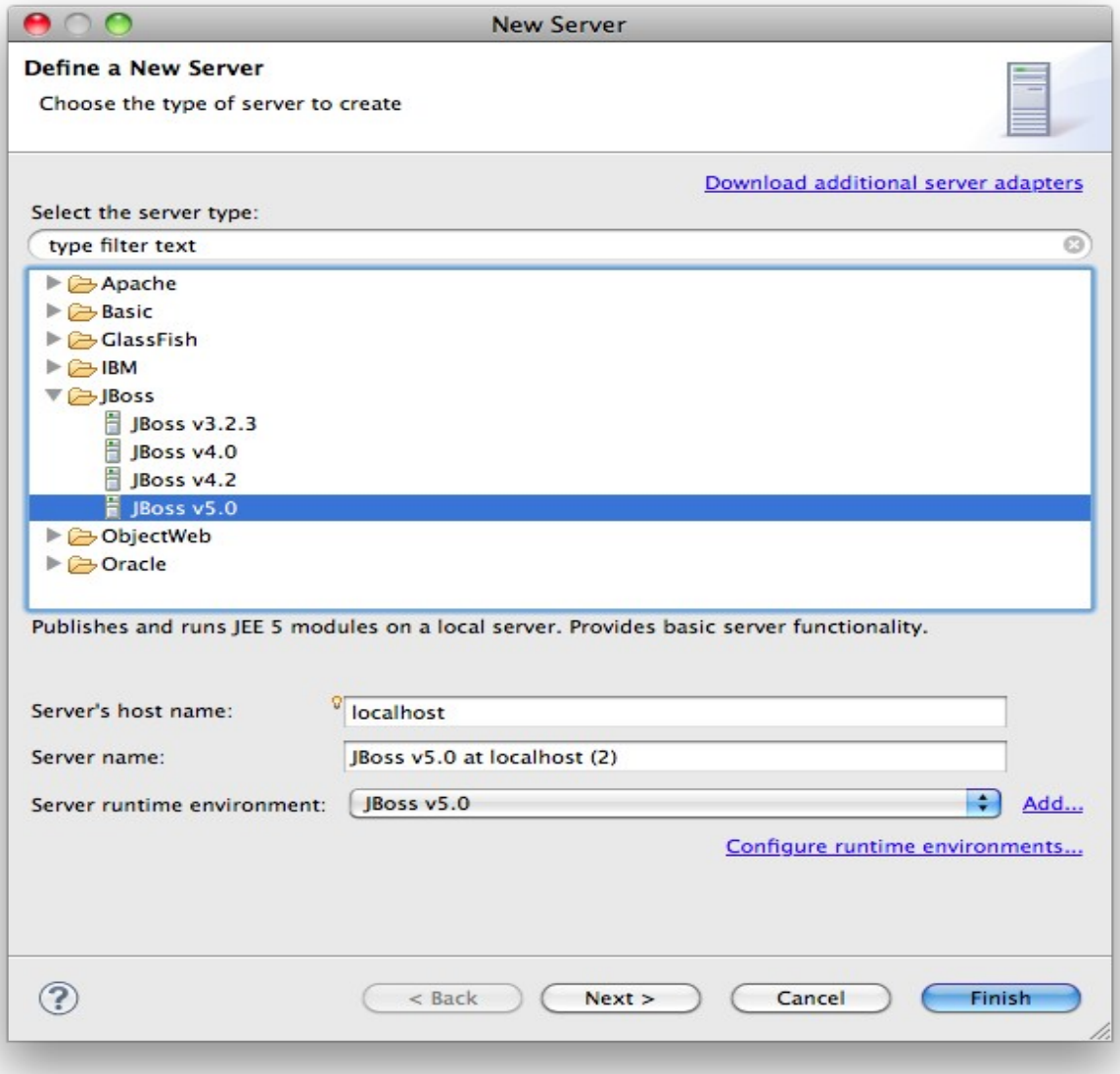
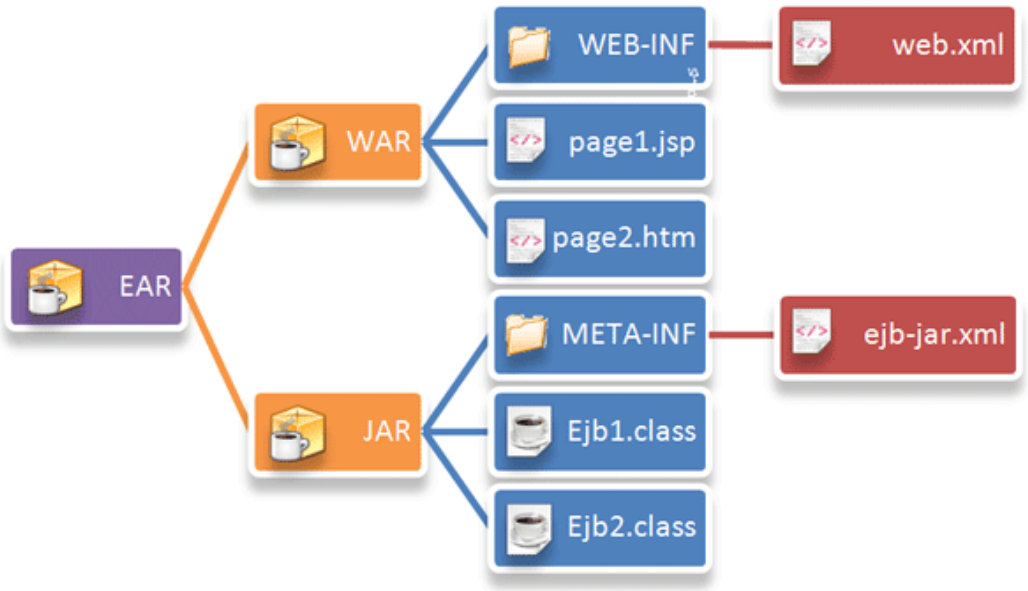
T.P. 4
EJB3, Serveur d'application JBoss

