Développement des systèmes d'information

JEE–06–Techonologie EJB

Parties 01 et 02

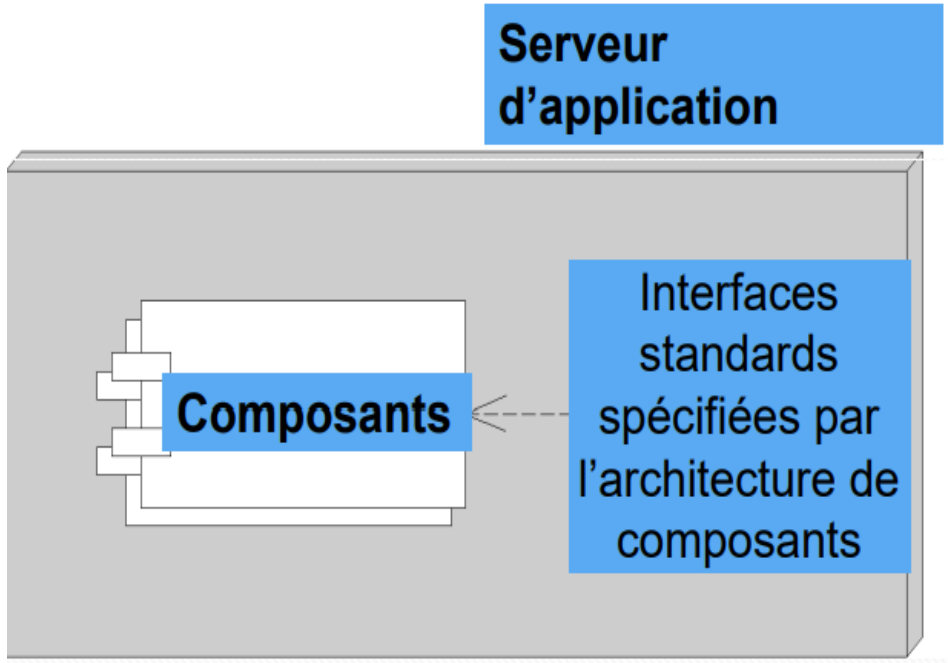
Mohamed ZAYANI & Sameh Hbaieb Turki

IPSAS-2023/2024

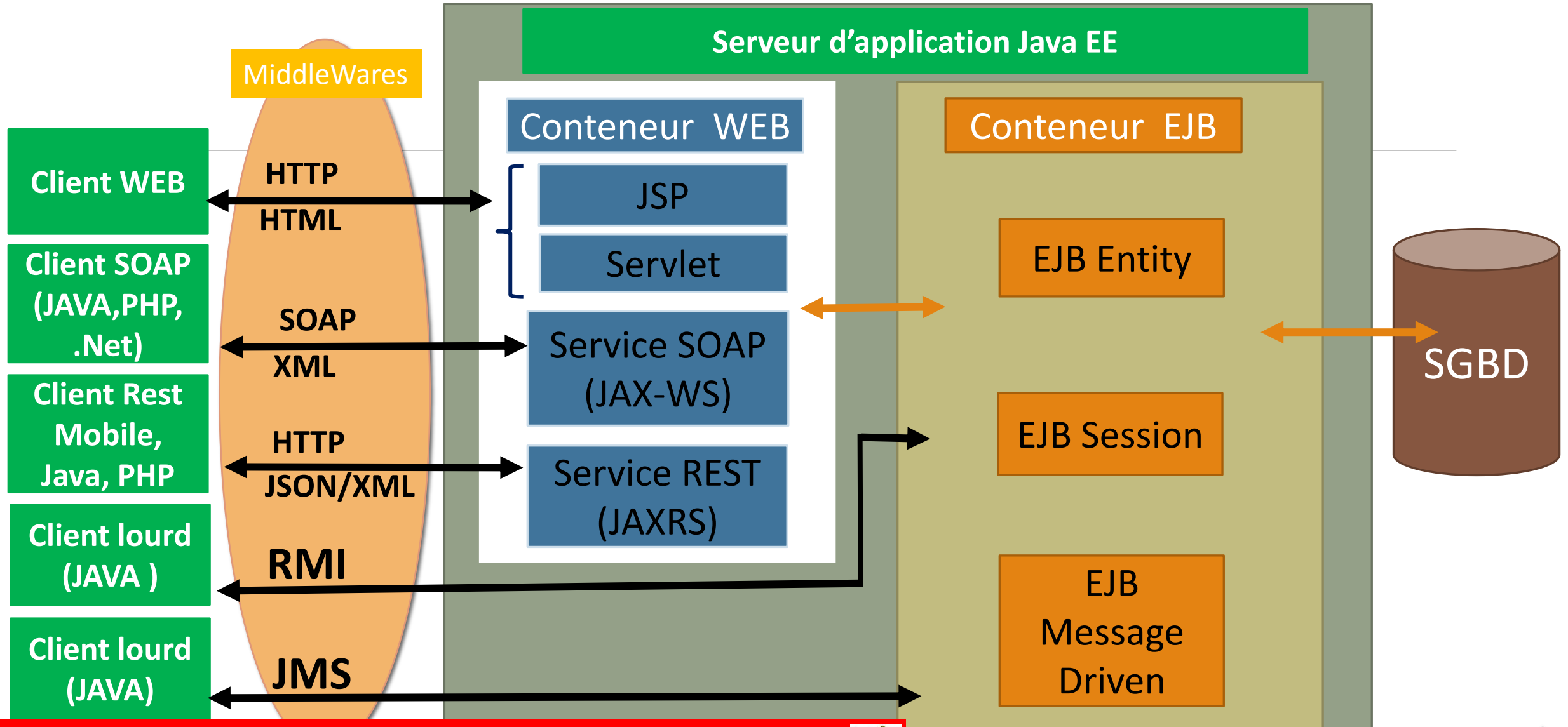
Partie 1. Les composants EJB

Architectures orientées composants

- C’est un ensemble de définitions d'interfaces entre le serveur d'application et les composants
- Chaque composant est géré par un conteneur spécifique.
- Le conteneur permet de **gérer le cycle de vie** de leurs composants: les compiler, les instancier, les détruire.
- Un tel standard s'appelle une architecture de composants



