



ARCHITECTURES APPLICATIVES

DÉVELOPPEMENT D'APPLICATIONS WEB

AVEC JAKARTA EE

🎓 3A cursus ingénieurs - Mention Architectures des Systèmes Informatiques
🏛️ CentraleSupélec - Université Paris-Saclay - 2023/2024



Idir AIT SADOUNE

idir.aitsadoune@centralesupelec.fr

PLAN

- Applications Web
- Java pour le Web
- Jakarta EE pour le Web
- Persistance des données

[Retour au plan](#) - [Retour à l'accueil](#)

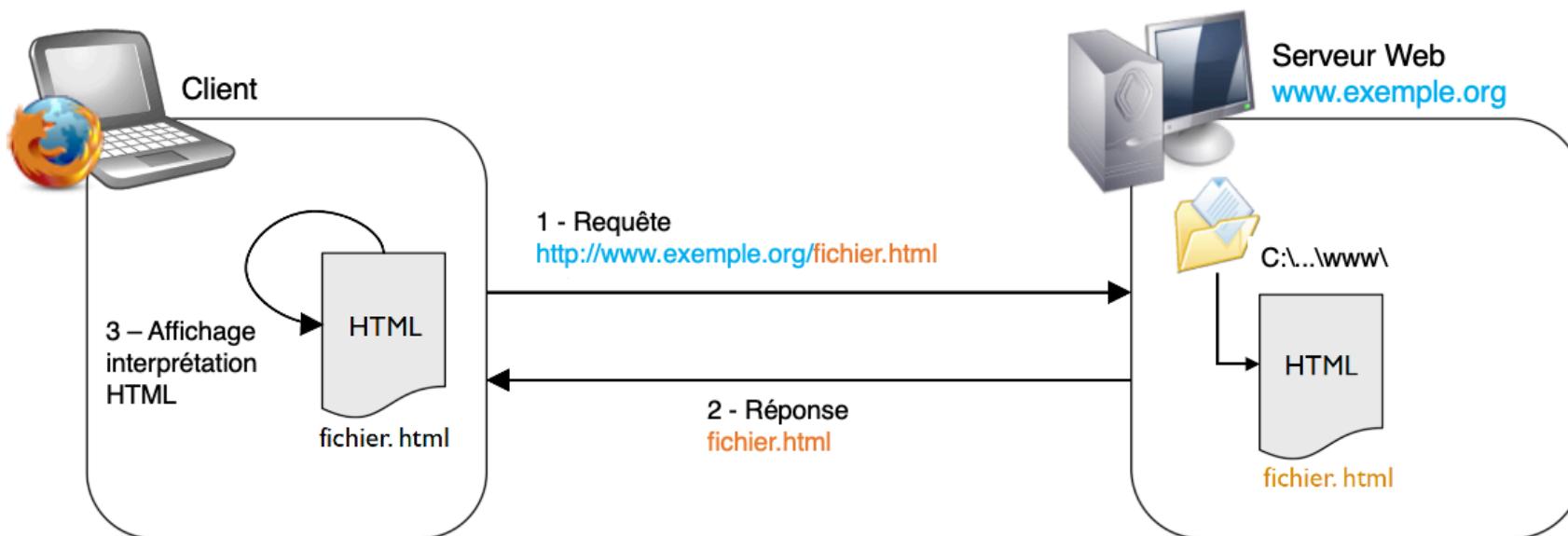
PLAN

-  Applications Web
-  Java pour le Web
-  Jakarta EE pour le Web
-  Persistance des données

[Retour au plan](#) - [Retour à l'accueil](#)

PRINCIPES DES SITES WEB

- Pages **HTML** statiques
- Obtention via le protocole **HTTP** (requête/réponse)



HTML

HTML = EXtensible HyperText Markup Language

- Un langage à base de **balises**
- ≈ **XML** avec une grammaire

STRUCTURATION DE TEXTE

```
1 ...
2 <body>
3   <h1>Gros titre</h1>
4   <p>Voici mon paragraphe</p>
5   <h2>Section</h2>
6   <h3>Sous-section</h3>
7   
8   <p>
9     Un nouveau paragraphe avec un
10    <a href="http://www.exemple.fr">
11      lien hypertexte
12    </a>.
13   </p>
14 </body>
15 ...
```



CSS - PERSONNALISATION DE L'APPARENCE

- Attribut de définition de style

```
<h1 style="color:purple;">Gros titre</h1>
```

- Surcharge de style par défaut

```
<h2>Section</h2>  
h2 { text-decoration: underline; }
```

- Identifiant

```
  
#ampoule { border: 3px dashed blue; }
```

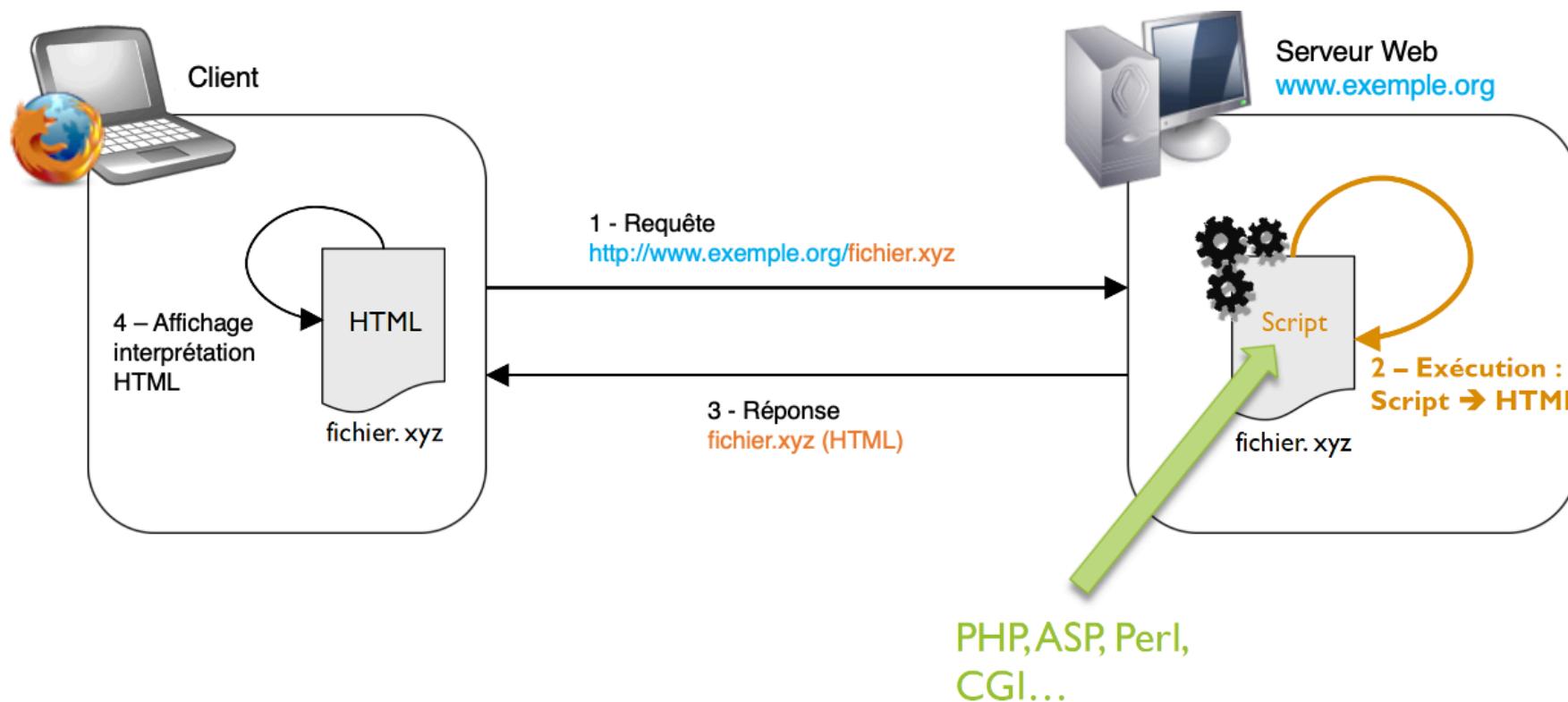
- Classe de style

```
im5 <p class="myp">Voici mon paragraphe</p>  
.myp { font-style: italic; }
```



PRINCIPE DES APPLICATIONS WEB

Génération dynamique du contenu **HTML** à partir
d'un **langage de programmation**.

