Systèmes d'information répartis
TD2 initiation aux servlets et EJB

1. Création d'une servlet connectée à une BD

1.1. Installer WAMPSERVER

Pour simplifier votre travail de gestion de la base de données, nous n'allons pas installer MySQL de manière isolée mais dans une distribution avec le serveur web Apache et l'application Web très pratique phpMyAdmin.

- Avant tout, wamp a besoin d'une dll pour fonctionner. Installez vcredist_x64.exe que vous trouverez dans l'archive
- Récupérez Wamp Server dans l'archive
- Installez Wamp Server dans C:\Serveur

Remarque : Si vous avez EasyPhp installé sur votre portable, n'installez rien et utilisez EasyPhp

1.2. Installer le jdbc

Le Java Data Base Connectivity (JDBC) pour MySQL s'appelle Connector/J.

- Telechargez mysql-connector-java-gpl-5.1.40.zip a l'adresse https://downloads.mysql.com/archives/c-j/
- Copier C:\Program Files (x86)\MySQL\MySQL Connector J\mysql-connector-java-5.1.31-bin.jar dans le répertoire C:\Serveur\apache-tomcat-8.0.9\lib. Cela permettra à vos servlets d'utiliser les classes du package pour se connecter à MySQL

1.3. Création d'une table 'personne' et ajout d'enregistrements

- Démarrez WampServer
- Démarrez PhpMyAdmin
- Allez dans la base **test** (créez-la si nécessaire)
- Créez une table **personne** (cle, nom, prenom)
    - cle clé primaire de type int
    - nom varchar
    - prenom varchar
- Ajoutez quelques enregistrements

1.4. Création du pool de connexion

Il y a deux manières d'accéder à une base de données. Soit chaque application J2EE crée sa connexion soit il existe un gestionnaire de connexions qui crée et gère les connexions et les files d'attente pour l'ensemble des applications. Il est évident que seule cette deuxième solution permet d'optimiser la charge sur le SGBD. Cette solution consiste à mettre en œuvre un pool de connexion.
Pour que cela fonctionne, chaque application qui a besoin d'une connexion doit pouvoir la demander au pool. A cet effet, J2EE permet de définir une ressource de manière globale au serveur par un nom qui sera ensuite réutilisable par n'importe quelle application J2EE. C'est le JNDI (Java Naming and Directory Interface).
Pour déclarer une ressource de niveau global au serveur, **il suffit d'éditer avec Notepad++ et de rajouter une balise <Resource />dans le fichier C:\Serveur\apache-tomcat-8.0.9\conf\serveur.xml à l'intérieur de la balise existante GlobalNamingResources** de la façon suivante :

```
<GlobalNamingResources>
…
      <!-- JNDI de la base test -->
      <Resource name="base_test"
            type="javax.sql.DataSource"
            password=""
            driverClassName="com.mysql.jdbc.Driver"
            maxIdle="2"
            maxWait="500"
            username="root"
            url="jdbc:mysql://localhost/test?autoReconnect=true"
            maxActive="500"
      />
```

```
…
</GlobalNamingResources>
```

Le nom visible de la base de données test est maintenant base_test.

Remarque : N'éliminez pas la datasource qui est déjà présente !!

1.5. Lien entre la ressource JNDI et l'application

La deuxième étape consiste à faire la mappage entre notre application web et la ressource JNDI. Pour cela, nous allons enrichir le contexte de notre application. Modifiez le fichier conf\catalina\localhost\ProjetServlet.xml de la façon suivante :

```
<Context path="/ProjetServlet"
      reloadable="true"
      docBase="c:\eclipseworkspace\ProjetServlet"
      workDir="c:\eclipseworkspace\ProjetServlet\work" >
      <ResourceLink
            name="base_test"
            global="base_test"
            type="javax.sql.DataSource"/>
</Context>
```

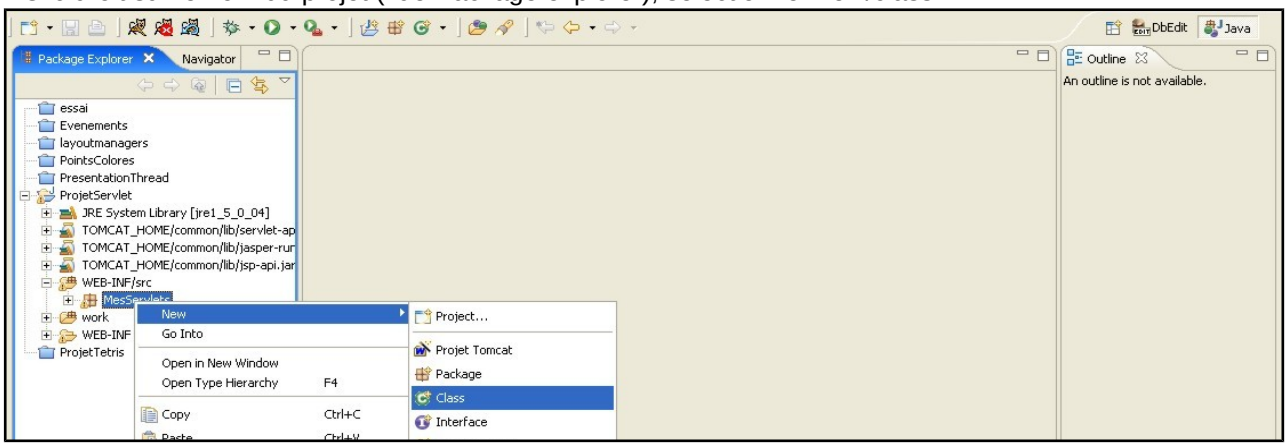
Ce fichier me permet de faire le lien entre la ressource utilisée dans mon fichier et le nom global. Il est alors par exemple, extrêmement simple de modifier une application web si la base de données change de nom…