Architecture d'application JavaEE



Les EJB ne fournissent pas de GUI !
Le client ne communique pas directement avec le bean mais avec le contain

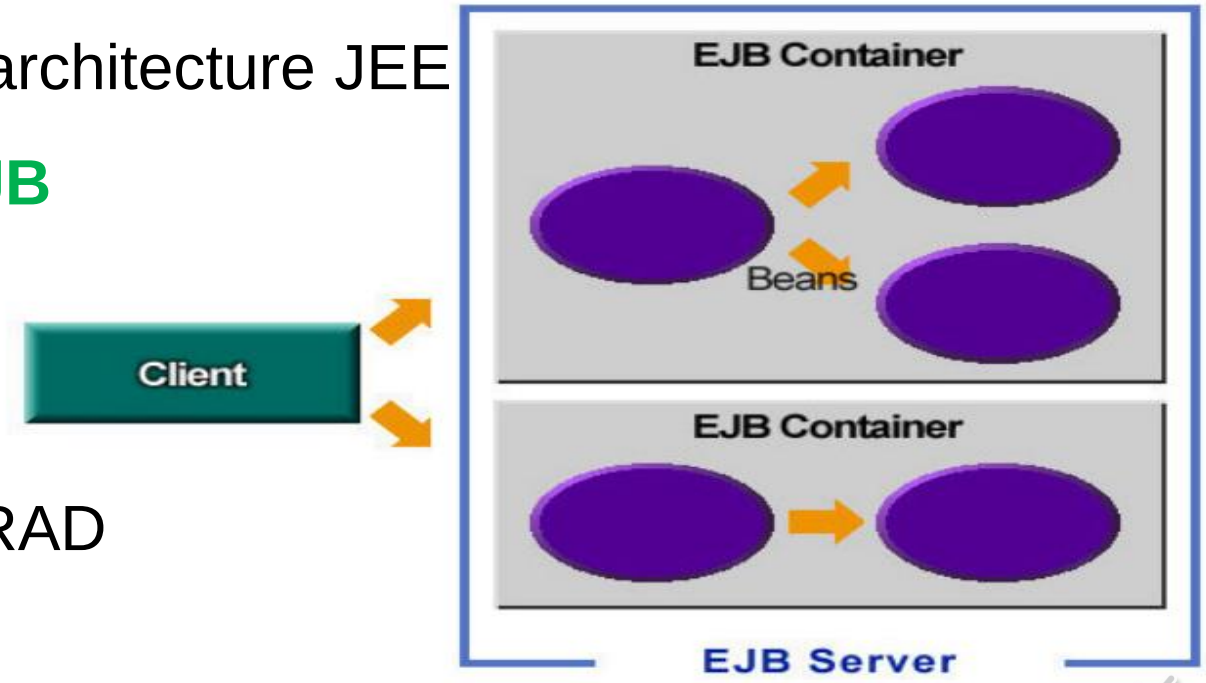
Qu’est ce que un EJB ?

- Un EJB est l’abréviation de Entreprise Java Bean

- Les EJB permettent d'implémenter des **objets métier** d’une manière souple et réutilisable
- Les EJB constituent la partie centrale de l’architecture JEE
- Les EJB **sont gérés par un conteneur EJB**

  ou bien Serveur EJB
- Facilement portables
- Bien adaptés pour le développement des RAD

  (Rapid Application Development)



Rôles des EJB

- EJB définit différents les rôles associés aux différentes parties intervenant dans le développement, le déploiement et l'exécution d'une application
- Les EJBs servent à :
  - Implémenter de la logique métier :
    - calcul des taxes sur un ensemble d'achats,
    - envoyer un mail de confirmation après une commande,…
  - Accéder à un SGBD
  - Accéder à un autre système d'information

Les conteneurs EJB

- L’architecture EJB repose sur la notion de conteneurs

- Les communications de l’extérieur avec les d'EJBs sont interceptées par le conteneur pour fournir un certain nombre de services :

  - Gérer le cycle de vie du bean
  - Gérer l’accès au bean
  - Sécurité d'accès
  - Accès concurrents
  - Transactions

Libère le développeur d’une grande charge de travail !

Dans quelles applications peut-on utiliser les EJBs ?

- Si l'application a au moins l'un de ces besoins:

1. Mise à l'échelle

- l'application doit supporter un grand nombre de clients
- L'application peut être distribuée sur plusieurs machines

