Master 2 Traitement
de l'Information

Université Claude Bernard Lyon 1

Architectures orientées composants

Conception d'Applications Hétérogènes Distribuées

Lionel Médini

Septembre-novembre 2015

Plan du cours

- Outils de programmation avancés
- Systèmes d'information distribués
- Architectures orientées composants
  - Principes fondamentaux
  - Quelques frameworks Java (Struts, Spring, JEE)
  - Exemples d'EJB 2 et de descripteurs de déploiement
  - EJB 3 : POJO et annotations

Principe général de la POC

- Une architecture est un assemblage de composants qui remplissent chacun un service particulier
- Chaque composant est un maillon d'un ou de plusieurs processus métier
- Les composants interagissent les uns avec les autres en échangeant des données
- Les composants sont faiblement couplés

➔*A priori*, rien de très nouveau par rapport à l'approche OO

➔Finalement, quelle différence avec l'objet ?

Spécificités d'un composant

- Un composant
  - Est une boîte noire correspondant à une interface donnée
  - Est une brique réutilisable du SI
  - Contient du code compilé séparément
    - Des composants différents peuvent être programmés dans des langages différents
    - L'implémentation est décorrélée de l'interface présentée
    - Indépendance des composants vis-à-vis de la maintenance et du cycle de vie

➔ Citez un exemple de structure pouvant contenir un composant en Java…

Typologies de composants (1/3)

- Composants techniques transverses
  - Ex : framework utilisé dans l'architecture
- Composants techniques « layer »
  - Ex : connecteur permettant la conversion des données échangée par d'autres composants

➔ Nécessitent un paramétrage

Typologies de composants (2/3)

- Composants applicatifs transverses
  - Ex : annuaire et gestion des utilisateurs, authentification, paye des salariés...
- Composants applicatifs métier
  - Ex (fac) : gestion de la scolarité, plateforme pédagogique...
  - Remarques : ce sont souvent les seuls composants à posséder une GUI

Typologies de composants (3/3)

- Composants applicatifs distribués
  - Composants métiers et transverses
- Composants IHM
  - Fournissent les points d'accès de l'utilisateur aux composants métier et applicatifs

- Remarque : pas d'incompatibilité entre les deux dernières dichotomies

  ➔ Nature du composant *vs* choix d'implémentation

Interopérabilité des composants

- Principe
  - Spécification d'une API pour l'échange de données entre les composants
- Problèmes
  - Plusieurs niveaux d'incompatibilité possibles
    - Technique : encodage des données
    - Informationnel : manque d'informations d'entrée pour un composant dans un processus
    - Sémantique : l'information y est, mais doit être interprétée

  ➔ Nécessite l'adjonction de composants d'interface (« connecteurs »)

Intégration des composants

- Lors d'un projet de conception, choix / développement des différents composants
  - Analyse de l'existant
  - Utilisation de référentiels
    - Schéma UML, … , architecture du SI
  - Choix des composants techniques
    - Frameworks, connecteurs, applications transverses

    ➔ Nécessitent un paramétrage
  - Conception / développement des composants métiers

    ➔ Spécifiques à l'application

Avantages de la POC

- Spécialisation des équipes de développement
- Sous-traitance de l'acquisition de composants
- Facilité de mise à jour
- Facilité de livraison/déplacement
- Choix des langages d'implémentation
- Productivité

Inconvénients de la POC

- Analyse et conception plus lourdes
  - Prise en compte de la généricité
  - Mécanismes d'encapsulation
- Appropriation des composants externes
- Développement des connecteurs

➔ Bénéfices sur le long terme

Outils de conception

- Patterns architecturaux
  - Conception basée sur des modèles (MDA)
  - Architectures Orientées Services
  - Référentiels de processus
  - …

Exemples d'outils

- Architectures Orientées Services
- Shared Objects (Linux), DLL (Windows)
- Services Web
- Beans (Java)
- Bundles OSGi
- ETL

Struts

http://struts.apache.org/

- Caractéristiques
  - Développement d'applications Web
  - Fondé sur le pattern MVC 2
    - Un contrôleur unique (ActionServlet)
    - Un délégateur de requêtes (ActionMapping)
    - Des « workers » (Action, *.do)
    - Des composants (Beans) pour le modèle métier
  - S'appuie sur les technologies servlets, JSP, JSTL...
- Avantage
  - Très connu / utilisé
- Inconvénients
  - Limité aux applications simples (modèle métier = quelques classes Java)
  - Difficilement testable

Spring

http://www.springframework.org/

- Caractéristiques
  - Framework fondé sur MVC 2
  - Conteneur léger (JavaBeans)
  - Support AOP : AspectJ
  - Intégration d'autres frameworks : Struts, JSF, AJAX DWR, support de portlets, ORM, JUnit...
- Avantages
  - Framework complet de développement d'applications
  - S'appuie sur d'autres frameworks ayant fait leurs preuves
- Inconvénient
  - Complexité croissante

