



# Master STS Mention Informatique Parcours ISI

Programmation distribuée  
M1 / 178EN003

## C4.3 – EJB 3.0

Thierry Lemeunier

[thierry.lemeunier@univ-lemans.fr](mailto:thierry.lemeunier@univ-lemans.fr)

[www-lium.univ-lemans.fr/~lemeunie](http://www-lium.univ-lemans.fr/~lemeunie)

---

# Plan du cours

- Communication par composant distribué :
  - Introduction à *EJB* 3.0
    - Principes
    - Les session bean
    - Les message-driven bean
    - Le cycle de vie des EJB

# EJB 3.0 – Principes (1 / 4)

- Un *EJB* est un composant côté serveur installé dans un conteneur Java EE et qui délivre un service métier au client
- Les deux types d'*EJB* :
  - *Session Bean* :
    - *EJB* synchrone
    - Il représente une session avec un client au sein du serveur d'application
  - *Message-Driven Bean* :
    - *EJB* asynchrone
    - Il reçoit des messages en provenance d'un service de messagerie auprès duquel il est abonné ou de n'importe quel autre composant
- Remarques :
  - L'*Entity Bean* a été remplacé par les entités de l'API de persistance
  - La spécification 3.0 a simplifié la mise en œuvre des *EJB* :
    - Le descripteur de déploiement n'est plus obligatoire
    - L'usage d'annotations rend plus transparent les liens avec le conteneur et facilite la description du déploiement (accès aux ressources, transaction, etc.)
    - Annotation : métadonnée incluse dans le code (apparue avec Java SE 5)

# EJB 3.0 – Principes (2/4)

- Description des dépendances aux ressources
  - Un composant *EJB* peut avoir besoin d'accéder à certaines ressources (base de données, service de messagerie, etc.)
  - Les ressources sont enregistrées dans le service de nom via JNDI par les administrateurs
  - Il existe trois moyens pour accéder à des ressources :
    - Soit dans le descripteur de déploiement du composant
    - Soit dans le fichier java par codage explicite d'accès au service JNDI
    - Soit dans le fichier java par injection de dépendances (annotation)
  - On peut panacher ces trois modes d'accès pour plus de souplesse
- Injection de dépendances
  - Le code est annoté avec *@Ressource* ou d'autres annotations spécifiques
  - Les attributs d'une ressource sont annotés ou déduits du code (cf. doc !)
  - Le conteneur est chargé « d'injecter » (initialiser) la ressource
  - La ressource peut-être annotée soit :
    - Au niveau de la classe entière (injection à l'exécution)
    - Au niveau d'un attribut (injection à l'initialisation de l'application)
    - Au niveau d'une méthode setter (idem)

# EJB 3.0 – Principes (3/4)

## Exemples d'injection de ressources

```
@Resources({
    @Resource(name="myMessageQueue", type="javax.jms.ConnectionFactory"),
    @Resource(name="myMailSession", type="javax.mail.Session")
})
public class SomeMessageBean { ... }
```

```
package com.example;
public class SomeClass {
    @Resource private javax.sql.DataSource myDB;
    ...
}
```

JNDI name: *com.example.SomeClass/myDB*  
Ressource type: *javax.sql.DataSource.class*

```
package com.example;
public class SomeClass {
    private javax.sql.DataSource myDB;
    ...
    @Resource(name="customerDB") private void setMyDB(javax.sql.DataSource ds) {
        myDB = ds;
    }
    ...
}
```

JNDI name: *customerDB*  
Ressource type : *javax.sql.DataSource.class*

# EJB 3.0 – Principes (4/4)

## ■ Développement d'un *EJB*

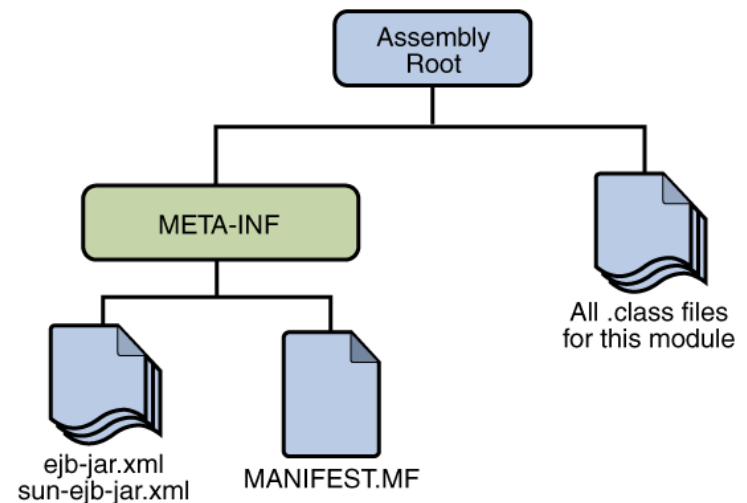
### □ Il faut fournir :

