

# Architectures des SI - Les architectures 3-tiers

## Partie I : les entités

Informatique, Statistique et IA 4<sup>ème</sup> année

Olivier Caron<sup>1</sup>

<http://ocaron.polytech-lille.net>

<sup>1</sup>École d'ingénieurs Polytech Lille  
Université de Lille

8 janvier 2026



© Bjarne Stroustrup

J'ai toujours rêvé que mon ordinateur soit aussi simple que mon téléphone. Ce rêve est devenu réalité : je ne comprends plus comment utiliser mon téléphone.

# Les architectures logicielles de type 3-tiers

- Séparation d'une application en 3 niveaux :

# Les architectures logicielles de type 3-tiers

- Séparation d'une application en 3 niveaux :

## Niveau 1 : L'interface Homme-Machine (IHM)

S'occupe de la présentation des données, de la vérification des données saisies, des actions de l'utilisateur

# Les architectures logicielles de type 3-tiers

- Séparation d'une application en 3 niveaux :

## Niveau 1 : L'interface Homme-Machine (IHM)

S'occupe de la présentation des données, de la vérification des données saisies, des actions de l'utilisateur

## Niveau 2 : La logique métier

S'occupe du traitement applicatif, applique les règles de gestion

# Les architectures logicielles de type 3-tiers

- Séparation d'une application en 3 niveaux :

## Niveau 1 : L'interface Homme-Machine (IHM)

S'occupe de la présentation des données, de la vérification des données saisies, des actions de l'utilisateur

## Niveau 2 : La logique métier

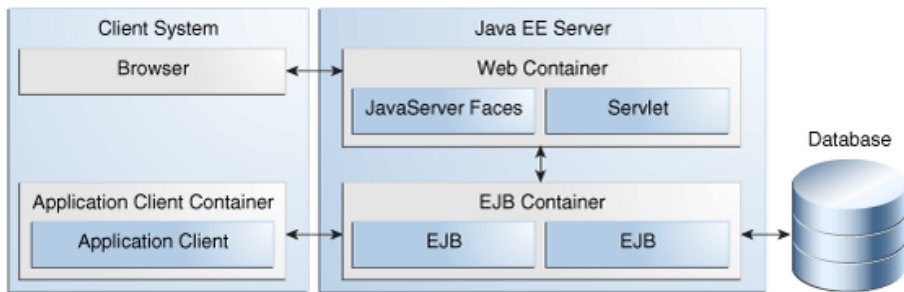
S'occupe du traitement applicatif, applique les règles de gestion

## Niveau 3 : Les données

S'occupe de la représentation des données persistantes.

# La vision Java des architectures logicielles de type 3-tiers

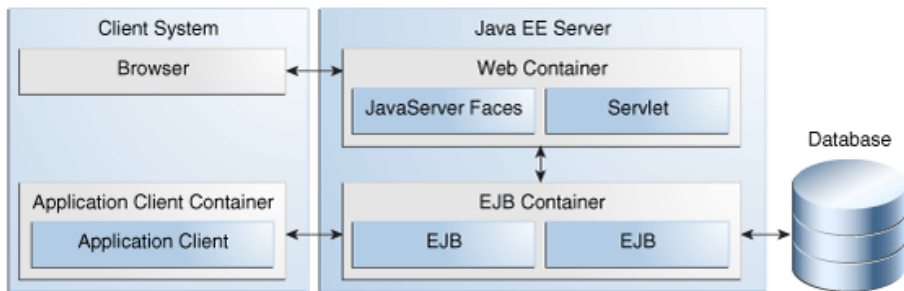
- **Java Edition Entreprise** : **norme** délivrée par Oracle Javasoftware puis par la fondation Eclipse.



(source image : Oracle Javasoftware)

# La vision Java des architectures logicielles de type 3-tiers

- Java Edition Entreprise : **norme** délivrée par Oracle Javasoftware puis par la fondation Eclipse.
- Notion de **serveur d'applications**



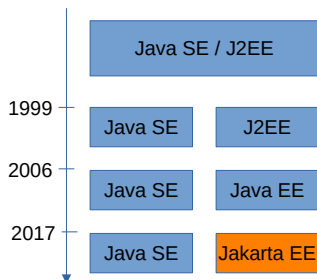
(source image : Oracle Javasoftware)

# Découpage des distributions Java au fil des ans

# Découpage des distributions Java au fil des ans

■ : droits détenus par Oracle

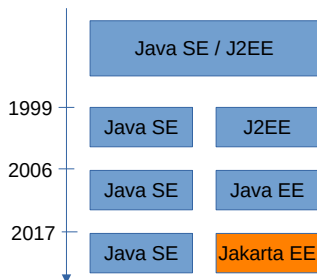
■ : droits détenus par Eclipse Foundation



# Découpage des distributions Java au fil des ans

■ : droits détenus par Oracle

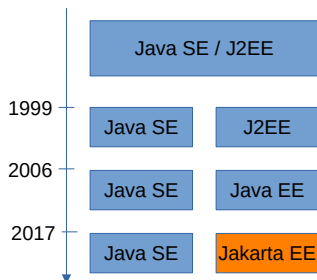
■ : droits détenus par Eclipse Foundation



# Découpage des distributions Java au fil des ans

 : droits détenus par Oracle

 : droits détenus par Eclipse Foundation



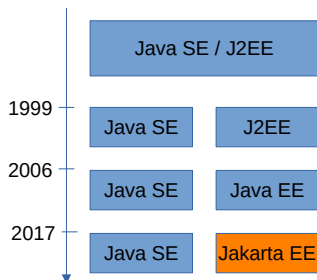
- Pour utiliser JPA, avant 2017 :

```
import javax.persistence.* ;
```

# Découpage des distributions Java au fil des ans

■ : droits détenus par Oracle

■ : droits détenus par Eclipse Foundation



- Pour utiliser JPA, avant 2017 :  

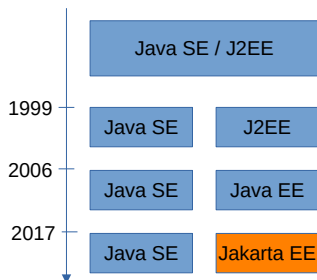
```
import javax.persistence.* ;
```
- A partir de 2017 :  

```
import jakarta.persistence.* ;
```

# Découpage des distributions Java au fil des ans

■ : droits détenus par Oracle

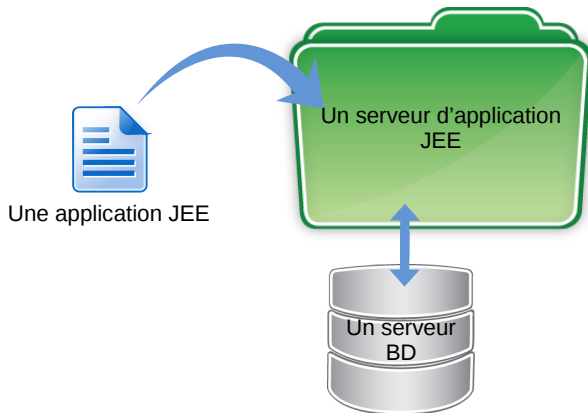
■ : droits détenus par Eclipse Foundation



- Pour utiliser JPA, avant 2017 :  
`import javax.persistence.* ;`
- A partir de 2017 :  
`import jakarta.persistence.* ;`
- La programmation ne change pas !

