TP08
Les composants « EJB Session »

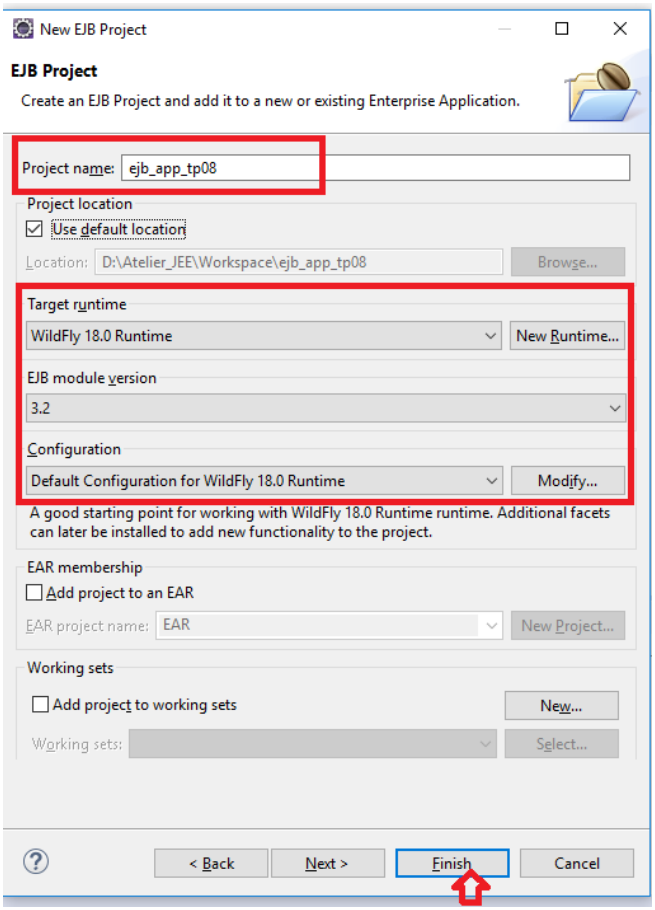
Objectifs

Réalisation des traitements sur les données à travers des EJB Session

1. **Implémenter les services EJB**
   - Créer un EJB Session
   - Utiliser EntityManager
   - Publier l'EJB avec un nom JNDI
2. **Invoquer les services EJB :**
   - A partir d'un client lourd JAVA
   - A partir d'un client WEB JAVA

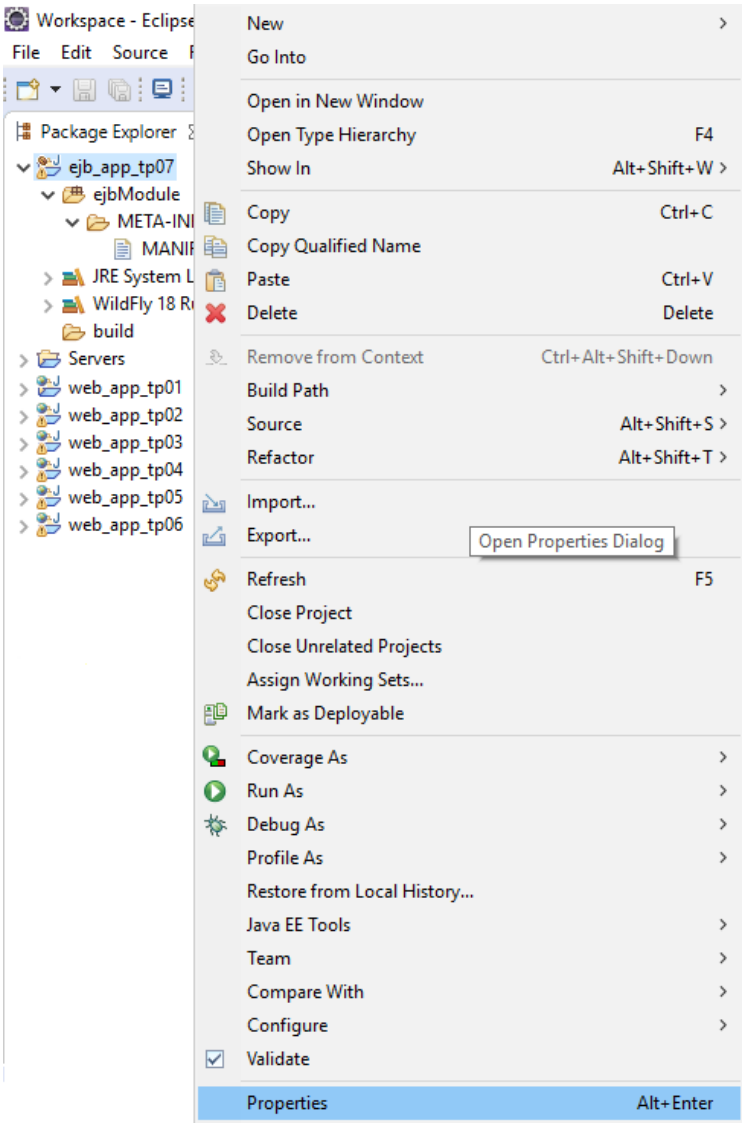
A. Créer un projet EJB

1. Dans **eclipse**, accéder au « **File/New/EJB Project** ».
2. Créer un nouveau projet EJB nommé «**ejb_app_tp08**» associé au serveur **WildFly 18** et appuyer sur « **Finish** » :

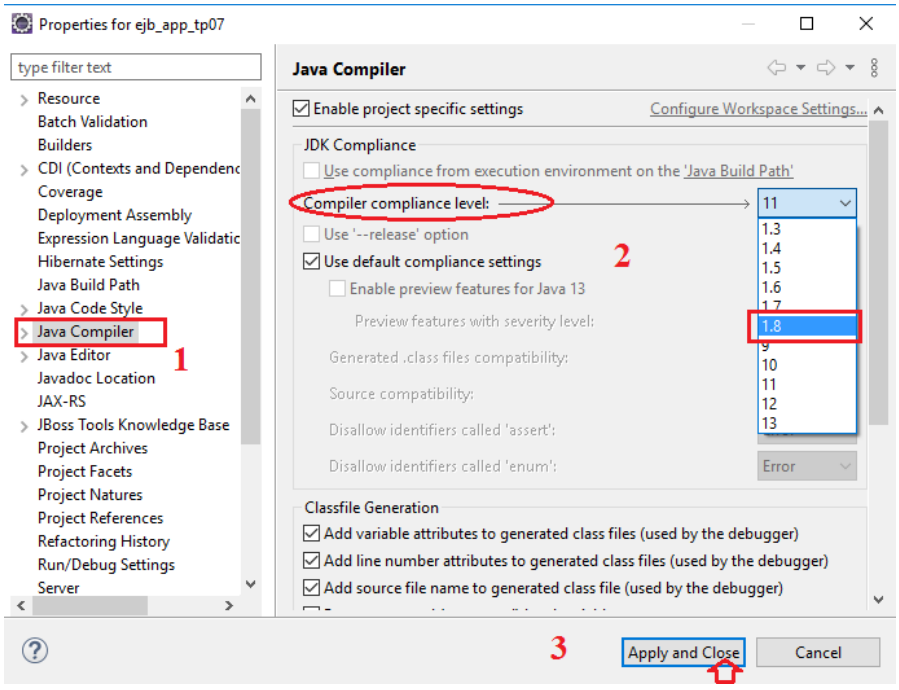


3. Rendre le niveau de conformité du compilateur à **1.8** :

a. Accéder aux propriétés du projet :



b. Rendre le niveau de conformité du compilateur à **1.8** :

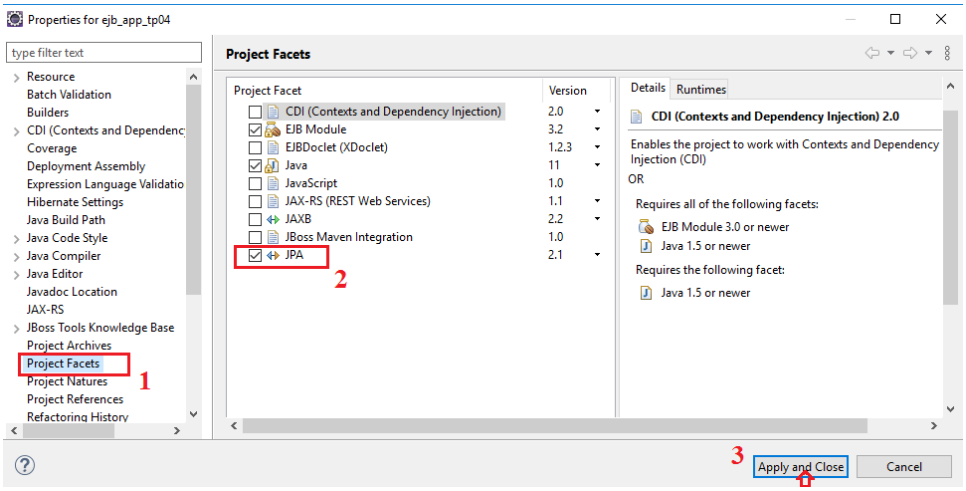


B. Créer un EJB Entity (couche « domaine »)

4. Ajouter les composants relatifs à JPA :

a. Accéder aux propriétés du projet.

