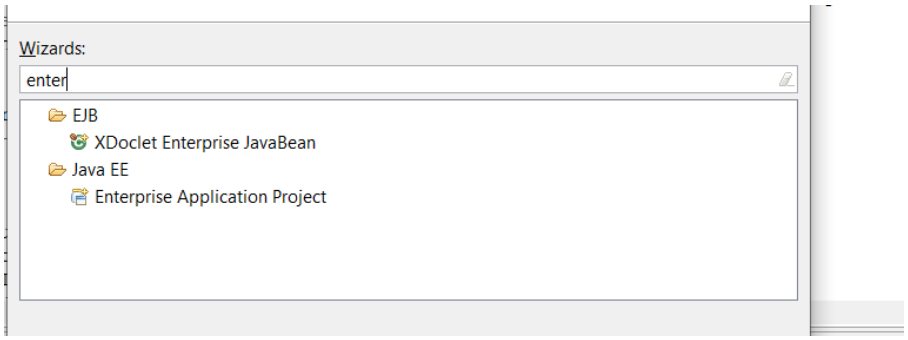
Workshops

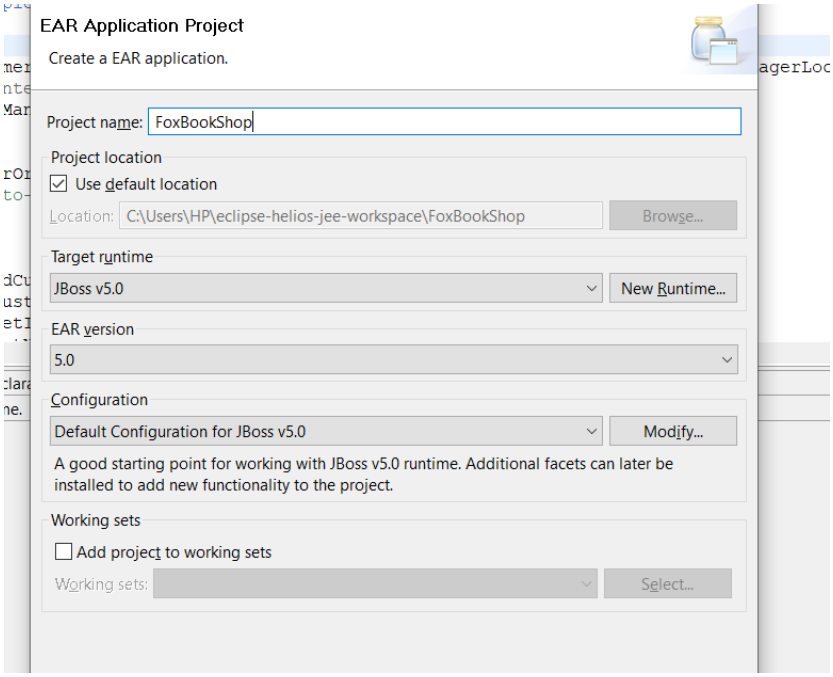
fait par : ZARHOUNI Fatima-Ezzahrae , 3A-GL2 , fatimeezzahrze_zarhouni@um5.ac.ma

TP 1- Création du projet d'application BookShop

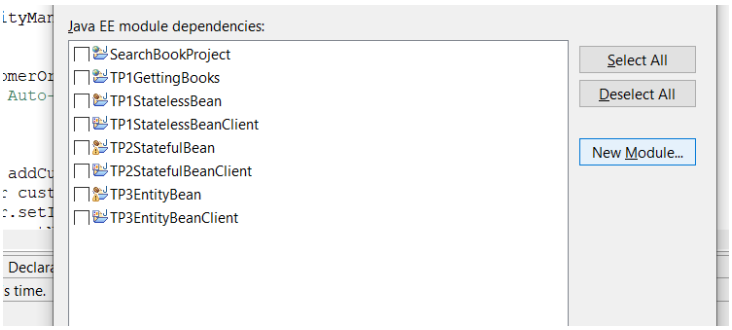
Exercice 1.1

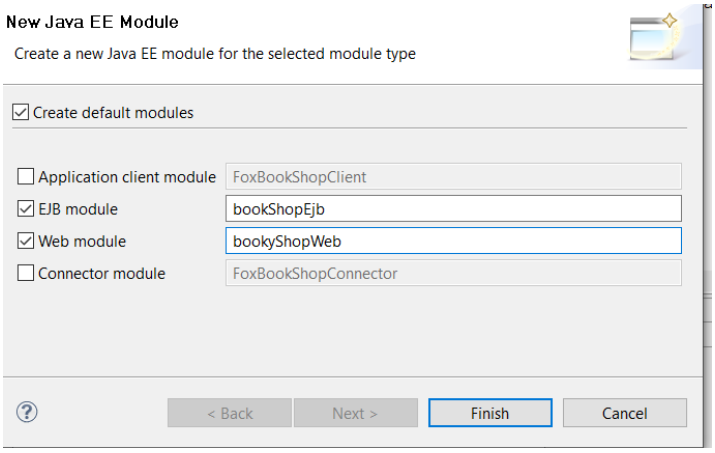
Créer une application d'entreprise :