# JEE, c'est magique

- Une démonstration (ou un tour de magie ?) :



`http://localhost:8080/demoBookmarks/accueil`

# Une application JEE

- Une application JEE, c'est une archive Java qui contient :

# Une application JEE

- Une application JEE, c'est une archive Java qui contient :
  - ▶ Du code Java compilé pour tous les niveaux de l'architecture 3-tiers

# Une application JEE

- Une application JEE, c'est une archive Java qui contient :
  - ▶ Du code Java compilé pour tous les niveaux de l'architecture 3-tiers
  - ▶ Des informations pour que le code soit correctement identifié, configuré et déployé sur le serveur.

# Une application JEE

- Une application JEE, c'est une archive Java qui contient :
  - ▶ Du code Java compilé pour tous les niveaux de l'architecture 3-tiers
  - ▶ Des informations pour que le code soit correctement identifié, configuré et déployé sur le serveur.
- Pour que le serveur d'applications JEE comprenne ces informations, il faut donc que :

# Une application JEE

- Une application JEE, c'est une archive Java qui contient :
  - ▶ Du code Java compilé pour tous les niveaux de l'architecture 3-tiers
  - ▶ Des informations pour que le code soit correctement identifié, configuré et déployé sur le serveur.
- Pour que le serveur d'applications JEE comprenne ces informations, il faut donc que :
  - ▶ Ces informations soient **rigoureusement formulées** (code XML et/ou annotations Java 5).

# Une application JEE

- Une application JEE, c'est une archive Java qui contient :
  - ▶ Du code Java compilé pour tous les niveaux de l'architecture 3-tiers
  - ▶ Des informations pour que le code soit correctement identifié, configuré et déployé sur le serveur.
- Pour que le serveur d'applications JEE comprenne ces informations, il faut donc que :
  - ▶ Ces informations soient **rigoureusement formulées** (code XML et/ou annotations Java 5).
  - ▶ Ces informations soient **rigoureusement localisées**

# Une application JEE

- Une application JEE, c'est une archive Java qui contient :
  - ▶ Du code Java compilé pour tous les niveaux de l'architecture 3-tiers
  - ▶ Des informations pour que le code soit correctement identifié, configuré et déployé sur le serveur.
- Pour que le serveur d'applications JEE comprenne ces informations, il faut donc que :
  - ▶ Ces informations soient **rigoureusement formulées** (code XML et/ou annotations Java 5).
  - ▶ Ces informations soient **rigoureusement localisées**

## La norme JEE, c'est :

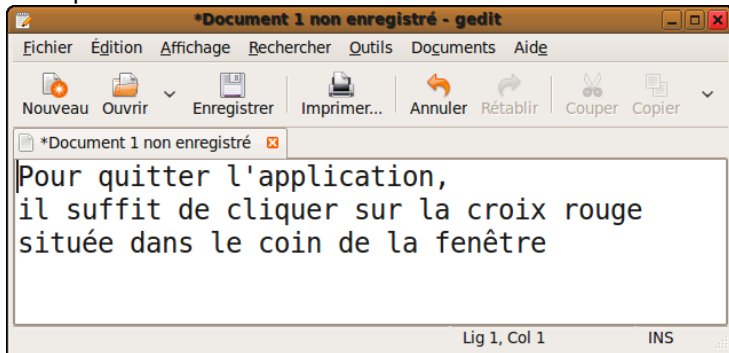
Une spécification complète qui précise où et comment stocker les composants de l'application dans l'archive, et comment les décrire.

# Un serveur JEE ne comprend pas les approximations

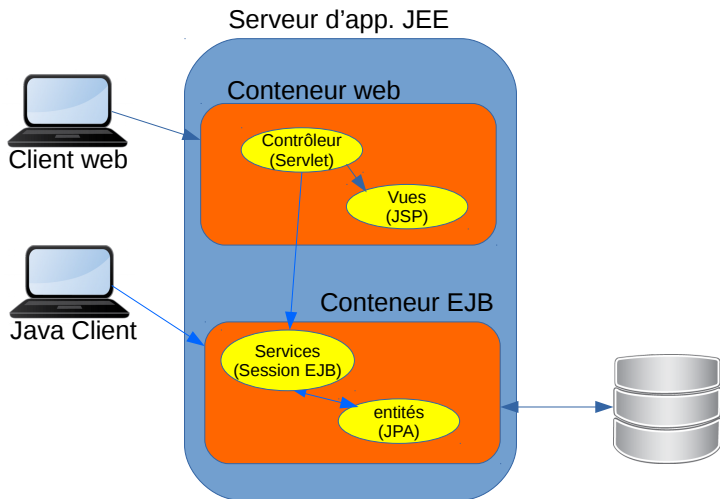
- Exemple 1 : Franquin à Peyo lors d'un repas au restaurant :  
*"Passe moi le schtroumpf"*

# Un serveur JEE ne comprend pas les approximations

- Exemple 1 : Franquin à Peyo lors d'un repas au restaurant :  
*"Passe moi le schtroumpf"*
- Exemple 2 :



## Zoom sur le conteneur EJB (1/2)



## Zoom sur le conteneur EJB (2/2)

- Le conteneur peut héberger des composants de différents types :

## Zoom sur le conteneur EJB (2/2)

- Le conteneur peut héberger des composants de différents types :
  - ▶ Les composants Session

## Zoom sur le conteneur EJB (2/2)

- Le conteneur peut héberger des composants de différents types :
  - ▶ Les composants Session
    - ★ Assurent les **services métiers**

## Zoom sur le conteneur EJB (2/2)

- Le conteneur peut héberger des composants de différents types :
  - ▶ Les composants Session
    - ★ Assurent les **services métiers**
    - ★ Interface avec les composants web ou clients

## Zoom sur le conteneur EJB (2/2)

- Le conteneur peut héberger des composants de différents types :
  - ▶ Les composants Session
    - ★ Assurent les **services métiers**
    - ★ Interface avec les composants web ou clients
  - ▶ Les composants Entités

## Zoom sur le conteneur EJB (2/2)

- Le conteneur peut héberger des composants de différents types :
  - ▶ Les composants Session
    - ★ Assurent les **services métiers**
    - ★ Interface avec les composants web ou clients
  - ▶ Les composants Entités
    - ★ Représentent les **objets métiers**

## Zoom sur le conteneur EJB (2/2)

- Le conteneur peut héberger des composants de différents types :
  - ▶ Les composants Session
    - ★ Assurent les **services métiers**
    - ★ Interface avec les composants web ou clients
  - ▶ Les composants Entités
    - ★ Représentent les **objets métiers**
    - ★ Interface avec les bases de données

