Atelier 4-B

Développement EJB 3.0 avec Eclipse WTP et JBOSS

Cet atelier a pour objectif de démontrer la programmation des EJB 3.0. Il démontre l'utilisation conjointe des beans Entité et Session conformément au design pattern "Session Façade".

Installation des outils

- Nous utilisons le J2SDK 1.5 de Sun (Tiger). La version J2SDK 1.6 (Mustung) convient également.
- Comme IDE, la version utilisée ici est Eclipse WTP 2.0 Europa. Cette version présente l'avantage de supporter le développement des projets EJB 3.0 et la facilité de leur déploiement (sans besoin de packaging avec XDoclet comme c'était le cas avec Eclipse JBoss IDE utilisé lors du TP numéro 1).

  Eclipse WTP 2.0 Europa est constitué de la version 3.3 de Eclipse (dite Eclipse Europa) muni du plugin WTP (Web Tools Platform) dans sa version 2.0. Il est téléchargeable en un seul zip file de 230 Mo environ depuis http://www.eclipse.org/downloads/moreinfo/jee.php.
  L'installation de Eclipse WTP est extrêmement simple, il suffit de décompresser le zip vers un endroit de votre système de de fichier. Faites attention au cas où votre cible de décompression contient un ancien eclipse, car la racine de décompression de Eclipse WTP est toujours le dossier eclipse. Pour assurer une décompression saine, renommer l'ancien répertoire eclipse, ou simplement choisir un répertoire de destination différent de celui de l'ancien eclipse.
- La version du serveur JBoss utilisée ici est la version JBoss 4.2.1.GA, téléchargeable toujours depuis Sourceforge. La version 4.2.1 présente l'avantage d'être déjà pré configurée pour le support des EJB 3. Pour l'installer, il suffit simplement de la décompresser dans un répertoire de votre système de fichiers.
- Nous utilisons également le serveur de base de données MySQL serveur pour illustrer un fournisseur de persistance autre que celui fournis par défaut par JBoss.

Scénario de l'atelier

L'objectif ici est de développer un module EJB 3 contenant :

- un EJB Entité appelé **Produit** qui représente un Produit appartenant à un stock de produits. Ce bean fera ainsi l'objet d'un mappage objet/relationnel. Le conteneur EJB se chargera de cette tâche.
- un EJB de session appelé **GestionProduit** sans état qui joue le rôle de façade pour le bean Produit. Il s'agit d'un bean sans état qui offre une interface (distante) aux programmes clients leur permettant de gérer des produits (ajouter un produit, consulter un produit, …).

Toutefois, et pour tester notre module EJB une fois déployé, nous développerons un client de test consistant en un simple programme Java autonome qui appellera les méthodes de la façade fonctionnelle **GestionProduit**. Ce programme de test sera crée dans un projet Java séparé, et il ne fera naturellement pas l'objet d'un quelconque déploiement sur le serveur d'application.

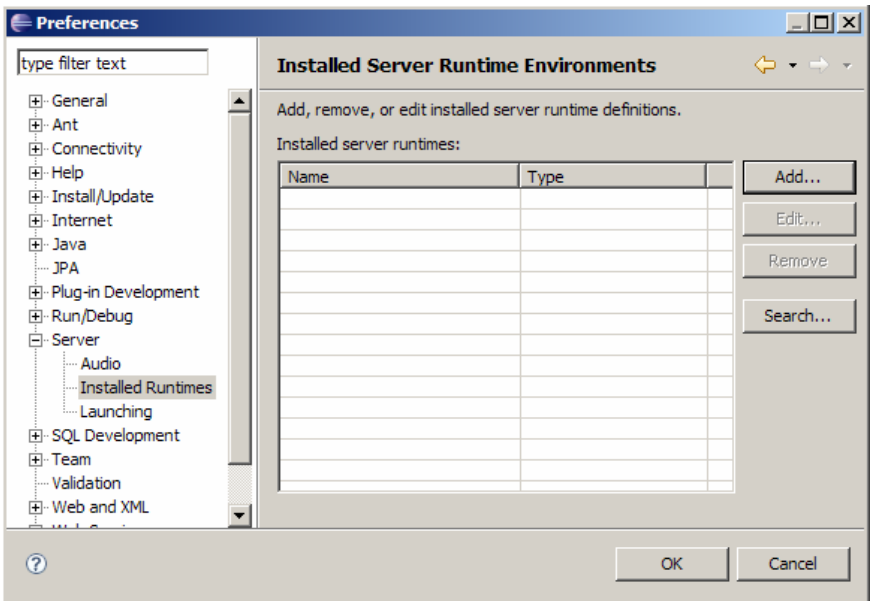
Les activités avancées du TP couvriront le sujet d'un mappage sur une base de données autre que celle fournie par défaut par JBoss et le sujet des relations managées entre les beans Entité.

Développement de l' EJB Entité "Produit"