2. L'intégrité des données doit être assurée par des transactions

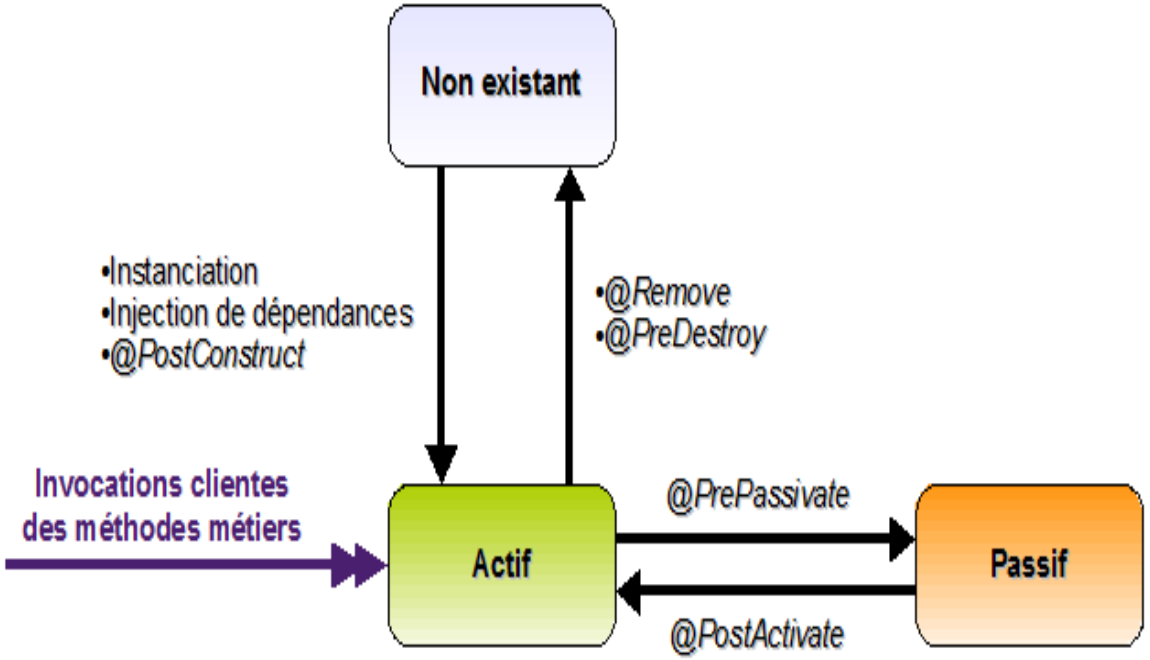
- Les EJB supportent les transactions (de façon relativement transparente)

3. Plusieurs types de clients

- La logique métier reste la même

Cycle de vie d'un EJB

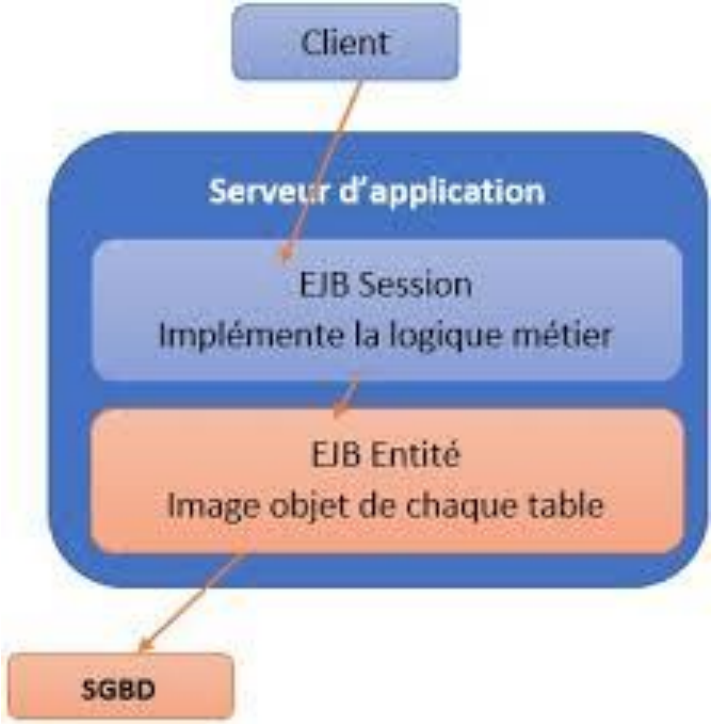
- Le conteneur gère le cycle de vie d'un bean, il fournit:



- L'administration du bean pour permettre aux clients de créer, détruire et rechercher un objet EJB
- L'appel des opérations correspondantes fournies par le bean (callbacks)
- La gestion de l' état: Activation (le bean est chargé en mémoire) et Désactivation (sauvegarder l' état du bean)

Les types des EJBs

❑ Trois types d'EJBs :



1. EJBs session : pour réaliser une tâche demandée par un client

- sans etat (@Stateless)
- Avec etat (@Stateful)
- Singleton (@Singleton)

2. EJBs Entity (@Entity) pour modéliser une classe persistante

3. EJB Message-Driven

(@MessageDriven) : pour les communications asynchrônes

Pour conclure (1/2)

- Les EJB permettent d'implémenter **la partie modèle** d'une application web JEE (selon MVC)

- Les EJB **facilitent** le développement d'application de **taille importante** en permettant au développeur de se concentrer sur l'implémentation des fonctionnalités de celle-ci.

- C'est **le conteneur d'EJB** qui se charge lui-même des aspects relatifs aux transactions et à la sécurité des composants métiers.

