Les architectures 3-tiers
Partie II : les EJB sessions

Olivier Caron

Polytech Lille
Avenue Paul Langevin Cité Scientifique Lille 1
59655 Villeneuve d'Ascq cedex

http://ocaron.plil.fr
Olivier.Caron@polytech-lille.fr

© 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

Le framework EJB minimal pour développer un composant Session

1. Il faut écrire du code Java :
   ➔des méthodes définies dans une classe.

Le framework EJB minimal pour développer un composant Session

1. Il faut écrire du code Java :
   ➔des méthodes définies dans une classe.
2. Il faut spécifier le type du composant session :
   ➔la classe sera annotée par `Stateless`, `Singleton` ou `Stateful`.

Le framework EJB minimal pour développer un composant Session

1. Il faut écrire du code Java :
   ➔des méthodes définies dans une classe.
2. Il faut spécifier le type du composant session :
   ➔la classe sera annotée 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 par `Local` et `Remote`.

Le framework EJB minimal pour développer un composant Session

1. Il faut écrire du code Java :
   ➔des méthodes définies dans une classe.
2. Il faut spécifier le type du composant session :
   ➔la classe sera annotée 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 par `Local` et `Remote`.
4. Le serveur JEE doit pouvoir analyser dynamiquement le composant (introspection) :
   ➔La classe dispose d'une fonction constructeur sans paramètre

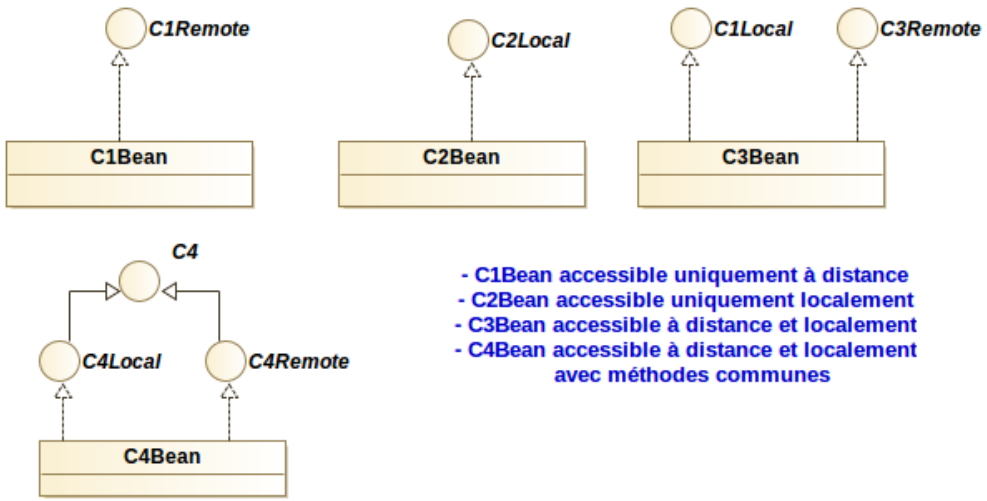
Le framework EJB minimal pour développer un composant Session

1. Il faut écrire du code Java :
   ➔des méthodes définies dans une classe.
2. Il faut spécifier le type du composant session :
   ➔la classe sera annotée 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 par `Local` et `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)

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)

