Enterprise Java Bean

Pr. Youssef Saadi

Master Informatique Décisionnelle

Faculté Des Sciences Et Techniques
Université Sultan Moulay Slimane Béni-Mellal

AU: 2019/2020

Idée essentielle

- Se concentrer sur la logique métier à développer (=modéliser, coder les notions du domaine, …) et sous traiter les problèmes connus de :
  - **Persistance**
  - **Transactions**
  - **Sécurité** (authentification, confidentialité, etc.)
  - **Réserve** (pool) d'objets, **équilibrage** de charge.
- à un conteneur.
  - Donc fabriquer des composants qui s'intégreront bien entre eux et avec le conteneur;
  - C'est le mariage entre le monde transactionnel et le monde des composants orienté objet;
  - Version 3.0 depuis le 27 juin 2005;
  - Compatibilité et interopérabilité avec les EJB 2.1 ;

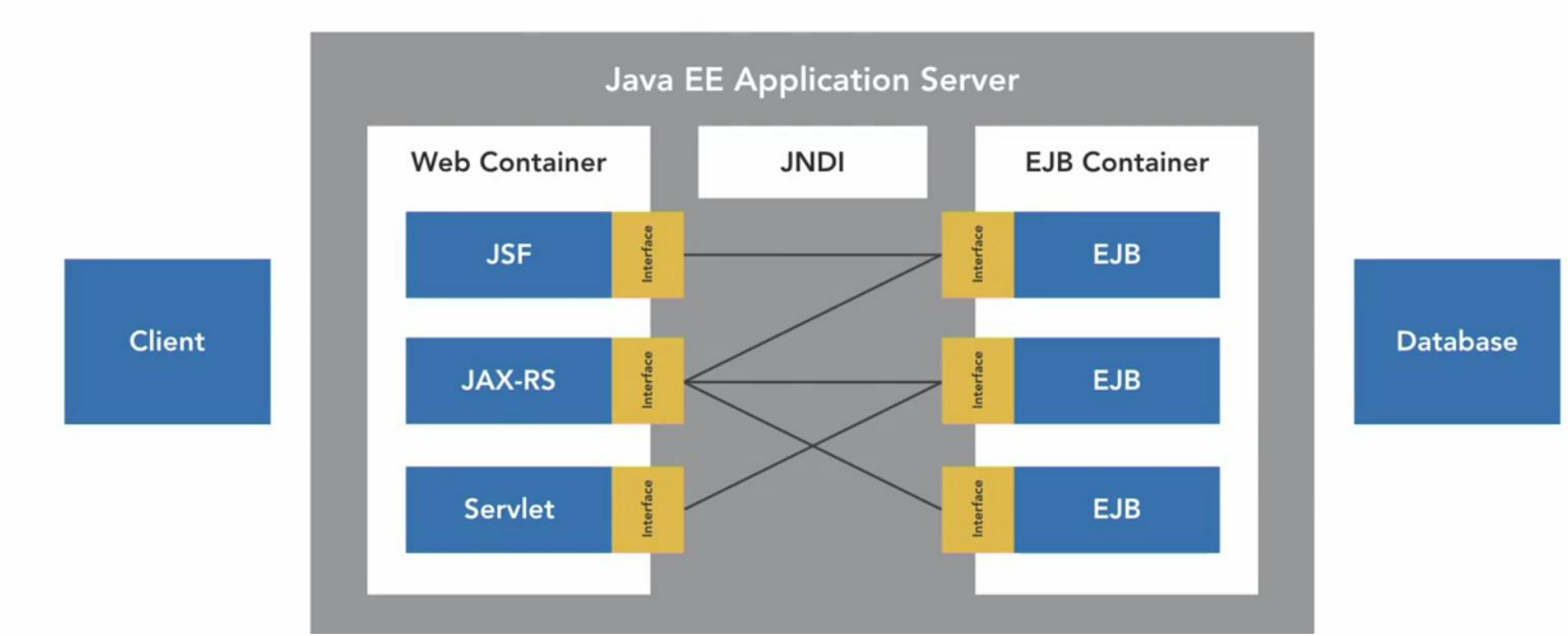
EJB et Logique Métier

- 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 (CICS, COBOL,…);
  - Applications web : intégration avec JSF/Servlets;
  - Web services basés sur XML (SOAP, UDDI,…) ;

Le conteneur d'EJB

- Les serveurs d'application contiennent des conteneurs;
- Un conteneur est l'environnement d'exécution des composants;
- Il gère l'interface entre les composants Java EE et les fonctionnalités bas-niveau :
  - multithreading,
  - le cache mémoire,
  - la sécurité,
  - l'accès aux données
  - etc.

