précis (*java.sql.Timestamp*)

```
@Temporal(TemporalType.DATE)
public Calendar getDateOfBirth() {
    return dateOfBirth ;
}
```

## L'écriture des entités beans

### Annotations liées aux propriétés simples

- ▶ **@Enumerated** : spécifier un ensemble de valeurs possibles pour un attribut.
- ▶ Il prend en paramètre un objet **EnumType** :
  - ▶ La valeur **EnumType.STRING** pour les chaînes de caractères.
  - ▶ La valeur **EnumType.ORDINAL** pour les entiers.

## L'écriture des entités beans

### Annotations liées aux propriétés simples

- ▶ **@Enumerated** : spécifier un ensemble de valeurs possibles pour un attribut.
- ▶ Il prend en paramètre un objet **EnumType** :
  - ▶ La valeur **EnumType.STRING** pour les chaînes de caractères.
  - ▶ La valeur **EnumType.ORDINAL** pour les entiers.

```
public enum SexType { MALE, FEMALE };  
//...  
@Enumerated(value=EnumType.STRING)  
public SexType getSex() { return sex; }
```

# L'écriture des entités beans

## Annotations liées aux colonnes simples

- ▶ **@Column** : permet de préciser le paramétrage des colonnes dans la table relationnelle.
- ▶ une description de certains attributs :
  - ▶ **name** : le nom de la colonne. Le nom de la propriété est utilisé par défaut.
  - ▶ **nullable** : si la colonne accepte des valeurs nulles ou non.
  - ▶ **updatable** : si la valeur doit être mise à jour lors de l'exécution de la requête SQL UPDATE (true par défaut).
  - ▶ **precision** : le nombre maximum de chiffre.
  - ▶ **scale** : le nombre fixe de chiffres après le séparateur décimal

## L'écriture des entités beans

### Annotations liées aux colonnes simples

- ▶ **@Column** : permet de préciser le paramétrage des colonnes dans la table relationnelle.
- ▶ une description de certains attributs :
  - ▶ **name** : le nom de la colonne. Le nom de la propriété est utilisé par défaut.
  - ▶ **nullable** : si la colonne accepte des valeurs nulles ou non.
  - ▶ **updatable** : si la valeur doit être mise à jour lors de l'exécution de la requête SQL UPDATE (true par défaut).
  - ▶ **precision** : le nombre maximum de chiffre.
  - ▶ **scale** : le nombre fixe de chiffres après le séparateur décimal

```
@Column(updatable = true, name = "price", nullable = false,  
precision=5, scale=2)  
public float getPrice() { return price; }
```

## L'écriture des entités beans

### Identificateur unique

- ▶ Un Entity Bean doit posséder un champ dont la valeur est unique.
- ▶ Ce champ permet de différencier chaque instance de l'entité des autres.
- ▶ Utiliser l'annotation **@Id** : pour dire qu'un attribut représente l'identificateur.
- ▶ Utiliser l'annotation **@GeneratedValue** : définit la stratégie de génération de l'identificateur.

## L'écriture des entités beans

### Identificateur unique

Exemple

```
@Entity
public class User implements Serializable {
    private int id;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO) // optionnel
    public int getId() {
        return id;
    }

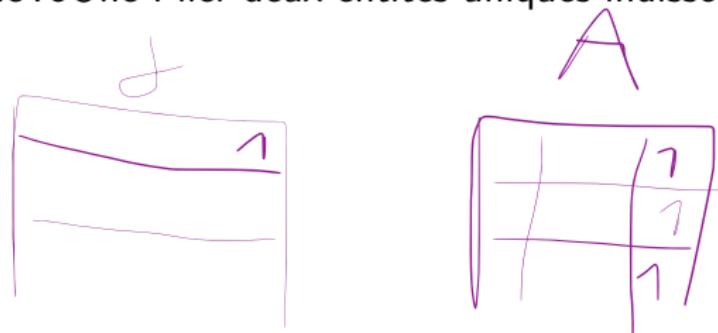
    public void setId(int id) {
        this.id = id;
    }
}
```

## L'écriture des entités beans

### Les champs relationnels

Le champs relationnel représente le lien entre différents entité bean :

- ▶ `@OneToOne` : lier deux entités uniques indissociables.