Illustration

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

```
package ejb.sessions ;

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

Premier Bilan

1. Programmation Java simple

Premier Bilan

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

Premier Bilan

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

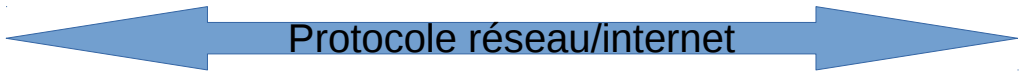
Premier Bilan

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

Premier Bilan

1. Programmation Java simple
2. Une fois compilé (`javac`), archivé (`jar`) et déployé sur un serveur (`cp`), ce composant est :
   - disponible à distance
     ➔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.

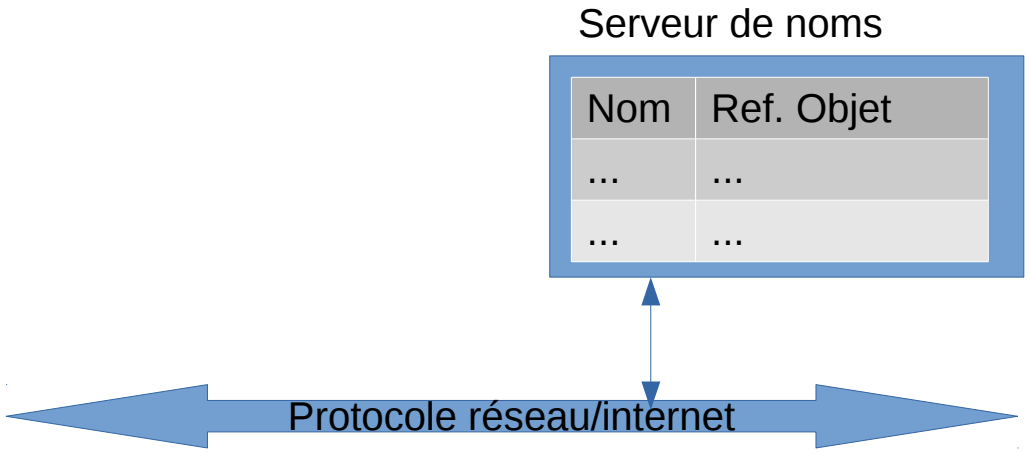
Cycle de vie d'une application par objets répartis



Programmation Java RMI :

