



Programmation Par Composants 2

Yassamine Seladji

yassamine.seladji@gmail.com

25 janvier 2021

Introduction

” Un composant est à l'objet ce que l'atome est à la molécule”.

- ▶ La programmation par composant est basée sur le fait d'intégrer, d'emboîter des composants entre eux pour former un programme.

Introduction

” Un composant est à l'objet ce que l'atome est à la molécule”.

- ▶ La programmation par composant est basée sur le fait d'intégrer, d'emboîter des composants entre eux pour former un programme.
- ▶ Il existe deux catégories :

Introduction

” Un composant est à l'objet ce que l'atome est à la molécule”.

- ▶ La programmation par composant est basée sur le fait d'intégrer, d'emboîter des composants entre eux pour former un programme.
- ▶ Il existe deux catégories :
 - ▶ Les composants généraux : client/IHM (Java Beans).

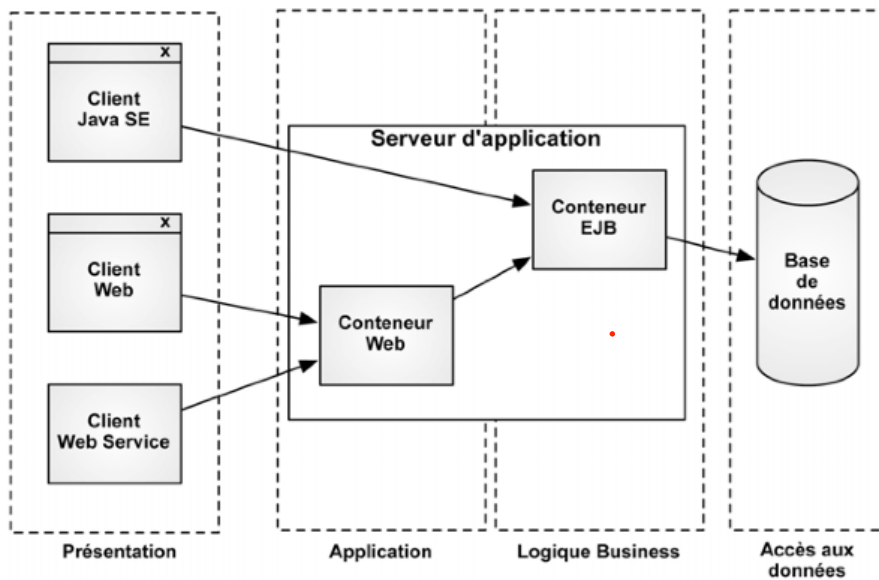
Introduction

” Un composant est à l'objet ce que l'atome est à la molécule”.

- ▶ La programmation par composant est basée sur le fait d'intégrer, d'emboîter des composants entre eux pour former un programme.
- ▶ Il existe deux catégories :
 - ▶ Les composants généraux : client/IHM (Java Beans).
 - ▶ Les composants serveur/métier : les EJBs (Entreprise Java Bean).

Introduction

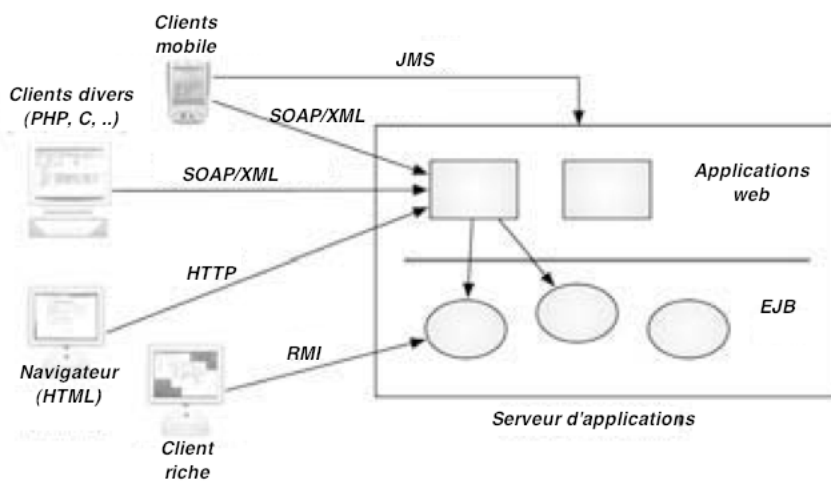
L'architecture logicielle d'une application est constituée de couches.