## L'écriture des entités beans

### Les champs relationnels

Le champs relationnel représente le lien entre différents entité bean :

- ▶ `@OneToOne` : lier deux entités uniques indissociables.
- ▶ `@ManyToOne` : lier à une unique instance d'une entité A, un groupe d'instances d'une entité B

## L'écriture des entités beans

### Les champs relationnels

Le champs relationnel représente le lien entre différents entité bean :

- ▶ `@OneToOne` : lier deux entités uniques indissociables.
- ▶ `@ManyToOne` : lier à une unique instance d'une entité A, un groupe d'instances d'une entité B
- ▶ `@OneToMany` : lier à un groupe d'instances d'une entité A, une unique instance d'une entité B

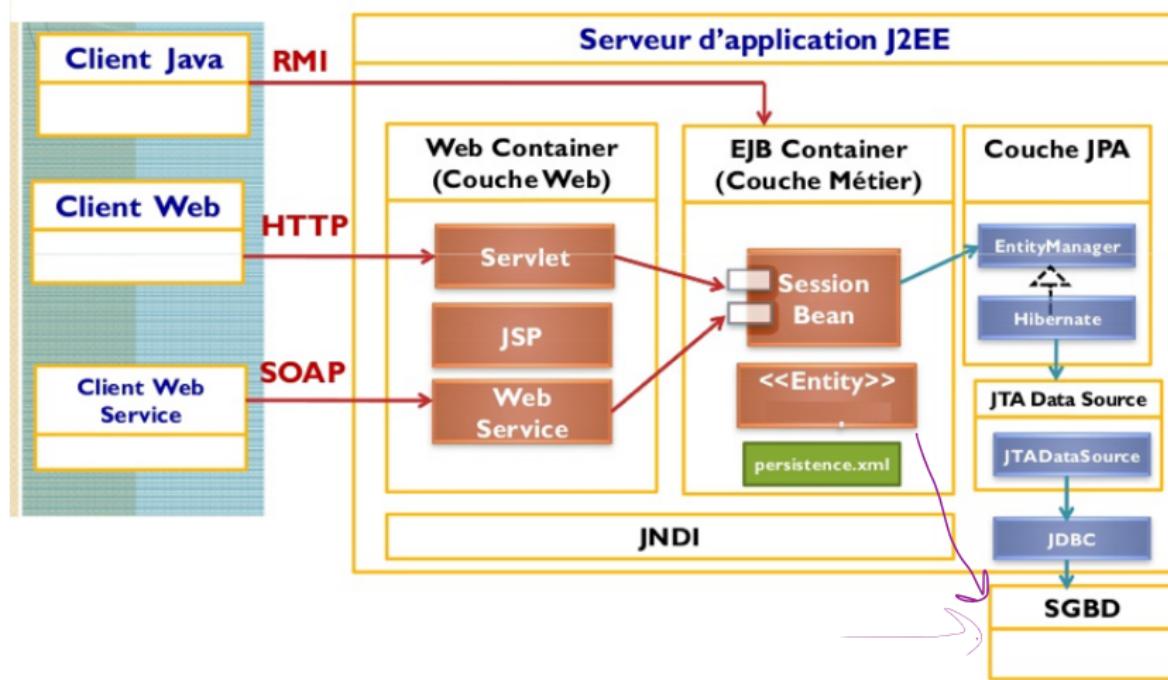
## L'écriture des entités beans

### Les champs relationnels

Le champs relationnel représente le lien entre différents entité bean :

- ▶ `@OneToOne` : lier deux entités uniques indissociables.
- ▶ `@ManyToOne` : lier à une unique instance d'une entité A, un groupe d'instances d'une entité B
- ▶ `@OneToMany` : lier à un groupe d'instances d'une entité A, une unique instance d'une entité B
- ▶ `@ManyToMany` : lier des instances de deux entités entre elles.

# L'unité de persistance



## L'unité de persistance

- ▶ Les entités beans sont managées par une **unité de persistance**.

## L'unité de persistance

- ▶ Les entités beans sont managées par une **unité de persistance**.
- ▶ L'unité de persistance permet **d'intégrer** l'utilisation des entité beans au sein d'applications Java EE

## L'unité de persistance

- ▶ Les entités beans sont managées par une **unité de persistance**.
- ▶ L'unité de persistance permet **d'intégrer** l'utilisation des entité beans au sein d'applications Java EE
- ▶ L'unité de persistance prend en charge la **sauvegarde** des informations dans la source de données.

## L'unité de persistance

Une unité de persistance est caractérisée par les points suivants :

- ▶ un ensemble d'entité Beans.