b. Ajouter les éléments JPA (**Facet JPA**) au projet :



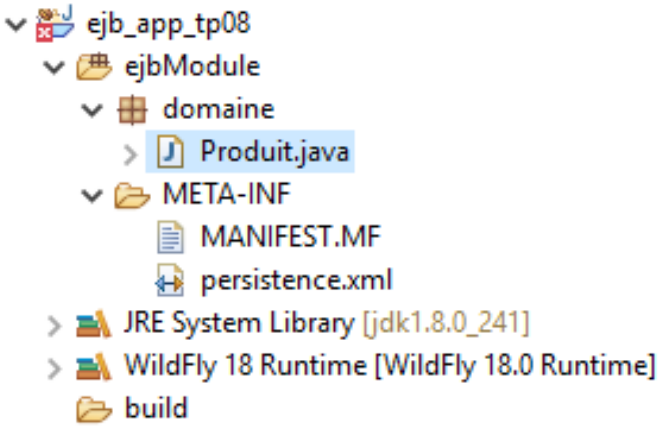
c. Remarquer l'ajout d'un fichier «**persistence.xml**» sous «**META-INF**» et le remplacer par le fichier «**persistence.xml**» donné en pièce jointe (utiliser la datasource «**MySqlDS**»).

d. Le fichier «**persistence.xml**» référence une classe «**Produit**» situé dans un package «**domaine**». Pour cette raison, il génère l'erreur suivante :

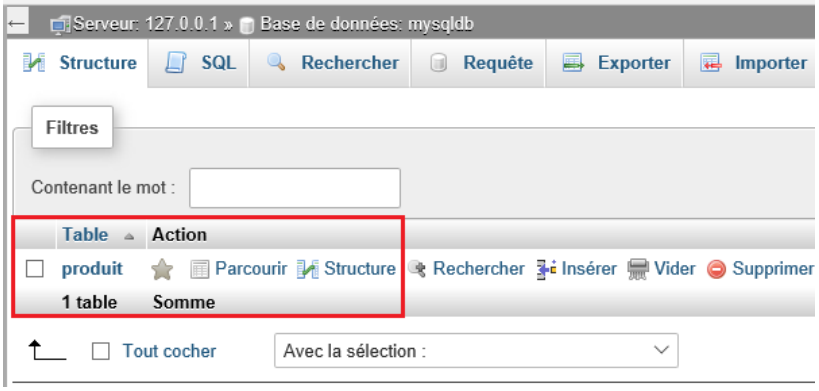
```
7    <persistence-unit name="ejb_app_tp08"
8                      transaction-type="JTA">
9        <jta-data-source>java:/MySqlDS</jta-data-source>
10       <class>domaine.Produit</class>
11       <properties>
12           <property name="javax.persistence.schema-generation.database.action"
13                     value="drop-and-create"/>
14       </properties>
15   </persistence-unit>
```

a. Créer donc, sous «**ejbModule**» du projet, un package «**domaine**».

b. Copier dans ce package «**domaine**» le fichier « **Produit.java** » donné en pièce jointe :



c. Lancer l'exécution du projet «**ejb_app_tp08**» avec **WildFly 18** et remarquer la création de la table «**produit**» dans la base de données «**MySqlDB**» :



C. Créer un EJB Session (couche « services » ou « métier »)

Un bean Session est une classe manipulée par le serveur JBoss qui présente un ensemble de fonctionnalités. Il implémente une interface qui expose son comportement. Cette interface peut être utilisée d'une manière locale et/ou distante.

5. Dans le dossier «**ejbModule**» créer un package «**services**».

6. Sélectionner le package «**services**» et créer une première interface «**ProduitSessionRemote**» à être utilisée à distance et ayant le code suivant :

```
package services;

import java.util.List;
import javax.ejb.Remote;
import domaine.Produit;

@Remote
public interface ProduitSessionRemote
{
    public Produit addProduit(Produit p);
    public Produit updateProduit(Produit p);
    public void deleteProduit(Long id);
    public Produit getProduit(Long id);
    public List<Produit> getAllProduits();
}
```

7. De même, créer dans le package «**services**» une autre interface locale nommée «**ProduitSessionLocal**» qui expose les mêmes méthodes :

```
package services;

import java.util.List;
import javax.ejb.Local;
import domaine.Produit;

@Local
public interface ProduitSessionLocal {
    public Produit addProduit(Produit p);
    public Produit updateProduit(Produit p);
    public void deleteProduit(Long id);
    public Produit getProduit(Long id);
    public List<Produit> getAllProduits();
}
```

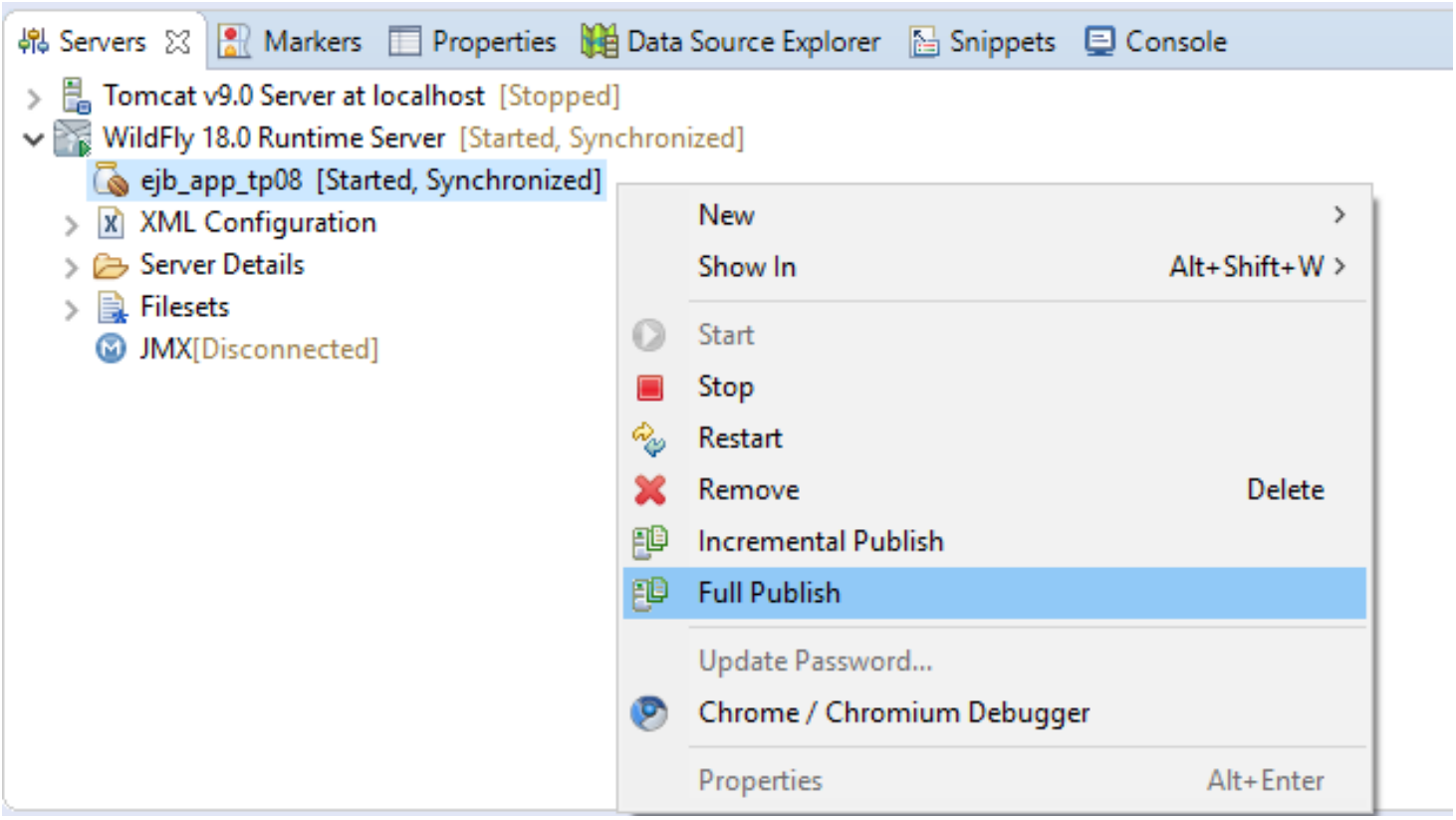
8. Créer, dans le même package «services», un EJB Session (nouvelle classe) nommé «**ProduitSession**» qui implémente les deux interfaces «**ProduitSessionRemote**» et «**ProduitSessionLocal**» et qui redéfinit toutes leurs méthodes :

```
package services;

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

import domaine.Produit;

@Stateless (name ="PR")
public class ProduitSession implements
ProduitSessionLocal, ProduitSessionRemote{

    @PersistenceContext
    private EntityManager em ;
```