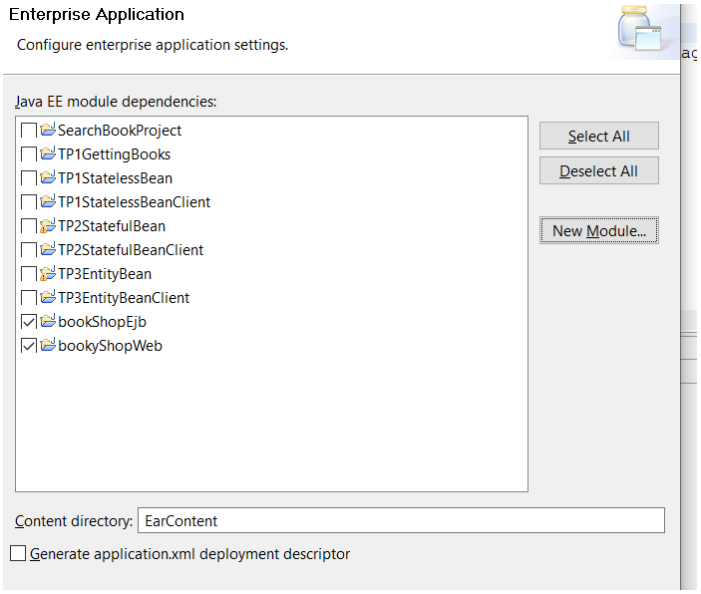




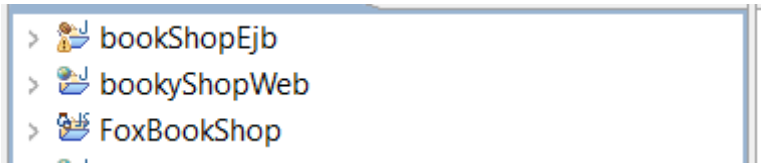
Cliquez sur Suivant pour sélectionner les modules qui seront créés dans votre application.
Cliquez sur new Module :







Dans l'explorateur de packages, vous obtiendrez quelque chose comme ceci :



Création des Classes :

Class Client :

```java
@Entity
public class Client implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int  identifiant ;
    private String nom ;
    private String prenom ;
    private String motDePasse ;
    @OneToMany(fetch = FetchType.LAZY , mappedBy = "Client")
    private ArrayList<Commande> commandes ;

    public Client(String nom, String prenom, String motDePasse) {
        super();
        this.nom = nom;
        this.prenom = prenom;
        this.motDePasse = motDePasse;
        this.commandes = new ArrayList<Commande>() ;
    }
    public int getIdentifiant() {
        return identifiant;
    }
    public void setIdentifiant(int identifiant) {
        this.identifiant = identifiant;
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public String getMotDePasse() {
        return motDePasse;
    }
    public void setMotDePasse(String motDePasse) {
        this.motDePasse = motDePasse;
    }
    public ArrayList<Commande> getCommandes() {
        return commandes;
    }
    public void setCommandes(ArrayList<Commande> commandes) {
        this.commandes = commandes;
    }
}
```

Class Command :

```java
@Entity
public class Commande implements  Serializable {
    @Id
    private int numeroCommande  ;

    @OneToMany
    private ArrayList<LigneDeCommande> ligneDeCommandes ;

    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="id_client")
    private Client clientId ;

    public Commande() {
        super();
        this.ligneDeCommandes = new ArrayList<LigneDeCommande>() ;
    }
    public int getNumeroCommande() {
        return numeroCommande;
    }
    public void setNumeroCommande(int numeroCommande) {
        this.numeroCommande = numeroCommande;
    }
    public ArrayList<LigneDeCommande> getLigneDeCommandes() {
        return ligneDeCommandes;
    }
    public void setLigneDeCommandes(LigneDeCommande ligneDeCommandes) {
        this.ligneDeCommandes.add(ligneDeCommandes) ;
    }
}
```

Class LigneDeCommande :

```java
@Entity
public class LigneDeCommande implements Serializable {
    private int quantite ;
    @ManyToOne() @JoinColumn()
    private Article article ;
    public LigneDeCommande(int quantite) {
        super();
        this.quantite = quantite;
        this.article = new Article() ;
    }
    public int getQuantite() {
        return quantite;
    }
    public void setQuantite(int quantite) {
        this.quantite = quantite;
    }
    public Article getArticle() {
        return article;
    }
    public void setArticle(Article article) {
        this.article = article;
    }

}
```