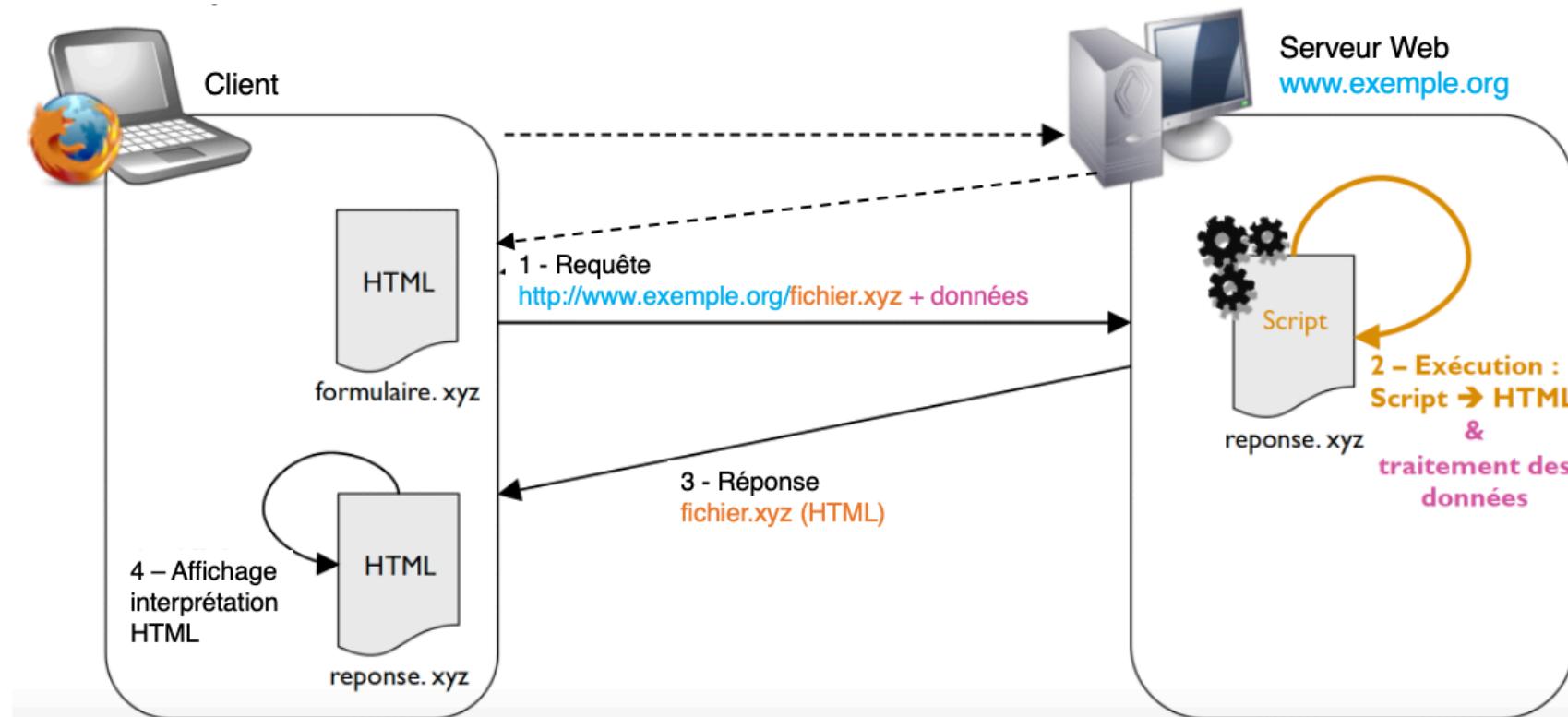


FORMULAIRES

- **Requête** = envoi de données

Interaction avec l'utilisateur

- **Réponse** = résultat du traitement des données



STRUCTURE DES FORMULAIRES

```
1 ...
2 <body>
3     <h1>Un formulaire</h1>
4     <form action="reponse.xyz" method="POST">
5         <label for="prenom">Prénom :</label>
6         <input type="text" id="prenom" name="prenom"/>
7         <button type="submit" value="S'inscrire !"/>
8     </form>
9 </body>
10 ...
```



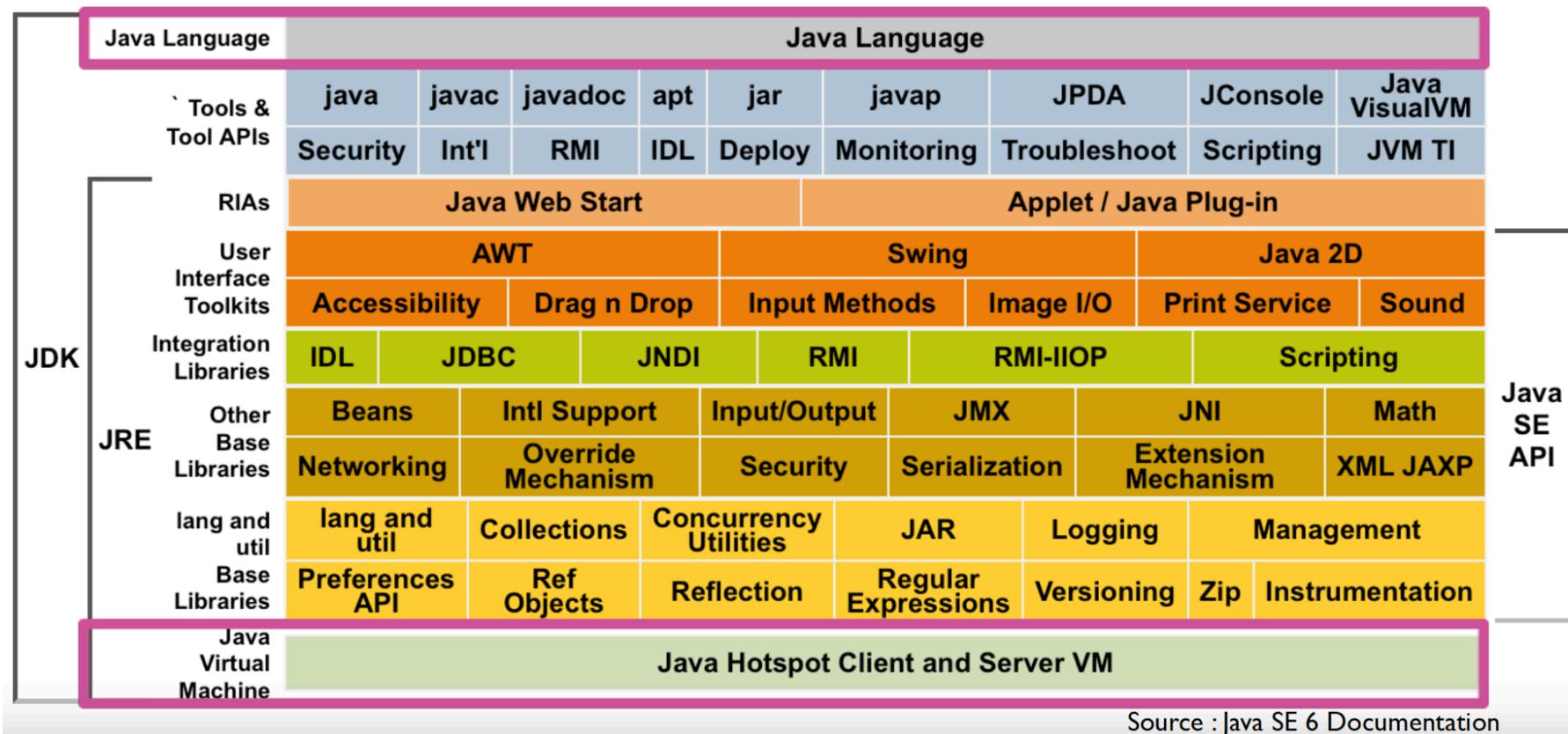
PLAN

- Applications Web
- Java pour le Web
- Jakarta EE pour le Web
- Persistance des données

[Retour au plan](#) - [Retour à l'accueil](#)

JAVA : LANGAGE ET PLATE-FORME

Java Platform Standard Edition (Java SE) =
JVM (machine virtuelle) + APIs (bibliothèques) + Java



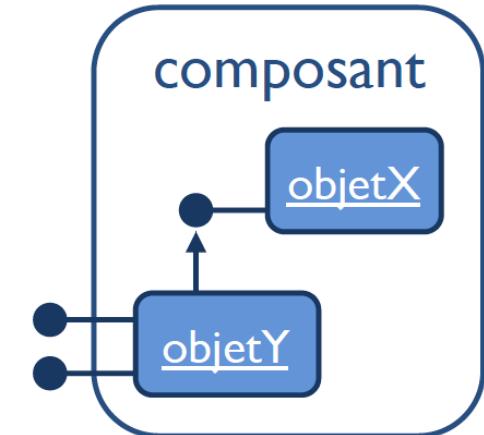
EXEMPLE

```
1 public class Person {  
2     private String name;  
3     private boolean deceased;  
4  
5     public Person(String n, boolean d){  
6         this.name = n;  
7         this.deceased = d;  
8     }  
9  
10    public String getName(){  
11        return this.name;  
12    }  
13  
14    public void setName(String name){  
15        this.name = name;  
16    }  
17  
18    public Integer doSomething(Integer i){  
19        return (i * i);  
20    }  
21 }
```

```
1 public class AutreClasse {  
2     private Integer oneNumber;  
3  
4     public AutreClasse (Integer o) {  
5         this. oneNumber = o;  
6     }  
7  
8     public Person createRobert(){  
9         Person robert;  
10        robert = new Person("Robert", false);  
11        return robert;  
12    }  
13 }
```