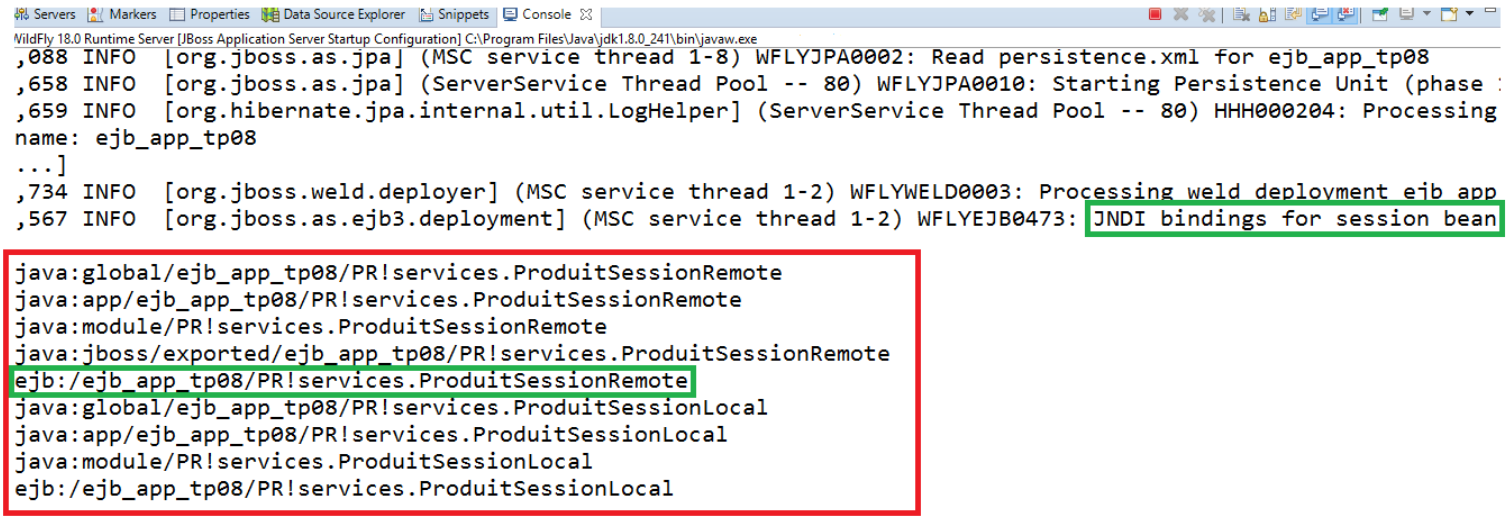
```
    @Override
    public Produit addProduit(Produit p) {
        em.persist(p);
        return p;
    }

    @Override
    public Produit getProduit(Long id) {
        Produit p =(Produit)em.find(Produit.class, id);
        return p;
    }

    @Override
    public List<Produit> getAllProduits() {
    Query sql =em.createQuery("select p from Produit p");
        return sql.getResultList();
    }

    @Override
    public Produit updateProduit(Produit p) {
    Produit pu=(Produit)em.find(Produit.class, p.getId());
        pu.setDateAchat(p.getDateAchat());
        pu.setDesignation(p.getDesignation());
        pu.setPrix(p.getPrix());
        pu.setQuantite(p.getQuantite());
        pu=em.merge(pu);
        return pu;
    }

    @Override
    public void deleteProduit(Long id) {
        Produit pd =(Produit)em.find(Produit.class, id);
        em.remove(pd);
    }
}
```

- **@Stateless : pour définir un bean sans état.**
- **name ="PR" : le nom de publication du bean dans l'annuaire JNDI.**
- **@PersistenceContext : pour récupérer une instance dynamique**
- **EntityManager em : déclaration d'un objet responsable de la persistance des entités et réalise le mapping ORM.**

9. Publier le projet afin d'enregistrer le bean «**ProduitSession**» dans l'annuaire **JNDI** (Sélectionner le projet dans le volet « **Servers** » et choisir la commande « **Full Publish** »



10. Visualiser, dans la console, le résultat de publication du bean Session. Remarquer les différentes nomenclatures attribuées.

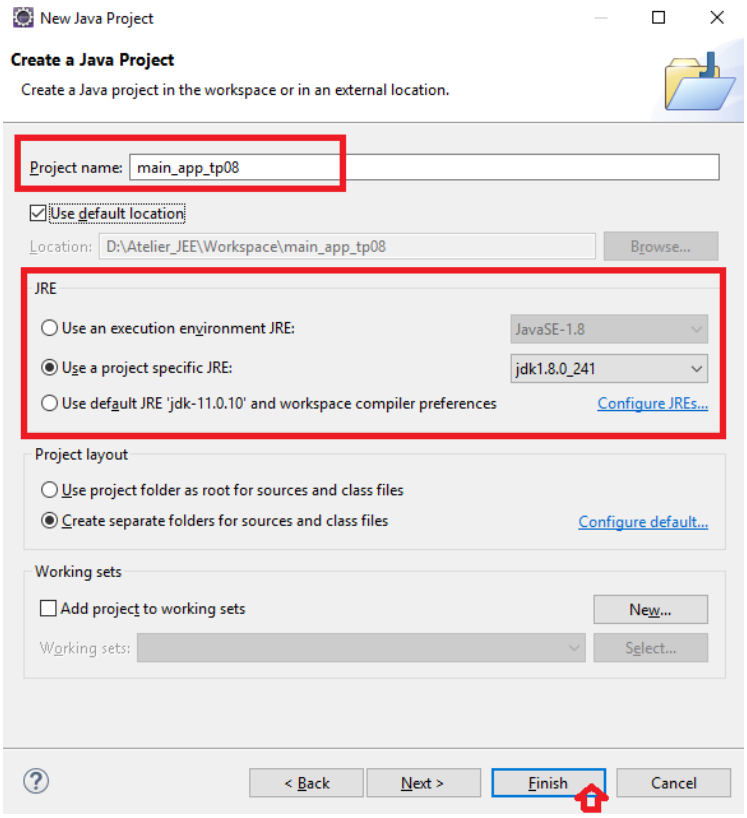


11. Nous utilisons, par la suite, pour un accès à distance celle qui est encadrée en vert :

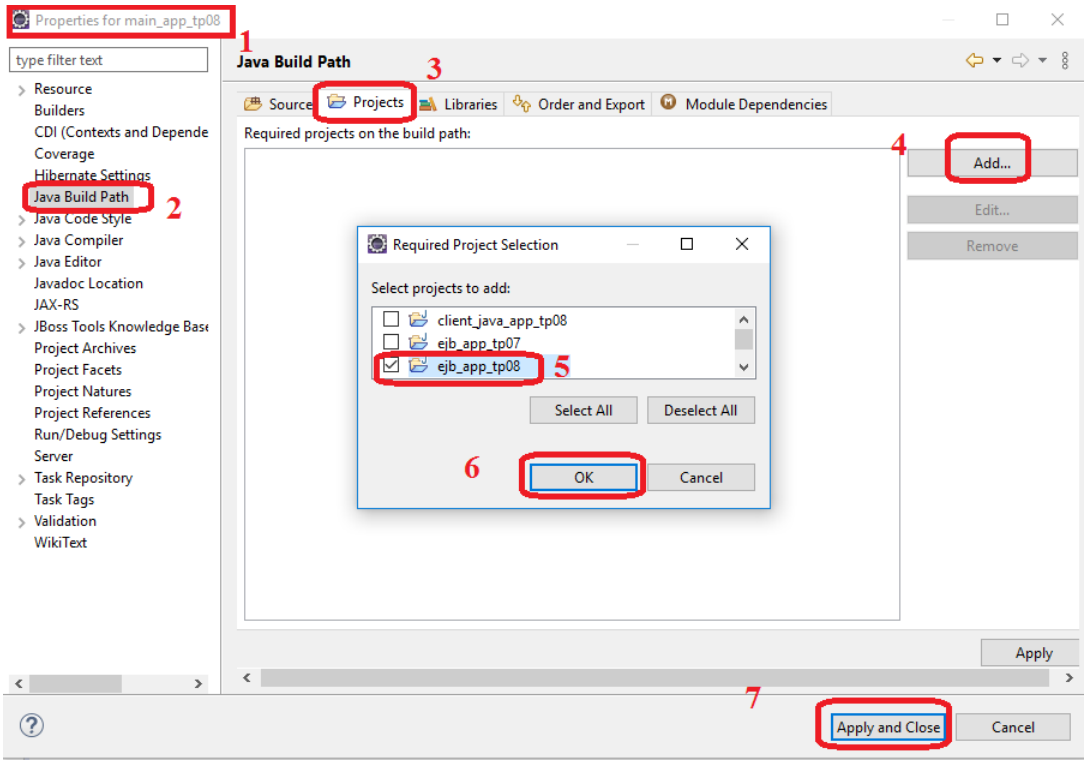
```
ejb:/ejb_app_tp08/PR!services.ProduitSessionRemote
```