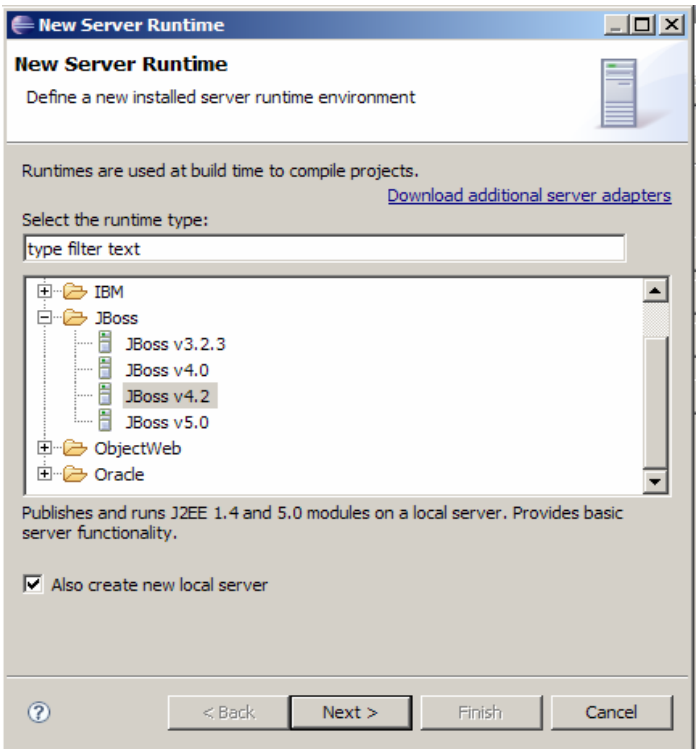
Déclaration de JBoss dans Eclipse

1. Lancez Eclipse WTP à partir du fichier **ECLIPSE_HOME\ eclipse.exe** ou éventuellement à partir d'un raccourcis que vous auriez mis sur votre bureau. Spécifiez au lancement d'Eclipse le répertoire à utiliser comme workspace.

2. Pour associer le serveur JBoss à WTP , procédez comme suit :
   - Ouvrir la page de préférences dans le menu **Window->Preferences-> Server->Installed RunTimes->Add**

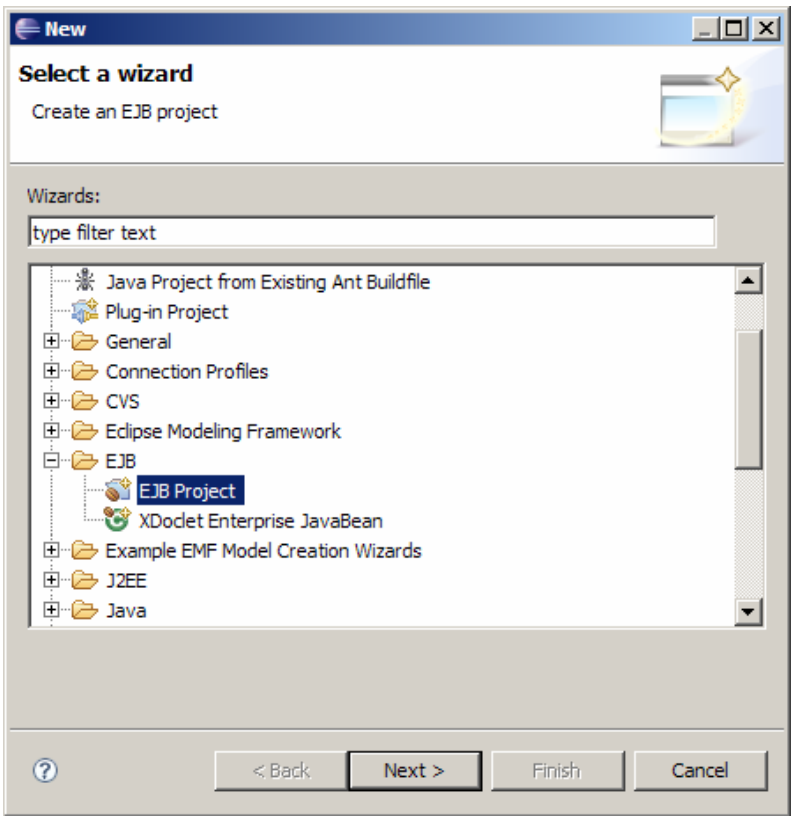


   - Dans la liste proposée, sélectionnez le type de serveur **"JBoss v4.2",** puis sélectionner la configuration **"defaut"** pour JBoss**.** Pour le JRE, il est conseillé de travailler avec le JDK 1.5 au lieu du JRE, pour cela suivre le lien "**Installed JRE Preferences"** présent sur cette boîte de dialogue et spécifier le chemin de votre JDK. Fournir le répertoire de localisation de votre JBoss 4.2.1GA puis **Next>** …

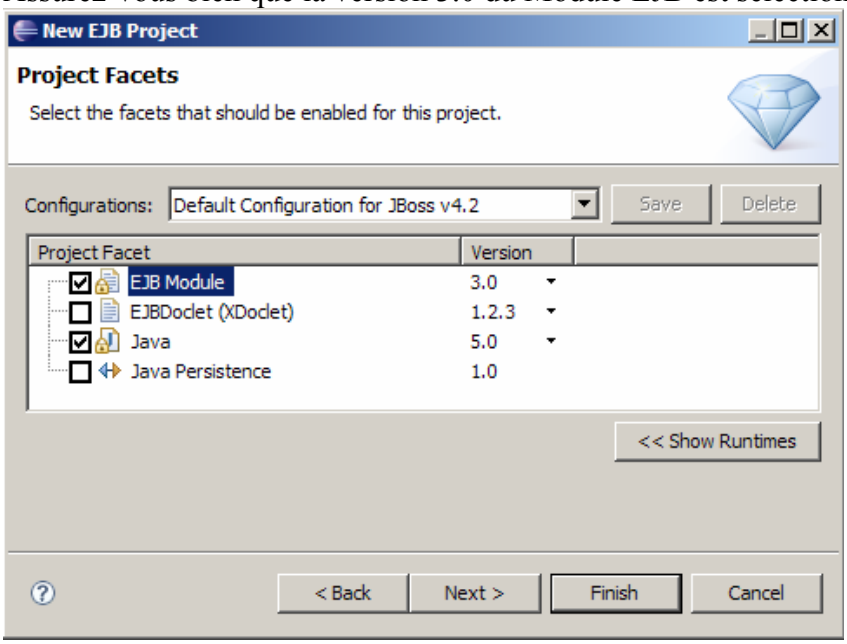


▪ Création d'un projet EJB 3

1. Créez un nouveau projet du type EJB Project comme suit : **Menu File->New->Projet->EJB Project puis Next**.



2. Nommez le projet, à titre d'exemple, **MonProjetProduits**. Sélectionnez l'environnement JBoss v4.2, précédemment crée, comme environnement d'exécution cible et gardez les autres par defaut, Faites **Next>**.
3. Assurez-vous bien que la version 3.0 du Module EJB est sélectionnée.