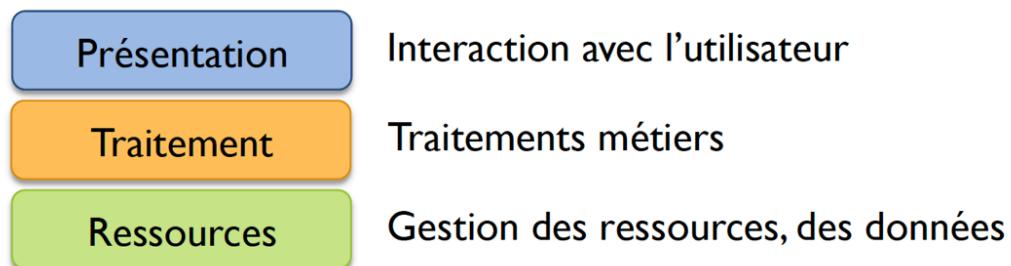
COMPOSANTS

- **Composant** = unité logique de traitement
 - Objectif : modularité et réutilisation
 - Propriétés :
 - Identification : nom unique, référencé dans un annuaire
 - Indépendance : utilisable tout seul
 - Réutilisation : utilisable dans différents contextes
 - Intégration : combinable avec d'autres composants
- Technologies d'implémentation multiples
- Déploiement sur serveur

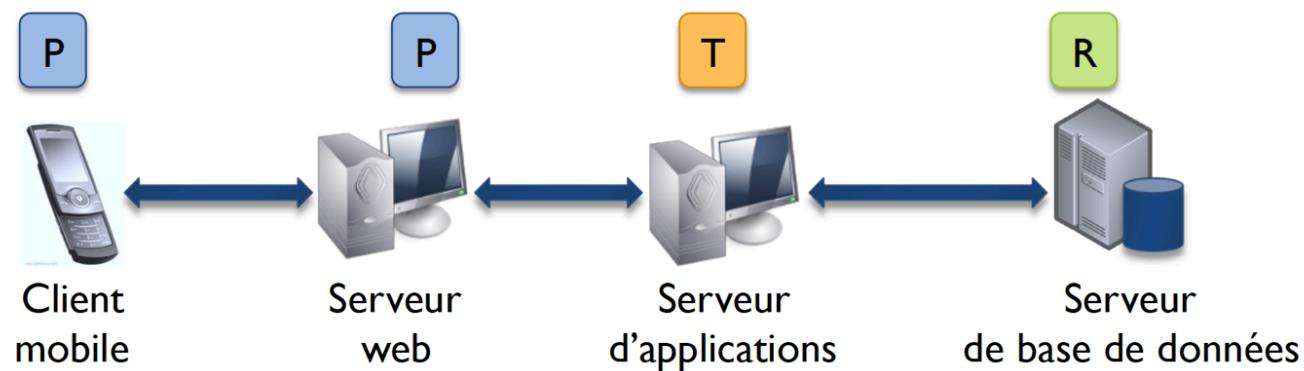


COMPOSANTS ET ARCHITECTURES N-TIERS

- 3 niveaux types de **responsabilités** pour les composants



- **N tiers** sur lesquels sont distribuées les composants



COMPOSANTS JAVA : JAVABEANS

- Composant implémenté par une classe Java
- ≈ Classe Java mais conventions à respecter
 - Sérialisation
 - Constructeur par défaut
 - Propriétés privées avec accesseurs (encapsulation et introspection)
 - `public <type> get<Propertynname>()`
 - `public void set<Propertynname>(parameter)`
 - Méthodes d'interception d'événements
 - Utilisation d'écouteurs et génération d'événements
 - Ex: `implements PropertyChangeListener`



EXEMPLE DE JAVABEANS

```
1 import java.io.Serializable;
2
3 public class PersonBean implements Serializable {
4     private String name;
5     private boolean deceased;
6
7     public PersonBean() { ... }
8
9     public String getName() {
10         return this.name;
11     }
12
13    public void setName(String name) {
14        this.name = name;
15    }
16
17    public boolean getDeceased() {
18        return this.deceased;
19    }
20
21    public void setDeceased(boolean deceased) {
22        this.deceased = deceased;
23    }
24 }
```

JAVA/JAKARTA ENTERPRISE EDITION

- Un ensemble de spécifications
 - Proposées par la société **Sun**, puis **Eclipse Foundation** depuis **2017**.
 - Dediées au développement et au déploiement d'**applications n-tiers** à base de composants centrées sur le serveur
 - ➡ applications d'entreprise
 - Basées sur **Java SE** avec
 - les spécifications du Serveur d'Applications
 - des bibliothèques pour le développement d'applications d'entreprises (**API**)
- Une implémentation de référence
 - ➡ le Serveur d'Applications **GlassFish** version 7.0.13 (certifiée **Jakarta EE 10**)
 - ➡ développé par **Eclipse Foundation** (Février 2024)

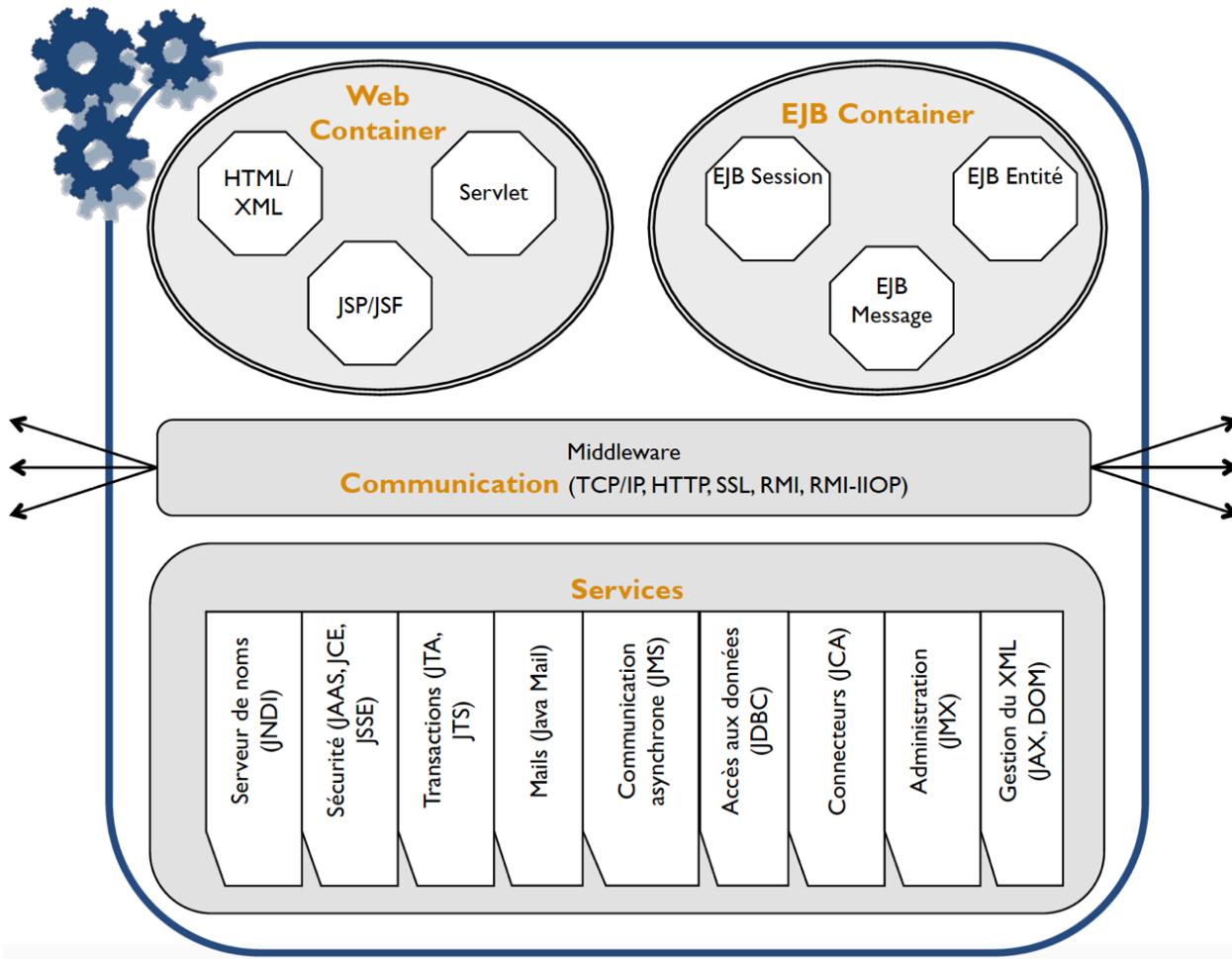


CE QUE DÉFINIT JAKARTA EE

- Des **composants** pour embarquer le code des éléments d'une application
 - composants web
 - composant métiers
- Des **conteneurs** pour héberger les composants d'une application
 - conteneur web
 - conteneur métier
- Des **services** support pour les aspects transverses
 - Sécurité, transactions ...
- Des **infrastructures** de communication