1.6. Création de la servlet

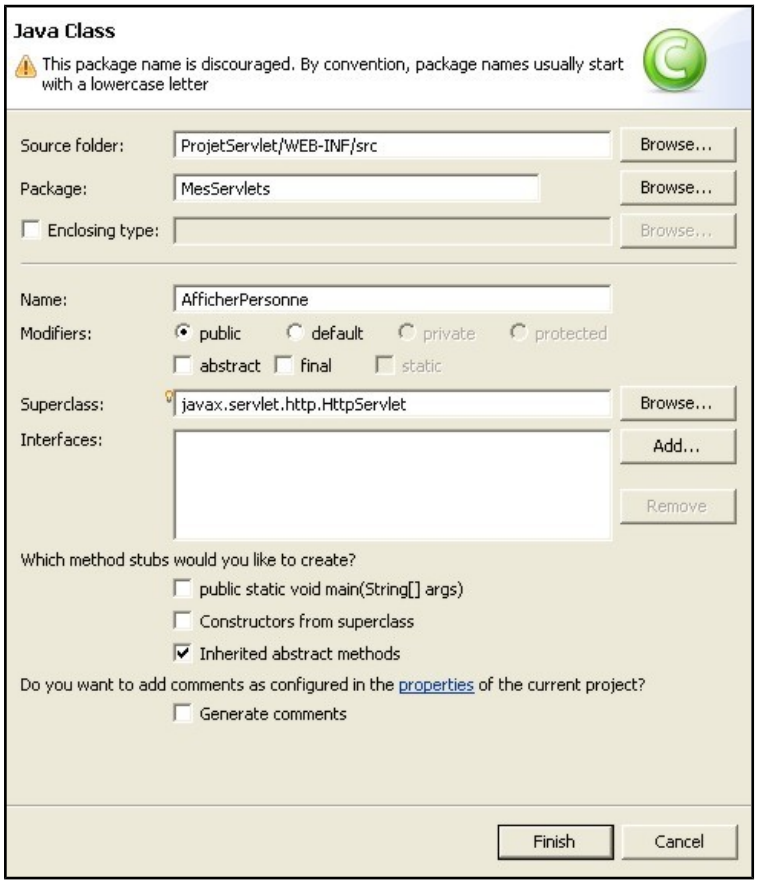
Nous allons créer une servlet toute simple qui récupère les enregistrements dans la table personne et les renvoie sous forme de texte. Nous pourrons ainsi visualiser le résultat dans un navigateur web.

Nous restons toujours dans le même projet. Repasser dans la perspective java.

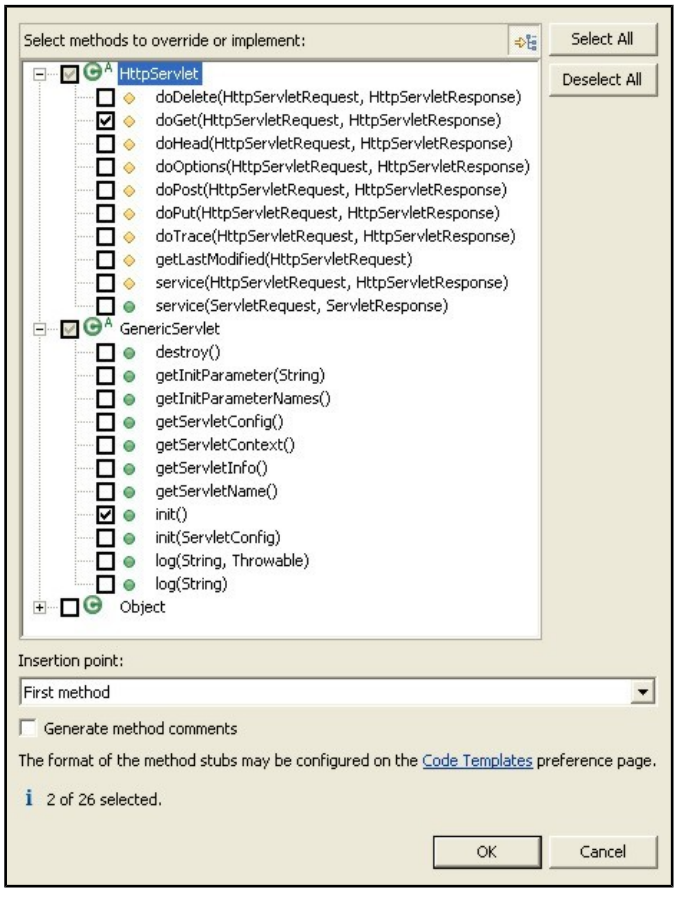
- Clic droit sur le nom du projet (vue Package explorer), sélectionner new/class



- Saisir les infos correspondant à votre nouvelle classe. Vous allez créer un package dans lequel vous allez stocker toutes vos servlets. Ici, MesServlets. Attention de bien définir la superclasse.



- Nous allons maintenant ajouter automatiquement les en-têtes
  - Clic droit sur la classe (cadre de gauche), sélectionner Source/override implement methods. Sélectionner doGet et init



Avant de pouvoir connecter la servlet à la base de données, nous devons tout d'abord créer le pool de connexion. Ce pool nous permettra notamment de maîtriser le nombre de connexions simultanées autorisées.