D. Créer un Client Java (Lourd) pour tester l'invocation à distance au bean Session

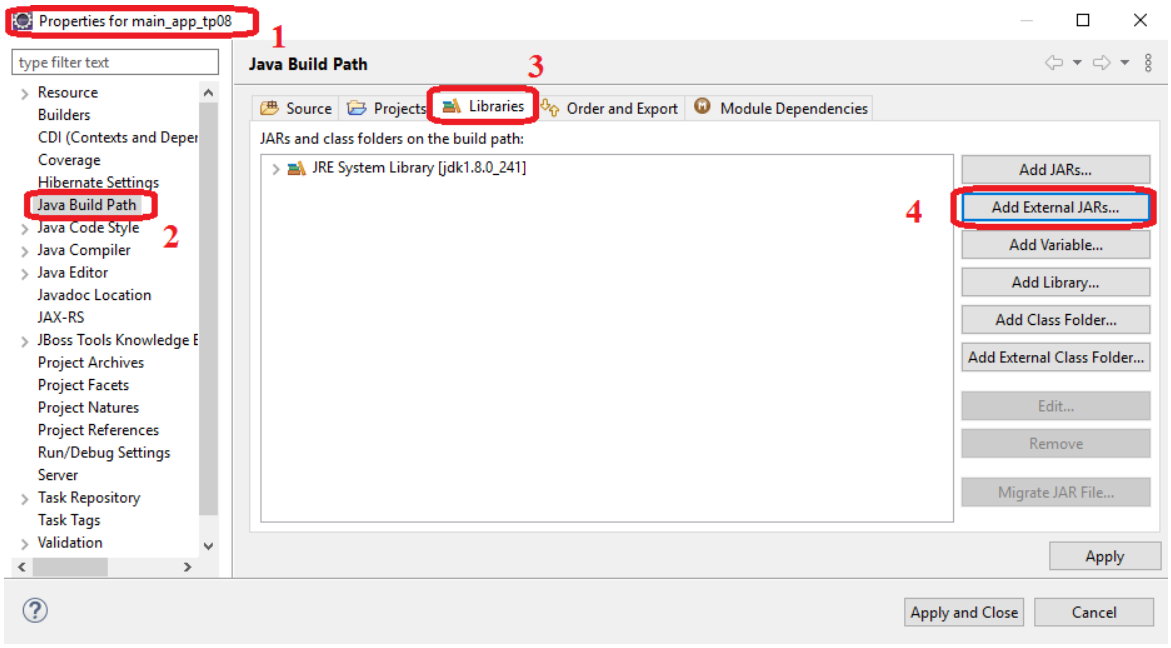
12. Créer un nouveau projet « **Java Project** » nommé «**main_app_tp08**» :



13. Ce projet client a besoin d'accéder à l'interface distante du bean session et à certains composants du projet «**ejb_app_tp08**» (Projet EJB). Pour ce faire, accéder aux propriétés du projet «**main_app_tp08**» et ajouter une dépendance au projet EJB ayant publié le bean session :

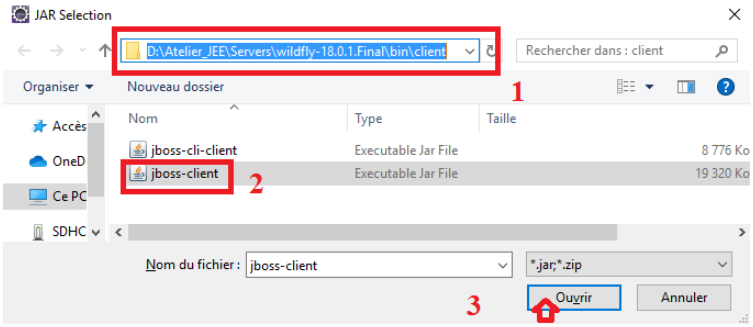


14. Le projet client a besoin aussi d'une autre dépendance (bibliothèque) pour accéder, comme client, au serveur **WildFly 18**. Pour ce faire, ajouter un fichier JAR externe nommé « **jboss-client.jar** » et situé sous le sous-dossier « **/bin/client** » du dossier racine du serveur **WildFly 18** :

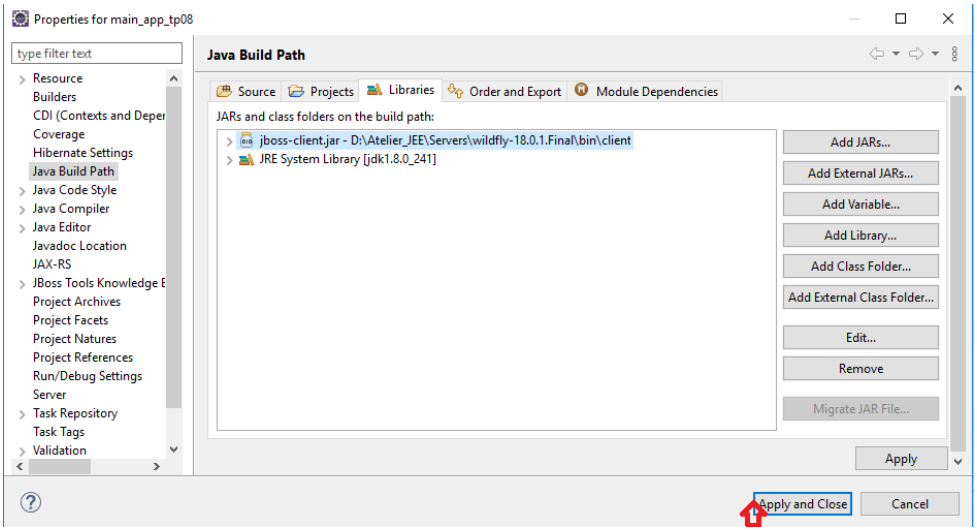


15. Spécifier le chemin du fichier JAR :

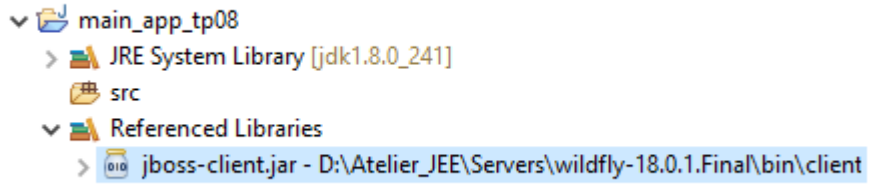
D:\Atelier_JEE\Servers\wildfly-18.0.1.Final\bin\client



16. Valider l'ajout du fichier JAR et fermer



17. Remarquer l'ajout du fichier JAR dans la liste des bibliothèques référencées :



18. Sélectionner « **src** » et créer une classe «**ClientEJBRemote**» dans un package «**presentation**»:

```
package presentation;
import java.util.List;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
// utilisation des comosants du projet EJB
import domaine.Produit;
import services.ProduitSessionRemote;
public class ClientEJBRemote {

    public static void main(String[] args) {

try {
    //Initialiser le contexte de connextion au serveur EJB (WildFly)
    Context ctx =new InitialContext();
    //Spécifier le nom de publication du bean "ProduitSession" dans JNDI
    String jndiBeanName="ejb:/ejb_app_tp08/PR!services.ProduitSessionRemote";
    // récupérer une référence au bean distant (comme un proxy)
    ProduitSessionRemote beanRemote
=(ProduitSessionRemote)ctx.lookup(jndiBeanName);

    //Accèder aux méthodes à distance (selon le protocole RMI)
    //Créer un produit (sans propriétés)
    Produit p = new Produit();
    //Appeler la méthode distante 'ajouterProduit'
    beanRemote.addProduit(p);
    //Afficher la liste des produits existants dans la BD
    List<Produit> lp = beanRemote.getAllProduits();
    System.out.println("--------------------");
    for( Produit pi : lp)
```

```
        {
            System.out.println(pi);
        }
        System.out.println("--------------------");

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

19. Lancer l'exécution de l'application client (**Run As Java Application**).

20. Remarquer le déclenchement de l'erreur suivante :

