Enterprise Java Beans

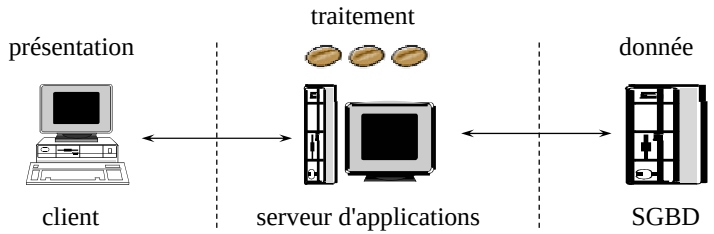
Plan

1. Composant EJB
   1.1 Session Bean
   1.2 Entity Bean
   1.3 Message Driven Bean
2. Services
3. Déploiement
4. Design pattern EJB
5. Conclusion

1. Composant EJB

Enterprise Java Bean (EJB)

Modèle de composants pour le développement d'applications d'entreprises



1. Composant EJB

Enterprise Java Bean (EJB)

A server-side component that encapsulates the business logic of an application

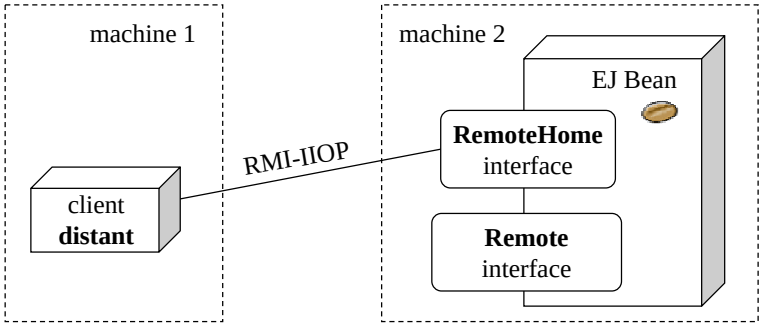
- on se focalise sur la logique applicative
- les services systèmes sont fournis par le conteneur
- la logique de présentation est du ressort du client
- les *beans* sont portables d'un serveur d'application à un autre

Types d'EJB

- Session : *performs a task for a client*
- Entity : *represents a business entity object that exists in persistent storage*
- Message-Driven : *listener processing messages asynchronously*

1. Composant EJB

Enterprise Java Bean



Chaque EJ Bean fournit 2 interfaces d'accès **distant**

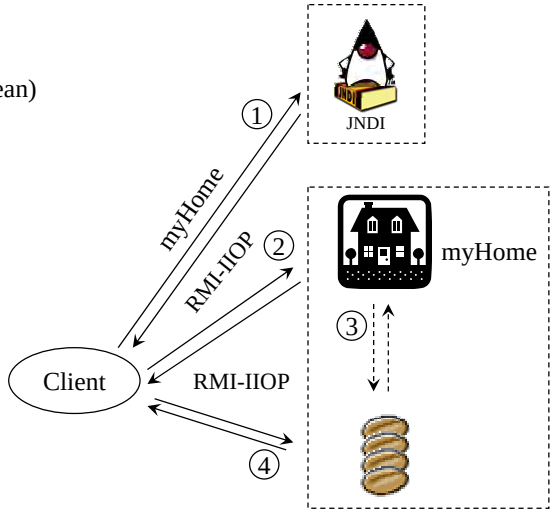
- Remote les services "métiers" (méthodes) fournis par le *bean*
- RemoteHome interface de gestion du composant (création, recherche, destruction d'instances de *bean*)

1. Composant EJB

Enterprise Java Bean

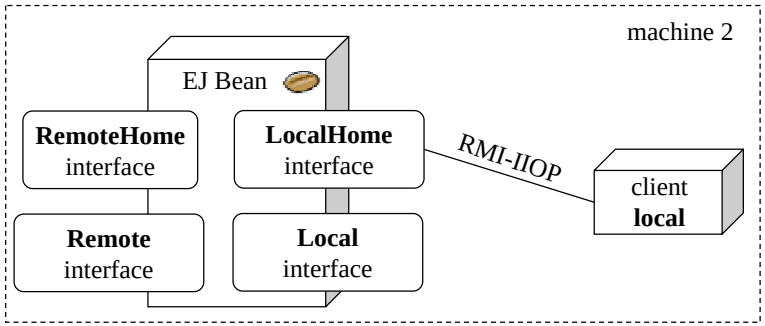
RemoteHome

- singleton (1 seule instance par bean)
- **unique point d'accès au bean** (jamais accès direct aux beans)
- recherche, création de beans
- enregistrée dans le service de nommage (JNDI)
- pas d'équivalent en RMI ou en CORBA (voir HomeFactory CCM)



1. Composant EJB

Enterprise Java Bean



+ éventuellement 2 interfaces d'accès **local** (à partir EJB 2.0)
(i.e. pour clients hébergés dans le même conteneur → meilleure performance)

- Local les services "métiers" (méthodes) fournis par le *bean*
- LocalHome interface de gestion du composant

1.1 Session Bean

Session Bean
- Définition
- Développement
  - Interface Remote
  - Interface RemoteHome
  - Interface SessionBean
  - Classe
- Cycle de vie
- Côté client

1.1 Session Bean

Session bean (SB)

- *bean* dont la durée de vie est liée à celle de son client
- meurt lorsque le client n'en a plus besoin
- *bean* à durée de vie plutôt courte

1. Stateless session bean

- sans état
- ne conserve pas d'information entre 2 appels successifs
- 2 instances qqconques d'un tel *bean* sont équivalentes

2. Stateful session bean

- avec un état (en mémoire)

1.1 Session Bean

Quand utiliser un session bean ?

- on veut représenter un **processus** ∈ à la logique métier
- pas de besoin spécifique de partage de données entre clients
- pas de besoin de persistance

Stateful

- l'état du *bean* représente l'état de l'interaction entre le client et le *bean*
- le *bean* doit conserver de l'information entre 2 invocations du client
- le *bean* sert de **médiateur** entre le clients et d'autres *beans* de l'application (en général des *entity beans*)
- le *bean* gère un *workflow*

Stateless

- pour des tâches génériques
- pour consulter en **lecture seule** des données persistantes

1.1 Session Bean

Développement

2 interfaces + 1 classe

Interface Remote
- services "métiers" fournis par le *bean*
- `extends javax.ejb.EJBObject`

Interface RemoteHome
- gestion du *bean*
- `extends javax.ejb.EJBHome`

Classe Java
- code des interfaces Remote et RemoteHome
- `implements javax.ejb.SessionBean`

+ éventuellement code interfaces Local et LocalHome

1.1 Session Bean

Interface Remote

- services "métiers" fournis par le *bean*
- `extends javax.ejb.EJBObject`
- profils méthodes libres
- chaque méthode appelable à distance → `throws java.rmi.RemoteException`

