# Les Entreprise JavaBeans (EJB)

**Jean-Marc Farinone**
farinone@cnam.fr

**Maître de Conférences**
**Conservatoire National des Arts et Métiers**
**CNAM Paris (France)**

# Plan de l'exposé

- Présentation des EJB
- Les différents types d'EJB
- Un peu de code
- Cycle de vie des EJB
  - La vie d'un EJB session stateless
  - La vie d'un EJB session stateful

# Présentation des Entreprise JavaBeans (EJB)

# L'idée essentielle

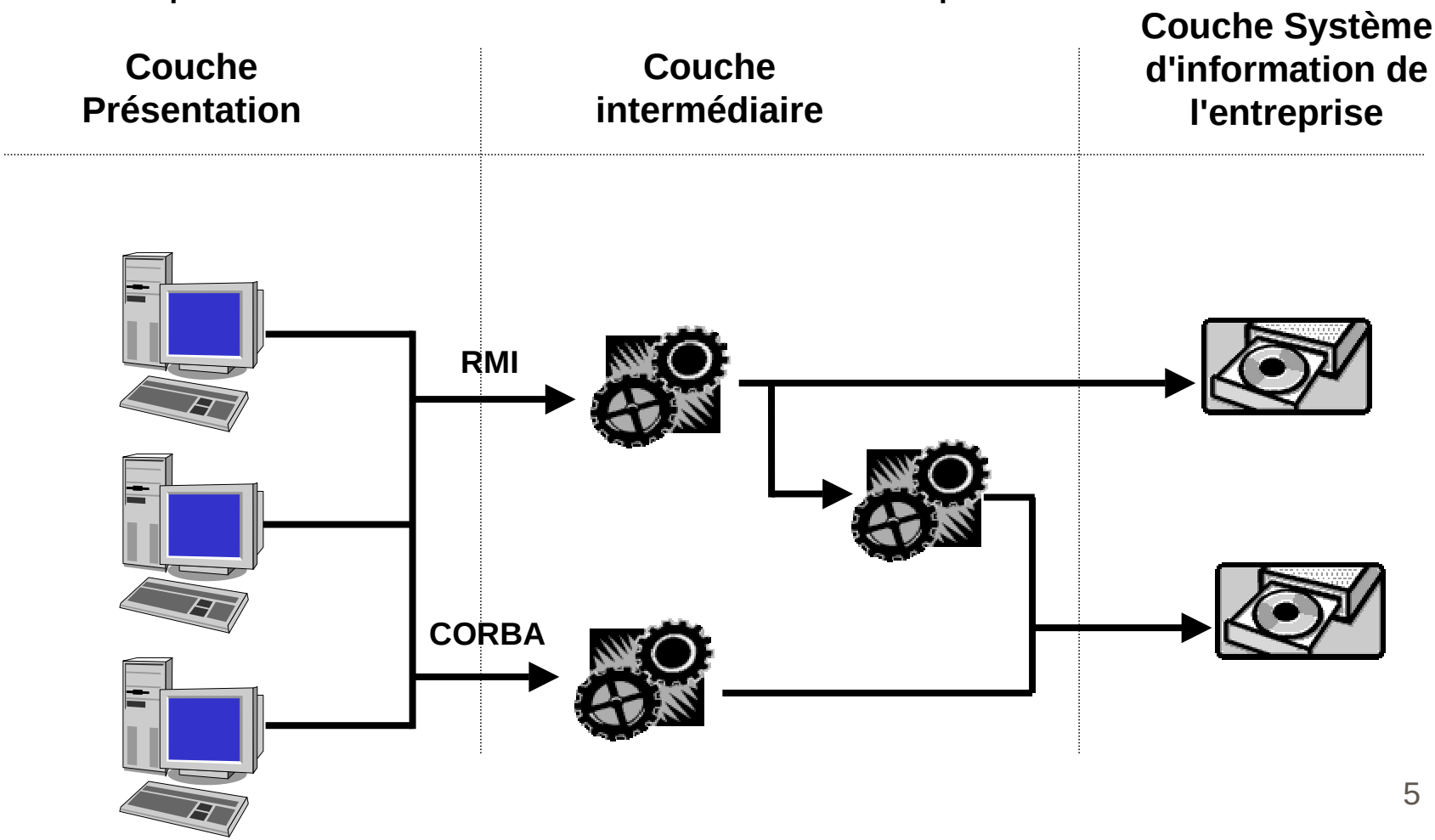
- Se concentrer sur la logique métier à développer (=modéliser, coder les notions du domaine, …) et sous traiter les problèmes connus de :
  - Persistance
  - Transactions
  - Sécurité (authentification, confidentialité, etc.)
  - Réserve (pool) d'objets, équilibrage de charge