```
javax.naming.NoInitialContextException: Need to specify class name in environment or system property, or as an applet parameter, or in an application resource file:  java.naming.factory.initial
        at javax.naming.spi.NamingManager.getInitialContext(NamingManager.java:662)
        at javax.naming.InitialContext.getDefaultInitCtx(InitialContext.java:313)
        at javax.naming.InitialContext.getURLOrDefaultInitCtx(InitialContext.java:350)
        at javax.naming.InitialContext.lookup(InitialContext.java:417)
        at presentation.ClientEJBRemote.main(ClientEJBRemote.java:22)
```

21. Pour corriger cette erreur, il est nécessaire de spécifier les propriétés de connexion en ajoutant dans le dossier «**src**» du projet les deux fichiers de propriétés suivants (donnés en pièce jointe):

- **jboss-ejb-client.properties**
- **jndi.properties**

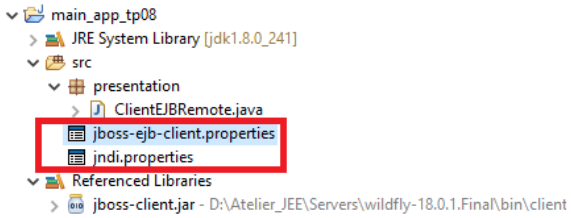
-Voici le contenu du fichier **jboss-ejb-client.properties** :

```
endpoint.name=client-endpoint
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=127.0.0.1
remote.connection.default.port=8080
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

-Voici le contenu du fichier **jndi.properties** :

```
java.naming.factory.url.pkgs=org.jboss.ejb.client.naming
```

22. Voici l'aperçu du projet ainsi configuré :

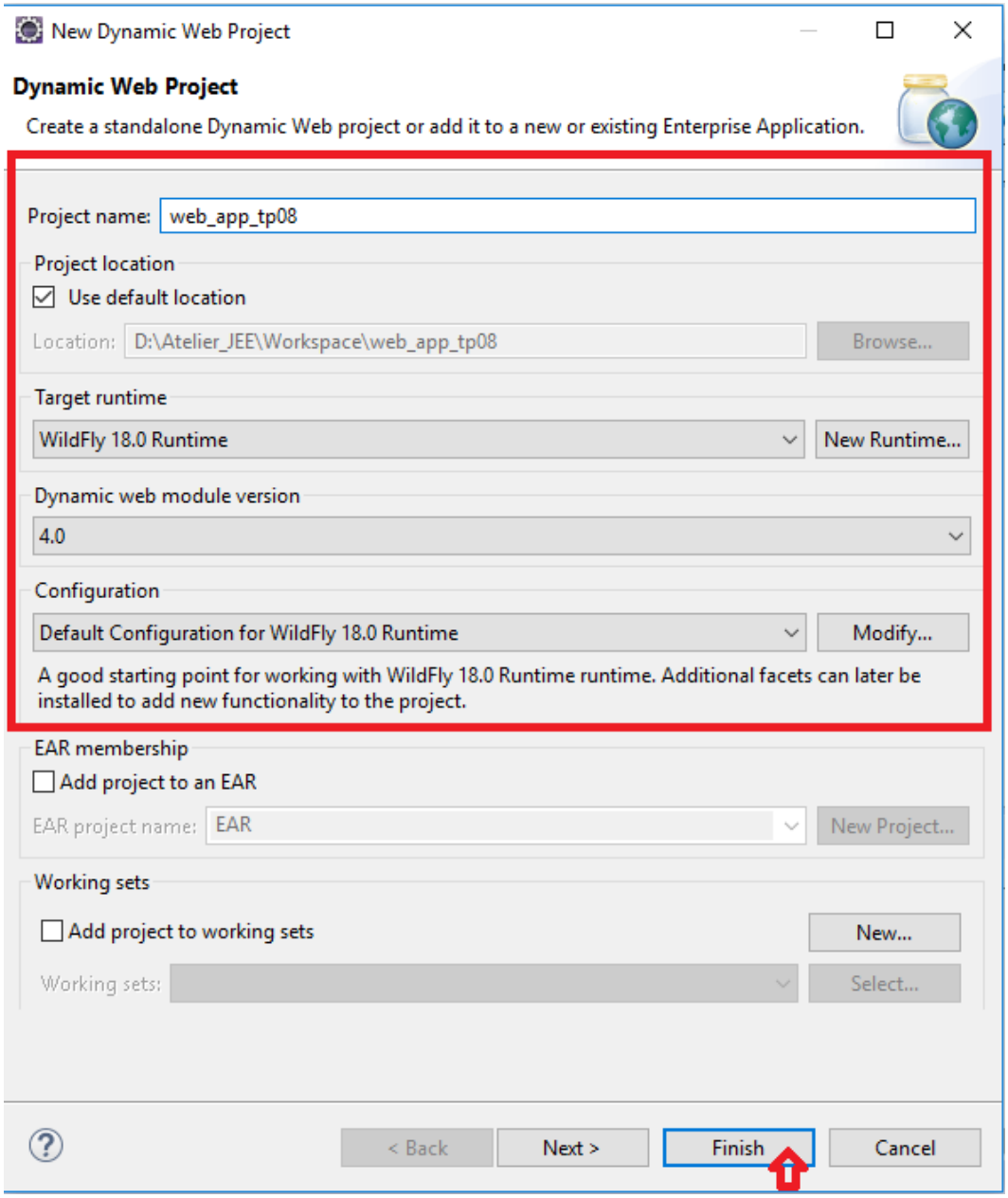


23. Relancer de nouveau l'exécution du client JAVA et visualiser l'affichage du produit inséré :

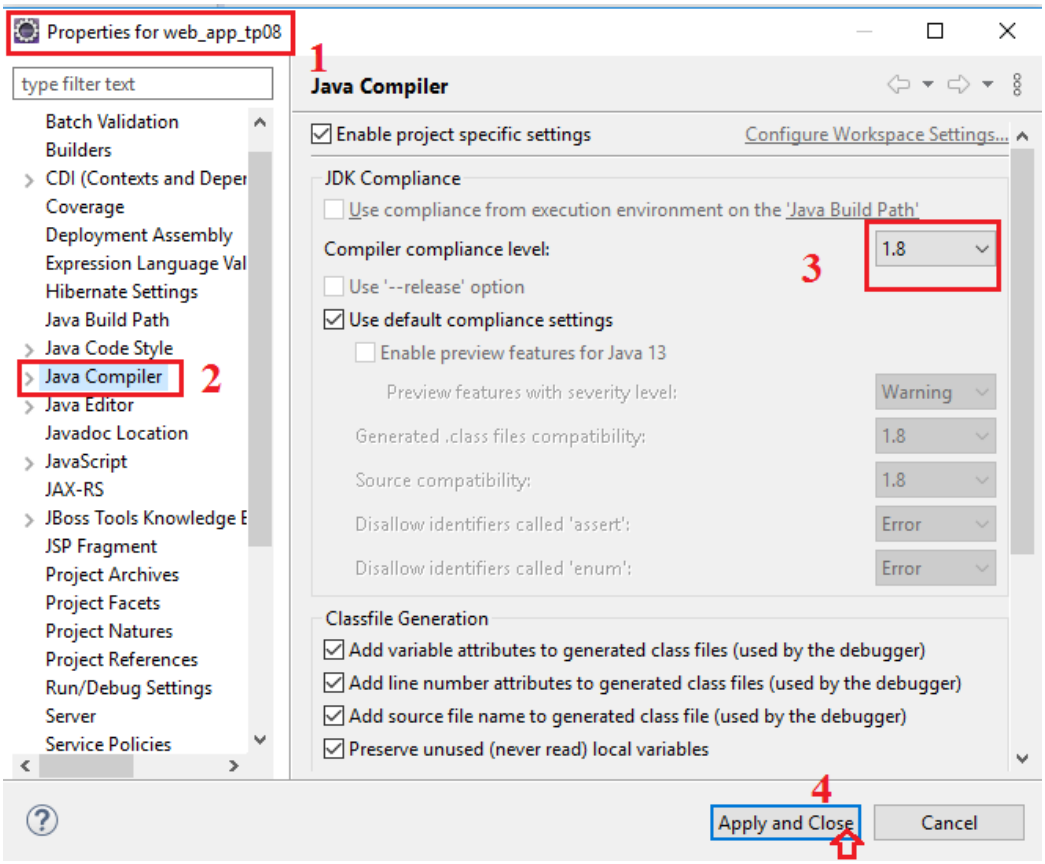
```
INFO: JBoss Remoting version 5.0.15.Final
avr. 05, 2021 2:05:39 PM org.jboss.ejb.client.legacy.ElytronLegacyConfigu
INFO: EJBCLIENT000069: Using legacy jboss-ejb-client.properties security
------------------
Produit [id=1, designation=null, prix=0.0, quantite=0, dateAchat=null]
------------------
```