1.7. Connexion à la base

Une servlet est chargée soit au lancement du serveur Tomcat soit lors de sa première utilisation et reste chargée en mémoire. Dans ces deux cas, il suffit de se récupérer une connexion à la base dans le pool une fois lors de ce chargement. Pour cela, nous allons utiliser la méthode init. Une fois la connexion obtenue, tous les appels à la servlet utiliseront cette connexion.
La méthode init() est lancée une fois lors de l'activation de la servlet.
Dans la méthode init, nous allons récupérer un pointeur sur la ressource JNDI (la base test) puis nous connecter à cette ressource. Pour cela, nous allons charger le contexte puis récupérer un pointeur sur la ressource base_test.

- Ajouter deux attributs dans la classe pour le stockage de la connexion :

```
Connection BD;
DataSource ds;
```

- Ecrire le corps suivant pour la méthode init() :

```
public void init() throws ServletException     {
   try {
   System.out.println("Récupération du contexte");
   Context initCtx = new InitialContext();
   System.out.println("lookup de env");
   Context            envCtx            =            (Context)
initCtx.lookup("java:comp/env");
   System.out.println("lookup de base_test");
   ds=(DataSource) envCtx.lookup("base_test");
   //System.out.println("Datasource chargée");
   }
   catch(Exception er) {
      System.out.println("Erreur de chargement du contexte " +
er);
   }  }
```

- **Ajouter les imports nécessaires avec l'option organize imports. Pour info, la classe Connection fait partie de java.sql.**
- Vous pouvez remarquer que l'on ne se connecte pas à la base comme vous avez pu le faire en php mais que l'on récupère une connexion existante. La connexion est gérée par le pool au niveau du serveur qui attribue aux demandeurs les connexions disponibles.

1.8. Affichage du contenu d'une table

Ecrivons maintenant le contenu de la méthode doGet()

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
  throws ServletException, IOException {
    try {
    BD=ds.getConnection();
    Statement s = BD.createStatement();
    ResultSet r = s.executeQuery("select * from personne");
    PrintWriter out=null;
    resp.setContentType("text/html");
    out = resp.getWriter();
    out.println("<html>");
    out.println("<head><title> Test servlet </title></head>");
    out.println("<body>");
    out.println("Contenu de la table personne <BR>");
    out.println("<table>");
    out.println("<TR>");
    out.println("<TD>Nom</TD>");
    out.println("<TD>Prénom</TD>");
    out.println("</TR>");
    while (r.next()) {
      out.println("<TR>");
      out.println("<TD>");
      out.println(r.getString("nom"));
      out.println("</TD>");
      out.println("<TD>");
      out.println(r.getString("prenom"));
      out.println("</TD>");
      out.println("</TR>");
      }
    out.println("</table>");
    out.println("</body>");
    out.println("</html>");
```

```
      r.close();
      s.close();
      BD.close();
      s = null;
      r = null;
    } catch (java.sql.SQLException ex) {
      System.out.println("Erreur d'exécution de la requête SQL \n"+ex);
    }
}
```

Vous pouvez voir ici deux classes très importantes pour la consultation de bases de données : Statement et RecordSet. **Statement** vous permet de manipuler des tables et d'exécuter vos requêtes. **RecordSet** contient le jeu d'enregistrement résultat de la requête.

1.9. Mappage de la servlet

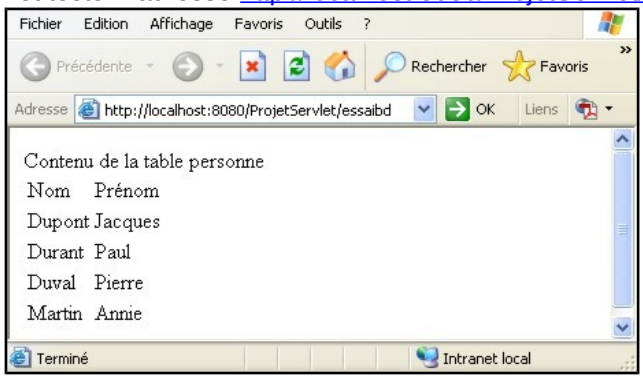
Tout comme pour notre première servlet, nous devons mapper notre servlet pour qu'elle soit reconnue par le serveur Tomcat. Vous devez rajouter les lignes suivantes dans le fichier web.xml dans le répertoire WEB-INF :

```
<servlet>
  <servlet-name>ServletBD</servlet-name>
  <servlet-class>MesServlets.AfficherPersonne</servlet-class>
  <description>Servlet d'essai de la connection BD</description>
</servlet>