Faites **Next>** , vous pouvez laisser le nom proposé (**ejbModule**) pour le répertoire source (par défaut). Laissez la case "**Generate Deployment Descriptor**" décochée, en effet, nous pouvons nous passer des descripteurs de déploiement dans le cas des EJB 3.

Notre projet est maintenant correctement configuré pour construire des EJBs !

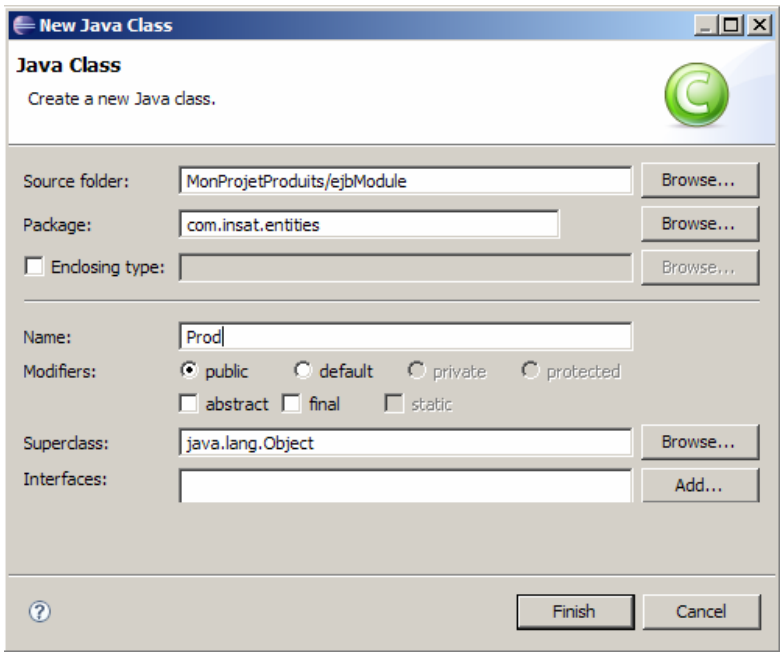
- ## Création de l'EJB Entité "Produit"

Petit rappel concernant les EJB Entité

La spécification EJB 3 revoit très largement le développement des Entity Beans. Les EJB Entity sont décrits dans une spécification complémentaire nommée JPA (Java Persistence API) dont les principaux apports sont :

1. Simplification du code via l'utilisation de l'approche "POJO" (Plain Old Java Object): un EJB Entité consiste en une classe Java.
2. Fonctionnalités de mappage objet-relationnel plus riches. La spécification JPA propose d'utiliser les annotations pour définir ce mappage. Des problématiques comme l'héritage ou l'optimisation via l'utilisation de requêtes SQL sont prises en compte. De façon générale, JPA aborde de façon complète et pragmatique le problème de persistance alors que les spécifications EJB précédentes adoptaient une approche plutôt dogmatique et faisaient l'impasse sur des fonctionnalités essentielles permettant de mieux gérer le compromis entre transparence et optimisation.
3. Utilisation possible en dehors d'un serveur d'applications J2EE : JPA peut être utilisée dans une application cliente, la présence d'un conteneur d'EJB n'est plus nécessaire. Des solutions intermédiaires sont aussi possibles : déploiement dans Tomcat d'une application à base de Servlets et JSP utilisant JPA.

1. Pointez le projet **MonProjetProduits** présent sur la "**Package Explorer**" et à partir du menu contextuel, faites "**New**"->"**Class**" pour créer la classe "POJO" du Bean entité. Nommez cette classe **Produit** et son package **com.insat.entities**.. Puis appuyez sur le bouton "**Finish**".



2. Ouvrir le fichier **Produit.java** et compléter son contenu dans l'éditeur d'Eclipse de manière à définir les attributs et les méthodes de notre.
   On annotera la classe Entity(`@Entity`) et on indiquera son attribut clé(`@Id`). Un exemple de contenu pour cette classe serait le suivant :

```
//Produit.java

package com.insat.entities;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
```

```
@Entity       //Annotation indiquant qu'il s'agisse d'une Entité
public class Produit implements Serializable {

  private static final long serialVersionUID = 1L;

   @Id          //Annotation indiquant que id est la clé de Produit
   private String id;
   private String libelle;
   private int quantiteEnStock;

   //Constructeurs
    public Produit() { super();  }
   public Produit(String id) {this.id = id;}

   public Produit(String id, String libelle, int quantiteEnStock) {
      this.id = id; this.libelle = libelle;
      this.quantiteEnStock = quantiteEnStock;
   }

   // Méthodes Setters et guetters de la classe
   public String getLibelle() { return libelle;}
   public void setLibelle(String libelle) {this.libelle = libelle;}

   public int getQuantiteEnStock() { return quantiteEnStock;}
   public void setQuantiteEnStock(int quantiteEnStock) {
      this.quantiteEnStock = quantiteEnStock;
   }

   public String getId() {return id; }

   public String toString() {       //toString() de la classe Produit
      return  "Produit n°" + id + " - " + libelle +
              " - quantité disponible : " + quantiteEnStock;
   }
}
```

3. Créez un nouveau fichier sous le répertoire **META-INF**, nommez le **persistence.xml** et ajoutez le code suivant.

```
<persistence>
   <persistence-unit name="MonEntiteEJB3">
   <jta-data-source>java:/DefaultDS</jta-data-source>
       <properties>
          <property name="hibernate.hbm2ddl.auto" value="update"/>
       </properties>
   </persistence-unit>
</persistence>
```

Ce fichier permet de configurer le **EntityManager**.