## L'unité de persistance

Une unité de persistance est caractérisée par les points suivants :

- ▶ un ensemble d'entité Beans.
- ▶ un fournisseur de persistance (Provider)

## L'unité de persistance

Une unité de persistance est caractérisée par les points suivants :

- ▶ un ensemble d'entité Beans.
- ▶ un fournisseur de persistance (Provider)
- ▶ une source de données (Datasource)

## L'unité de persistance

Le rôle de l'unité de persistance est :

- ▶ De savoir **ou** et comment stocker les informations.

## L'unité de persistance

Le rôle de l'unité de persistance est :

- ▶ De savoir où et comment stocker les informations.
- ▶ De s'assurer de l'unicité des instances de chaque identité persistante.

## L'unité de persistance

Le rôle de l'unité de persistance est :

- ▶ De savoir où et comment stocker les informations.
- ▶ De s'assurer de l'unicité des instances de chaque identité persistante.
- ▶ De gérer les instances et leur cycle de vie : c'est le gestionnaire d'entité (Entity Manager).

## Le gestionnaire d'entité

Le gestionnaire d'entité définit les méthodes qui gèrent le cycle de vie de la persistance des entités :

- ▶ La méthode **persist()**

## Le gestionnaire d'entité

Le gestionnaire d'entité définit les méthodes qui gèrent le cycle de vie de la persistance des entités :

- ▶ La méthode **persist()**
- ▶ La méthode **find()**

## Le gestionnaire d'entité

Le gestionnaire d'entité définit les méthodes qui gèrent le cycle de vie de la persistance des entités :

- ▶ La méthode **persist()**
- ▶ La méthode **find()**
- ▶ la méthode **remove()**

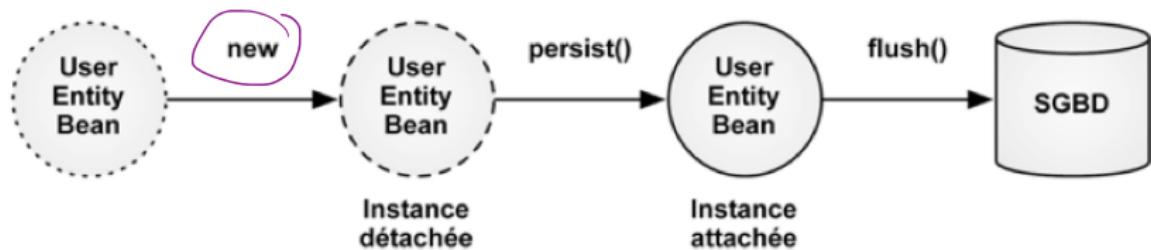
## Le gestionnaire d'entité

Le gestionnaire d'entité définit les méthodes qui gèrent le cycle de vie de la persistance des entités :

- ▶ La méthode **persist()**
- ▶ La méthode **find()**
- ▶ la méthode **remove()**
- ▶ La méthode **merge()**

## Le gestionnaire d'entité

La méthode **persist()** : Permet d'enregistrer une entité c-a-d insérer dans la base de données.



## Le gestionnaire d'entité

### La méthode **persist()**.

```
User newUser = new User();
// affecte les relations
newUser.setFirstname("Salim");
newUser.setEmail("popom@supinfo.com");
//...
entityManager.persist(newUser);
```