Pour conclure (2/2)

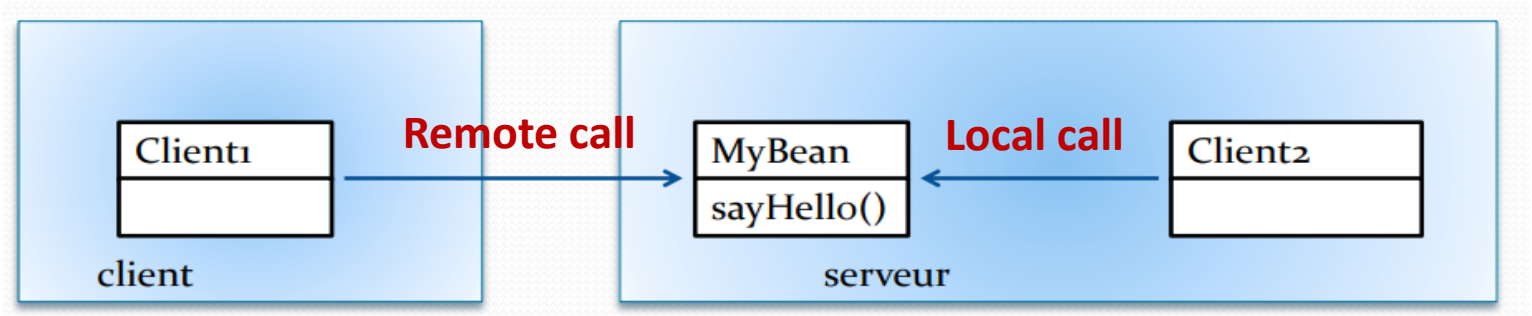
- Étant des composants **portables** et reposant sur des **API standards**, ils pourront s'exécuter sur n'importe quel serveur compatible Java EE.

- Les EJB sont particulièrement adaptés aux applications dont le nombre **d'utilisateurs peut croître conséquemment** car ils peuvent être distribués sur plusieurs serveurs et machines différentes, cette délocalisation restant transparente pour le client.

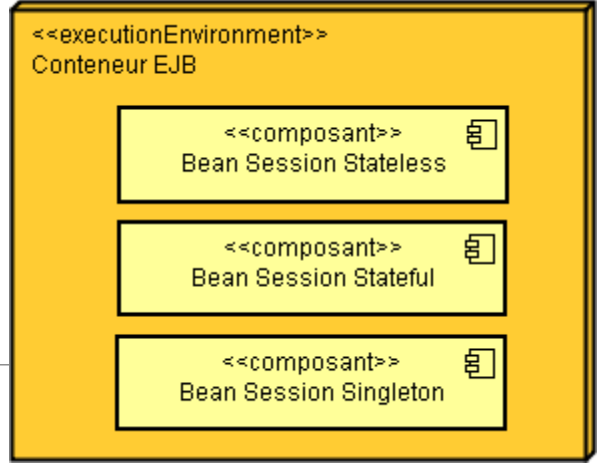
Partie 2. Les Session Beans

Qu’est ce qu’un Session Bean?

- Un Bean Session encapsule la logique métier (les fonctionnalités du système)
- C’est une classe Java **POJO** (Plain Old Java Object)
- Il peut être appelé par un client (par un autre code, ou une classe):
  - **A distance**, d’un autre container (remote) (sur une autre machine)
  - Ou **Localement** au container (sur la machine locale)



Les types de Session Bean



- **Sans état (Stateless)** : traite les tâches qui peuvent être accomplies en méthode: **pas d'état maintenu entre 2 appels de méthode**

Exemple : afficher la liste des comptes bancaires d'un client

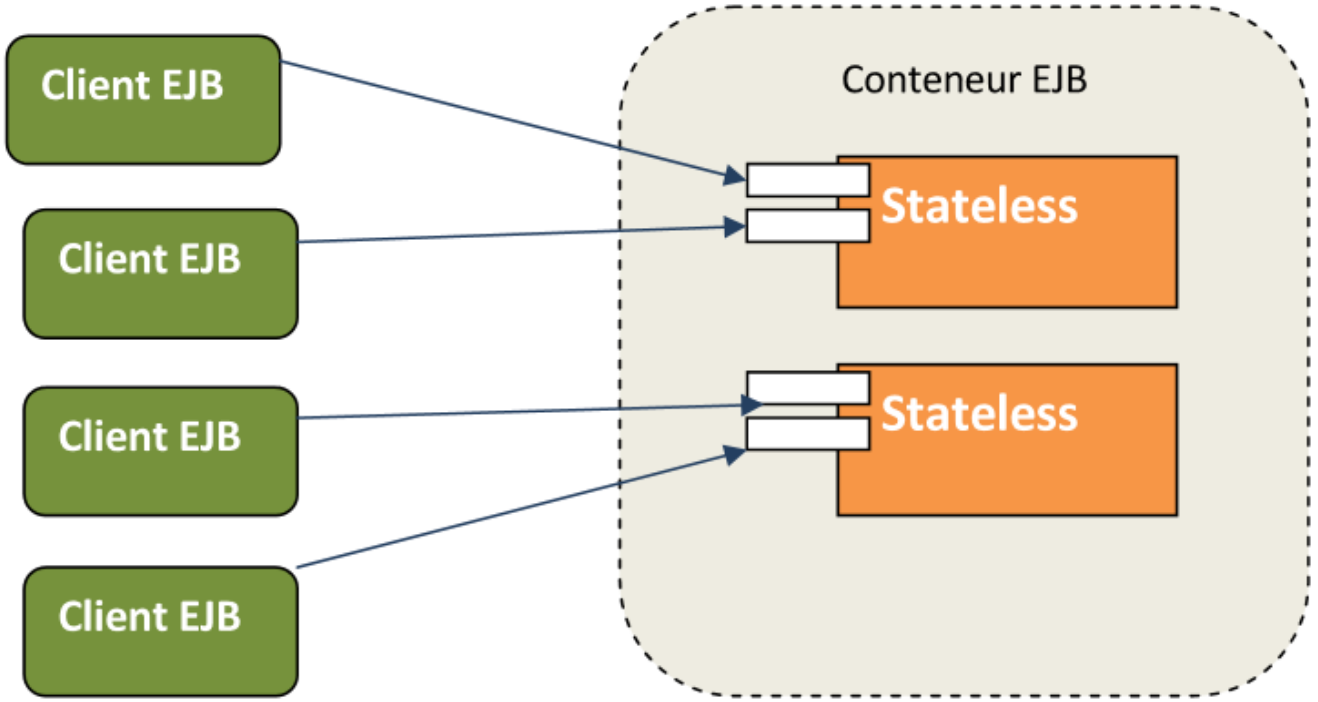
- **Avec état (Stateful)**: est associé à un seul client: maintient un état entre plusieurs appels de méthodes pour les tâches accomplies en plusieurs étapes