- Une interface métier
- Une classe *bean* d'implémentation de l'interface métier
- Les classes utilitaires ou d'exception utilisées par le *bean*
- Des fichiers de déploiement éventuels

### □ Il faut créer un module (fichier jar)

## ■ Convention de nommage

EJB name	<name>Bean	AccountBean
EJB class	<name>Bean	AccountBean
Business interface	<name>	Account



# Plan du cours

- Communication par composant distribué :
  - Introduction à *EJB* 3.0
    - ✓ Principes
    - Les session bean
    - Les message-driven bean
    - Le cycle de vie des EJB

# EJB 3.0 – *Session Bean* (1 / 4)

- Définition d'un *session bean*
  - Il représente une session d'interaction entre un client qui invoque ses méthodes et le serveur d'application
  - Il implémente une interface métier (non si c'est un service web)
  - Il est unique par client
  - Il n'est pas persistant (variables d'instances non stockées)
- Les deux types de *session bean*
  - *Stateful Session Bean* :
    - L'état « conversationnel » de la session (les variables d'instances) est maintenu tant que la session n'est pas terminée par le client
    - Cela permet de maintenir des informations concernant le client utiles à plusieurs invocations de méthode (pour gérer un *workflow...*)
    - La classe est annotée avec `@Stateful`
  - *Stateless Session Bean* :
    - Les variables d'instance ne stockent pas d'informations du client
    - Le service fournit est simple (les paramètres suffisent) et identique pour tous les clients (pas besoin de le personnaliser)
    - La classe est annotée avec `@Stateless`



# EJB 3.0 – *Session Bean* (2/4)

- Les 3 types d'accès d'un *session bean* :
  - Distant :
    - Le client s'exécute dans une JVM différente de celle du *bean*
    - Le client : une application cliente, un autre *bean*, un composant web
    - Deux manières de créer un accès distant :
      - Soit annoter la classe interface métier avec `@Remote`
      - Soit annoter la classe du *bean* avec `@Remote(InterfaceName.class)`
  - Local :
    - Le client s'exécute dans la même JVM que celle du *bean*
    - Le client (un autre *bean* ou un composant web) doit connaître le *session bean*
    - Deux manières de créer un accès local :
      - Soit annoter la classe interface métier avec `@Local`
      - Soit annoter la classe du *bean* avec `@Local(InterfaceName.class)`
  - Web :
    - Le *session bean* fournit un service web demandé par le client via HTTP
    - Le *session bean* est annotée avec `@Stateless` et `@WebService...` (cf. doc !)
- Remarques :
  - L'accès distant se fait avec *RMI-IIOP* (protocole Corba)
  - On peut définir à la fois un accès local et un accès distant sur la classe d'implémentation !

# EJB 3.0 – Session Bean (3/4)

## *Extrait d'une l'interface métier*

```
...  
@Remote  
public interface Cart {  
    public void initialize(String person) throws BookException;  
    public void initialize(String person, String id) throws BookException;  
    public void addBook(String title);  
    public void removeBook(String title) throws BookException;  
    public List<String> getContents();  
    public void remove();  
}
```

## *Extrait d'un composant à état*

```
...  
@Stateful  
public class CartBean implements Cart {  
    List<String> contents;  
    String customerId;  
    String customerName;  
    ...  
    @Remove public void remove() {  
        contents = null;  
    }  
}
```

## *Extrait d'un composant web*

```
...  
@Stateless  
@WebService  
public class HelloServiceBean {  
    private String message = "Hello, ";  
    public void HelloServiceBean() {}  
    @WebMethod  
    public String sayHello(String name) {  
        return message + name + ".";  
    }  
}
```

## *EJB 3.0 – Session Bean (4/4)*

```
...
public class CartClient {
    @EJB private Cart cart;
    ...
    public void doTest() {
        try {
            cart.initialize("Duke d'Url", "123");
            cart.addBook("Kafka on the Shore");
            ...
            cart.remove();
            System.exit(0);
        } catch (BookException ex) {
            System.err.println("Caught a BookException: " + ex.getMessage());
            System.exit(1);
        } catch (Exception ex) {
            System.err.println("Caught an unexpected exception!");
            ex.printStackTrace();
            System.exit(1);
        }
    }
}
```