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

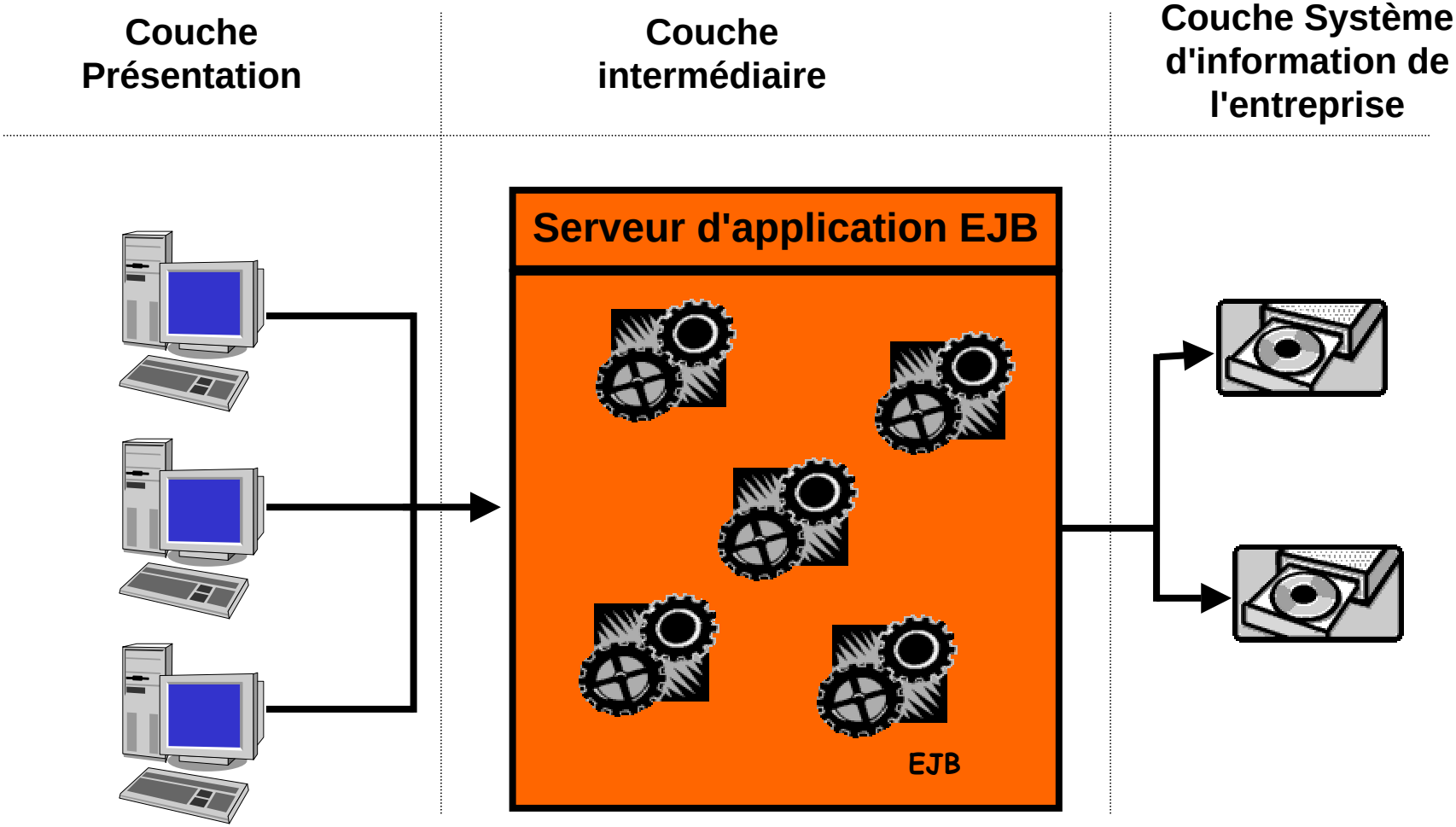
# Architecture des EJB (1/2)

- On passe d'une architecture N-tiers classique :



# Architecture des EJB (2/2)

- … à une architecture qui encapsule les composants



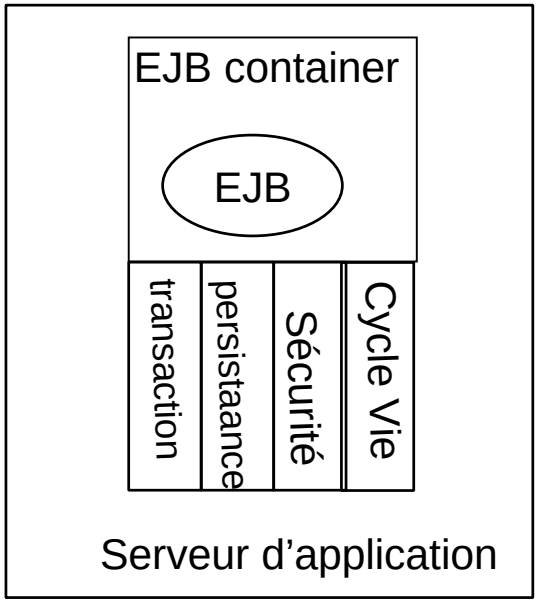
# Fournisseur de serveur/conteneur d'EJB

- Voir à : `http://java.sun.com/j2ee/compatibility.html`
- On peut citer :

| | | |
|---|---|---|
| **BEA Systems** | **BEA WebLogic Server** | **www.bea.com** |
| **JBoss** | **JBoss Application Server** | **www.jboss.org** |
| **Oracle** | **Oracle Container 10g** | **www.oracle.com** |
| **IBM** | **WebSphere** | **www.ibm.com** |
| **Sun Microsystems** | **Application Server Platform Edition 8.2** | **www.java.sun.com/j2ee** |

# Ce qu'amène le conteneur

- encapsulation d'un composant
- fournit des services de nommage, de gestion du cycle de vie, persistance, sécurité, transactionnel
- ces services peuvent être demandés par appels de méthodes par le client. Il seront réalisés par des méthodes propres au bean