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface CalculatriceRemoteInterface extends EJBObject {

  public double add(double v1,double v2) throws RemoteException;
  public double sub(double v1,double v2) throws RemoteException;
  public double mul(double v1,double v2) throws RemoteException;
  public double div(double v1,double v2) throws RemoteException;
}
```

1.1 Session Bean

Interface RemoteHome

- gestion du *bean*
- `extends javax.ejb.EJBHome`
- méthodes possibles
  - `create` : création d'instances (**retourne interface Remote**)
- plusieurs méthodes `create` peuvent être définies avec ≠ profils
- autres méthodes (choisies par le développeur)

```
import javax.ejb.EJBHome;
import java.rmi.RemoteException;
import javax.ejb.CreateException;

public interface CalculatriceRemoteHome extends EJBHome {

  public CalculatriceRemoteInterface create()
    throws RemoteException, CreateException;
}
```


1.1 Session Bean

Interface `javax.ejb.SessionBean`

- doit être implantée par tout SB
- interface de notification

Method Summary	
void	ejbActivate() The activate method is called when the instance is activated from its "passive" state.
void	ejbPassivate() The passivate method is called before the instance enters the "passive" state.
void	ejbRemove() A container invokes this method before it ends the life of the session object.
void	setSessionContext(SessionContext ctx) Set the associated session context.

1.1 Session Bean

Class Java du SB

- `implements javax.ejb.SessionBean`

- code des méthodes de l'interface Remote
- code des méthodes de l'interface `javax.ejb.SessionBean`

- code des méthodes de l'interface RemoteHome
  - une méthode `ejbCreate` pour chaque `create`
  - même profil que `create`
  - retourne `void`

1.1 Session Bean

Class Java du SB

```
public class CalculatriceBean implements javax.ejb.SessionBean {

  /* Méthodes de l'interface Remote. */

  public double add(double v1,double v2) {return v1+v2;}
  public double sub(double v1,double v2) {return v1-v2;}
  public double mul(double v1,double v2) {return v1*v2;}
  public double div(double v1,double v2) {return v1/v2;}

  /* Méthodes de l'interface RemoteHome. */

  public void ejbCreate() throws CreateException {}

  /* Méthodes de l'interface SessionBean. */

  public void ejbActivate() {}
  public void ejbPassivate() {}
  public void ejbRemove() {}
  public void setSessionContext(SessionContext sc) {}
}
```

1.1 Session Bean

Côté client

Référence Home enregistrée dans serveurs noms JNDI lors du déploiement (nom logique)

1. Récupération référence serveurs de noms JNDI
2. Recherche de la référence de la Home (à partir nom logique)
3. Construction d'une souche cliente pour l'accès (distant) à la Home
4. Appel des méthodes de la Home (`create`, …)
5. Appel des méthodes des beans

1.1 Session Bean

Côté client

```
import javax.naming.Context;
import javax.naming.InitialContext;

public class Client {
  public static void main(String args[]) throws Exception {

    /* 1. JNDI: récupération ref. serveur de noms. */
    Context ic = new InitialContext();

    /* 2. Recherche de la Home enregistrée avec MaCalculatrice. */
    Object obj = ic.lookup("MaCalculatrice");

    /* 3. Construction souche cliente pour l'accès (distant) à la Home */
    CalculatriceRemoteHome home = (CalculatriceRemoteHome)
      PortableRemoteObject.narrow(obj, CalculatriceRemoteHome.class);

    /* 4. Appel méthodes de la Home */
    CalculatriceRemoteInterface bean = home.create();

    /* 5. Appel méthode du Bean */
    double res = bean.add(3,6);
} }
```


1.2 Entity Bean

Entity Bean
- Définition
- Développement CMP
  - Interface Remote
  - Interface RemoteHome
  - Interface EntityBean
  - Classe
- Clé primaire
- Finders
- Côté client
- Cycle de vie
- Développement BMP

1.2 Entity Bean

Entity bean (EB)

- n'est pas lié à la durée de vie des sessions avec les clients
- peut être **partagé** par +sieurs clients
- ses **données** sont gérées de manière **persistante**
- est identifié de manière unique par une **clé primaire**
- peut être **relié** à d'autres *entity beans* (≡ relations dans un SGBDR)

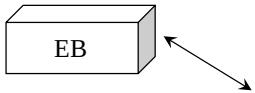
1. Bean managed persistence (BMP)
   - la gestion de la persistence est à la charge du *bean*
     - optimiser la gestion des données
     - utiliser d'autres supports que JDBC (fichiers, JDO, etc.)

2. Container managed persistence (CMP)
   - la persistence des données est gérée automatiquement par le conteneur
   - configuration via un descripteur de déploiement

1.2 Entity Bean

Entity bean (EB)

≡ à un tuple dans une table relationnelle



Nom	Solde
John	100.00
Anne	156.00
Marcel	55.25

3 catégories de variables d'instance dans un *entity bean*

- persistante ≡ attribut de la table
- relationnelle ≡ clé étrangère qui réf. un attribut d'une autre table/bean
- temporaire ≡ donnée "de travail" non sauvegardée

Cardinalité des relations entre entity beans

- 1-1
- 1-n (ex. : une commande contient n lignes)
- n-1 (ex. : plusieurs ligne de commandes peuvent concerner le même produit)
- n-n (ex. : un cours comporte +sieurs étudiants qui suivent +sieurs cours)

Relations peuvent être mono- ou bi-directionnelle

1.2 Entity Bean CMP

Développement CMP

2 interfaces + 1 classe

Interface Remote
- services "métiers" fournis par le *bean*
- `extends javax.ejb.EJBObject`

Interface RemoteHome
- gestion du *bean*
- `extends javax.ejb.EJBHome`

Classe Java
- code des interfaces Remote et RemoteHome
- `implements javax.ejb.EntityBean`

+ éventuellement code interfaces Local et LocalHome
+ éventuellement classe clé primaire

1.2 Entity Bean CMP

Interface Remote

- services "métiers" fournis par le *bean*
- `extends javax.ejb.EJBObject`
- profil des méthodes libre
- chaque méthode appelable à distance → `throws java.rmi.RemoteException`

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface CompteRemoteInterface extends EJBObject {

  public void deposer(double montant) throws RemoteException;
  public boolean retirer(double montant) throws RemoteException;
  public double solde() throws RemoteException;
}
```


1.2 Entity Bean CMP

Interface RemoteHome

- gestion du *bean*
- `extends javax.ejb.EJBHome`

- méthodes possibles (+sieurs méthodes possibles avec ≠ profils)
  - `create :` création d'instances (**retourne interface Remote**)
- autres méthodes (choisies par le développeur)

```
import javax.ejb.EJBHome;
import javax.ejb.RemoteException;
import javax.ejb.CreateException;

public interface CompteRemoteHome extends EJBHome {

  public CompteRemoteInterface create(String nom)
     throws RemoteException, CreateException;
  public CompteRemoteInterface create(String nom, double montant)
     throws RemoteException, CreateException;
}
```