## Zoom sur le conteneur EJB (2/2)

- Le conteneur peut héberger des composants de différents types :
  - ▶ Les composants Session
    - ★ Assurent les **services métiers**
    - ★ Interface avec les composants web ou clients
  - ▶ Les composants Entités
    - ★ Représentent les **objets métiers**
    - ★ Interface avec les bases de données
    - ★ Persistance gérée ou pas par le conteneur (CMP, BMP)

# Zoom sur le conteneur EJB (2/2)

- Le conteneur peut héberger des composants de différents types :
  - ▶ Les composants Session
    - ★ Assurent les **services métiers**
    - ★ Interface avec les composants web ou clients
  - ▶ Les composants Entités
    - ★ Représentent les **objets métiers**
    - ★ Interface avec les bases de données
    - ★ Persistance gérée ou pas par le conteneur (CMP, BMP)
  - ▶ Les composants "Message Driven"

# Zoom sur le conteneur EJB (2/2)

- Le conteneur peut héberger des composants de différents types :
  - ▶ Les composants Session
    - ★ Assurent les **services métiers**
    - ★ Interface avec les composants web ou clients
  - ▶ Les composants Entités
    - ★ Représentent les **objets métiers**
    - ★ Interface avec les bases de données
    - ★ Persistance gérée ou pas par le conteneur (CMP, BMP)
  - ▶ Les composants "Message Driven"
    - ★ permet la gestion asynchrone, événementielle

# Pourquoi des conteneurs? (1/3)

- Des objets trop complexes
-

# Pourquoi des conteneurs? (1/3)

- Des objets trop complexes
-

# Pourquoi des conteneurs ? (1/3)

- Des objets trop complexes

```
public class ObjetServeur {  
    private int x ;  
    public void setX(int value)  
  
        this.x=value ;  
  
}  
public int getX()  
  
    return this.x ;  
}  
public ObjetServeur()  
}
```

# Pourquoi des conteneurs ? (1/3)

- Des objets trop complexes

```
public class ObjetServeur extends UnicastRemoteObject {  
    private int x ;  
    public void setX(int value) throws RemoteException {  
  
        this.x=value ;  
  
    }  
    public int getX() throws RemoteException {  
  
        return this.x ;  
    }  
    public ObjetServeur() throws RemoteException { ... }  
}
```

---

Gestion du réseau

# Pourquoi des conteneurs ? (1/3)

- Des objets trop complexes

```
public class ObjetServeur extends UnicastRemoteObject {  
    private int x ;  
    public void setX(int value) throws RemoteException {  
  
        if (User.getAuthentication()...  
            this.x=value ;  
  
    }  
    public int getX() throws RemoteException {  
  
        if (User.getAuthentication()...  
  
        return this.x ;  
    }  
    public ObjetServeur() throws RemoteException { ... }  
}
```

---

Gestion du réseau

Gestion de la sécurité

# Pourquoi des conteneurs? (1/3)

- Des objets trop complexes

```
public class ObjetServeur extends UnicastRemoteObject {  
    private int x ;  
    public void setX(int value) throws RemoteException {  
  
        if (User.getAuthentication()...  
        this.x=value ;  
        // code JDBC : stockage de X  
        ...  
    }  
    public int getX() throws RemoteException {  
  
        if (User.getAuthentication()...  
        //code JDBC : restauration de X  
        ...  
        return this.x ;  
    }  
    public ObjetServeur() throws RemoteException { ... }  
}
```

---

Gestion du réseau  
Gestion de la sécurité  
Gestion de la persistance

# Pourquoi des conteneurs ? (1/3)

- Des objets trop complexes

```
public class ObjetServeur extends UnicastRemoteObject {
    private int x ;
    public void setX(int value) throws RemoteException {
        Tx.beginTransaction() ;
        if (User.getAuthentication()...
            this.x=value ;
        // code JDBC : stockage de X
        ...
        Tx.commitTransaction() ;
    }
    public int getX() throws RemoteException {
        Tx.beginTransaction() ;
        if (User.getAuthentication()...
        //code JDBC : restauration de X
        ...
        Tx.commitTransaction() ;
        return this.x ;
    }
    public ObjetServeur() throws RemoteException { ... }
}
```

---

Gestion du réseau  
Gestion de la sécurité  
Gestion de la persistance  
Gestion des transactions

# Pourquoi des conteneurs ? (1/3)

- Des objets trop complexes

```
public class ObjetServeur extends UnicastRemoteObject {  
    private int x ;  
    public void setX(int value) throws RemoteException {  
        Tx.beginTransaction() ;  
        if (User.getAuthentication()...  
        this.x=value ;  
        // code JDBC : stockage de X  
        ...  
        Tx.commitTransaction() ;  
    }  
    public int getX() throws RemoteException {  
        Tx.beginTransaction() ;  
        if (User.getAuthentication()...  
        //code JDBC : restauration de X  
        ...  
        Tx.commitTransaction() ;  
        return this.x ;  
    }  
    public ObjetServeur() throws RemoteException { ... }  
}
```

---

Gestion du réseau  
Gestion de la sécurité  
Gestion de la persistance  
Gestion des transactions

## Pourquoi des conteneurs ? (2/3)

- 1ère étape : une meilleure structuration objet

---

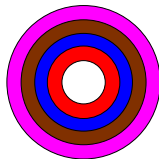
## Pourquoi des conteneurs ? (2/3)

- 1ère étape : une meilleure structuration objet

---

# Pourquoi des conteneurs? (2/3)

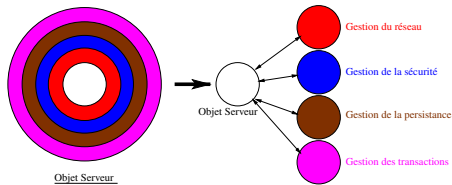
- 1ère étape : une meilleure structuration objet



Objet Serveur

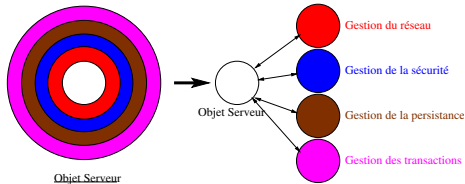
# Pourquoi des conteneurs? (2/3)

- 1ère étape : une meilleure structuration objet



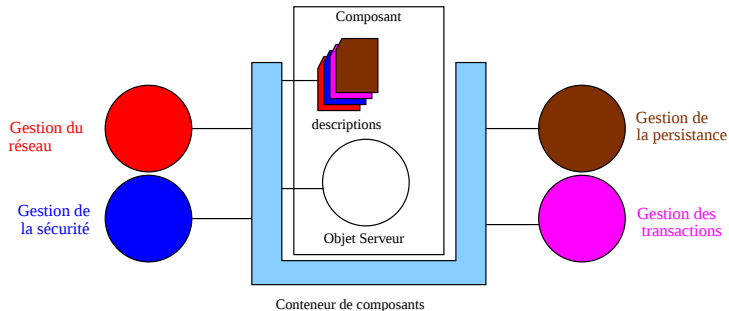
# Pourquoi des conteneurs? (2/3)