`http://ocaron.plil.fr/enseignement/al_gis4/coursRMI.pdf`

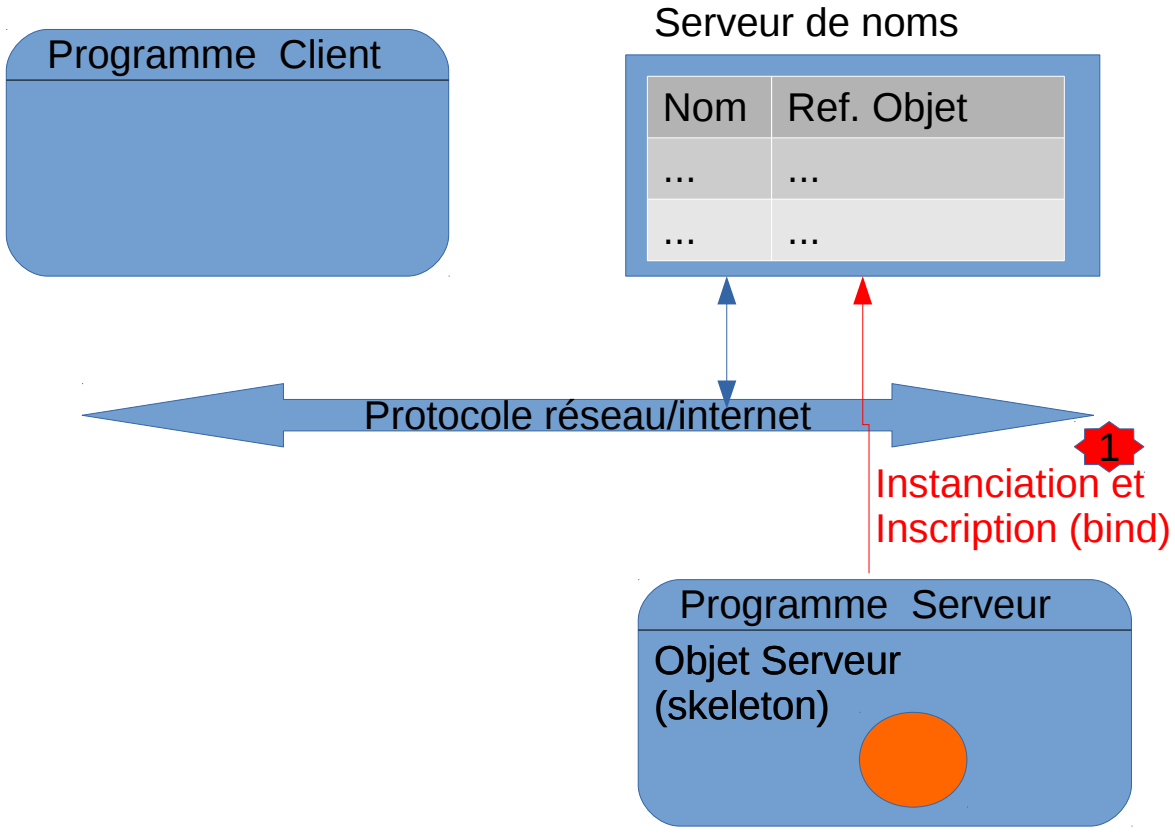
Cycle de vie d'une application par objets répartis



Programmation Java RMI :

`http://ocaron.plil.fr/enseignement/al_gis4/coursRMI.pdf`

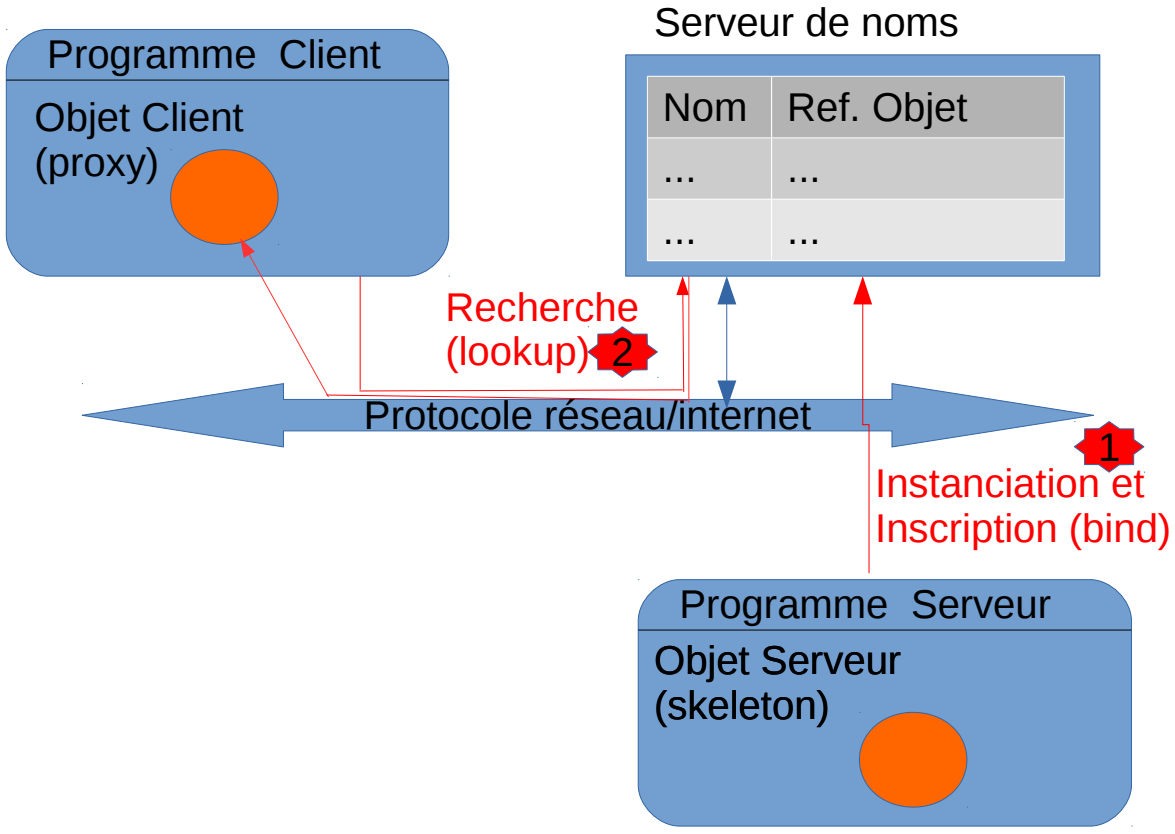
Cycle de vie d'une application par objets répartis



Programmation Java RMI :

`http://ocaron.plil.fr/enseignement/al_gis4/coursRMI.pdf`

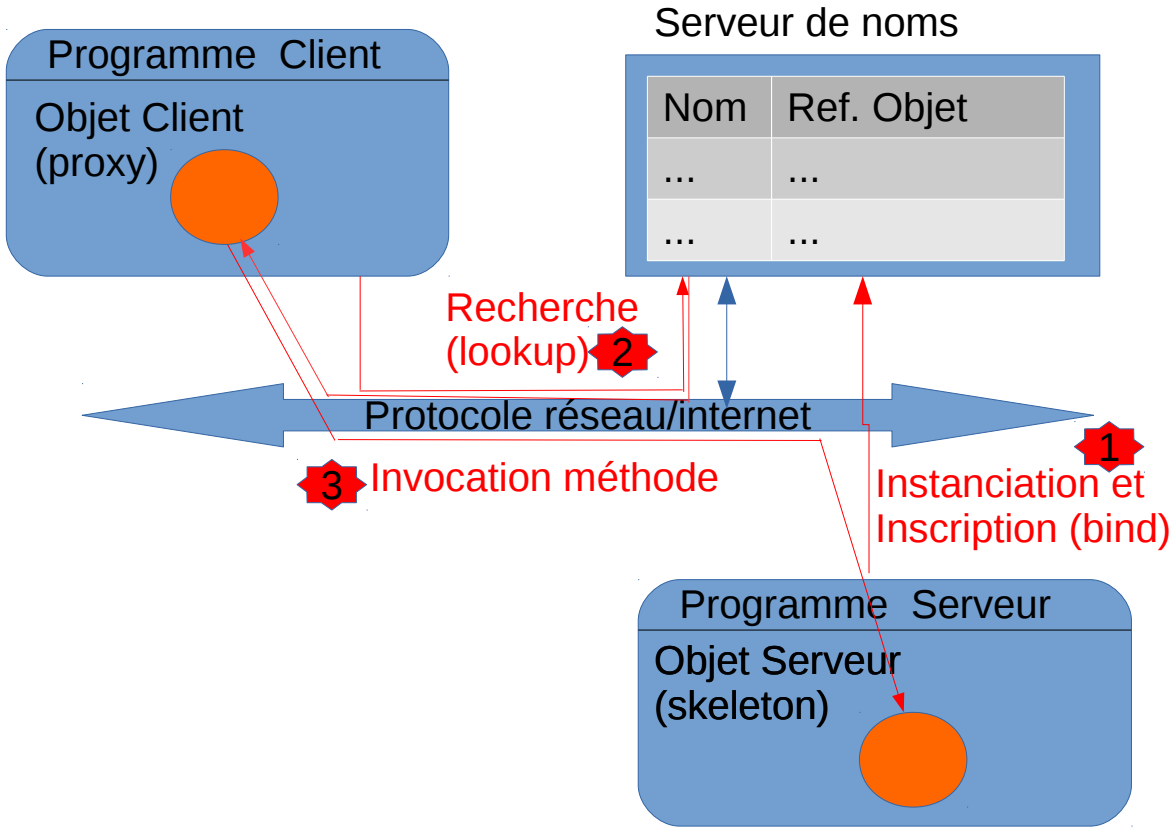
Cycle de vie d'une application par objets répartis



Programmation Java RMI :

`http://ocaron.plil.fr/enseignement/al_gis4/coursRMI.pdf`

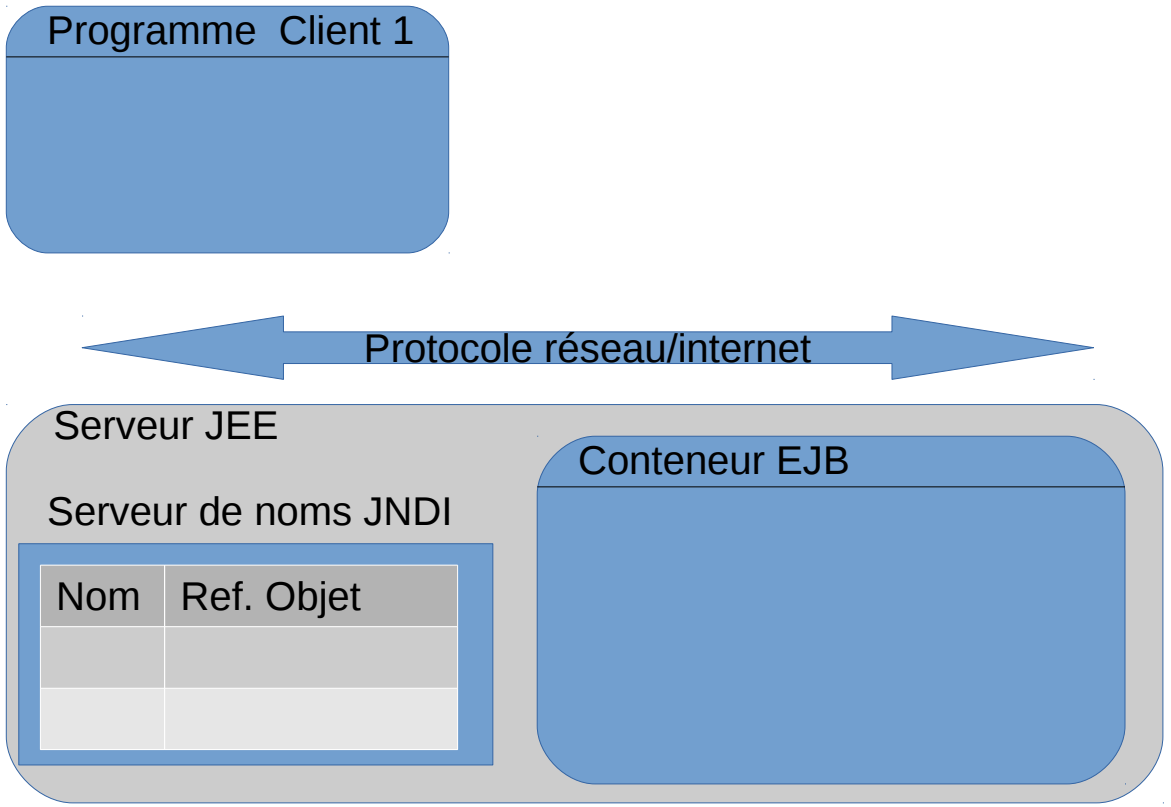
Cycle de vie d'une application par objets répartis



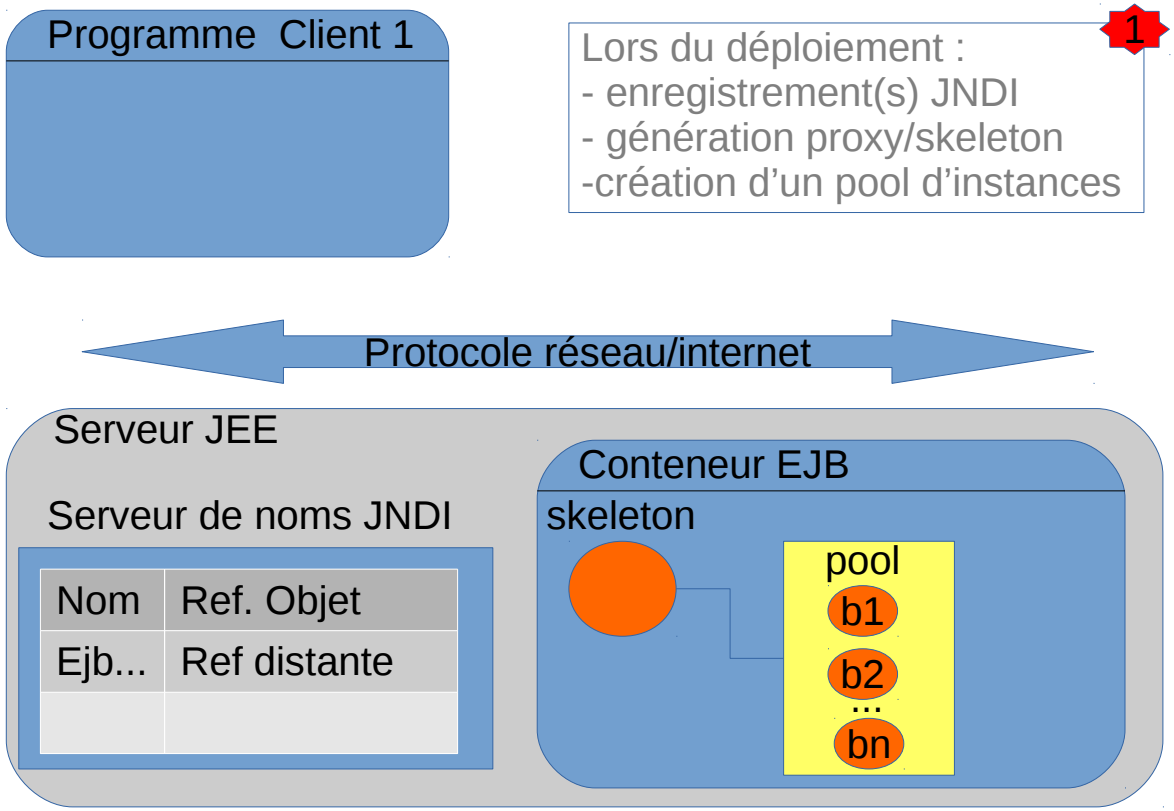
Programmation Java RMI :