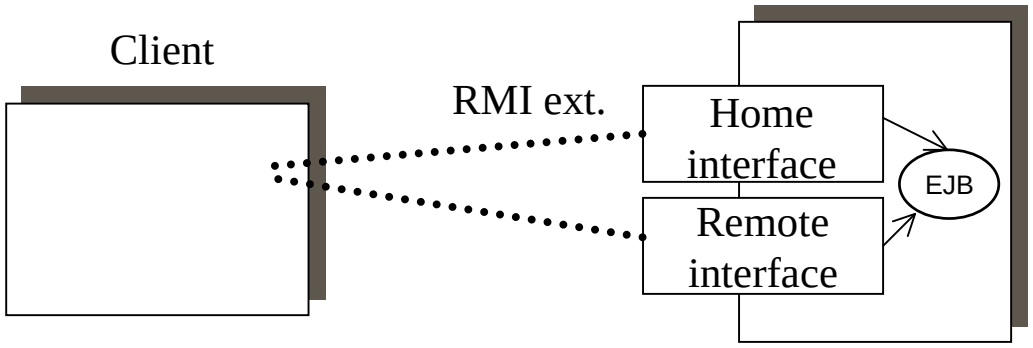


# Ce que doit faire le client

- localiser le Bean par JNDI (Java Naming and Directory Interface)
- utiliser le Bean
  - via *Home* Interface : méthodes liées au cycle de vie du bean : *create(), remove(), …*
  - via *Remote* Interface : services métiers par le bean
- Le conteneur implémente le mécanisme de délégation permettant de faire suivre l'appel au bean

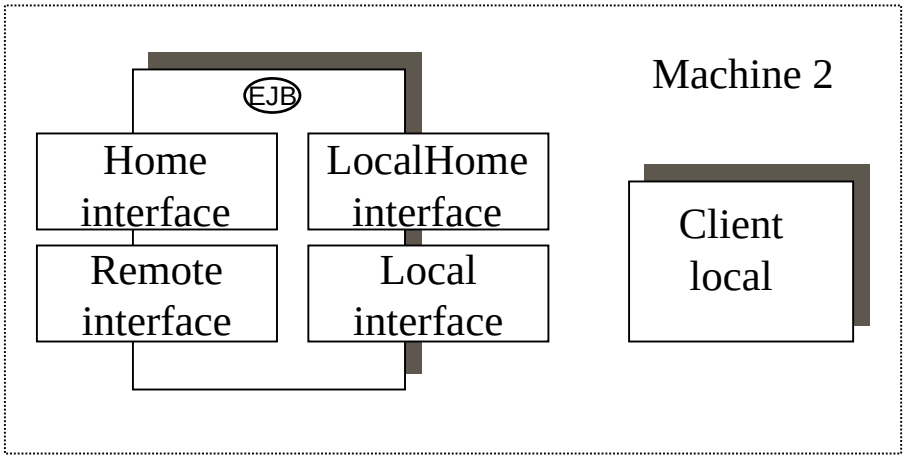
# L’architecture d’utilisation d’un EJB

- Les clients d'un Bean lui parlent *au travers d'interfaces*
  - Ces interfaces, de même que le Bean, suivent la spécification EJB
  - Cette spécification requiert que le Bean *expose certaines méthodes*
- En général (EJB 2.1 Session et Entity), le bean doit proposer une interface de création (Home) et une interface d’utilisation (Remote)



# Architecture

- + éventuellement 2 interfaces d'accès **local**
- pour lier deux clients partageant la même machine virtuelle



# Processus de développement

- Développement d'un bean
  - Ecrire la Home interface
  - Ecrire la Remote interface
  - Ecrire l'implémentation (classe) du bean
  - Compiler ces classes et interfaces
- Déploiement du Bean
  - Construire le descripteur de déploiement
  - Construire le fichier d'archive .jar ou .ear
- Utilisation du Bean
  - Ecrire, compiler et exécuter le client
    - qui peut être une servlet ou une application Java ou une applet, etc.

# Les différents Entreprise JavaBeans (EJB)

# Les trois sortes d'EJB

- EJB Session
  - stateless
  - stateful
- EJB Entity
  - CMP (Container Managed Persistence)
  - BMP (Bean Managed Persistence)
- EJB Message Driven

# Les EJB Session

- Modélise une (euh) … session
- C'est-à-dire une suite d'interaction avec les données
- En général c'est par ces beans que commence la première connexion
- Puis les beans sessions accèdent aux entity beans
- Si le service proposé est un ajout fonctionnel au client, définir un EJB Session stateless (sans état)
  - Exemple : un convertisseur de monnaies
  - Un tel EJB est partageable consécutivement par de multiples clients
- Si l'interaction doit être suivie, on utilise un EJB session stateful (avec état)
  - Exemple : un panier de commandes
  - Un tel EJB est propre à un client pendant toute la durée de la session
  - Il gère une conversation avec un client

# Les Entity EJB

- Modélise une donnée persistance
- Et par exemple une interface sur un ligne dans une table de BD
- En général la gestion de la persistance est assurée par le container (EJB entity CMP). Mais on peut la traiter complètement par l'EJB (EJB entity BMP)

# Les EJB Message Driven

- Sont les beans réactifs à des événements
- Sont abonnées à des fils de discussions (suite de messages)
- Sont informés lorsqu'un message doit leur être délivré
- En fait s'appuie sur JMS (Java Message Service)

# Un peu de code

# Les EJB Session : un peu de syntaxe

- Un EJB session 2.1 =
  - 2 interfaces : une interface de création + une interface d'utilisation
  - + une classe (d'implémentation)
- Cela a été simplifié en version 3.0