- 1ère étape : une meilleure structuration objet



# Pourquoi des conteneurs ? (3/3)

- 2nde étape : complète séparation, notion de **conteneur**



# Interaction code métier Java - Conteneur JEE

- Les informations à fournir au conteneur pour que le code métier soit correctement déployé sont définies soit par :

# Interaction code métier Java - Conteneur JEE

- Les informations à fournir au conteneur pour que le code métier soit correctement déployé sont définies soit par :  
Des descripteurs XML (tendance : ⬇)

# Interaction code métier Java - Conteneur JEE

- Les informations à fournir au conteneur pour que le code métier soit correctement déployé sont définies soit par :

Des descripteurs XML (tendance : ⬇)

- ▶ Avantage : stricte séparation code métier Java - informations

# Interaction code métier Java - Conteneur JEE

- Les informations à fournir au conteneur pour que le code métier soit correctement déployé sont définies soit par :

Des descripteurs XML (tendance : ⬇)

- ▶ Avantage : stricte séparation code métier Java - informations
- ▶ Inconvénient : verbeux

# Interaction code métier Java - Conteneur JEE

- Les informations à fournir au conteneur pour que le code métier soit correctement déployé sont définies soit par :

Des descripteurs XML (tendance : ⬇)

- ▶ Avantage : stricte séparation code métier Java - informations
- ▶ Inconvénient : verbeux

Des annotations Java (tendance : ⬆)

# Interaction code métier Java - Conteneur JEE

- Les informations à fournir au conteneur pour que le code métier soit correctement déployé sont définies soit par :

Des descripteurs XML (tendance : ⬇)

- ▶ Avantage : stricte séparation code métier Java - informations
- ▶ Inconvénient : verbeux

Des annotations Java (tendance : ⬆)

- ▶ Avantage : codage très rapide (annotations par défaut)

# Interaction code métier Java - Conteneur JEE

- Les informations à fournir au conteneur pour que le code métier soit correctement déployé sont définies soit par :

## Des descripteurs XML (tendance : ⬇)

- ▶ Avantage : stricte séparation code métier Java - informations
- ▶ Inconvénient : verbeux

## Des annotations Java (tendance : ⬆)

- ▶ Avantage : codage très rapide (annotations par défaut)
- ▶ Inconvénient : nécessite une recompilation des classes

# Les annotations Java

- Introduite à partir de Java 5
- Permet de spécifier des **métadonnées**
- Les annotations peuvent être analysées par introspection
- Certaines annotations standards sont exploitées lors de la compilation :

```
public class Foo1 extends Foo {  
    ...  
    @Override  
    public void bar() ;  
}
```

La compilation provoque une erreur si la méthode bar n'existe pas dans l'une des sur-classes de Foo1.

# Définition des annotations (par l'exemple)

```
package fr.polytech.lille.is ;

import java.lang.annotation.ElementType;
import java.lang.annotation.Target;
import java.lang.annotation.RetentionPolicy ;
import java.lang.annotation.Retention ;

// remarque: annotation conservée dans le code source et byte code:
@Retention(RetentionPolicy.RUNTIME)

// cette annotation est uniquement applicable aux méthodes
@Target({ElementType.METHOD})
public @interface Developpeur {
    String createur();
    String [] autresContributeurs() default {}; // optionnel
    String dateDerniereModif();
}
```

# Utilisation des annotations (par l'exemple)

```
import fr.polytech.lille.is.Developpeur ;

public class Test {
    ...
    @Developpeur( createur="john", dateDerniereModif="27-03-2017")
    public void process() {...}

    @Developpeur( createur="laura",
                  autresContributeurs={"james", "john"},
                  dateDerniereModif="25-03-2017")
    public void anotherProcess() { ...}
}
```

# Processus de développement des EJB entités (ou JPA)

## Conception UML



## Conception UML - EJB



## Traduction Java



Adaptation BD (facultatif)

XML



Archivage (packaging)

déploiement

6



BD



Serveur JEE

# Étape 1 : conception UML des objets métiers (Entity)

- Sources Java disponibles sur le serveur GitLab de l'université<sup>1</sup> :



```
git clone https://gitlab.univ-lille.fr/olivier.caron/democoursal.git
```

---

1. Le serveur de l'université nécessite de se connecter (login : `prenom.nom.etu`) puis de fixer un mot de passe.

# Étape 1 : conception UML des objets métiers (Entity)

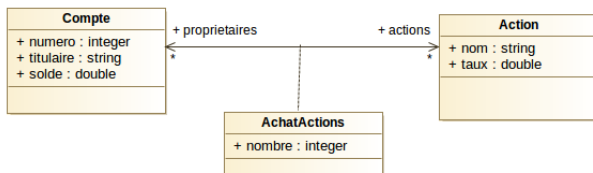
- Sources Java disponibles sur le serveur GitLab de l'université <sup>1</sup> :



`git clone https://gitlab.univ-lille.fr/olivier.caron/democoursal.git`

- Durant cette phase, on se focalise sur les **données** métiers (classes, attributs, associations) et pas sur les **traitements** (méthodes).

Figure – La modélisation UML des entités



1. Le serveur de l'université nécessite de se connecter (login : `prenom.nom.etu`) puis de fixer un mot de passe.

# Étape 2 : conception UML-EJB

## Conception UML



## Conception UML - EJB



## Traduction Java



Adaptation BD  
(facultatif)

XML



Archivage  
(packaging)

déploiement



BD



Serveur JEE

## Étape 2 : conception UML-**EJB** des objets métiers (Entity) (2/2)

- Modélisation UML non directement exploitable pour EJB :

## Étape 2 : conception UML-**EJB** des objets métiers (Entity) (2/2)

- Modélisation UML non directement exploitable pour EJB :
  - ▶ Le modèle EJB ne gère que l'héritage simple

## Étape 2 : conception UML-**EJB** des objets métiers (Entity) (2/2)

- Modélisation UML non directement exploitable pour EJB :
  - ▶ Le modèle EJB ne gère que l'héritage simple
  - ▶ Le modèle EJB ne gère **que** des associations binaires

## Étape 2 : conception UML-**EJB** des objets métiers (Entity) (2/2)

- Modélisation UML non directement exploitable pour EJB :
  - ▶ Le modèle EJB ne gère que l'héritage simple
  - ▶ Le modèle EJB ne gère **que** des associations binaires
  - ▶ Le modèle EJB ne gère pas les classes-associations (propriétés des associations)

## Étape 2 : conception UML-**EJB** des objets métiers (Entity) (2/2)

- Modélisation UML non directement exploitable pour EJB :
  - ▶ Le modèle EJB ne gère que l'héritage simple
  - ▶ Le modèle EJB ne gère **que** des associations binaires
  - ▶ Le modèle EJB ne gère pas les classes-associations (propriétés des associations)
  - ▶ Les entités doivent posséder un identifiant

## Étape 2 : conception UML-**EJB** des objets métiers (Entity) (1/2)

- Transformation du schéma UML pour devenir compatible EJB :