`http://ocaron.plil.fr/enseignement/al_gis4/coursRMI.pdf`

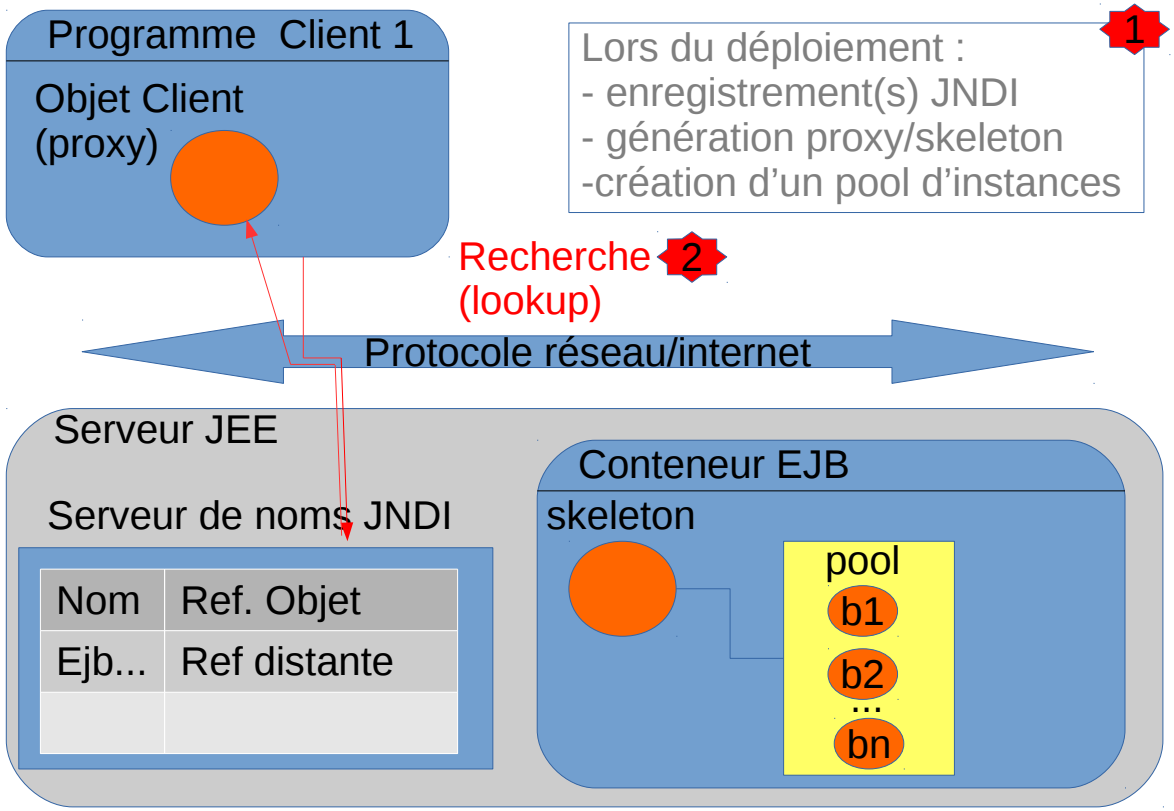
Application aux sessions EJB à distance



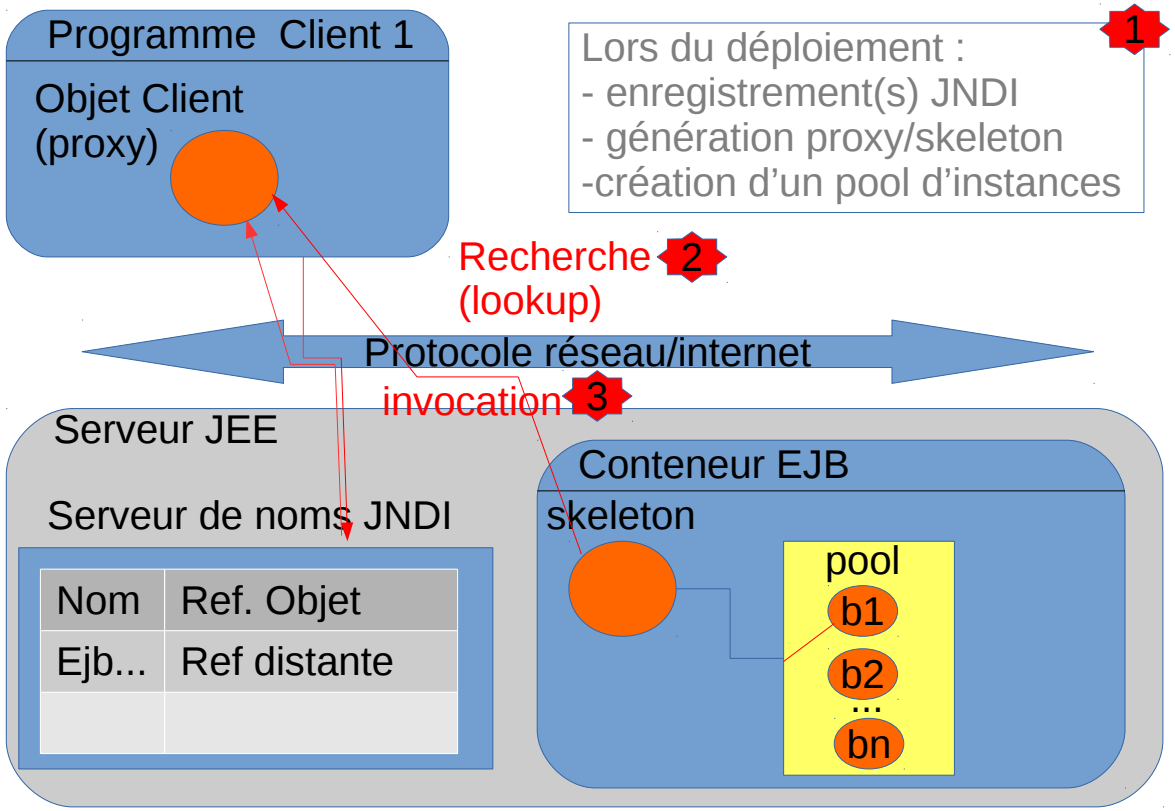
Application aux sessions EJB à distance



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 d'application
  - `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)
- librairie JNDI, fichier `jndi.properties` :

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

Exécution avec plate-forme WildFly

- Le classpath doit contenir les classes réseaux WildFly pour accéder au serveur de noms JNDI WildFly (JBoss)
- librairie JNDI, fichier `jndi.properties` :

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

- Localisation serveur, mode d'accès, sécurité définies dans fichier `jboss-ejb-client.properties` :