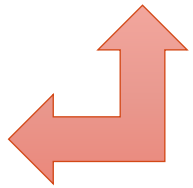
Architecture des EJB



Les types d'EJB

- On trouve trois types d'EJBs :
- **EJBs session** :
  - Sans état (@**Stateless**)
  - Avec état (@**Stateful**)
  - Singleton (@**Singleton**)
- **EJBs Entity** (@**Entity**)
- **EJB Message-Driven** (@MessageDriven):
  - Ne sera pas abordé au niveau de ce cours
- Java EE 6 repose à tous les niveaux sur de l'injection de code via des annotations de code.

Souvent, on ne fera pas de « new », les variables seront créées/initialisées par injection de code.



Session Bean

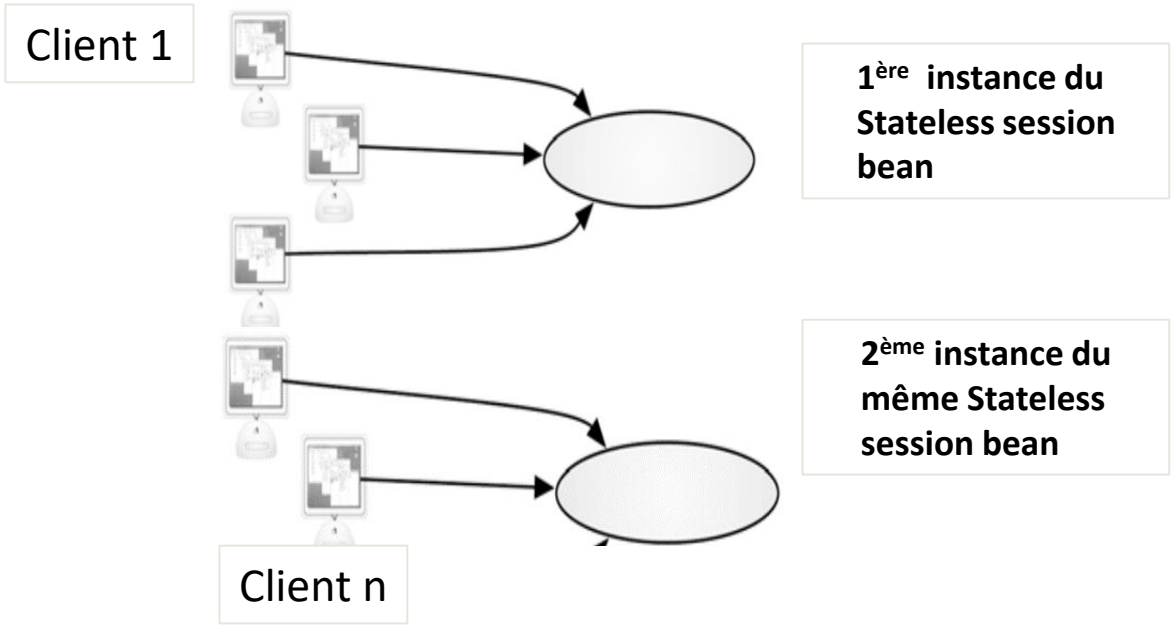
- Les principes fondamentaux de l'architecture métier définissent la création de services en tant qu'intermédiaires entre les applications clientes et l'accès aux données.
- Au sein d'une architecture Java EE, ce sont des EJB qui rempliront cette fonction : les *Session Beans*.
- Un Session Bean représente :
  - Une action, un verbe, une logique métier, un algorithme, Un enchaînement de tâches… Exemples:
    - Saisie d'une commande,
    - Compression vidéo,
    - Gestion d'un caddy, d'un catalogue de produits,
    - Transactions bancaires…