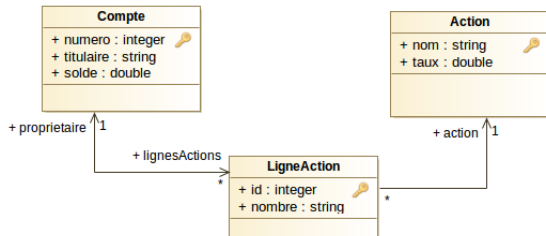
## Étape 2 : conception UML-**EJB** des objets métiers (Entity) (1/2)

- Transformation du schéma UML pour devenir compatible EJB :
  - ▶ Ajout d'entité(s) pour se substituer aux classes n-aires ( $n > 2$ ) ou aux classes-associations

## Étape 2 : conception UML-**EJB** des objets métiers (Entity) (1/2)

- Transformation du schéma UML pour devenir compatible EJB :
  - ▶ Ajout d'entité(s) pour se substituer aux classes n-aires ( $n > 2$ ) ou aux classes-associations
  - ▶ Utilisation d'un stéréotype UML `id` pour définir des identifiants pour chaque entité.

Figure – La modélisation UML-**EJB** des entités



# Étape 3 : traduction Java

## Conception UML



## Conception UML - EJB



## Traduction Java



Adaptation BD  
(facultatif)

XML



Archivage  
(packaging)

déploiement

6



BD



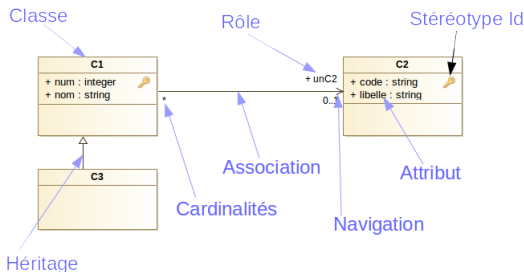
Serveur JEE

## Étape 3 : traduction Java

- Règles de traduction (**mapping**) entre concepts UML et concepts Java :

## Étape 3 : traduction Java

- Règles de traduction (**mapping**) entre concepts UML et concepts Java :
- Les concepts UML :



## Étape 3 : traduction Java

- Traduction des classes :
  - ▶ Une classe UML devient une classe Java annotée par `@Entity`
  - ▶ Fonction constructeur sans paramètre **obligatoire** (pour introspection)
  - ▶ Classe sérializable (propagation réseau)
- Traduction de l'héritage UML via `extends`

```
package demo ;

import jakarta.persistence.Entity ;
import java.io.Serializable ;

@Entity public class C1
    implements Serializable {
    ...
    public C1() {...}
    ...
}
```

```
package demo ;

import jakarta.persistence.Entity ;

@Entity
public class C3 extends C1 {
    ...
    public C3() {...}
    ...
}
```

## Étape 3 : traduction Java des attributs et identifiants

- Annotation @Id pour désigner la clé primaire
- Annotation @GeneratedValue pour générer automatiquement des valeurs entières de clés.
- La norme JPA 2.0 autorise deux modes de traduction :
  - ❶ Via les variables d'instances (**celle qu'on utilisera**)
  - ❷ Via les fonctions accesseurs (getter/setter)

```
package demo ; /* méthode 1 */
import jakarta.persistence.* ;

@Entity public class C1
    implements java.io.Serializable {

    @Id private int num ;
    private String nom ; ...
}
```

```
package demo ; /* méthode 2 */
import jakarta.persistence.* ;

@Entity public class C1
    implements java.io.Serializable {
    @Id public int getNum() { ... }
    public void setNum(int v) { ... }

    public String getNom() { ... }
    public void setNom(String v) { ... }
}
```

## Étape 3 : traduction Java des attributs

- **Quelques** types d'attributs possibles autorisés sont :

## Étape 3 : traduction Java des attributs

- **Quelques** types d'attributs possibles autorisés sont :
  - ▶ Les types primitifs : int, double, boolean, char, etc

## Étape 3 : traduction Java des attributs

- **Quelques** types d'attributs possibles autorisés sont :
  - ▶ Les types primitifs : int, double, boolean, char, etc
  - ▶ Les "wrappers" des types primitifs (Integer, Double, etc)

## Étape 3 : traduction Java des attributs

- **Quelques** types d'attributs possibles autorisés sont :
  - ▶ Les types primitifs : int, double, boolean, char, etc
  - ▶ Les "wrappers" des types primitifs (Integer, Double, etc)
  - ▶ Les types objets : `java.lang.String`, `java.math.BigInteger`, `java.math.BigDecimal`, `java.util.Date`, `java.util.Calendar`, `java.sql.Date`, `java.sql.Time` et `java.sql.Timestamp`.

## Étape 3 : traduction Java des attributs

- **Quelques** types d'attributs possibles autorisés sont :
  - ▶ Les types primitifs : int, double, boolean, char, etc
  - ▶ Les "wrappers" des types primitifs (Integer, Double, etc)
  - ▶ Les types objets : `java.lang.String`, `java.math.BigInteger`, `java.math.BigDecimal`, `java.util.Date`, `java.util.Calendar`, `java.sql.Date`, `java.sql.Time` et `java.sql.Timestamp`.
  - ▶ Des classes "sérialisables" définies par l'utilisateur

## Étape 3 : traduction Java des attributs

- **Quelques** types d'attributs possibles autorisés sont :
  - ▶ Les types primitifs : int, double, boolean, char, etc
  - ▶ Les "wrappers" des types primitifs (Integer, Double, etc)
  - ▶ Les types objets : java.lang.String, java.math.BigInteger, java.math.BigDecimal, java.util.Date, java.util.Calendar, java.sql.Date, java.sql.Time et java.sql.Timestamp.
  - ▶ Des classes "sérialisables" définies par l'utilisateur
  - ▶ Les types byte[], Byte[], char [], Character []

## Étape 3 : traduction Java des attributs

- **Quelques** types d'attributs possibles autorisés sont :
  - ▶ Les types primitifs : `int`, `double`, `boolean`, `char`, etc
  - ▶ Les "wrappers" des types primitifs (`Integer`, `Double`, etc)
  - ▶ Les types objets : `java.lang.String`, `java.math.BigInteger`, `java.math.BigDecimal`, `java.util.Date`, `java.util.Calendar`, `java.sql.Date`, `java.sql.Time` et `java.sql.Timestamp`.
  - ▶ Des classes "sérialisables" définies par l'utilisateur
  - ▶ Les types `byte[]`, `Byte[]`, `char []`, `Character []`
- Utilisation de l'annotation `@Transient` pour indiquer que la variable d'instance ne doit pas être une donnée persistante.