```
endpoint.name=client-endpoint
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port = 8080

...
```

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

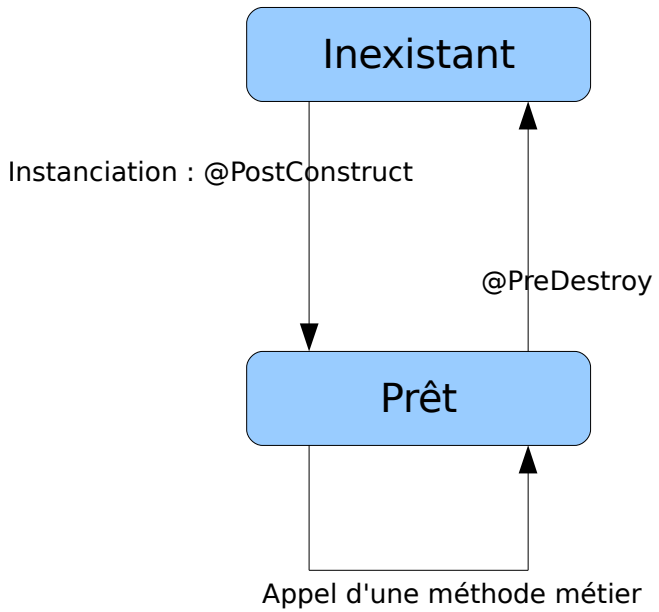
- Un composant `Stateless` n'est associé qu'à un client que pour le temps de l'exécution d'une méthode
- Il peut donc être associé successivement à plusieurs clients

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 `javax.ejb.Singleton`

Le composant `Singleton`

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

Le composant `Singleton`

- Un composant singleton est annoté par `javax.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 `javax.ejb.Stateful`

Les composants session avec état

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

Les composants session avec état

- Annotés par `javax.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.

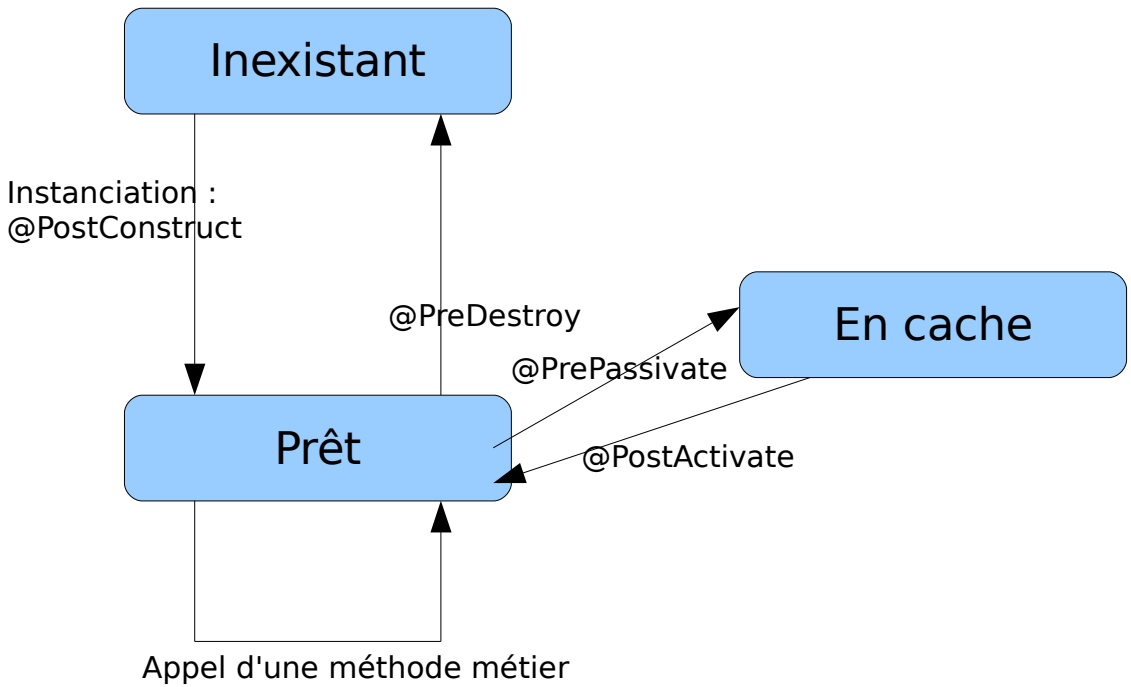
Les composants session avec état

- Annotés par `javax.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 `javax.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 paramètres annotée par `javax.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 ;

@javax.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 :

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

Sécurité et transactions

- Aspect 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, fournit une interface pour s'authentifier.
  - Annotations Java pour définir les droits d'accès. Exemple:

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

- Aspect transaction
  - 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.

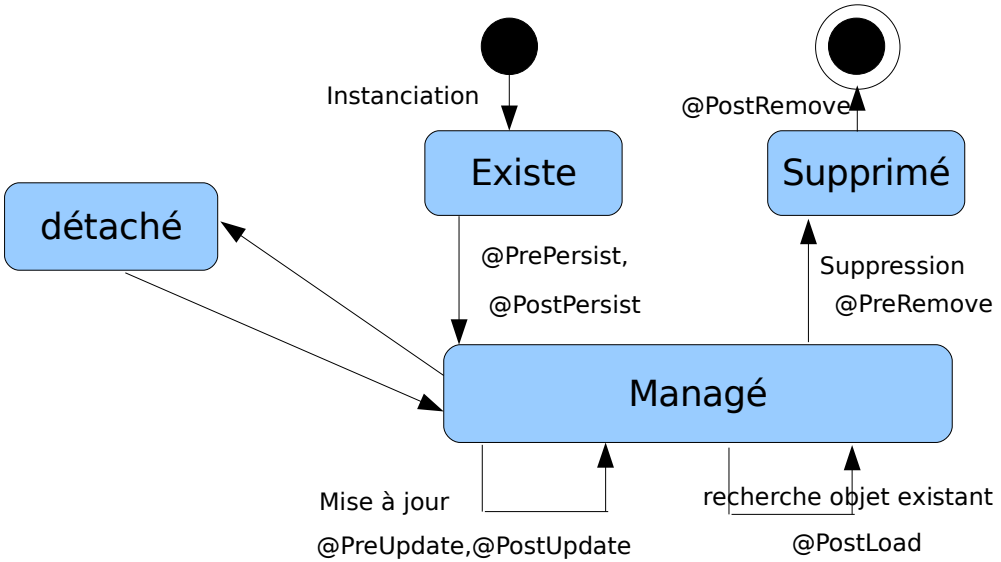


Figure: Cycle de vies des entités EJB

Le gestionnaire d'entités

- Une seule classe : `javax.persistence.EntityManager`
- Ne peut être employée que dans un contexte transactionnel (c'est le cas des sessions)
- Déclaration :

```
1  @Stateless public class MonSessionBean implements MonInterface {
2    @PersistenceContext(unitName="maSource")
3    protected EntityManager em;
```

- Ligne 2 : "maSource" définie dans descripteur xml `persistence.xml` d'un module ejb entité. Le gestionnaire d'entités sera reliée à la source de données (base) 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)

EntityManager : récupération d'un objet BD → objet Java

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

```
Personne pers ;
pers = (Personne) em.find(Personne.class,"dupont@free.fr" );
```

EntityManager : insertion d'un objet dans la base

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

```
if (em.find(Personne.class,email)==null) {
  Personne pers = new Personne();
  pers.setEmail(email) ; pers.setNom(nom) ; pers.setAge(age) ;
  em.persist(pers);
}
```

- Méthode `persist(Object instance)`
- Exception si l'objet existait dans la base (intérêt du `if`)

état de l'instance avant	**après**
état nouveau	état géré
état géré	état géré (pas de changement)
état détaché	exception IllegaleArgumentException
état supprimé	état géré

EntityManager : suppression d'un objet entité

```
em.remove(pers);
```

- 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(pers);
```

- 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	**aprés**
état nouveau	état géré
état géré	action ignorée
état détaché	propagation base, état géré
état supprimé	action ignorée

EntityManager : Requêtes JPQL (1/6)

- Un langage de requêtes défini dans la norme : JPQL

EntityManager : Requêtes JPQL (1/6)

- 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 (1/6)

- 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 (1/6)

- 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 (1/6)

- 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 (1/6)

- 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 : requêtes simples JPQL (2/6)

- Emploi de la méthode : `createQuery(String)`
- La méthode `getResultList()` fournit une collection d'entités.
- Un exemple :

```
public Collection<Personne> getToutesLesPersonnes() {
  return em.createQuery("from Personne p").getResultList() ;
}
```

EntityManager : requêtes JPQL paramétrées (3/6)

- On fournit des valeurs à la requête
- Un exemple :

```
public Collection<Personne> getPersonnesParTrancheAge(int min, int max) {
  return (Collection<Personne>) em.createQuery(
  "from Personne p where p.age >= :mini and p.age < :maxi"
  ).setParameter("mini",new Integer(min))
   .setParameter("maxi", new Integer(max))
   .getResultList() ;
}
```

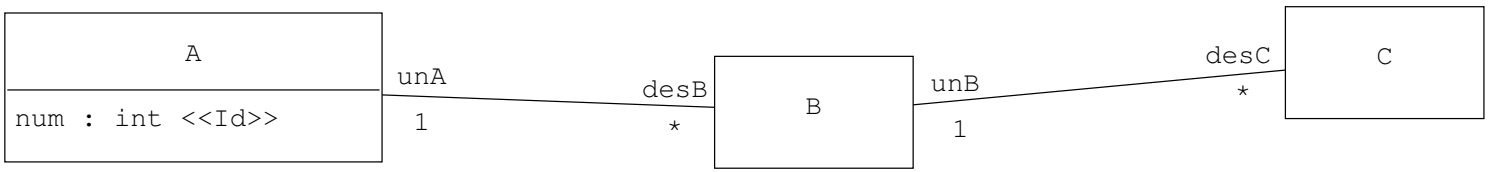
EntityManager : requêtes JPQL avec un objet en retour (4/6)

- Récupération d'un seul objet entité (pas une collection)
- Via la méthode `getSingleResult()`.
- Prendre en compte `NoResultException` (erreur Runtime)
- Un exemple :

```
public Personne getPersonne(String email) throws NoResultException {
  return (Personne)
    em.createQuery("from Personne p where email like :mail")
        .setParameter("mail",email)
        .getSingleResult() ;
}
```

EntityManager : requêtes JPQL et associations (5/6)

- Expression des jointures via les rôles de l'association
- Permet de récupérer un type d'entité relié par association
- Un exemple :




```
public Collection<C> recherche(int numero) {
  return (Collection<C>)
        em.createQuery("select c from A a, B b, C c where " +
                       "a.num=:num and b.unA=a and c.unB=b")
                  .setParameter("num", numero).getResultList() ;
}
```

EntityManager : requêtes JPQL de modification (6/6)

- Pas besoin de charger des objets java pour modifier la base
- Un exemple :

```
public void supprimerPersonnes() {
  int nbrePersonnesSupprimees =
      em.createQuery("delete from Personne").executeUpdate() ;
}
```

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.