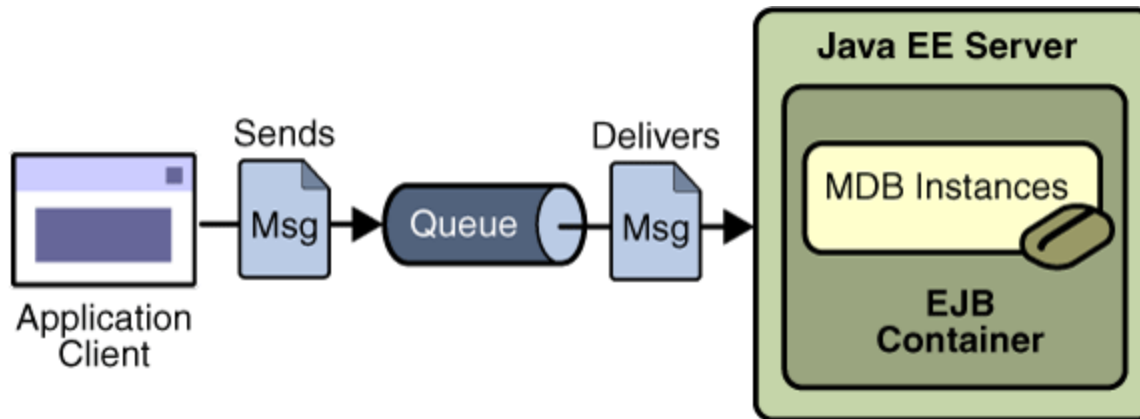
# Plan du cours

- Communication par composant distribué :
  - Introduction à *EJB 3.0*
    - ✓ Principes
    - ✓ Les session bean
  - Les message-driven bean
  - Le cycle de vie des EJB

# *EJB 3.0 – Message-Driven Bean (1/3)*

## ■ Définition

- ❑ Il permet un fonctionnement asynchrone
- ❑ Le client n'y accède pas directement
- ❑ Il implémente la classe *MessageListener*
- ❑ Il est sans état et non persistant
- ❑ Il est annoté avec *@MessageDriven*



## *EJB 3.0 – Message-Driven Bean (2/3)*

...

```
@MessageDriven(mappedName = "jms/Queue")
public class SimpleMessageBean implements MessageListener {
    static final Logger logger = Logger.getLogger("SimpleMessageBean");
    @Resource private MessageDrivenContext mdc;

    public SimpleMessageBean() {}

    public void onMessage(Message inMessage) {
        TextMessage msg = null;

        try {
            if (inMessage instanceof TextMessage) {
                msg = (TextMessage) inMessage;
                logger.info("MESSAGE BEAN: Message received: " + msg.getText());
            } else {
                logger.warning("Message of wrong type: " + inMessage.getClass().getName());
            }
        } catch (JMSEException e) {
            e.printStackTrace(); mdc.setRollbackOnly(); // Cancel permanently the transaction
        } catch (Throwable te) { te.printStackTrace(); }
    }
}
```