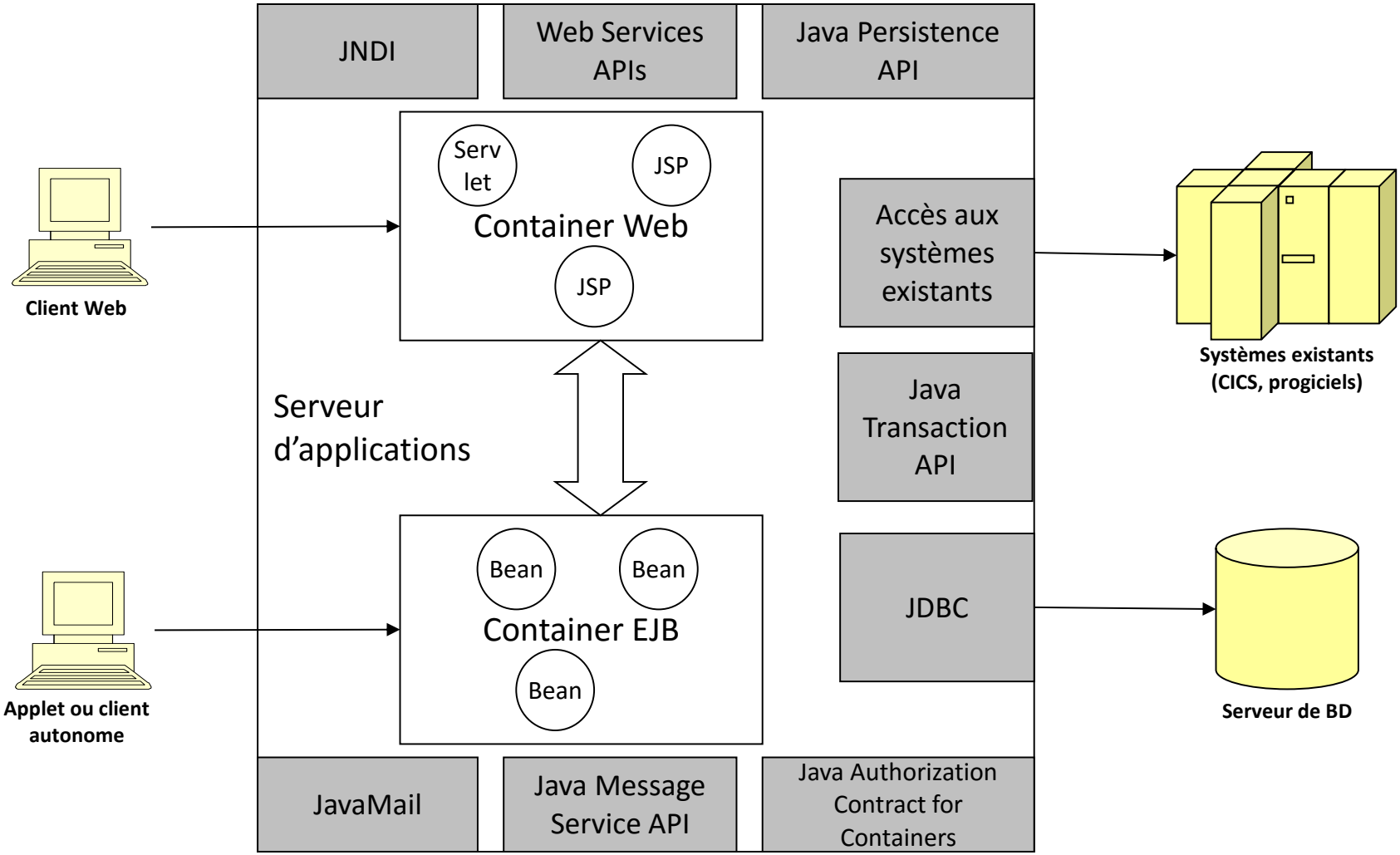
Java Enterprise Edition

- Caractéristiques
  - Plutôt une spécification qu'un framework
  - Intégré aux serveurs d'applications Java
  - Permet le développement d'applications intégrant des EJB
    - Conteneur EJB
    - Serveur JNDI
    - Gestion des services spécifiques (JTA, JMS, JCA...)
- Avantage
  - Très complet
  - Simplifié depuis la spécification EJB 3.0
- Inconvénient
  - Trop complet ?

