Nancy-Université

Mise en œuvre des serveurs d'application

UE 203d

Master 1 IST-IE

Printemps 2008

Ces transparents, ainsi que les énoncés des TDs, seront disponibles à l'adresse :

`http://www.loria.fr/~burel/empty_cours.html`

Quatrième partie IV

Logique métier

Plan

- Introduction
- Entreprise Java Beans
  - Bean session
  - Bean entité
  - Bean contrôlé par messages
- Configuration
- XDoclet

Application de la bibliothèque imaginaire

Nous avons établi la couche web, il nous faut maintenant créer la couche métier

Services demandés

Il faut définir clairement quelles requêtes les clients peuvent demander à la couche métier

- emprunt d'un livre
- retour d'un livre
- recherche d'un livre par titre
- recherche des livres empruntés
- ...

Base de données

Il nous faut aussi savoir la façon dont les données persistantes sont conservées

- livres
- utilisateurs
- relation entre les deux

Entreprise Java Beans

Classes java implémentant les interfaces

- `javax.ejb.EntityBean`
- `javax.ejb.SessionBean`
- `javax.ejb.MessageDrivenBean` et `java.ejb.MessageListener`

Contenu d'un conteneur d'EJB

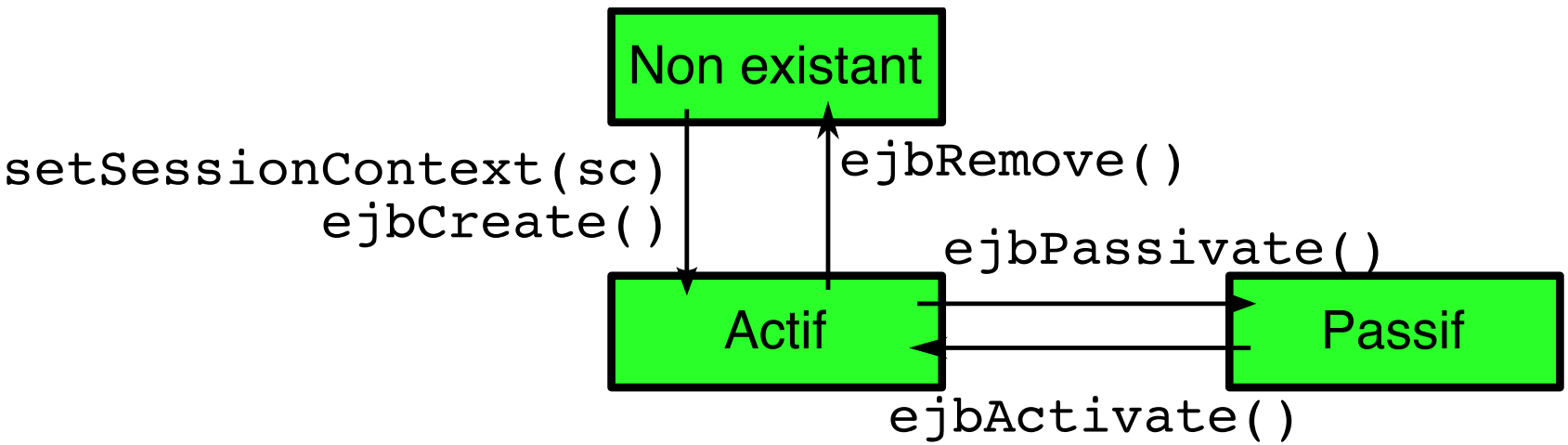
Fichier `.jar` contenant :

- ensemble des classes des beans, y compris leurs interfaces distantes, maison, locales et locales maison.
- un répertoire `META-INF` contenant :
  - `ejb-jar.xml` : configuration du conteneur
  - `MANIFEST.MF` : informations sur le conteneur
  - d'autres fichiers de configuration spécifiques au serveur d'application utilisé (ex : `jboss.xml`, `sun-ejb-jar.xml`, `jbosscmp-jdbc.xml`...)