LE SERVEUR D'APPLICATION JAKARTA EE

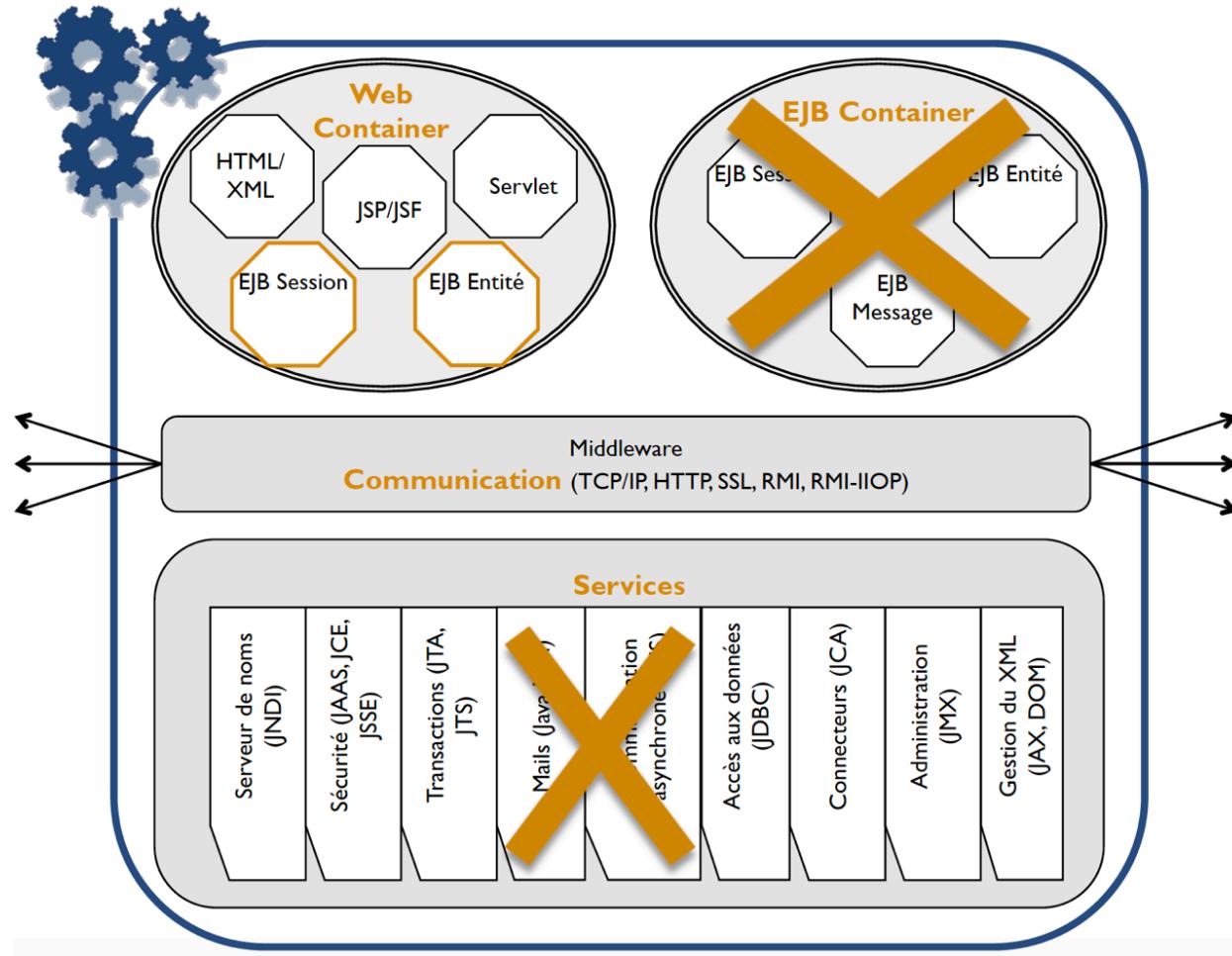


JAKARTA EE WEB PROFILE

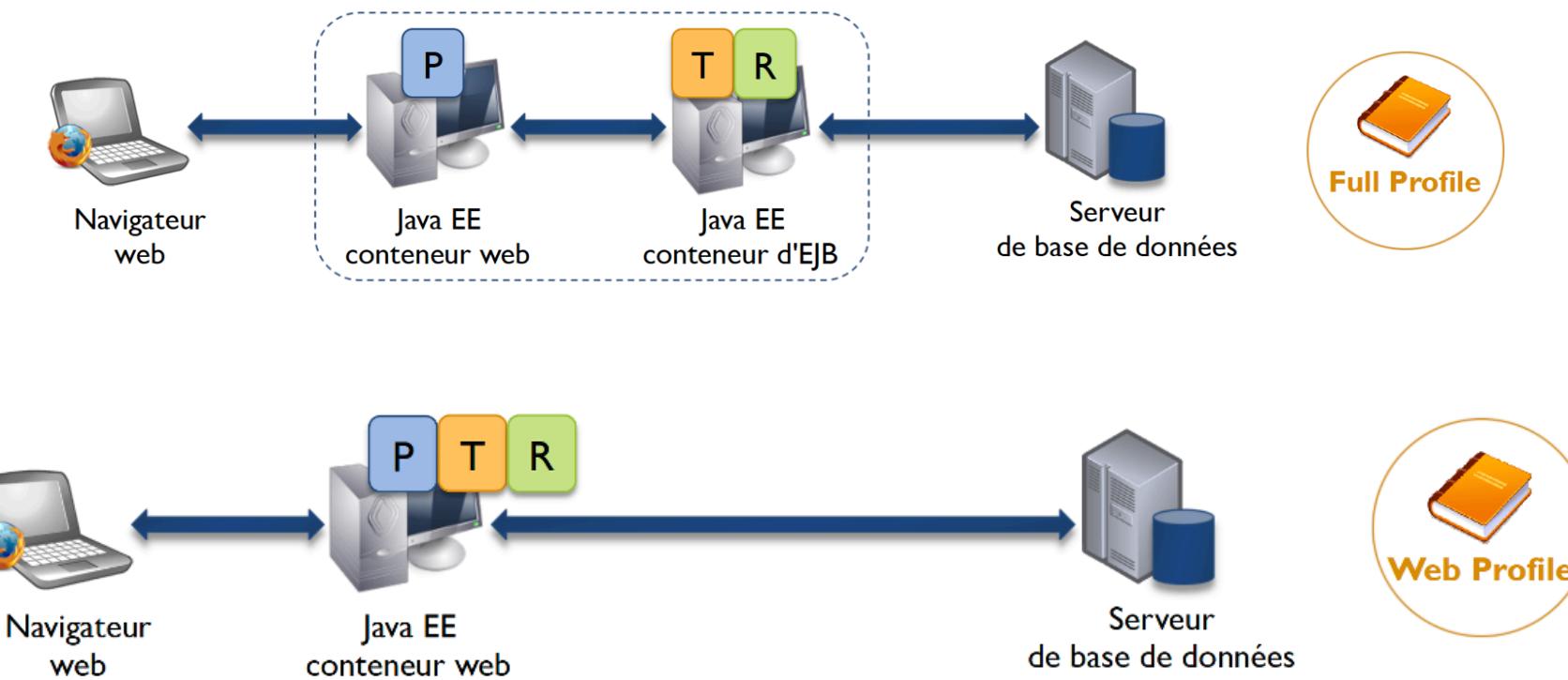
- Sous-ensemble de spécifications séparé dédié au développement web
- Inclut
 - composants web
 - composants métier légers
 - API communes
- N'utilise que le conteneur Web