# Les interfaces d'un EJB Session 2.1 : un peu de syntaxe

- L'interface de création (Home) :

```
import javax.ejb.*;
import java.rmi.*;

public interface PanierDeCommandeHome extends EJBHome
{
   PanierDeCommande create( ) throws
         CreateException, RemoteException;
}
```

- L'interface d'utilisation (Remote) :

```
import javax.ejb.*;
import java.rmi.*;
import java.util.*;

public interface PanierDeCommande extends EJBObject
{
   public void buyItem(Product item)
                  throws RemoteException;

   public ArrayList getItemsList()
                  throws RemoteException;
}
```

# La classe d'implantation d'un EJB Session 2.1 : un peu de syntaxe

- On indique dans cette classe le code applicatif de cet objet métier

```
import javax.ejb.*;
import java.util.*;

public class PanierDeCommandeBean implements SessionBean
{
    private ArrayList items;
    public void ejbCreate(){
      this.items = new ArrayList();
   }
    public void buyItem(Product item) {
       items.add(item);
    }

    public ArrayList getItemsList( ) {
      return items;
    }
    // …
}
```

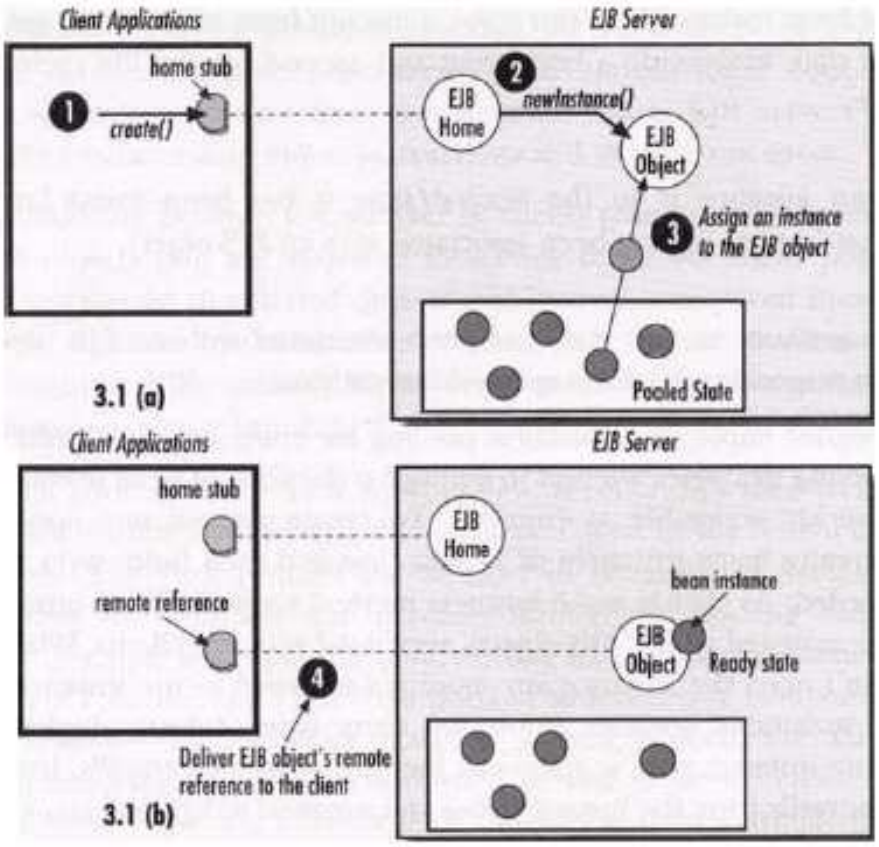
# Un client utilisant un EJB Session 2.1 : un peu de syntaxe

```
import javax.rmi.*;
import javax.naming.*;
import javax.ejb.*;

public class ConnectDemo {

  public static void main (String args[]) {

   // Retrouver l'EJB a partir du service de nommage
    Context ctx = new InitialContext();
   // trouver l'objet local
    Object obj = ctx.lookup("PanierDeCommandeHome");

    // conversion dans la bonne classe
    PanierDeCommandeHome bHome =
             (PanierDeCommandeHome)PortableRemoteObject.narrow(
                                    obj, PanierDeCommandeHome.class);

   PanierDeCommande b = (PanierDeCommande) bHome.create();

    b.buyItem(new EnregistrementMusical("Les Beatles", "Let it be", ……) );
}
```

# Cycle de vie des EJB

# Gestion des ressources

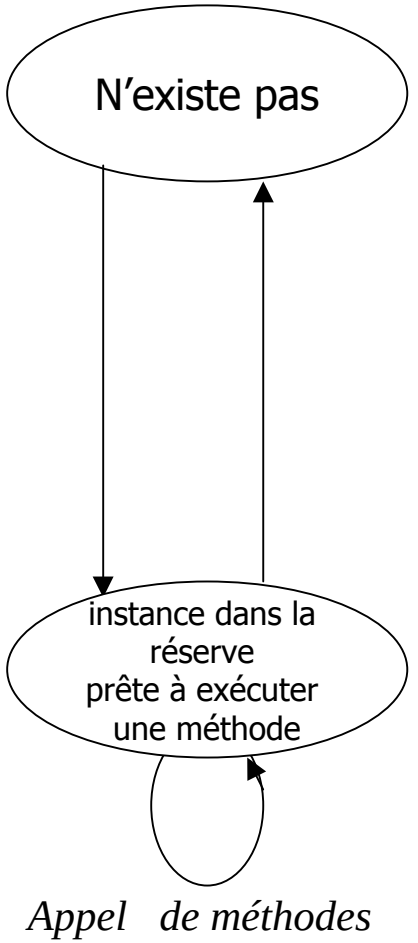
- Le serveur EJB maintient un « pool » (une réserve) d'instances de beans (pour les EJB entity et les EJB session stateless)