La couche présentation

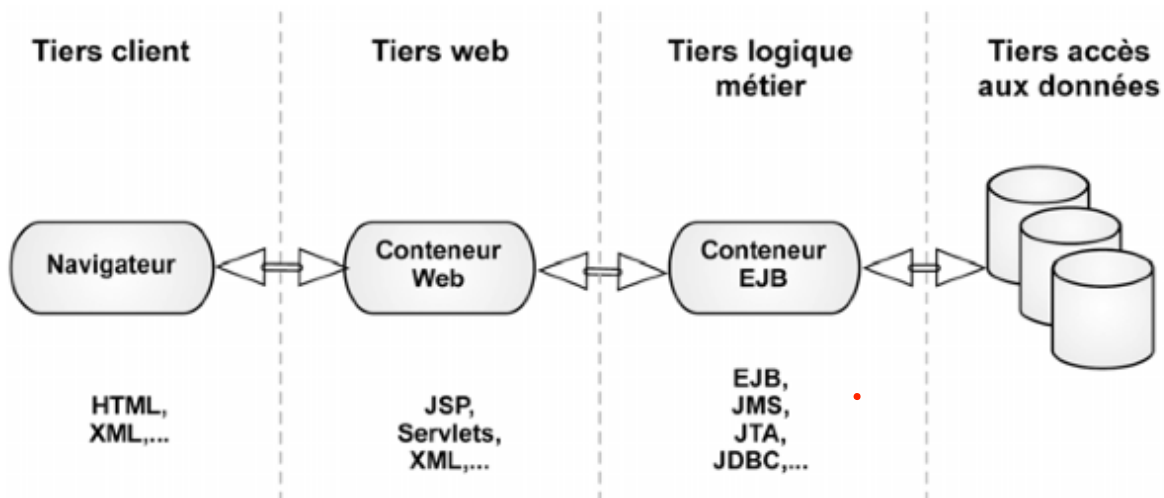
Le client peut être :

- ▶ un client riche : des applications graphiques fenêtrées.
- ▶ un client léger : navigateur web.



La couche application

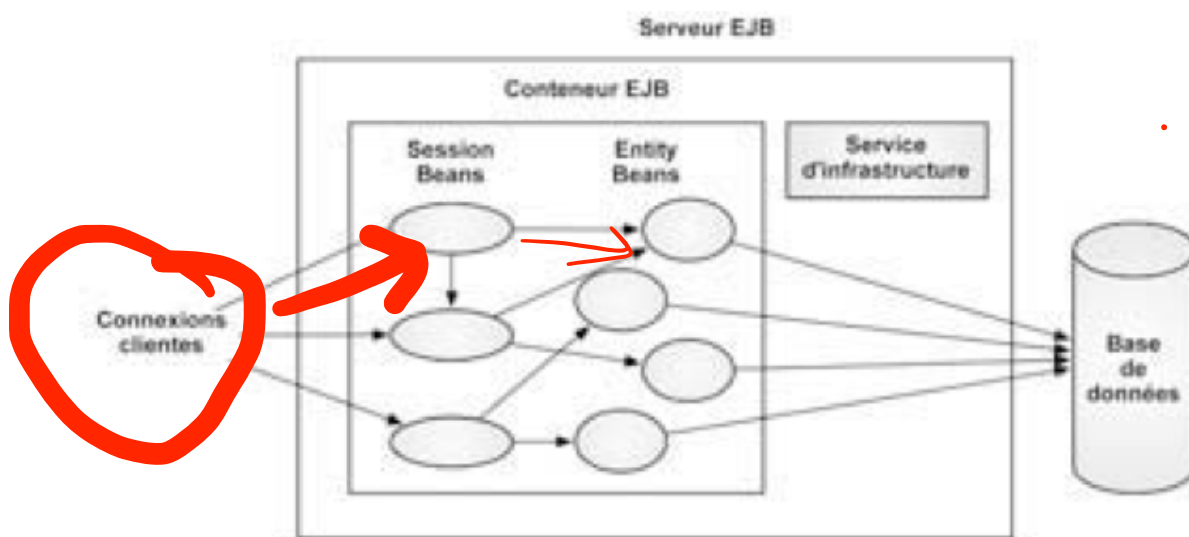
- ▶ Les EJBs sont utilisés dans la couche logique métier.
- ▶ Côté serveur dans l'architecture N-tiers.