1.2 Entity Bean CMP

Interface `javax.ejb.EntityBean`

Method Summary	
void	ejbActivate() A container invokes this method when the instance is taken out of the pool of available instances to become associated with a specific EJB object.
void	ejbLoad() A container invokes this method to instruct the instance to synchronize its state by loading it state from the underlying database.
void	ejbPassivate() A container invokes this method on an instance before the instance becomes disassociated with a specific EJB object.
void	ejbRemove() A container invokes this method before it removes the EJB object.
void	ejbStore() A container invokes this method to instruct the instance to synchronize its state by storing it to the underlying database.
void	setEntityContext(EntityContext ctx)
void	unsetEntityContext()

1.2 Entity Bean CMP

Classe Java de l'EB

- classe **abstraite**
- 1 couple de méthodes getter/setter **abstraites** par donnée persistante
- `implements javax.ejb.EntityBean`
- code des méthodes de l'interface Remote
- code des méthodes de l'interface `javax.ejb.EntityBean`
- code des méthodes de l'interface RemoteHome
  - une méthode `ejbCreate` pour chaque `create`
  - une méthode `ejbPostCreate` pour chaque `create`

1.2 Entity Bean CMP

Classe Java de l'EB

(1/5)

```
public abstract class CompteBean implements javax.ejb.EntityBean {
  /* Setters/getters pour l'acces aux données du bean. */
  public abstract String getNom();
  public abstract double getSolde();
  public abstract void setNom(String nom);
  public abstract void setSolde(double solde);
```

- le développeur **n'écrit pas de classe concrète**
- la plate-forme **génère une sous-classe concrète**
  - les setters/getters contiennent du code JDBC pour l'accès aux données persistantes

1.2 Entity Bean CMP

Classe Java de l'EB

(2/5)

- code méthodes interface Remote sans `java.rmi.RemoteException`

```
  /* Méthodes de l'interface Remote. */
  public void deposer(double montant) {
    setSolde( getSolde() + montant );
  }
  public boolean retirer(double montant) {
     double solde = getSolde();
     if (solde>=montant) {
        setSolde( solde-montant );
        return true; }
     else return false;
  }
  public double solde() { return getSolde(); }
```

1.2 Entity Bean CMP

Classe Java de l'EB

(3/5)

- une méthode `ejbCreate` par méthode `create` de l'interface Home
- même profil que `create`
- type de retour : la clé primaire de l'*entity bean*
- `return null`

```
/* Méthodes de l'interface RemoteHome. */
public String ejbCreate(String nom) throws CreateException {
  setNom(nom); setSolde(0.0);
  return null;
}
public String ejbCreate(String nom, double montant) throws ... {
  setNom(nom); setSolde(montant);
  return null;
}
```


1.2 Entity Bean CMP

Classe Java de l'EB

(4/5)

- une méthode `ejbPostCreate` par méthode `create` de l'interface Home
- pas d'exception `CreateException`

- appelée après `ejbCreate`
- le *bean* existe et ses éventuelles **relations** existent

```
/* Méthodes de l'interface Home. */
public void ejbPostCreate(String nom) {}
public void ejbPostCreate(String nom, double montant) {}
```


1.2 Entity Bean CMP

Classe Java de l'EB

(5/5)

```
/* Méthodes de l'interface EntityBean. */
public void ejbActivate() {}
public void ejbPassivate() {}
public void ejbRemove() {}
public void ejbLoad() {}
public void ejbStore() {}
public void setEntityContext(EntityContext ec) {}
public void unsetEntityContext() {}
}
```


1.2 Entity Bean CMP

Classe clé primaire

Chaque instance de *bean* peut être identifié par une clé primaire

- soit une variable d'instance du *bean*
- soit une instance d'une classe fournie par le développeur du *bean*
  - implantant l'interface `java.io.Serializable`
  - définissant les méthodes `hashCode` et `equals`

```
public class ComptePK implements java.io.Serializable {
  public String nom; public String prenom;
  public ComptePK(String n,String p) { nom=n; prenom=p; }
  public ComptePK() {}
  public int hashCode() {
    return nom.hashCode() + prenom.hashCode();
  }
  public boolean equals(Object other) {
    return /** nom==other.nom && prenom==other.prenom */;
} }
```

1.2 Entity Bean CMP

Finder – clé primaire

Méthode de recherche d'instances de *beans*

- nom `findByPrimaryKey` imposé
- définie dans l'interface Home

- paramètre : la classe de la clé primaire
- retour : l'interface Remote (ou `null`)

- son code est généré automatiquement

```
public interface CompteRemoteHome extends EJBHome {
  public CompteRemoteInterface findByPrimaryKey(String nom)
    throws java.rmi.RemoteException, javax.ejb.FinderException;
}
```


1.2 Entity Bean CMP

Autres *Finders*

Méthodes de recherche d'instances de *beans*

- nom commençant par `find` (ex. : `findAll`, `findBySolde`, ...)
- définies dans l'interface Home
- paramètres libres
- retour
  - soit un seul *bean* (interface Remote)
  - soit un ensemble de *beans* (`java.util.Collection`)
  - soit `null`
- implantées par une requête EJB QL

```
public interface CompteRemoteHome extends EJBHome {
  public Collection findAll() throws RemoteException,FinderException;
  public CompteRemoteInterface findBySolde(double solde) throws ...;
  public CompteRemoteInterface findByPrimaryKey(String nom) throws ...;
}
```


1.2 Entity Bean CMP

Autres *Finders*

Définition de requêtes EJB QL associées

- chaque EB est associé à un nom de table choisi par le développeur
- mot clé `OBJECT` dans la requête EJB QL : on sélectionne un objet
- les paramètres de la méthode `find...` sont accessibles par `?`*`numéro`*

Exemples

- `findAll` `SELECT OBJECT(c) FROM Compte c`
- `findBySolde` `SELECT OBJECT(c) FROM Compte c WHERE c.solde>=?1`

1.2 Entity Bean

Côté client

même principe que pour les clients de Session Bean

référence Home enregistrée dans serveurs noms JNDI lors du déploiement (nom logique)

1. Récupération référence serveurs de noms JNDI
2. Recherche de la référence de la Home (à partir nom logique)
3. Construction d'une souche cliente pour l'accès (distant) à la Home
4. Appel des méthodes de la Home (`create`, `findByPrimaryKey`, …)
5. Appel des méthodes des beans