***Exemple** : remplir son caddy avec des articles dont les caractéristiques sont affichées sur des pages différentes*

- **Singleton :** Instancié une et une seule fois pour une application et existe pout toute la durée de vie de l'application

Exemple : **afficher une liste de pays (pour améliorer les performances)**

Stateless Session Bean



- Le client passe toutes les données nécessaires au traitement lors de l'appel de méthode
- Le container est responsable de la création et de la destruction du Bean
- Il peut le détruire juste après un appel de méthode, ou le garder en mémoire pendant un certain temps pour **réutilisation**.
- Une instance de Stateless Session Bean n'est pas propre à un client donné, elle peut être **partagée** entre chaque appel de méthode. Le serveur maintient un **pool de beans** sans état

Quand utiliser un Stateless Session Bean?

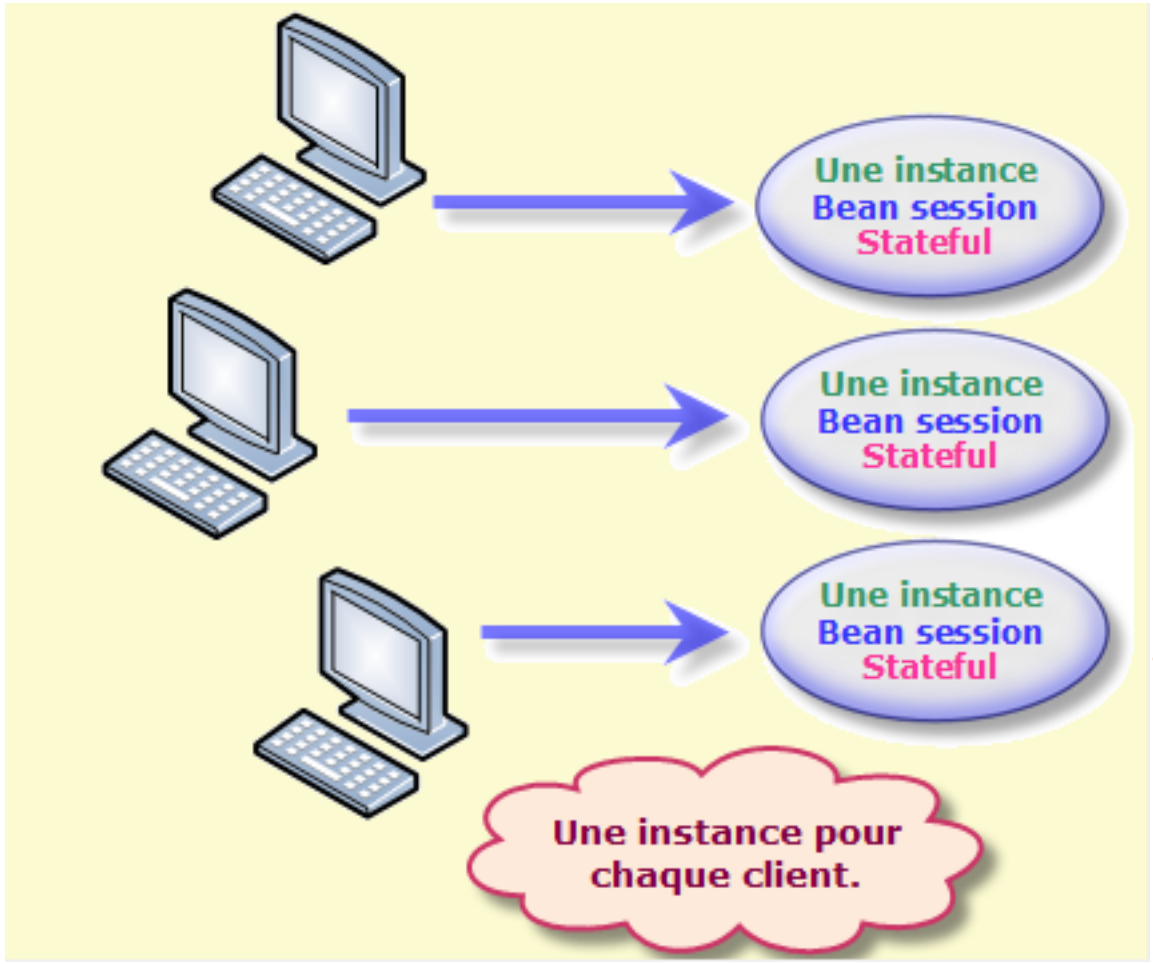
Si le bean à développer possède l'une des caractéristiques suivantes:

- L'état du bean est indépendant des clients: il ne possède pas de données spécifiques à un client particulier.

- Dans chaque méthode invoquée, le bean exécute une tâche générique à tous les clients.

Exemple: il est possible d'utiliser un stateless session bean pour envoyer un email de confirmation d'une commande.

