Architectures des SI - Les architectures 3-tiers
Partie II : les sessions EJB

Informatique, Statistique et IA 4ème année

Olivier Caron[1]
http://ocaron.polytech-lille.net

[1]École d'ingénieurs Polytech Lille
Université de Lille

15 janvier 2026

© Auteur inconnu

Il y a 10 sortes de personnes, ceux qui comprennent le binaire et ceux qui ne le comprennent pas.

Les composants EJB Session

- Assurent les services métiers

Les composants EJB Session

- Assurent les services métiers
- Accèdent aux composants entités

Les composants EJB Session

- Assurent les services métiers
- Accèdent aux composants entités
- Point d'entrée pour les interface utilisateurs (Applications Web, Java Swing,...)

Les composants EJB Session

- Assurent les services métiers
- Accèdent aux composants entités
- Point d'entrée pour les interface utilisateurs (Applications Web, Java Swing,. . .)
- Avec ou sans état transactionnel (Stateless, Stateful)

Les composants EJB Session

- Assurent les services métiers
- Accèdent aux composants entités
- Point d'entrée pour les interface utilisateurs (Applications Web, Java Swing,...)
- Avec ou sans état transactionnel (Stateless, Stateful)
- S'exécute au sein d'un conteneur EJB qui s'occupe :

Les composants EJB Session

- Assurent les services métiers
- Accèdent aux composants entités
- Point d'entrée pour les interface utilisateurs (Applications Web, Java Swing,...)
- Avec ou sans état transactionnel (Stateless, Stateful)
- S'exécute au sein d'un conteneur EJB qui s'occupe :
  - de l'accès à distance (plus de programmation réseau)

Les composants EJB Session

- Assurent les services métiers
- Accèdent aux composants entités
- Point d'entrée pour les interface utilisateurs (Applications Web, Java Swing,...)
- Avec ou sans état transactionnel (Stateless, Stateful)
- S'exécute au sein d'un conteneur EJB qui s'occupe :
  - de l'accès à distance (plus de programmation réseau)
  - des transactions

Les composants EJB Session

- Assurent les services métiers
- Accèdent aux composants entités
- Point d'entrée pour les interface utilisateurs (Applications Web, Java Swing,...)
- Avec ou sans état transactionnel (Stateless, Stateful)
- S'exécute au sein d'un conteneur EJB qui s'occupe :
  - de l'accès à distance (plus de programmation réseau)
  - des transactions
  - de la sécurité

Les composants EJB Session

- Assurent les services métiers
- Accèdent aux composants entités
- Point d'entrée pour les interface utilisateurs (Applications Web, Java Swing,...)
- Avec ou sans état transactionnel (Stateless, Stateful)
- S'exécute au sein d'un conteneur EJB qui s'occupe :
  - de l'accès à distance (plus de programmation réseau)
  - des transactions
  - de la sécurité
  - de la montée en charge du serveur

Framework EJB minimal pour un EJB Session

1. Il faut écrire du code dans une classe Java ☺

Framework EJB minimal pour un EJB Session

1. Il faut écrire du code dans une classe Java ☺
2. Il faut spécifier le type du composant session en annotant la classe soit par `Stateless`, `Singleton` ou `Stateful`.

Framework EJB minimal pour un EJB Session

1. Il faut écrire du code dans une classe Java ☺
2. Il faut spécifier le type du composant session en annotant la classe soit par `Stateless`, `Singleton` ou `Stateful`.
3. Il faut préciser si ce composant sera accessible localement (au sein du conteneur) et/ou à distance :
   ➔La classe implémentera 1 ou 2 interfaces qui seront annotées soit par `Local` et/ou `Remote`.

Framework EJB minimal pour un EJB Session

1. Il faut écrire du code dans une classe Java ☺
2. Il faut spécifier le type du composant session en annotant la classe soit par `Stateless`, `Singleton` ou `Stateful`.
3. Il faut préciser si ce composant sera accessible localement (au sein du conteneur) et/ou à distance :
   ➔La classe implémentera 1 ou 2 interfaces qui seront annotées soit par `Local` et/ou `Remote`.
4. Le serveur JEE doit pouvoir analyser dynamiquement le composant (introspection) :
   ➔La classe dispose d'une fonction constructeur sans paramètre

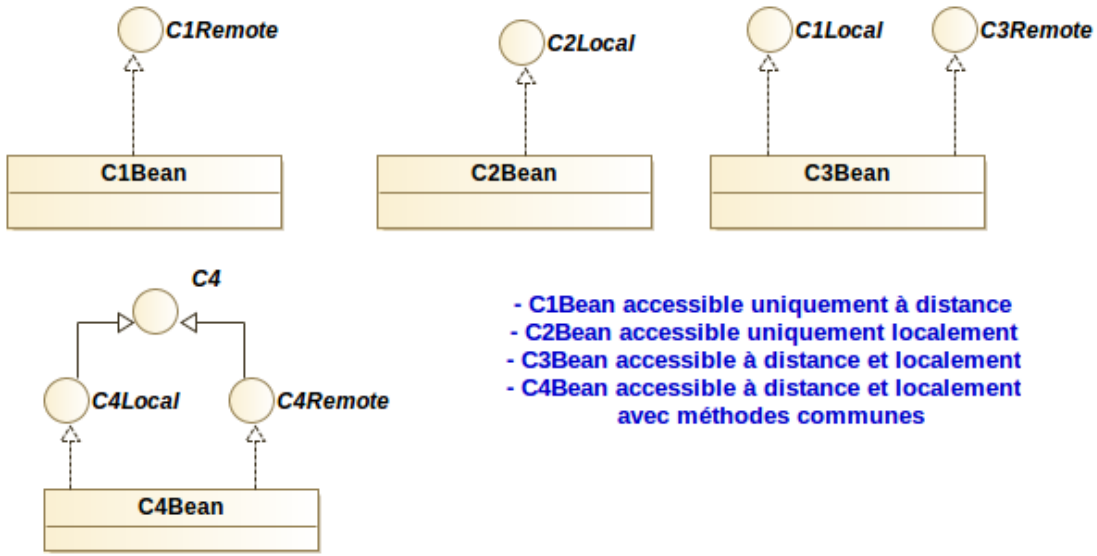
Framework EJB minimal pour un EJB Session

1. Il faut écrire du code dans une classe Java ☺
2. Il faut spécifier le type du composant session en annotant la classe soit par `Stateless`, `Singleton` ou `Stateful`.
3. Il faut préciser si ce composant sera accessible localement (au sein du conteneur) et/ou à distance :
   ➔La classe implémentera 1 ou 2 interfaces qui seront annotées soit par `Local` et/ou `Remote`.
4. Le serveur JEE doit pouvoir analyser dynamiquement le composant (introspection) :
   ➔La classe dispose d'une fonction constructeur sans paramètre

Important : si l'une de ces règles n'est pas appliquée, le serveur considère que ce n'est **pas** un composant session.

Spécification des interfaces

Figure – **Quelques** alternatives de programmation des interfaces



- Conventions de nommage :
  - Suffixe `Bean` au nom de la classe.
  - Suffixes `Remote` et `Local` pour les interfaces annotées.

Processus de développement des sessions

1. Conception UML (diagramme de classes, séquence, ...)

Processus de développement des sessions

1. Conception UML (diagramme de classes, séquence, ...)
2. Programmation Java

Processus de développement des sessions

1. Conception UML (diagramme de classes, séquence, ...)
2. Programmation Java
3. Archivage (archivage des classes compilées via jar)

Processus de développement des sessions

1. Conception UML (diagramme de classes, séquence, ...)
2. Programmation Java
3. Archivage (archivage des classes compilées via jar)
4. Déploiement (Wildfly ➔copie du jar dans un répertoire constamment scruté par le serveur)

Processus de développement des sessions

1. Conception UML (diagramme de classes, séquence, . . .)
2. Programmation Java
3. Archivage (archivage des classes compilées via jar)
4. Déploiement (Wildfly ➔copie du jar dans un répertoire constamment scruté par le serveur)

Illustration dans les diapositives suivantes

Programmation d'un composant session sans état accessible à distance et localement avec les mêmes fonctionnalités

Exemple de composant session sans état (1/3)

```
package ejb.sessions ;

  import jakarta.ejb.Stateless ;

  @Stateless
  public class CalculSalaireBean
    implements CalculSalaireRemote, CalculSalaireLocal {

    public CalculSalaireBean() {}

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

Exemple de composant session sans état (2/3)

```
package ejb.sessions ;

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

Exemple de composant session sans état (3/3)

```
package ejb.sessions ;

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

```
package ejb.sessions ;

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

Premier Bilan

1. Programmation "réseau" Java simple ☺

Premier Bilan

1. Programmation "réseau" Java simple ☺
2. Une fois compilé (`javac`), archivé (`jar`) et déployé sur un serveur (`cp`), ce composant est :

Premier Bilan

1. Programmation "réseau" Java simple ☺
2. Une fois compilé (`javac`), archivé (`jar`) et déployé sur un serveur (`cp`), ce composant est :
   - disponible à distance (car `@Remote`)
     ➔programmation réseau géré par le serveur,

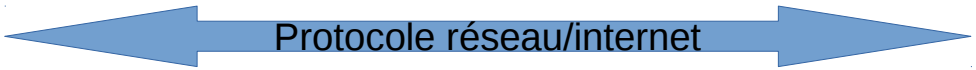
Premier Bilan

1. Programmation "réseau" Java simple ☺
2. Une fois compilé (`javac`), archivé (`jar`) et déployé sur un serveur (`cp`), ce composant est :
   - disponible à distance (car `@Remote`)
     ➔programmation réseau géré par le serveur,
   - de manière sécurisé ou pas
     ➔sécurité gérée par le serveur

Premier Bilan

1. Programmation "réseau" Java simple ☺
2. Une fois compilé (`javac`), archivé (`jar`) et déployé sur un serveur (`cp`), ce composant est :
   - disponible à distance (car `@Remote`)
     ➔programmation réseau géré par le serveur,
   - de manière sécurisé ou pas
     ➔sécurité gérée par le serveur
   - pour une multitude de clients
     ➔accès concurrents/transactions gérés par le serveur, montée en charge gérée par le serveur.

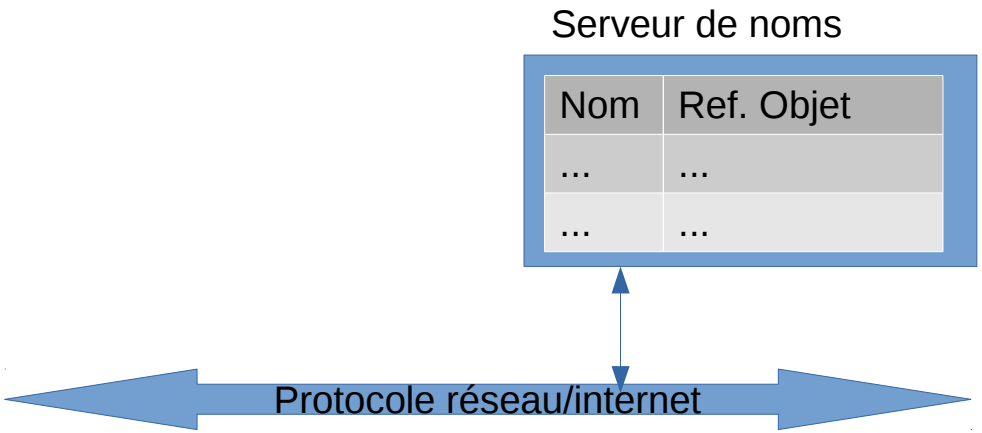
Cycle de vie d'une application par objets répartis



Programmation Java RMI :

`http://ocaron.polytech-lille.net/enseignement/is4/al/coursRMI.pdf`

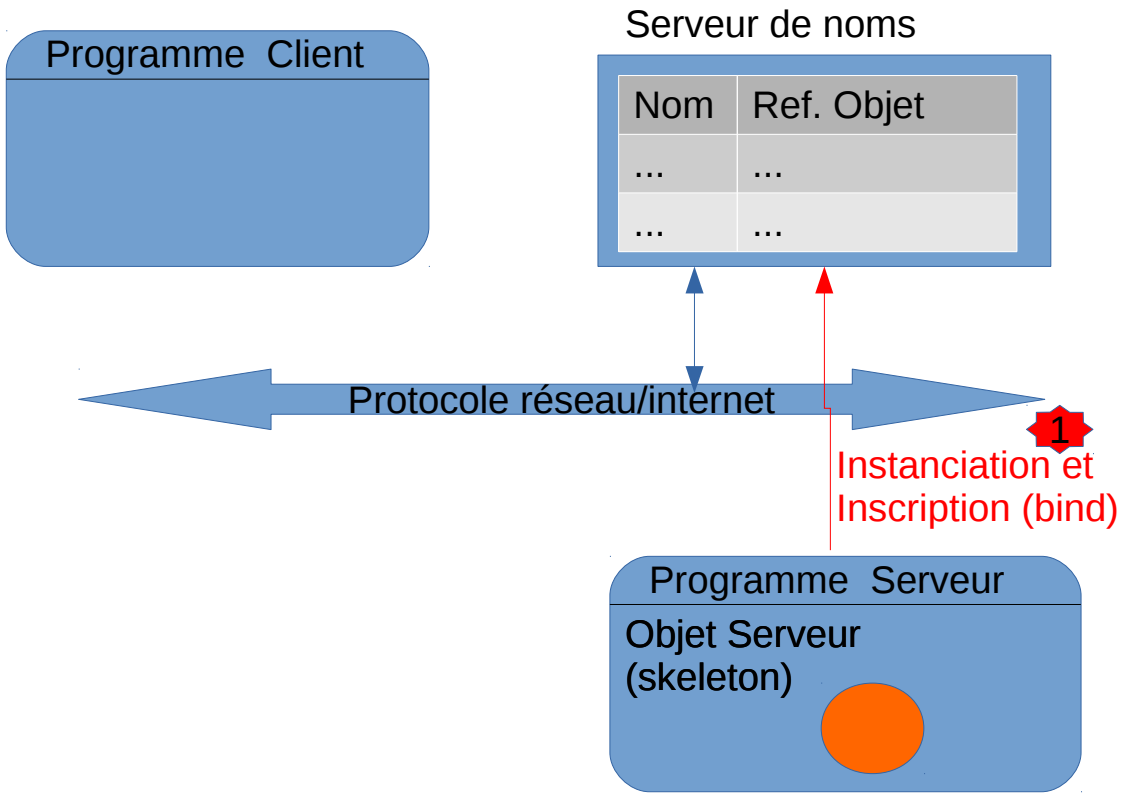
Cycle de vie d'une application par objets répartis



Programmation Java RMI :