1.2 Entity Bean

Classe Java du client

(1/2)

```
import javax.naming.Context;
import javax.naming.InitialContext;

public class Client {
  public static void main(String args[]) throws Exception {
    /* JNDI: récupération ref. serveur de noms. */
    Context ic = new InitialContext();
    /* Recherche du bean enregistré sous le nom MonCompte. */
    Object obj = ic.lookup("MonCompte");
    /* obj: souche d'accès à l'itf Home du compte -> cast */
    CompteRemoteHome home = (CompteRemoteHome)
      PortableRemoteObject.narrow(obj,CompteRemoteHome.class);
```


1.2 Entity Bean

Classe Java du client

(2/2)

```
/* Recherche d'un bean existant. */
CompteRemoteInterface cr = home.findByPrimaryKey("Robert");
cr.deposer(130);
System.out.println( cr.solde() );

/* Création d'un nouveau bean. */
CompteRemoteInterface cl = home.create("Lionel",167);
System.out.println( cl.solde() );
```

Avant

nom	solde
Robert	150
Bob	200

Après

nom	solde
Robert	280
Bob	200
Lionel	167

1.2 Entity Bean BMP

Entity Bean BMP

- gestion de la persistance par le *bean*
- possibilité de stocker les données : JDBC, JDO, Hibernate, fichiers, …

- pour : optimisation possibles, meilleures performances
- contre : plus de code à écrire

1.2 Entity Bean BMP

Différences CMP/BMP

- pas de différence pour les interfaces Home et Remote
- pas de différence pour la classe clé primaire

	CMP	**BMP**
classe	`abstract`	normale
accès au SGBD	généré automatiquement	**à coder**
état persistent	pas de variable (`abstract`)	variables normales
getter/setter	obligatoires	facultatifs
`findByPrimaryKey`	généré automatiquement	**à coder**
autres finders	requête EJB QL	**à coder**
valeur de retour de `ejbCreate`	`null`	la valeur de la clé primaire

1.2 Entity Bean BMP

Classe Java de l'EB

(1/6)

```
public class CompteBean implements javax.ejb.EntityBean {
  private String nom;
  private double solde;
```


1.2 Entity Bean BMP

Classe Java de l'EB

(2/6)

- code méthodes interface Remote sans `java.rmi.RemoteException`

```
  /* Méthodes de l'interface Remote. */
  public void deposer(double montant) {
     solde += montant;
  }
  public boolean retirer(double montant) {
     if (solde>=montant) {
        solde -= montant;
        return true; }
     else return false;
  }
  public double solde() { return solde; }
```


1.2 Entity Bean BMP

Classe Java de l'EB

(3/6)

- **codage de la persistence des données**
- type de retour : la clé primaire de l'*entity bean*
- `return` la clé primaire

```
  /* Méthodes de l'interface RemoteHome. */
  public String ejbCreate(String nom) throws CreateException {
    return ejbCreate(nom,0.0);
  }
  public String ejbCreate(String nom, double montant) throws ... {
    this.nom = nom;
    this.solde = solde;
    /* JDBC SQL: INSERT INTO tablename VALUES ('nom',solde) */
    return nom;
  }
```


1.2 Entity Bean BMP

Classe Java de l'EB

(4/6)

- une méthode `ejbPostCreate` par méthode `create` de l'interface Home
- pas d'exception `CreateException`

- appelée après `ejbCreate`
- le *bean* existe et ses éventuelles **relations** existent

```
  /* Méthodes de l'interface Home. */
  public void ejbPostCreate(String nom) {}
  public void ejbPostCreate(String nom, double montant) {}
```

1.2 Entity Bean BMP

Classe Java de l'EB

(5/6)

```
/* Méthodes de l'interface EntityBean. */
public void ejbRemove() {
  /* JDBC SQL: DELETE FROM tablename WHERE nom='nom' */
}
public void ejbLoad() {}
  /* JDBC SQL: SELECT * FROM tablename WHERE nom='nom' */
  /* this.solde = resultset.getDouble("solde") */
}
public void ejbStore() {
  /* JDBC SQL: UPDATE tablename SET solde=solde WHERE nom='nom' */
}
public void ejbActivate() { nom = (String)ec.getPrimaryKey(); }
public void ejbPassivate() { nom = null; }
public void setEntityContext(EntityContext ec) {this.ec=ec;}
public void unsetEntityContext() {this.ec=null;}
```


1.2 Entity Bean BMP

Classe Java de l'EB

(6/6)

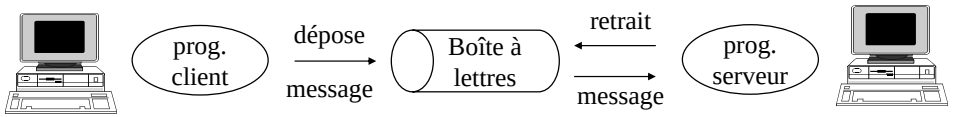
- préfixe `ejb`, exception `FinderException`
- `ejbFindByPrimaryKey`: teste l'existence de la clé passée en paramètre

```
/* Méthodes Finder. */
public String ejbFindByPrimaryKey(String key) throws FinderException {
  /* JDBC SQL: SELECT nom FROM tablename WHERE nom='key' */
  if key existe dans la table { return key; }
  else return null;
}
public Collection ejbFindAll() throws FinderException {
  /* JDBC SQL: SELECT nom FROM tablename */
  return une collection avec tous les noms
}
```


1.3 Message-driven Bean

Message-driven bean (MDB)

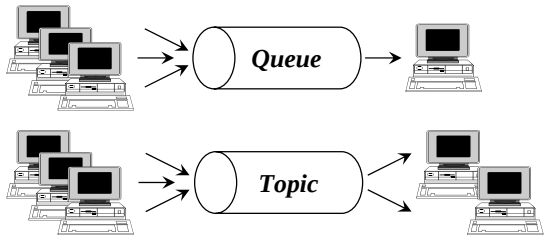
Interaction **par messagerie** (MOM : *Message-Oriented Middleware*)



≈ CORBA COSEvent

2 modes

- n vers 1 (*queue*)
- n verrs m (*topic*)



1.3 Message-driven Bean

Caractéristiques

- consomme des messages asynchrones
- pas d'état (≡ stateless session bean)
- toutes les instances d'une même classe de MDB sont équivalentes
- peut traiter les messages de clients ≠
- pas d'itfs home, ni remote, ni local
- 1 seule méthode métier (`onMessage`)
  - paramètres imposés
  - pas de valeur de retour
  - pas d'exception

Quand utiliser un MDB

• éviter appels bloquants
• découpler clients (producteurs) et serveurs (consommateurs)
• besoin de fiabilité : protection crash serveurs

1.3 Message-driven Bean

Concepts

MDB basé sur les specs JMS (`java.sun.com/jms`)

- ConnectionFactory créer des connexions vers queue/topic
- Connection vers queue/topic
- Session
  intervale de temps pour l'envoi de messages dans queue/topic
  1 ou +sieurs sessions par connexion
  peut être rendue transactionnelle