Stateful Session Bean



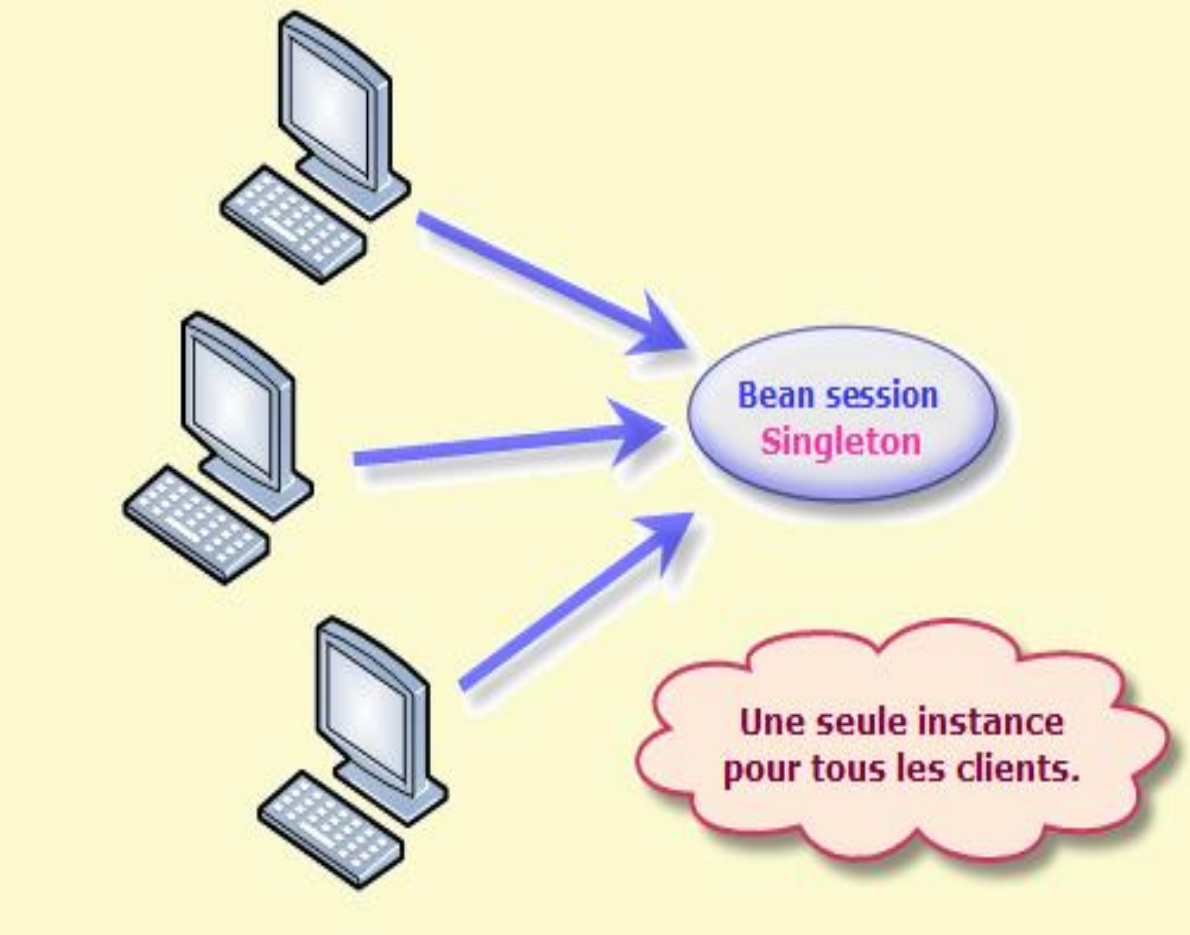
- Un Stateful Session Bean est utile pour maintenir un **état** pendant la durée de vie du client
  - Au cours d'appels de méthodes successifs
  - Au cours de transactions successives.
- Si un appel de méthode change l'état du Bean, lors d'un autre appel de méthode l'état sera disponible.
- **une instance de Stateful Session Bean par client.**

Quand utiliser un Stateful Session Bean?

Si le bean à developer remplit l'une des conditions suivantes:

- L'état du bean depend d'un client particulier
- Le bean a besoin des données spécifiques à un client pour l'invocation des methods
- Le bean joue le rôle de médiateur entre le client et un autre composant de l'application, présentant ainsi une vue simplifiée du client.

Singleton Session Bean



- Un Singleton est tout simplement un session bean qui n'est instancié qu'une seule fois dans l'application.

- Il garantit qu'une seule instance d'une classe existera dans l'application et fournit un point d'accès global vers cette classe.

Quand utiliser un Singleton Session Bean?

- Le besoin de partager l'état du bean au sein de l'application.

- Un seul bean doit être accessible par plusieurs threads de manière concurrente.

- Le bean implémente un web service

- Le besoin de réaliser des tâches particulières au démarrage et à la fermeture de l'application

Le développement d'un session bean

- Dans les versions antérieures à la version 3.0 des EJB, le développeur était obligé de créer de nombreuses entités pour respecter l'API EJB: création d'une interface locale (pour les clients locaux), d'une interface distante (pour les clients distants) et d'une classe bean qui implémente ces interfaces.

- Une interface représente la vue du bean pour le client.

- Depuis la version 3.1, ces interfaces sont maintenant optionnelles pour les session beans.

Le développement d’un session bean

- Un EJB de type session est maintenant une simple classe, qui peut implémenter une interface métier.

- Le descripteur de déploiement n'est plus obligatoire puisqu'il peut être remplacé par l'utilisation d'annotations dédiées directement dans les classes des EJB.

- Pour créer un EJB, il est obligatoire de préciser son type à travers les annotations suivantes **(@javax.ejb.Stateless**, @javax.ejb.Stateful ou @javax.ejb.MessageDriven).