`http://ocaron.polytech-lille.net/enseignement/is4/al/coursRMI.pdf`

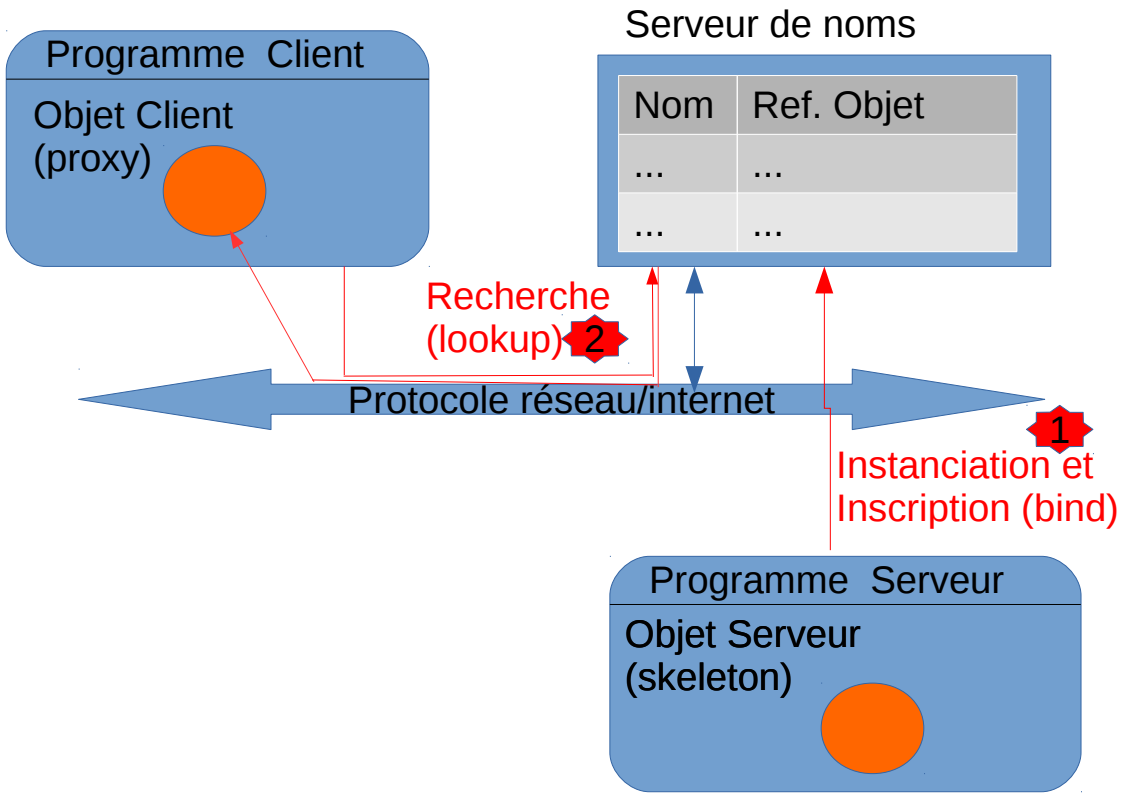
Cycle de vie d'une application par objets répartis



Programmation Java RMI :

`http://ocaron.polytech-lille.net/enseignement/is4/al/coursRMI.pdf`

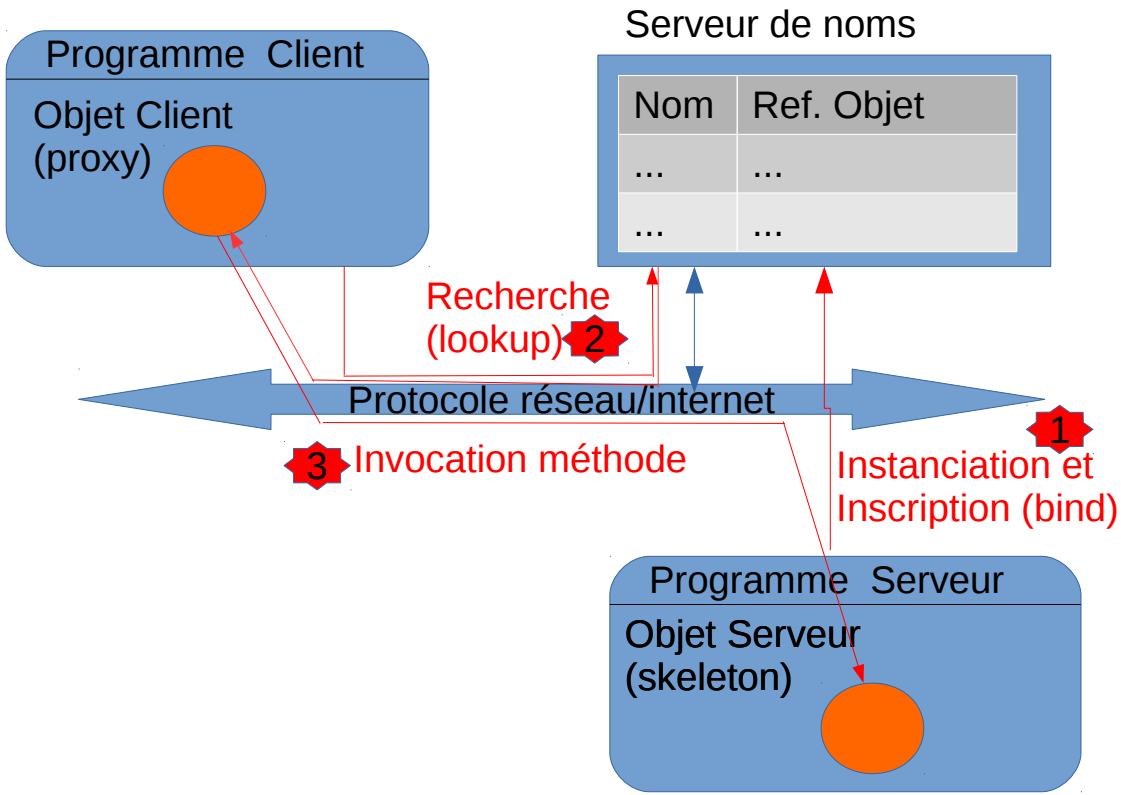
Cycle de vie d'une application par objets répartis



Programmation Java RMI :

`http://ocaron.polytech-lille.net/enseignement/is4/al/coursRMI.pdf`

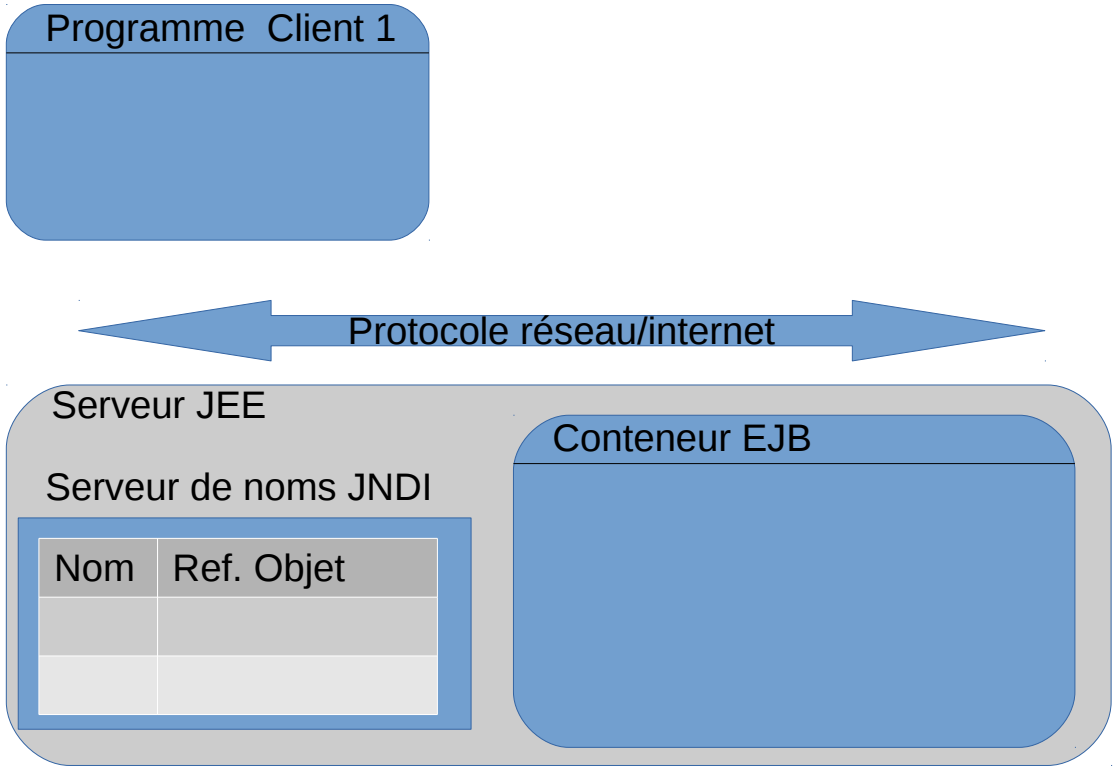
Cycle de vie d'une application par objets répartis



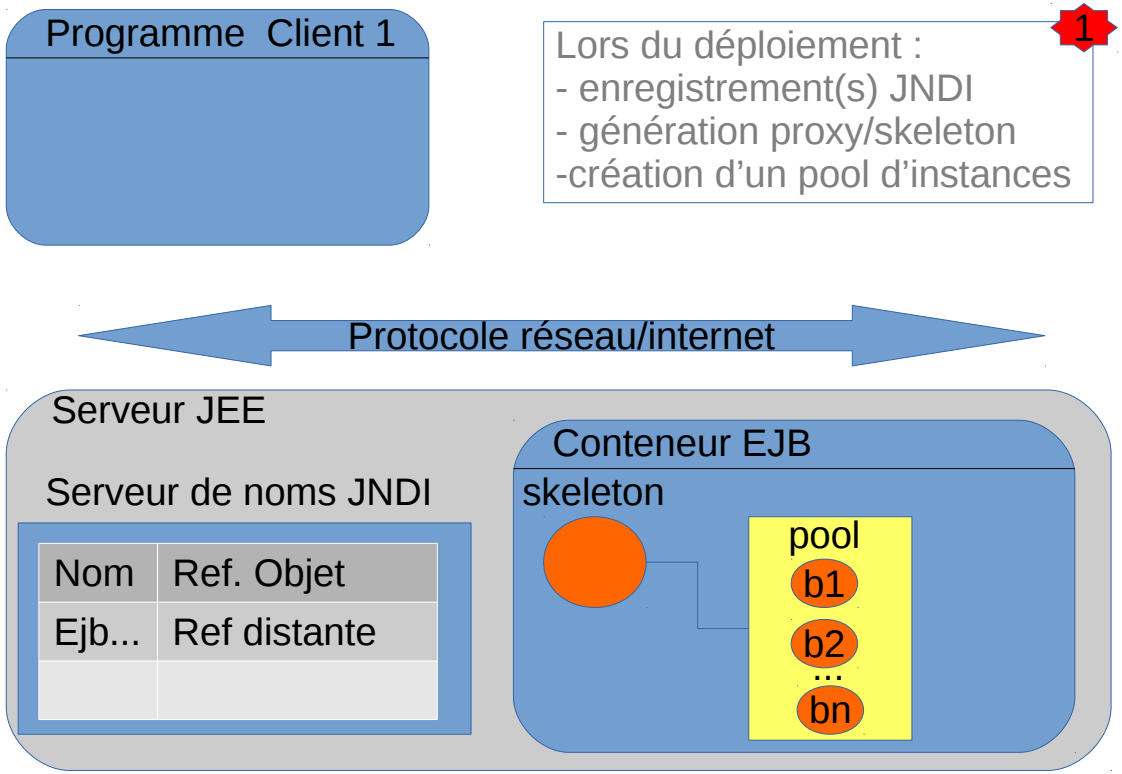
Programmation Java RMI :

`http://ocaron.polytech-lille.net/enseignement/is4/al/coursRMI.pdf`

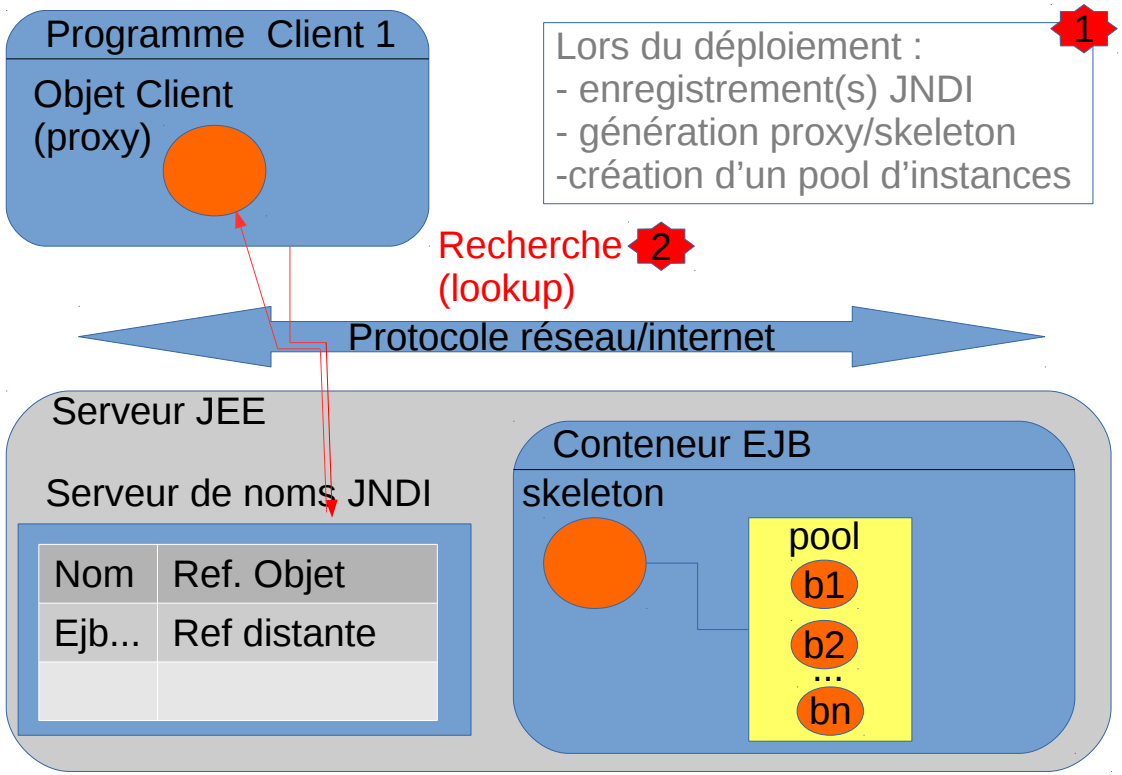
Application aux sessions EJB à distance



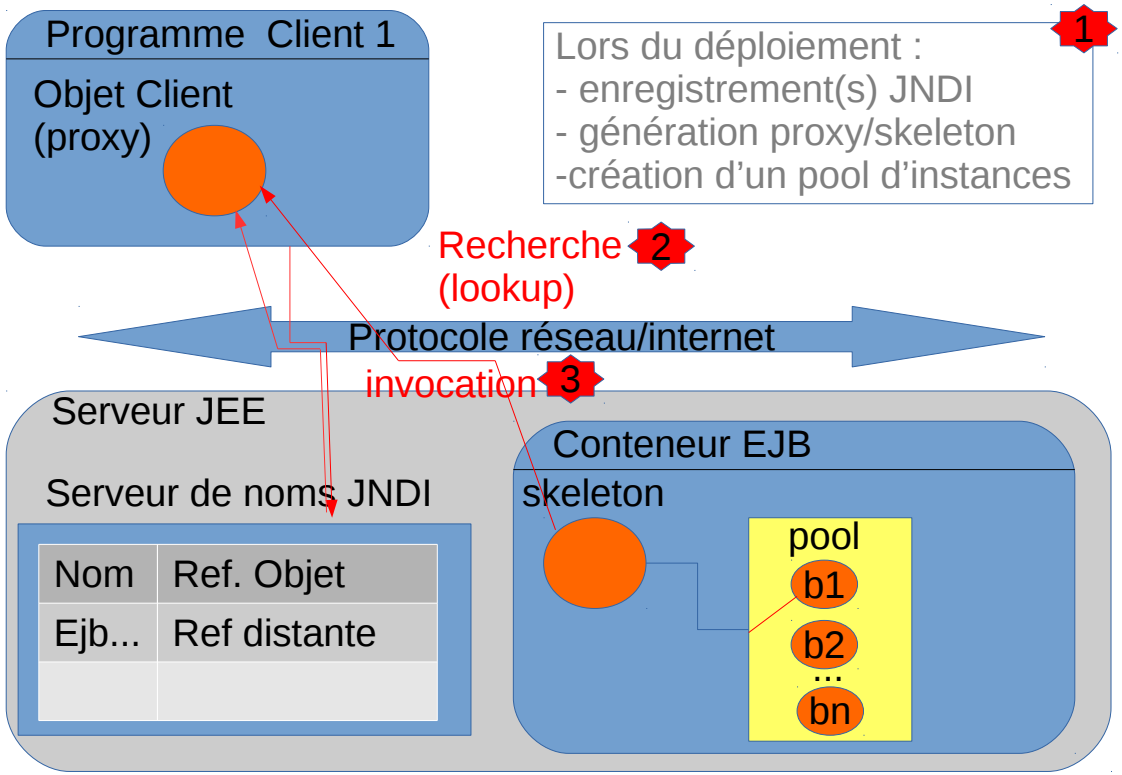
Application aux sessions EJB à distance



Application aux sessions EJB à distance



Application aux sessions EJB à distance



Le serveur de noms JEE

- Conforme à la norme JNDI (Java Naming Directory Interface) :