Plan

- Introduction
- Entreprise Java Beans
  - Bean session
  - Bean entité
  - Bean contrôlé par messages
- Configuration
- XDoclet

Cycle de vie d'un bean session



Interface

javax.ejb.SessionBean :

```
public void ejbCreate();
public void ejbRemove();
public void ejbActivate();
public void ejbPassivate();
public void setSessionContext(SessionContext sc);
```

Interface distante

On y définit les méthodes que le client peut appeler
Exemple :

```
package biblio;
public interface Admin extends javax.ejb.EJBObject {

  public Date emprunte(String nom, String cote)
    throws java.rmi.RemoteException;

  public void retourneLivre(String cote)
    throws java.rmi.RemoteException;

  ...
}
```

Interface maison

On y définit les méthodes concernant toutes les instances du bean
Exemple :

```
package biblio;
public interface AdminHome extends javax.ejb.EJBHome {

  public Admin create()
    throws java.rmi.RemoteException,
           javax.ejb.CreateException;

}
```

Interfaces locales

- Interface locale : étend `javax.ejb.EJBLocalObject`
  Comme interface distante, mais pas de `RemoteException`
  Peut fournir méthodes différentes de l'interface distante
- Interface locale maison : étend `javax.ejb.EJBLocalHome`
  Comme interface maison, pas de `RemoteException` mais quand même des `CreateException`