Petit rappel concernant le "EntityManager"

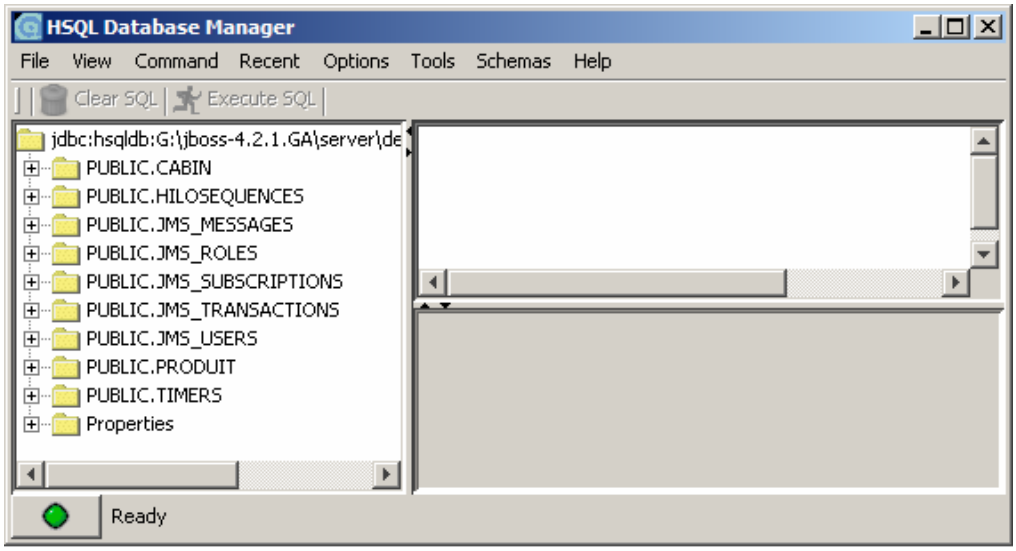
> Lors du déploiement, le serveur d'application groupe les EJB entité dans des unités de persistance. Chaque unité de persistance doit être associée à une source de données. L'EntityManager est le service qui permet de gérer ces unités. **persistence.xml** est le descripteur de déploiement qui contient cette configuration de l'EntityManager. Il est un artéfact nécessaire.

Les propriétés de bases de **persistence.xml** sont:

- la valeur **"update"** de la propriété **"hibernate.hbm2ddl.auto"** indique que nous souhaitons qu'Hibernate crée la structure de données automatiquement et la mette à jour si nécessaire
- **"MonEntiteEJB3"** est le nom que nous avons donné ici à l'unité de persistance, ce nom sera utilisé dans le bean session lors de la déclaration d'un EntityManager.
- **java:/DefaultDS** est le nom de la Data source.

Après le déploiement du bean entité, JBoss se charge de créer les tables dans son serveur de base de données intégré (HyperSonic DB) si elles n'existaient pas.

JBoss intègre également un outil permettant d'accéder à cette base de données. Cet outil peut nous permettre de lister les tables, de consulter les données qui s'y trouvent ainsi que d'exécuter diverses requêtes SQL. Pour ouvrir cet outil : accéder à la console d'administration de JBoss via l'url **http://localhost:8080/jmx-console**, dans la section nommée **'jboss'**, cliquer sur **'database=localDB,service=Hypersonic'**. Dans la page qui s'affiche, cliquer sur le bouton **'Invoke'** qui se trouve sous la signature de méthode **'void startDatabaseManager()'**. L'outil **'HSQL Database Manager'** est alors lancé. La figure ci-dessous donne un aperçu de cet outil.



Développement de l' EJB de Session " GestionProduits "

▪ Création de l'EJB de Session "GestionProduits"

Petit rappel concernant le design pattern "Session Facade"

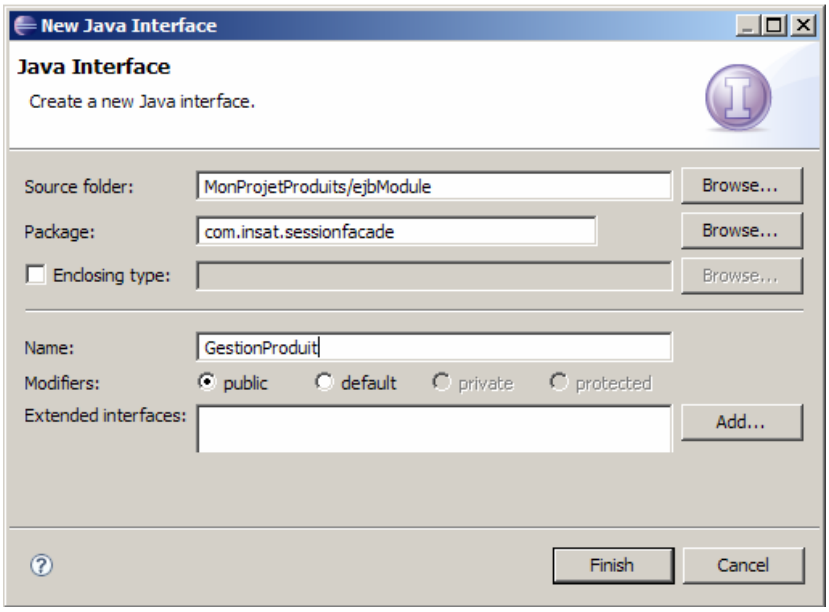
> Les bonnes pratiques de conception JEE exigent de ne pas exposer directement les méthodes d'un EJB Entité étant donné que ces méthodes permettent de modifier directement la cohérence des données stockées dans les bases de données. Ainsi les EJB Entité se limitent à exposer une interface locale et n'exposent pas d'interface distante. On recommande alors l'utilisation des EJB de type session afin d'exposer une logique saine vis-à-vis de l'utilisation des entités. Ce type d'EJB de session sont appelés "Session Facade". Ils exposent alors leurs services aux clients via une interface distante.
> Dans le cas de spécification EJB 3, les beans de session pourrant agir sur les beans entité à travers le service de gestion des unités de persistance qui est appelé EntityManager.