⇨ nombreuses similitudes avec JDBC

2. Services

Serveur d'application

Services fournis

- cycle de vie
- transaction
- nommage
- sécurité

≠ par rapport à CORBA : services intégrés dès le départ à la plate-forme

2.1 Cycle de vie

Service de cycle de vie des *beans*

- Passivation d'instances

  sauvegarde temporaire du bean
  lorsque le conteneur a besoin de mémoire

- *Pooling* d'instances

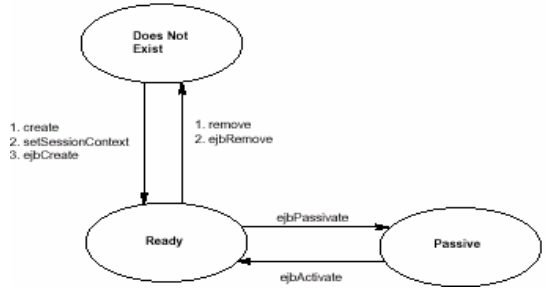
  - pour des raisons de performances, le conteneur peut instancier moins de *beans* qu'il n'y a de client
    ⇨ +sieurs clients partagent un même *bean*
  - n'est possible que si le *bean* ne gère pas de variables d'instance

2.1 Cycle de vie

Cycle de vie d'un *session bean*

Stateful

`create` et `remove`
invoquées par le client



Le conteneur peut décider
de "passiver" le *bean*
⇨ sauvegarde de son état
⇨ en général politique *least recently used*

Stateless

même diagramme d'états sauf **pas de** passivation

2.1 Cycle de vie

Cycle de vie d'un *entity bean*

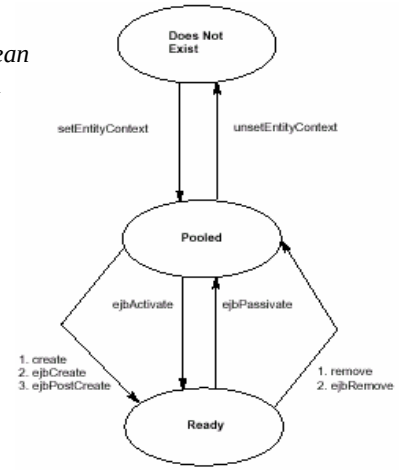
Le conteneur EJB gère un *pool* d'instances de *bean*
Après instanciation, le *bean* est mis dans le pool

⇨ il n'est pas associé à aucune donnée
mais est prêt à servir

`create` et `remove` invoquées par le client

Toutes les autres méthodes

- implantées par le *bean*
- invoquées par le conteneur



2.2 Transactions

Service de transactions

Assure des propriétés **ACID** pour des transactions plates

• des travaux (cf. JOnAS) visent à fournir des modèles de transactions + riches

Exemple classique : un transfert bancaire (débit, crédit)

• atomicité	soit les 2 opérations s'effectuent complètement, soit aucune
• cohérence	le solde d'un compte ne doit jamais être négatif
• isolation	des transferts // doivent fournir le même résultat qu'en séq.
• durabilité	les soldes doivent être sauvegardés sur support stable

Support complètement intégré au serveur EJB
Véritable + / aux *middlewares* style CORBA

2.2 Transactions

Granularité des transactions

Comment démarquer (délimiter) les transactions ?

Attribut transactionnel avec 6 valeurs

- SUPPORTS
- NOT_SUPPORTED
- REQUIRED
- REQUIRES_NEW
- MANDATORY
- BEAN_MANAGED : transactions à la charge du bean

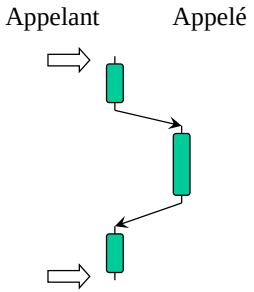
2 cas pour le *bean* appelant

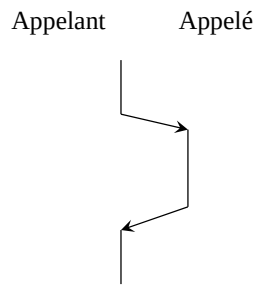
• soit il s'exécute dans une transaction
• soit il s'exécute en dehors de tout contexte transactionnel

2.2 Transactions

Granularité des transactions

SUPPORTS

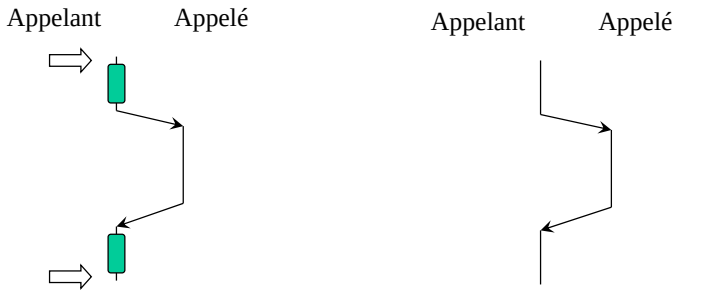




2.2 Transactions

Granularité des transactions

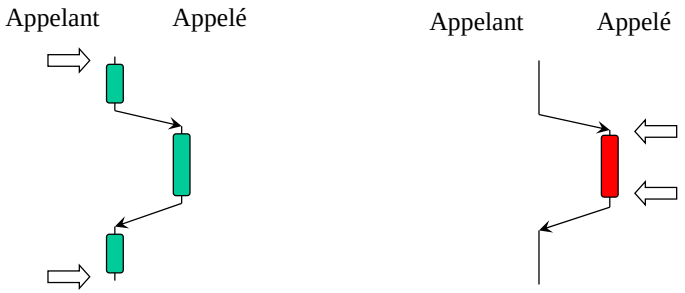
NOT_SUPPORTED



2.2 Transactions

Granularité des transactions

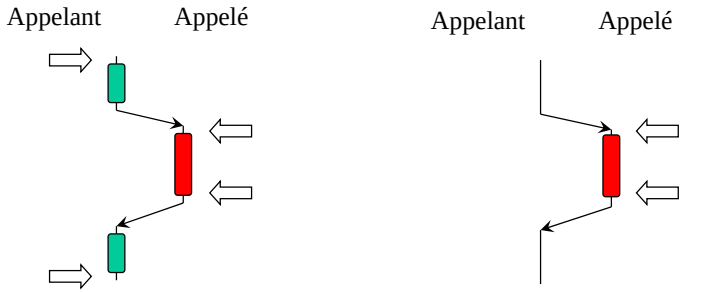
REQUIRED



2.2 Transactions

Granularité des transactions

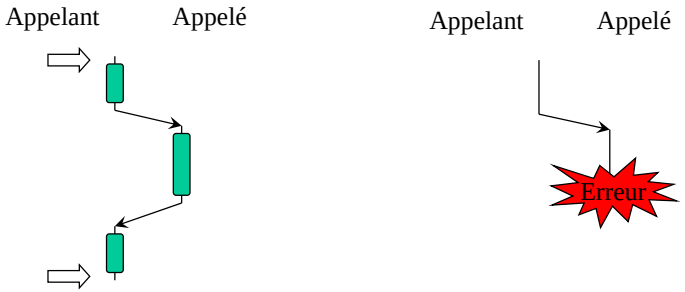
REQUIRES_NEW



2.2 Transactions

Granularité des transactions

MANDATORY



2.2 Transactions

Isolation des transactions

(≡ à quel moment les résultats des transactions sont visibles)