# Les EJB session stateless

# Cycle de vie d'un Session Bean Stateless (1/4)

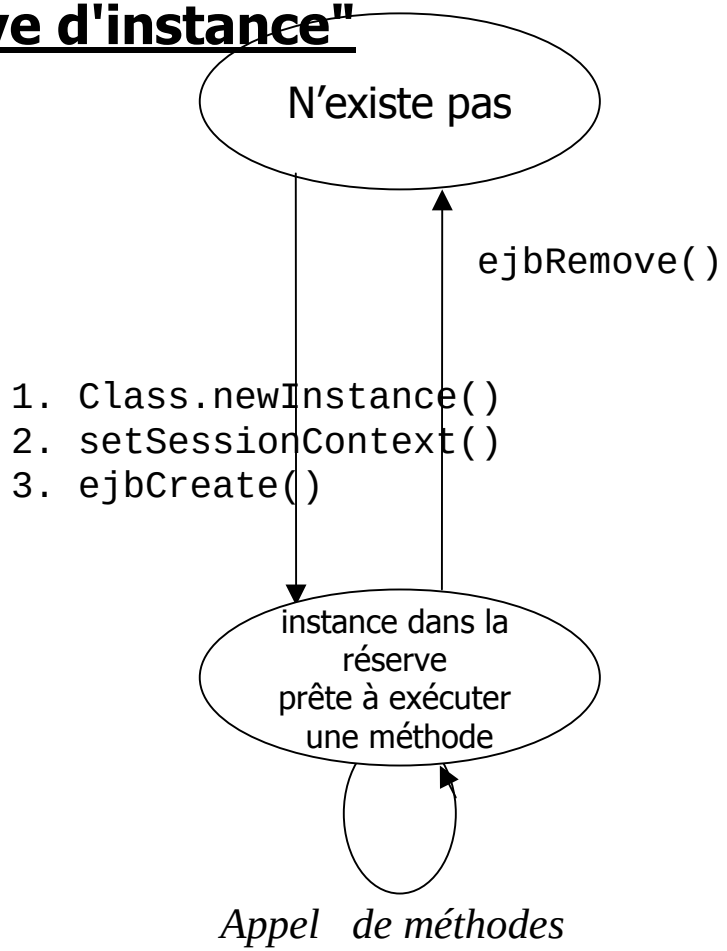
- 2 états :
  - "n'existe pas"
  - "instance de la réserve prêt à exécuter une méthode"
- L'état "n'existe pas" indique que l'EJB n'existe pas dans la mémoire du système : l'instance EJB n'a pas encore été créée.
- Quand le serveur démarre, il place un certain nombre d'EJB session stateless dans la réserve prêt à exécuter une méthode.
- (en fait dépend de l'implémentation du serveur)
- Quand le nombre d'instance de la réserve est insuffisant, d'autres instances sont créées et ajoutées à la réserve.



# Cycle de vie d'un Session Bean Stateless : automate général (2/4)

**Passage de l'état "N'existe pas" à la "réserve d'instance"**

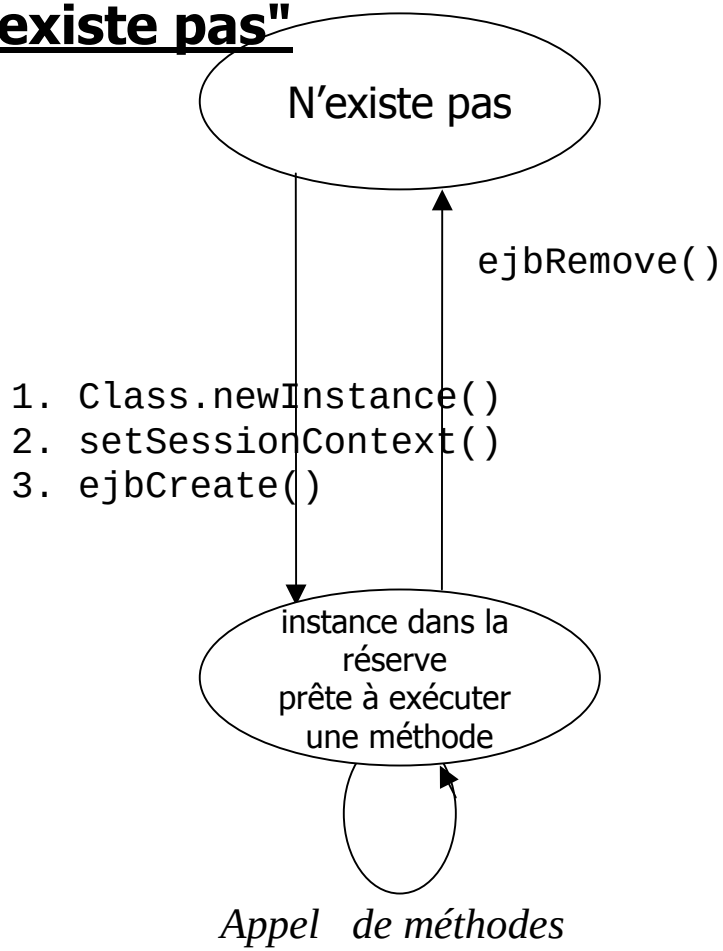
- Le conteneur crée une instance en lançant :
  `newInstance()`
  sur l'objet de la classe `Class` qui modélise la classe de cet EJB session stateless (et oui !!)
- Puis la méthode `setSessionContext(SessionContext ctx)` de la classe `SessionBean` est lancée sur le bean. Elle permettra au bean de connaître son contexte (d'exécution)
- Puis la méthode `ejbCreate()` est lancée sur le bean. Pour un EJB session sans état il n'y a qu'une seule méthode `ejbCreate()` sans argument.