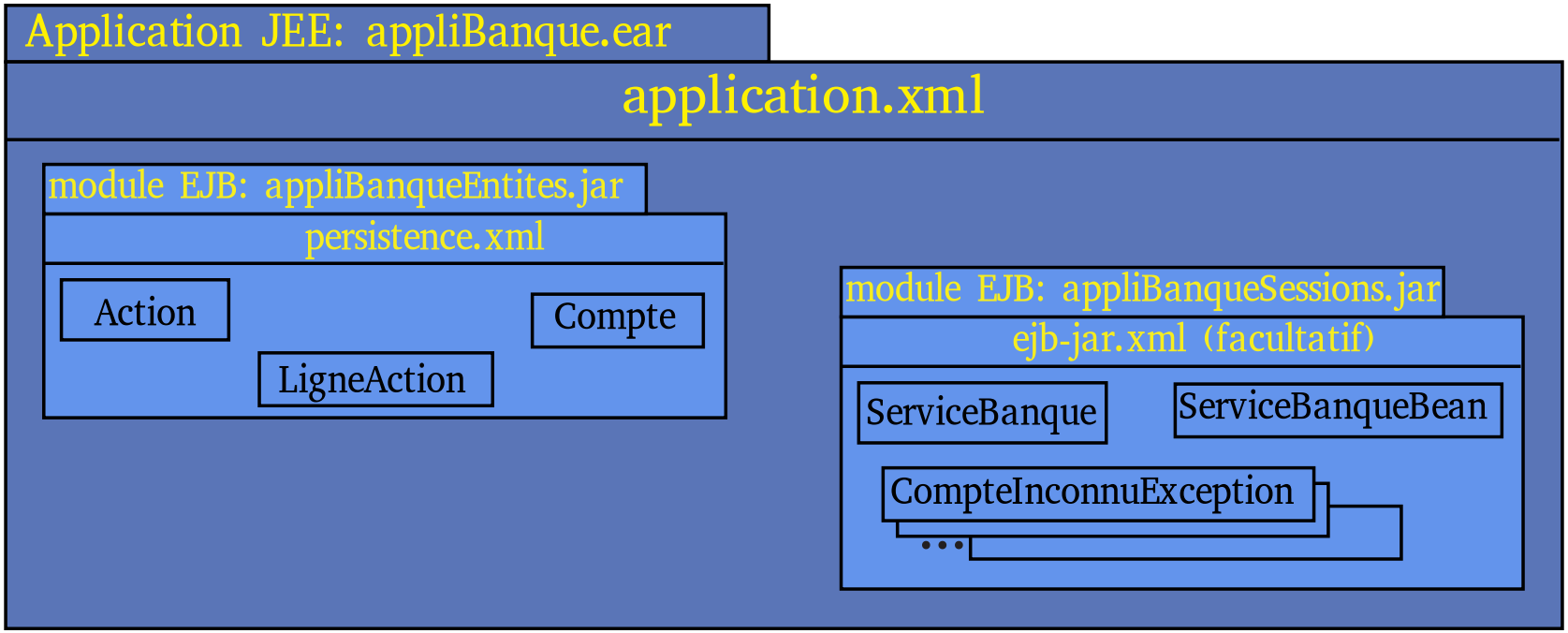
Développement d'applications JEE

- 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://archives.plil.fr/ocaron/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="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="6"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
 http://java.sun.com/xml/ns/javaee/application_6.xsd">
  <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 ;

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

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

```
    public void virementVers(int numCompteDebit, int numCompteCredit,
                             double montant)
      throws CompteInconnuException,ApprovisionnementException ;
    public void acheteActions(int numeroCompte, String nomAction, int nb)
      throws CompteInconnuException, ActionInconnueException,
             ApprovisionnementException ;
    public void vendActions(int numeroCompte, String nomAction, int nb)
      throws CompteInconnuException, ActionInconnueException,
      ApprovisionnementException ;
    public java.util.Set<LigneAction> getActionsAchetees(int numeroCompte)
      throws CompteInconnuException ;
}
```

Le composant session (1/3)

```
package ejb.sessions;

import ...

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

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

  public ServiceBanqueBean() { }
```

Le composant session (2/3)

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

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

Le composant session (3/3)

```
public void addCompte(int numeroCompte, String nomTitulaire,
  double soldeDepart) throws CompteDejaExistantException {
  try {
    this.getCompte(numeroCompte) ;
    throw new CompteDejaExistantException() ;
  } catch(CompteInconnuException e) {
    Compte c=new Compte() ;
    c.setNumero(numeroCompte); 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 = ctx.lookup("ejb:appliBanque/appliBanqueSessions//"+
6          "ServiceBanqueBean!ejb.sessions.ServiceBanque");
7      ServiceBanque service = (ServiceBanque) obj ;
8      System.out.println("les␣actions␣du␣compte␣no:␣1") ;
9      for (LigneAction la : service.getActionsAchetees(1))
10       System.out.println(la.getNombre()+"␣action(s)␣"+
11       la.getAction().getNom()+
12       "␣au␣taux␣de␣"+la.getAction().getTaux()) ;
13       ...
```

Mode de chargement des instances reliées par association

- Dans une application distante (hors serveur JEE), les objets obtenus sont en mode détaché.

Mode de chargement des instances reliées par association

- Dans une application distante (hors serveur JEE), les objets obtenus sont en mode détaché.
- Les méthodes des composants sessions doivent donc fournir le graphe d'objets que le client désire.

Mode de chargement des instances reliées par association

- Dans une application distante (hors serveur JEE), les objets obtenus sont en mode détaché.
- Les méthodes des composants sessions doivent donc fournir le graphe d'objets que le client désire.
- Au niveau du serveur :

Mode de chargement des instances reliées par association

- Dans une application distante (hors serveur JEE), les objets obtenus sont en mode détaché.
- Les méthodes des composants sessions doivent donc fournir le graphe d'objets que le client désire.
- Au niveau du serveur :
  - le mode par défaut d'obtention des instances Java d'entités en mémoire est le mode paresseux (`LAZY`) : ne se charge en mémoire que si on le demande
    Plus efficace mais attention aux `java.lang.NullException` !