Class Article:

```java
@Entity
public class Article implements Serializable {
    @Id
    @GeneratedValue
    private int numeroArticle ;
    private int libelle ;
    private float prix ;

    public Article() {
        super();
    }
    public int getNumeroArticle() {
        return numeroArticle;
    }
    public void setNumeroArticle(int numeroArticle) {
        this.numeroArticle = numeroArticle;
    }
    public int getLibelle() {
        return libelle;
    }
    public void setLibelle(int libelle) {
        this.libelle = libelle;
    }
    public float getPrix() {
        return prix;
    }
    public void setPrix(float prix) {
        this.prix = prix;
    }
}
```

NOTE : J'ai utilisé les interfaces Serializable car j'ai rencontré l'erreur suivante liée au fichier de persistance lorsque je lance le serveur. J'ai essayé de la résoudre, mais je n'ai pas trouvé de solution ou n'en ai pas trouvé une par moi-même.

```
org.jboss.deployers.client.spi.IncompleteDeploymentException: Summary of incomplete deployments (SEE PREVIOUS ERRORS FOR DETAILS):

DEPLOYMENTS MISSING DEPENDENCIES:
  Deployment "jboss.j2ee:ear=FoxBookShop.ear,jar=bookShopEjb.jar,name=BookShopFacadeBean,service=EJB3" is missing the following dependencies:
    Dependency "<UNKNOWN jboss.j2ee:ear=FoxBookShop.ear,jar=bookShopEjb.jar,name=BookShopFacadeBean,service=EJB3>" (should be in state "Described", but is

DEPLOYMENTS IN ERROR:
  Deployment "persistence.unit:unitName=FoxBookShop.ear/bookShopEjb.jar#TP" is in error due to the following reason(s): org.hibernate.AnnotationException:
  Deployment "<UNKNOWN jboss.j2ee:ear=FoxBookShop.ear,jar=bookShopEjb.jar,name=BookShopFacadeBean,service=EJB3>" is in error due to the following reason(s)

        at org.jboss.deployers.plugins.deployers.DeployersImpl.checkComplete(DeployersImpl.java:863)
        at org.jboss.deployers.plugins.main.MainDeployerImpl.checkComplete(MainDeployerImpl.java:806)
        at org.jboss.system.server.profileservice.hotdeploy.HDScanner.scan(HDScanner.java:293)
        at org.jboss.system.server.profileservice.hotdeploy.HDScanner.run(HDScanner.java:221)
        at java.util.concurrent.Executors$RunnableAdapter.call(Unknown Source)
        at java.util.concurrent.FutureTask.runAndReset(Unknown Source)
```

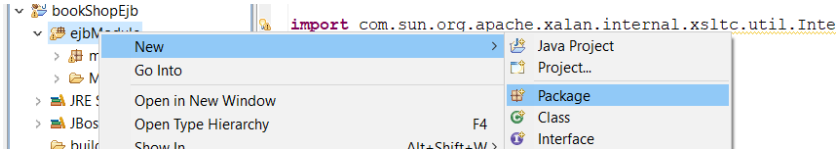
Parce que je n'ai pas pu utiliser le fichier de persistance, les résultats ne peuvent pas être téléchargés depuis une base de données vers le client, mais doivent être envoyés directement du les méthodes de EJB au client. Cela ne peut pas être fait sans sérialisation. -Lorsque je n'ai pas utilisé de fichier de persistance, en essayant d'obtenir le résultat, le serveur impose que les entités ne soient pas sérialisables-

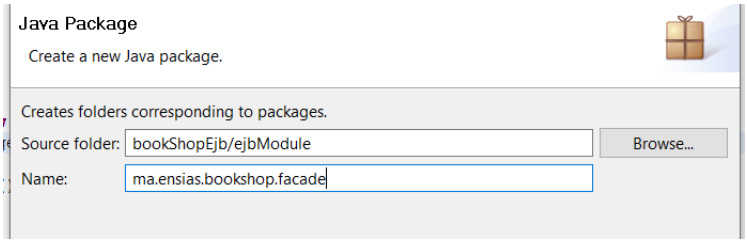
Normalement, le fichier de persistance ressemble à ceci :

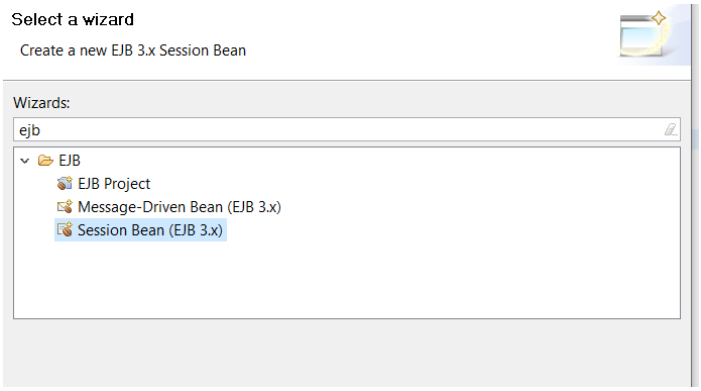
```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
version="1.0">
    <persistence-unit name="WorkShop">
        <jta-data-source>java:/DefaultDS</jta-data-source>
        <properties>
            <property name="hibernate.hbm2ddl.auto"
            value="create-drop" />
        </properties>
    </persistence-unit>
</persistence>
```