Le serveur de noms JEE

- Conforme à la norme JNDI (Java Naming Directory Interface) :
  - Principe similaire à JDBC

Le serveur de noms JEE

- Conforme à la norme JNDI (Java Naming Directory Interface) :
  - Principe similaire à JDBC
  - Définit une interface (API) unifiée pour gérer un service de noms

Le serveur de noms JEE

- Conforme à la norme JNDI (Java Naming Directory Interface) :
  - Principe similaire à JDBC
  - Définit une interface (API) unifiée pour gérer un service de noms
  - De multiples implémentations (LDAP, DNS, etc)

Le serveur de noms JEE

- Conforme à la norme JNDI (Java Naming Directory Interface) :
  - Principe similaire à JDBC
  - Définit une interface (API) unifiée pour gérer un service de noms
  - De multiples implémentations (LDAP, DNS, etc)
- Autorise une structuration arborescente des services,
  exemple :`chemin/sous-chemin/sous-sous-chemin/nom`

Déploiement du composant session accessible à distance

- Lors du déploiement d'une archive/module de composant(s) sessions, le serveur :

Déploiement du composant session accessible à distance

- Lors du déploiement d'une archive/module de composant(s) sessions, le serveur :
  1. extrait l'archive (`jar`)

Déploiement du composant session accessible à distance

- Lors du déploiement d'une archive/module de composant(s) sessions, le serveur :
  1. extrait l'archive (`jar`)
  2. analyse les classes de l'archive (introspection).

Déploiement du composant session accessible à distance

- Lors du déploiement d'une archive/module de composant(s) sessions, le serveur :
  1. extrait l'archive (`jar`)
  2. analyse les classes de l'archive (introspection).
  3. détecte le ou les composants sessions (cf règles slide 4)

Déploiement du composant session accessible à distance

- Lors du déploiement d'une archive/module de composant(s) sessions, le serveur :
  1. extrait l'archive (`jar`)
  2. analyse les classes de l'archive (introspection).
  3. détecte le ou les composants sessions (cf règles slide 4)
  4. fournit des points d'accès selon le ou les interfaces du composant via le serveur de noms JNDI, règles de nommage par défaut

Serveur Wildfly, règle de nommage JNDI

- Dans le cas des sessions `Stateless` accessibles à distance, le serveur wildfly génère l'adresse JNDI suivante :

  ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<interface-FQN>

  - `app-name` : nom de l'archive de l'application (`.ear`) contenant l'archive (`.jar`) du(des) composant(s) session(s), chaîne vide si pas d'archive ear
  - `module-name` : nom de l'archive du(des) composant(s) session(s)
  - `distinct-name` nom optionnel du bean si spécifié via annotations
  - `bean-name` nom de la classe du bean.
  - `interface-FQN` nom complet de l'interface (exemple : `ejb.sessions.CalculSalaireRemote`)

Un programme client

```
package client ;

import javax.naming.InitialContext;
import javax.naming.NamingException ;
import ejb.sessions.CalculSalaireRemote ;

public class MainCalculSalaire {
  public static void main(String[] args) {
    String appName="" , moduleName="calculSalaireSessions" ;
    String distinctName="" , beanName="CalculSalaireBean" ;
    String remoteInterfaceName=CalculSalaireRemote.class.getName();
    String adresseJNDI="ejb:"+appName+"/"+moduleName+"/"+distinctName+
                       "/"+beanName+"!"+remoteInterfaceName ;
    try {
      InitialContext ctx = new InitialContext();
      Object obj = ctx.lookup(adresseJNDI);
      CalculSalaireRemote salaire = (CalculSalaireRemote) obj ;
      System.out.println("salaire : "+ salaire.getSalaire(24))  ;
    } catch(NamingException e1) {
      System.err.println("erreur, acces au serveur de noms") ;
    }
  }
}
```

Exécution avec plate-forme WildFly

- Le classpath doit contenir les classes réseaux WildFly pour accéder au serveur de noms JNDI WildFly (JBoss)

Exécution avec plate-forme WildFly

- Le classpath doit contenir les classes réseaux WildFly pour accéder au serveur de noms JNDI WildFly (JBoss)
- Il faut aussi connaître la localisation du serveur

Exécution avec plate-forme WildFly

- Le classpath doit contenir les classes réseaux WildFly pour accéder au serveur de noms JNDI WildFly (JBoss)
- Il faut aussi connaître la localisation du serveur
- Librairie JNDI, fichier `jndi.properties` :

```
java.naming.factory.url.pkgs=org.jboss.ejb.client.naming
java.naming.factory.initial=org.wildfly.naming.client.WildFlyInitialContextFactory

java.naming.provider.url=http-remoting://localhost:8080
```

Exécution avec plate-forme WildFly

- Le classpath doit contenir les classes réseaux WildFly pour accéder au serveur de noms JNDI WildFly (JBoss)
- Il faut aussi connaître la localisation du serveur
- Librairie JNDI, fichier `jndi.properties` :

```
java.naming.factory.url.pkgs=org.jboss.ejb.client.naming
java.naming.factory.initial=org.wildfly.naming.client.WildFlyInitialContextFactory

java.naming.provider.url=http-remoting://localhost:8080
```

- Dans cet exemple, le programme client s'exécute sur la même machine que le serveur JEE (incluant le serveur de noms JNDI).

Cycle de vie des composants session sans état

Cycle de vie des composants session sans état

Cycle de vie des composants session sans état

- Un composant `Stateless` n'est associé qu'à un client que pour le temps de l'exécution d'une méthode

Cycle de vie des composants session sans état

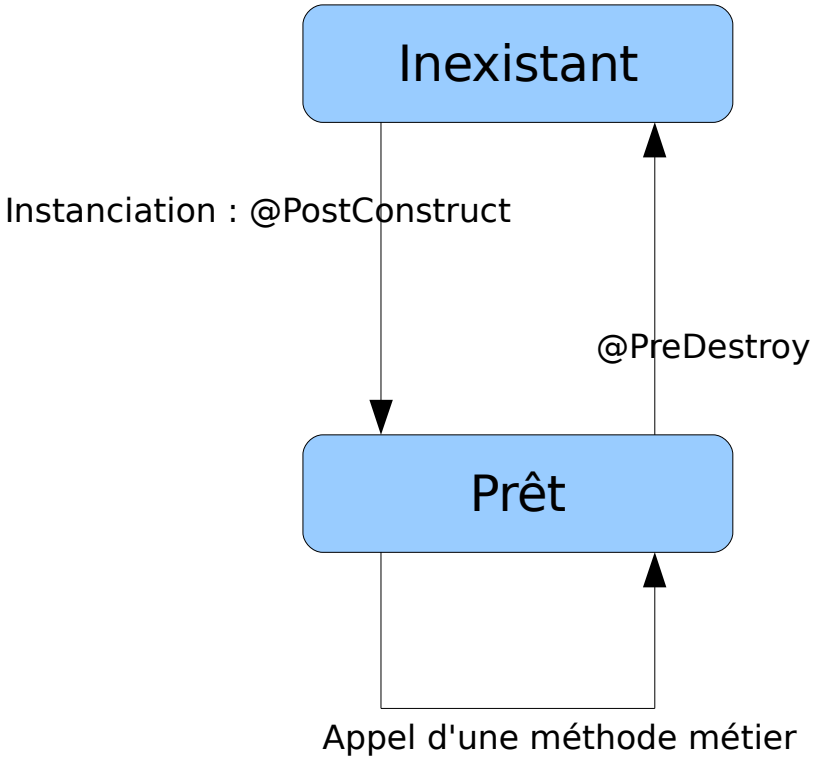
- Un composant `Stateless` n'est associé qu'à un client que pour le temps de l'exécution d'une méthode
- Il peut donc être associé successivement à plusieurs clients

Cycle de vie des composants session sans état

- Un composant `Stateless` n'est associé qu'à un client que pour le temps de l'exécution d'une méthode
- Il peut donc être associé successivement à plusieurs clients
- Il peut être supprimé par le conteneur en cas de montée en charge par exemple.

Cycle de vie des composants session sans état

- Un composant `Stateless` n'est associé qu'à un client que pour le temps de l'exécution d'une méthode
- Il peut donc être associé successivement à plusieurs clients
- Il peut être supprimé par le conteneur en cas de montée en charge par exemple.



Le composant `Singleton`

- Un composant singleton est annoté par `jakarta.ejb.Singleton`

Le composant `Singleton`

- Un composant singleton est annoté par `jakarta.ejb.Singleton`
- Même cycle de vie qu'un `Stateless`

Le composant `Singleton`

- Un composant singleton est annoté par `jakarta.ejb.Singleton`
- Même cycle de vie qu'un `Stateless`
- Le serveur assure le fait qu'il n'existe qu'une seule instance :
  ➔n clients se partagent la même instance

Les composants session avec état

- Annotés par `jakarta.ejb.Stateful`

Les composants session avec état

- Annotés par `jakarta.ejb.Stateful`
- Après la première invocation de méthode, le composant est associé au client.

Les composants session avec état

- Annotés par `jakarta.ejb.Stateful`
- Après la première invocation de méthode, le composant est associé au client.
- Possibilité d'utiliser des variables d'instances entre deux appels de méthodes.

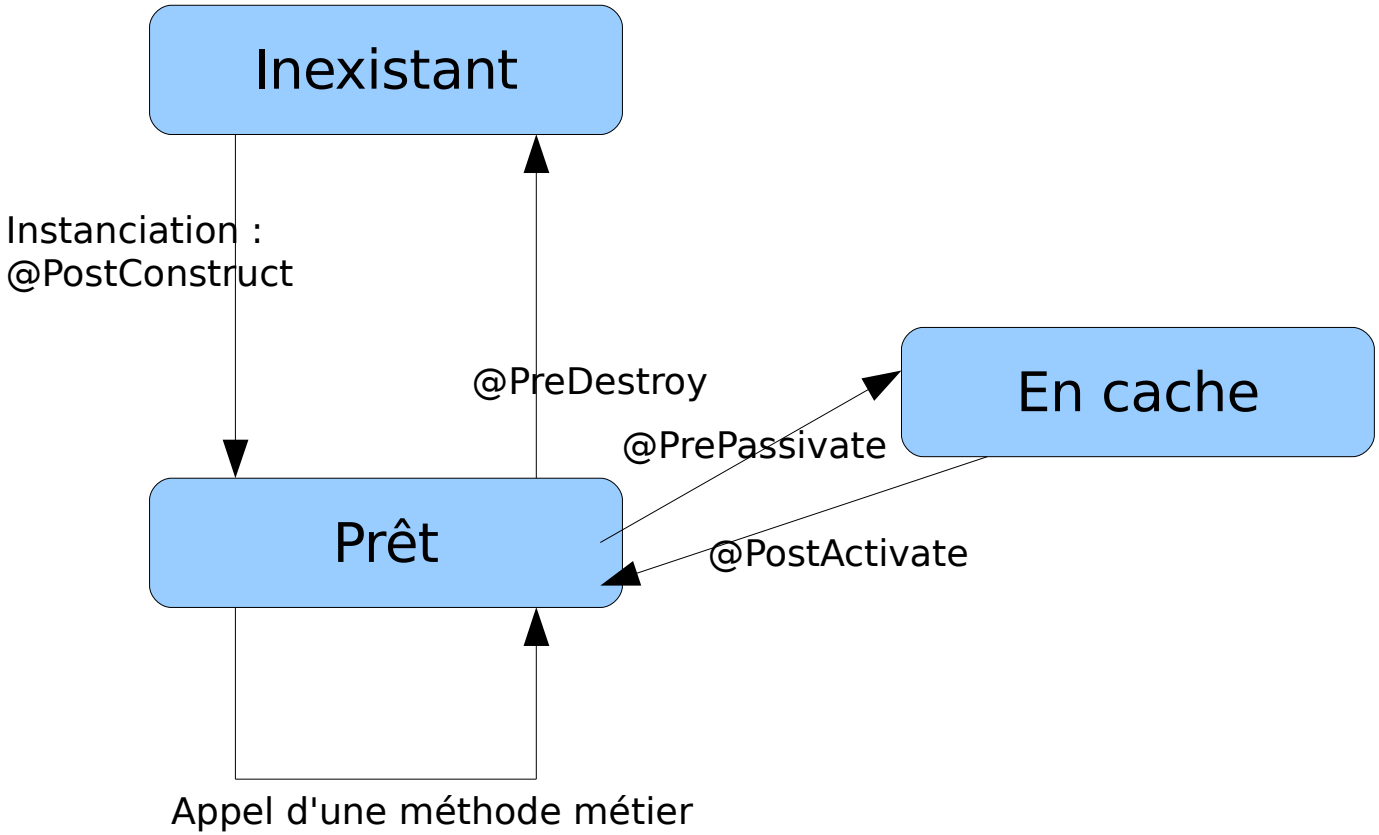
Les composants session avec état

- Annotés par `jakarta.ejb.Stateful`
- Après la première invocation de méthode, le composant est associé au client.
- Possibilité d'utiliser des variables d'instances entre deux appels de méthodes.
- Le conteneur peut décider de mettre le composant en mémoire secondaire (montée en charge) : mécanisme de passivation et activation.