Java Enterprise Edition

- But : développement, déploiement et exécution des applications distribuées
- Mêmes principes de base que les infrastructures middleware…
  - Communication synchrone (RMI/IIOP) et asynchrone (JMS) entre objets distribués, serveurs de nom (JNDI), intégration d'objets CORBA (JavaIDL)…
- …plus de nombreux services techniques supplémentaires
  - Génération d'objets transactionnels distribués (EJB), gestion du cycle de vie des objets, gestion des transactions (JTA et JTS), sécurité, accès aux BD (JDBC), interfaces graphiques dynamiques (Servlets , JSP, JSF)…

L'architecture Java EE



Les JavaBeans (1996)

- Définition
  - *Composants logiciels réutilisables* d'applications **non réparties** (en pratique, des classes Java)
- Structure
  - Les propriétés sont cachées et accessibles par des méthodes publiques *public String getNom()* et *public void setNom(String valeur)*
  - Les autres méthodes sont privées
- Utilisation depuis une JSP
  - Déclaration : *<jsp:useBean class="package.NomBean" id="testbean" scope="application" />*
  - Accès : *<%= testbean.getProperty0() %>*

Les JavaBeans

- Communications entre beans : le modèle événementiel
  - Principe des « écouteurs » (*listeners*) Java : un objet s'enregistre comme écouteur d'un certain événément (interface *java.util.EventListener*)
  - Un bean peut d'émettre un événement (classe *java.util.EventObject*), « capté » par le ou les écouteurs.

  $\Rightarrow$ Les beans peuvent être assemblés en applications
- Persistance : un bean doit pouvoir être sauvegardé et restitué (en pratique, il doit implémenter *java.io.Serializable*)
- Interface : un bean peut être manipulé *visuellement* dans un outil d'aide à la construction d'applications (dans ce cas, étendre *java.awt.Component*)
- Introspection
  - les propriétés et événements des beans sont découverts par *introspection* par l'outil de construction d'application (classe *XXXBeanInfo* qui implémente *java.beans.BeanInfo*)
  - La découverte des beans peut être réalisée par une instance de la classe *java.beans.Introspector*

Les Enterprise JavaBeans (1998)

- Définition : composants transactionnels accessibles à distance
- Un EJB est intégré à un serveur d'applications et représente une partie de la logique métier d'une application
- Un EJB est accessible via un conteneur qui permet
  - De l'intégrer à un serveur d'applications (accès local ou distant via une interface réseau)
  - De gérer les aspects distribués
  - De fournir des services supplémentaires
    - Aspects middleware, gestion du cycle de vie, sécurité (accès par les interfaces rendues disponibles par le conteneur), récupération d'un « contexte »…

Les Enterprise JavaBeans

- Trois types de beans
  - Beans session
    - Représentent les processus métiers
    - Méthodes accessibles par le client
    - Ne peuvent avoir qu'un seul client à un moment donné
    - Deux types de Beans session
      - Avec états : c'est le même client qui réalise toutes les invocations (exemple : panier électronique)
      - Sans état : le Bean peut être utilisé successivement par plusieurs clients (exemple : calcul d'une distance)

Les Enterprise JavaBeans

- Trois types de beans
  - Beans entités
    - Représentent des objets métiers persistants (exemple : une commande, un article, un compte bancaire…)
    - Permettent d'accéder aux données persistantes
    - Identifiés par des clés primaires (*Int*, *String*, *Object*)
    - Deux moyens de gérer la persistance
      - CMP : Container Managed Persistance
      - BMP : Bean Managed Persistance

Les Enterprise JavaBeans

- Trois types de beans
  - Beans messages (Message Driven Beans, MDB)
    - Composants asynchrones qui implémentent l'approche Message-Oriented Middleware (MOM)
    - Bidirectionnels (producteurs / consommateurs)
    - Peuvent utiliser un fournisseur Java Message Service (JMS)
      - Permettent d'abstraire la gestion de la « tuyauterie » JMS
        - » ConnectionFactory, Destination, Connection, Session…
      - Similaires aux EJB entités avec JPA
    - Les beans message ne sont en général pas accédés par les clients, mais par d'autres beans (pas d'interface home)

Les Enterprise JavaBeans

- Principes de base
  - Un EJB n'est pas accédé directement par le client, il dispose d'interfaces au niveau du conteneur
    - Interfaces « Home »
      - Gestion du cycle de vie (création de l'instance)
    - Interfaces métier
      - Présente les méthodes mises à disposition par la classe d'implémentation

    ⇒Depuis EJB 2.0, il existe des interfaces distantes et des interfaces locales

EJB session

- Exemples de nommage pour un EJB session sans état Calc (calculatrice)
  - Interfaces home

public interface CalcHome extends EJBHome {
public Calc create() throws RemoteException, CreateException; }

public interface CalcHomeLocal extends EJBLocalHome {
public CalcLocal create() throws CreateException; }

  - Interfaces métier

public interface Calc extends EJBObject {
public double add(double val1, double val2) throws RemoteException; }

public interface CalcLocal extends EJBLocalObject {
public double add(double val1, double val2); }

EJB session

- Classe d'implémentation (nom : *CalcBean*)
  - Code de la logique métier
  - Implémente une interface spécifique au type d'EJB
    - *javax.ejb.SessionBean*
    - *javax.ejb.EntityBean*
    - *javax.ejb.MessageDrivenBean*