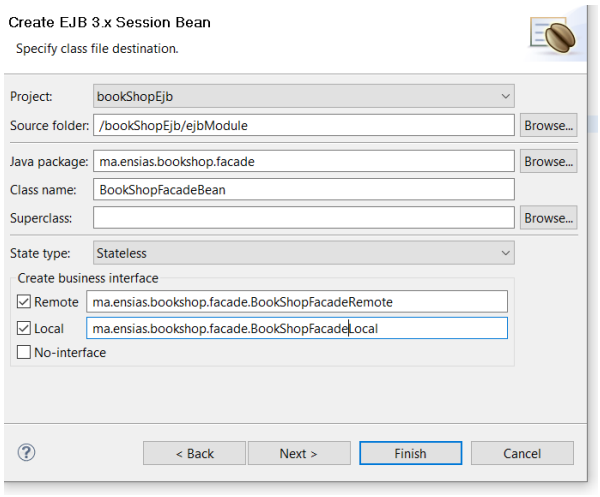
Exercice 1.2 - Créer l'ejb stateless BookShopFacade

Create the ejb package :










```java
@Stateless(mappedName="BookShopFacade")
public class BookShopFacadeBean implements BookShopFacadeRemote, BookShopFacadeLocal {
```

Exercice 1.3 - Créer les méthodes métiers de l'EJB BookShopFacade

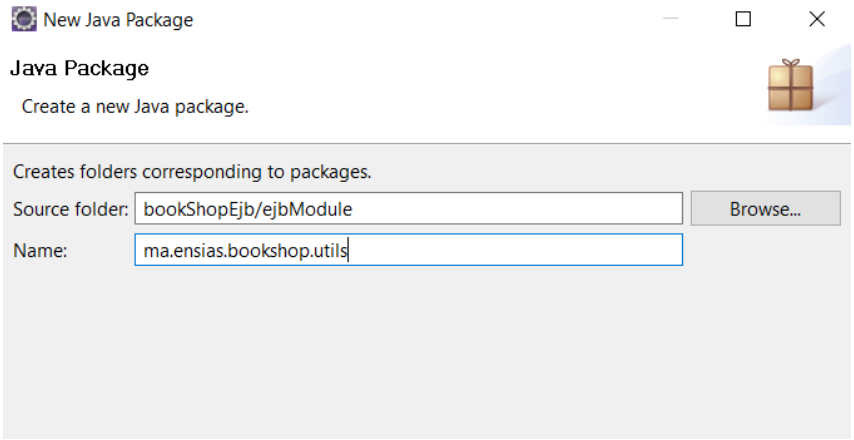
Interface Locale :

```java
@Local
public interface BookShopFacadeLocal {
    public Collection<Article> getAllArticlesFromFiliere(String filiere);
    public Collection<Article> getAllArticles();
    public void addArticle(Article article);
}
```

Interface Remote :

```java
@Remote
public interface BookShopFacadeRemote {
    public Collection<Article> getAllArticlesFromFiliere(String filiere);
    public Collection<Article> getAllArticles();
    public void addArticle(Article article);
}
```

Créez les packages ma.ensias.bookshop.utils :



Créez la classe BookShopBouchon :

```java
public class BookShopBouchon {
    private ArrayList<Article> articles ;

    public BookShopBouchon(){
        super() ;
        this.articles = new ArrayList<Article>() ;
    }

    public java.util.Collection<Article> getAllArticlesFromFiliere(String filiere){
        return null ;
    }
    public java.util.Collection<Article> getAllArticles(){
        //this.addArticles() ;
        return this.articles ;
    }
    public void addArticle(Article article){
        Article a = new Article() ;
        a.setNumeroArticle(article.getNumeroArticle()) ;
        a.setLibelle(article.getLibelle()) ;
        a.setPrix(article.getPrix());
        this.articles.add(a) ;
        printOut();
    }
    public void printOut(){
        for(Article a : articles){
            System.out.println("Libelle : "+a.getLibelle()+", prix : "+a.getPrix());
        }
    }
}
```

```java
bookShopFacade = new BookShopFacadeBean();
```

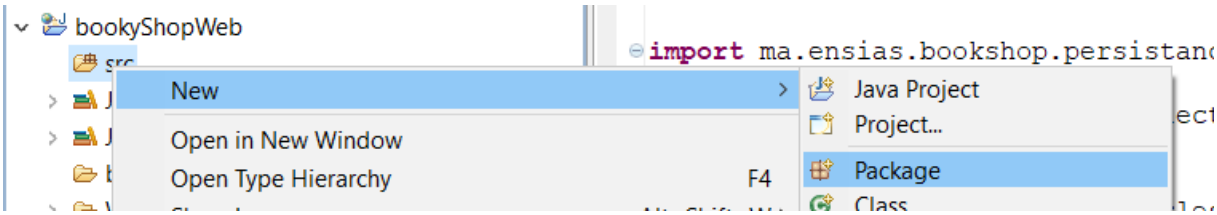
Cette ligne de code est correcte car nous travaillons en local. `bookShopFacade` ne peut pas être une instance d'une interface. Si nous faisons cela, nous obtiendrons une erreur de compilation.