Les composants session avec état

- Annotés par `jakarta.ejb.Stateful`
- Après la première invocation de méthode, le composant est associé au client.
- Possibilité d'utiliser des variables d'instances entre deux appels de méthodes.
- Le conteneur peut décider de mettre le composant en mémoire secondaire (montée en charge) : mécanisme de passivation et activation.
- Une méthode sans paramètre annotée par `jakarta.ejb.Remove` notifie le conteneur que le client n'a plus besoin du bean.

Cycle de vie des composants session avec état



Exemple d'un composant Stateful

```
package ejb.sessions ;
@jakarta.ejb.Stateful public class SalaireBean
  implements SalaireInterfaceRemote, SalaireInterfaceLocal {
  private double tauxHoraire=8.03 ;

  public SalaireBean() {}
  @Override public void setTauxHoraire(double taux) {
    this.tauxHoraire=taux ;
  }
  @Override public double getTauxHoraire() {
   return this.tauxHoraire ;
  }
  @Override public double getSalaire(int nbreHeures) {
    return this.getTauxHoraire()*nbreHeures ;
  }
}
```

Serveur Wildfly, règle de nommage JNDI

Dans le cas des sessions `Stateful` accessibles à distance, le serveur wildfly génère l'adresse JNDI suivante :

```
String appName="" , moduleName="salaireSessions" ;
String distinctName="" , beanName="SalaireBean" ;
String remoteInterfaceName=
       SalaireInterfaceRemote.class.getName();
```

```
String adresseJNDI="ejb:"+appName+"/"+moduleName+
       "/"+distinctName+"/"+beanName+
       "!"+remoteInterfaceName ?stateful
```

Sécurité

- Par défaut, les méthodes du bean sont accessibles à toutes les applications clientes
- Possibilité de paramétrer/restreindre les accès à ces méthodes.
- Tout serveur JEE peut gérer des utilisateurs, des groupes d'utilisateurs, fournir une interface pour s'authentifier.
- Annotations Java pour définir les droits d'accès. Exemple :

```
@RolesAllowed("Admin")
public void foo(int value) { ...}

@PermitAll String toString() {...}
```

Transactions

- Par défaut, chaque méthode forme une transaction.
- Tout serveur JEE intègre donc un moniteur transactionnel.
- Annotations Java pour paramétrer les transactions. Exemple :

```
@TransactionAttribute
   (value=TransactionAttributeType.REQUIRED)
public void setTauxHoraire(double taux) {
   this.tauxHoraire=taux ;
}
```

Programmation d'entités au sein de sessions

- Sessions et entités s'exécutent au sein d'un conteneur d'EJB
- Le code métier (sessions) va exploiter un gestionnaire d'entités du conteneur EJB pour assurer la persistance des entités.

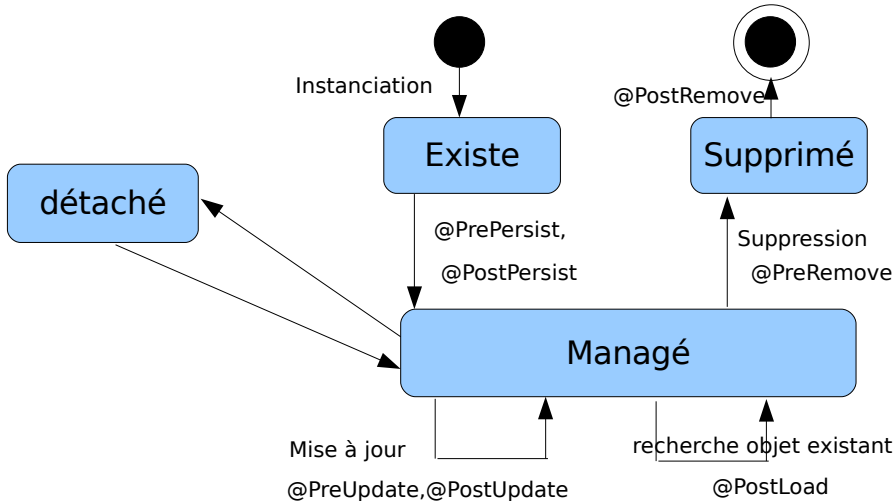
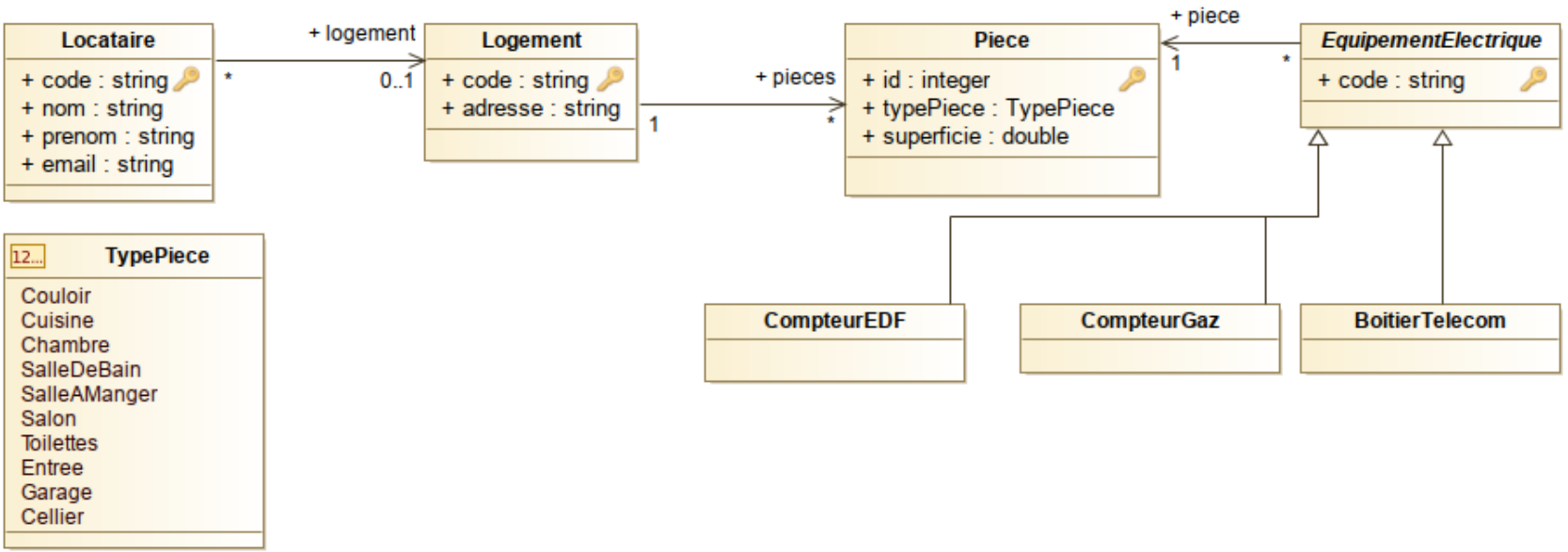


Figure – Cycle de vies des entités EJB



Gestion d'une agence immobilière

L'agence gère des locataires affectés à un logement. Les logements disposent de pièces. Des équipements nécessaires à la gestion (relevés de compteur) sont localisés dans les pièces des logements.

Le gestionnaire d'entités

- Une seule classe : `jakarta.persistence.EntityManager`
- Assure le lien entre objets Java et données BD
- Déclaration :

```
1  @Stateless public class GestionAgence implements IGestionAgence{
2    @PersistenceContext(unitName="agence-unit")
3    protected EntityManager em;
```

- Ligne 2 : "**agence-unit**" provient de `persistence.xml` d'un module ejb entité. Le gestionnaire d'entités sera reliée à la source de données (BD) spécifiée dans ce descripteur.
- Ligne 3 : c'est le serveur qui associe la variable `em` au gestionnaire d'entités (pas d'instanciation à faire par le programmeur)
- `em` ne peut être employé que dans un contexte transactionnel (c'est le cas des sessions)

Serveur JEE - Transactions par défaut

- **Par défaut**, une méthode au sein d'un composant session équivaut à une transaction.
- Le code suivant :

```
1  public void m() {
2    /* début code de m() */ ... /* fin code de m() */
3  }
```

- est donc équivalent à :

```
1  public void m() {
2    em.getTransaction().begin() ;
3    /* début code de m() */ ... /* fin code de m() */
4    em.getTransaction().commit() ;
5  }
```

- Par défaut : les transactions sont gérées par le conteneur EJB.

Fonctionnalités de base du gestionnaire d'entités

Fonction	Description	équivalent BD
`find`	Recherche d'instance en base guidée par clé primaire	`select`
`persist`	Ajout d'une instance en base	`insert`
`merge`	Synchronisation objet Java→ objet BD	`insert` et `update`
`flush`	Écriture des modifications en base avant fin de transaction	`insert`,`update` et `delete`
`remove`	Suppression d'instance en base	`delete`
`query`	Actions sur groupe d'objets en base	`select`,`update` et `delete`

EntityManager : récupération d'un objet BD ➔objet Java

- Pré-requis : l'objet existe dans la base.
- méthode `find`, deux paramètres requis :
  1. le type de la classe `@Entity` recherchée
  2. la valeur de la clé primaire (`@Id`)
- l'instance récupérée se trouve à l'état **géré**
- renvoie `null` si l'instance n'existe pas dans la base

```
Locataire loc ;
loc = (Locataire) em.find(Locataire.class,"loc227");
if (loc==null) System.err.println("locataire loc227 inconnu");
else loc.setEmail("jdupont@laposte.net");
     // sera modifié en base car état géré
```

EntityManager : insertion d'un objet dans la base

- Pré-requis : l'objet n'existe **pas** dans la base (sinon erreur lors du commit).

```
Logement log = new Logement(); log.setCode("LD_221b_BS");
log.setAdresse("221b Baker Street, london");
em.persist(log);
}
```

- Méthode `persist(Object instance)`
- Exception `IllegaleArgumentException` si l'objet n'est pas une entité

état de l'instance avant	**après**
état nouveau	état géré
état géré	état géré (pas de changement)
état détaché/non géré	exception lors du commit
état supprimé	état géré

EntityManager : suppression d'un objet entité

```
em.remove(log);
```

- Méthode `remove(Object instance)` :

état de l'instance avant	**après**
état nouveau	action ignorée
état géré	état supprimé
état détaché	une exception intervient
état supprimé	action ignorée

EntityManager : fusion d'un objet entité

```
em.merge(log);
```

- Propagation de l'état détaché vers la base (pour les attributs autre que la clé primaire)
- Méthode `merge(Object instance)` :

état de l'instance avant	**après**
état nouveau	état géré (équivalent `persist`)
état géré	action ignorée (déjà synchronisé)
état détaché	propagation base, état géré
état supprimé	action ignorée

EntityManager : Mise à jour des écritures en base

```
em.flush();
```

- Les opérations telles que `persist`, `remove`, `merge` ont pour effet de modifier la base mais ces écritures en base ne sont réalisées qu'à la fin de la transaction (par défaut, à la fin de la méthode du session)
- La commande `flush()` provoque ces écritures en base
- Peut donc déclencher des exceptions SQL (duplication de clé primaire, etc) qu'il est alors possible de capturer.

Traitement et gestion des exceptions liées à la base de données

- Au niveau du serveur JEE :

Traitement et gestion des exceptions liées à la base de données

- Au niveau du serveur JEE :
  - Par défaut, c'est le conteneur EJB qui termine la transaction. Les erreurs liées à la base de données ne sont pas gérables par le développeur EJB

Traitement et gestion des exceptions liées à la base de données

- Au niveau du serveur JEE :
  - Par défaut, c'est le conteneur EJB qui termine la transaction. Les erreurs liées à la base de données ne sont pas gérables par le développeur EJB
  - Seule la commande `flush` permet d'intercepter ces potentielles erreurs.

Traitement et gestion des exceptions liées à la base de données

- Au niveau du serveur JEE :
  - Par défaut, c'est le conteneur EJB qui termine la transaction. Les erreurs liées à la base de données ne sont pas gérables par le développeur EJB
  - Seule la commande `flush` permet d'intercepter ces potentielles erreurs.
- Au niveau d'un programme client distant :

Traitement et gestion des exceptions liées à la base de données

- Au niveau du serveur JEE :
  - Par défaut, c'est le conteneur EJB qui termine la transaction. Les erreurs liées à la base de données ne sont pas gérables par le développeur EJB
  - Seule la commande `flush` permet d'intercepter ces potentielles erreurs.
- Au niveau d'un programme client distant :
  - Quelque soit l'erreur au niveau du serveur, le client reçoit une unique exception de type "runtime" : `jakarta.ejb.EJBException`

Traitement et gestion des exceptions : illustrations (1/3)

Exception déclenchée lors du commit (implicite) et pas lors du `persist` :

code client	code EJB session
`try { System.out.print("create l1 "); serv.newLogement("l1", "adr"); System.out.print("create l1 "); serv.newLogement("l1", "adr"); System.out.print("success"); } catch(EJBException e) { System.err.print("failed"); }`	`public void newLogement (String code, String adr) { Logement l = new Logement(); l.setCode(code); l.setAdresse(adr) ; em.persist(l) ; System.out.print(" ok:"+code); }`
console cliente	logs du serveur JEE
create l1 create l1 failed	ok :l1 ok :l1 SQLException ...

Traitement et gestion des exceptions : illustrations (2/3)

Impossible de capturer l'exception `SQLException`

code client	code EJB session
`try { System.out.print("create l1 "); serv.newLogement("l1", "adr"); System.out.print("create l1 "); serv.newLogement("l1", "adr"); System.out.print("success"); } catch(EJBException e) { System.err.print("failed"); }`	`public void newLogement (String code, String adr) { try { Logement l = new Logement(); l.setCode(code); l.setAdresse(adr); em.persist(l); System.out.print(" ok:"+code); } catch(Exception e){ System.err.println("failed"); } }`
console cliente	logs du serveur JEE
create l1 create l1 failed	ok :l1 ok :l1 SQLException ...

Traitement et gestion des exceptions : illustrations (3/3)

On force l'écriture en base (`flush`)

code client	code EJB session

`try {`
`  System.out.print("create l1 ");`
`  serv.newLogement("l1", "adr");`
`  System.out.print("create l1 ");`
`  serv.newLogement("l1", "adr");`
`  System.out.print("success");`
`} catch(EJBException e) {`
`  System.err.print("failed");`
`}`
	`public void newLogement`
	`(String code, String adr) {`
	`  try {`
	`  Logement l = new Logement();`
	`  l.setCode(code);`
	`  l.setAdresse(adr);`
	`  em.persist(l); em.flush();`
	`  System.out.print(" ok:"+code);`
	`  } catch(Exception e){`
	`  System.err.println("failed");`
	`  }`
	`}`

console cliente	logs du serveur JEE
create l1 create l1 success	ok :l1 SQLException failed

Traitement et gestion des exceptions

En résumé, anticipez les erreurs et programmez les exceptions. Le bon "code pattern" est :

```
public void newLogement (String code, String adr)
        throws LogementExisteDejaException {
   Logement l = (Logement) em.find(Logement.class,code) ;
   if (l!=null) throw new LogementExisteDejaException() ;
   l = new Logement();
   l.setCode(code); l.setAdresse(adr);
   em.persist(l);
}
```

EntityManager : Requêtes JPQL

- Un langage de requêtes défini dans la norme : JPQL

EntityManager : Requêtes JPQL

- Un langage de requêtes défini dans la norme : JPQL
- JPQL se rapproche de plus en plus de SQL : introduction de group by, having, sous-requêtes, etc dans la dernière version.