JAKARTA EE WEB PROFILE



ARCHITECTURES JAKARTA EE TYPE

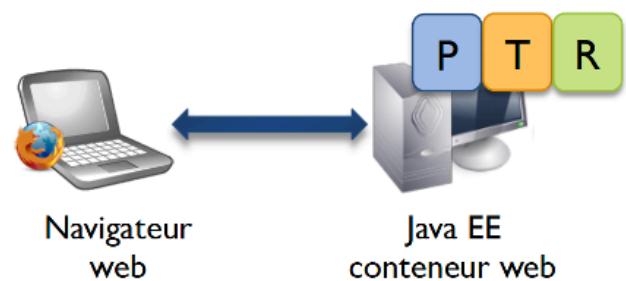


PLAN

- Applications Web
- Java pour le Web
- Jakarta EE pour le Web
- Persistance des données

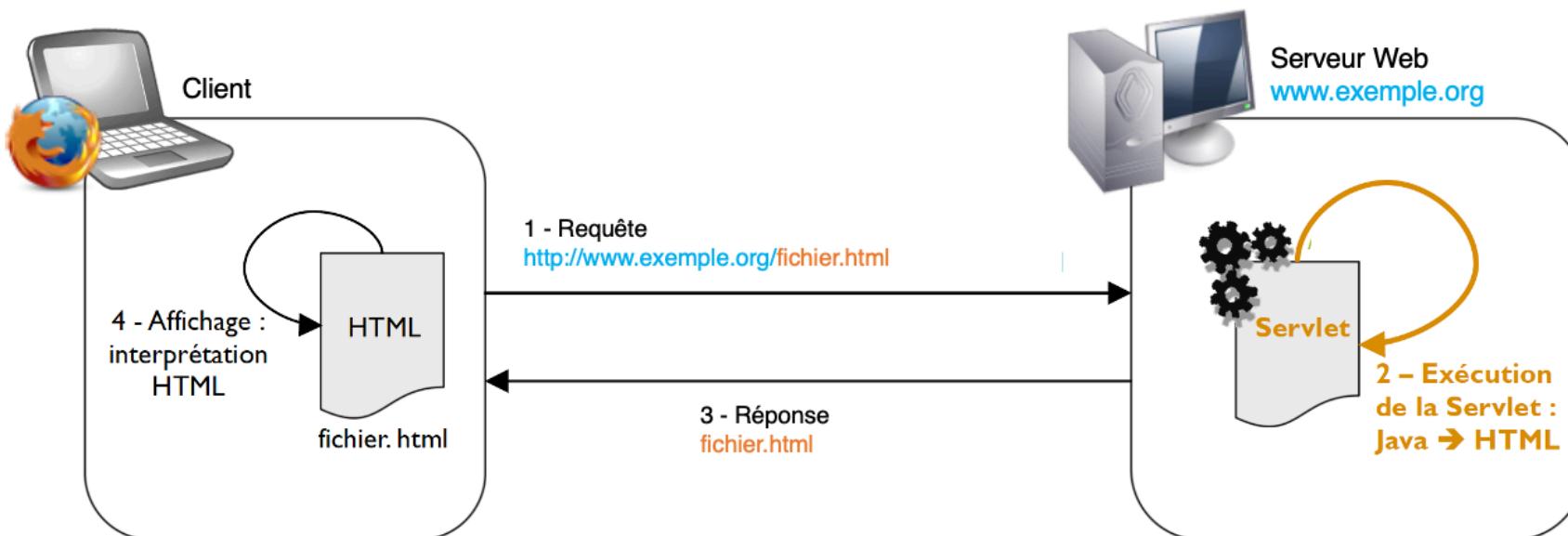
[Retour au plan](#) - [Retour à l'accueil](#)

PARTIE WEB DE JAVA EE



SERVLETS

- **Servlet** = composant **Java**, accessible à une **URL** donnée, destiné à :
 1. Récupérer les requêtes HTTP de l'utilisateur
 2. Invoquer les traitements applicatifs **Java** correspondants
 3. **Construire la page HTML de réponse** pour l'utilisateur



EXEMPLE DE SERVLETS

```
1 @WebServlet(name="PremiereServlet", urlPatterns={"/premiereServlet"})
2 public class PremiereServlet extends HttpServlet {
3     public void doGet(HttpServletRequest request,
4         HttpServletResponse response) throws IOException, ServletException {
5
6         response.setContentType("text/html");
7         PrintWriter pw = response.getWriter();
8         try {
9             pw.print("<html>");
10            pw.print("<head>");
11            pw.print("<title>Ma premiere servlet</title>");
12            pw.print("</head>");
13            pw.print("<body>");
14            pw.print("<h1>Ca marche !</h1>");
15            pw.print("</body>");
16            pw.print("</html>");
17        } finally {
18            out.close();
19        }
20    }
21 }
```

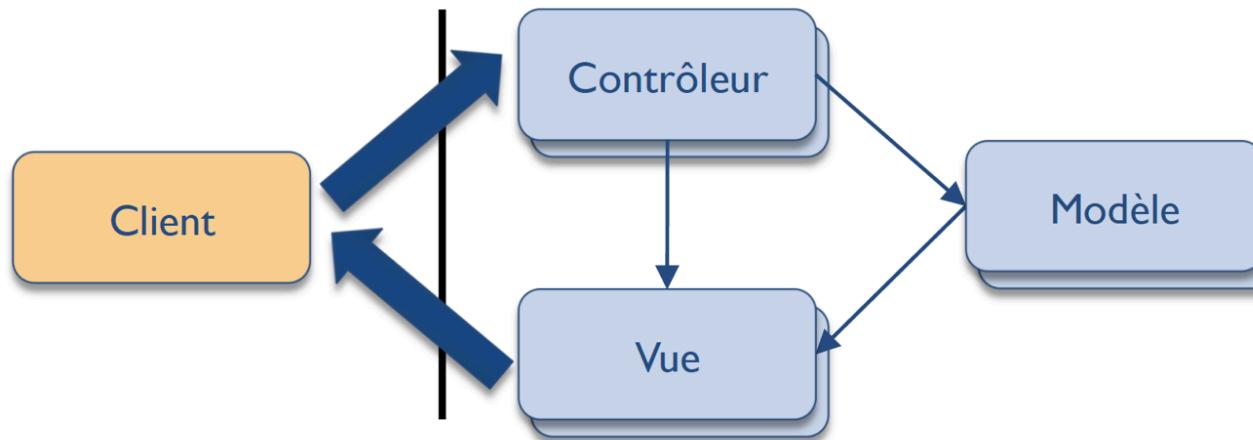
EXAMPLE DE SERVLETS



Problèmes

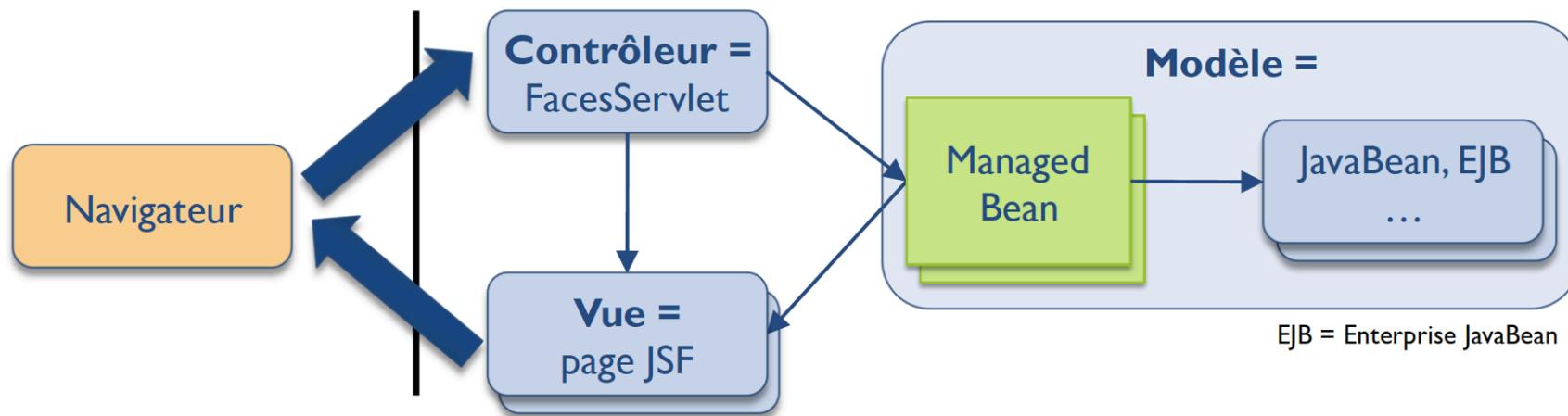
maintenance, évolution... !