Session Bean

- **Sans état** : traite les tâches qui peuvent être accomplies en un seul appel de méthode ; pas d'état maintenu entre 2 appels de méthode
  - Exemple : afficher la liste des comptes bancaires d'un client.
- **Avec état** : 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** : quand on veut être assuré qu'il n'y a qu'une seule instance du bean pour tous les utilisateurs de l'application
  - Exemple : cache d'une liste de pays (pour améliorer les performances)

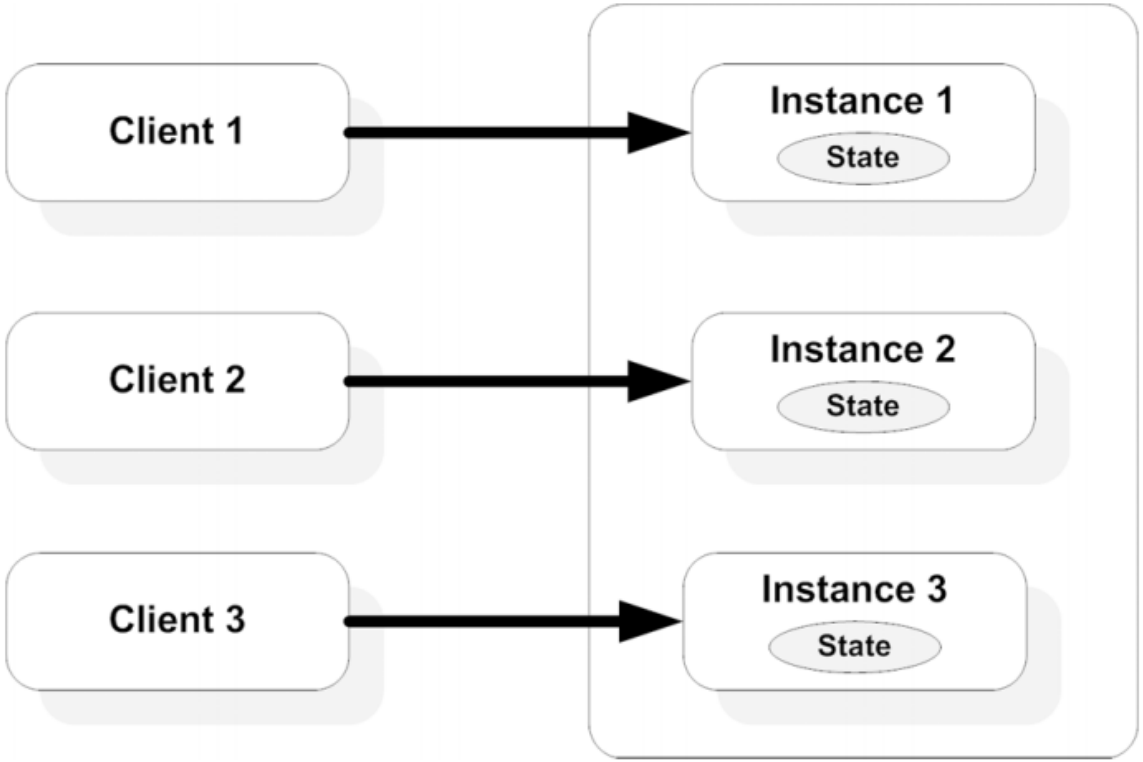
Stateless Bean (Bean sans état)

- Un Stateless Session Bean est une collection de services dont chacun est représenté par une méthode.
- Stateless : aucun état n'est conservé entre deux invocations de méthodes.
- L'avantage du type *Stateless* est sa performance. En effet, plusieurs clients utilisent la même instance de l'EJB.