<servlet-mapping>
  <servlet-name>ServletBD</servlet-name>
  <url-pattern>/essaibd</url-pattern>
</servlet-mapping>
```

1.10. Exécution de la servlet

- Lancer Tomcat (il faut le démarrer ou le redémarrer pour la prise en compte du pool de connexion).
- Démarrer votre navigateur et testez l'adresse http://localhost:8080/ProjetServlet/essaibd



Initiation aux EJB3 avec Eclipse
Ecriture d'un ejb session remote stateless

---

1 Installation de la plateforme

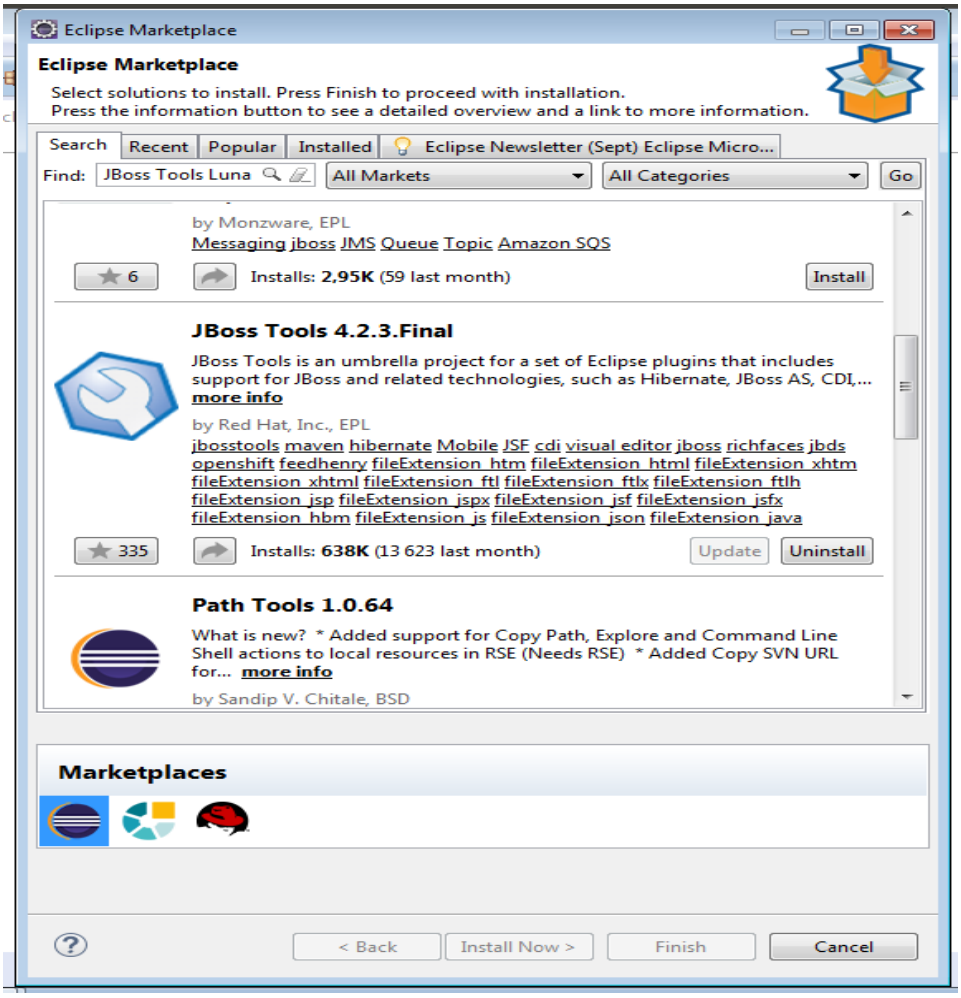
Nous réutilisons le JDK et Eclipse que nous avons installé au TD1.

1.1 Installation de JBOSS

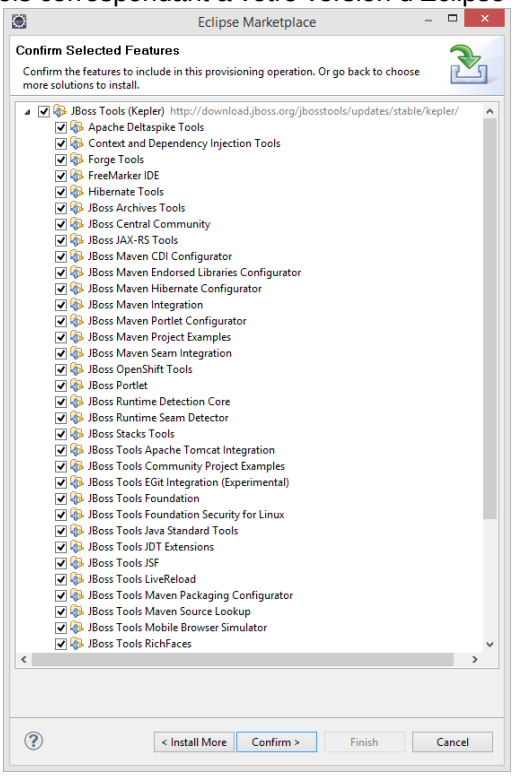
Vous pouvez télécharger Jboss/WildFly à l'adresse http://wildfly.org/downloads/ et decompressez l'archive dans C:\Serveurs. **La version utilisee dans ce TD est 10.1.0.Final.**

1.2 Installation de JBOSS tools

- Démarrer Eclipse
- Menu Help>Eclipse Marketplace
- Cherchez JBoss Tools pour votre version d'Eclipse (ici Luna)



- Installez la version de Jboss tools correspondant à votre version d'Eclipse en choisissant tous les composants

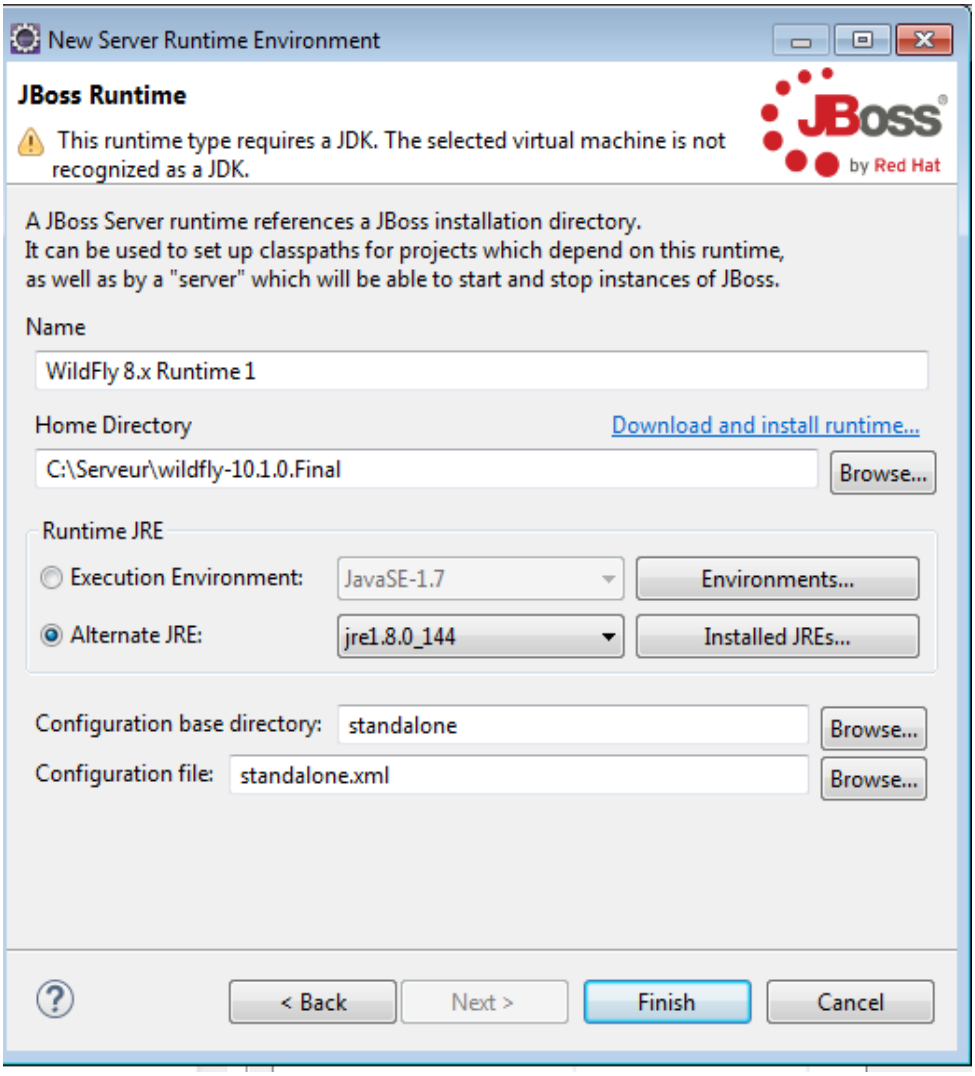


- Cliquez sur Next, acceptez la licence et Finish. Attendez que l'installation se déroule…
- Redémarrer Eclipse

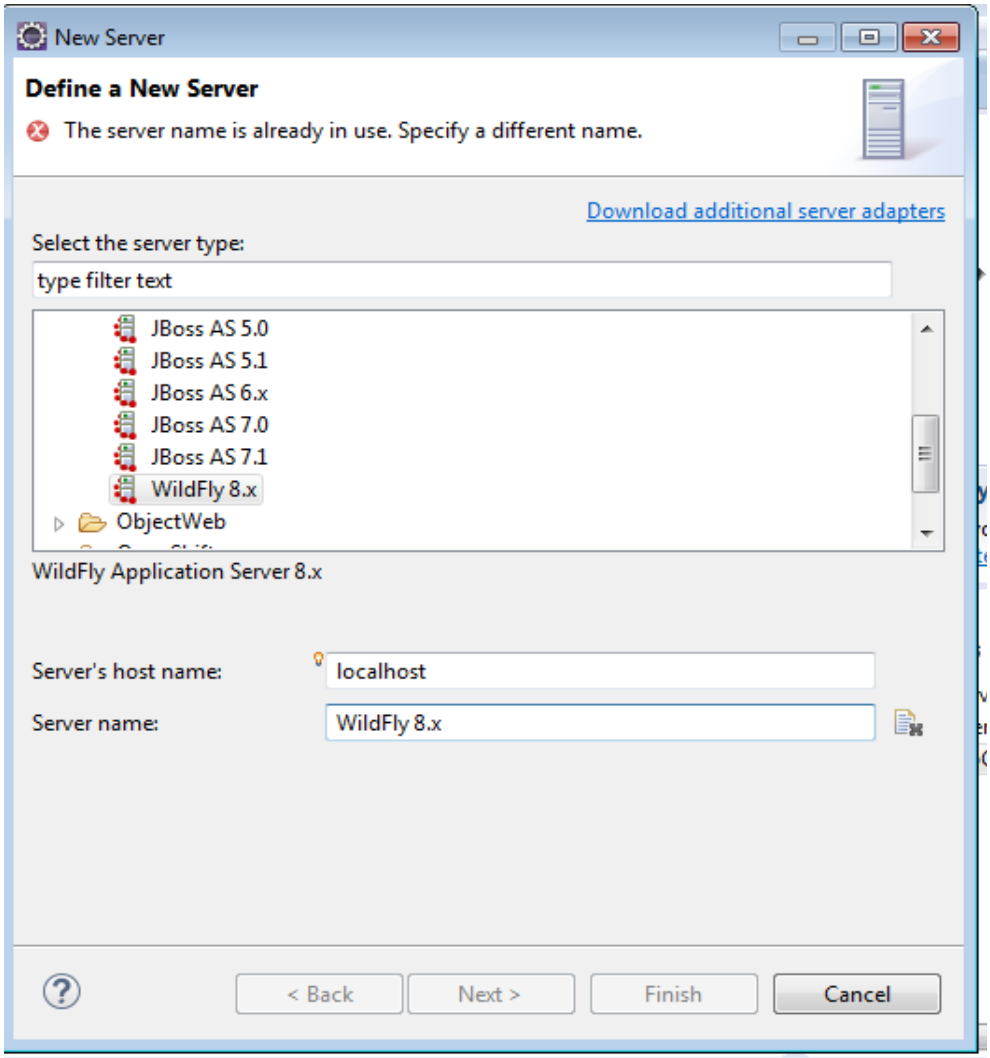
1.3 Déclaration du serveur JBOSS dans ECLIPSE

- Ouvrir la page de préférences **Preferences->Server->Runtime Environments**

- Dans la liste proposée par le bouton '**Add...**',
- Sélectionner le type de serveur JBoss Community > **« WildFly 8.x Runtime »**
- Cliquez sur Browse au niveau du Home Directory et allez pointer sur votre répertoire d'installation de Jboss



- Sélectionner la vue '**Serveurs**' (elle est généralement en bas de l'écran à côté de la console).
- Utiliser le menu contextuel : '**New->Server**'.
- Vérifier que le type de serveur sélectionné est '**WildFly 8.x**' et cliquer sur '**Terminer**' pour demander la création du serveur. > Finish.



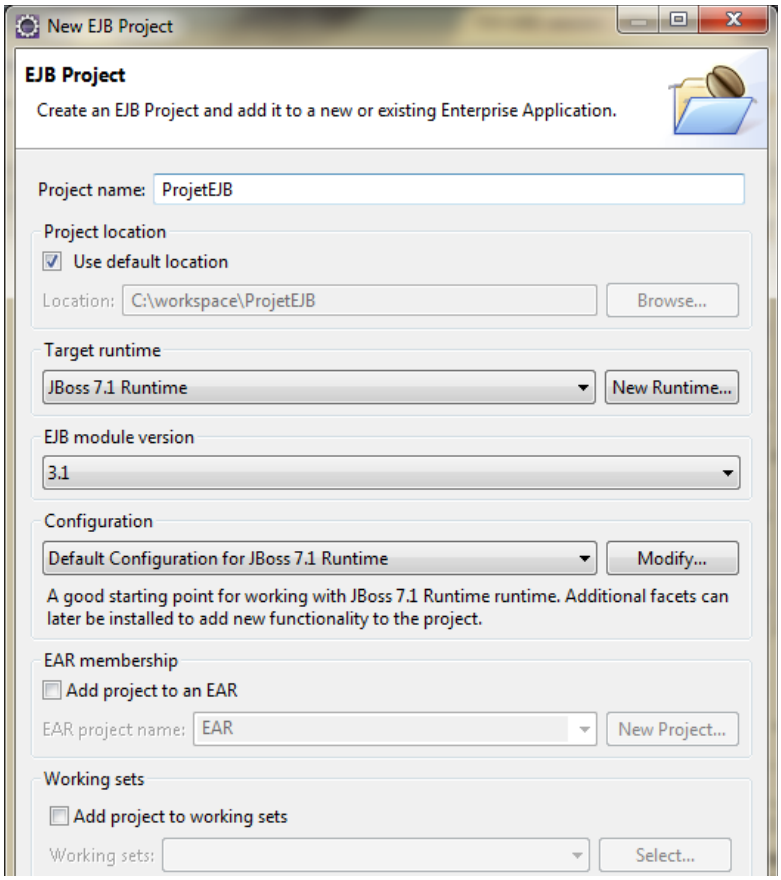
Une fois ces étapes de configuration effectuées, tester le bon fonctionnement du serveur en demandant son exécution à partir de la vue '**Serveurs**'. Si votre serveur démarre correctement, vous devez voir les informations suivantes :