Création d'un Stateless session bean

```
import javax.ejb.Stateless;

@Stateless
public class Conversion {
    private final double TAUX = 3.2;

    public double euroToDinar(double euro) {
        return euro*TAUX;
    }

    public double dinarToEuro(double dinar) {
        return dinar/TAUX;
    }}
```

Bean partagé par plusieurs clients

Création d'une Stateful session bean

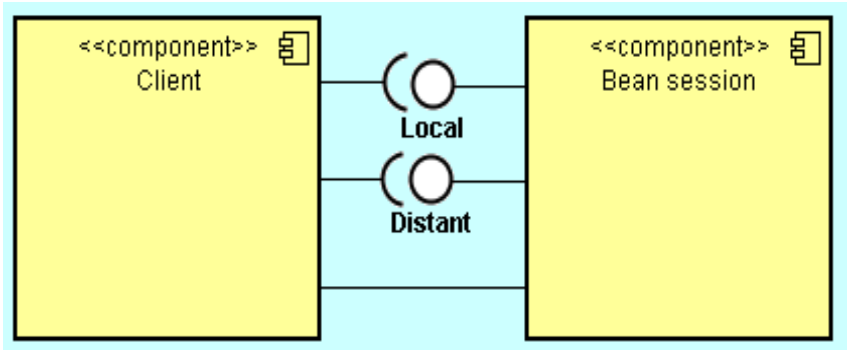
```
import javax.ejb.Stateful;

@Stateful      ←——— Bean pour chaque client

public class Caddy {
    private List<Article> articles = new ArrayList<Article>();
       public void ajouterArticle(Article article) {
        if (!articles.contains(article)) articles.add(article);
    }
    public void enleverArticle(Article article) {
        if (articles.contains(article)) articles.remove(article);
    }
    public double getTotal() {
        if (articles == null || articles.isEmpty()) return 0.0;
        double total = 0.0;
        for (Article article : articles)  total += article.getPrix();
        return total;
    } }
```

Interface Locale / Interface Remote

- En réalité, une application cliente peut accéder à un bean session par l'une des interfaces (locale ou distante) ou indirectement en invoquant la classe elle-même.
- Un bean session peut implémenter plusieurs interfaces ou aucune.
- Une interface métier est une interface classique Java qui n'hérite d'aucune interface EJB spécifique.
- Comme toute interface Java, les interfaces métiers énumèrent les seules méthodes qui seront disponibles pour l'application cliente.



Interface Locale / Interface Remote

- Une interface métier utilise l'une des annotations suivantes:

@Remote : permet un accès à l'EJB depuis un client hors de la JVM

@Local : permet un accès à l'EJB depuis un client dans la même JVM que celle de l'EJB

- Par défaut, l'interface d'appel est locale si aucune annotation n'est indiquée.

Exemple d'implémentation d'interfaces

❑ Exemple:

```
import javax.ejb.Remote;
@Remote
public interface ConversionRemote
 {
   double euroFranc(double euro);
   double francEuro(double franc);
   double getTaux(); }
```

```
@Local
public interface ConversionLocal
{
   double euroFranc(double euro);
   double francEuro(double franc);
}
```

```
import javax.ejb.Stateless;
import java.text.*;
@Stateless
public class ConversionBean implements ConversionRemote,
ConversionLocal {
   private final double TAUX = 6.55957;
   public double euroFranc(double euro) {
       return euro*TAUX;
    }
   public double francEuro(double franc) {
       return franc/TAUX;
    }
    public double getTaux() {
       return TAUX;
    } }
```

La méthode getTaux() ne pourra être appelée que par des clients distants

Accès à un session bean

- Pour appeler une méthode d'un bean session, un client n'instancie pas directement le bean avec l'opérateur new.

- Il peut obtenir une instance via l'injection de dépendances (avec l'annotation @EJB) ou par une recherche JNDI:

1. **@EJB** : Les EJB utilisent l'injection de dépendances pour accéder à différents types de ressources (autres EJB, ressources d'environnement, etc.). Dans ce modèle, le conteneur pousse les données dans le bean. L'injection a lieu lors du déploiement.

2. **JNDI** : JNDI est une API permettant d'accéder à différents types de services d'annuaires, elle permet au client de lier et de rechercher des objets par leur nom.

Accès local par injection de dépendance @EJB

- L'injection de dépendances n'est possible que dans des environnements gérés, comme les conteneurs EJB, les conteneurs Web et les conteneurs clients d'application

- **Cas 1**: Le client obtient une référence à la classe Conversion en utilisant l'annotation @EJB

```
// Côté serveur
@Stateless
public class Conversion {
...
}
```

```
// Code client
@EJB Conversion convertir;
```

Accès local par injection de dépendance @EJB

❑ **Cas 2**: Le bean ConversionBean implémente deux interfaces

```
// Côté serveur
@Stateless
public class ConversionBean implements
ConversionRemote, ConversionLocal {
...
}
```

```
// Code client
@EJB ConversionBean convertir; // ATTENTION,
interdit !
@EJB ConversionRemote convertirRemote;
@EJB ConversionLocal convertirLocal;
```

Accès local par injection de dépendance @EJB

❑ **Cas 3**: Le bean ConversionBean implémente deux interfaces

Il faut préciser le type d'accès @LocalBean / @RemoteBean

```
// Côté serveur
@Stateless
@LocalBean
public class ConversionBean implements ConversionRemote,
ConversionLocal {
...
}
```

```
// Code client
@EJB ConversionBean convertir; // Maintenant cette écriture est autorisée
@EJB ConversionRemote convertirRemote;
@EJB ConversionLocal convertirLocal;
```

Accès local par injection de dépendance @EJB

- **Cas 4**: Accès local implicite

```
// Côté serveur
@Stateless
@LocalBean
public class Conversion implements ConversionRemote {
...
}
```

```
// Code client
@EJB Conversion convertir;
@EJB ConversionRemote convertirRemote;
```

Accès global par JNDI

- Tous les EJB de type Session sont enregistrés dans un annuaire avec **un nom unique** accessible par un client via un contexte JNDI.
- Le nom JNDI d'un session bean prend diverses formes :

Exemple:

```
ejb:[/<nom-application>]/<nom-module>/<nom-bean>!<nom-interface>
```

Accès global par JNDI

- **<nom-application>** : C'est le nom de l'application dans lequel le bean est packagé. Le type de fichier d'une application qui doit être déployé dans le serveur d'application porte l'extension "ear".