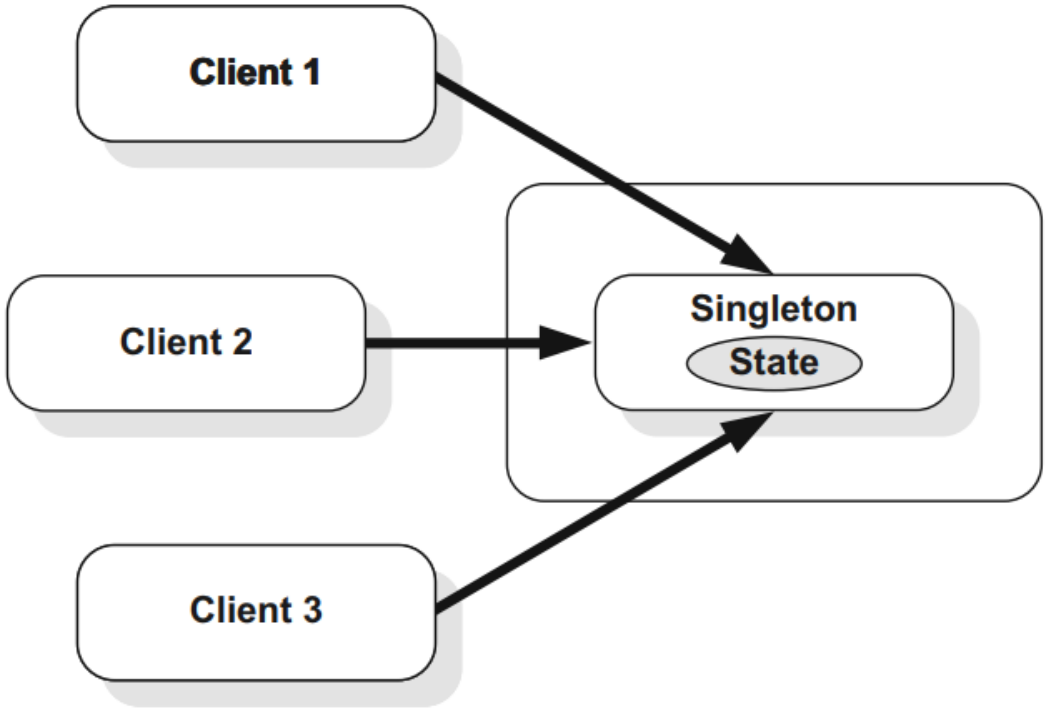
Stateful Session Bean (avec état)

- Un Stateful Session Bean introduit le concept de session entre le client et le serveur.
- Cet EJB est partagé par toutes les méthodes pour un unique client. Chaque client est lié à une instance de l'EJB.



Singleton Bean

- Un EJB session **singleton** est un objet java situé côté serveur qui est instancié une seule fois et cette instance est fonctionnelle durant tout le cycle de vie de l'application au sein de laquelle il est développé.



EJB3

- On utilise les annotations @Remote ou @Local pour définir respectivement si le Session Bean fournit les méthodes à des clients distants ou locaux.
- Il en est de même pour les types de Session Bean, grâce aux annotations **@Stateless** ou **@Stateful**.

EJB 3 : Session Bean

```java
package com.youssadi.tutot.ejb;

import javax.ejb.Stateless;

/**
 * Session Bean implementation class HelloBean
 */
@Stateless(mappedName = "HB")
public class HelloBean implements HelloBeanRemote, HelloBeanLocal {

    /**
     * Default constructor.
     */
    public HelloBean() {
        // TODO Auto-generated constructor stub
    }

    @Override
    public String processText(String txt) {
        return txt.toUpperCase();
    }

}
```

```java
package com.youssadi.tutot.ejb;

import javax.ejb.Local;

@Local
public interface HelloBeanLocal {
    public String processText(String txt);

}
```

```java
package com.youssadi.tutot.ejb;

import javax.ejb.Remote;

@Remote
public interface HelloBeanRemote {
    public String processText(String txt);
}
```

EJB Session : LocalBean

```java
import javax.ejb.LocalBean;

/**
 * Session Bean implementation class HelloEJB
 */
@Stateless(mappedName = "hellobean")
@LocalBean
public class HelloEJB {

    /**
     * Default constructor.
     */
    public HelloEJB() {
        // TODO Auto-generated constructor stub
    }
    public String sayHelloTo(String name) {
        return "hello Mr " + name;
    }

}
```

Cycle de vie d'un Session Bean

- Pour exploiter un Session Bean celui-ci doit être instancié. De façon très simplifiée, le conteneur s'occupe de cette instanciation par l'appel de la méthode **newInstance**() → HelloBean.**class.newInstance();**
- Le conteneur va ensuite analyser cette instance afin de déceler les éventuelles injections de dépendance à effectuer.
- **L'injection de dépendance** est le processus qui permet au conteneur d'initialiser des propriétés de l'EJB automatiquement.
- Le développeur spécifie ces dépendances grâce à des annotations telles que **@EJB**, **@Resource**, …
- Après cela, le conteneur va éventuellement exécuter les méthodes *callback interceptors* annotées avec **@PostConstruct, @PreDestroy**.
- Les méthodes *callback interceptors* doivent être annotées avec les annotations de *callback*.