E. Créer un Client WEB (Avec WildFly) pour tester l'invocation à distance au bean Session

24. Passer maintenant à invoquer à distance au bean Session à partir d'un client WEB. Pour ce faire créer un nouveau projet web « **Dynamic Web Project** » nommé «**web_app_tp08**» associé au serveur WildFly 18 qui peut se comporter aussi comme un conteneur WEB :

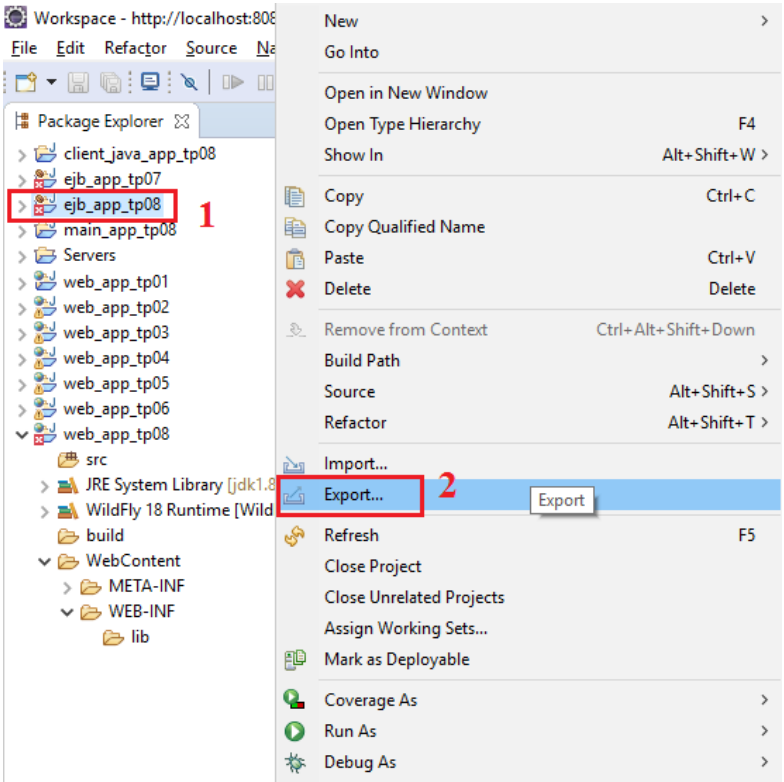


25. Rendre le niveau de conformité du compilateur à 1.8 :

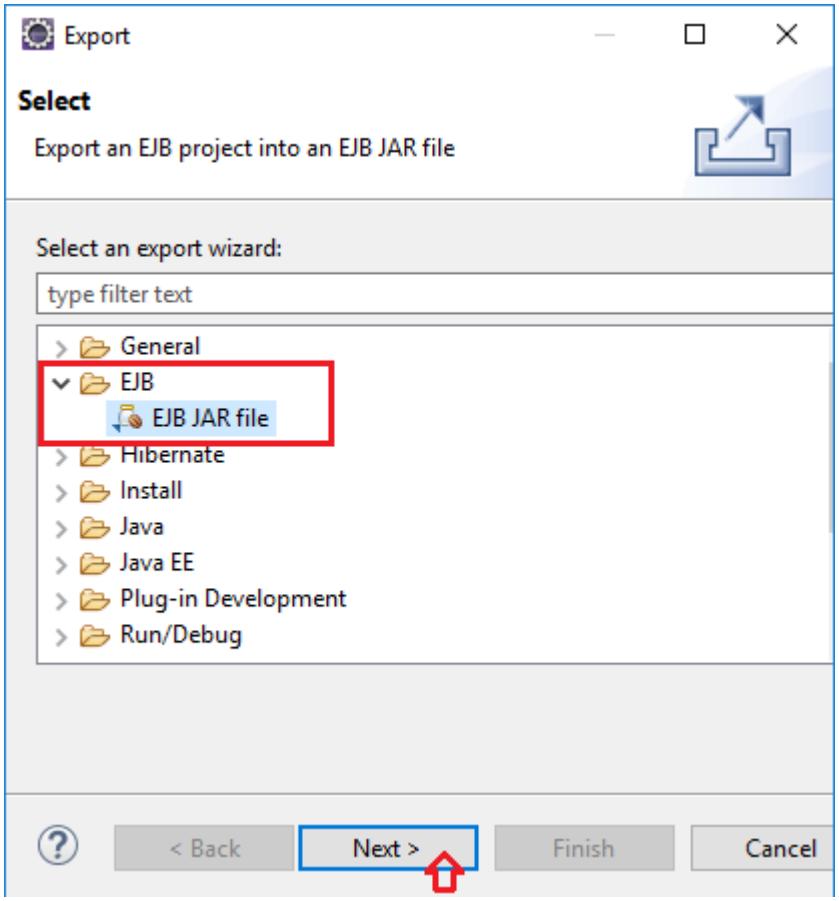


26. On a besoin, dans ce projet WEB, d'utiliser des classes et des interfaces du projet «**ejb_app_tp08**». Pour cette raison, exporter le projet «**ejb_app_tp08**» sous forme d'un fichier JAR et le placer dans le dossier «**WebContent/WEB-INF/lib**» du projet WEB :

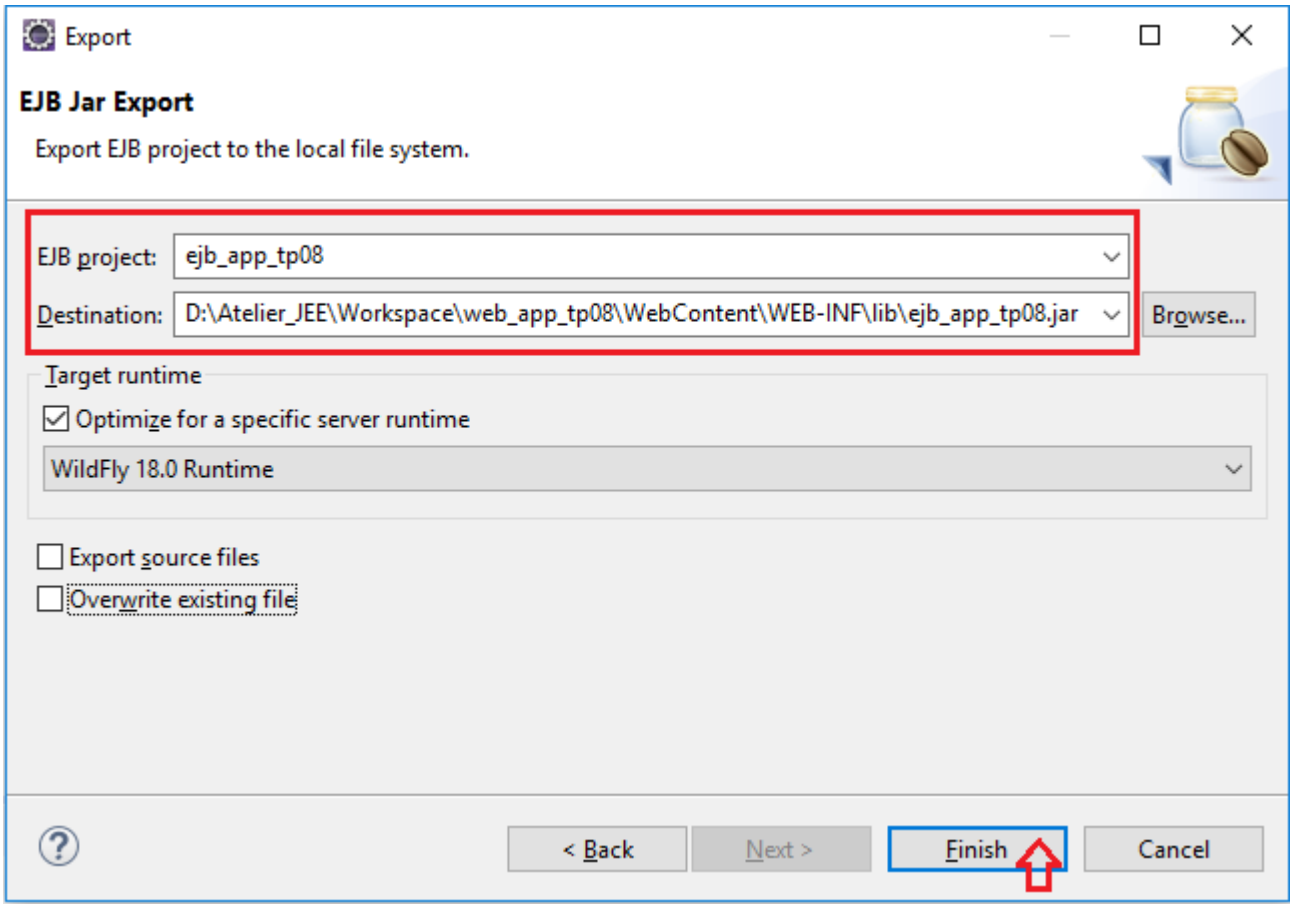
a. Exporter le projet «**ejb_app_tp08**»:



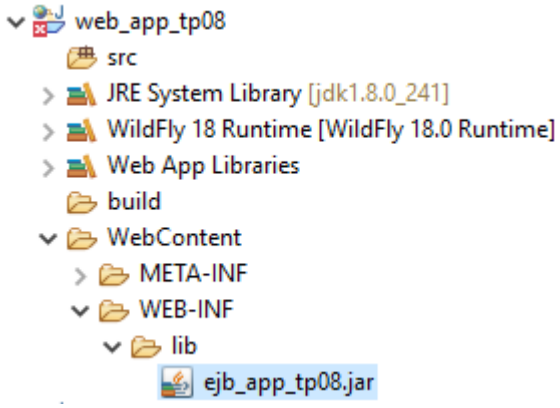
b. Choisir le format **EJB JAR file** :



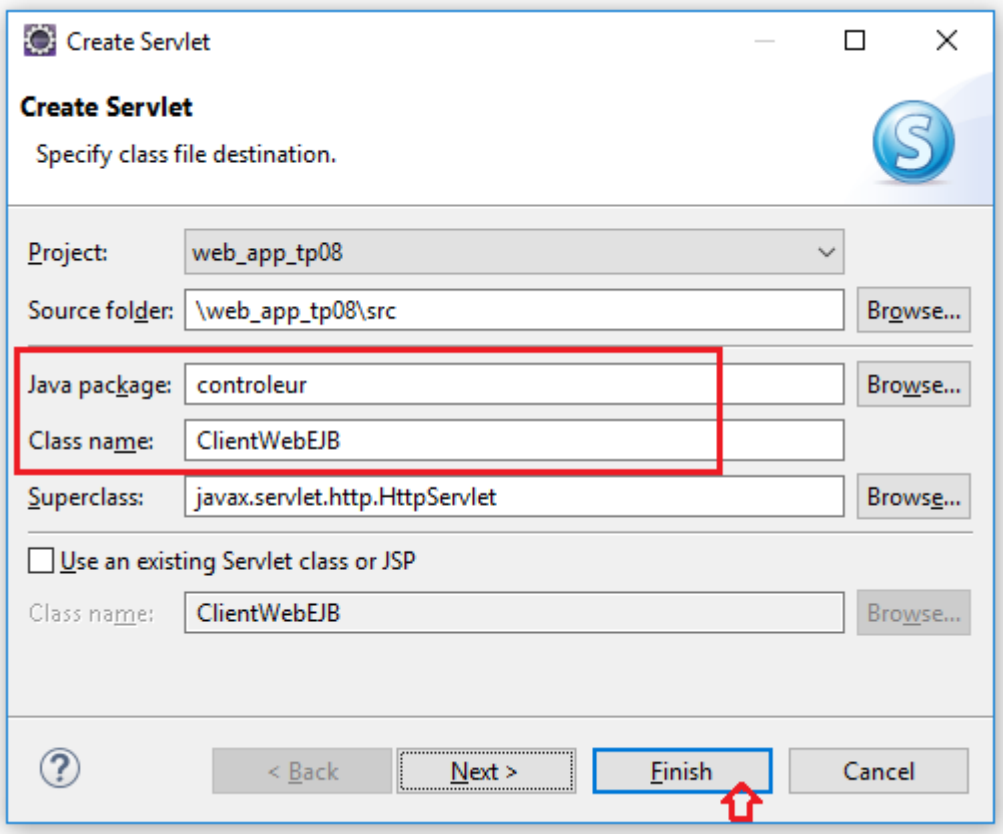
c. Enregistrer l'export dans le dossier «**WebContent/WEB-INF/lib**» du projet WEB «**web_app_tp08**» :



d. Réaliser un rafraîchissement « **Refresh** » du projet WEB pour visualiser le fichier JAR nouvellement créé :



27. Créer, dans un package «**controleur**» une servlet nommée «**ClientWebEJB**» qui permet d'invoquer des méthodes du bean Sesion «**ProduitSession**» :



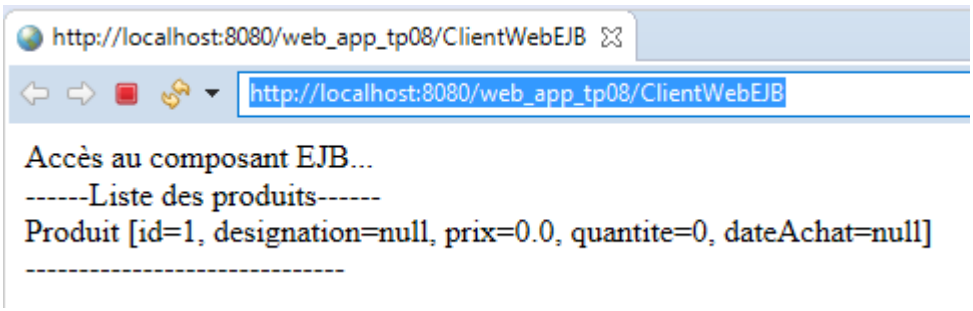
28. Prendre le code suivant de la servelt «**ClientWebEJB**» qui utilise un objet «**metier**» qui référence à distance le bean Session et appelle les méthodes « **addProduit** » et « **getAllProduits** » :

```
package controleur;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;
```

```
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import domaine.Produit;
import services.ProduitSessionRemote;

/**
 * Servlet implementation class ClientWebEJB
 */
@WebServlet("/ClientWebEJB")
public class ClientWebEJB extends HttpServlet {

// injecter une instance d'un bean session qui implémente cette interface
    @EJB(lookup ="ejb:/ejb_app_tp08/PR!services.ProduitSessionRemote")
    private ProduitSessionRemote metier;


    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ClientWebEJB() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {

        // Récupérer l'objet d'écriture de la réponse
```