EntityManager : Requêtes JPQL

- Un langage de requêtes défini dans la norme : JPQL
- JPQL se rapproche de plus en plus de SQL : introduction de group by, having, sous-requêtes, etc dans la dernière version.
- JPQL et SQL possèdent pratiquement la même syntaxe.

EntityManager : Requêtes JPQL

- Un langage de requêtes défini dans la norme : JPQL
- JPQL se rapproche de plus en plus de SQL : introduction de group by, having, sous-requêtes, etc dans la dernière version.
- JPQL et SQL possèdent pratiquement la même syntaxe.
- Les jointures JPQL sont conformes SQL-2 (left, right, outer join)

EntityManager : Requêtes JPQL

- Un langage de requêtes défini dans la norme : JPQL
- JPQL se rapproche de plus en plus de SQL : introduction de group by, having, sous-requêtes, etc dans la dernière version.
- JPQL et SQL possèdent pratiquement la même syntaxe.
- Les jointures JPQL sont conformes SQL-2 (left, right, outer join)
- Introduction des requêtes update et delete

EntityManager : Requêtes JPQL

- Un langage de requêtes défini dans la norme : JPQL
- JPQL se rapproche de plus en plus de SQL : introduction de group by, having, sous-requêtes, etc dans la dernière version.
- JPQL et SQL possèdent pratiquement la même syntaxe.
- Les jointures JPQL sont conformes SQL-2 (left, right, outer join)
- Introduction des requêtes update et delete
- **JPQL exprime des requêtes sur les objets Java pas sur les objets de la base**

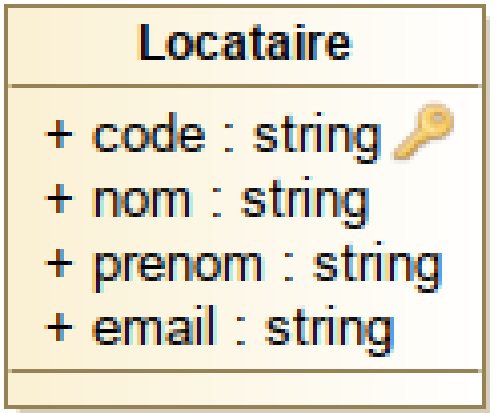
EntityManager : programmation des requêtes

```
1 String reqJPQL = "select ..." ;      // définition de la requête
2 Query q = em.createQuery(reqJPQL) ; // création requête
3 q.setParameter("nomPar1", valeur) ; // optionel :
4 q.setParameter("nomPar2", valeur) ; // définir paramètres
5 Object result = q.getResultList();  // exécution et résultat
```

- Méthode : `createQuery(String requeteJPQL)`, fournit un objet de type `query` (ligne 2)
- Possibilité de fournir des valeurs aux éventuels paramètres de la requête JPQL via la méthode `setParameter` (lignes 3-4)
- Selon la requête, l'exécution fournit :
  - une unique ligne résultat via la méthode `getSingleResult()`
  - une collection de lignes résultats via la méthode `getResultList()` (ligne 5).
- Variable `result` devra être convertie selon le bon type (ligne 5).

EntityManager : requête mono-table paramétrée (version 1)

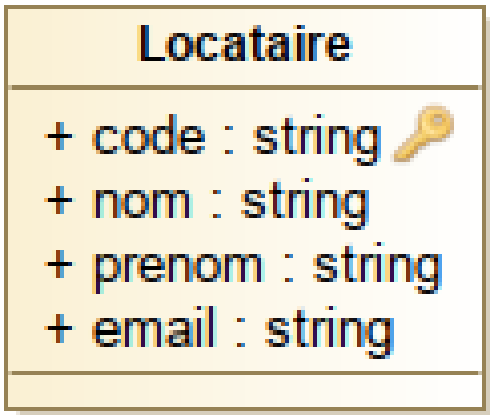
- Ex : liste des locataires filtrés par leur nom.
- JPQL : langage de requêtes sur objets **Java** : classe `Locataire` et attribut `nom` (ligne 3).
- Cette requête retourne une collection de locataires, **cast explicite** (lignes 1 et 8)




```
1  @SuppressWarnings("unchecked")
2  public Collection<Locataire>
3         getLocatairesParNom(String debutMot) {
4    String reqJPQL="select loc from Locataire loc "+
5                   " where loc.nom like :pattern" ;
6    Query q=em.createQuery(reqJPQL) ;
7    q.setParameter("pattern",debutMot+"%");
8    return (Collection<Locataire>) q.getResultList();
9  }
```

EntityManager : requête mono-table paramétrée (version 2)

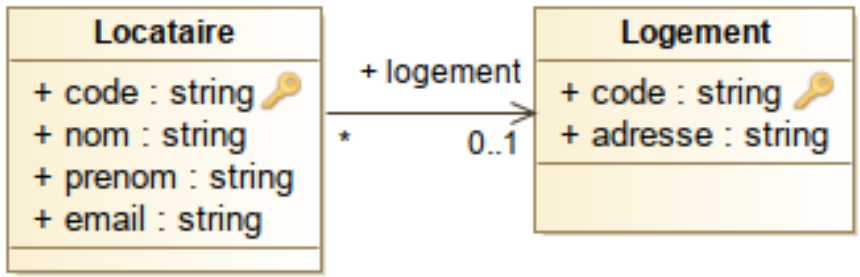
- La même requête que la diapositive précédente mais :
- Utilisation de `TypedQuery` et ajout paramètre à `createQuery` (ligne 4)
- Ne fonctionne que pour requêtes multi-tables avec jointures explicites.
- L'annotation `@SuppressWarnings` peut être retirée.




```
1  public Collection<Locataire>
2          getLocatairesParNomV2(String debutMot){
3   String reqJPQL="select loc from Locataire loc "+
4                  " where loc.nom like :pattern" ;
5   TypedQuery<Locataire> q=em.createQuery(reqJPQL,Locataire.class) ;
6   q.setParameter("pattern",debutMot+"%");
7   return (Collection<Locataire>) q.getResultList();
8  }
```

EntityManager : requête jointure - version 1

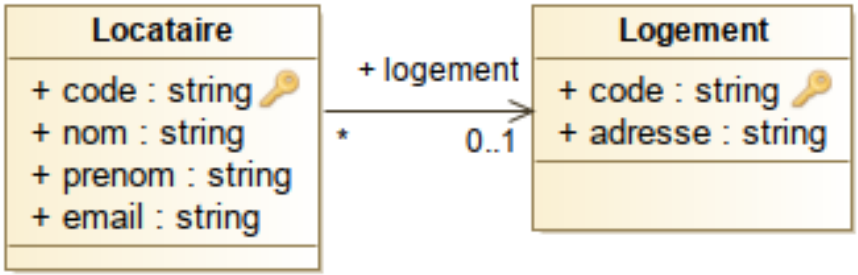
- Ex : liste des logements occupés.
- Sol 1 : navigation par rôle (jointure non explicite)
- Syntaxe JPQL (ex ligne 4) : `variable.nomRole`




```
1  @SuppressWarnings("unchecked")
2  @Override
3  public Collection<Logement> getLogementsOccupesV1() {
4   String reqJPQL="select loc.logement from Locataire loc" ;
5   Query q=em.createQuery(reqJPQL);
6   return (Collection<Logement>)q.getResultList();
7  }
```

EntityManager : requête jointure - version 2

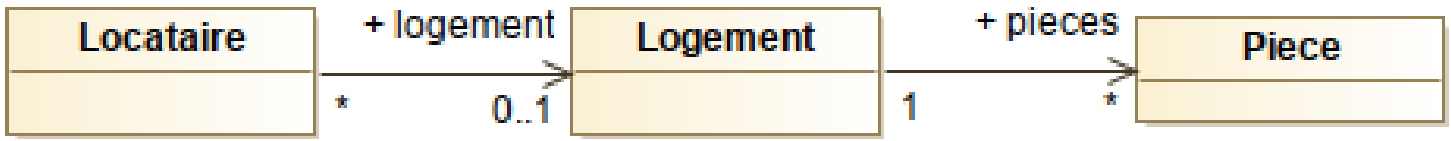
- Ex : liste des logements occupés.
- Sol 2 : opérateur `join`
- Syntaxe JPQL (ex ligne 4) :
  `var1 join var1.role var2`
- Jointure interne, externe,...
- Jointure explicite : `TypedQuery` possible




```
1  public Collection<Logement> getLogementsOccupesV2() {
2   String reqJPQL="select logementsOccupes from Locataire "+
3                  "loc join loc.logement logementsOccupes"   ;
4   TypedQuery<Logement> q=em.createQuery(reqJPQL, Logement.class);
5   return (Collection<Logement>)q.getResultList();
6  }
```

EntityManager : requête multi jointures - version 1

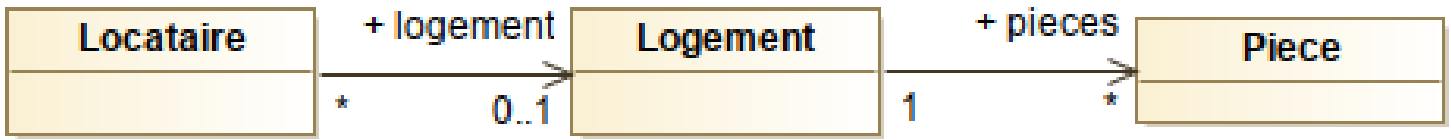
- Ex : liste des pièces des logements occupés.
- Sol 1 : navigation par rôles




```
1  @SuppressWarnings("unchecked")
2  public Collection<Piece> getPiecesLogementsOccupesV1() {
3    String reqJPQL="select loc.logement.pieces from Locataire loc" ;
4    Query q=em.createQuery(reqJPQL);
5    return (Collection<Piece>) q.getResultList();
6  }
```

EntityManager : requête multi jointures - version 2

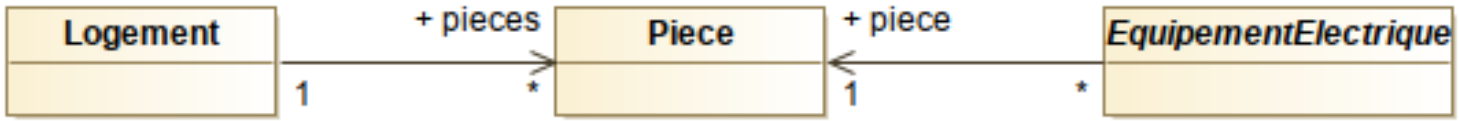
- Ex : liste des pièces des logements occupés.
- Sol 2 : opérateur `join`




```
1  @SuppressWarnings("unchecked")
2  public Collection<Piece> getPiecesLogementsOccupesV2() {
3    String reqJPQL="select pieces from Locataire loc "+
4                   "join loc.logement logementsOccupes "+
5                   "join logementsOccupes.pieces pieces";
6    TypedQuery<Piece> q=em.createQuery(reqJPQL,Piece.class);
7    return (Collection<Piece>) q.getResultList();
8  }
```

EntityManager : requête jointure - navigation inverse

- Association non navigable (ex : `Piece` vers `EquipementElectrique`) :



- Unique solution : opérateur `join` et clause `on` :

```
var1 join Entite2 var2 on var2.role=var1
```

```
1  public Collection<EquipementElectrique>
2    getEquipementsDuLogement(String codeLogement)
3      throws LogementInconnuException {
4    Logement log= (Logement) em.find(Logement.class, codeLogement);
5    if (log==null) throw new LogementInconnuException() ;
6    String reqJPQL="select equip from Logement lo join lo.pieces p "+
7      "join EquipementElectrique equip on equip.piece=p where lo.code=:code";
8    TypedQuery<EquipementElectrique> q ;
9    q=em.createQuery(reqJPQL,EquipementElectrique.class) ;
10   q.setParameter("code", codeLogement);
11   return (Collection<EquipementElectrique>) q.getResultList();
12 }
```

EntityManager : requêtes JPQL avec fonctions d'agrégation

- Possibilité d'utiliser les fonctions d'agrégation de SQL
- Récupération d'un seule ligne ou valeur via la méthode `getSingleResult()`
- Prendre en compte `NoResultException` (erreur Runtime)
- Un exemple :

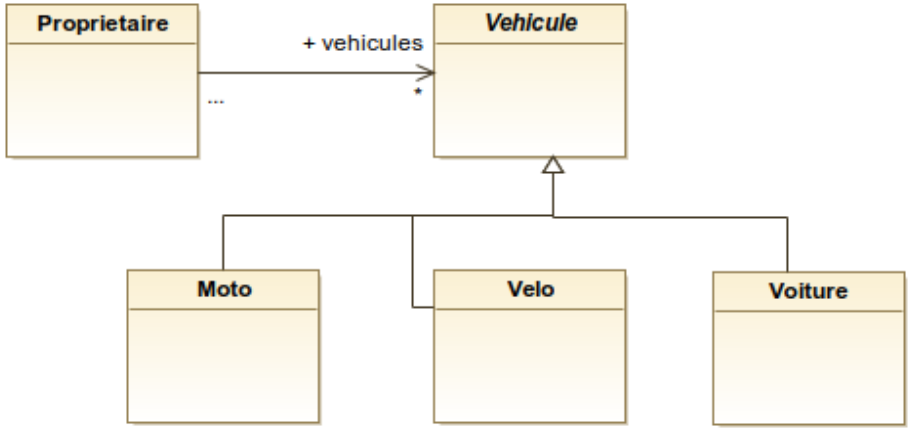
```
1  public double getSuperficieLogementsOccupes()
2    throws NoResultException {
3    String reqJPQL="select sum(p.superficie) "+
4    "from Locataire loc join loc.logement lo join lo.pieces p";
5    Query q=em.createQuery(reqJPQL);
6    return (Double) q.getSingleResult();
7  }
```

EntityManager : multi valeurs

- Possibilité de fixer une liste d'expressions dans la clause `select`
- Récupération des données via `List<Object []>`
- Exemple du nombre de pièces par logement :

```
String reqJPQL="select lo, count(*) as nombre "+
                "from Logement lo join lo.pieces p group by lo";
Query q=em.createQuery(reqJPQL) ;
List<Object[]> resultat= (List<Object []>) q.getResultList();
System.out.println("Les logements et leur pièces:");
for (Object ligne[] : resultat) {
  Logement lg= (Logement) ligne[0] ;
  long nb = (Long) ligne[1];
  System.out.println("Code "+lg.getCode()+": "+nb+" pièce(s)");
}
```

EntityManager : requête avec gestion héritage



- Obtenir une hiérarchie de classes :
  `select v from Vehicule v`
  On obtient toutes les instances des 3-sous-classes
- Obtenir une sous-classe particulière :
  `select m from Moto m`
- Obtenir une partie de la hiérarchie de classes :
  `select m from Proprietaire a join a.vehicules m where TYPE(m)=Moto`
  On obtient les seules instances de `Moto`

EntityManager : requêtes JPQL de modification

- Pas besoin de charger des objets Java pour modifier des données en base
- Méthode `executeUpdate()`
- Retourne le nombre de ligne(s) concernées par l'opération
- Un exemple :

```
public int supprimerLocataires() {
  int nbrePersonnesSupprimees =
    em.createQuery("delete from Locataire").executeUpdate() ;
  return nbrePersonnesSupprimees;
}
```

Conclusion JPQL

- Ce cours est une **introduction** à JPQL

Conclusion JPQL

- Ce cours est une **introduction** à JPQL
- Pouvoir d'expression des requêtes (sous-requêtes, clauses exists, all, any, group by, having, ...)