Cycle de vie d'un Session Bean

- Les méthodes beginShopping() et endShopping() sont appelées respectivement, par le conteneur, après la création de l'instance du Bean et juste avant sa suppression.

```
@PostConstruct
public void beginShopping() {...};

@PreDestroy
public void endShopping() {...};

...
```

Cycle de vie : Passivation & Activation

- **Passivation** : pour économiser la mémoire, le serveur d'application peut retirer temporairement de la mémoire centrale les Beans sessions avec état pour les placer sur le disque.
- **Activation** : le bean sera remis en mémoire dès que possible quand les clients en auront besoin:
  - Pendant la passivation il est bon de libérer les ressources utilisées par le bean (connexions avec la BD par exemple)
  - Au moment de l'activation, il faut alors récupérer ces ressources.

Activation/Passivation callbacks

- Lorsqu'un bean va être mis en passivation, le container peut l'avertir **(@PrePassivate)**:
  - peut libérer des ressources (connexions…)
- Idem lorsque le bean vient d'être activé **(@PostActivate)**

```
@Stateful
public class MyBean  {
    @PrePassivate
    public void passivate() {
        <close socket connections, etc...
    }
    ...
}
```

```
@Stateful
 public class MyBean  {
    @PostActivate
    public void activate() {
        <open socket connections, etc...>
    }
    ...
}
```

Cycle de vie d'un Session Bean

- À n'importe quel instant, seulement un client a accès à l'instance du Session Bean.
- L'état du Bean **n'est pas persistant**, il n'existe que pour une courte durée (environ quelques heures). Le service peut être accessible également *via* un **service web**.
- La fin de l'utilisation de l'instance d'un **Stateful bean** est donné par un appel à une méthode annotée **@Remove.**

Cycle de vie d'un Session Bean

```
@Stateful
@StatefulTimeout(300000) // 5 minutes
public class CaddyEJB {
   private List<Item> caddy =
      new ArrayList<Item>();
   public void addItem(Item item) {
      ...
   }
   @Remove
   public void checkout() {
      caddy.clear();
   }
   ...
}
```

Avec Item un java Bean.