- **<nom-module>** : est le nom du module dans lequel l'EJB est packagé

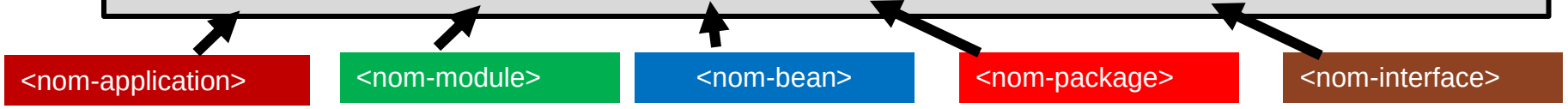
- **<nom-bean>** : est le nom du bean session. Par défaut, c'est le nom de la classe d'implémentation de l'EJB sauf si le nom est précisé via l'attribut name de l'annotation @Stateless, @Stateful et @Singleton.

- **<nom-inteface>** : est le nom pleinement qualifié de l'interface métier sous laquelle l'EJB est exposé. Dans le cas des vues sans interface, ce nom est le nom pleinement qualifié de la classe du bean.

Accès global par JNDI

Exemple1: Accès au nom JNDI d'un session bean dans un projet EJB nommé « **ejbModule** » qui est intégré dans une application «**ejbApp** »

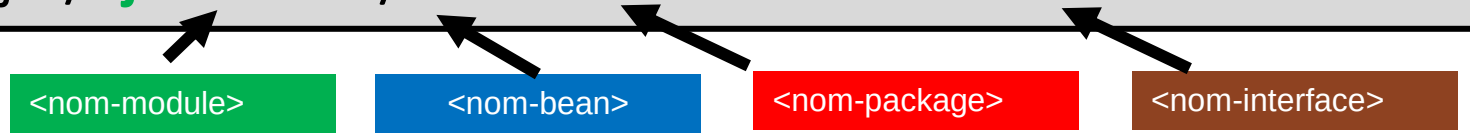
```
/ Côté serveur
package metier;
@Stateless (name =« CB")
@LocalBean
public class ConversionBean implements ConversionRemote, ConversionLocal  {
...
}
// nom JNDI associé
ejb:/ejbApp/ejbModule/CB!metier.ConversionRemote
ejb:/ejbApp/ejbModule/CB!metier.ConversionLocal
```



Accès global par JNDI

Exemple2 : Accès au nom JNDI d'un session bean dans un projet EJB nommé « **ejbModule** » et non intégré dans une application

```
/ Côté serveur
package metier;
@Stateless (name =« CB")
@LocalBean
public class ConversionBean implements ConversionRemote, ConversionLocal {
...
}
// nom JNDI associé
ejb:/ejbModule/CB!metier.ConversionRemote
ejb:/ejbModule/CB!metier.ConversionLocal
```



Comment accéder à un service d'un session bean ?

- La première chose à faire est de récupérer une instance de l'EJB au moyen de **JNDI**

- Il est alors nécessaire de faire appel à la classe **InitialContext** dont le but est de récupérer les informations relatives au serveur d'applications

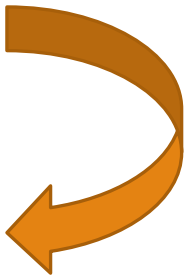
- Pour appeler un EJB Session, un client distant doit passer par trois étapes :

1. Récupération de la référence de l'annuaire JNDI
2. Recherche du bean dans l'annuaire
3. Appel des méthodes du bean

Comment accéder à un service d'un session bean ?

Exemple : L'interface et le bean session sont définis dans un package « metier » dans un projet EJB nommé « **ejbModule** » et non intégré dans une application

```
// Code serveur
@Remote
public interface InterfaceName { ... }
@Remote(InterfaceName.class)
@Stateless (name =« BN »)
public class BeanName implements InterfaceName { ...
}
```



Appel de SessionBean

```
// Code client
javax.naming.Context InitialContext =new javax.naming.InitialContext();   1
InterfaceName bean=(InterfaceName ) InitialContext.lookup("ejb:/ejbModule/BN!metier. InterfaceName");   2
Bean.method();   3
```

Exercice pratique

Question 1

Écrire un composant EJB session sans état qui permet de faire la conversion d'un montant du dinar en euro et vise-versa

Question 2

Réaliser une page JSP qui contient un formulaire de saisie du montant et du choix de la monnaie actuelle et la monnaie cible et qui appelle la méthode de conversion réalisée par la première question (en passant par l'intermédiaire d'une servlet).

Le résultat de la conversion est affiché sur la même JSP.

Question 3

Réaliser une page JSP qui permet d'ajouter, chaque fois, un montant au solde d'un client donné. Le solde est initialisé à 0. Utiliser un EJB Session pour réaliser la partie métier et une servlet pour réaliser l'appel de l'EJB Session.

La page JSP doit actualiser suite à chaque ajout la valeur du solde du client actuel.