La couche application

Le serveur EJB

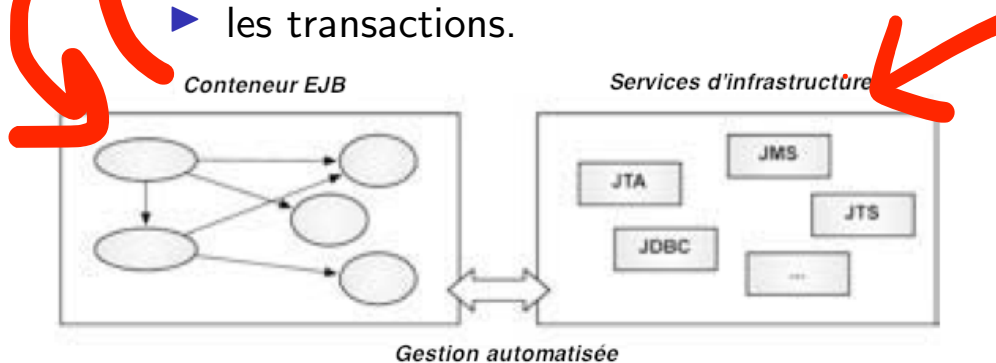
- ▶ Le serveur dirige l'ensemble des services et leur cycle de vie.
- ▶ Le serveur aiguille l'ensemble des requêtes.
- ▶ Le serveur gère l'ensemble des conteneurs et services.



La couche application

Le conteneur EJB

- ▶ Le conteneur et le serveur fournissent l'environnement d'exécution pour les **EJB**.
- ▶ Le conteneur **EJB** gère
 - ▶ la gestion du cycle de vie du EJB.
 - ▶ l'accès au EJB.
 - ▶ la sécurité d'accès.
 - ▶ l'accès concurrents.
 - ▶ les transactions.



Avantages des EJBs

- ▶ Ils permettent aux développeurs de se concentrer sur la logique métier.
- ▶ L'environnement d'exécution prend en charge les traitements techniques :
 - ▶ La gestion des transactions.
 - ▶ La persistance des données. •
 - ▶ La concurrence.
 - ▶ La sécurité.
 - ▶ ...
- ▶ Permettent de séparer le code métier du code technique. •

Enterprise Java Beans

Enterprise Java Beans :

- ▶ est une architecture de composants logiciels côté serveur pour des applications Java EE.
- ▶ est une technologie Java hébergée au sein d'un serveur applicatif.
- ▶ propose un cadre pour créer des composants distribués.

La couche application

Il existe plusieurs serveurs d'EJB commerciaux :

- ▶ BEA Weblogic.
- ▶ IBM Webpsphere.
- ▶ Sun IPlanet.
- ▶ Macromedia JRun.
- ▶ Borland AppServer.
- ▶ ...

Il existe aussi des serveurs d'EJB open source dont les plus avancés sont **JBoss**, **glassfish** et **Jonas**.

Entreprise Java Beans

Plusieurs version d'EJB ont vues le jour :

- ▶ EJB 3.0 introduit l'utilisation d'annotations Java pour simplifier le développement en cachant les détails d'implémentation.
- ▶ L'étude de EJB 2.X permet de comprendre comment fonctionne les EJBs.

Entreprise Java Beans

Les EJB se composent de :

→ **session beans** :