Pour développer l'EJB de Gestion "GestionProduits", nous avons besoin de développer son interface distante, puis la classe du bean implémentant cette interface.

Commençons par créer l'interface du service. L'interface définit les méthodes à exposer aux clients.

Programmation de l'interface du bean

1. Pointezle projet **MonProjetProduits** présent sur la "**Package Explorer**" et à partir du menu contextuel, faites "**New**"->"**Interface**". Nommer cette interface **GestionProduits** et son package **com.insat.sessionfacade**. Puis appuyer sur le bouton "**Finish**". Notez que nous voudrions créer ici un package différent de celui du bean entité. En effet, il est conceptuellement recommandé de séparer les packages des beans de sessions de celui des beans entités.



2. Ouvrir le fichier **GestionProduits.java** et compléter son contenu dans l'éditeur d'Eclipse de manière à définir les méthodes de l'interface. L'annotation `@Remote` pour spécifier au conteneur d'EJB qu'il s'agisse d'une interface distante d'EJB.

```
//GestionProduits.java
package com.insat.sessionfacade;

import com.insat.entities.*;
import java.util.List;
import javax.ejb.Remote;            //Pour l'annotation Remote

@Remote
public interface GestionProduits {
   public void ajouter(Produit produit);
   public Produit rechercherProduit(String id);
   public List<Produit> listerTousLesProduits();
}
```

Programmation de la classe du bean

1. Pointez le projet **MonProjetProduits** présent sur la "**Package Explorer**" et à partir du menu contextuel, faites "**New**"->"**Class**". Nommezcette interface **GestionProduitsBean** et son package **com.insat.sessionfacade**. Puis appuyez sur le bouton "**Finish**".

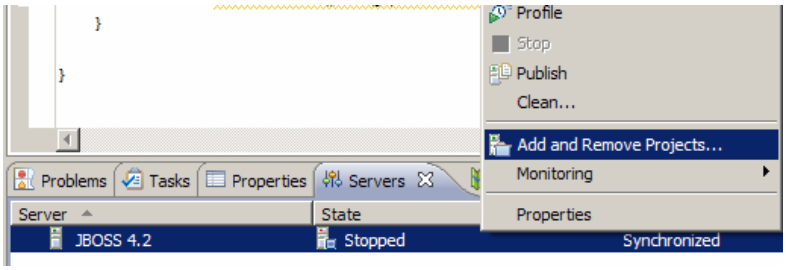
2. Ouvrir le fichier **GestionProduitsBean.java** et compléter son contenu dans l'éditeur d'Eclipse de manière à définir les méthodes de l'interface. L'annotation `@Remote` pour spécifier au conteneur d'EJB qu'il s'agisse d'une interface distante d'EJB.

```
// GestionProduitsBean.java
package com.insat.sessionfacade;

import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import com.insat.entities.Produit;

@Stateless
public class GestionProduitsBean implements GestionProduits {

   @PersistenceContext(name="MonEntiteEJB3")
   Private EntityManager em;        //L'Entity Manager

   public void ajouter(Produit produit) {
      em.persist(produit);
   }

   public Produit rechercherProduit(String id) {
      return em.find(Produit.class, id);
   }

   public List<Produit> listerTousLesProduits() {
      return em.createQuery("SELECT p FROM Produit p ORDER
                             BY p.quantiteEnStock").getResultList();
   }

}
```

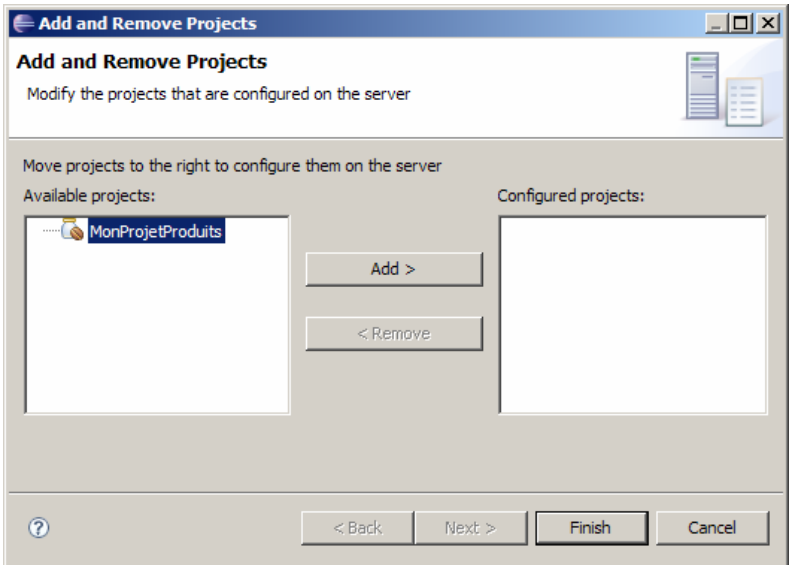
Déploiement du module EJB " MonProjetProduits "

Le déploiement du module EJB est extrêmement simple. Il suffit, d'ajouter le projet à l'instance du serveur. Eclipse se chargera alors de créer le fichier jar et de le place dans le répertoire de déploiement, à savoir, **%JBossHome%\default\deploy.**

1. Pour ajouter le projet au serveur, allez à la vue "**Servers**", sélectionnez l'instance du serveur JBoss 4.2, puis à travers le menu contextuel sélectionnez l'option "**Add and Remove Projets**". La vue "**Servers**" doit être visible quand le workbench d'Eclipse WTP est dans la perspective **Java JEE**. Eclipse WTP aurait déjà basculé vers cette perspective depuis que nous avions crée le projet EJB.