`TRANSACTION_SERIALIZABLE` (cas "normal")

⇨ les transactions apparaissent comme si
elles avaient été exécutées **en séquence**

`TRANSACTION_REPEATABLE_READ`

- t_1 lit un enregistrement
- t_2 modifie cet enregistrement
- si t_1 lit à nouveau l'enregistrement, il lit la **même valeur** que précédemment
- enregistrements "fantômes" possibles

2.2 Transactions

Isolation des transactions

`TRANSACTION_READ_COMMITED`

dirty reads interdits

- t_1 modifie un enregistrement
- t_2 ne peut pas lire cet enregistrement
tant que t_1 **n'a pas été engagée** (ou annulée)

`TRANSACTION_READ_UNCOMMITED`

dirty reads autorisés

- t_1 modifie un enregistrement
- t_2 peut lire cet enregistrement

`TRANSACTION_NONE`

pas de support pour les transactions

2.2 Transactions

API JTA (Java Transaction API)

Impératif
(programmatique)

Permet de manipuler explicitement des transactions

Ex. : transaction sur 2 *entity beans*

```
UserTransaction utx = (UserTransaction)
  PortableRemoteObject.narrow(
    initialContext.lookup("javax.transaction.UserTransaction"),
    UserTransaction.class );
  utx.begin();
  cl.deposer(50);
  boolean ok = cr.retirer(50);
  if (ok)  utx.commit();
  else     utx.rollback();
```

Ne convient pas dans tous les cas (ex : clients légers Web)

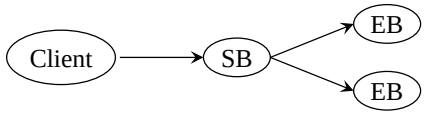
2.2 Transactions

API JTA (Java Transaction API)

Déclaratif

2ème sol.

⇨ *session bean* avec méthode `transfert`
⇨ attribut transactionnel `REQUIRES_NEW` sur `transfert`



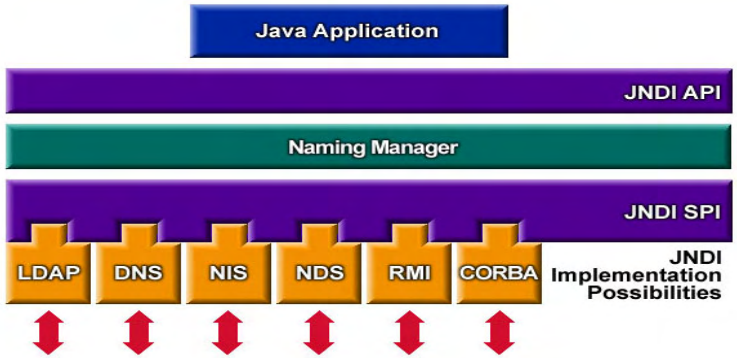
Meilleure
solution

```
public boolean transfert( String de, String vers, double montant ) {
  Compte from = home.findByPrimaryKey(de);
  Compte to = home.findByPrimaryKey(vers);
  to.crediter(montant);
  ok = from.debiter(montant);
  if (!ok)  sessionContext.setRollbackOnly();
  return ok; }
```

2.3 Nommage

JNDI (Java Naming and Directory Interface)

- 1 unique API pour accéder à ≠ serveurs de nom
- unification, simplification



2.3 Nommage

JNDI

URL JNDI *`préfixe : schéma`*

- préfixe identifiant unique par type de serveurs de noms
- schéma spécifique à chaque type de serveur de noms

Exemples

- `iiop://foo.com/x` RMI-IIOP
- `ldap://localhost:389` LDAP
- `java:comp/env`

Type `java:`

- serveur de noms en mémoire
- accessible seulement dans le contexte mémoire des *beans*
- stocke des réf. vers des ressources fournies par J2EE aux *beans* (ex accès à l'API JTA, réf. des interfaces locales, …)

2.3 Nommage

JNDI

Récupération d'un point d'entrée vers JNDI

```
Context ic = new InitialContext();
```

le serveur de noms RMI-IIOP local

```
Properties props = new Properties();
props.put( InitialContext.PROVIDER_URL, "iiop://host.com:1050" );
Context ic = new InitialContext(props);
```

un serveur de noms RMI-IIOP sur la machine host.com et le port 1050

```
Properties props = new Properties();
props.put( InitialContext.INITIAL_CONTEXT_FACTORY,
           "com.sun.jndi.cosnaming.CNCtxFactory" );
props.put( InitialContext.PROVIDER_URL, "..." );
Context ic = new InitialContext(props);
```


3. Déploiement

Packaging des applications

1 application EJB = 1 archive .ear

- 1 descripteur XML de l'application
- 1 archive .war par *web bean*
- 1 archive .jar par *bean*

1 archive .war

- 1 descripteur XML du *web bean*
- JSP ou servlet.class

1 archive .jar

- 1 descripteur XML du *bean*
- les .class du *bean*

3. Déploiement

Descripteur XML du bean

Fichier XML (ejb-jar.xml) conforme à une DTD définie dans les specs des EJB

⇨ informations pour configurer et déployer le bean
⇨ 46 balises
⇨ plates-formes fournissent des **assistants** pour sa création

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ejb-jar>
  <description>Beans de gestion de compte bancaire</description>
  <enterprise-beans>          <!-- les beans de l'application -->
    ...
  </enterprise-beans>
  <assembly-descriptor>       <!-- la configuration des services -->
    ...
  </assembly-descriptor>
</ejb-jar>
```


3. Déploiement

Exemple