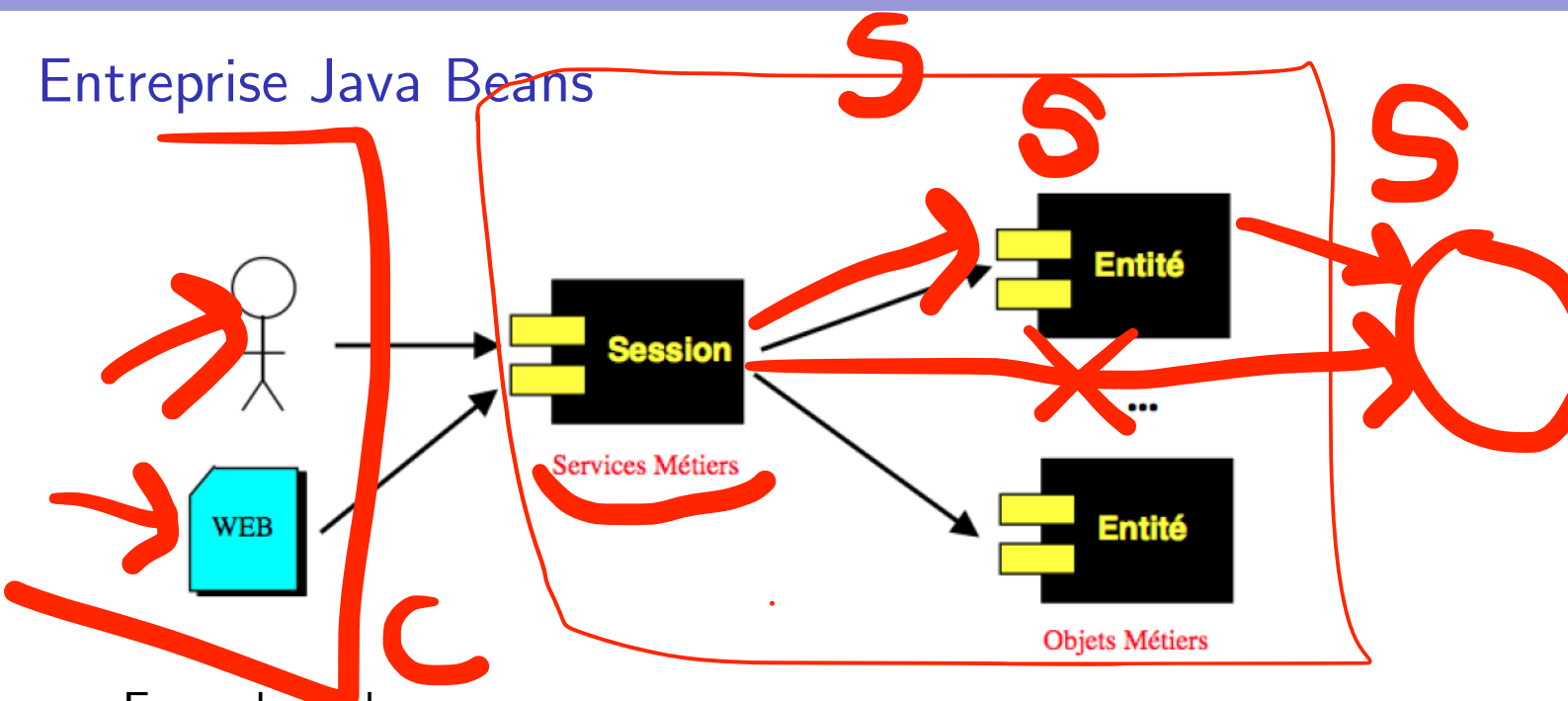
- ▶ une extension du client sur le serveur.
- ▶ un pont entre les clients et les données .

→ **entité beans** :

- ▶ représentation des données de la BDD.
- ▶ servent à accéder aux données.

▶ **beans orientés message** : gestion asynchrone des évènements.

Entreprise Java Beans



Exemple de banque :

- ▶ faire un virement : service (session)
- ▶ récupérer le solde : donnée (entité)

Session Bean

Un **Session Bean** :

- ▶ est une application côté **serveur**.
- ▶ permet de fournir un ou plusieurs **services** à différentes applications clientes.
- ▶ contient les services métiers de l'application.
- ▶ Le bean est supprimé lorsque le client n'en a plus besoin.

Session Bean

Il existe deux types de **Session Bean** :

- ▶ session bean avec état : **Statful**
- ▶ session bean sans état : **Stateless**

Session Bean

Stateless signifie que le service est **autonome** dans son exécution.

Un Session Bean Stateless :

- ▶ ne conserve aucun état entre deux invocations de méthodes.

Session Bean

Stateless signifie que le service est **autonome** dans son exécution.

Un Session Bean Stateless :