Conclusion JPQL

- Ce cours est une **introduction** à JPQL
- Pouvoir d'expression des requêtes (sous-requêtes, clauses exists, all, any, group by, having, ...)
- Permet d'obtenir une sélection d'objets Java sans charger au préalable en mémoire tous les éléments nécessaires de la requête.

Conclusion JPQL

- Ce cours est une **introduction** à JPQL
- Pouvoir d'expression des requêtes (sous-requêtes, clauses exists, all, any, group by, having, ...)
- Permet d'obtenir une sélection d'objets Java sans charger au préalable en mémoire tous les éléments nécessaires de la requête.
- Le modèle UML-EJB est le guide parfait pour l'élaboration des requêtes JPQL (nom des classes, attributs et rôles)

Conclusion JPQL

- Ce cours est une **introduction** à JPQL
- Pouvoir d'expression des requêtes (sous-requêtes, clauses exists, all, any, group by, having, ...)
- Permet d'obtenir une sélection d'objets Java sans charger au préalable en mémoire tous les éléments nécessaires de la requête.
- Le modèle UML-EJB est le guide parfait pour l'élaboration des requêtes JPQL (nom des classes, attributs et rôles)
- Références :

  `https://jakarta.ee/specifications/persistence/3.2/jakarta-persistence-spec-3.2.pdf`

  `https://thorben-janssen.com/jpql/`

Développement d'applications JEE

- Une **application** JEE est constituée de plusieurs **modules**. On distingue :

Développement d'applications JEE

- Une **application** JEE est constituée de plusieurs **modules**. On distingue :
- Les modules entités : archive **jar** des composants entités avec descripteur `META-INF/persistence.xml`

Développement d'applications JEE

- Une **application** JEE est constituée de plusieurs **modules**. On distingue :
- Les modules entités : archive **jar** des composants entités avec descripteur `META-INF/persistence.xml`
- Les modules sessions : archive **jar** des composants sessions qui vont gérer des composants entités

Développement d'applications JEE

- Une **application** JEE est constituée de plusieurs **modules**. On distingue :
- Les modules entités : archive **jar** des composants entités avec descripteur `META-INF/persistence.xml`
- Les modules sessions : archive **jar** des composants sessions qui vont gérer des composants entités
- Les modules webs : archive **war** des composants webs qui vont interagir avec les composants sessions de l'application.

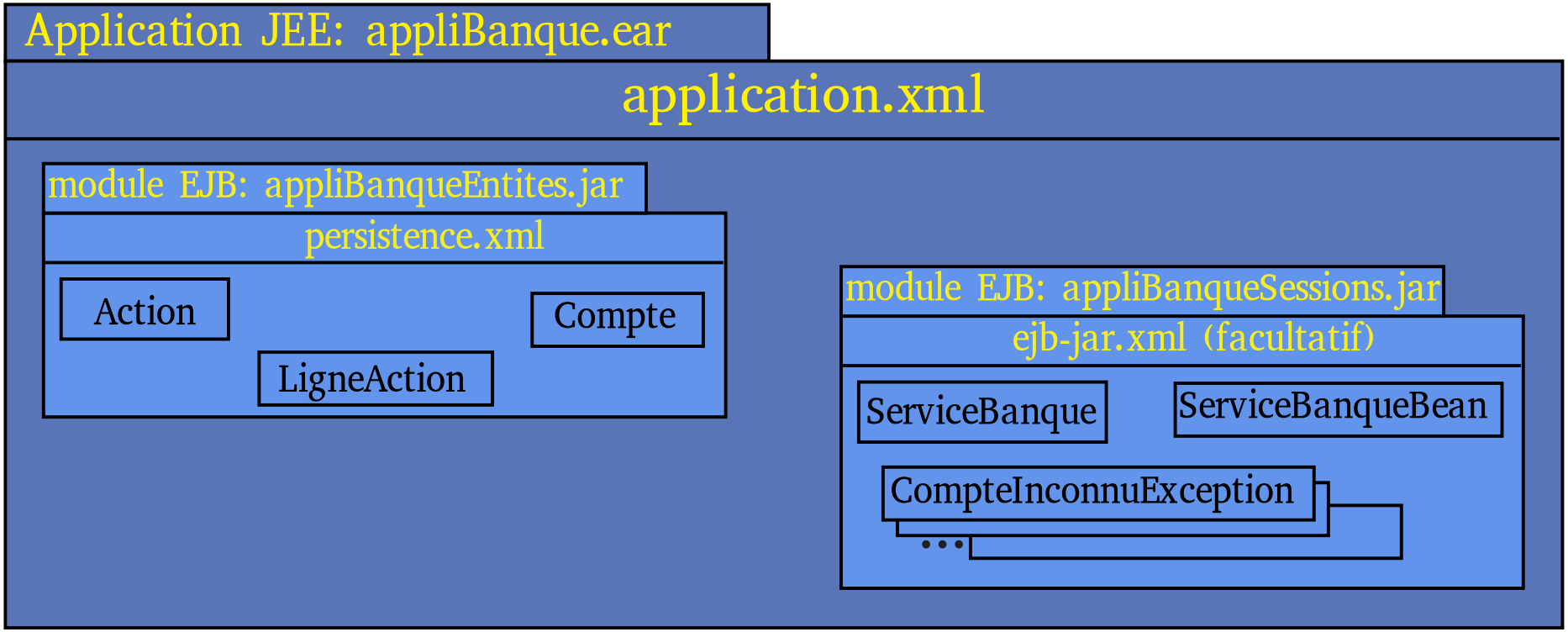
Développement d'applications JEE

- Une **application** JEE est constituée de plusieurs **modules**. On distingue :
- Les modules entités : archive **jar** des composants entités avec descripteur `META-INF/persistence.xml`
- Les modules sessions : archive **jar** des composants sessions qui vont gérer des composants entités
- Les modules webs : archive **war** des composants webs qui vont interagir avec les composants sessions de l'application.

Définition d'une application JEE

Une application JEE est une unité de déploiement. **C'est une archive qui contient des archives**. Le descripteur `META-INF/application.xml` décrit le contenu de l'archive.

Un exemple d'application : l'application Banque



Code complet : `https://gitlab.univ-lille.fr/olivier.caron/democoursal.git`

Descripteur application.xml

Contenu archive :

```
appliBanque/
 META-INF/
  application.xml
 appliBanqueSessions.jar
 appliBanqueEntites.jar
```

Fichier application.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="https://jakarta.ee/xml/ns/jakartaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/application_10.xsd"
  version="10">

    <display-name>Appli Banque</display-name>

    <module>
      <ejb>appliBanqueSessions.jar</ejb>
    </module>
    <module>
      <ejb>appliBanqueEntites.jar</ejb>
    </module>
</application>
```

```
jar cvf appliBanque.ear appliBanque/*
```

L'interface distante du composant session (1/2)

```
package ejb.sessions;

import ejb.entites.LigneAction ;

@jakarta.ejb.Remote public interface ServiceBanque {
  public void addCompte(int noCompte, String nomTitulaire,
                        double soldeDepart)
    throws CompteDejaExistantException ;
  public void addAction(String nomAction, double taux)
    throws ActionDejaExistanteException ;
  public void crediterCompte(int noCompte, double montant)
    throws CompteInconnuException ;
  public void debiterCompte(int noCompte, double montant)
    throws CompteInconnuException ;
```

L'interface distante du composant session (2/2)

```
public void virementVers(int noCompteDebit, int noCredit,
                         double montant)
 throws CompteInconnuException, ApprovisionnementException ;
 public void acheteActions(int noCompte, String nomAction, int nb)
   throws CompteInconnuException, ActionInconnueException,
          ApprovisionnementException ;
 public void vendActions(int noCompte, String nomAction, int nb)
   throws CompteInconnuException, ActionInconnueException,
   ApprovisionnementException ;
 public java.util.Set<LigneAction> getLignesActions(int noCompte)
   throws CompteInconnuException ;
}
```

Le composant session (1/3)

```
package ejb.sessions;

import ...

@jakarta.ejb.Stateless
public class ServiceBanqueBean implements ServiceBanque {

   @PersistenceContext(unitName="appliBanque")
   protected EntityManager em ;

   public ServiceBanqueBean() { }
```

Le composant session(2/3)

```
public Set<LigneAction> getLignesActions(int noCompte)
  throws CompteInconnuException {
  Compte c = this.getCompte(noCompte) ;
  return c.getLignesActions() ;
}

private Compte getCompte(int noCompte)
throws CompteInconnuException {
  Compte c= (Compte) em.find(Compte.class, noCompte) ;
  if (c==null) throw new CompteInconnuException() ;
  return c;
}
```

Le composant session (3/3)

```
  public void addCompte(int noCompte, String nomTitulaire,
    double soldeDepart) throws CompteDejaExistantException {
    try {
      this.getCompte(noCompte) ;
      throw new CompteDejaExistantException() ;
    } catch(CompteInconnuException e) {
      Compte c=new Compte() ;
      c.setNumero(noCompte); c.setSolde(soldeDepart);
      c.setTitulaire(nomTitulaire);
      em.persist(c);
    }
  ...
}
```

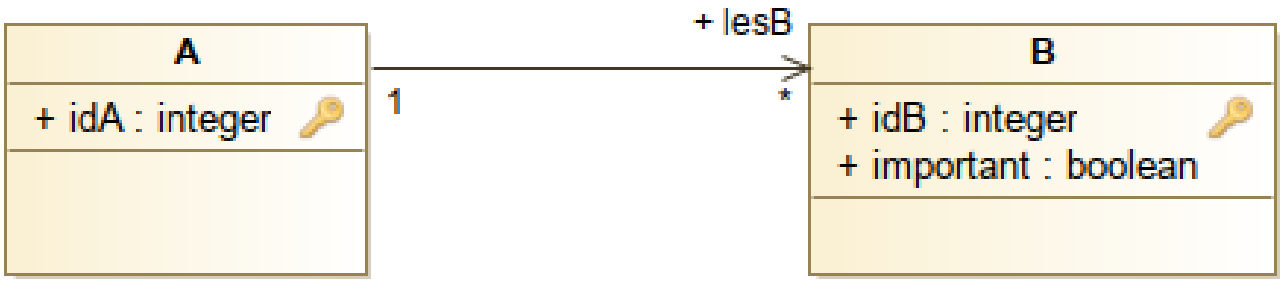
Utilisation du service

```
1  public static void main(String[] args) {
2   try {
3     InitialContext ctx = new InitialContext();
4     System.out.println("Accès au service distant") ;
5     Object obj ;
6     obj = ctx.lookup("ejb:appliBanque/appliBanqueSessions//"
7           +"ServiceBanqueBean!ejb.sessions.ServiceBanque");
8     ServiceBanque service = (ServiceBanque) obj ;
9     System.out.println("les actions du compte no: 1") ;
10    for (LigneAction la : service.getLignesActions(1))
11      System.out.println(la.getNombre()+" action(s) "+
12                         la.getAction().getNom()+
13                         " au taux de "+la.getAction().getTaux());
14        ...
```

- Potentiel problème `LazyInitialisationException` à partir de la ligne 10
- Distinction entre entités gérées et détachées (cas de ce programme).

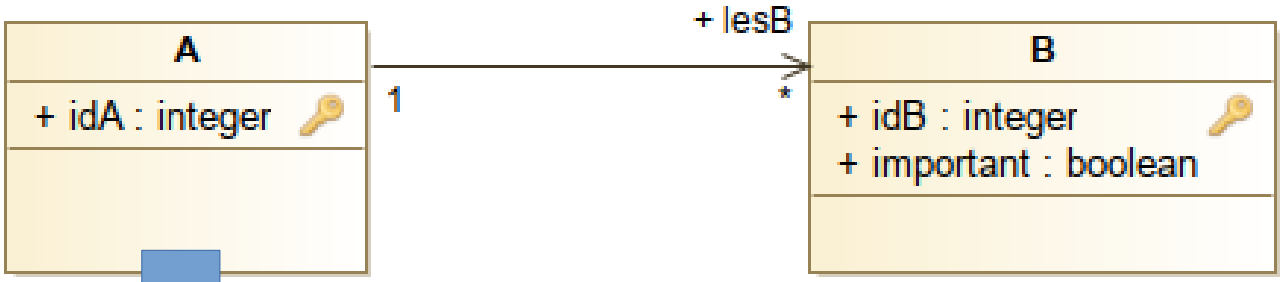
Entités gérées vs détachées (1/4)

Partie entités :



Entités gérées vs détachées (1/4)

Partie entités :




```
@Entity
public class A implements java.io.Serializable {
  @Id private int idA ;
  private @OneToMany Set<B> lesB ;

  public A() {}
   ...
```

Entités gérées vs détachées (2/4)

Partie sessions (chargement des instances **si nécessaire**)

```
@jakarta.ejb.Remote
public interface IGestion {
  /** these methods return null if not exists **/
  public A getAIf_lesB_are_important(int idA)  ;
  public A getA(int idA)  ;
}
```

Entités gérées vs détachées (2/4)

Partie sessions (chargement des instances **si nécessaire**)

```java
@jakarta.ejb.Remote
public interface IGestion {
  /** these methods return null if not exists **/
  public A getAIf_lesB_are_important(int idA)  ;
  public A getA(int idA)  ;
}
```

```java
@Override
public A getAIf_lesB_are_important(int idA) {
  A a = this.getA(idA) ;
  if (a != null) {
    for (B unB : a.getLesB())
      if (! unB.isImportant()) return null ;
    return a ;
  }
  return null ;
}
@Override public A getA(int idA) {
  return (A) em.find(A.class,idA) ;
}
```

Entités gérées vs détachées (2/4)

Partie sessions (chargement des instances **si nécessaire**)

```java
@jakarta.ejb.Remote
public interface IGestion {
  /** these methods return null if not exists **/
  public A getAIf_lesB_are_important(int idA)  ;
  public A getA(int idA)  ;
}
```