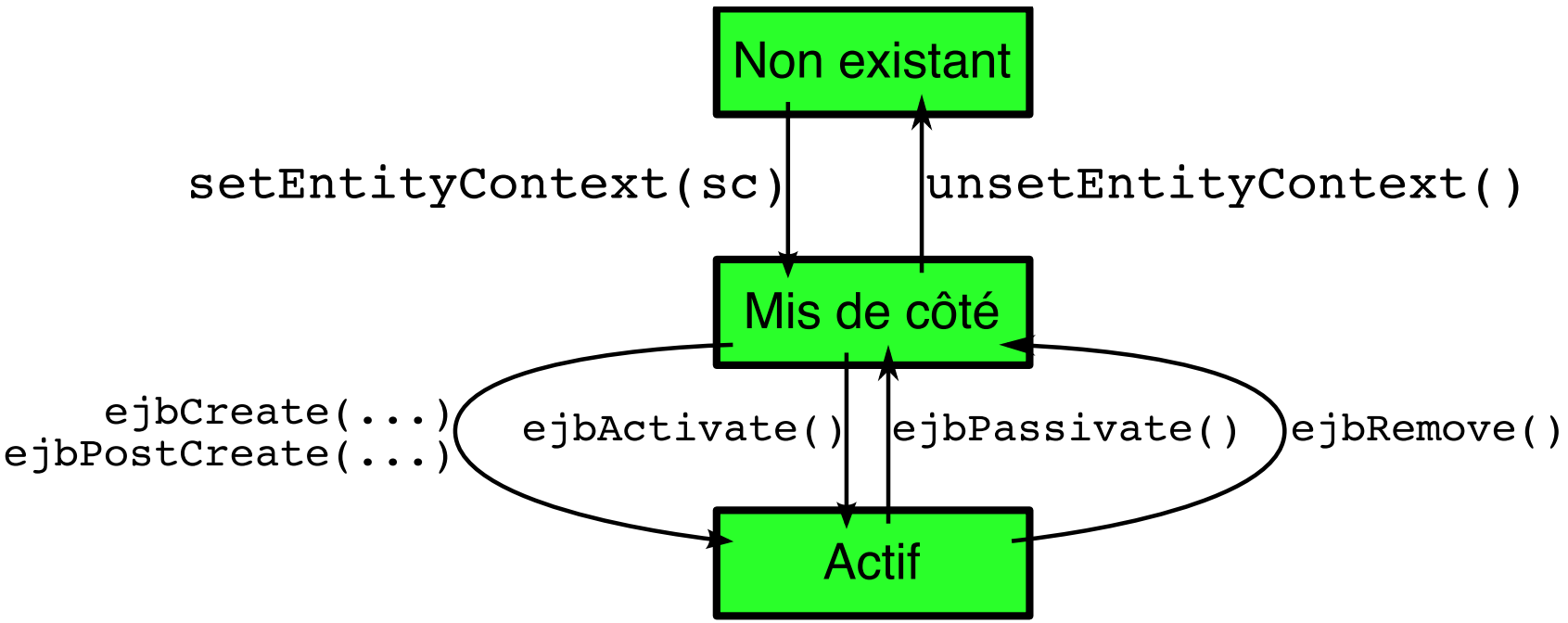
Classe du bean

Implante les fonctionnalités décrites dans les interfaces
Exemple :

```
package biblio;
public class AdminBean
    implements javax.ejb.SessionBean {
  public Date emprunte(String nom, String cote) {
    // implantation de l'emprunt
  }

  public void ejbCreate() {
    // code appelé lors de la création du bean
  }
  ...
}
```

Cycle de vie d'un bean entité



Interface

`javax.ejb.EntityBean` :

```
public void ejbCreate(...);
public void ejbPostCreate(...);
public void ejbRemove();
public void ejbActivate();
public void ejbPassivate();
public void setEntityContext(SessionContext sc);
public void unsetEntityContext();
```

Donnée

Un bean entité représente une donnée persistante présente dans une base de données

Exemple : livre

Champ	cote	titre	retour	emprunteur
Type	String	String	Date	Utilisateur
Remarque	Clé primaire			Clé extérieure

Interface distante

On y définit les méthodes pour accéder aux champs
Exemple :

```
package biblio;
public class Livre extends javax.ejb.EJBObject {

  public String getCote()
    throws java.rmi.RemoteException;

  public void setCote(String cote)
    throws java.rmi.RemoteException;

  ...
}
```

Interface maison

On y définit aussi des méthodes de recherche
Exemple :

```
package biblio;
public class LivreHome extends javax.ejb.EJBHome {

  public Livre create(String titre)
    throws java.rmi.RemoteException,
           javax.ejb.CreateException;

  public Collection findByTitle(String titre)
    throws java.rmi.RemoteException,
           javax.ejb.FinderException;
  ...
}
```

Gestion de la persistance

Il existe ensuite deux possibilités suivant la façon dont on veut gérer la persistance :

- gestion par le bean : le lien avec la base de données est codé dans la classe du bean
- gestion par le conteneur : seules des fonctions abstraites sont définies dans la classe du bean, le lien avec la base de données est défini dans le fichier de configuration du conteneur (`META-INF/ejb-jar.xml`)

Bean gérant la persistance

Implante toutes les fonctions de création, recherche...

```
package biblio;
public class LivreBean extends javax.ejb.EntityBean {
  public void ejbCreate(String titre)
    throws CreateException {
    // code appelé lors de la création du bean
  }
  public String ejbFindByPrimaryKey(String cote)
    throws FinderException {
    // code appelé pour faire la recherche
  }
  public String getCote () { ... }
  ...
}
```

Exemple

```
public String ejbCreate(String titre)
    throws CreateException {
  String cote = Utils.genereCote(titre);
  String requeteSQL = "INSERT INTO livres VALUES ("
      + cote + ", " + titre + ", NULL, NULL)";
  try {
    Utils.effectueRequete(requeteSQL);
    return cote;
  } catch (Exception e) {
    throw new  EJBException("ejbCreate: " + e);
  }
}
```

Gestion des relations

Dans le cas de la persistance gérée par le bean, les relations entre beans sont également gérées par ces derniers

Dans notre exemple, `getEmprunteur` renverrait non pas un objet de la classe `Utilisateur`, mais une chaîne représentant son nom (login)