PATRON MODÈLE-VUE-CONTRÔLEUR (MVC)



- Principe = séparation des responsabilités
 - **Vue** : gère les sorties = visualisation (interface utilisateur)
 - **Contrôleur** : gère les entrées
 - Traitement des actions de l'utilisateur et choix des vues
 - Gestion des modifications du modèle
 - **Modèle** : gère la logique et les données

JAVA SERVER FACES (JSF)

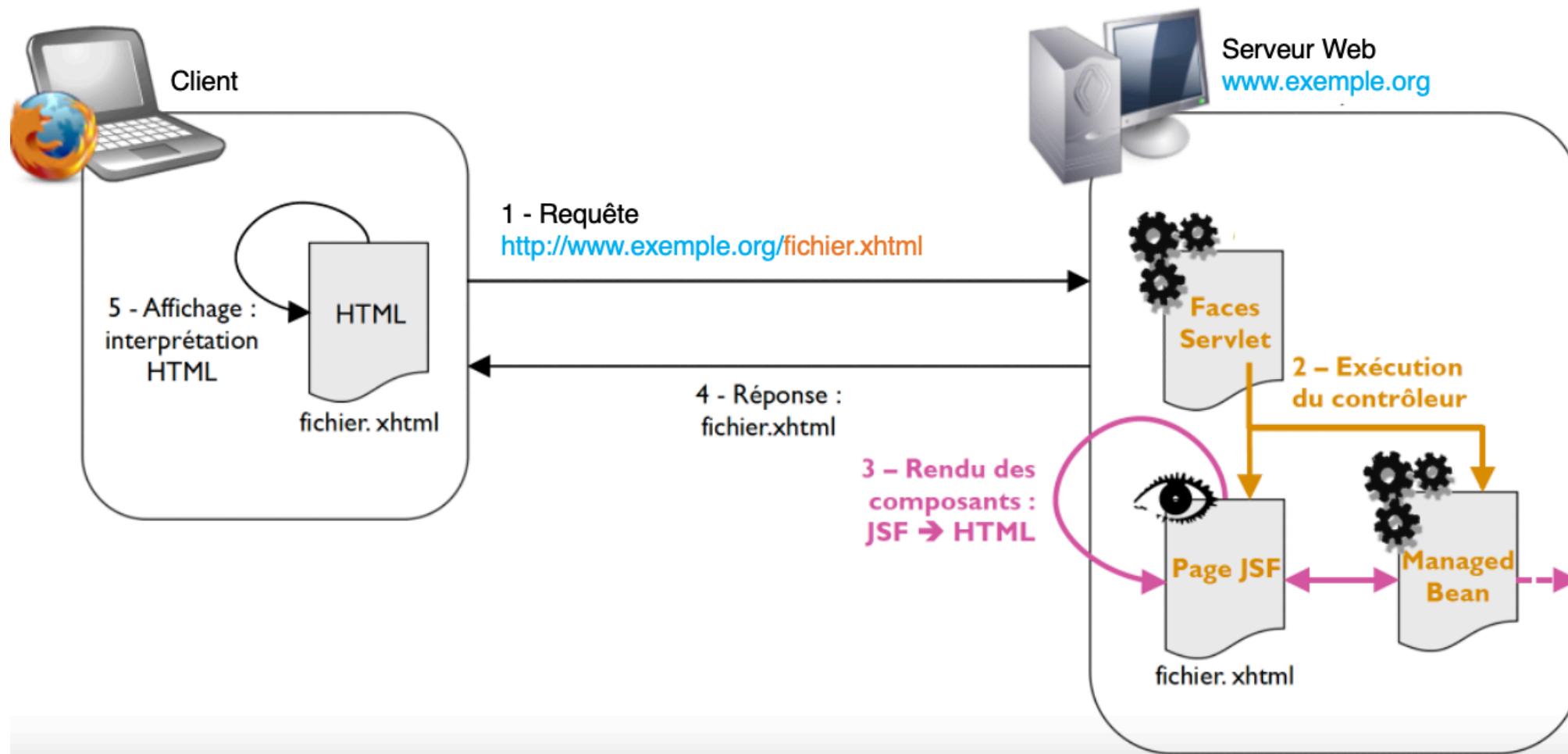


- **Framework de composants web** basé sur **MVC** amélioré :
 - ➡ modèle = logique + données

CE QUE JSF DÉFINIT

- **Contrôleur unique = FacesServlet**
 - Gère les événements et les interactions avec le client
- **Modèle =**
 - Logique = **Managed Beans** ou "backing beans"
 - Données = JavaBeans, Enterprise JavaBeans...
- **Vue = page JSF** = arbre de composants
 - Composants prédéfinis avec un rendu **HTML**
 - Modèles de navigation
 - ➡ **Statique** : liens hypertextes entre vues
 - ➡ **Dynamique** : règles de navigation s'appuyant sur un bean
 - Outils : convertisseur et validateurs

PRINCIPE DE JSF



COMPOSANTS JSF

- Une **classe** : gère les événements utilisateur

```
1 UISelectOne choiceMenu;
```

- Un **tag** : permet de l'inclure dans une vue JSF

```
1 <h:selectOneMenu id="choice" value="...">>
  2   <f:selectItems value="..." />
  3 </h:selectOneMenu>
```

- Un **rendu** : génère la partie graphique de l'interface utilisateur
(HTML par défaut)

```
1 <select id="j_idt7:choice" name="j_idt7:choice" size="1">
  2   <option value="1" selected="selected">item 1</option>
  3   <option value="2">item 2</option>
  4 </select>
```

COMPOSANTS JSF

Bibliothèques de composants prédéfinis et composables

The image shows a screenshot of a Java application interface. On the left, there is a calendar component displaying the month of July 2005. The date 08 July is highlighted in blue. The calendar grid shows weeks from 26 to 30, with the 8th marked as a Friday. A tooltip at the bottom of the calendar says "Today is Fri, 8 Jul 2005". On the right, there is a hierarchical file tree structure. The root node is "Frank Foo", which has several sub-folders: "Requires Foo", "Requires Foo Reviewer (3)", "Requires Foo Recommendation", and "Requires Foo Approval (6)". The "Requires Foo Approval (6)" folder contains six files: J050001, J050002, J050003, E050011, R050002, and C050003.

MANAGED BEANS (BACKING BEANS)

- Composant géré par le contrôleur **JSF**
 - ≈ **JavaBean** mais avec **annotations**
 - Gestion par le **conteneur** (création, cycle de vie...)
- **Services basiques**
 - Identification unique, **utilisation dans les composants JSF**
 - Expression Language (**EL**)
 - Gestion du **scope** (durée de conservation de l'état)
 - Gestion du cycle de vie via des **callbacks**
 - Injection
 - Conversion/validation des données

MANAGED BEANS (BACKING BEANS)

```
1 import jakarta.enterprise.context.SessionScoped;
2 import jakarta.inject.Named;
3 import jakarta.annotation.PostConstruct;
4
5 @SessionScoped
6 @Named
7 public class MyManagedBean implements Serializable {
8     private String name;
9
10    public MyManagedBean (){...}
11
12    @PostConstruct
13    public void mySetup () {
14        this.name = "Duke";
15    }
16
17    public String getName(){return name;}
18    public void setName(String n){this.name = n;}
19
20    public String sayHello() {
21        return "Hello, I'm " + this.name;
22    }
23 }
```

MANAGED BEANS ET COMPOSANTS JSF

```
1 <f:view>
2   <h:form>
3     <h1>Creation d'un compte</h1>
4     <p>Nom :</p>
5     <h:inputText
6       value="#{personBean.name}"
7       required="true"/>
8   </h:inputText>
9
10    <p>Prenom :</p>
11    <h:inputText
12      value="#{personBean.fName}">
13      <f:validateLength maximum="50"/>
14    </h:inputText>
15
16    <h:commandButton value="Valider"
17      action="#{personBean.createAccount()}" />
18  </h:form>
19 </f:view>
```

```
1 @SessionScoped
2 @Named
3 public class PersonBean implements Serializable {
4   private String name;
5   private String fName;
6
7   public CatalogBean(){
8     ...
9   }
10  ...
11  public String createAccount(){
12    ...
13    return "createAccountSuccess";
14  }
15 }
```

JSF ET NAVIGATION WEB

- A l'aide de **règles de navigation** décrites dans **faces-config.xml**, on déterminer, en fonction de la page courante, la page cible.
- **Outcome** =
 - Chaine de caractère représentant le **résultat d'un traitement** (Ex : fail, pass, success, case5, caseTruc...)
 - Peut conditionner l'activation d'une règle de navigation

```
1 <navigation-rule>
2 <!-- Si l'outcome renvoie est HelloWorld
3 alors JSF passe a la page /hello-world.jsp -->
4   <from-view-id>/index.jsp</from-view-id>
5   <navigation-case>
6     <from-outcome>HelloWorld</from-outcome>
7     <to-view-id>/hello-world.jsp</to-view-id>
8   </navigation-case>
9 </navigation-rule>
```

TYPES DE NAVIGATION

- **Navigation statique** = navigation par identifiant de vue/outcome
- **Navigation dynamique** =
 - ➡ méthode d'un backing bean rendant plusieurs outcomes possibles
 - ➡ règles de navigation différentes suivant les outcomes