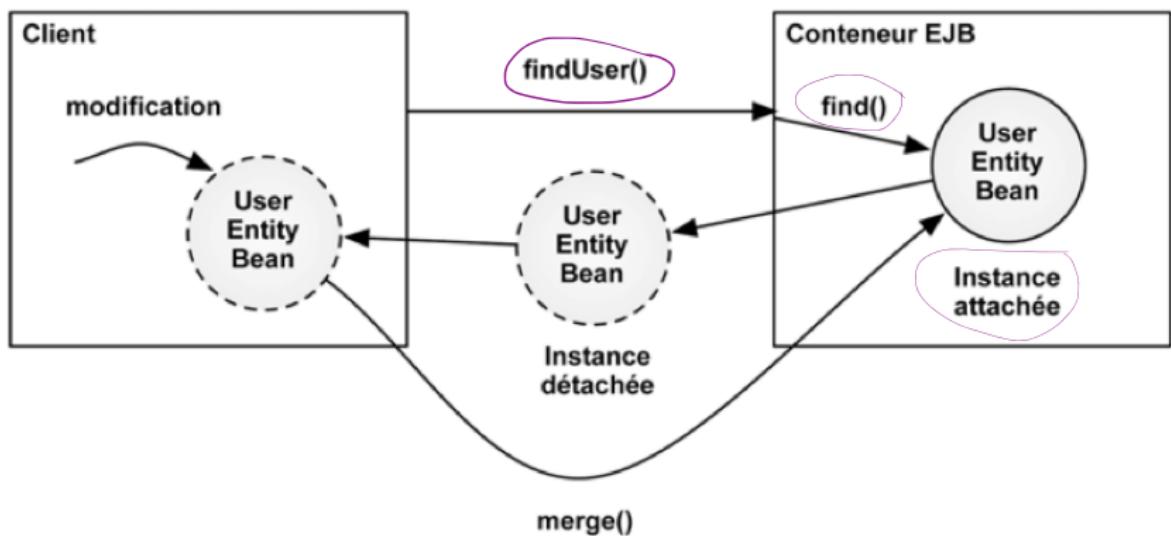
## Le gestionnaire d'entité

La méthode **find()** : permet de récupérer les entités sauvegardées.

```
User user1 = entityManager.find(User.class, 1);
if(user1 == null) { ... }
```

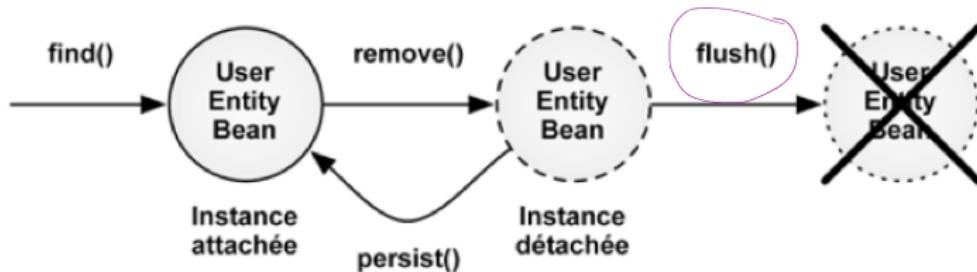
## Le gestionnaire d'entité

La méthode **merge()** : permet de modifier les entités sauvegardées.



## Le gestionnaire d'entité

La méthode **remove()** : permet de demander la suppression d'une entité de la base de données.

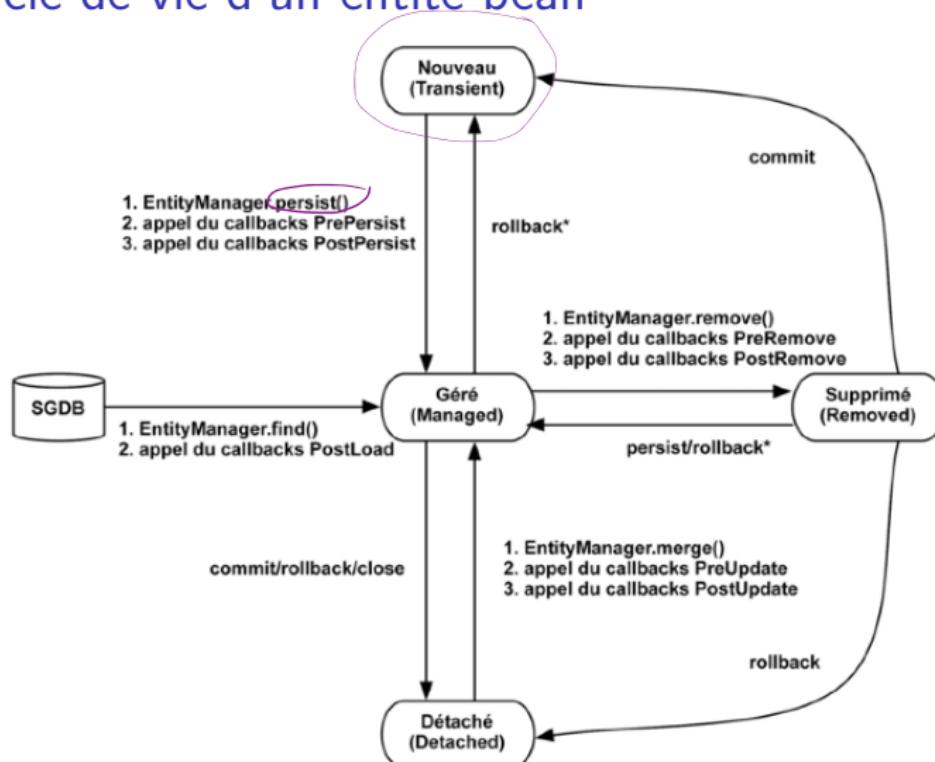


## Le gestionnaire d'entité

### La méthode **remove()**

```
@PersistenceContext  
EntityManager em;  
  
public void removeUser(int userId) {  
    User userToRemove = em.find(userId);  
    em.remove(userToRemove);  
}
```

## Le cycle de vie d'un entité bean



## Le cycle de vie d'un entité bean

Les annotations liées à la gestion du cycle de vie :

- ▶ **@PrePersist** ou **@PreRemove** sont invoquées sur un Entité Bean avant l'exécution des méthodes **persist()** et **remove()** de l'Entity Manager.