Gestion de la persistance par le conteneur

La classe du bean reste abstraite

```
package biblio;
public abstract class LivreBean extends javax.ejb.Entity
  public void ejbCreate(String titre)
    throws CreateException {
    // code appelé lors de la création du bean
  }

  // pas de ejbFind...

  public abstract String getCote ();
  public abstract void setCote(String cote);
  ...
}
```

Exemple

```
public String ejbCreate(String titre)
    throws CreateException {
  String cote = Utils.genereCote(titre);
  setCote(cote);
  setTitre(titre);
  return null;
}
```

Gestion des relations

Dans le cas de la persistance gérée par le conteneur, les relations entre beans sont également gérées par ce dernier

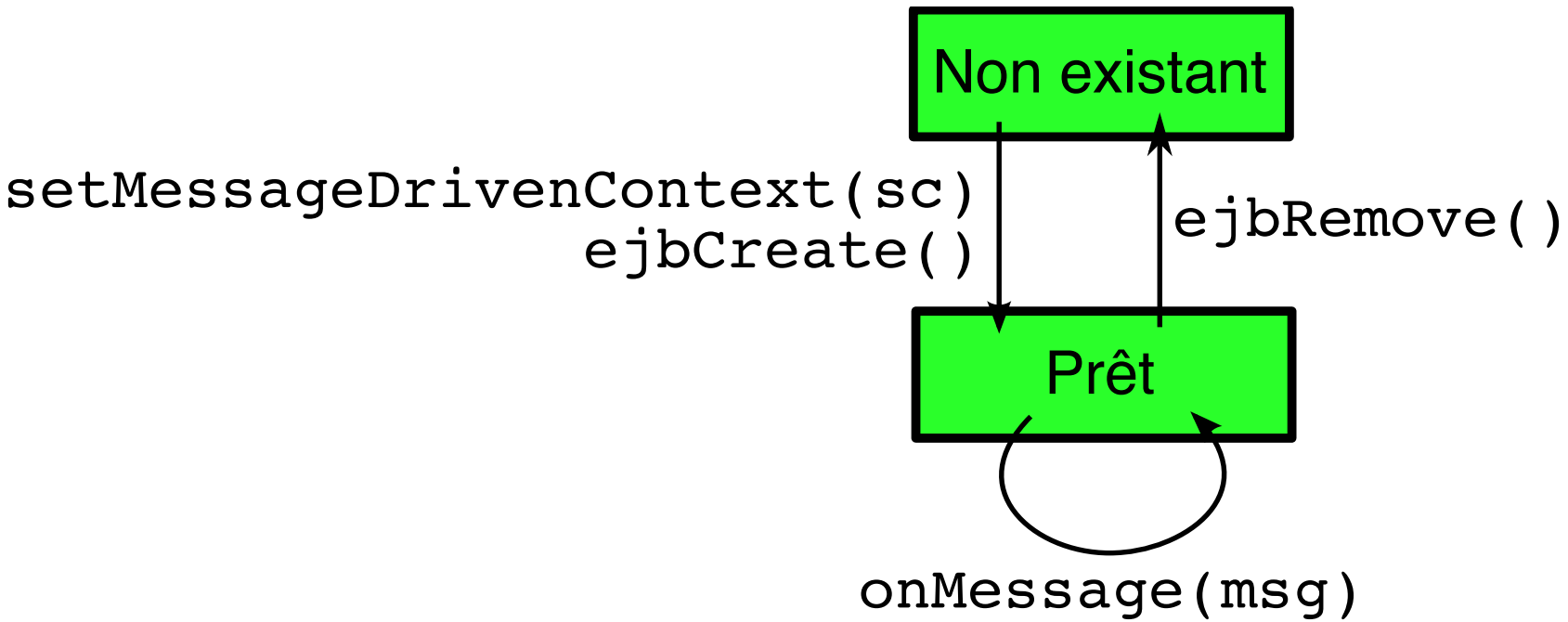
$\rightarrow$ un champ (presque) comme les autres

```
public abstract Utilisateur getEmprunteur();
public abstract void setEmprunteur(Utilisateur u);
```