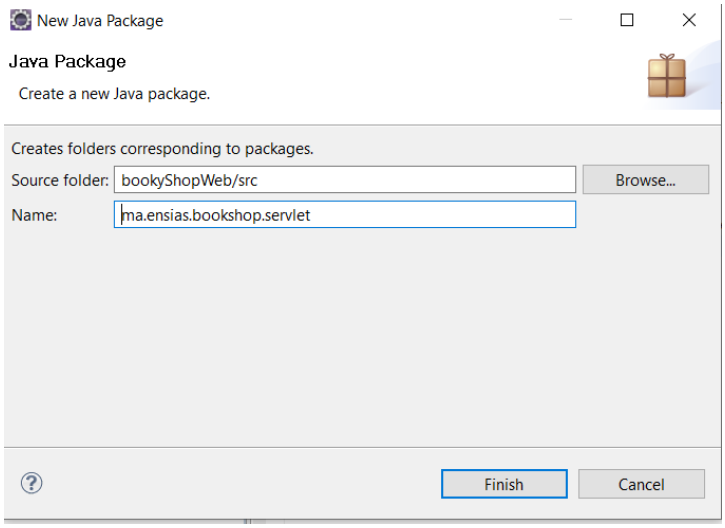
La class BookShopFacadeBean:

```java
public class BookShopFacadeBean implements BookShopFacadeRemote, BookShopFacadeLocal {

    private BookShopBouchon controller ;


    public BookShopFacadeBean() {
        controller = new BookShopBouchon();
    }

    @Override
    public Collection<Article> getAllArticlesFromFiliere(String filiere) {

        return null;
    }

    @Override
    public Collection<Article> getAllArticles() {
        return this.controller.getAllArticles();
    }

    @Override
    public void addArticle(Article article) {
        this.controller.addArticle(article);
    }

}
```

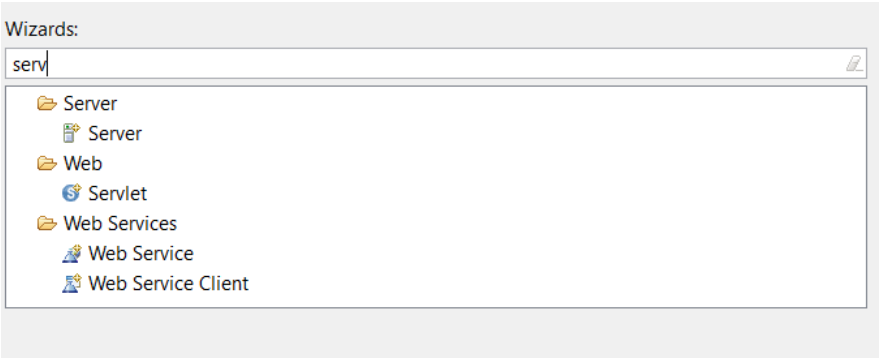
TP 2- Invocation locale et distante de l'EJB

Exercice 2.1 - Invocation de l'ejb en local (le client et l'EJB sont dans la même JVM) Au niveau du module Web :



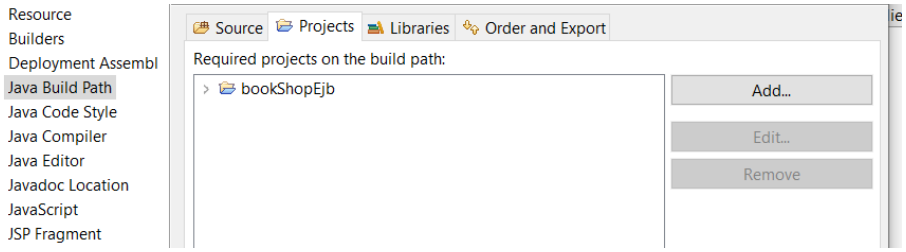


Création du servlet:



```java
package ma.ensias.bookshop.servlet;

import java.io.IOException;

/**
 * Servlet implementation class ListeArticlesServlets
 */
public class ListeArticlesServlets extends HttpServlet {
    private static final long serialVersionUID = 1L;

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

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse resp
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        // TODO Auto-generated method stub
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse res
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response
        // TODO Auto-generated method stub
    }

}
```

Ajoutez le projet EJB dans le java build path du module Web :