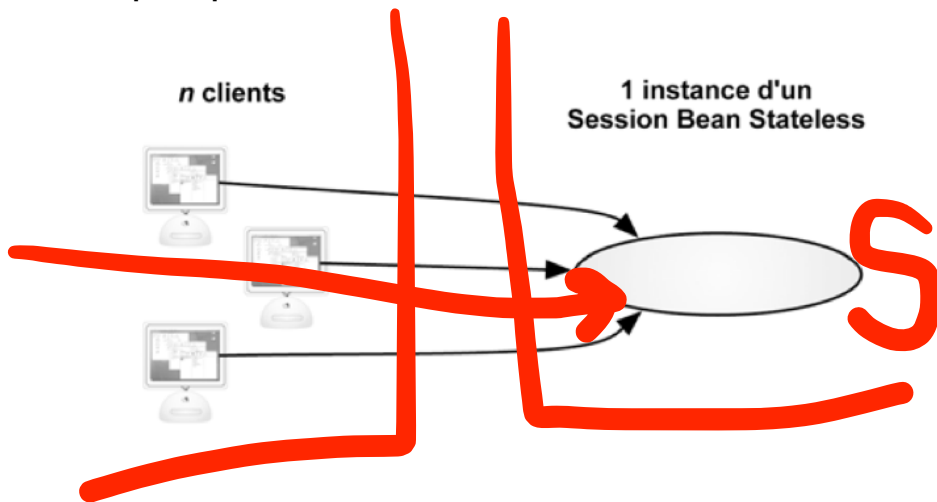
- ▶ ne conserve aucun état entre deux invocations de méthodes.
- ▶ reste général afin de pouvoir être réutilisé dans d'autres contextes.

Session Bean

Stateless signifie que le service est **autonome** dans son exécution.

Un Session Bean Stateless :

- ▶ ne conserve aucun état entre deux invocations de méthodes.
- ▶ reste général afin de pouvoir être réutilisé dans d'autres contextes.
- ▶ est utilisé par plusieurs clients.



Session Bean

Un Stateful Session Bean :

- ▶ est une extension de l'application cliente.

Session Bean

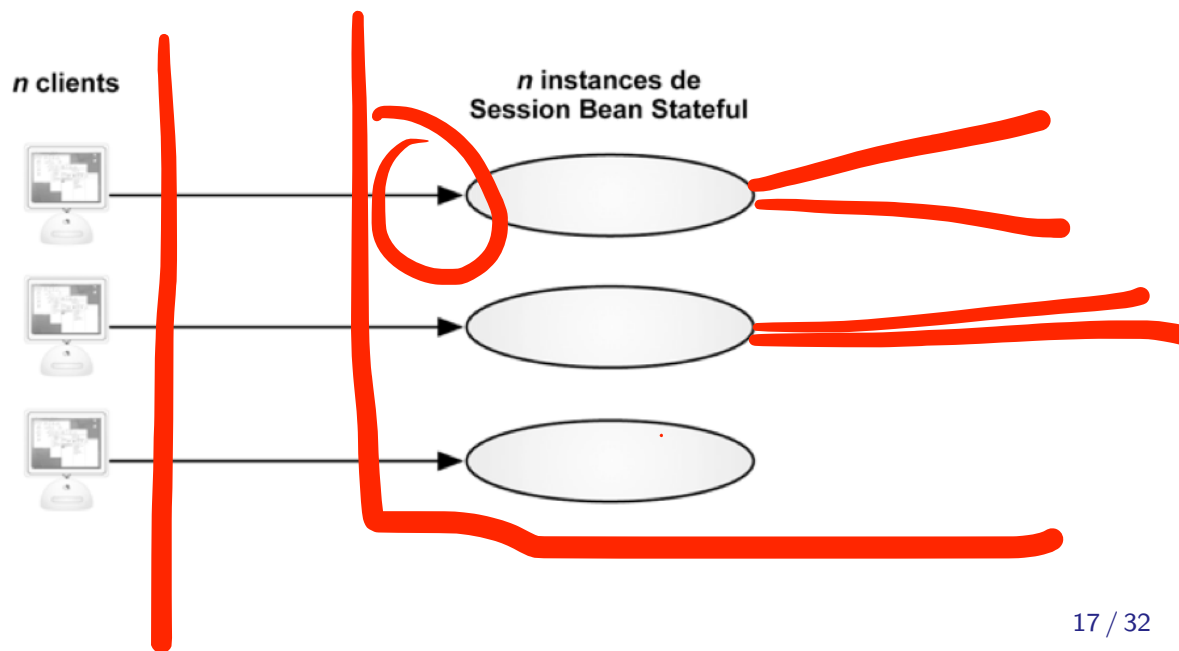
Un Stateful Session Bean :

- ▶ est une extension de l'application cliente.
- ▶ est partagé par toutes les méthodes pour un unique client.

Session Bean

Un Stateful Session Bean :

- ▶ est une extension de l'application cliente.
- ▶ est partagé par toutes les méthodes pour un unique client.
- ▶ est spécifique à l'application cliente.