Dans `UtilisateurBean` :

```
public abstract Collection getEmprunts();
public abstract void setEmprunts(Collection livres);
```

Cycle de vie d'un bean contrôlé par messages



Interface

`javax.ejb.MessageDrivenBean` :

```
public void ejbCreate();
public void ejbRemove();
public void setMessageDrivenContext(
                MessageDrivenContext sc);
public void onMessage(javax.jms.Message msg);
```

Interfaces distantes et locales

Contrairement aux beans entités et session, pas d'interfaces distante, locale, maison ni locale maison

L'accès au bean se fait exclusivement par le biais de la file
Le bean traite les messages de la file de façon asynchrone grâce à `onMessage`
Le lien entre le bean et la file est géré par le conteneur (définit dans les fichiers de configuration propres au serveur d'application utilisé)

Classe du bean

Implante le traitement du message

```
package biblio;
public class ConfirmeBean
    implements javax.ejb.MessageDrivenBean,
               java.ejb.MessageListener {
  public void onMessage(Message msg) {
    // traitement du message
  }

  public void ejbCreate() {
    // code appelé lors de la création du bean
  }
  ...
}
```

Plan

Fichier de configuration

Fichier `ejb-jar.xml` :

```
<ejb-jar id="ident" version= "2.1">
  <display-name>Nom</display-name>
  <enterprise-beans>...</enterprise-beans>
  <relation-ships>...</relation-ships>
  <assembly-descriptor>...</assembly-descriptor>
    <!-- sécurité, destination des messages, ... -->
  <ejb-client-jar>client.jar</ejb-client-jar>
    <!-- contient les classes utilisables uniquement
          par les clients -->
</ejb-jar>
```

Configuration

```
<enterprise-beans>
  <session id="admin">
     <display-name>Admin</display-name>
     <ejb-name>Admin</ejb-name>

     <home>biblio.AdminHome</home>
     <remote>biblio.Admin</remote>
     <local-home>biblio.AdminLocalHome</local-home>
     <local>biblio.AdminLocal</local>
     <ejb-class>biblio.AdminBean</ejb-class>
     <session-type>Stateless</session-type>
     <transaction-type>Container</transaction-type>
  </session>
  ...
</enterprise-beans>
```

Configuration

```
<enterprise-beans>
  ...
  <entity id="livre">
    <display-name>...</ejb-class>
    <persistence-type>Container</persistence-type>
    <prim-key-class>java.lang.String</prim-key-class>
    <cmp-field><field-name>cote</field-name></cmp-field
    <cmp-field><field-name>titre</field-name></cmp-fiel
    <cmp-field><field-name>retour</field-name></cmp-fie
    <primkey-field>cote</primkey-field>
    <query>
      <!-- voir slide suivant -->
    </query>
  </entity>
  ...
</enterprise-beans>
```

Configuration

```
<query>
  <query-method>
    <method-name>findByTitle</method-name>
      <method-params>
        <method-param>java.lang.String</method-param>
      </method-params>
    </query-method>
    <ejb-ql>
      SELECT OBJECT(a) FROM livres a
        WHERE a.titre LIKE ?1
    </ejb-ql>
</query>
```

Configuration

```
<enterprise-beans>
  ...
  <message-driven id="confirme">
    <display-name>Confirme</display-name>
    <ejb-name>Confirme</ejb-name>
    <ejb-class>biblio.ConfirmeBean</ejb-class>
    <transaction-type>Container</transaction-type>
    <messaging-type>javax.jms.MessageListener</messagin
    <message-destination-type>javax.jms.Queue</message-
    <activation-config>...</activation-config>
  </message-driven>
</enterprise-beans>
```