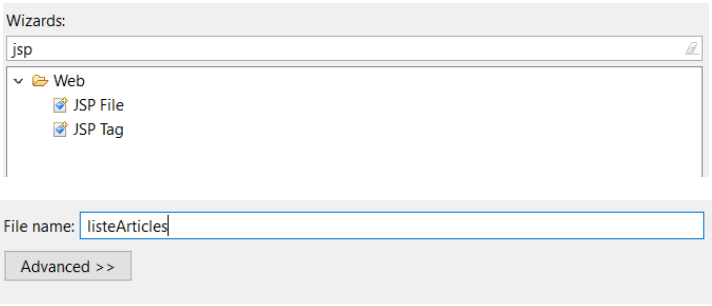


Si vous n'ajoutez pas le projet EJB au chemin, le servlet ne reconnaîtra pas l'interface locale lorsque vous essayez de l'importer.

```java
public class ListeArticlesServlets extends HttpServlet {
    private static final long serialVersionUID = 1L;
    @EJB
    private BookShopFacadeLocal bookShopFacade ;

    public ListeArticlesServlets() {
        super();
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
         this.processRequest(request, response) ;
    }
    protected void processRequest(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException{
        bookShopFacade = new BookShopFacadeBean();
        req.getSession().setAttribute("ArticleList", bookShopFacade.getAllArticles());
        req.getRequestDispatcher("listeArticles.jsp").forward(req, res);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
    }


}
```

*** Pour le moment, nous allons ajouter les articles de manière statique dans la classe BookShopBouchon, car l'envoi d'articles n'est pas le but de ce TP.**

Créez le fichier JSP :

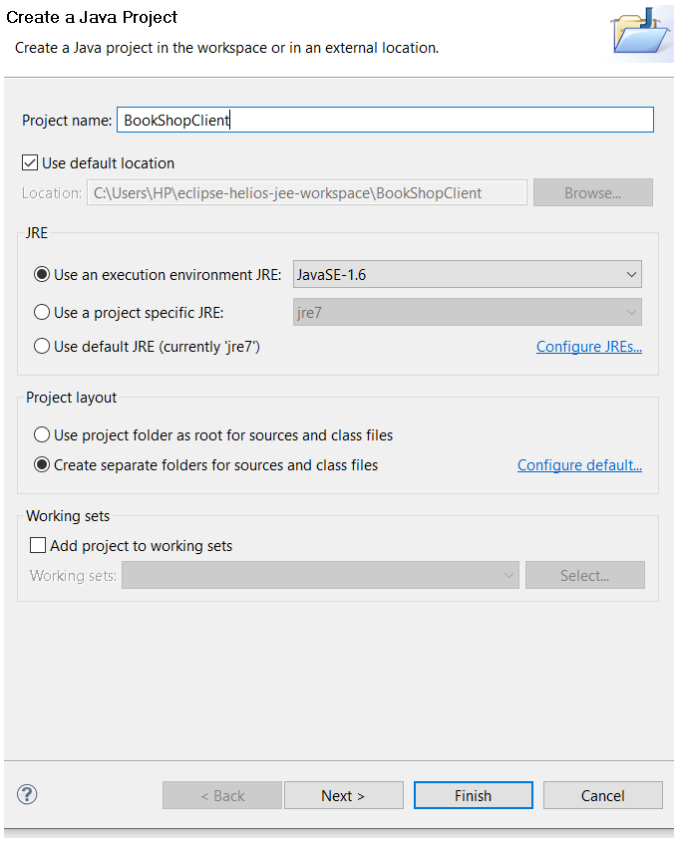


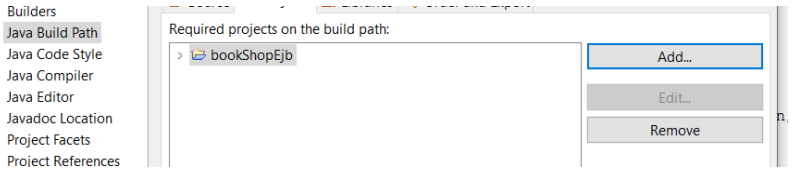
Le contenu du fichier JSP est :

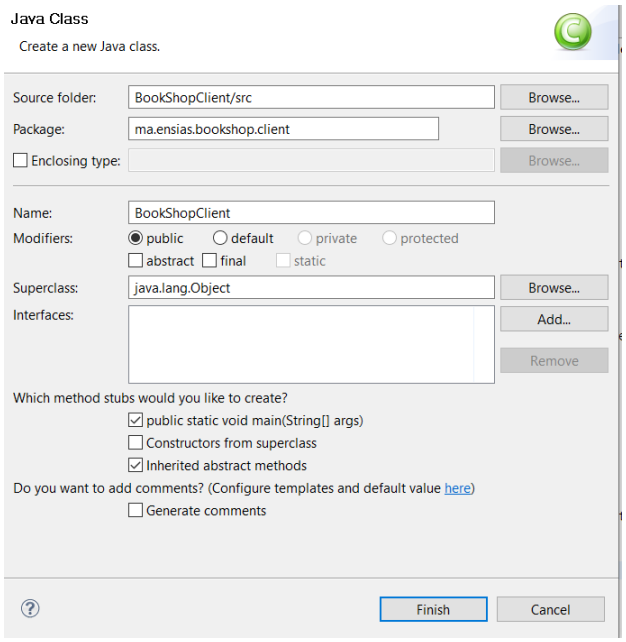
```jsp
<%@page import="ma.ensias.bookshop.persistance.Article"%>
<%@page import="java.util.List"%>
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Liste des Articles </title>
</head>
<body>
    <h1>
    Liste des Articles
    </h1>
    <ul>
        <%
            List<Article> articles = (List<Article>)request.getSession().getAttribute("ArticleList");
            if(articles != null){
                for(Article article : articles){
                    %><li> libelle : <%=  article.getLibelle() %> , prix : <%=  article.getPrix() %> </li><%
                }
            }
            else{
                %><li>Aucun Article disponible.</li><%
            }
        %>
    </ul>
</body>
</html>
```