Les Beans Session

Quand utiliser un session bean ?

- ▶ Représenter un processus appartenant à la logique métier.
- ▶ Partager des données entre clients.
- ▶ Besoin de persistance.
- ▶ Le service peut être accessible également via un service web.

Les Beans Session

Stateful :

- ▶ l'état du bean représente l'état de l'interaction entre le client et le bean.
- ▶ le bean doit conserver de l'information entre 2 invocations du client.
- ▶ le bean sert de médiateur entre le clients et d'autres beans de l'application (en général des entity beans).

Les Beans Session

Stateless :

- ▶ L'état du Bean n'a pas de données spécifiques à un client.
- ▶ pour des tâches génériques.
- ▶ pour consulter en lecture seule des données persistantes.


L'écriture d'un session bean

Un session bean se compose :


- ▶ d'une interface **Remote** : déclarer les méthodes accessible a **distance**.
- ▶ d'une interface **Local** : déclarer les méthodes accessible en **local**.
- ▶ la classe du bean : qui implémente les deux interfaces.

Les Beans Session : Exemple

```
public interface CalculSalaire {  
    public final static double tauxHoraire = 8.03 ;  
    public double getSalaire(int nbreHeures) ;  
}
```




```
import javax.ejb.Remote ;  
@Remote public interface CalculSalaireRemote  
    extends CalculSalaire {}
```



```
import javax.ejb.Local ;  
@Local public interface CalculSalaireLocal  
    extends CalculSalaire {}
```

Les Beans Session : Exemple

```
public interface CalculSalaire {  
    public final static double tauxHoraire = 8.03 ;  
    public double getSalaire(int nbreHeures) ;  
}
```




```
import javax.ejb.Remote ;  
@Remote public interface CalculSalaireRemote  
    extends CalculSalaire {}
```

```
import javax.ejb.Local ;  
@Local public interface CalculSalaireLocal  
    extends CalculSalaire {}
```

Ce composant est accessible en local et à distance (même fonctionnalités).

Les Beans Session sans état : Exemple




@Stateless

public class CalculSalaireBean

implements CalculSalaireRemote, CalculSalaireLocal {

public CalculSalaireBean() {}



public double getSalaire(**int** nbreHeures) {

return nbreHeures*tauxHoraire ;

}

}

Les Beans Session sans état : Exemple

```
@Stateless
public class CalculSalaireBean
    implements CalculSalaireRemote , CalculSalaireLocal {

    public CalculSalaireBean() {}

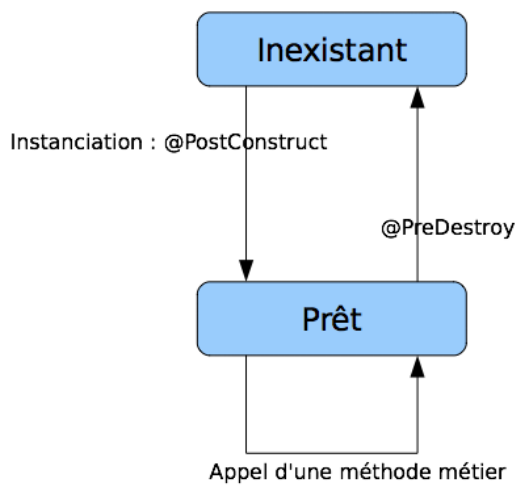
    public double getSalaire(int nbreHeures) {
        return nbreHeures*tauxHoraire ;
    }
}
```

Si le bean implémente une seule interface, elle doit être locale, si elle n'implémente aucune interface alors la classe joue le rôle d'interface locale.

Les Beans Session sans état : Cycle de vie

- ▶ Un Stateless est associé à un client que pour l'exécution d'une méthode.
- ▶ Il peut être associé successivement à plusieurs clients.
- ▶ Le cycle de vie est géré par les méthodes "callbacks".