    } Dérivent de l'interface *javax.ejb.EnterpriseBean*
  - Contenu
    - Méthodes métiers : *add()*
    - Constructeur : *CalcBean()*
    - Méthode de création : *ejbCreate()*
    - Méthodes de gestion du cycle de vie : *ejbActivate(), ejbPassivate(), ejbRemove()*

EJB session

- Classe d'implémentation (nom : *CalcBean*)
  - Exemple

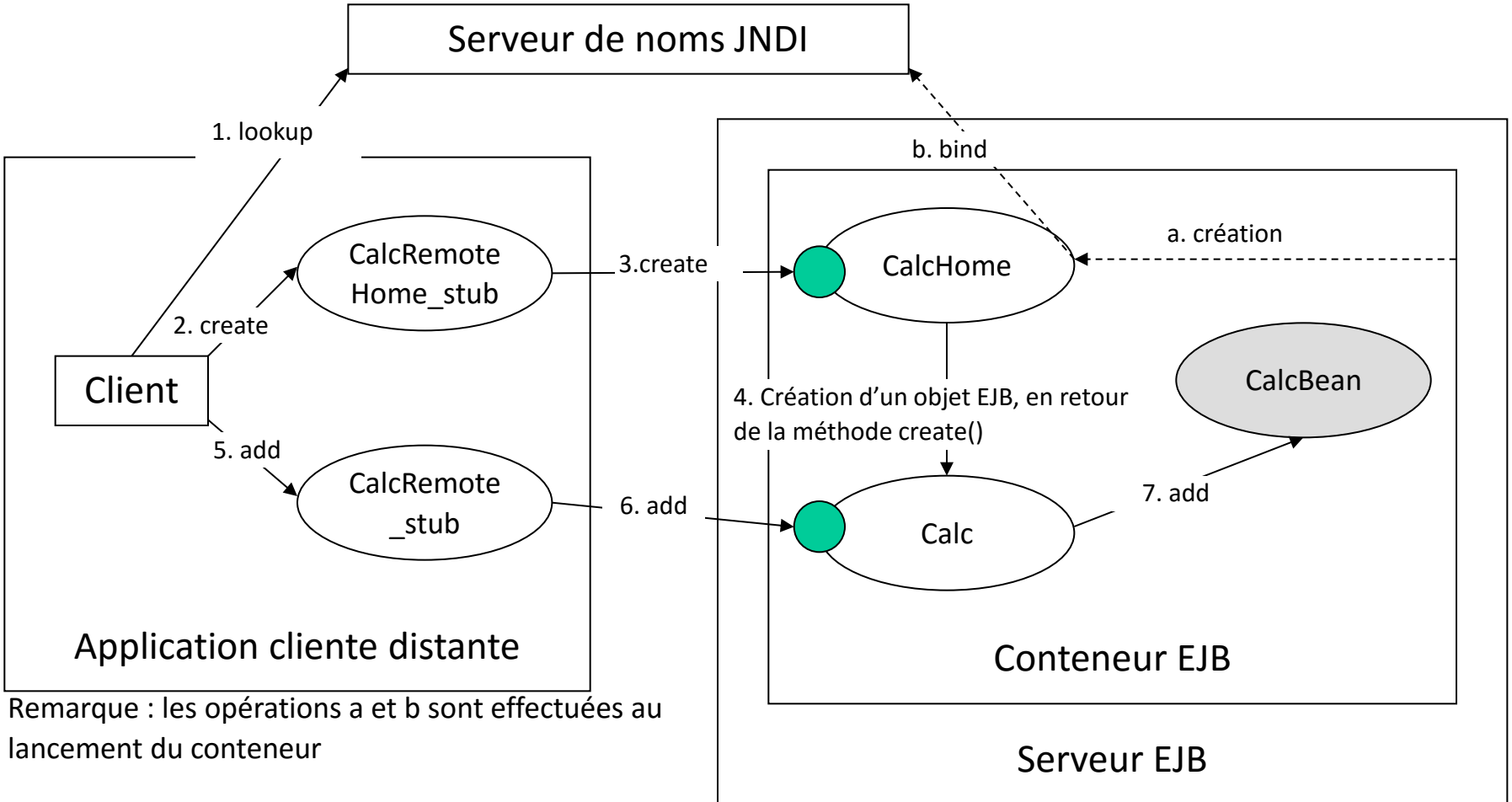
```
import javax.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class CalcBean implements SessionBean {

   public double add(double val1, double val2) {
      return val1+val2; }
   public CalcBean() {}
   public void ejbCreate() {}
   public void setSessionContext(SessionContext sc) {}
   public void ejbActivate() {}
   public void ejbPassivate() {}
   public void ejbRemove() {}
}
```

EJB session

- Fonctionnement du serveur et du client



Remarque : les opérations a et b sont effectuées au lancement du conteneur

EJB session

- Programmation du client
  - Tâches à effectuer
    - Contact du serveur de noms
    - Récupération d'une référence sur l'interface home distante du bean
    - Création d'un objet EJB pour accéder au bean
    - Invocation des méthodes métier