# Cycle de vie d'un Session Bean Stateless : automate général (3/4)

**Passage de l'état "réserve d'instance" à "N'existe pas"**

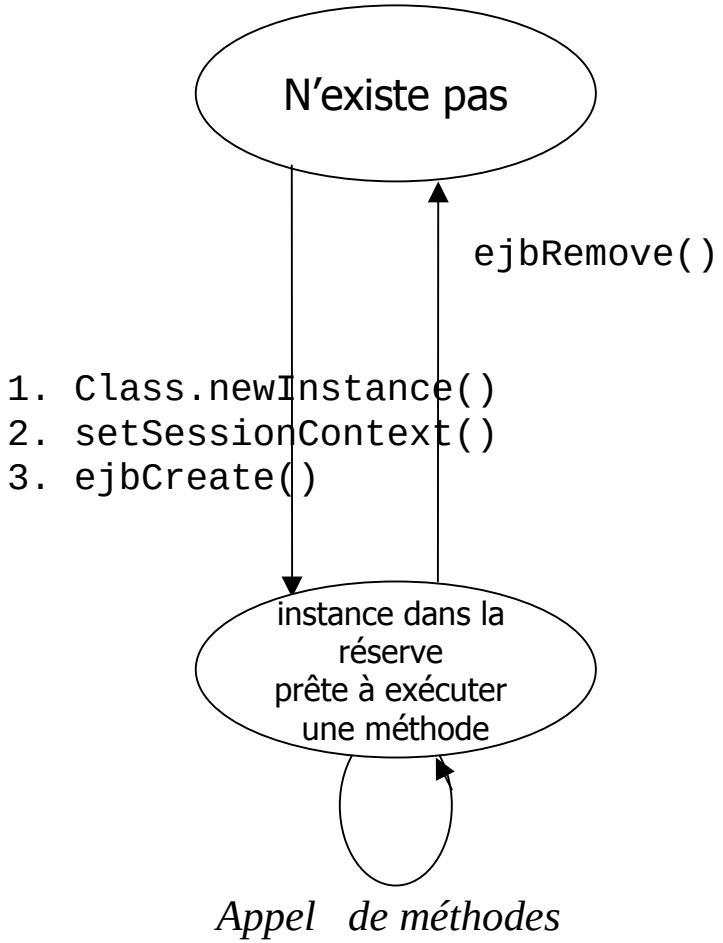
- Se produit quand le serveur n'a plus besoin de cet EJB (par exemple s'il lui manque de la mémoire)
- `ejbRemove()` est lancée sur l'instance.



# Cycle de vie d'un Session Bean Stateless : automate général (4/4)

**Remarques**

- C'est le conteneur qui décide (parfois sur proposition du client) du lancement de ces méthodes.
- Par exemple l'appel `remove()` du client sur le home de l'EJB session stateless ne lance pas `ejbRemove()` sur le bean. Elle indique simplement que ce client n'utilise plus le bean (et le stub sur le bean est libéré) : ce n'est pas le cas pour les bean session stateful et entity (évidemment ;-))
- De même l'appel `create()` du client sur le home de l'EJB session stateless ne lance pas `ejbCreate()` sur le bean : elle ne fait que renvoyer un stub pour un bean "sorti de la réserve"



# Manipulation d'un EJB Session stateless par un client

- On utilise JNDI

```
import javax.rmi.*;
import javax.naming.*;
import javax.ejb.*;

…
// Retrouver l'EJB a partir du service de nommage
    Context ctx = new InitialContext();
// trouver l'objet local
    Object obj = ctx.lookup("XXXHome");
// conversion dans la bonne classe
    XXXHome bHome = (XXXHome)PortableRemoteObject.narrow(obj, XXXHome.class);

   XXX b = (XXX) bHome.create();
}
```

- XXX est l'interface Remote qui permet, coté client, de manipuler le bean (et de lui demander de lancer des méthodes)