EXEMPLE DE NAVIGATION STATIQUE

```
1 <navigation-rule>
2     <from-view-id>/index.jsp</from-view-id>
3     <navigation-case>
4         <from-outcome>outcom1</from-outcome>
5         <to-view-id>/action1.jsp</to-view-id>
6     </navigation-case>
7     <navigation-case>
8         <from-outcome>outcom2</from-outcome>
9         <to-view-id>/action2.jsp</to-view-id>
10    </navigation-case>
11 </navigation-rule>
```

```
1 <!-- Dans index.jsp -->
2 <h:commandLink action="outcom1" value="Exemple Outcom 1"/>
3 <h:commandLink action="outcom2" value="Exemple Outcom 2"/>
```

EXEMPLE DE NAVIGATION DYNAMIQUE

```
1 <navigation-rule>
2   <from-view-id>/index.jsp</from-view-id>
3   <navigation-case>
4     <from-outcome>outcom1</from-outcome>
5     <to-view-id>/action1.jsp</to-view-id>
6   </navigation-case>
7   <navigation-case>
8     <from-outcome>outcom2</from-outcome>
9     <to-view-id>/action2.jsp</to-view-id>
10  </navigation-case>
11 </navigation-rule>
```

```
1 public class MyBean implements Serializable {
2 ...
3   public String allerEtapeSuivante(){
4 ...
5     if(...) return "outcom1";
6     else return "outcom2";
7   }
8 }
```

EXEMPLE DE NAVIGATION DYNAMIQUE

```
1 <navigation-rule>
2   <from-view-id>/index.jsp</from-view-id>
3   <navigation-case>
4     <from-outcome>outcom1</from-outcome>
5     <to-view-id>/action1.jsp</to-view-id>
6   </navigation-case>
7   <navigation-case>
8     <from-outcome>outcom2</from-outcome>
9     <to-view-id>/action2.jsp</to-view-id>
10  </navigation-case>
11 </navigation-rule>
```

```
1 <!-- Dans index.jsp -->
2 <h:commandLink action="#{mybean.allerEtapeSuivante()}" value="étape suivante" />
```

SESSIONS ET COMPOSANTS WEB

- **Session web** = maintient d'un état conversationnel avec un client à travers plusieurs couples requête/réponse
 - ➡ permet de stocker des informations alors que l'utilisateur change plusieurs fois de pages web
- **Session JSF** = **Session web**
 - Mettre des données en sessions avec un **ManagedBean**

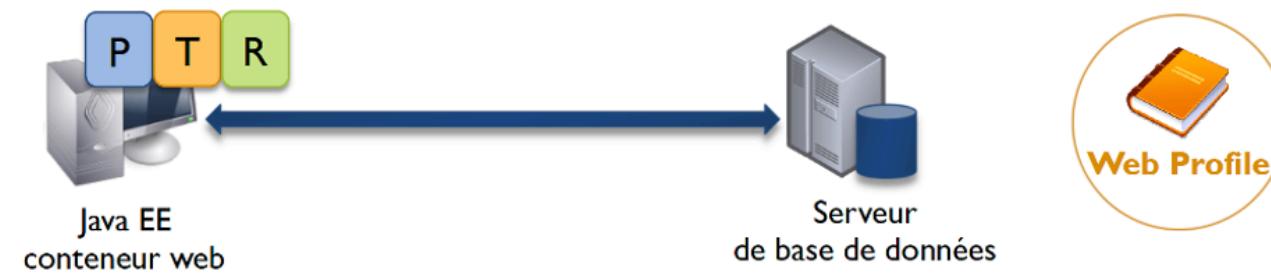
```
1 @SessionScoped  
2 @Named  
3 public class PersonBean implements Serializable {...}
```

PLAN

- Applications Web
- Java pour le Web
- Jakarta EE pour le Web
- Persistance des données

[Retour au plan](#) - [Retour à l'accueil](#)

PARTIE PERSISTANCE DE JAKARTA EE



BASES DE DONNÉES

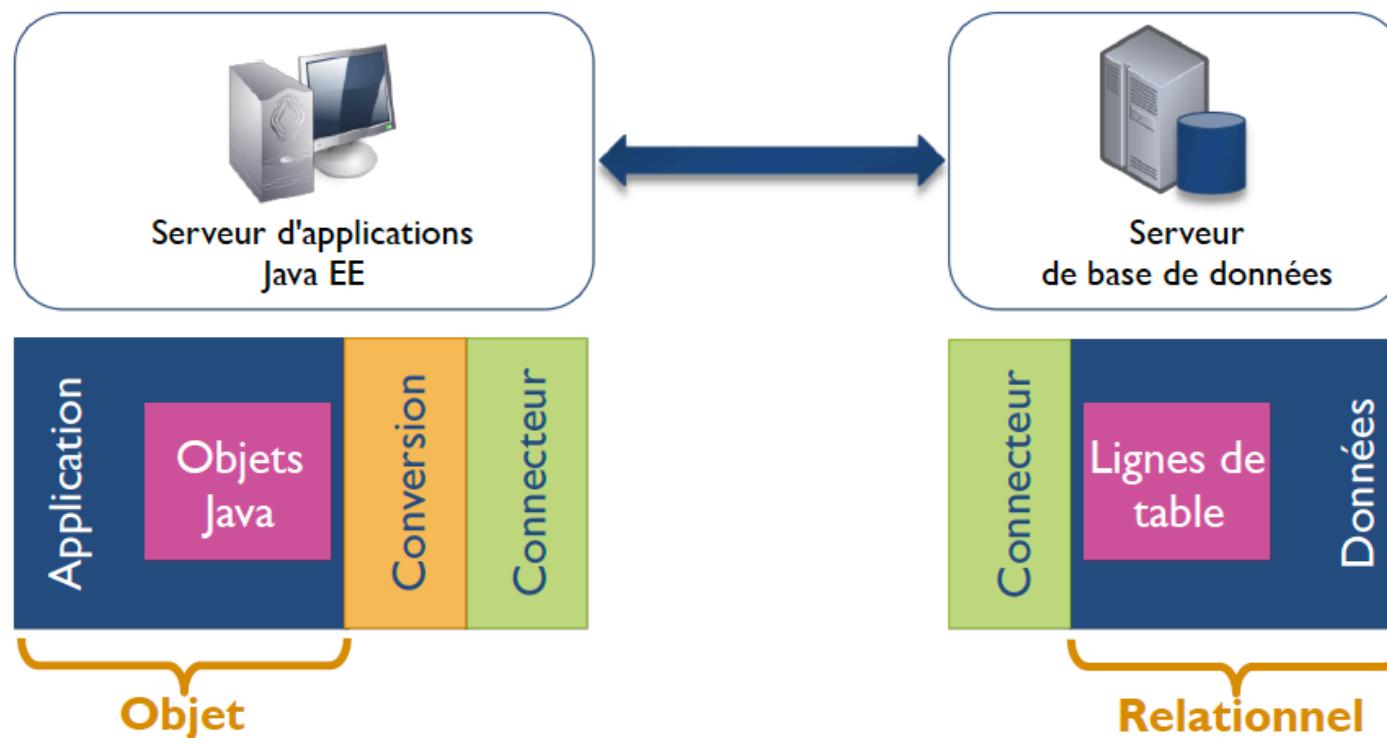
- Ensemble de tables
 - **Table** = colonnes × lignes
 - **Colonnes** = propriétés
 - **Ligne** = ensemble de valeurs pour les différentes propriété
 - **Clé primaire** = identifiant unique d'une ligne
- Stockage persistant sur un serveur
- Gestion par un **SGBD** (Système de Gestion de Base de Données)

ID	NAME	SELLING_PRICE
1	PC configuration de base	400.0
2	PC configuration jeux vidéos	800.0
3	MacBook configuration de base	600.0
4	MacBook configuration performance	1000.0
5	sdfsdf	3.0