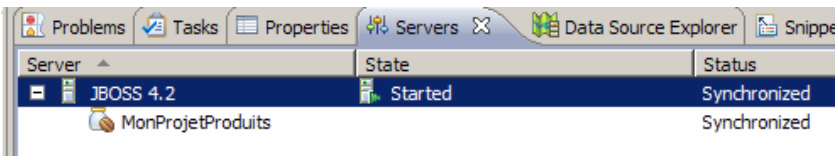


2. Ajoutez le projet en le sélectionnant et en appuyant sur le bouton "**Add**".

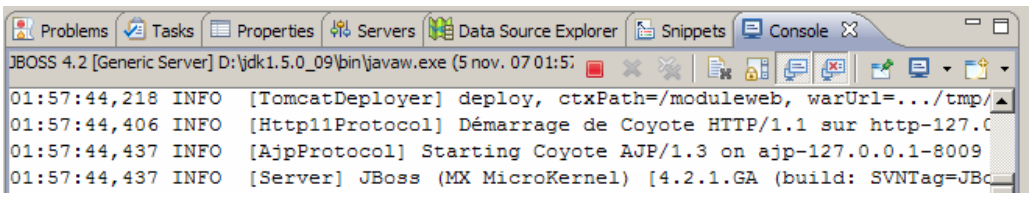


3. Démarrez le serveur en le sélectionnant de la vue "Servers" et en choisissant l'option "**Start**" du menu contextuel.

   L'ajout ou le retrait du projet peut se faire également à chaud, c'est-à-dire quand le serveur soit déjà démarré.



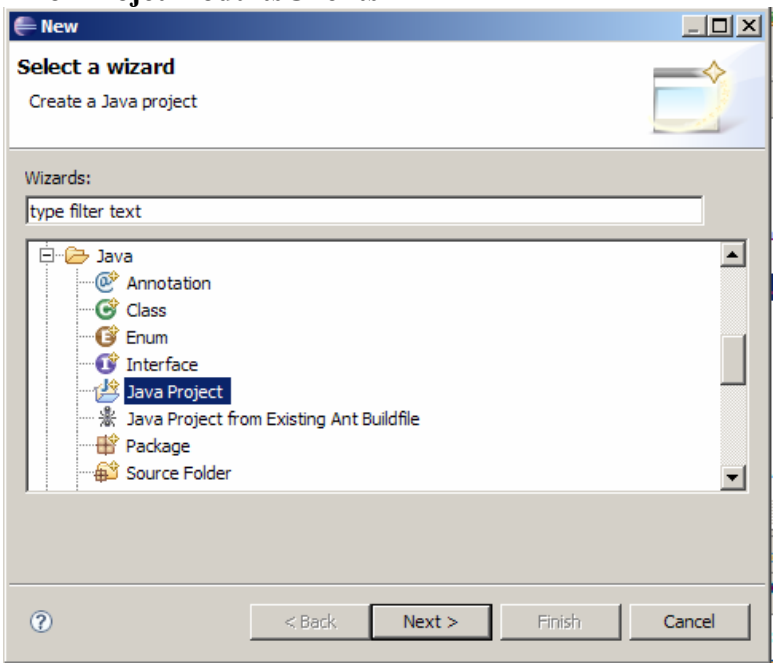
Les messages du serveur sont accessibles à partir de la vue Console.



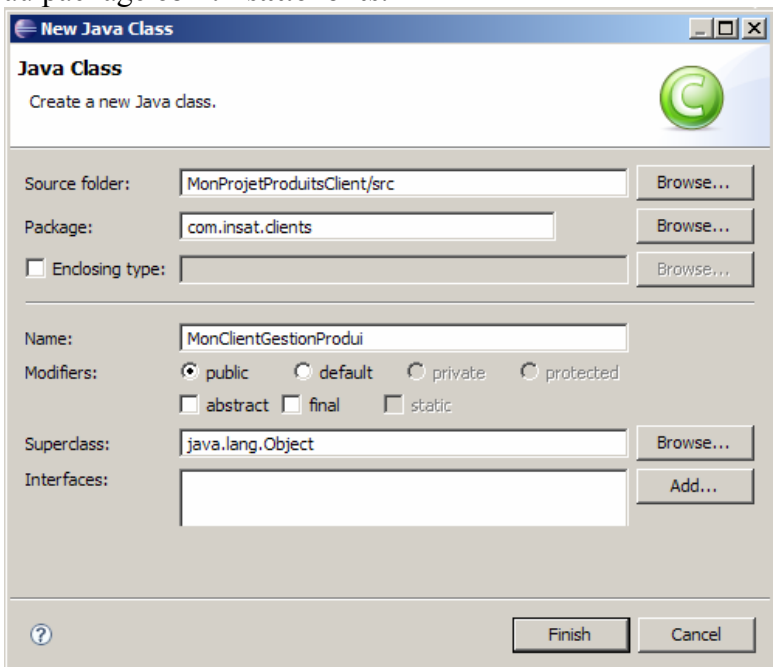
Développement du programme du Client

- ## Création du projet Java du Client "MonProjetProduitsClients"

1. Créez un nouveau projet du type "**Java Project**" comme suit : **Menu File->New->Project…->Java->Java Project puis Next**. Nommer ce projet "**MonProjetProduitsClients**"



2. Ajoutez au projet la classe Java du client. Cette classe contiendra un main. Nous la la nommons par exemple **MonClientGestionProduits.java** et nous l'assignons par exemple au package **com.insat.clients**.