```
16:35:01,801 INFO  [org.jboss.as] (Controller Boot Thread) JBAS015874: JBoss AS 7.1.1.Final "Brontes" started in 13396ms - Started 133 of 208 services (74 services are passive or on-demand)
```

2 Premier EJB session remote sans état – le hello world

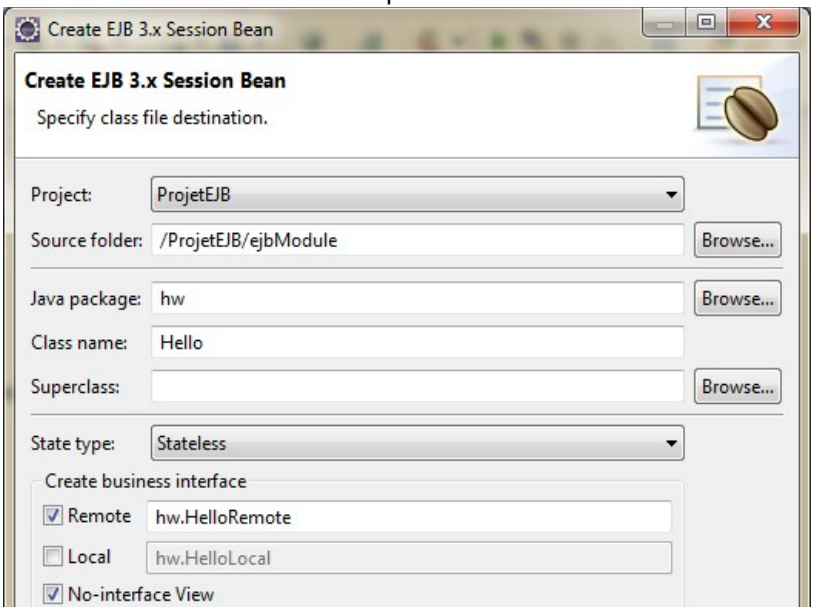
2.1 Créer le projet

Créer un projet EJB : File->New->Other->EJB Project comme ci-dessous puis Finish



2.2 Créer l'EJB session

- Dans le package explorer, faire un clic droit sur le ProjetEJB puis New>Other>EJB>Session Bean (EJB 3.x)
- Choisir un EJB session remote comme ci-dessous puis Finish



2.3 Ajouter un prototype de méthode dans l'interface Hello