Mode de chargement des instances reliées par association

- Dans une application distante (hors serveur JEE), les objets obtenus sont en mode détaché.
- Les méthodes des composants sessions doivent donc fournir le graphe d'objets que le client désire.
- Au niveau du serveur :
  - le mode par défaut d'obtention des instances Java d'entités en mémoire est le mode paresseux (`LAZY`) : ne se charge en mémoire que si on le demande
    Plus efficace mais attention aux `java.lang.NullException` !
- Deux solutions :

Mode de chargement des instances reliées par association

- Dans une application distante (hors serveur JEE), les objets obtenus sont en mode détaché.
- Les méthodes des composants sessions doivent donc fournir le graphe d'objets que le client désire.
- Au niveau du serveur :
  - le mode par défaut d'obtention des instances Java d'entités en mémoire est le mode paresseux (`LAZY`) : ne se charge en mémoire que si on le demande
    Plus efficace mais attention aux `java.lang.NullException` !
- Deux solutions :
  1. Forcer le chargement des objets avant l'envoi des données (méthode `getActionsAchetees`)

Mode de chargement des instances reliées par association

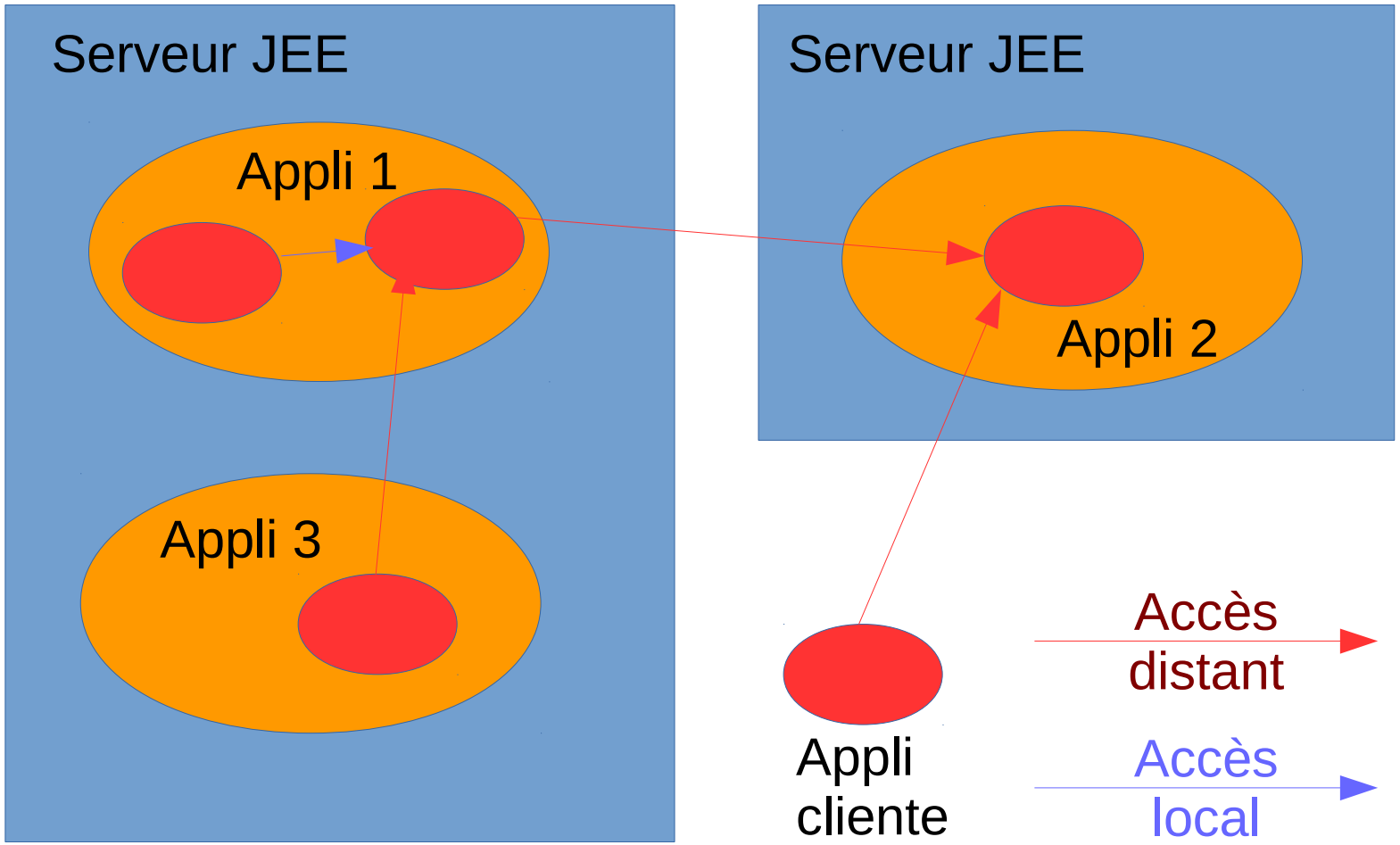
- Dans une application distante (hors serveur JEE), les objets obtenus sont en mode détaché.
- Les méthodes des composants sessions doivent donc fournir le graphe d'objets que le client désire.
- Au niveau du serveur :
  - le mode par défaut d'obtention des instances Java d'entités en mémoire est le mode paresseux (`LAZY`) : ne se charge en mémoire que si on le demande
    Plus efficace mais attention aux `java.lang.NullException` !
- Deux solutions :
  1. Forcer le chargement des objets avant l'envoi des données (méthode `getActionsAchetees`)
  2. Fixer le mode d'acquisition des données en mode non paresseux (`EAGER`)

Version opérationnelle

```
// fichier Compte.java
...
@OneToMany(mappedBy="proprietaire", fetch=FetchType.EAGER)
public Set<LigneAction> getLignesActions() { ... }
```

```
// fichier LigneAction.java
...
@ManyToOne(fetch=FetchType.EAGER)
public Action getAction() { ... }
```

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 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`

Principe de Ant

```
<!-- 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

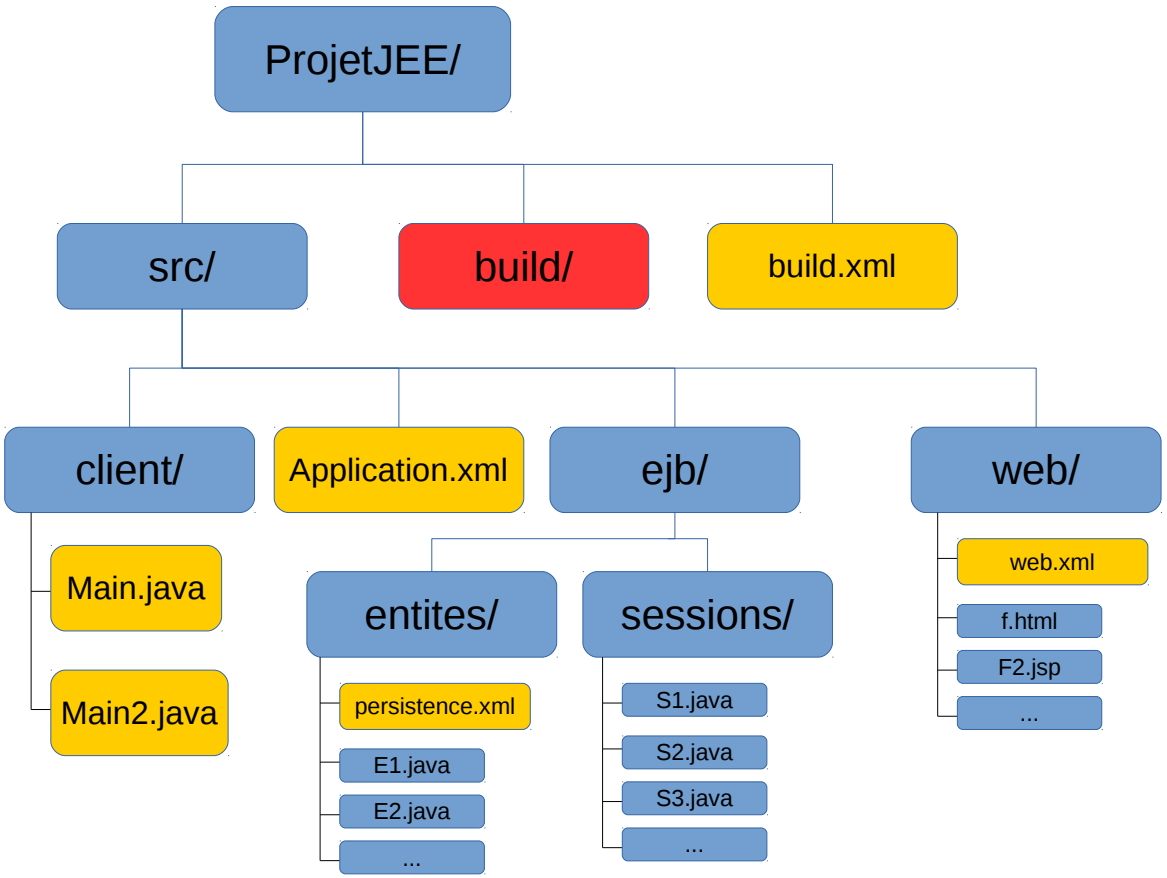
```
<project name="exemple"
 default="compile" basedir=".">