## Étape 3 : traduction Java des associations

- Lien navigable → une variable d'instance (nom du rôle)

## Étape 3 : traduction Java des associations

- Lien navigable → une variable d'instance (nom du rôle)
- Lien non navigable → rien

## Étape 3 : traduction Java des associations

- Lien navigable → une variable d'instance (nom du rôle)
- Lien non navigable → rien
- Cardinalité max  $> 1$  → variable de type Collection, Set, List ou Map

## Étape 3 : traduction Java des associations

- Lien navigable → une variable d'instance (nom du rôle)
- Lien non navigable → rien
- Cardinalité max  $> 1$  → variable de type `Collection`, `Set`, `List` ou `Map`
- Ajout annotation Java pour les cardinalités de l'association : `ManyToOne`, `ManyToMany`, `OneToMany`, `OneToOne`.

## Étape 3 : traduction Java des associations

- Lien navigable → une variable d'instance (nom du rôle)
- Lien non navigable → rien
- Cardinalité max  $> 1$  → variable de type `Collection`, `Set`, `List` ou `Map`
- Ajout annotation Java pour les cardinalités de l'association : `ManyToOne`, `ManyToMany`, `OneToMany`, `OneToOne`.
- Attribut `mappedBy` de l'annotation pour les associations bi-directionnelles (les deux liens navigables).

## Étape 3 : traduction Java des associations

- Lien navigable → une variable d'instance (nom du rôle)
- Lien non navigable → rien
- Cardinalité max  $> 1$  → variable de type Collection, Set, List ou Map
- Ajout annotation Java pour les cardinalités de l'association : ManyToOne, ManyToMany, OneToMany, OneToOne.
- Attribut mappedBy de l'annotation pour les associations bi-directionnelles (les deux liens navigables).

Mieux comprendre avec

<http://ocaron.polytech-lille.net/enseignement/polyJPA.pdf>

## Étape 3 : traduction Java des associations, illustration <sup>2</sup>

```
package ejb.entites;  
import jakarta.persistence.*  
  
@Entity public class LigneAction  
    implements java.io.Serializable {  
  
    @Id @GeneratedValue  
    private int id ;  
    private int nombre ;  
    @ManyToOne  
    private Compte proprietaire ;  
    @ManyToOne(fetch=FetchType.EAGER)  
    private Action action ;  
    ...  
}
```

```
package ejb.entites;  
import java.io.Serializable ;  
import java.util.Set;  
import jakarta.persistence.*  
  
@Entity public class Compte  
    implements Serializable {  
    @Id private int numero ;  
    private double solde ;  
    private String titulaire ;  
    @OneToMany(mappedBy="proprietaire",  
                fetch=FetchType.EAGER)  
    private Set<LigneAction>  
        lignesactions ;  
    ...  
}
```

2. FetchType.EAGER permet de charger automatiquement en mémoire les objets reliés par association

# Étape 4 : Adaptation BD (Base de Données)

## Conception UML



## Conception UML - EJB



## Traduction Java



Adaptation BD (facultatif)

XML



Archivage (packaging)

déploiement

6



BD



Serveur JEE

## Étape 4 : Adaptation BD (Base de Données), problématique

- Les entités Java représentent des objets issus d'une base de données.

## Étape 4 : Adaptation BD (Base de Données), problématique

- Les entités Java représentent des objets issus d'une base de données.
- Il existe un mapping (traduction) **par défaut** pour chaque concept.  
Ex : une entité Java de nom `Action` représente une table de nom `Action`.

## Étape 4 : Adaptation BD (Base de Données), problématique

- Les entités Java représentent des objets issus d'une base de données.
- Il existe un mapping (traduction) **par défaut** pour chaque concept.  
Ex : une entité Java de nom `Action` représente une table de nom `Action`.
- Aucun problème dans le cas de création d'une base de données (création automatique de la base) ➔ étape 4 optionnelle.

## Étape 4 : Adaptation BD (Base de Données), problématique

- Les entités Java représentent des objets issus d'une base de données.
- Il existe un mapping (traduction) **par défaut** pour chaque concept.  
Ex : une entité Java de nom `Action` représente une table de nom `Action`.
- Aucun problème dans le cas de création d'une base de données (création automatique de la base) ➔ étape 4 optionnelle.
- Que faire lorsqu'on dispose déjà d'une base et qu'on désire **réutiliser** du code EJB ?

## Étape 4 : Adaptation BD (Base de Données), problématique

- Les entités Java représentent des objets issus d'une base de données.
- Il existe un mapping (traduction) **par défaut** pour chaque concept.  
Ex : une entité Java de nom `Action` représente une table de nom `Action`.
- Aucun problème dans le cas de création d'une base de données (création automatique de la base) ➔ étape 4 optionnelle.
- Que faire lorsqu'on dispose déjà d'une base et qu'on désire **réutiliser** du code EJB ?
  - 👉 Modifier le code Java (plus vraiment de la réutilisation de code)

## Étape 4 : Adaptation BD (Base de Données), problématique

- Les entités Java représentent des objets issus d'une base de données.
- Il existe un mapping (traduction) **par défaut** pour chaque concept.  
Ex : une entité Java de nom `Action` représente une table de nom `Action`.
- Aucun problème dans le cas de création d'une base de données (création automatique de la base) ➔ étape 4 optionnelle.
- Que faire lorsqu'on dispose déjà d'une base et qu'on désire **réutiliser** du code EJB ?
  - 👎 Modifier le code Java (plus vraiment de la réutilisation de code)
  - 👍 Disposer d'autres règles de mapping

## Étape 4 : Adaptation BD (Base de Données), problématique

- Les entités Java représentent des objets issus d'une base de données.
- Il existe un mapping (traduction) **par défaut** pour chaque concept.  
Ex : une entité Java de nom `Action` représente une table de nom `Action`.
- Aucun problème dans le cas de création d'une base de données (création automatique de la base) ➔ étape 4 optionnelle.
- Que faire lorsqu'on dispose déjà d'une base et qu'on désire **réutiliser** du code EJB ?
  - 👎 Modifier le code Java (plus vraiment de la réutilisation de code)
  - 👍 Disposer d'autres règles de mapping
- Solution JEE : jeu d'annotations Java pour modifier les règles par défaut

## Étape 4 : Adaptation BD des classes

- **Règle par défaut** : le nom de la classe entité correspond au nom de la table.
- L'annotation `Table` permet de modifier le nom de table par défaut.

```
package ejb.entites;  
import jakarta.persistence.*  
  
@Table(name="t_ligne_action")  
@Entity public class LigneAction  
    implements java.io.Serializable {  
  
    ...  
}
```

## Étape 4 : Adaptation BD des attributs

- **Règle 1 par défaut** : le nom de l'attribut correspond au nom de la colonne.
- **Règle 2 par défaut** : un attribut de type String correspond à une colonne de type char(255).
- L'attribut `@Column.name` permet de modifier le nom de la colonne par défaut.
- L'attribut `@Column.length` permet de modifier la taille du type char de la colonne par défaut.

```
...
@Entity public class Compte
implements Serializable {
    @Id @Column(name="num_compte") private int numero ;
    private double solde ;
    @Column(name="client", length=50)
    private String titulaire ;
}
```

## Étape 4 : Adaptation BD des associations

- **Règle par défaut** : dans une classe, un attribut Java de nom `r1` de type d'une classe `C` correspond à une colonne clé étrangère `r1_id`, `id` étant le nom de la colonne clé primaire de la table associée à `C`.
- L'attribut `@JoinColumn.name` permet de modifier le nom de la colonne par défaut.

```
...
@Entity public class LigneAction
    implements Serializable {
    ...
    @ManyToOne
    @JoinColumn(name="ref_compte") // "proprietaire_numero" -> "ref_compte"
    private Compte proprietaire ;
    @JoinColumn(name="ref_action") // "action_nom" -> "ref_action"
    @ManyToOne(fetch=FetchType.EAGER)
    private Action action ;
}
```

## Étape 4 : Adaptation BD de l'héritage de classes

- L'héritage Java entre EJB entités est possible.