Les Beans Session sans état : Cycle de vie

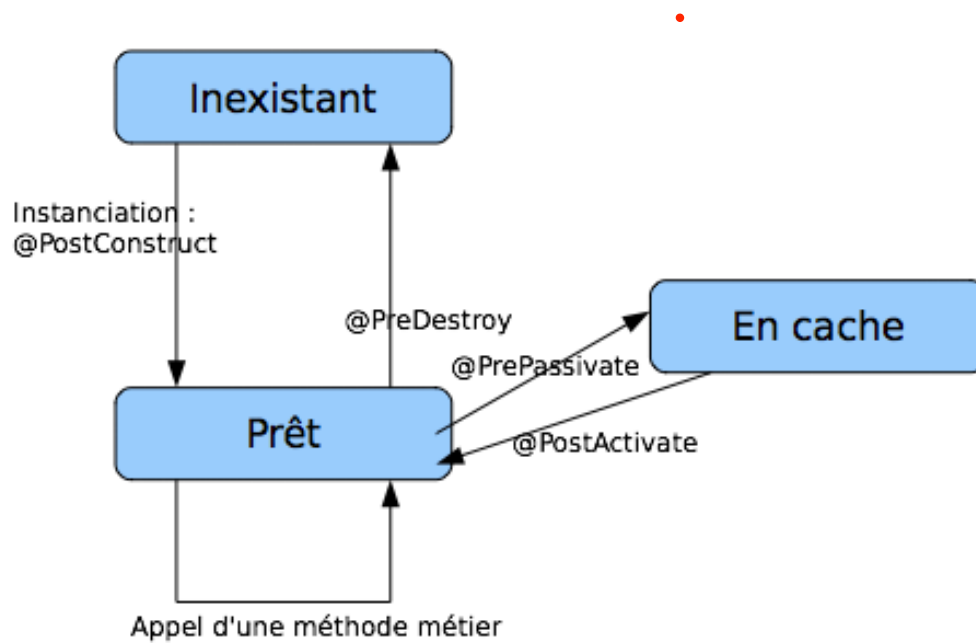


- ▶ les méthodes de la forme **void nomMethode()** ;
- ▶ utilisation d'annotation Java.
- ▶ **javax.annotation.PostConstruct** : automatiquement appelé après initialisation du composant.
- ▶ **javax.annotation.PreDestroy** : automatiquement appelé avant suppression du composant.

Les Beans Session avec état

- ▶ annotés par **javax.ejb.Stateful**
- ▶ après la première invocation de méthode, le composant est associé au client.
- ▶ une méthode annoté par **javax.ejb.Remove** notifie le conteneur que le client n'a plus besoin du bean.

Les Beans Session avec état : Cycle de vie



Les Beans Session avec état : Cycle de vie

`javax.annotation.PostConstruct` et

`javax.annotation.PreDestroy`

`javax.ejb.PrePassivate` : méthode automatiquement appelée avant la passivation du bean (charge mémoire)

`javax.ejb.PostActivate` : méthode automatiquement appelée après la restauration du bean

Les Beans Session avec état : Exemple

```
package ocaron.exempleStateful ;

import javax.ejb.Stateful ;
import javax.annotation.PostConstruct ;
import javax.annotation.PreDestroy ;
import javax.ejb.PrePassivate ;
import javax.ejb.PostActivate ;
import javax.ejb.Remove ;

@Stateful public class SalaireBean
    implements SalaireInterfaceRemote, SalaireInterfaceLocal {
    private double tauxHoraire=8.03 ;



    public SalaireBean() {}

    public void setTauxHoraire(double taux) {
        this.tauxHoraire=taux ;
    }
}
```

Les Beans Session avec état : Exemple

```
public double getTauxHoraire() { return this.tauxHoraire ; }
```

```
public double getSalaire(int nbreHeures) {  
    return this.getTauxHoraire()*nbreHeures ;  
}
```

```
 @PostActivate public void postActivate() {  
    System.out.println("PostActivate ...") ; }  
 @PrePassivate public void prePassivate() {  
    System.out.println("PrePassivate ...") ; }  
@PostConstruct public void postConstruct() {  
    System.out.println("PostConstruct ...") ; }  
@PreDestroy public void preDestroy() {  
    System.out.println("PreDestroy ...") ; }  
@Remove public void remove() {  
    System.out.println("Remove ...") ; }  
}
```

Exercice 1

- ▶ Écrire un composant EJB session qui permet de faire la conversion d'un montant en dinars dans d'autres monnaies (euro, dollars ..).
- ▶ Préciser le choix du session bean, en justifiant votre réponse.
- ▶ Définir les méthodes callbacks si nécessaire.

Exercice 2

- ▶ Écrire un composant EJB session qui permet de récupérer les données d'une formulaire et créer le compte du client associé.
- ▶ Préciser le choix du session bean, en justifiant votre réponse.
- ▶ Définir les méthodes callbacks si nécessaire.