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

public class CalcClient {
  public static void main(String args[]) {
    try {
      // Obtention du contexte initial par défaut
      Context  initialContext  =  new InitialContext();

      //Recherche de l'interface home de distante de l'EJB
      Object objref = initialContext.lookup("Calc");
      CalcHome home = (CalcHome)PortableRemoteObject.narrow(objref, CalcHome.class);

      // Création et utilisation du bean
      Calc myCalc = home.create();
      System.out.println("3 + 2 = " + myCalc.add(3., 2.));

    } catch (Exception e) {
        e.printStackTrace(); }
  }
}
```

EJB session

- Cycle de vie
  - Pooling d'instances
    - Le conteneur possède un « pool » (réserve) d'instances qui lui évitent de créer et de détruire des objets pour chaque client
    - Le conteneur crée une instance de la classe d'implémentation d'un bean et la place dans le pool
      - À l'appel de la méthode create() par un client
      - S'il n'en a aucun de disponible
      - Dans la limite du nombre fixé par l'administrateur
    - Fonctionnement pour les différents types de beans session
      - Les beans sans état, une fois créés, restent opérationnels. Après utilisation, ils sont remis dans le pool ou détruits par le conteneur
      - Les beans avec états doivent être désactivés (*ejbPassivate()*) pour être remis dans le pool, et réactivés (*ejbActivate()*) avant de re-servir

EJB entités

- Principes généraux
  - Permettent l'accès aux données métier stockées
    - Dans une BD
    - Dans un système propriétaire
    - Dans un système de stockage hétérogène
  - Chaque EJB entité représente un concept métier
    - Exemple : un compte en banque, un client, un achat…

⇒Il y a autant d'instances d'EJB entités que de données archivées

⇒Les EJB ont une « identité », matérialisée par une classe de clé primaire

EJB entités

- Principes généraux
  - Les données sont accessibles par des méthodes spécifiques
  - Plusieurs utilisateurs peuvent utiliser le même bean en même temps
    - Accès successifs sans transaction
    - Accès transactionnels simultanés

    $\Rightarrow$La gestion des accès concurrents est effectuée par le container

EJB entités

- Persistance (gérée par le conteneur)
  - Rappels
    - un bean entité encapsule des données métier pour sécuriser leur utilisation
    - Chaque ensemble de données est représentée par un bean identifié (clé primaire)
    - Les données continuent d'exister après l'utilisation du bean

  ⇒Il faut pouvoir sauvegarder l'état d'un bean (sérialisation)

  ⇒La sauvegarde et la restauration des données sont des étapes critiques (gestion des transactions)

EJB entités

- Persistance (gérée par le conteneur)
  - États d'un bean entité
    - Inexistant
      - Levée d'une exception à partir de n'importe quelle méthode

      ⇒Doit être créé par une méthode newInstance(), suivie de setEntitycontext()
    - Dans le pool
      - Le bean est desactivé, et n'est associé à aucune donnée

      ⇒Peut être activé
    - Prêt
      - Le bean est associé à un objet entité déterminé (il connaît sa clé primaire)
      - Il peut exécuter des méthodes métier
      - Le conteneur appelle les méthodes *ejbLoad()* et *ejbStore()* pour gérer la synchronisation du bean avec le support de persistance

EJB entités

- Développement d'un bean CMP
  - L'interface métier distante
    - Définit l'interface métier accessible (données et méthodes)
    - Déclare des méthodes qui lancent des exceptions javax.rmi.RemoteException
    - Dérive de EJBObject : méthodes particulières héritées
      - public Object getPrimaryKey() throws RemoteException
      - public Boolean isIdentical(EJBObject) throws RemoteException
      - public void remove() throws RemoteException, RemoveException
  - L'interface métier locale
    - Mêmes méthodes que l'interface distante *a priori*
    - Pas d'exception RemoteException

EJB entités


```
public interface Exemple extends EJBObject {

  public InfosExemple getInfosExemple() throws remoteException;
  public void setInfosExemple(InfosExemple ie) throws
remoteException;
  ...
}

public class InfosExemple implements java.io.Serializable {

  public final Integer champInt;
  public final String champString;

  public InfosExemple(Integer i, String s) {
    champInt = i;
    champString = s;
  }
}
```

EJB entités

- Développement d'un bean CMP
  - L'interface home distante
    - Méthodes spécifiques héritées
      - void remove(Handle), void remove(Object)
    - Méthodes à déclarer
      - Création (createXxx) : insertion dans la dase de données
        public MonEJB create(…) throws RemoteException, CreateException
      - Finder (findXxx) : recherche de bean(s) existant(s)
        public MonEJB findByPrimaryKey(int) throws RemoteException, FinderException
        public Collection findByChampTexte(String) throws…
        » Déclarations obligatoires
        » Implémentées par le conteneur (requêtes EJB QL)

EJB entités

```
public Collection findByCategorie(String categorie);