## Étape 4 : Adaptation BD de l'héritage de classes

- L'héritage Java entre EJB entités est possible.
- La spécification EJB propose trois mappings :

## Étape 4 : Adaptation BD de l'héritage de classes

- L'héritage Java entre EJB entités est possible.
- La spécification EJB propose trois mappings :
  - ▶ Une table pour une hiérarchie de classe (**Règle par défaut**)

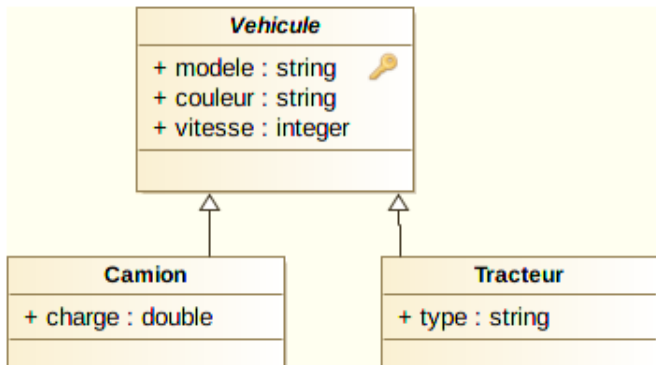
## Étape 4 : Adaptation BD de l'héritage de classes

- L'héritage Java entre EJB entités est possible.
- La spécification EJB propose trois mappings :
  - ▶ Une table pour une hiérarchie de classe (**Règle par défaut**)
  - ▶ Une table par classe concrète

## Étape 4 : Adaptation BD de l'héritage de classes

- L'héritage Java entre EJB entités est possible.
- La spécification EJB propose trois mappings :
  - ▶ Une table pour une hiérarchie de classe (**Règle par défaut**)
  - ▶ Une table par classe concrète
  - ▶ Jonctions de tables

## Étape 4 : Adaptation BD de l'héritage de classes, exemple



## Étape 4 : traduction héritage, option une table par hiérarchie (1/4)

- C'est l'option par défaut sans annotations supplémentaires

## Étape 4 : traduction héritage, option une table par hiérarchie (1/4)

- C'est l'option par défaut sans annotations supplémentaires
- Une seule table pour toute la hiérarchie de classes

## Étape 4 : traduction héritage, option une table par hiérarchie (1/4)

- C'est l'option par défaut sans annotations supplémentaires
- Une seule table pour toute la hiérarchie de classes
- Une colonne permet de distinguer le type de l'instance (**Règle par défaut** : colonne de nom "dtype" de type chaîne de caractères)

## Étape 4 : traduction héritage, option une table par hiérarchie (1/4)

- C'est l'option par défaut sans annotations supplémentaires
- Une seule table pour toute la hiérarchie de classes
- Une colonne permet de distinguer le type de l'instance (**Règle par défaut** : colonne de nom "dtype" de type chaîne de caractères)
- Chaque classe fixe une valeur pour cette colonne (**Règle par défaut** : nom de l'entité)

## Étape 4 : traduction héritage, option une table par hiérarchie (2/4)

```
import jakarta.persistence.DiscriminatorColumn;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Inheritance;
import jakarta.persistence.InheritanceType ;

@Entity
@DiscriminatorColumn(name="record_type")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE,
discriminatorValue="vehicule")
public class Vehicule implements java.io.Serializable {

    @Id private String modele ;

    ...
}
```

## Étape 4 : traduction héritage, option une table par hiérarchie (3/4)

```
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Inheritance;

@Entity
@Inheritance(discriminatorValue="camion")
public class Camion extends Vehicule {
    private float charge ;

    public Camion() {}
    ...
}
```

## Étape 4 : traduction héritage, option une table par hiérarchie (4/4)

- Le mapping BD correspondant :

| Table "public.vehicule" |                        |               |
|-------------------------|------------------------|---------------|
| Colonne                 | Type                   | Modificateurs |
| record_type             | character varying(10)  | not null      |
| modele                  | character varying(255) | not null      |
| couleur                 | character varying(255) |               |
| vitesse                 | integer                | not null      |
| charge                  | double precision       |               |
| type                    | character varying(255) |               |

Index : "vehicule\_pkey" PRIMARY KEY, btree (modele)

## Étape 4 : traduction héritage, option une table par classe (1/4)

- Duplication des propriétés héritées

## Étape 4 : traduction héritage, option une table par classe (1/4)

- Duplication des propriétés héritées
- Nécessite d'utiliser l'opérateur algébrique `union` pour collecter les instances de la hiérarchie.

## Étape 4 : traduction héritage, option une table par classe (2/4)

```
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Inheritance;
import jakarta.persistence.InheritanceType ;

@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Vehicule implements java.io.Serializable {
    ...
}
```

## Étape 4 : traduction héritage, option une table par classe (3/4)

```
import jakarta.persistence.Entity;  
  
@Entity  
public class Camion extends Vehicule {  
    private float charge ;  
    ...  
}
```

## Étape 4 : traduction héritage, option une table par classe (4/4)

- Le mapping BD correspondant :

| Table "public.camion" |                        |               |
|-----------------------|------------------------|---------------|
| Colonne               | Type                   | Modificateurs |
| -----+-----+-----     |                        |               |
| modele                | character varying(255) | not null      |
| couleur               | character varying(255) |               |
| vitesse               | integer                | not null      |
| charge                | double precision       | not null      |

Index : "camion\_pkey" PRIMARY KEY, btree (modele)

## Étape 4 : traduction héritage, option jonction de tables (1/4)

- Une table par classe mais pas de duplication de propriétés

## Étape 4 : traduction héritage, option jonction de tables (1/4)

- Une table par classe mais pas de duplication de propriétés
- Ajout d'une colonne pour relier les tables

## Étape 4 : traduction héritage, option jonction de tables (1/4)

- Une table par classe mais pas de duplication de propriétés
- Ajout d'une colonne pour relier les tables
- Nécessite d'utiliser l'opérateur algébrique de jointure pour collecter les instances de la hiérarchie.