```java
@Override
public A getAIf_lesB_are_important(int idA) {
  A a = this.getA(idA) ;
  if (a != null) {
    for (B unB : a.getLesB())
      if (! unB.isImportant()) return null ;
    return a ;
  }
  return null ;
}
@Override public A getA(int idA) {
  return (A) em.find(A.class,idA) ;
}
```

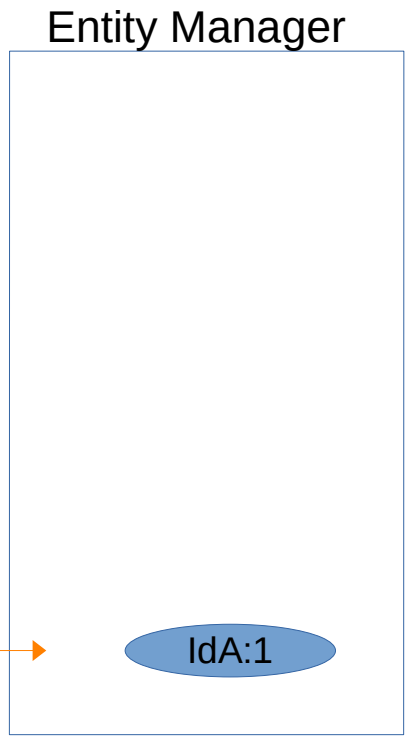
Entity Manager

Entités gérées vs détachées (2/4)

Partie sessions (chargement des instances **si nécessaire**)

```java
@jakarta.ejb.Remote
public interface IGestion {
  /** these methods return null if not exists **/
  public A getAIf_lesB_are_important(int idA)  ;
  public A getA(int idA)  ;
}
```

```java
@Override
public A getAIf_lesB_are_important(int idA) {
  A a = this.getA(idA) ;
  if (a != null) {
    for (B unB : a.getLesB())
      if (! unB.isImportant()) return null ;
    return a ;
  }
  return null ;
}
@Override public A getA(int idA) {
  return (A) em.find(A.class,idA) ;
}
```

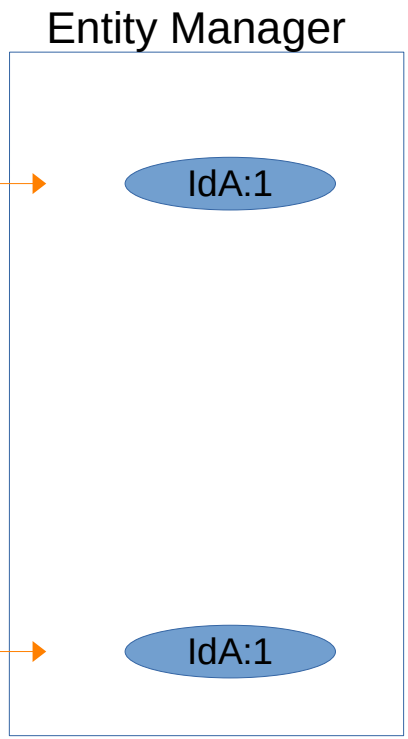


Entités gérées vs détachées (2/4)

Partie sessions (chargement des instances **si nécessaire**)

```java
@jakarta.ejb.Remote
public interface IGestion {
  /** these methods return null if not exists **/
  public A getAIf_lesB_are_important(int idA)  ;
  public A getA(int idA)  ;
}
```

```java
@Override
public A getAIf_lesB_are_important(int idA) {
  A a = this.getA(idA) ;
  if (a != null) {
    for (B unB : a.getLesB())
      if (! unB.isImportant()) return null ;
    return a ;
  }
  return null ;
}
@Override public A getA(int idA) {
  return (A) em.find(A.class,idA) ;
}
```

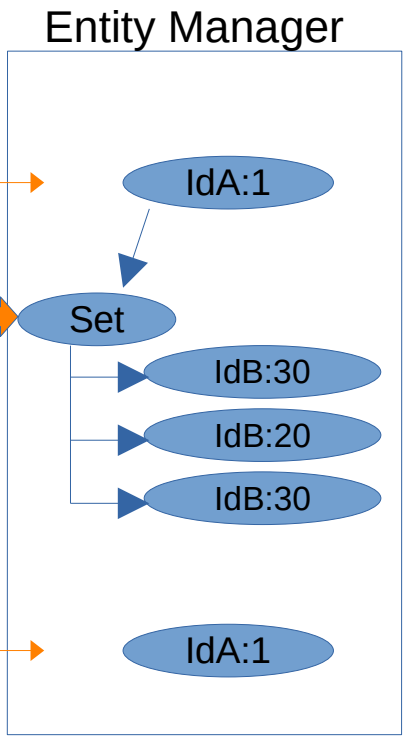


Entités gérées vs détachées (2/4)

Partie sessions (chargement des instances **si nécessaire**)

```java
@jakarta.ejb.Remote
public interface IGestion {
  /** these methods return null if not exists **/
  public A getAIf_lesB_are_important(int idA)  ;
  public A getA(int idA)  ;
}
```

```java
@Override
public A getAIf_lesB_are_important(int idA) {
  A a = this.getA(idA) ;
  if (a != null) {
    for (B unB : a.getLesB())
      if (! unB.isImportant()) return null ;
    return a ;
  }
  return null ;
}
@Override public A getA(int idA) {
  return (A) em.find(A.class,idA) ;
}
```



Entités gérées vs détachées (3/4)

Partie cliente distante : les entités propagées par le réseau sont **détachées**

```
InitialContext ctx = new InitialContext();
System.out.println("Accès au service distant") ;
Object obj = ctx.lookup("ejb:demo/demoSessions//Gestion"+
                        "!ejb.sessions.IGestion");

IGestion service = (IGestion) obj ;
A unA = service.getAIf_lesB_are_important(1) ;
if (unA !=null) {
  System.out.println ("1. unA.idA="+unA.getIdA()) ;
  for (B unB : unA.getLesB())
    System.out.println ("1. unB.idB="+unB.getIdB()) ;
}
unA = service.getA(1) ;
if (unA !=null) {
  System.out.println ("2. unA.idA="+unA.getIdA()) ;
  for (B unB : unA.getLesB())
    System.out.println ("2. unB.idB="+unB.getIdB()) ;
}
```

Entités gérées vs détachées (3/4)

Partie cliente distante : les entités propagées par le réseau sont **détachées**

```
InitialContext ctx = new InitialContext();
System.out.println("Accès au service distant") ;
Object obj = ctx.lookup("ejb:demo/demoSessions//Gestion"+
                        "!ejb.sessions.IGestion");

IGestion service = (IGestion) obj ;
A unA = service.getAIf_lesB_are_important(1) ;
if (unA !=null) {
  System.out.println ("1. unA.idA="+unA.getIdA()) ;
  for (B unB : unA.getLesB())
    System.out.println ("1. unB.idB="+unB.getIdB()) ;
}
unA = service.getA(1) ;
if (unA !=null) {
  System.out.println ("2. unA.idA="+unA.getIdA()) ;
  for (B unB : unA.getLesB())
    System.out.println ("2. unB.idB="+unB.getIdB()) ;
}
```

```
1. unA.idA=1
1. unB.idB=30
1. unB.idB=20
1. unB.idB=40
```

Entités gérées vs détachées (3/4)

Partie cliente distante : les entités propagées par le réseau sont **détachées**

```
InitialContext ctx = new InitialContext();
System.out.println("Accès au service distant") ;
Object obj = ctx.lookup("ejb:demo/demoSessions//Gestion"+
                        "!ejb.sessions.IGestion");

IGestion service = (IGestion) obj ;
A unA = service.getAIf_lesB_are_important(1) ;
if (unA !=null) {
  System.out.println ("1. unA.idA="+unA.getIdA()) ;
  for (B unB : unA.getLesB())
    System.out.println ("1. unB.idB="+unB.getIdB()) ;
}
unA = service.getA(1) ;
if (unA !=null) {
  System.out.println ("2. unA.idA="+unA.getIdA()) ;
  for (B unB : unA.getLesB())
    System.out.println ("2. unB.idB="+unB.getIdB()) ;
}
```

```
1. unA.idA=1
1. unB.idB=30
1. unB.idB=20
1. unB.idB=40
```

```
2. unA.idA=1
```

Entités gérées vs détachées (3/4)

Partie cliente distante : les entités propagées par le réseau sont **détachées**

```
InitialContext ctx = new InitialContext();
System.out.println("Accès au service distant") ;
Object obj = ctx.lookup("ejb:demo/demoSessions//Gestion"+
                        "!ejb.sessions.IGestion");

IGestion service = (IGestion) obj ;
A unA = service.getAIf_lesB_are_important(1) ;
if (unA !=null) {
  System.out.println ("1. unA.idA="+unA.getIdA()) ;
  for (B unB : unA.getLesB())
    System.out.println ("1. unB.idB="+unB.getIdB()) ;
}
unA = service.getA(1) ;
if (unA !=null) {
  System.out.println ("2. unA.idA="+unA.getIdA()) ;
  for (B unB : unA.getLesB())
    System.out.println ("2. unB.idB="+unB.getIdB()) ;
}
```

```
1. unA.idA=1
1. unB.idB=30
1. unB.idB=20
1. unB.idB=40
```

```
2. unA.idA=1
```

```
Exception in thread "main" org.hibernate.LazyInitializationException: ...
```

Entités gérées vs détachées (4/4)

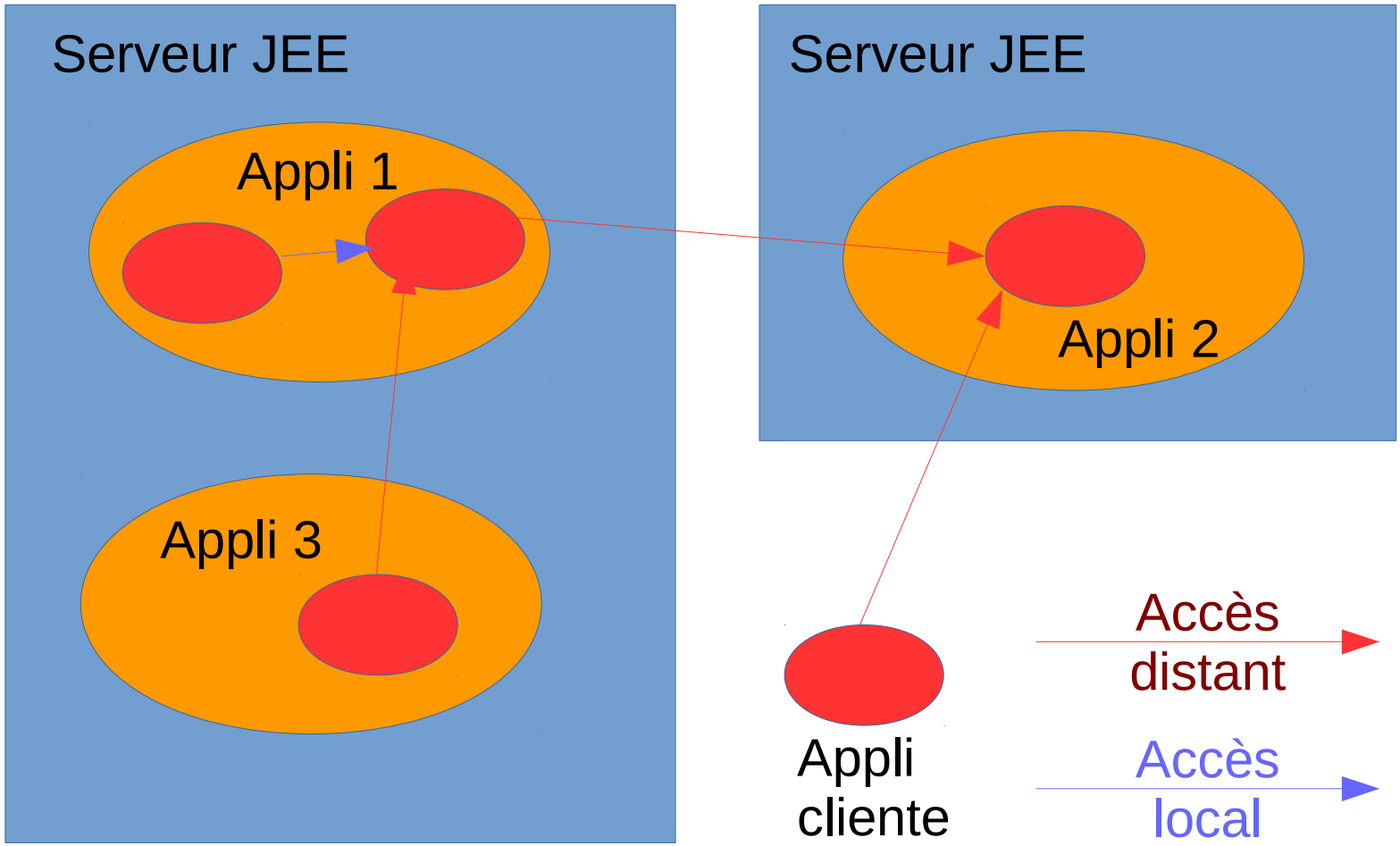
Niveau Serveur - état géré le mode par défaut d'obtention des instances Java d'entités en mémoire est le mode `LAZY`("paresseux" eq. "load on demand").

Niveau prog. client - état détaché risque de `java.lang.NullPointerException` ou de `org.hibernate.LazyInitialisationException` si les instances téléchargées sont incomplètes.

- Deux solutions :
  1. Forcer le chargement des objets avant l'envoi des données (méthode `getAIf_lesB_are_important`)
  2. Fixer le mode d'acquisition des données en mode `EAGER` :