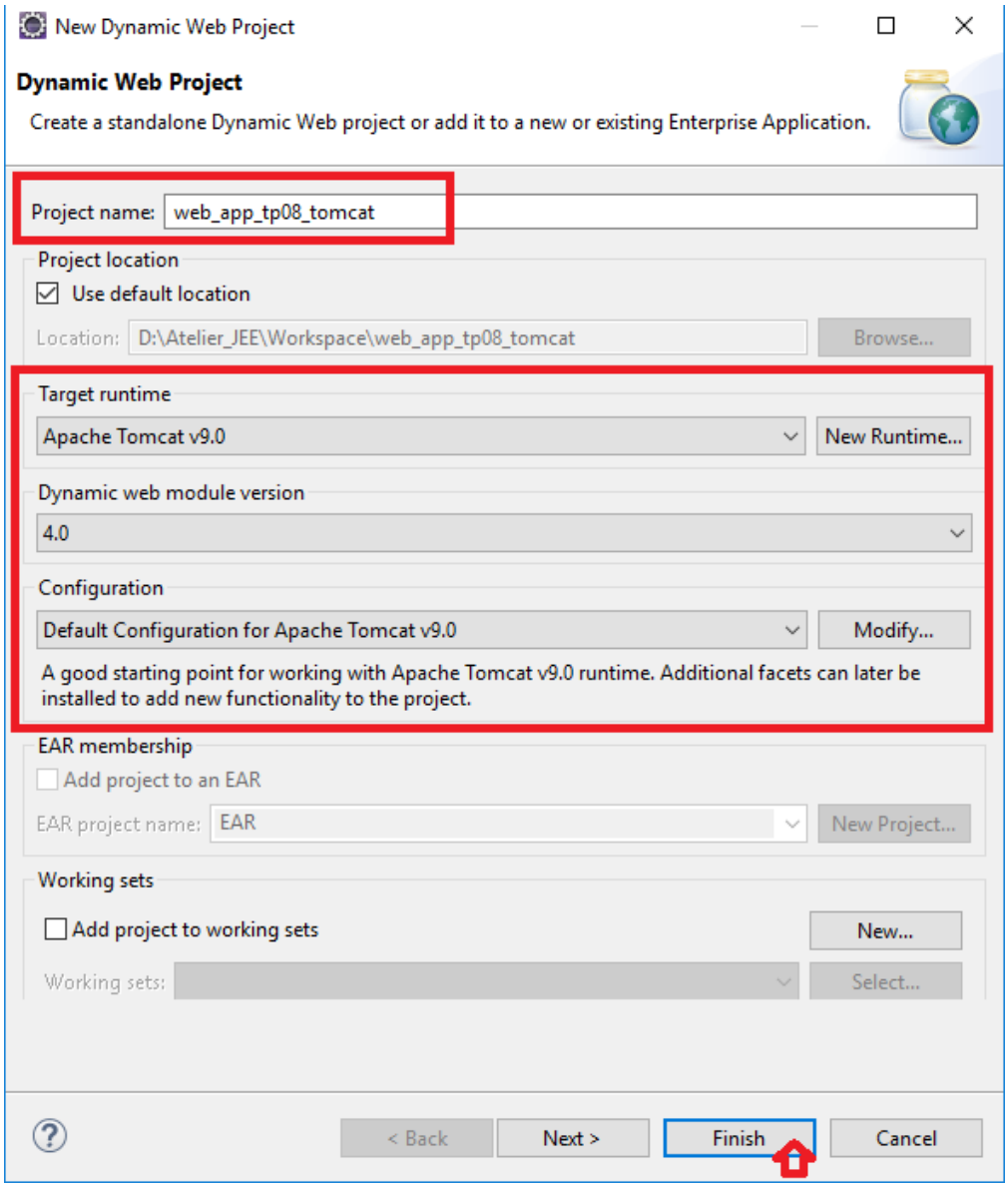
```
        PrintWriter out = response.getWriter();

        out.println("Accès au composant EJB...<br>");

        //Créer un produit (sans propriétés)
        Produit p = new Produit();
        //Appeler la méthode distante 'ajouterProduit'
        metier.addProduit(p);

    //Afficher la liste des produits existants dans la BD
    List<Produit> lp = metier.getAllProduits();
    out.println("------Liste des produits------<br>");
                for( Produit pi : lp)
                {
                    out.println(pi);
                    out.println("<br>");
                }
    out.println("------------------------------<br>");

    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest
request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

29. Lancer l'exécution de la servlet :



F. Créer un Client WEB (avec Tomcat) pour tester l'invocation à distance au bean Session

30. Passer maintenant à invoquer à distance au bean Session à partir d'un client WEB administré par le conteneur « **Tomcat** ». Pour ce faire créer un nouveau projet web « **Dynamic Web Project** » nommé «**web_app_tp08_tomcat**» associé au serveur **Tomcat 9** :



31. Rendre le niveau de conformité du compilateur à **1.8** (voir question25)

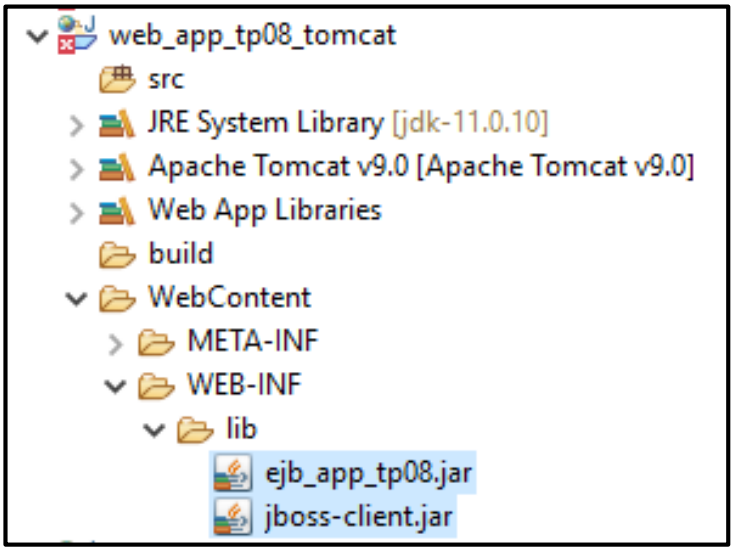
32. Exporter le projet « **ejb_app_tp08** » sous un format JAR dans le dossier le dossier «**WebContent/WEB-INF/lib**» du projet WEB «**web_app_tp08_tomcat**». (voir question 26).

33. Le projet «**web_app_tp08_tomcat**» a besoin aussi d'une autre dépendance (bibliothèque) pour accéder, comme client, au serveur **WildFly 18**. Pour ce faire :

a. copier le fichier JAR externe nommé « **jboss-client.jar** » situé sous le sous-dossier « **/bin/client** » du dossier racine du serveur **WildFly 18**

b. le coller dans le dossier le dossier «**WebContent/WEB-INF/lib**» du projet WEB «**web_app_tp08_tomcat**».

34. Rafraichir le projet WEB «**web_app_tp08_tomcat**» pour visualiser les deux fichiers JAR nouvellement ajoutés :



35. Créer, dans un package «**controleur**» une servlet nommée «**ClientWebEJBTomcat**» qui permet d'invoquer des méthodes du bean Sesion «**ProduitSession**» :

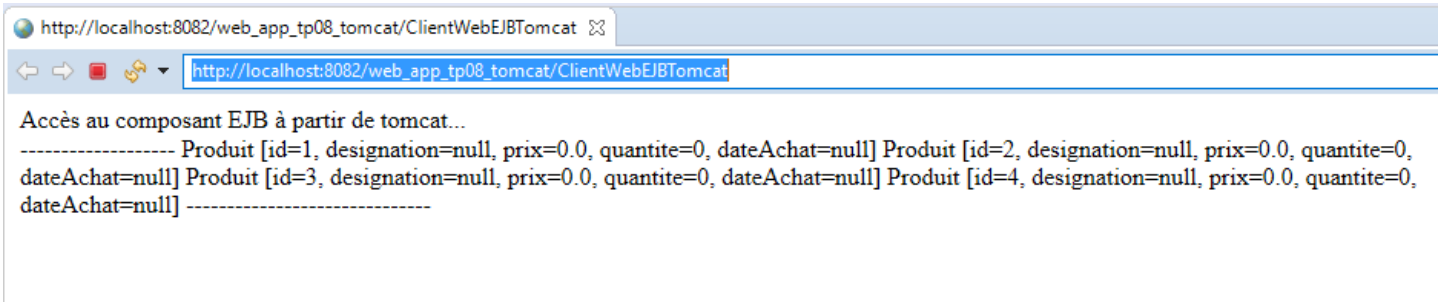
```
package controleur;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Hashtable;
import java.util.List;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import domaine.Produit;
import services.ProduitSessionRemote;

/**
 * Servlet implementation class ClientWebEJBTomcat
 */
@WebServlet("/ClientWebEJBTomcat")
public class ClientWebEJBTomcat extends HttpServlet {
    private static final long serialVersionUID = 1L;

    // Déclaration de la structure qui contient les propriétés:
    Hashtable<String, String> props = null;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ClientWebEJBTomcat() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse
     *      response)
     */
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {

        // Récupérer l'objet d'écriture de la réponse
        PrintWriter out = response.getWriter();

out.println("Accès au composant EJB à partir de tomcat...<br>");

        try {

            // Initialiser le contexte de connextion au serveur
EJB (WildFly)
            Context ctx = new InitialContext(props);
// Spécifier le nom de publication du bean "ProduitSession" dans JNDI
            String jndiBeanName =
"ejb:/ejb_app_tp08/PR!services.ProduitSessionRemote";
// récupérer une référence au bean distant (comme un proxy)
            ProduitSessionRemote beanRemote =
(ProduitSessionRemote) ctx.lookup(jndiBeanName);
```

```
    // Accèder aux méthodes à distance (selon le protocole RMI)
    // Créer un produit (sans propriétés)
            Produit p = new Produit();
    // Appeler la méthode distante 'ajouterProduit'
            beanRemote.addProduit(p);
    //Afficher la liste des produits existants dans la BD
            List<Produit> lp = beanRemote.getAllProduits();
            out.println("--------------------");
            for (Produit pi : lp) {
                out.println(pi);
            }
            System.out.println("--------------------");

        } catch (NamingException e) {
            e.printStackTrace();
        }

        out.println("-----------------------------<br>");

    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse
     *      response)
     */
    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }

    @Override
    public void init(ServletConfig config) throws
ServletException {
        // TODO Auto-generated method stub
        super.init(config);

// Charger les proriétés utiles
props = new Hashtable<String, String>();
// propriétés du fichier "jndi.properties
props.put("java.naming.factory.url.pkgs", "org.jboss.ejb.client.naming");
props.put("endpoint.name", "client-endpoint");
props.put("remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED", "false");
props.put("remote.connections", "default");
props.put("remote.connection.default.host", "127.0.0.1");
props.put("remote.connection.default.port", "8080");
```

```
props.put("remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NO
ANONYMOUS", "false");
      }
}
```

36. Lancer l'exécution de la servlet :



NB : Tomcat n'est pas un conteneur EJB : Il ne supporte pas l'annotation « **@EJB** ».

Donc, il est nécessaire d'utiliser le nom JNDI avec «InitialContext» en chargeant les propriétés de connexion dans un objet «Hashtable».