```
@Remote
public interface HelloRemote {
	public String sayHello();
}
```

Rappel : un EJB local ne peut être appelé qu'à l'intérieur de la même machine virtuelle Java alors qu'un EJB remote peut être appelé depuis n'importe où.

2.4 Implémentez la méthode dans la classe HelloBean

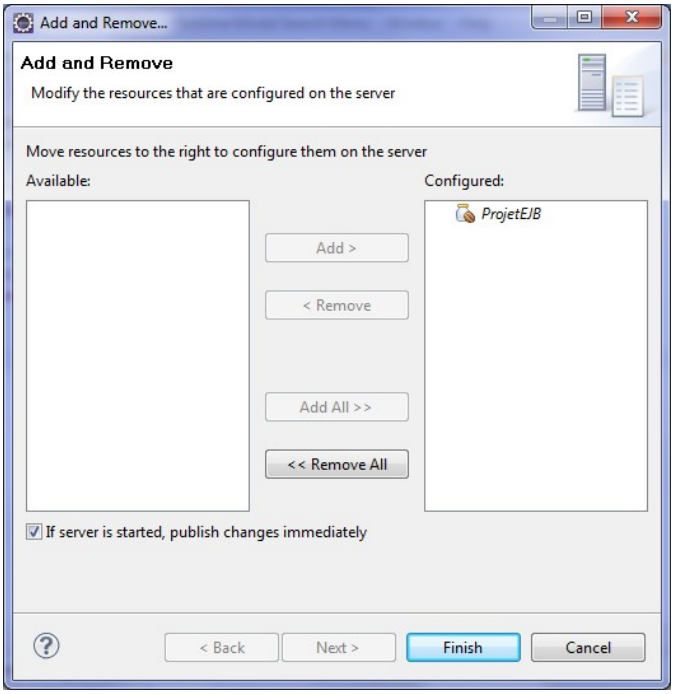
```
@Stateless
@LocalBean
```