## Étape 4 : traduction héritage, option jonction de tables (2/4)

```
import jakarta.persistence.Entity ;
import jakarta.persistence.Id ;
import jakarta.persistence.Inheritance ;
import jakarta.persistence.InheritanceType ;

@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Vehicule implements java.io.Serializable {
    @Id private String modele ;
    private String couleur ;
    private int vitesse ;
    ...
}
```

## Étape 4 : traduction héritage, option jonction de tables (3/4)

```
import jakarta.persistence.Entity;
import jakarta.persistence.PrimaryKeyJoinColumn;

@Entity
@PrimaryKeyJoinColumn(name="ref_vehicule")
public class Camion extends Vehicule {
    private float charge ;

    public Camion() {}
    ...
}
```

## Étape 4 : traduction héritage, option jonction de tables (4/4)

- Le mapping BD correspondant :

Table "public.camion"

| Colonne      | Type                   | Modificateurs |
|--------------|------------------------|---------------|
| ref_vehicule | character varying(255) | not null      |
| charge       | double precision       | not null      |

Index : "camion\_pkey" PRIMARY KEY, btree (ref\_vehicule)

Contraintes de clés secondaires :

"fkce0ca1498abbb75c" FOREIGN KEY (ref\_vehicule)

REFERENCES vehicule(modele)

# Étape 5 : Archivage

## Conception UML



## Conception UML - EJB



## Traduction Java



Adaptation BD  
(facultatif)

XML



Archivage  
(packaging)

déploiement

6



BD



Serveur JEE

## Étape 5 : Archivage

- Une archive des classes Java compilées ne suffit pas.

## Étape 5 : Archivage

- Une archive des classes Java compilées ne suffit pas.
- Dans quelle base de données, le serveur JEE va associer les entités de l'archive ?

## Étape 5 : Archivage

- Une archive des classes Java compilées ne suffit pas.
- Dans quelle base de données, le serveur JEE va associer les entités de l'archive ?
- Quel va être le comportement du serveur lors du déploiement/retrait de l'archive ?

## Étape 5 : Archivage

- Une archive des classes Java compilées ne suffit pas.
- Dans quelle base de données, le serveur JEE va associer les entités de l'archive ?
- Quel va être le comportement du serveur lors du déploiement/retrait de l'archive ?
- Solution JEE :

## Étape 5 : Archivage

- Une archive des classes Java compilées ne suffit pas.
- Dans quelle base de données, le serveur JEE va associer les entités de l'archive ?
- Quel va être le comportement du serveur lors du déploiement/retrait de l'archive ?
- Solution JEE :
  - ▶ Un descripteur XML `persistence.xml` va spécifier ces informations.

## Étape 5 : Archivage

- Une archive des classes Java compilées ne suffit pas.
- Dans quelle base de données, le serveur JEE va associer les entités de l'archive ?
- Quel va être le comportement du serveur lors du déploiement/retrait de l'archive ?
- Solution JEE :
  - ▶ Un descripteur XML `persistence.xml` va spécifier ces informations.
  - ▶ Ce descripteur se trouve dans le répertoire `META-INF` situé à la racine de l'archive.

## Étape 5 : Archivage, l'outil jar

- Implémentation Java structure ZIP

## Étape 5 : Archivage, l'outil jar

- Implémentation Java structure ZIP
- Java donc multi-plate-forme

## Étape 5 : Archivage, l'outil jar

- Implémentation Java structure ZIP
- Java donc multi-plate-forme
- exemples utilisation commande :  
`jar cvf mesComposants.jar mesComposants/`  
`jar tvf mesComposants.jar`  
`jar xvf mesComposants.jar`

## Étape 5 : Archivage, l'outil jar

- Implémentation Java structure ZIP
- Java donc multi-plate-forme
- exemples utilisation commande :  

```
jar cvf mesComposants.jar mesComposants/
```

```
jar tvf mesComposants.jar
```

```
jar xvf mesComposants.jar
```
- Utilisé pour composants webs et ejb, java et applications JEE

# Archivage des composants entités

## Fichier persistence.xml :

```
mesEntites/  
META-INF/  
  persistence.xml  
ejb/  
  entites/  
    Compte.class  
    Action.class  
    LigneAction.class  
jar cvf entites.jar  
  mesEntites/*
```

```
1 <persistence xmlns="https://jakarta.ee/xml/ns/persistence "  
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "  
3   xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence  
4   https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd "  
5   version="3.0">  
6   <persistence-unit name="appliBanque">  
7     <jta-data-source>java:/PostgresDS</jta-data-source>  
8     <properties>  
9       <property name="hibernate.hbm2ddl.auto "  
10        value="create-drop"/>  
11     </properties>  
12   </persistence-unit>  
13 </persistence>
```

Ligne 6 : définition d'une **unité de persistance**

Ligne 7 : unité associée à une source BD (**spécifiée au niveau du serveur**)

Lignes 9-10 : options de déploiement (ici création/suppression des tables)

## Étape 6 : Déploiement

- Chaque serveur a son protocole de déploiement.

## Étape 6 : Déploiement

- Chaque serveur a son protocole de déploiement.
- Cas du serveur Wildfly/JBoss :

## Étape 6 : Déploiement

- Chaque serveur a son protocole de déploiement.
- Cas du serveur Wildfly/JBoss :
  - ▶ Un répertoire est constamment scruté par le serveur

## Étape 6 : Déploiement

- Chaque serveur a son protocole de déploiement.
- Cas du serveur Wildfly/JBoss :
  - ▶ Un répertoire est constamment scruté par le serveur
  - ▶ Opération de déploiement : copier l'archive dans ce répertoire

# Étape 6 : Déploiement

- Chaque serveur a son protocole de déploiement.
- Cas du serveur Wildfly/JBoss :
  - ▶ Un répertoire est constamment scruté par le serveur
  - ▶ Opération de déploiement : copier l'archive dans ce répertoire
  - ▶ Opération de retrait : suppression de l'archive dans ce répertoire

## Étape 6 : Déploiement

- Chaque serveur a son protocole de déploiement.
- Cas du serveur Wildfly/JBoss :
  - ▶ Un répertoire est constamment scruté par le serveur
  - ▶ Opération de déploiement : copier l'archive dans ce répertoire
  - ▶ Opération de retrait : suppression de l'archive dans ce répertoire
  - ▶ Consulter les logs du serveur pour vérifier la bonne exécution des opérations.

# Conclusion

- Ce cours est une **introduction** aux EJB/JPA (EJB = JPA dans un serveur d'applications)

# Conclusion

- Ce cours est une **introduction** aux EJB/JPA (EJB = JPA dans un serveur d'applications)
- Très peu de contraintes de programmation

# Conclusion

- Ce cours est une **introduction** aux EJB/JPA (EJB = JPA dans un serveur d'applications)
- Très peu de contraintes de programmation
- Configuration avec très peu d'annotations grâce aux règles par défaut.

# Conclusion

- Ce cours est une **introduction** aux EJB/JPA (EJB = JPA dans un serveur d'applications)
- Très peu de contraintes de programmation
- Configuration avec très peu d'annotations grâce aux règles par défaut.
- Description approfondie dans le poly JPA

Pour finir

Si vous ne réussissez pas du premier coup, appelez ça "Version 1.0"