## *EJB 3.0 – Message-Driven Bean (3/3)*

...

```
public class MDBAppClient {  
    @Resource(mappedName="MDBQueueConnectionFactory")  
    private static QueueConnectionFactory queueCF;  
  
    @Resource(mappedName="jms/Queue")  
    private static Queue mdbQueue;  
  
    public static void main(String args[]) {  
        try {  
            QueueConnection queueCon = queueCF.createQueueConnection();  
            QueueSession session = queueCon.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);  
            QueueSender queueSender = session.createSender(null);  
            TextMessage msg = session.createTextMessage("hello");  
            queueSender.send(mdbQueue, msg);  
            queueCon.close();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

# Plan du cours

- Communication par composant distribué :
  - Introduction à *EJB 3.0*
    - ✓ Principes
    - ✓ Les session bean
    - ✓ Les message-driven bean
  - Le cycle de vie des EJB

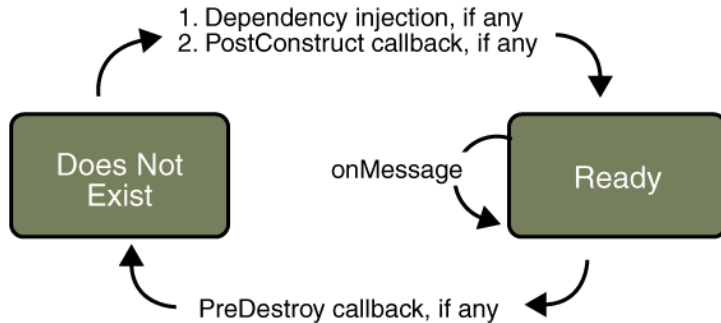


# *EJB 3.0 – La vie des EJB (1 / 3)*

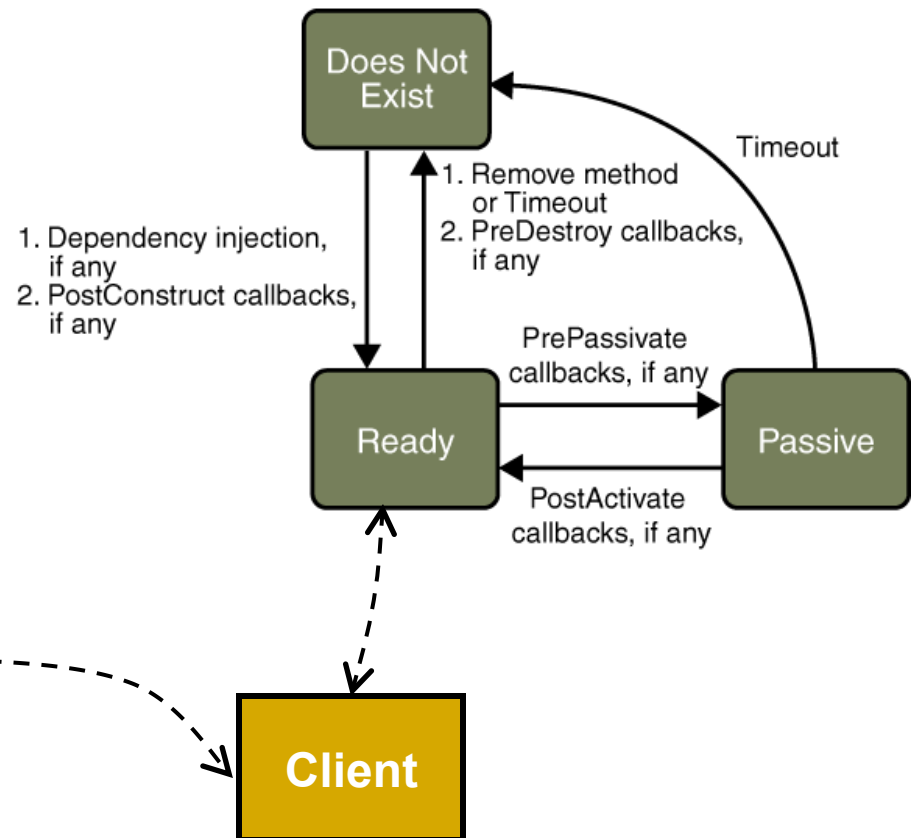
- La vie d'un *EJB* passe par les étapes suivantes :
  - ❑ Le client obtient une référence sur l'*EJB*
  - ❑ Le conteneur procède aux injections de dépendances
  - ❑ Le conteneur invoque la méthode annotée *@PostConstruct*
  - ❑ L'*EJB* est prêt à recevoir des invocations de méthodes
  - ❑ Dans le cas d'un *session bean*, le client invoque la méthode annotée *@Remove* pour fermer la connection
  - ❑ Le conteneur invoque l'opération annotée *@PreDestroy* avant d'être prêt pour le *garbage collector*
  
- Pour gagner de la place mémoire, un *stateful session bean* ancien non fermé peut être temporairement stocké sur disque
  - ❑ Le conteneur invoque la méthode annotée *@PrePassivate* juste avant la désactivation
  - ❑ Si le client invoque une méthode, le container active le *bean* puis invoque la méthode annotée *@PostActivate*
  - ❑ Le *stateful bean* exécute ensuite la méthode invoquée

# EJB 3.0 – La vie des EJB (2/3)

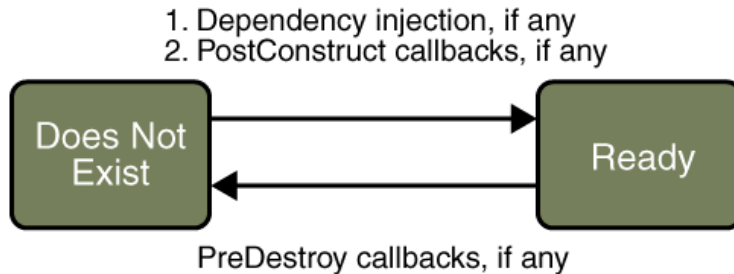
## Message-Driven Bean Lifecycle



## Stateful Session Bean Lifecycle



## Stateless Session Bean Lifecycle



# EJB 3.0 – La vie des EJB (3/3)

**@Stateless(mappedName = "ejb/stateless/Order")**

```
public class OrderBean implements OrderRemote, OrderLocal {  
    private ConnectionFactory connectionFactory;  
    private Connection connection;  
  
    @PostConstruct public void openConnection() {  
        connection = connectionFactory.createConnection();  
    }  
  
    @PreDestroy public void closeConnection() {  
        if (connection != null) connection.close();  
    }  
}
```

**@Stateful**

```
public class ShoppingCartBean implements ShoppingCartLocal {  
    private List<CartItem> cartItems;  
  
    @PostConstruct public void initialize() {  
        cartItems = new ArrayList<CartItem>();  
    }  
  
    @PreDestroy public void clear() {  
        cartItems = null;  
    }  
}
```