<entity>
 <display-name>ExempleEJB</display-name>
 <ejb-name>ExempleEJB</ejb-name>
 ...
 <abstract-schema-name>ExempleSN</abstract-schema-name>
 ...
<query>
 <query-method>
  <method-name>findByCategorie</method-name>
  <method-params>
   <method-param>java.lang.String</ method-param>
  </method-params>
 </query-method>
 <ejb-ql>
  select Object(a) from ExempleSN a where a.categorie = ?1
 </ejb-ql>
</query>
```

EJB entités

- Développement d'un bean CMP
  - L'interface home locale
    - Mêmes méthodes que l'interface distante *a priori*
    - Pas d'exception RemoteException

EJB entités

- Développement d'un bean CMP/BMP
  - La classe d'implémentation
    - Bean BMP : codage de la sérialisation dans cette classe
    - Bean CMP : classe déclarée abstraite pour que le conteneur puisse la sous-classer et implémenter les méthodes de gestion de la persistance
    - Contenu (BMP)
      - Déclaration des méthodes de gestion des champs
      - Méthodes métiers (déclarées dans les interfaces métier)
      - Constructeur sans paramètre (appelé par le conteneur)
      - Méthodes de création (déclarées dans les interfaces home)
      - Méthodes sans entités (déclarées dans les interfaces home)
      - Méthodes internes d'accès aux données (méthodes Select)
      - Méthodes de rappel (appelées par le conteneur)

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Déclaration des méthodes de gestion des champs

```
public abstract String getChampTexte();
public abstract void setchampTexte(String texte);
public abstract Integer getChampInt();
public abstract void setchampInt(Integer int);
```

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Déclaration des méthodes de gestion des champs

      public abstract String getChampTexte();

      public abstract void setchampTexte(String texte);

      public abstract Integer getChampInt();

      public abstract void setchampInt(Integer int);
    - Implémentation des méthodes métier

      Cf. beans session
    - Constructeur

      …

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Méthodes de création
      - ejbCreateXxx() : initialise la création du bean
        - » Met à jour les champs à partir des paramètres (utilise les *setters*)
        - » Implémentation des méthodes déclarées dans les interfaces home
        - » Type de la valeur de retour = type de la clé primaire
        - » Doit retourner *null* (c'est le conteneur qui retrouve la clé primaire)

```
public Integer ejbCreateAvecUnParametre(Type1 param1) throws
    CreateException {
     if (param1==null) throw new CreateException;
    else setParam1(param1);
    return null; }
```

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Méthodes de création
      - ejbPostCreateXxx() : appelée une fois le support de persistance (BD) mis à jour
        - » Traitements nécessaires pour finaliser la création du bean
        - » Ne doivent pas obligatoirement comporter une clause *throw CreateException*
        - » Type de retour void
        - » Mêmes noms et paramètres que la méthode create() correspondante

        public void ejbPostCreateAvecUnParametre(Type1 param1) { }

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Méthodes sans entité : ejbHomeXxx
      - Doivent être déclarées dans les interfaces Home
      - Permettent de faire des traitements de lots

        Ex : trouver combien de clients se sont connectés à une certaine date.
      - Peuvent utiliser des méthodes Finder ou des requêtes à la base
      - Peuvent nécessiter de nombreux accès aux données

      $\Rightarrow$ À ne pas utiliser

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Méthodes Select : ejbSelectXxx
      - Méthode abstraite (instanciée par le conteneur)
      - À usage interne (contrairement aux méthodes *Finder*)
      - Déclaration :

      public abstract Collection ejbSelectObjetsCommeCa(String ca) throws FinderException;
      - Association avec une requête EJB QL définie dans le descripteur de déploiement

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation

```
<query>
 <query-method>
  <method-name>
    ejbSelectObjetsCommeCa
  </method-name>
  <method-params>
   < method-param>java.lang.String</ method-param>
  </method-params>
 </query-method>
 <ejb-ql>
  select a.idDonnees from DonneesSN a where a.champCa = ?1
 </ejb-ql>
</query>
```

EJB entités

- Développement d'un bean CMP
  - La classe d'implémentation
    - Les méthodes de rappel : setEntitycontext(), unsetEntitycontext(), ejbActivate(), ejbLoad(), ejbStore(), ejbRemove(), ejbPassivate()
      - Sont appelées par le conteneur pour communiquer avec le bean
      - Pour les beans CMP, 5 d'entre elles n'ont pas besoin d'être définies, mais il faut en fournir une implémentation vide :

        public void ejbXxx() {}
      - Les méthodes de contexte permettent de mettre à jour une variable globale de type EntityContext de la classe d'implémentation

EJB entités

- Développement d'un bean CMP

```
public abstract class MonEJBBean implements
javax.ejb.EntityBean {
  EntityContext ec = null;
  public void setEntitycontext(javax.ejb.EntityContext param) {
   ec = param; }
  public void unsetEntitycontext( ) {
   ec = null; }
  public void ejbActivate( ) { }
  public void ejbLoad( ) { }
  public void ejbStore( ) { }
  public void ejbRemove( ) { }
  public void ejbPassivate( ) { }