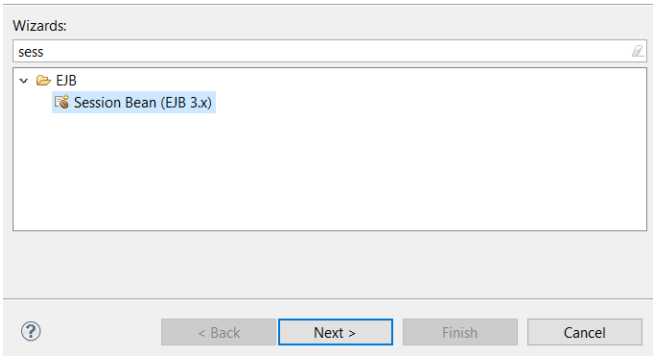
Lors de l'exécution du servlet :

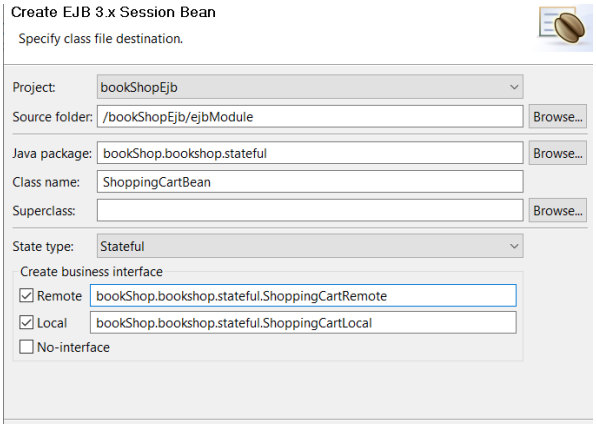
Exercice 2.2 - Invocation de l'ejb à distance (le client et l'EJB sont dans deux JVM distantes)


Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: BookShopClient
Use default location
Location: C:\Users\HP\eclipse-helios-jee-workspace\BookShopClient
Browse...
JRE
Use an execution environment JRE: JavaSE-1.6
Use a project specific JRE: jre7
Use default JRE (currently 'jre7')
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Finish
Cancel

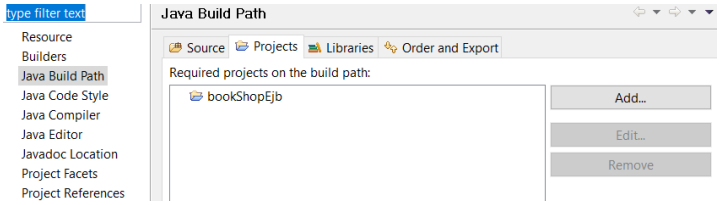

Builders
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project Facets
Project References
Required projects on the build path:
bookShopEjb
Add...
Edit...
Remove


Java Class
Create a new Java class.
Source folder: BookShopClient/src
Browse...
Package: ma.ensias.bookshop.client
Browse...
Enclosing type:
Browse...
Name: BookShopClient
Modifiers: public default private protected
abstract final static
Superclass: java.lang.Object
Browse...
Interfaces:
Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel





Ajoutez le projet EJB dans le Java build path Java du client :



La classe client :

```java
public class BookShopClient {

    public BookShopClient() {
        super();
    }

    public static void main(String[] args) {
        BookShopClient client = new BookShopClient() ;
        client.Do();

    }
    InitialContext getInitialContext() throws javax.naming.NamingException {
        Properties p = new Properties();
        p.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
        p.put(Context.URL_PKG_PREFIXES, "jboss.naming:org.jnp.interfaces");
        p.put(Context.PROVIDER_URL, "jnp://localhost:1099");
        return new InitialContext(p);
        }

    public void Do(){
        try {
            InitialContext ic = getInitialContext();
            BookShopFacadeRemote bookShopFacade = (BookShopFacadeRemote) ic.lookup("FoxBookShop/BookShopFacadeBean/remote");
            ArrayList<Article> articles = (ArrayList<Article>) bookShopFacade.getAllArticles() ;
            for(Article article : articles){
                System.out.println("libelle : "+ article.getLibelle() +" prix : "+article.getPrix()+ " .");
            }

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

    }
}
```

Cette classe a la méthode Do, cette méthode recherche l'interface distante et obtient tous les articles, puis les imprime tous dans la console.