```
// fichier A.java
...
private @OneToMany(fetch=FetchType.EAGER) Set<B> lesB ;
```

Accès distants ou locaux



Spécifications JNDI pour composants EJB 3.1

- La spécification EJB 3.1 comporte un nommage standardisé pour accéder aux composants sessions.

Spécifications JNDI pour composants EJB 3.1

- La spécification EJB 3.1 comporte un nommage standardisé pour accéder aux composants sessions.
- Dans un serveur JEE, plusieurs noms sont possibles et dépendent de la "distance" du composant.

Spécifications JNDI pour composants EJB 3.1

- La spécification EJB 3.1 comporte un nommage standardisé pour accéder aux composants sessions.
- Dans un serveur JEE, plusieurs noms sont possibles et dépendent de la "distance" du composant.
- Trois espaces de noms sont possibles : `global`, `application` et `module`

JNDI : espace de noms standard `Global`

- L'espace de noms `global` permet d'accéder à des composants d'une **autre** application d'un **même** serveur JEE.
- La syntaxe des noms JNDI :

  `java:global/<app-name>/<module-name>/<bean-name>[!<interface-FQN>]`

- Exemple :

  `java:global/appliCS/moduleCS/CalculSalaireBean!ejb.sessions.CalculSalaireBeanRemote`

JNDI : espace de noms standard App et Module

- L'espace de noms app permet d'accéder à des composants de la même application JEE.
- La syntaxe des noms JNDI :

  java:app/<module-name>/<bean-name>[!<**interface**-FQN>]

  Exemple :

  java:app/moduleCS/CalculSalaireBean!ejb.sessions.CalculSalaireBeanLocal

- L'espace de noms `module` permet d'accéder à des composants issus du même module.
- La syntaxe des noms JNDI :

  java:module/<bean-name>[!<**interface**-FQN>]

Booster le développement d'applications JEE

- Le développement de l'application `appliBanque` nécessite :

Booster le développement d'applications JEE

- Le développement de l'application `appliBanque` nécessite :
  1. de développer et compiler les classes entités

Booster le développement d'applications JEE

- Le développement de l'application `appliBanque` nécessite :
  1. de développer et compiler les classes entités
  2. d'écrire le descripteur `persistence.xml`

Booster le développement d'applications JEE

- Le développement de l'application `appliBanque` nécessite :
  1. de développer et compiler les classes entités
  2. d'écrire le descripteur `persistence.xml`
  3. d'archiver classes entités et descripteur dans un `jar`

Booster le développement d'applications JEE

- Le développement de l'application `appliBanque` nécessite :
  1. de développer et compiler les classes entités
  2. d'écrire le descripteur `persistence.xml`
  3. d'archiver classes entités et descripteur dans un `jar`
  4. de développer et compiler les sessions

Booster le développement d'applications JEE

- Le développement de l'application `appliBanque` nécessite :
  1. de développer et compiler les classes entités
  2. d'écrire le descripteur `persistence.xml`
  3. d'archiver classes entités et descripteur dans un `jar`
  4. de développer et compiler les sessions
  5. d'archiver les sessions dans un `jar`

Booster le développement d'applications JEE

- Le développement de l'application `appliBanque` nécessite :
  1. de développer et compiler les classes entités
  2. d'écrire le descripteur `persistence.xml`
  3. d'archiver classes entités et descripteur dans un `jar`
  4. de développer et compiler les sessions
  5. d'archiver les sessions dans un `jar`
  6. d'archiver les deux modules dans une archive `ear` avec le descripteur `application.xml`

Booster le développement d'applications JEE

- Le développement de l'application `appliBanque` nécessite :
  1. de développer et compiler les classes entités
  2. d'écrire le descripteur `persistence.xml`
  3. d'archiver classes entités et descripteur dans un `jar`
  4. de développer et compiler les sessions
  5. d'archiver les sessions dans un `jar`
  6. d'archiver les deux modules dans une archive `ear` avec le descripteur `application.xml`
  7. de déployer l'application `ear`

Booster le développement d'applications JEE

- Le développement de l'application `appliBanque` nécessite :
  1. de développer et compiler les classes entités
  2. d'écrire le descripteur `persistence.xml`
  3. d'archiver classes entités et descripteur dans un `jar`
  4. de développer et compiler les sessions
  5. d'archiver les sessions dans un `jar`
  6. d'archiver les deux modules dans une archive `ear` avec le descripteur `application.xml`
  7. de déployer l'application `ear`

➜ un net intérêt pour une gestion automatisée !

Booster le développement d'applications JEE

- Le développement de l'application `appliBanque` nécessite :
  1. de développer et compiler les classes entités
  2. d'écrire le descripteur `persistence.xml`
  3. d'archiver classes entités et descripteur dans un `jar`
  4. de développer et compiler les sessions
  5. d'archiver les sessions dans un `jar`
  6. d'archiver les deux modules dans une archive `ear` avec le descripteur `application.xml`
  7. de déployer l'application `ear`

➔ un net intérêt pour une gestion automatisée !

- Une solution : l'outil `Ant`

Booster le développement d'applications JEE

- Le développement de l'application `appliBanque` nécessite :
  1. de développer et compiler les classes entités
  2. d'écrire le descripteur `persistence.xml`
  3. d'archiver classes entités et descripteur dans un `jar`
  4. de développer et compiler les sessions
  5. d'archiver les sessions dans un `jar`
  6. d'archiver les deux modules dans une archive `ear` avec le descripteur `application.xml`
  7. de déployer l'application `ear`

→ un net intérêt pour une gestion automatisée !

- Une solution : l'outil `Ant`
- D'autres solutions existent (ex : `Maven`)

Principe de Ant

http://ant.apache.org/manual

```
<!-- fichier build.xml -->
<project name="exemple"
  default="B" basedir=".">
 <target name="A">
  <tache1 .../>
  <tache2 .../>
 </target>
 <target name="B" depends="A">
   ...
 </target>
 <target name="C"> ...
 </target>
 <target name="D" depends="B,C">
  ...
 </target>
</project>
```

```
ant

ant C

ant -f build.xml

ant -f todo.xml D
```

Ant par l'exemple

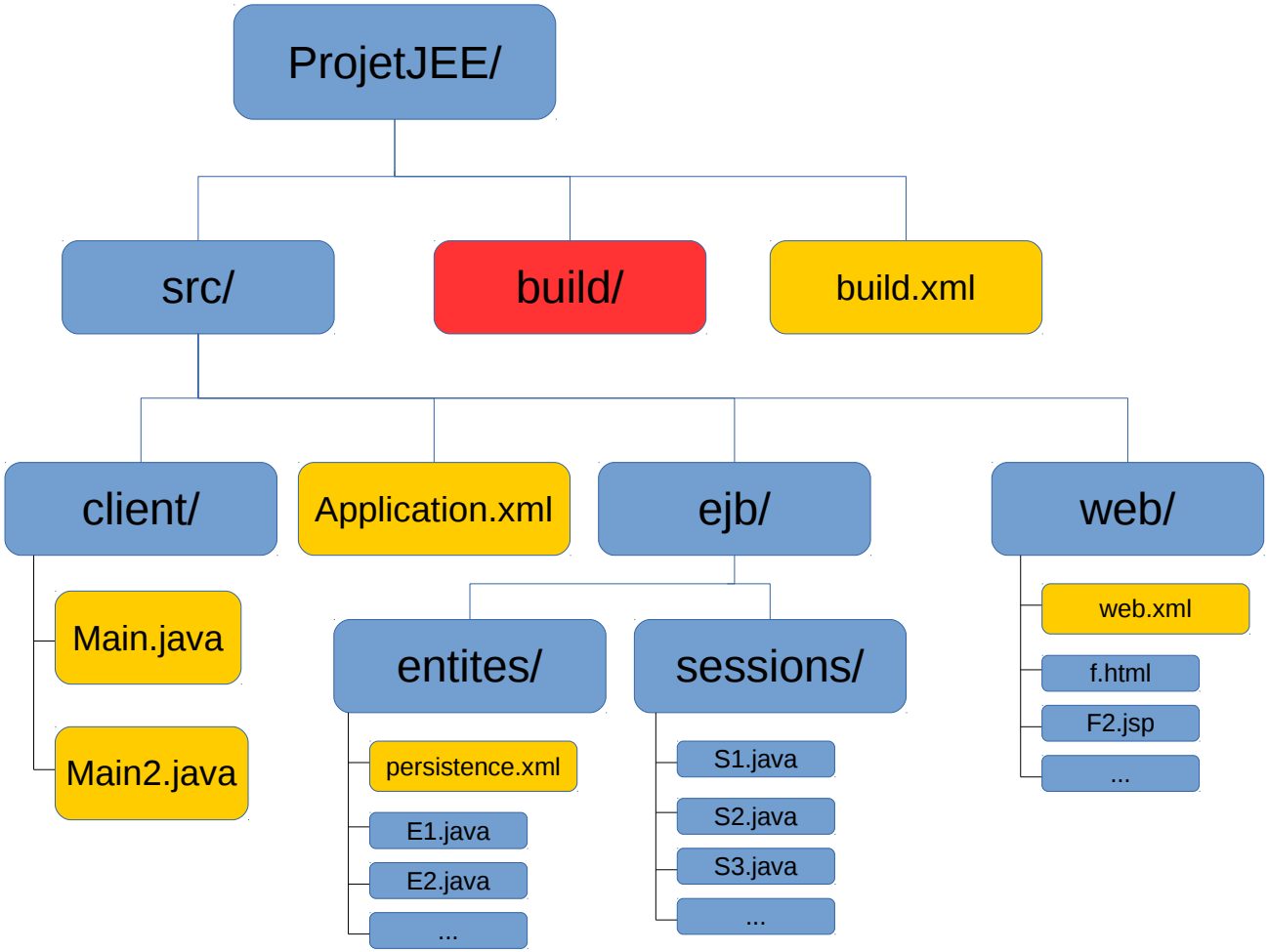
```
<project name="exemple"
         default="compile"
         basedir=".">

 <target name="init">
  <property name="name"
            value="proj" />
  <property name="src"
            value="src" />
  <property name="build"
            value="build" />
  <mkdir dir="${build}" />
 </target>
```

```
 <target name="compile"
         depends="init">
  <javac srcdir="${src}"
         destdir="${build}" />
  <jar jarfile="${name}.jar"
       basedir="${build}" />
 </target>

 <target name="clean">
  <delete dir="${build}" />
  <delete file="${name}.jar" />
 </target>
</project>
```

Le framework Polytech




```
# Des cibles Ant
# pour chaque tâche,
# un exemple :
ant package-sessions

# tout le processus :
ant deploy
```

Message Driven Bean (1/3)

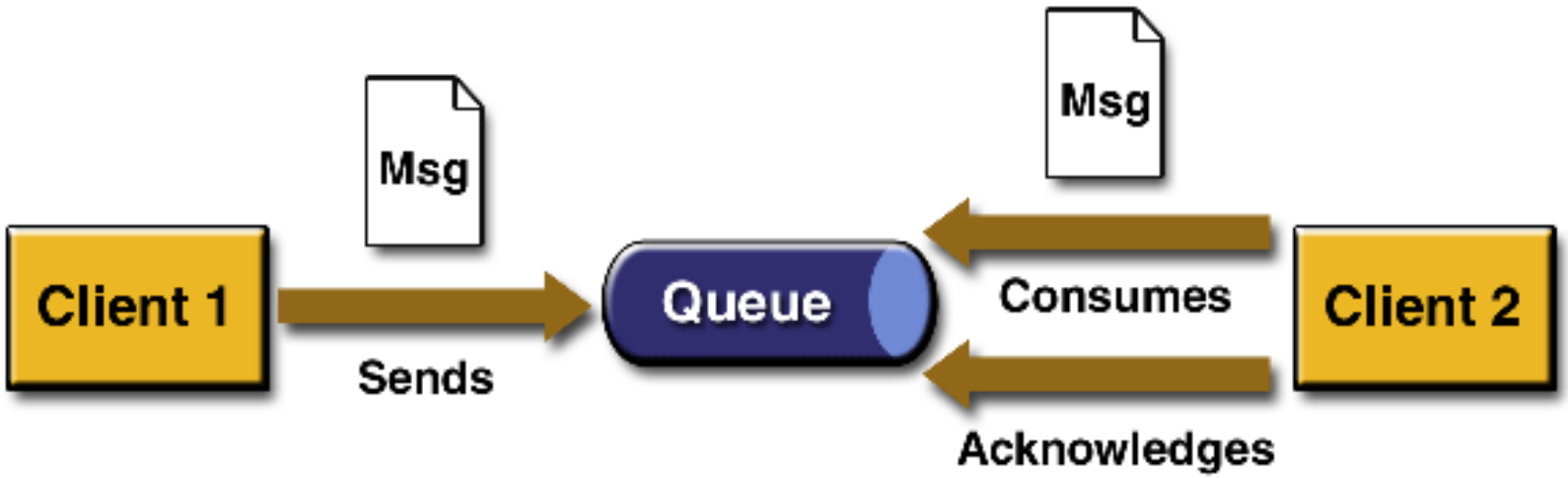
- Envoi de messages asynchrones

Message Driven Bean (1/3)

- Envoi de messages asynchrones
- Les composants ont un couplage faible, pas reliés par leur interface

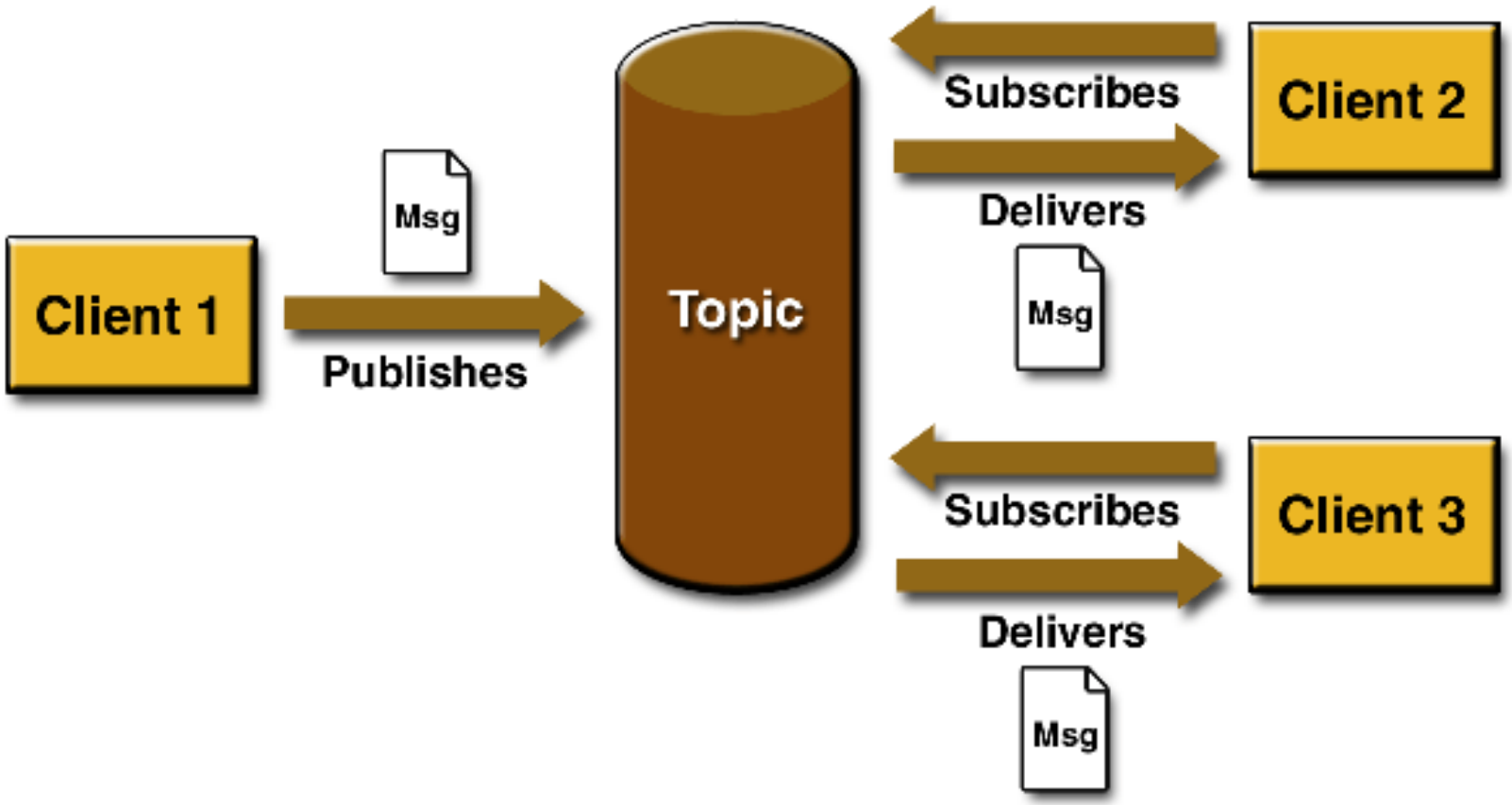
Message Driven Bean (2/3)

- Liaison Point à point :



Message Driven Bean (3/3)

- Diffusion multiple :



Le mot de la fin

© Auteur inconnu

Vous ne pouvez pas comprendre la récursivité sans avoir d'abord compris la récursivité.