```
public class Hello implements HelloRemote {
    public Hello() {}
      @Override
      public String sayHello() {
            return "Hello world";
      }
}
```

Rappel : un EJB stateful gère l'état conversationnel avec le client (chaque client a son propre EJB qui lui est dédié) alors qu'un EJB stateless peut répondre aux demandes de plusieurs clients.

2.4.1 Construction et déploiement de l'EJB

Pour construire et déployer l'EJB, il suffit de le déclarer dans le serveur JBOSS. A partir de la vue 'Serveurs', sélectionner le serveur JBoss et utiliser l'option '**Ajouter et supprimer des projets...**' du menu contextuel pour déployer le projet EJB :



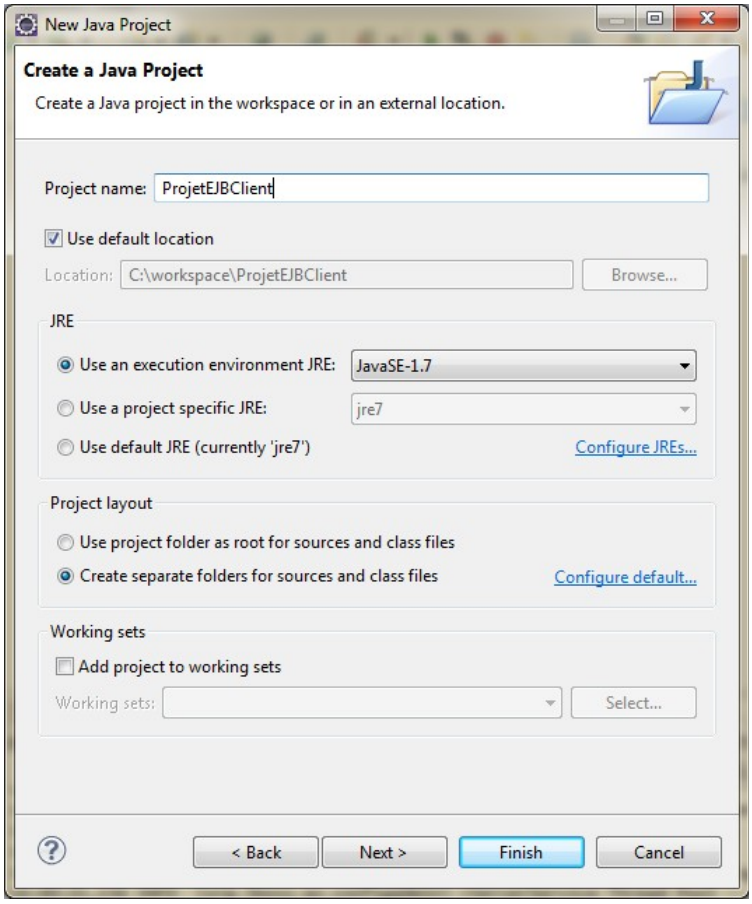
Si l'EJB est correctement déployé les lignes suivantes avec les JNDI de l'EJB doivent apparaître dans la console :