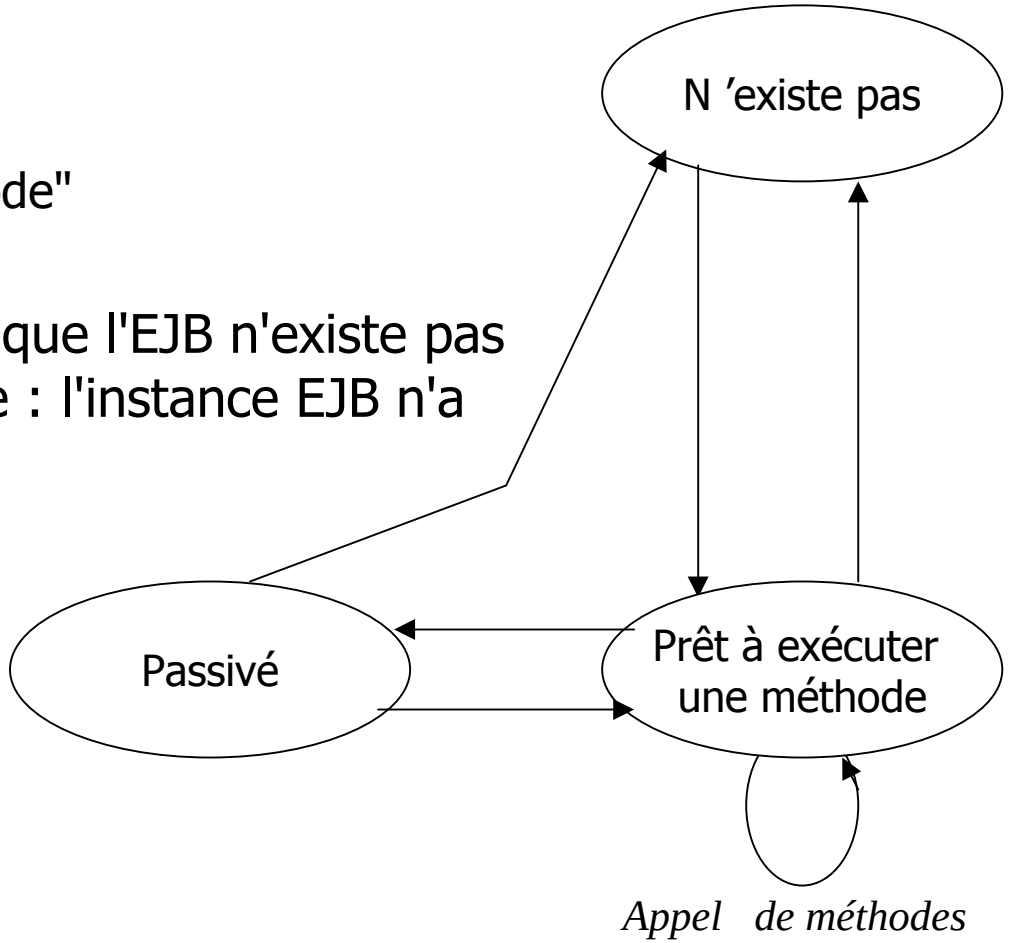
# Les EJB session stateful

# Cycle de vie d'un Session Bean avec état (1/4)

- Un EJB session stateful est dédié à un seul client pour toute sa durée de vie
- Ils ne sont pas échangés entre clients même consécutivement à deux appels
- Ils ne sont pas conservés dans une réserve
- Un EJB session stateful maintient une conversation (un workflow) entre le client et le serveur c'est à dire une session avec état (évidemment ;-))

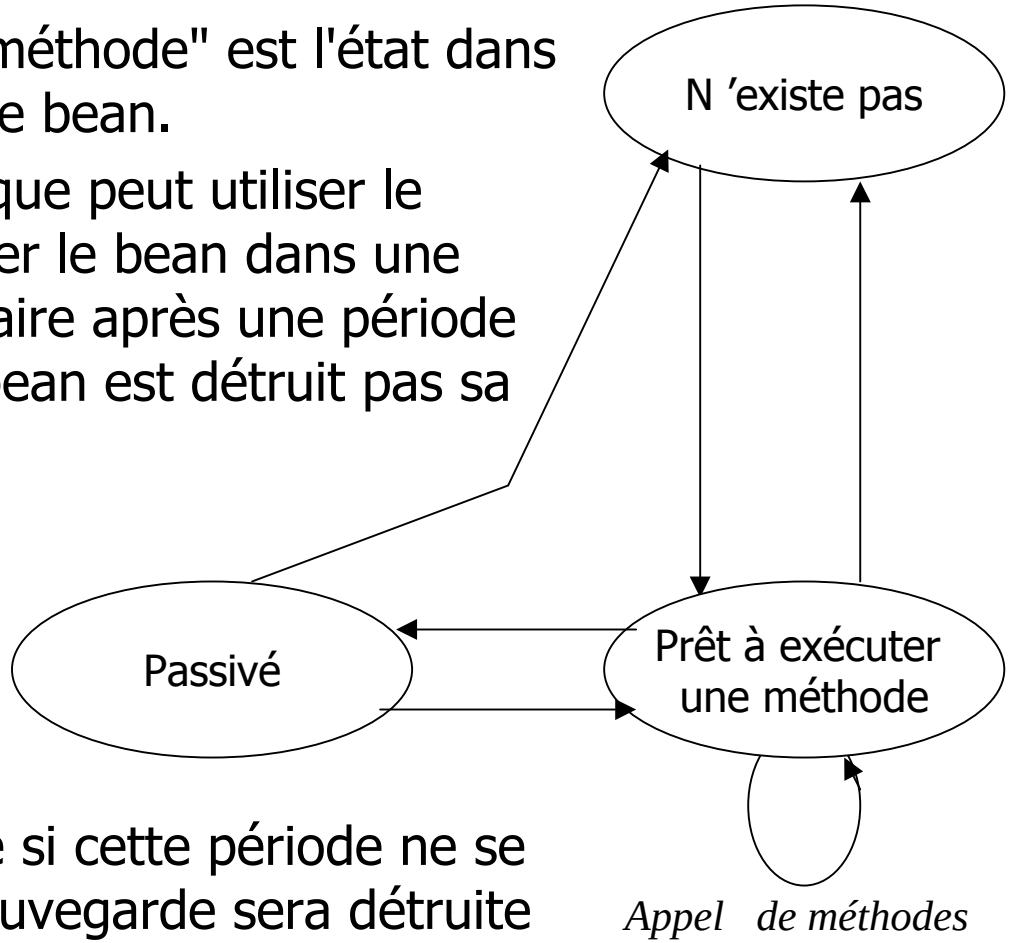
# Cycle de vie d'un Session Bean avec état (2/4)