Configuration

```
<relationships>
  <ejb-relation>
    <ejb-relationship-role>
      <multiplicity>Many</multiplicity>
      <relationship-role-source>
        <ejb-name>Livre</ejb-name>
      </relationship-role-source>
      <cmr-field>
        <cmr-field-name>emprunteur</cmr-field-name>
        <cmr-field-type>biblio.Utilisateur</cmr-field-t
      </cmr-field>
    </ejb-relationship-role>
    <ejb-relationship-role>...</ejb-relationship-role>
  </ejb-relation>
</relationships>
```

Configuration

```
<assembly-descriptor>
  <security-role>
    <role-name>bibliothecaire</role-name>
  </security-role>
  <method-permission>
    <role-name>bibliothecaire</role-name>
    <method>
      <ejb-name>Livre</ejb-name>
      <method-intf>Home</method-intf>
      <method-name>create</method-name>
      <method-params>
        <method-param>java.lang.String</method-param>
      </method-params>
    </method>
  </method-permission>
</assembly-descriptor>
```

Utilisation des EJB

Dans les applications clientes, les EJBs sont appelés de façon distante en utilisant JNDI

```
Properties props = new Properties();
props.setProperty("java.naming.provider.url",
                  "localhost:1099");
props.setProperty("java.naming.factory.url.pkgs",
                  "org.jboss.naming");
InitialContext jndi = new InitialContext(props);
Object ref = jndi.lookup("ejb/Admin");
AdminHome adminHome = (AdminHome)
  PortableRemoteObject.narrow(ref, AdminHome.class);
Admin session = adminHome.create();
session.emprunte("Toto", "A678GD90");
```

Utilisation du bean contrôlé par messages

```
Queue q = (Queue) jndi.lookup("queue/Confirme");
QueueConnectionFactory qcf = (QueueConnectionFactory)
    jndi.lookup("java:/JmsXA");
QueueConnection qc = qcf.createQueueConnection();
QueueSession qs = qc.createQueueSession(false,
                        Session.AUTO_ACKNOWLEDGE);
QueueSender envoyeur = qs.createSender(q);
TextMessage msg = qs.createTextMessage("Bonjour");
envoyeur.send(msg);
qs.close();
```

Plan

- Introduction
- Entreprise Java Beans
  - Bean session
  - Bean entité
  - Bean contrôlé par messages
- Configuration
- XDoclet

Bureaucratie

Nombreux fichiers à écrire :

- Pour chaque EJB :
  - Interface distante
  - Interface maison
  - Interface locale
  - Interface locale maison
  - Classe du bean
- fichier de configuration `ejb-jar.xml`
- fichiers de configuration spécifiques au serveur

Inconvénients

- laborieux
- redondant
- difficile de maintenir cohérence entre interfaces et classe de bean

Solution : XDoclet

Plus qu'un seul fichier par EJB à écrire, la classe de bean

Ajout d'annotations pour définir les méthodes des interfaces, les recherches possibles, le lien avec la base de données, etc.

xdoclet génère ensuite automatiquement les interfaces, les fichiers de configuration ainsi que quelques classes utiles

```
/**
 * @ejb.bean name="Admin"
 *           display-name="Admin"
 *           jndi-name="Admin"
 *           type="Stateless"
 *           transaction-type="Container"
 */
public abstract class AdminBean
  implements javax.ejb.SessionBean {

  ...

}
```

```
/**
 * @ejb.bean name="Livre" ...
 * @ejb.persistence table-name="LIVRES"
 * @ejb.finder
 *   query="SELECT OBJECT(a) FROM LivreSCHEMA a WHERE a.t
 *   signature="java.util.Collection findByTitle(java.la
 * @ejb.pk class="java.lang.String"
 * @jboss.persistence datasource="java:/DefaultDS"
 *    table-name="LIVRES" datasource-mapping="Hypersonic
 *    create-table="true" remove-table="true" alter-tabl
 */
public abstract class LivreBean
  implements javax.ejb.EntityBean {
    ...
}
```

Dans `AdminBean.java` :

```
/**
 * @ejb.create-method view-type="remote"
 */
public void ejbCreate() { ... }

/**
 * @ejb.interface-method view-type="both"
 */
public Date emprunte(String nom, String cote) {
   ...
}
```