  […]
}
```

EJB entités

- Développement d'un bean CMP
  - La clé primaire
    - Permet d'identifier le bean de manière unique
    - Classe sérialisable
      - Dérivant de java.lang.Object (Integer, String…)
      - Spécifique au bean MonBeanId, mais la clé doit être composée d'un seul champ
      - Classe annexe sérialisable : méthodes à implémenter
        - » Constructeur par défaut
        - » public boolean equals(Object other)
        - » public int hashcode ()
    - Le descripteur de déploiement doit indiquer
      - Le type de la classe (balise <prim-key-class>)
      - Le nom du champ (balise <primkey-field>)

EJB messages

- Principe général
  - Utilisation avec JMS
  - Abstraction de la mécanique par configuration
  - Interfaces spécifiques au modèle choisi
    - Rappel (JMS)

Générique	Queue	Topic
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection		TopicConnection
Destination	Queue	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher

  - Pas d'interface Home (accédés par d'autres beans)

EJB messages

- Principe général
  - Un MDB implémente deux interfaces
    - javax.ejb.MessageDrivenBean
      - public void setMessageDrivenContext(MessageDrivenContext ctx) throws EJBException;
      - public void ejbRemove( ) throws EJBException;
    - javax.jms.MessageListener
      - public void onMessage(Message msg);
  - Configuration dans un descripteur de déploiement (ejb-jar.xml)
    - Spécifie le modèle, le type de transaction, la destination…

Les Enterprise JavaBeans

- Packaging et déploiement
  - Packaging : assembler tous les éléments dans un fichier ejb-jar
    - Interfaces métier et home (locales et/ou distantes)

Calc.class, *CalcLocal.class*, *CalcHome.class*, *CalsHomeLocal.class*

    - Classe d'implémentation

      CalcBean.class

    - Classe de clé primaire (beans entités)

      MyEntityBeanKey.class

    - Descripteur de déploiement

      ejb-jar.xml

Les Enterprise JavaBeans

- Packaging et déploiement
  - Déploiement : réalisé par le serveur d'applications
    - Extraction du contenu du JAR
    - Génération du code déployé (code nécessaire à la communication du bean et de ses clients)
      - Objets EJB local et distant
        - » Implémentent l'interface métier
        - » Peuvent prendre en charge les fonctions de gestion de l'EJB (persistance, sécurité, transactions…)
      - Objets EJB Home local et distant
        - » Implémentent l'interface home (méthode *create()*)
      - Stubs, skeletons, etc. en fonction du type de serveur
    - Enregistrement d'une référence à l'objet EJB Home dans un serveur de noms accessible via JNDI

Les Enterprise JavaBeans

- Principes de base
  - Descripteur de déploiement : indique au serveur J2EE la structure d'un EJB (fichier XML)
  - Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
 <description>Descripteur de déploiement du bean Calc</description>
 <display-name>Calc</display-name>
 <enterprise-beans>
  <session>
   <description>Calculatrice simple</description>
   <display-name>Calc</display-name>
   <ejb-name>Calc</ejb-name>
   <home>calcul.CalcHome</home>
   <remote>calcul.Calc</remote>
   <ejb-class>calcul.CalcBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
  </session>
 </enterprise-beans>
 <assembly-descriptor>
  <container-transaction>
    <method>
     <ejb-name>Calc</ejb-name>
     <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
 </assembly-descriptor>
</ejb-jar>
```

Les EJB entités

- Configuration d'un bean CMP
  - Le descripteur de déploiement
    - Spécificités par rapport aux beans session

```
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN'
'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>
<ejb-jar>
 <enterprise-beans>
  <entity>
   <ejb-name>ExempleBean</ejb-name>
   <home>com.demo.ExempleHome</home>
   <remote>com.demo.ExempleObject</remote>
   <ejb-class>com.demo.ExempleBean</ejb-class>
   <persistence-type>Container</persistence-type>
   <prim-key-class>java.lang.String</prim-key-class>
   <reentrant>False</reentrant>
   <abstract-schema-name>ExempleSN</abstract-schema-name>
   <cmp-field>
    <field-name>ExempleId</field-name>
   </cmp-field>
   <cmp-field>
    <field-name>ExempleData</field-name>
   </cmp-field>
   <primkey-field>ExempleId</primkey-field>
   <query>
    <query-method>
     <method-name>findByCategorie</method-name>
     <method-params>
      <method-param>java.lang.String</ method-param>
     </method-params>
    </query-method>
    <ejb-ql>select Object(a) from ExempleSN a where a.categorie = ?1</ejb-ql>
   </query>
   <query>...</query>
  </entity>
 </enterprise-beans>
 <assembly-descriptor>
  ...
```

Les Enterprise JavaBeans

- Conclusion
  - Les EJB 2.0 sont des composants
    - Sûrs
    - Pratiques
    - puissants
    - Extensibles
    - … mais pas simples