```
16:49:22,026 INFO  [org.jboss.as.server.deployment] (MSC service thread 1-10) JBAS015876: Starting deployment
of "ProjetEJB.jar"
16:49:22,366  INFO    [org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeploymentUnitProcessor]  (MSC
service thread 1-9) JNDI bindings for session bean named Hello in deployment unit deployment "ProjetEJB.jar"
are as follows:
       java:global/ProjetEJB/Hello!hw.Hello
       java:app/ProjetEJB/Hello!hw.Hello
       java:module/Hello!hw.Hello
       java:global/ProjetEJB/Hello!hw.HelloRemote
       java:app/ProjetEJB/Hello!hw.HelloRemote
       java:module/Hello!hw.HelloRemote
       java:jboss/exported/ProjetEJB/Hello!hw.HelloRemote
16:49:22,698 INFO  [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS018559: Deployed "ProjetEJB.jar"
```

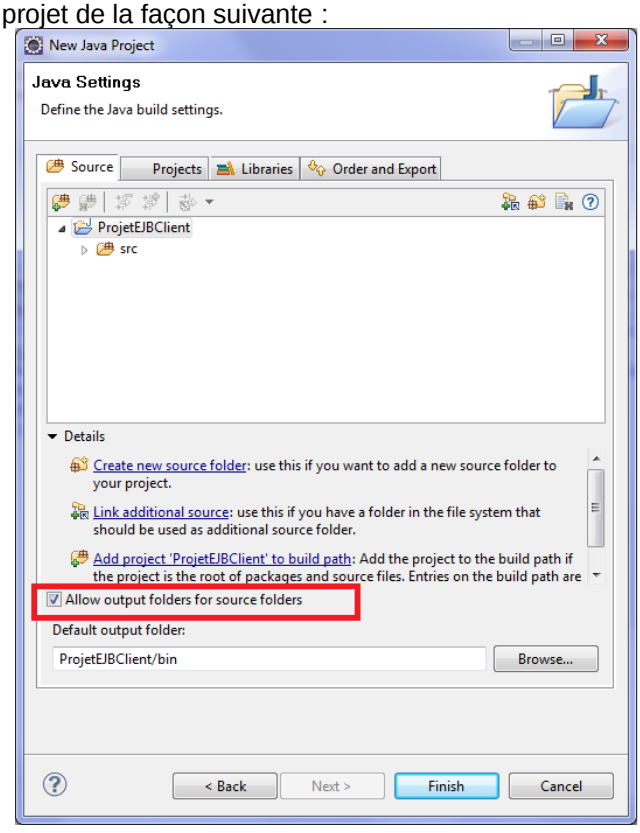
3 Ecriture d'un client

3.1 Créer un projet java

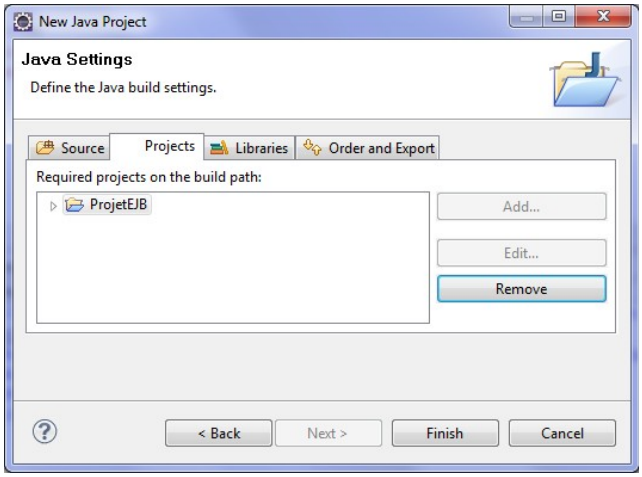
La dernière étape pour tester l'EJB est l'écriture du client. Pour cela, nous allons créer un **nouveau projet Java** avec les paramètres suivants :



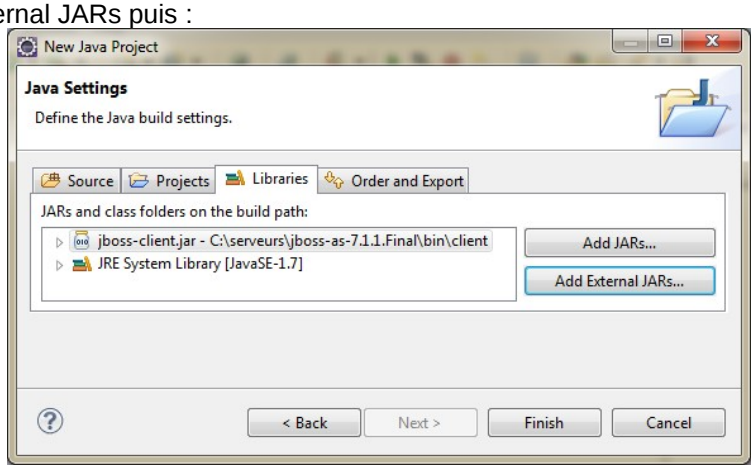
Cliquez sur Next et configurez le projet de la façon suivante :



Dans **l'onglet projects**, rajoutez le ProjetEJB pour lier les deux projets.



Dans **l'onglet Libraries** ajoutez la bibliothèque nécessaire côté client pour le fonctionnement d'une application utilisant un EJB. Cliquez sur Add external JARs puis :



3.2 Ajouter une classe cliente

Rajoutez un package client et une classe HelloClient dans votre projet. Collez le corps suivant dans votre client :

```
public class HelloClient {
   static {
      Security.addProvider(new JBossSaslProvider());
   }

   public static void main(String[] args) throws Exception {
      // Appel du stateless bean
      HelloRemote helloBean = lookupRemoteStatelessHello();
      System.out.println(helloBean.sayHello());
   }

    // Connexion au serveur et lookup du bean
     private static HelloRemote lookupRemoteStatelessHello() throws NamingException {
        HelloRemote remote=null;
      try {
          Properties jndiProps = new Properties();
          jndiProps.setProperty(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");
          InitialContext ctx = new InitialContext(jndiProps);
          remote = (HelloRemote) ctx.lookup("ejb:/ProjetEJB/Hello!hw.HelloRemote");
         } catch (Exception e) {
          e.printStackTrace();
                  }
      return remote;
   }
}
```

3.3 Paramètres de connexion à JBOSS

Il ne nous reste plus qu'à donner les informations de connexion à JBoss. Comme toujours en J2EE, ces paramètres ne sont pas stockés « en dur » mais dans un fichier xml. Dans Eclipse, clic droit sur ProjetEJBClient\src > New > Other > General > File. Créer un fichier jboss-ejb-client.properties dans ProjetEJBClient/src.

Copier dans ce fichier les informations qui permettront à l'application cliente de se connecter au service de nommage du serveur JBoss :

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port = 8080
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

3.4 Exécution de l'application

Run/Run As/ Java Application sur la classe cliente.