Dans `LivreBean.java` :

```
/**
 * @ejb.persistent-field
 * @ejb.persistence
 *    column-name="COTE"
 *    jdbc-type="VARCHAR"
 *    sql-type="VARCHAR(8)"
 *    read-only="false"
 * @ejb.pk-field
 * @ejb.interface-method  */
public abstract String getCote();
/**
 * @ejb.interface-method view-type="local"
 */
public abstract void setCote(String cote);
```

Dans `LivreBean.java` :

```
/**
 * @ejb.relation
 *      name="utilisateur-livres"
 *      role-name="livre emprunte par utilisateur"
 * @jboss.relation related-pk-field = "nom"
 *      fk-column = "emprunteur"
 *      fk-constraint = "true"
 * @ejb.interface-method
 */
public abstract Utilisateur getEmprunteur();

/**
 * @ejb.interface-method
 */
public abstract void setEmprunteur(Emprunteur u);
```

Autres classes générées par XDoclet

À partir de `ExempleBean`

- `ExempleUtil` : classe permettant de récupérer facilement les EJB avec JNDI : `getHome(props)`, `getLocalHome()`
  permet aussi de générer des identifiants uniques (par exemple cotes des livres) : `generateGUID(this)`
- pour les beans entité, `ExempleData` : classe simple contenant uniquement les données de l'entité (pas les clefs extérieures par exemple)

Implantation de la fonction d'emprunt

`Date emprunte(String nom, String cote)`

Cette méthode doit :

- chercher le livre par sa cote
- chercher l'utilisateur par son nom
- vérifier que le livre n'est pas déjà emprunté
- calculer la date de retour et le nombre maximum de livres à emprunter en fonction du statut de l'utilisateur
- vérifier que ce nombre ne sera pas dépassé
- mettre à jour le livre et l'utilisateur
- envoyer un mail de confirmation avec la date de retour
- retourner cette date

En cas d'erreur on revoit une exception

```
/**
 * @ejb.interface-method view-type="both"
 */
public Date emprunte(String nom, String cote)
                throws Exception {
   LivreLocalHome livreHome = LivreUtil.getLocalHome();
   LivreLocal livre = livreHome.findByPrimaryKey(cote);
   if (livre.getRetour() != null)
     throw new Exception("Le livre est déjà emprunté.");

  UtilisateurLocalHome utilHome =
                UtilisateurUtil.getLocalHome();
   UtilisateurLocal u = utilHome.findByPrimaryKey(nom);
```

```
int mois = 0;
int maxLivres = 0;
String statut = u.getStatut();
if (statut == "etudiant")
      { mois = 1; maxLivres = 2; }
else if (statut == "doctorant")
      { mois = 3; maxLivres = 4; }
else if (statut == "enseignant")
      { mois = 6; maxLivres = 8; }
else {
  throw new Exception("Le statut "
              + statut + " est inconnu.");
};
```

```
if (u.getLivres().size() + 1 > maxLivres)
  throw new Exception("Le nombre maximum " +
      "de livres empruntés a été atteint.");

Calendar cal = new GregorianCalendar();
cal.add(Calendar.MONTH, mois);

livre.setRetour(cal.getTime());
livre.setEmprunteur(u);
```

```
    Queue q = ConfirmeUtil.getQueue();
    QueueConnection qc = ConfirmeUtil.getQueueConnection(
    QueueSession qs = qc.createQueueSession(false,
                                Session.AUTO_ACKNOWLEDGE);
    QueueSender envoyeur = qs.createSender(q);
    MapMessage msg = qs.createMapMessage();
    msg.setString("adresse mail", u.getAdresse());
    msg.setString("titre", livre.getTitre());
    msg.setString("date",
                livre.getRetour().toLocaleString());
    msg.setString("nom", u.getNom());
    envoyeur.send(msg);
    qs.close();

    return livre.getRetour();
}
```