 <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



 # Une sous-tâche précise,
 # un exemple :
ant package-sessions

 # tout le processus :
ant deploy

Message Driven Bean (1/3)

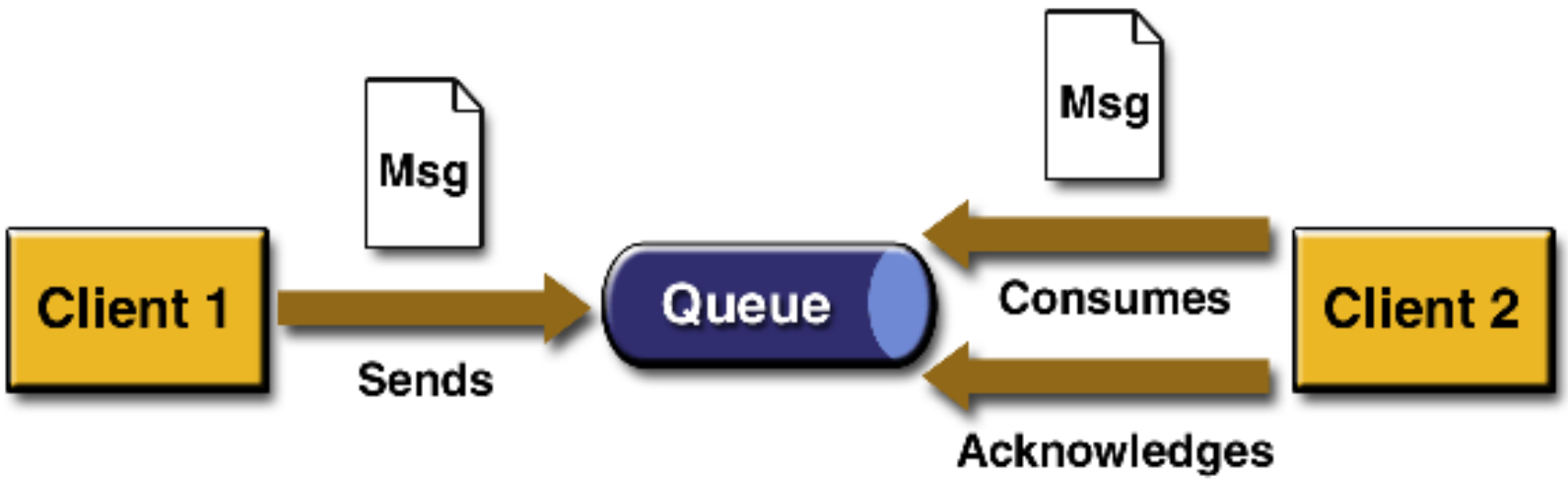
- Envoi de messages asynchrones

Message Driven Bean (1/3)

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

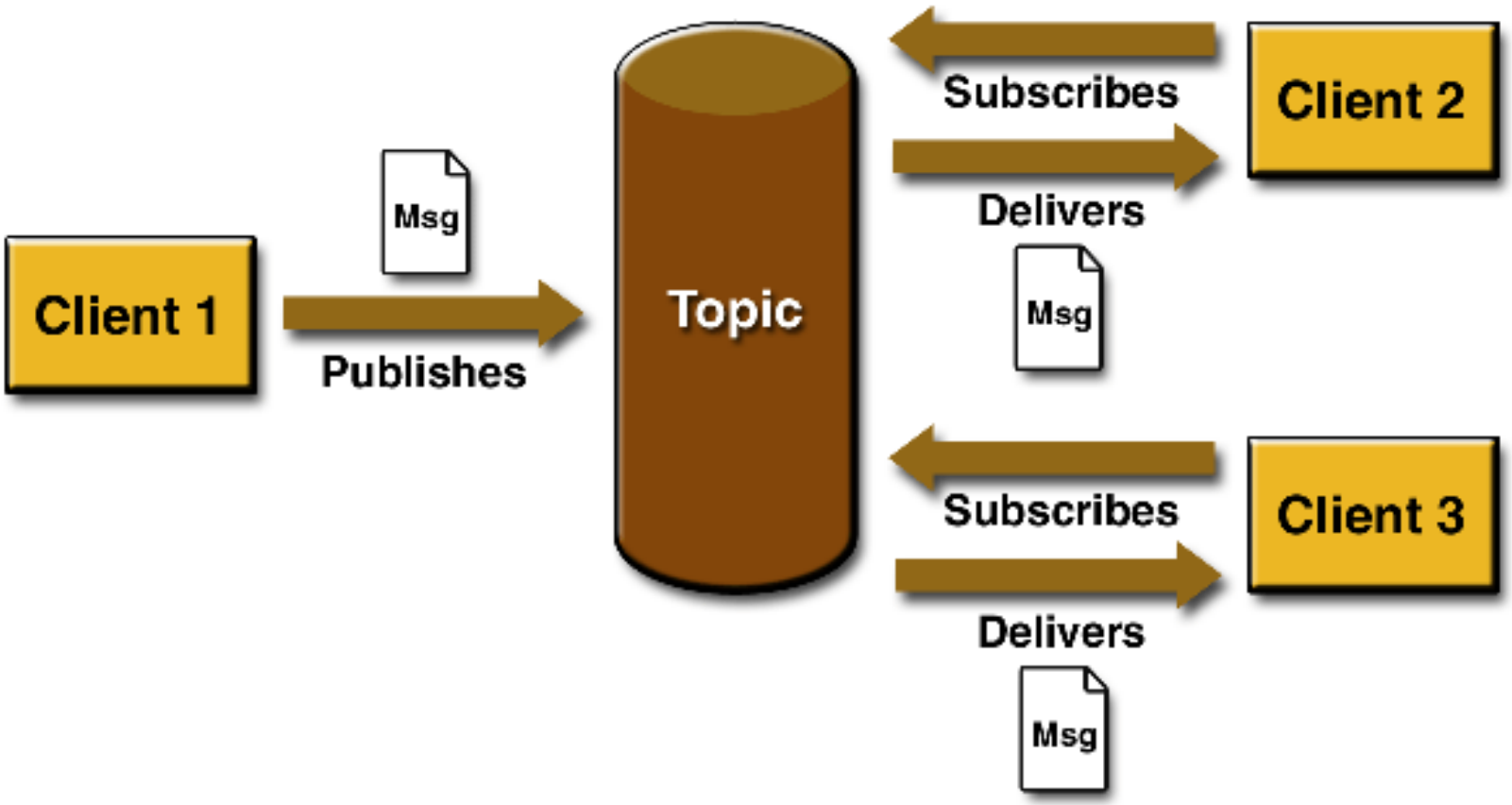
Message Driven Bean (2/3)

- 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.