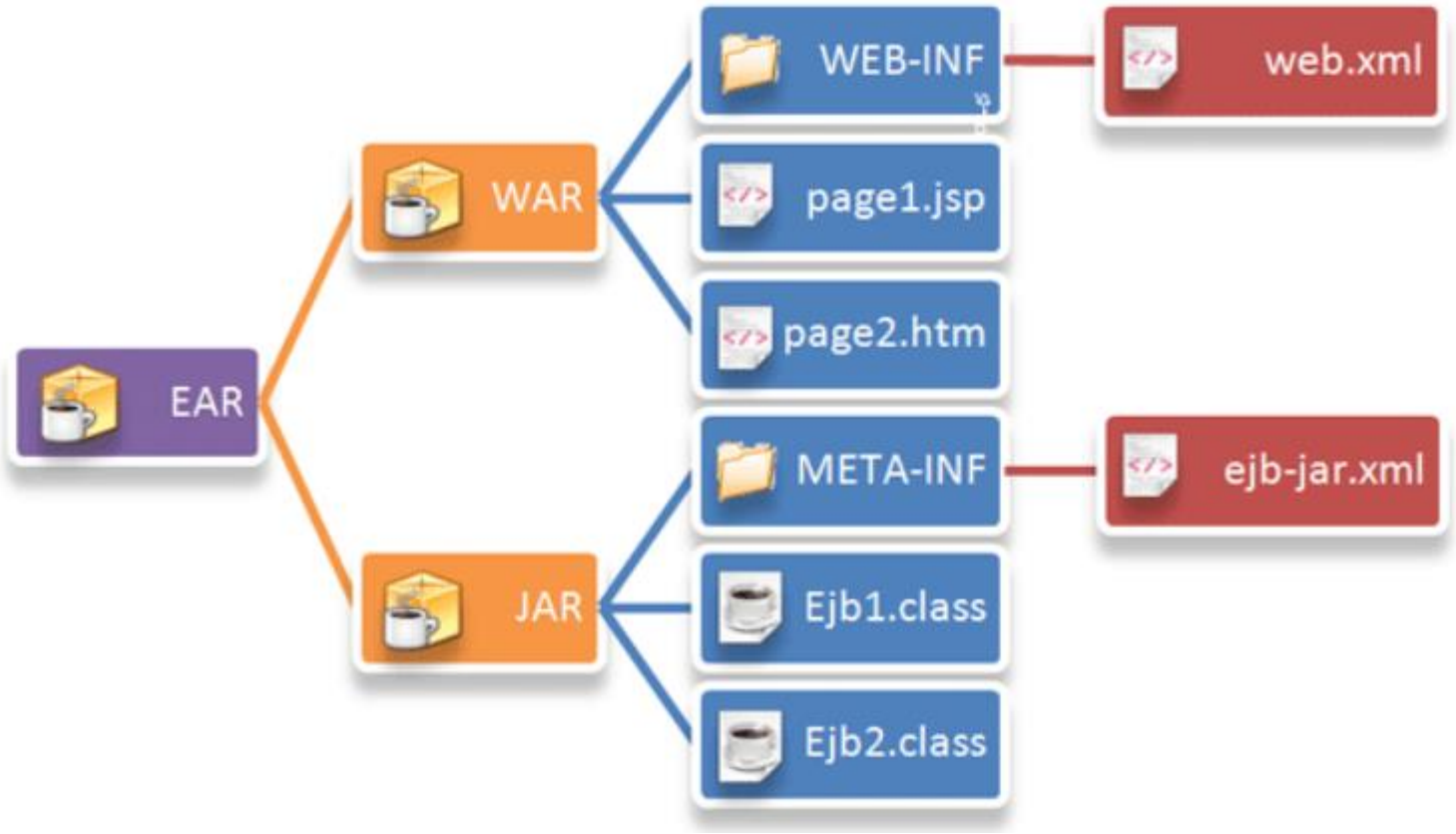
Les interfaces métier

- Leurs méthodes doivent respecter les règles suivantes :
  - Leur **nom** est **libre** tant qu'il ne commence pas par « ejb » pour éviter tout conflit avec les méthodes *callback interceptors*.
  - Elles doivent être déclarées comme étant **public**.
  - Elles ne doivent pas être déclarées comme **final** ou **static**.
  - Dans le cas d'un accès de type ***remote***, le type de retour et le type des arguments doivent être compatibles avec des transports de type **RMI/IIOP**.
  - De même, dans le cas d'un service web, le type de retour et des arguments doivent être compatibles avec un transport de type **JAX-WS/JAX-RPC**.