- 3 états :
  - "n'existe pas"
  - "Prêt à exécuter une méthode"
  - "Passivé"
- L'état "n'existe pas" indique que l'EJB n'existe pas dans la mémoire du système : l'instance EJB n'a pas encore été créée.



# Cycle de vie d'un Session Bean avec état (3/4)

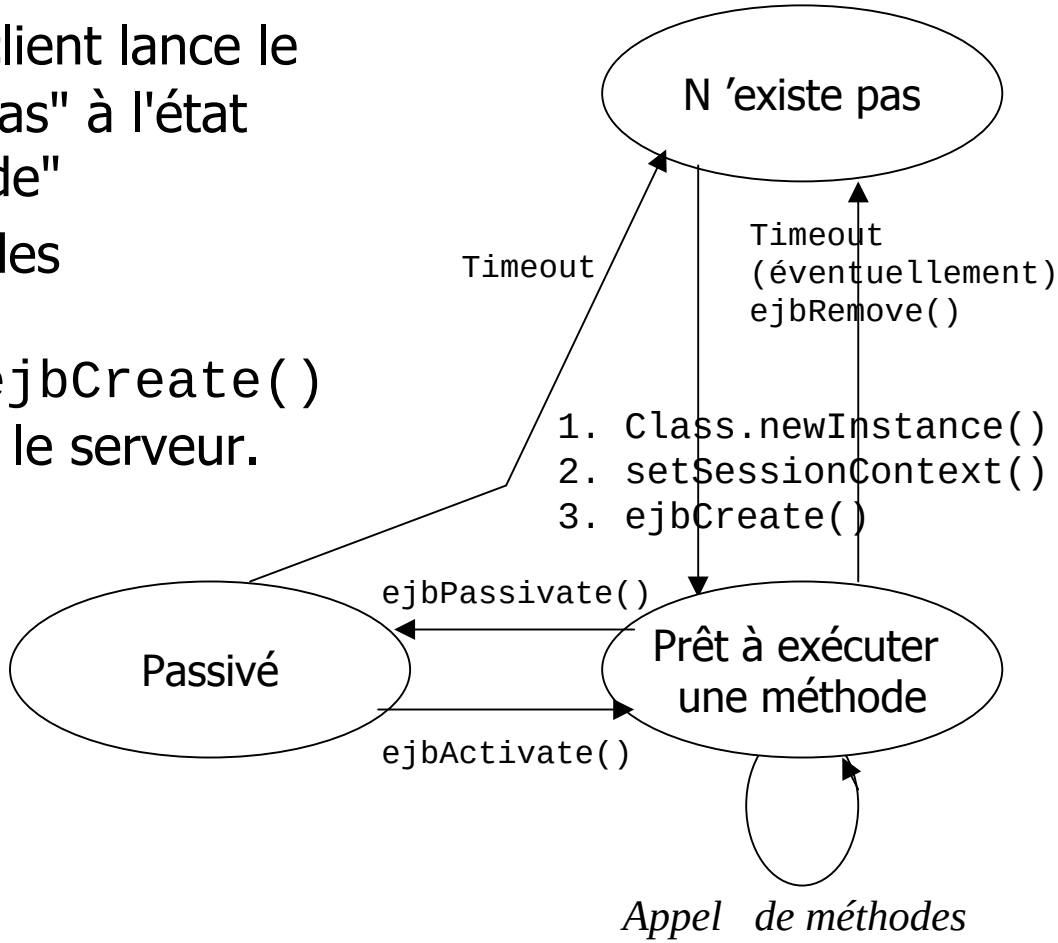
- L'état "Prêt à exécuter une méthode" est l'état dans lequel le client peut utiliser le bean.
- L'état "Passivé" est un état que peut utiliser le conteneur d'EJB pour déposer le bean dans une zone de sauvegarde secondaire après une période d'inactivité. Dans ce cas le bean est détruit pas sa sauvegarde.




- Le bean pourra être réactivé si cette période ne se prolonge pas, sinon cette sauvegarde sera détruite et le bean revient dans l'état "N'existe pas"

# Cycle de vie d'un Session Bean avec état (4/4)

- La méthode `create()` du client lance le passage de l'état "N'existe pas" à l'état "Prêt à exécuter une méthode"
- Dans ce cas les trois méthodes `newInstance()`, `setSessionContext()`, `ejbCreate()` sont lancées sur le bean par le serveur.



# Bibliographie EJB

- La littérature est énorme mais les bibles se trouvent à :
- Site originel `http://java.sun.com/javaee/`
- Tutorial à `http://java.sun.com/javaee/5/docs/tutorial/doc/`

- Support de cours CNAM de Alexandre Tauveron cours Entreprise JavaBeans
- Entreprise JavaBeans, Richard Monson-Haefel; ed O'Reilly traduit en français

# Fin