- ## Rajout du descripteur "jndi.properties" au projet du Client

Le fichier **jndi.properties** permet au client d'accéder aux paramètres de l'annuaire JNDI de JBoss. Ajouter un fichier sous le répertoire src. Pour ce faire, sélectionnez d'abord le

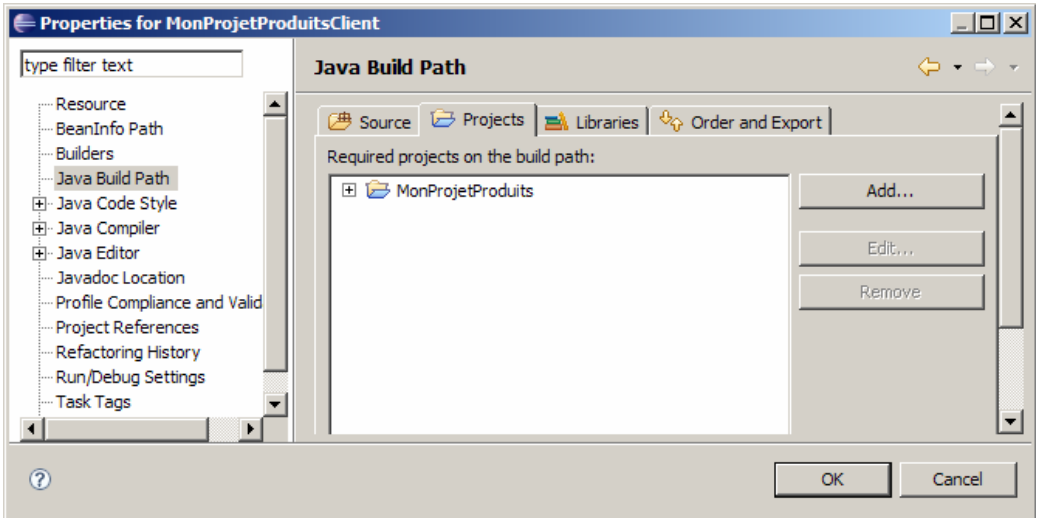
répertoire "**src**" du projet du client, puis faites **Add->New->Other->General->File.** Ajouter les lignes suivantes à ce fichier :

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming.client
java.naming.provider.url=jnp://localhost:1099
```

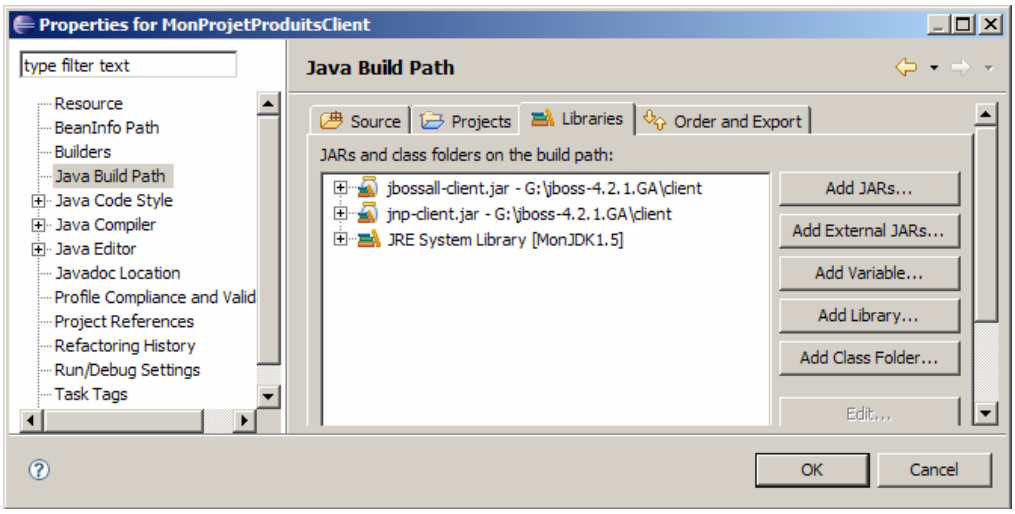
- ## Configuration du "Java Build Path" du projet Client

Pour que le client puisse communiquer avec le module EJB déployé sur le JBoss, il est nécessaire d'ajouter un ensemble de librairies utiles à ce dialogue. Pour réaliser cette configuration, pointez le projet du client, puis faites apparaître le menu contextuel et sélectionnez l'entrée "**Properties**". Cliquez alors sur l'entrée "**Java Build Path**" afin de configurer le Build Path.

1. Ajoutez la référence au projet EJB "**MonProjetProduits**", en appuyant sur l'onglet "**Projets**" et en cliquant ensuite sur le bouton "**Add**". Cette configuration permet au programme client de se compiler et d'utiliser lors de l'exécution les proxies des objets EJB distants.



2. Ajoutez les références aux deux librairies **jboss-client.jar** et **jnp-client.jar** en appuyant sur l'onglet "**Libraries**" et en cliquant ensuite sur le bouton "**Add External JARs**". Ces deux archives jar se trouvent dans le sous-répertoire nommé "**client**" du repertoire d'installation de votre serveur JBoos 4.2.1.



- **Ecriture du code du programme Client "MonClientGestionProduits.java"**

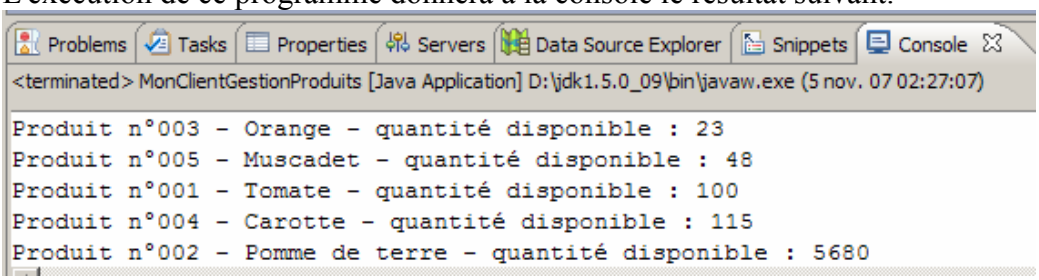
  Complétez maintenant le code du programme du client **MonClientGestionProduits.java**. Un Exemple de code est donné ci-dessous. La logique développée permet au client de créer quelques produits (des EJB Entité) puis de lister tous les produits créés. Les EJB entité sont utilisés à travers la façade **GestionProduits** qui n'est autre qu'un EJB de session offrant une interface distante.

  Attention à ne pas faire l'ajout plusieurs fois des mêmes produits à cause de la propriété d'unicité des clés.

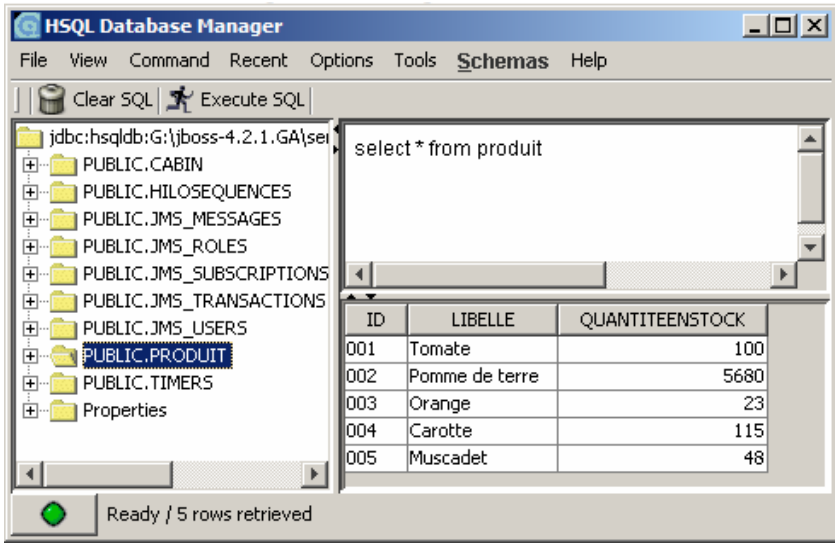
```
package com.insat.clients;