Lors de l'exécution de la classe client :

```
<terminated> BookShopClient (1) [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (Nov 15, 2023 8:17:21 PM)
libelle : 1 prix : 1.0 .
libelle : 2 prix : 23.0 .
libelle : 3 prix : 678.0 .
libelle : 4 prix : 91.0 .
libelle : 5 prix : 6.0 .
```

TP 3- Création d'un EJB Session Stateful

Exercice 3.1 - Création de l'EJB Session Stateful

Ajout des méthodes métiers à l'EJB

L'interface Locale :

```java
import javax.ejb.Local;

@Local
public interface ShoppingCartLocal {
    public Commande getCommande();
    public void addLigneCommande(int numeroArticle);
    public void removeLigneCommande(int numeroArticle);
}
```

L'interface Remote :

```java
@Remote
public interface ShoppingCartRemote {
    public Commande getCommande();
    public void addLigneCommande(int numeroArticle);
    public void removeLigneCommande(int numeroArticle);
}
```

Le Bean stateful ShoppingCartBean :

```java
@Stateful
public class ShoppingCartBean implements ShoppingCartRemote, ShoppingCartLocal {

    private Commande commande ;

    public ShoppingCartBean() {
        // TODO Auto-generated constructor stub
    }
    @PostConstruct
    public void initialize(){
        commande = new Commande();
    }
    @Override
    public Commande getCommande() {
        System.out.println("Passage dans la Methode getCommande()");
        return null;
    }

    @Override
    public void addLigneCommande(int numeroArticle) {
        System.out.println("Passage dans la Methode addLignCommande");

    }

    @Override
    public void removeLigneCommande(int numeroArticle) {
        System.out.println("Passage dans la methode removeLignCommande");

    }

}
```

*** Pour l'instant prévoir l'impression d'un message comme implémentation de chacune des méthodes.**

Exercice 3.2 - Test de l'EJB Session stateful avec un client de test

La classe runnable :

```java
public class ShoppingCartClient {

    public static void main(String[] args) {
        ShoppingCartClient client = new ShoppingCartClient();
        client.Shop();
    }

    public ShoppingCartClient() {
        super();
        // TODO Auto-generated constructor stub
    }
    public void Shop(){
        try {
            InitialContext ic = getInitialContext();
            ShoppingCartRemote shoppingCart = (ShoppingCartRemote) ic.lookup("FoxBookShop/ShoppingCartBean/remote");
            shoppingCart.getCommande() ;
        } catch (Throwable e) {
            e.printStackTrace();
        }
    }
    InitialContext getInitialContext() throws javax.naming.NamingException {
        Properties p = new Properties();
        p.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
        p.put(Context.URL_PKG_PREFIXES, "jboss.naming:org.jnp.interfaces");
        p.put(Context.PROVIDER_URL, "jnp://localhost:1099");
        return new InitialContext(p);
        }
}
```

Cette classe a la méthode Shop, cette méthode recherche l'interface distante et invoque getCommand. Comme getCommand ne fait que imprimer dans la console pour le moment, le résultat sera visible dans la console du serveur.

Donc Lors de l'exécution de la classe ShoppingCartClient, au niveau du serveur, nous avons ce qui suit imprimer :

```
20:48:27,556 INFO  [STDOUT] Passage dans la Methode getCommande()
```

Exercice 3.3 - Invocation de l'EJB Session stateful à partir d'une servlet

```java
public class AddArticleToShoppingCart extends HttpServlet {
    private static final long serialVersionUID = 1L;
    @EJB
    private BookShopFacadeRemote bookShopFacade ;
    public AddArticleToShoppingCart() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        this.processRequest(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    }

    InitialContext getInitialContext() throws javax.naming.NamingException {
        Properties p = new Properties();
        p.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
        p.put(Context.URL_PKG_PREFIXES, "jboss.naming:org.jnp.interfaces");
        p.put(Context.PROVIDER_URL, "jnp://localhost:1099");
        return new InitialContext(p);
     }
    protected void processRequest(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException{
        try{
            InitialContext ic = getInitialContext();
            bookShopFacade = (BookShopFacadeRemote) ic.lookup("FoxBookShop/BookShopFacadeBean/remote");
            Article a = new Article();
            Article b = new Article();
            a.setLibelle(6);
            a.setPrix(89);
            bookShopFacade.addArticle(a);
            b.setLibelle(7);
            b.setPrix(100);
            bookShopFacade.addArticle(b);
            //req.getRequestDispatcher("addArticleToShoppingCart.jsp").forward(req, res);

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

La méthode processRequest ajoute statiquement deux articles constants en utilisant l'EJB, Elle est invoquée lorsque la méthode doGet est appelée.

Étant donné que le bean BookShopFacadeBean est stateless, nous pouvons exécuter la servlet ListeArticle pour obtenir tous les articles.