```
<entity>
  <description>Bean correspondant à un compte</description>
  <ejb-name>Compte</ejb-name>
  <home>mypkg.CompteRemoteHome</home>    <!-- les classes et interf. -->
  <remote>mypkg.CompteRemoteInterface</remote>
  <ejb-class>mypkg.CompteBean</ejb-class>
  <prim-key-class>java.lang.String</prim-key-class>
  <!-- Définition des attributs persistants -->
  <persistence-type>Container</persistence-type>
  <cmp-field> <field-name>nom</field-name> </cmp-field>
  <cmp-field> <field-name>solde</field-name> </cmp-field>
  <reentrant>False</reentrant>
</entity>
```


3. Déploiement

Exemple (suite)

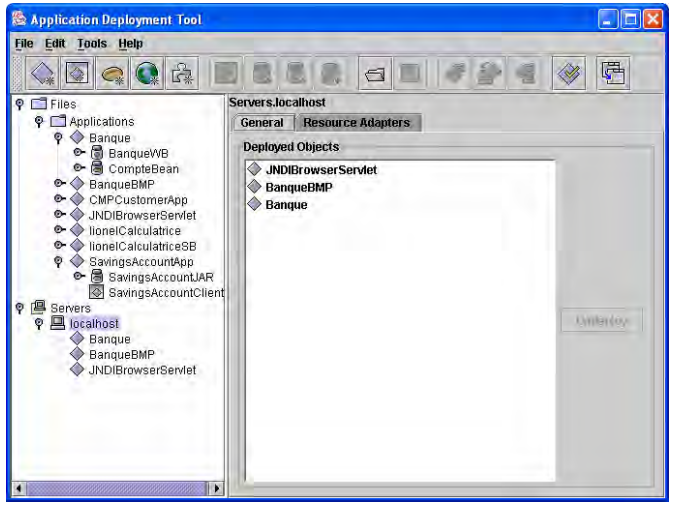
```
<assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name>Compte</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```


3. Déploiement

Outillage

Exemple : deploytool (Sun RI)

- génère les descripteurs XML
- déploie/retire les applications



4. Design Patterns EJB

Gabarit (ou patrons) de conception

Problèmes de codage récurrents

- parcourir un arbre de données dont les noeuds sont typés
- maj une fenêtre en fonction de modifications sur des données (et vice-versa)
- …

⇨ design pattern (DP) : solutions reconnues d'organisation du code

But

- améliorer la clarté, la compréhension du code
- mettre en avant des éléments d'architecture logicielle

DP courramment utilisés avec les EJB

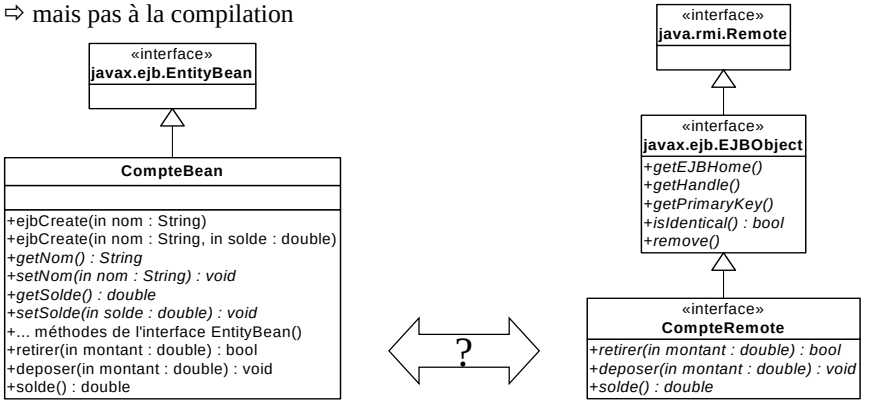
- business interface, home factory, session façade, data transfert object
- …

4. Design Patterns EJB

DP Business interface

Pb : perte de typage entre interface Remote et la classe du Bean

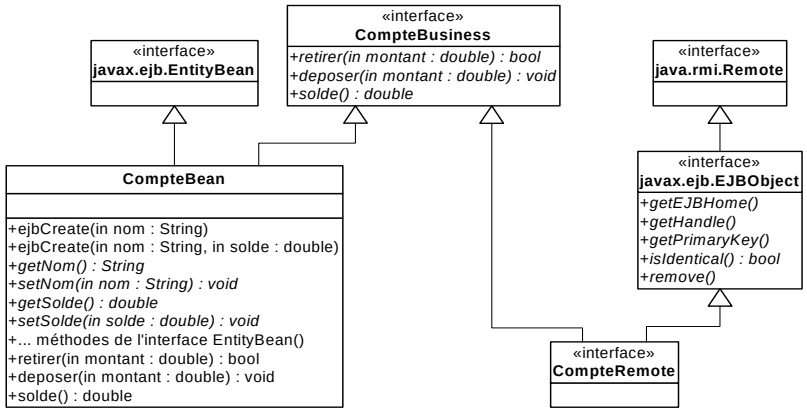
⇨ vérification de conformité à l'exécution
⇨ mais pas à la compilation



4. Design Patterns EJB

DP Business interface

Solution : introduire une interface avec (seulement) les méthodes métier



4. Design Patterns EJB

DP Home factory

Pb : accès à JNDI + récupération souche itf Home mélangés au code métier

⇨ mauvaise modularité
⇨ changements longs, sujets à erreur

```
/** JNDI: récupération ref. serveur de noms. */
Properties props = new Properties();
props.put( ... , "..." );
Context ic = new InitialContext(props);

/** Recherche du bean enregistré sous le nom MonCompte. */
Object obj = ic.lookup("MonCompte");

/** obj: souche d'accès à l'itf Home du compte -> cast */
CompteRemoteHome home = (CompteRemoteHome)
  PortableRemoteObject.narrow(obj,CompteRemoteHome.class);
```

DP Home factory

Solution

- déléguer la recherche des interfaces Home à une classe factory
+ design pattern singleton : 1 seule instance de HomeFactory

+ éventuellement : utilisation d'un cache d'instances Home

```
public class HomeFactory {
    private static HomeFactory hf = new HomeFactory();
    public static HomeFactory get() { return hf; }
    private static Contect ctx;
    public Context getContext() throws NamingException {
        if ( ctx == null ) {
            Properties props = new Properties();
            props.put( ... , "..." );
            ctx = new InitialContext(props);
        } return ctx; }
    ...
```


DP Home factory

Home Factory (2/2)

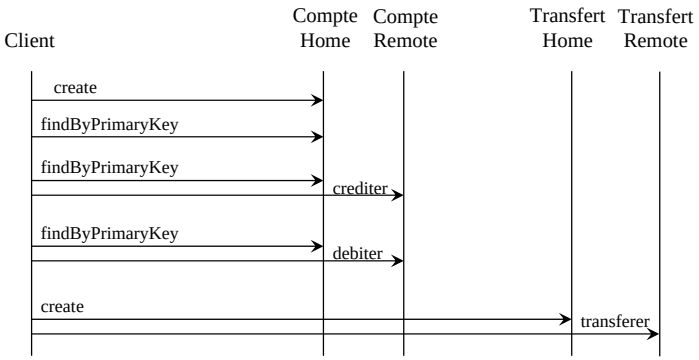
```
    private static Map homes = new HashMap();
    public EJBHome getHome( String name, Class cl )
        throws NamingException {
        EJBHome home = (EJBHome) homes.get(name);
        if ( home == null ) {
            Context ctx = getContext();
            Object obj = ctx.lookup(name);
            home = (EJBHome) PortableRemoteObject.narrow(obj,cl);
            homes.put(name,home);
        }
        return home; } }
```