import java.util.Iterator;
import java.util.List;

import com.insat.sessionfacade.*;
import com.insat.entities.*;

import javax.naming.*;

public class MonClientGestionProduits {
  public static void main(String[] args) {
     try {
       Context context = new InitialContext();
       GestionProduits stock = (GestionProduits)
                       context.lookup("GestionProduitsBean/remote");

       // Ne pas faire l'ajout plusieurs fois, commenter ces lignes
       // après la première exécution.
       stock.ajouter(new Produit("001", "Tomate", 100));
       stock.ajouter(new Produit("002", "Pomme de terre", 5680));
       stock.ajouter(new Produit("003", "Orange", 23));
       stock.ajouter(new Produit("004", "Carotte", 115));
       stock.ajouter(new Produit("005", "Muscadet", 48));

       List<Produit> produits = stock.listerTousLesProduits();
       for (Iterator<Produit> iter = produits.iterator();
                                                  iter.hasNext();) {
         Produit eachProduit = (Produit) iter.next();
         System.out.println(eachProduit);   //Invocation de toString()
       }
     } catch (javax.naming.NamingException ne) {
       ne.printStackTrace();
     }
  }
}
```

Une fois que les éventuelles erreurs de compilation résolues, lancer ce programme avec le **Menu Run->Run As->Java Application**. Assurez-vous avant de lancer ce client que le serveur est en état de service et que le module EJB ait été déjà déployé dessous avec succès.

L'exécution de ce programme donnera à la console le résultat suivant.



Il est possible également de vérifier à travers la console JMX que JBoss a créé dans sa base de données intégrée HyperSonic une table appelée **Produit**. Cette table contient le résultat de persistance des instances EJB Entité que nous venons de créer.



Modification de la source de persistance JPA

L'objectif est ici est d'utiliser plutôt le serveur de base de données MySQL au lieu de HyperSonicDB. Procéder ainsi :

1. Placer le fichier mysql-connector-java-5.0.3-bin.jar (voir le répertoire Outils) dans \jboss-4.2.1.GA\server\default\lib
2. Récupérer le fichier de configuration exemple mysql-ds.xml situé dans C:\jboss-4.2.1.GA\docs\examples\jca directory
3. Copier le ficher mysql-ds.xml dans C:\jboss-4.2.1.GA\server\default\deploy.
4. Reconfigurer le fichier selon le model suivant :
   - <connection-url>jdbc:mysql://<hostName>:3306/<databaseName></connection-url>
   - <user-name><username></user-name>
   - <password><password></password>

Dans notre cas, la configuration est la suivante :

```
<jndi-name>MySqlDS</jndi-name>
<connection-url>jdbc:mysql://localhost:3306/testdb</connection-url>
<driver-class>com.mysql.jdbc.Driver</driver-class>
<user-name>root</user-name>
<password> </password>
```

Modifier maintenant de derscipteur persistence.xml qui se trouve sous le répertoire META-INF afin qu'il pointe la source de données MySQL:

```
<persistence>
   <persistence-unit name="titan">
   <!--    décommenter cette ligne pour utiliser la base de données locale de JBoss
           dans ce cas, si les tables n'existent pas elles serons créés pas JBoss
      <jta-data-source>java:/DefaultDS</jta-data-source>-->
      <!-- pour utiliser MySqlDS, il faut créer la base d'abord!!
      pui il faut la déclarer dans JBoss_Home\server\default\deploy\mysql-ds.xml   -->
    <jta-data-source>java:/MySqlDS</jta-data-source>
       <properties>
          <property name="hibernate.hbm2ddl.auto" value="update"/>
       </properties>
   </persistence-unit>
</persistence>
```