  ⇒Nécessitent

  ⇒Une bonne connaissance de l'API EJB JEE

  ⇒D'écrire beaucoup de code « inutile »

Les EJB 3.0

- Principe
  - Conserver/augmenter la puissance des EJB 2
    - Mêmes mécanismes sous-jacents
    - Intégration de fonctionnalités JEE5
  - Simplifier l'utilisation pour le développeur
    - Le cycle de vie est géré par le framework
    - Utilisation de POJOs
    - Annotations Java
    - Injection de dépendances
    - Utilisation sous JSE ou JEE

Les EJB 3.0

- Exemple
  - Bean session sans état
    - Interface

```
package calc;
@Remote
public interface AddRemote {
    public float add(float a, float b);
}
```

    - Classe d'implémentation

```
package calc;
@Remote(AddRemote.class)
@Stateless(name="calc")
public class AddBean implements AddRemote {
    public float add(float a, float b) {
        return a+b;
    }
}
```

Les EJB 3.0

- Exemple
  - Bean session avec état
    - Interface

```
@Local
public interface MemLocal {
    public void setMem(float a);
    public float getMem();
}
```

    - Classe d'implémentation

```
@Local(name="mem")
@Stateful(MemLocal)
public class MemBean implements MemLocal {
    float mem = 0.;
    public void setMem(float a) {
        mem = a;
    }   ...
}
```

Les EJB 3.0

- Exemple
  - Bean entité

```
@Entity
public class Personne {
    @Id
    @GeneratedValue
    private int idPersonne;
    private String nom;
    private String prenom;
    ...
}
```

Les EJB 3.0

- Exemple
  - Bean message

```
@MessageDriven(mappedName="jms/Queue", activationConfig =  {
    @ActivationConfigProperty(propertyName = "acknowledgeMode",
                              propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName = "destinationType",
                              propertyValue = "javax.jms.Queue")
})

public class MyMessageBean implements MessageListener {
    @Resource
    private MessageDrivenContext mdc;

    public void onMessage(Message message) {
        ...
    }
}
```

Source : http://docs.oracle.com/javaee/6/tutorial/doc/bnbpk.html

Les EJB 3.0

- Exemple
  - Client d'un bean session

```
public class Main {
 static AddRemote add;
 static String BeanName = "ejb:/calc_source//AddBean!calc.AddRemote";

 public static void main(String[] args) {
    float a = 3, b = 3;
    add = Main.lookupAddBean(BeanName);
    System.out.println("3+3=" + add.add(a,b)); //Utilisation du bean
  }
 public static AddRemote lookupAddBean(name) {
   final Hashtable jndiProps=new Hashtable();
   jndiProps.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");
   final Context ctxt = new InitialContext(jndiProps);
   try {
     //Récupération du serveur JNDI
     Context c = new InitialContext(jndiProps);
     //Récupération de la référence sur une instance du bean
     return (AddRemote) c.lookup(name);
   } catch (NamingException ne) { ... } } }
```

Les EJB 3.0

- Exemple
  - Client d'un bean session (servlet)

```
@WebServlet
public class ClientServlet extends javax.servlet.http.HttpServlet {
    private AddRemote calc;

    @EJB(name = "calc")
    public void setAdd (AddRemote calc) {
        this.calc = calc;
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp)...
}
```

  - Remarques
    - Le nom JNDI complet de l'EJB est EARName/BeanName/remote
    - L'utilisation de ces annotations nécessite que l'application Web utilise un descripteur de déploiement avec la version 2.5 (ou supérieure)

Injection de dépendances

- Injection par annotations (JavaEE)
  - Singletons (contexte) :

```
@Resource
javax.ejb.SessionContext ctx;
```

  - Ressources mappées dans JNDI :

```
@Resource(mappedName="DefaultDS")
private javax.sql.DataSource ds;
```

  - EJB

```
@EJB(name="calc")
```

Serveurs d'applications Java EE

- Oracle
  - GlassFish
  - WebLogic server
- Red Hat
  - WildFly / JBoss
- IBM
  - WebSphere
- Apache
  - Geronimo
  - TomEE

Références

- Quelques liens pour l'utilisation d'EJB 3 dans JBoss 7.1
  - JBoss : http://jboss.org/
  - EJB + Maven dans Eclipse : https://community.jboss.org/wiki/SettingUpTheJBossEnterpriseRepositories?_sscc=t
  - JBoss Tools (plugin Eclipse) :
    - http://download.jboss.org/jbosstools/updates/development/indigo/
    - http://kousikraj.wordpress.com/tag/jboss-as-7-server-adapter-for-eclipse/
  - Docs JBoss Tools : http://docs.jboss.org/tools/3.3.0.Final/en/beginners_guide/html_single/index.html
  - Documentation JBoss sur la création d'un client : https://docs.jboss.org/author/display/AS71/EJB+invocations+from+a+remote+client+using+JNDI