## Le cycle de vie d'un entité bean

Les annotations liées à la gestion du cycle de vie :

- ▶ **@PrePersist** ou **@PreRemove** sont invoquées sur un Entité Bean avant l'exécution des méthodes **persist()** et **remove()** de l'Entity Manager.
- ▶ **@PostPersist** ou **@PostRemove** sont invoquées sur un entité Bean après l'exécution des méthodes **persist()** et **remove()** de l'Entity Manager

## Le cycle de vie d'un entité bean

Les annotations liées à la gestion du cycle de vie :

- ▶ **@PrePersist** ou **@PreRemove** sont invoquées sur un Entité Bean avant l'exécution des méthodes **persist()** et **remove()** de l'Entity Manager.
- ▶ **@PostPersist** ou **@PostRemove** sont invoquées sur un entité Bean après l'exécution des méthodes **persist()** et **remove()** de l'Entity Manager
- ▶ **@PreUpdate** ou **@PostUpdate** sont invoquées sur un entité Bean respectivement avant ou après la mise à jour de la base de données.

## La persistance

- ▶ Les propriétés de bases de **persistence.xml** sont :
  - ▶ la valeur "**update**" de la propriété **"hibernate.hbm2ddl.auto"** indique que nous souhaitons qu'Hibernate crée la structure de données automatiquement et la mette à jour si nécessaire.
  - ▶ "**names**" est le nom que nous avons donné ici à l'unité de persistance, ce nom sera utilisé dans le bean session lors de la déclaration d'un *EntityManager*.
  - ▶ **java :/dsnames** est le nom de la Data source.
- ▶ Après le déploiement du bean entité, JBoss se charge de créer les tables dans son serveur de base de données si elles n'existaient pas.

## La persistance

L'unité de persistance est configurée dans le fichier META-INF/persistance.xml.

```
<?xml version="1.0" encoding="UTF-8"?>

```

## Exemple

### Name.java

```
package helloWorld;  
import javax.persistence.*;  
import java.io.Serializable;  
  
@Entity  
@Table(name="names")  
public class Name implements Serializable {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private int id;  
    @Column(unique = true)  
    private String name;  
  
    public Name () { this.name = ""; }  
    public Name (String name) { this.name = name; }  
    public int getId () { return this.id; }  
    public void setId (int id) { this.id = id; }  
    public String getName () { return this.name; }  
  
    ...
```

## Exemple

### Name.java

```
...
@Override
public boolean equals(Object o) {
    if (this == o)
        return true;
    if (o == null || getClass() != o.getClass())
        return false;
    Name that = (Name)o;
    if (this.name != null ? !this.name.equals(that.name) : that.name != null)
        return false;
    return true;
}
@Override
public int hashCode() {
    return this.name != null ? this.name.hashCode() : 0;
}
@Override
public String toString() {
    return this.name;
}
```

## Exemple

### PersistentHelloBean.java

```
package helloWorld;  
import javax.ejb.Stateless;  
import javax.persistence.*;  
import java.util.Collection;  
import java.util.Iterator;  
  
@Stateless  
public class PersistentHelloBean implements PersistentHelloLocal, PersistentHelloRemote  
{  
  
    @PersistenceContext(unitName="names")  
    EntityManager em;  
  
    public String sayHello(String s) {  
        String hello = "Hello " + s + " !";  
        Collection<Name> nameCollection;  
        em.persist(new Name(s));  
        nameCollection = em.createQuery("from Name n").getResultList();  
        hello += "\nListe des personnes déjà passées : ";  
        for(Iterator<Name>names=nameCollection.iterator(); names.hasNext(); ){  
            hello += " " + names.next().getName();  
        }  
        return hello;  
    }  
}
```

## Exercice

Nous souhaitons mettre en place une application de gestionnaire de vidéo. Chaque vidéo est représentée par son nom, sa taille et son emplacement.

- ▶ Donner la classe entité bean qui représente la vidéo.
- ▶ Proposer une classe qui permet de gérer l'ensemble des vidéos (création, ajout, suppression).