Utilisation

```
HomeFactory hf = HomeFactory.get();
CompteRemoteHome home =
  (CompteRemoteHome) hf.getHome(
    "MonCompte",CompteRemoteHome.class);
```

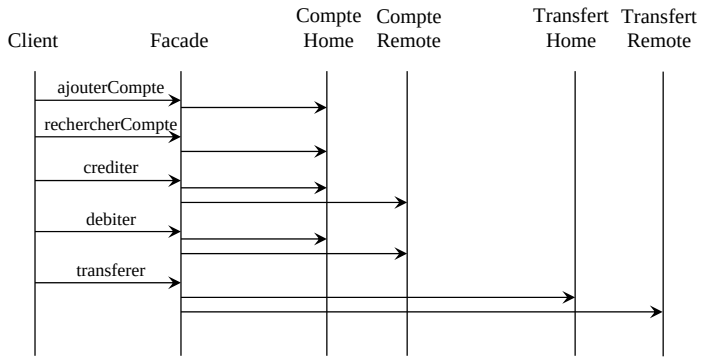

DP Session facade

Pb : nombreuses dépendances entre les clients et les beans



DP Session facade

Solution : présenter aux clients **une seule interface façade**
(*stateless session bean*)



4. Design Patterns EJB

DP Session facade

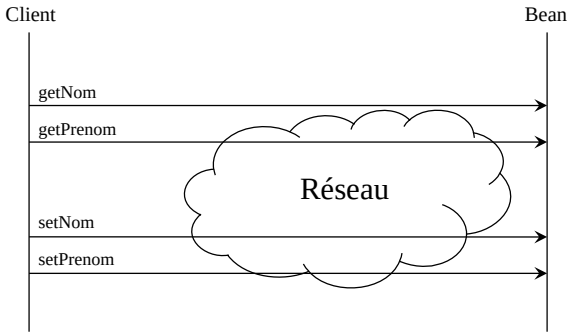
- traitements effectués par la façade minimaux
  essentiellement **délégation**
- éventuellement +sieurs façades sur un même ensemble de beans
  ⇨ différentes vues

`http://www.theserverside.com/cartoons/TalesFromTheServerSide.tss`

4. Design Patterns EJB

DP Data Transfert Object

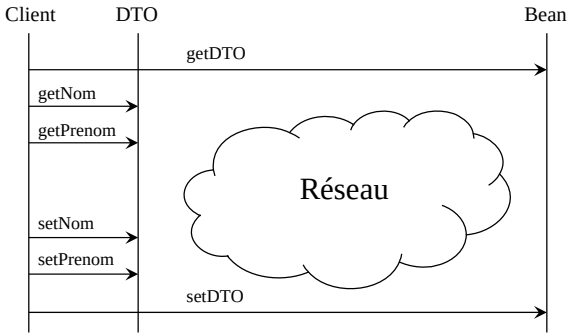
Aussi connu sous le terme : Value Object
Pb : nombreux échanges réseaux pour de simples get/set



4. Design Patterns EJB

DP Data Transfert Object

Solution : transmettre une instance par valeur



4. Design Patterns EJB

Autres DP

- application service : centraliser un processus métier s'étendant sur +sieurs beans
- composite entity : rassembler dans 1 seul EB les données persistantes de +sieurs EB
- transfert object assembler : aggregation de +sieurs DTO

- data access object :
  encapsuler l'accès et la manipulation des données persistantes dans un objet (en BMP)
- service activator : invoquer des services de façon asynchrone

5. Conclusion

EJB

Approche intéressante

- prise en charge de services techniques par la plate-forme
- le composant se focalise sur le métier
- *packaging* & déploiement ++

Quelques critiques/pistes d'améliorations possibles

- composants applicatifs (ok) / composants "systèmes" (not ok)
- Java
- domaine applicatif "limité" aux applications Java/Web
  ⇨ contrôle/commande ? temps-réel ? embarqué ? CAO ? infographie ? ...
- déploiement sur un seul serveur à la fois

5. Conclusion

EJB

Modèle de programmation parfois "lourd"

1. instanciation `new Compte("Lionel",167)` !! non !!

```
Context initialContext = new InitialContext();
CompteRemoteHome home = (CompteRemoteHome)PortableRemoteObject.narrow(
       initialContext.lookup("Compte"), CompteRemoteHome.class );
Compte cl = home.create("Lionel",167);
```

- long à écrire
- perte d'informations de typage ⇨ repose sur une chaîne **`"Compte"`**

2. transmission de référence `this` !! non !!

```
(CompteBean) sessionContext.getLocalObject()
```


5. Conclusion

EJB

17 règles de programmation à respecter !! (voir spec EJB)

1. EJBean ne doit pas lire/écrire champs non static
2. EJBean ne doit pas utiliser les primitives de synchronisation de thread
3. EJBean ne doit pas utiliser l'AWT
4. Ne pas utiliser java.io pour lire/écrire fichiers (⇨ sauv. données JDBC, JDO)
5. Ne pas utiliser les sockets
6. Ne pas manipuler les sockets factory
7. Ne pas créer de classloader
8. Ne pas essayer de gérer les threads (start, interrupt)
9. Ne pas lire directement les descripteurs XML
10. Ne pas charger de librairies natives
11. Ne pas tenter de modifier directement les règles de sécurité
12. Ne pas utiliser this

.....

5. Conclusion

Commerciaux

• WebSphere	IBM	`www-306.ibm.com/software/webservers`
• WebLogic	BEA	`www.weblogic.com`
• App Server	Borland	`www.borland.com/appserver`
• iPortal	IONA	`www.iona.com/products`

• Oracle, Sybase, HP, Fujitsu, ATG, Hitachi, Macromedia, SilverStream, ...

voir `java.sun.com/j2ee/compatibility.html`

Gratuits

• J2EE RI (Sun)	`java.sun.com/j2ee`
• JOnAS (ObjectWeb)	`www.objectweb.org`
• JBoss (JBoss Group)	`www.jboss.org`
• Geronimo (Apache)	`geronimo.apache.org`
• JFox	`www.huihoo.org/jfox`

5. Conclusion

EJB 3.0

- nouvelle version des spécifications
- conjointement avec JEE 5

- importantes simplifications dans le modèle de programmation des EJB
- utilise intensivement les annonations de Java 5

- nouveau modèle de persistance
  POJO
  + simple
  vers une unification des modèles EJB et JDO ?
- éliminitation des descripteurs de déploiement XML *server specific*
- suppression (autant que possible) des descripteurs de dépoiement XML
- + de tests pour valider les applications avant déploiement
- amélioration au niveau du support du développement Web