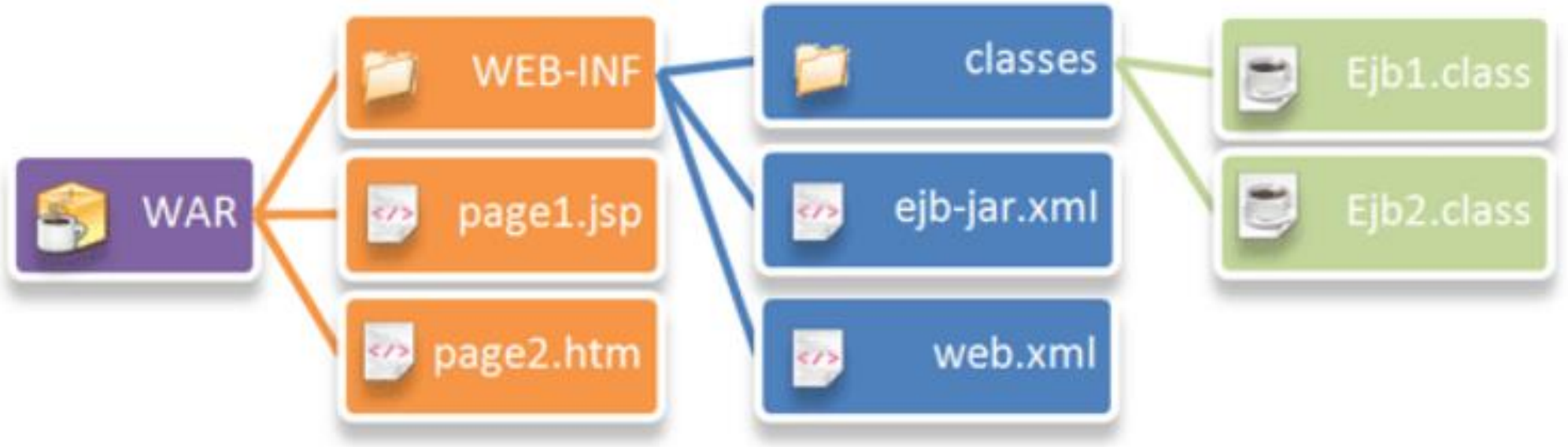
Packaging

- Une fois les différents EJB session conçus et développés, il reste des étapes supplémentaires **avant de passer à leur utilisation** : packaging et déploiement.
- Le packaging consiste à créer une brique contenant toutes les informations concernant les EJB de l'application JEE. Il est convenu de les placer dans une archive « **ejb-jar** ».
- Un fichier de type « **ejb-jar** » doit contenir les fichiers suivants :
  - **Toute classe d'EJB** (Session Bean, Entity Bean, Message-driven Bean);
  - **Toute interface d'EJB** et **interface** de **service web;**
  - Toute **classe d'intercepteur**;
  - Toute **classe** correspondant à la **clé primaire** d'un EJB entité;
  - Le **descripteur** de **déploiement** des EJB dans le fichier « META-INF/**ejbjar.xml** ». Ce fichier XML contient la configuration des différents EJB et permettra leur exploitation au sein de l'application. Il est toutefois non obligatoire.

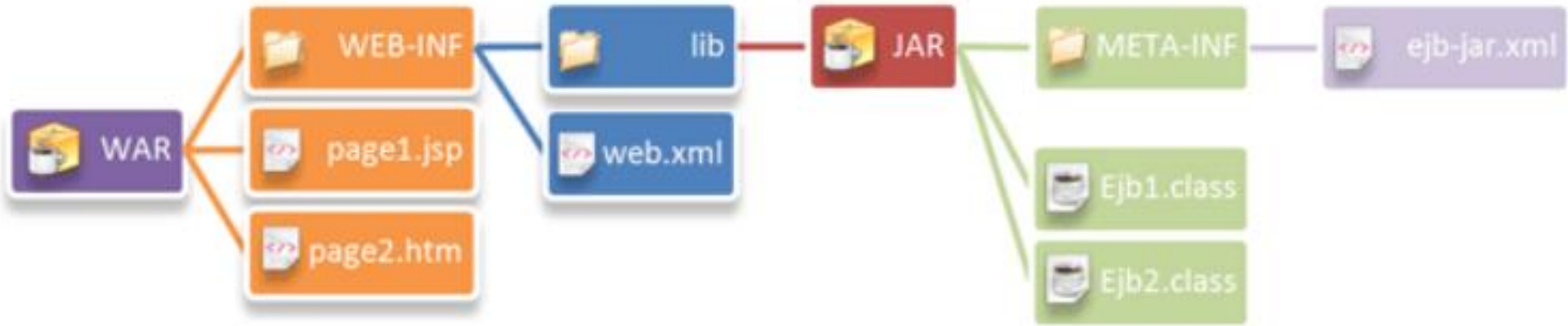
EAR Application



WAR application



WAR Application



Déploiement

- La phase de déploiement pour le développeur est généralement très simple. Il suffit de placer l'archive packagée dans un répertoire spécial du serveur d'applications ou d'utiliser l'administration de celui-ci.
- Le dossier de déploiement dans wildfly 10.0.1 est :
  - wildfly-10.1.0\standalone\deployments

Démo1

Session Bean