PERSISTANCE EN BASE DE DONNÉES

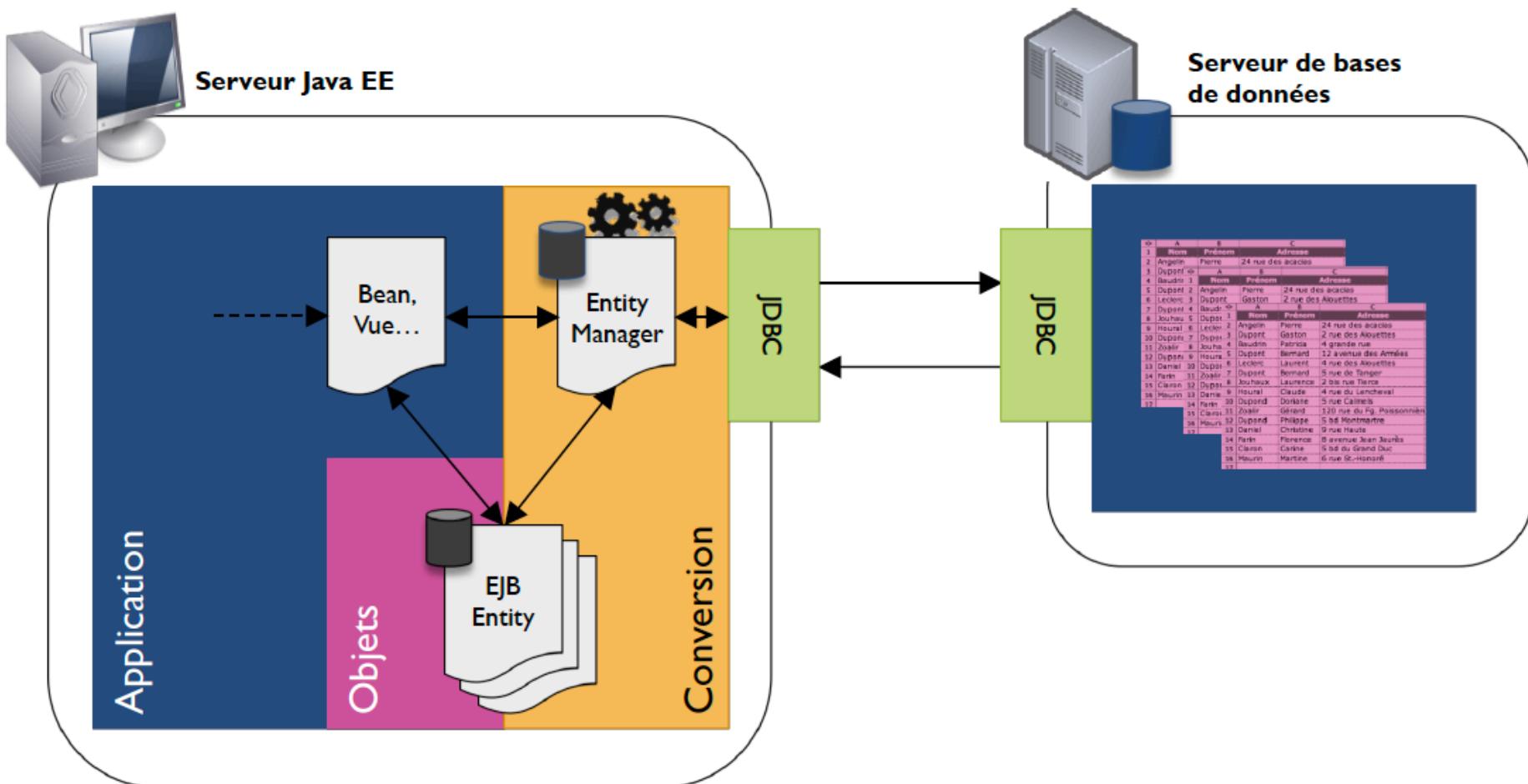
- Besoin = Create + Read + Update + Delete (**CRUD**)
- Problème = mapping relationnel/objet des données



JAVA PERSISTENCE API (JPA)

- JPA = spécification d'un mapping relationnel/objet (Object Relational Mapping – ORM)
 - Composant pour accéder à la base de données = EntityManager
 - ➡ prédefini (pas besoin de l'écrire)
 - ➡ générique
 - Composants pour stocker les données = EJB Entity
 - Langage de requêtes = Java Persistence Query Language (JPQL)
- Plusieurs implémentations
 - EclipseLink (Oracle TopLink)
 - JBoss Hibernate...
- S'appuie sur un connecteur de base de données
Ex : JDBC

PRINCIPE DE JPA



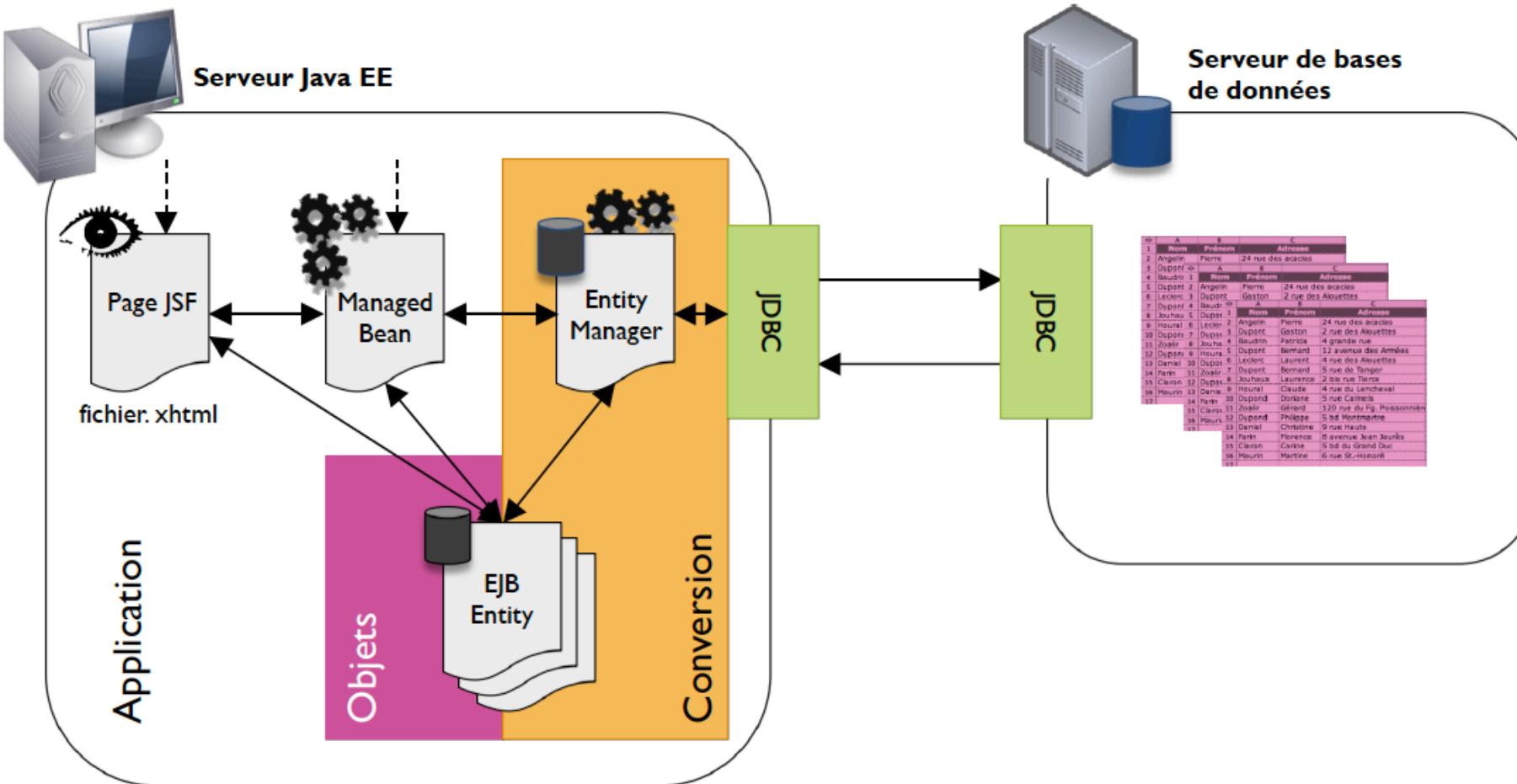
EJB ENTITY

- Composant **persistent** partagé par plusieurs clients
 - Représente des données métiers stockées dans une base de données
 - ➡ une **classe** correspond à une **table**
 - ➡ les **attributs** définis dans la classe correspondent aux **colonnes de la table**
 - ➡ une **instance** de la classe (= un objet) représente une **ligne de la table**
 - Possède une **identité explicite (clé primaire)**
- **EJB Entity** ≈ **JavaBean** mais ...
 - ➡ avec des **annotations** qui définissent le **mapping relationnel/objet**
 - ➡ gestion par le conteneur (création, cycle de vie...)

EXAMPLE

```
1 @Entity
2 @Table(name = "person")
3 public class Personne implements Serializable {
4     @Id
5     @GeneratedValue(strategy = GenerationType.SEQUENCE)
6     @Column(name = "id")
7     private int id;
8
9     @Basic(optional = false)
10    @Column(name = "name")
11    private String nom;
12
13    public Personne() { super(); }
14    public int getId() { return this.id; }
15    public void setId(int id) { this.id = id; }
16    public String getNom() { return this.nom; }
17    public void setNom(String nom) { this.nom = nom; }
18    ...
19 }
```

JPA ET JSF



MERCI

[Retour à l'accueil](#) - [Retour au plan](#)