1. *Installation d'un serveur d'application JBoss*:

- télécharger l'archive du serveur JBoss à l'adresse: http://sourceforge.net/projects/jboss/files/JBoss/JBoss-5.0.0.GA/jboss-5.0.0.GA.zip/download
- décompresser l'archive téléchargé afin de créer le répertoire contenant le serveur JBoss: *jboss-5.0.0.GA/*
- Configurer un nouveau serveur JBoss 5.0 dans Eclipse (même procédure que la création d'un serveur Tomcat dans Eclipse):



2. *Développement et déploiement dans JBoss d'un EAR (EJBs + Servlets)* :



Une application d'entreprise est généralement packagée dans une archive de type EAR composée de deux parties :

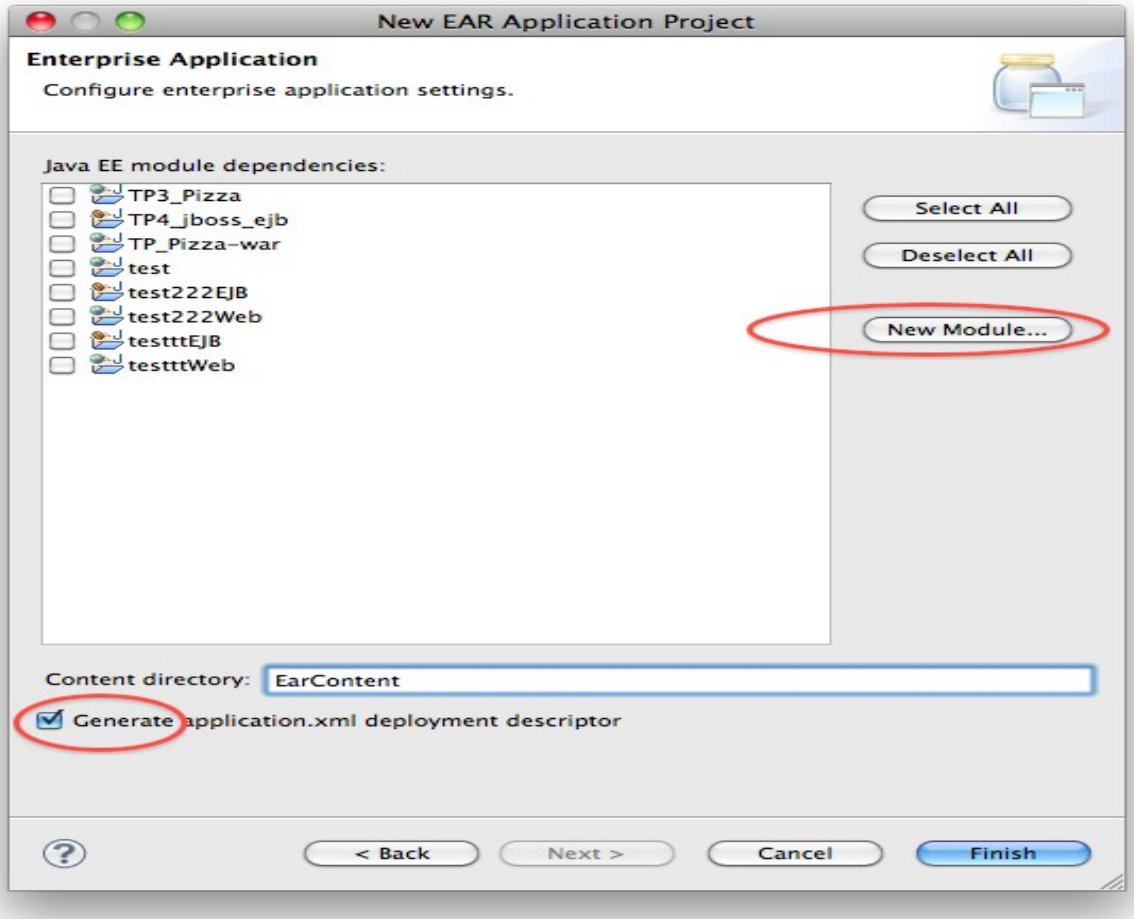
- Une couche EJBs packagée dans une archive de type JAR dédiée.
- Une couche WEB (Servlet, JSP,...) sous la forme d'une webapp packagée dans une archive de type WAR.

Dans la section suivante, on va utiliser Eclipse pour créer un projet «*TP4_Pizza*» de type EAR composé de deux (sous)-projets:

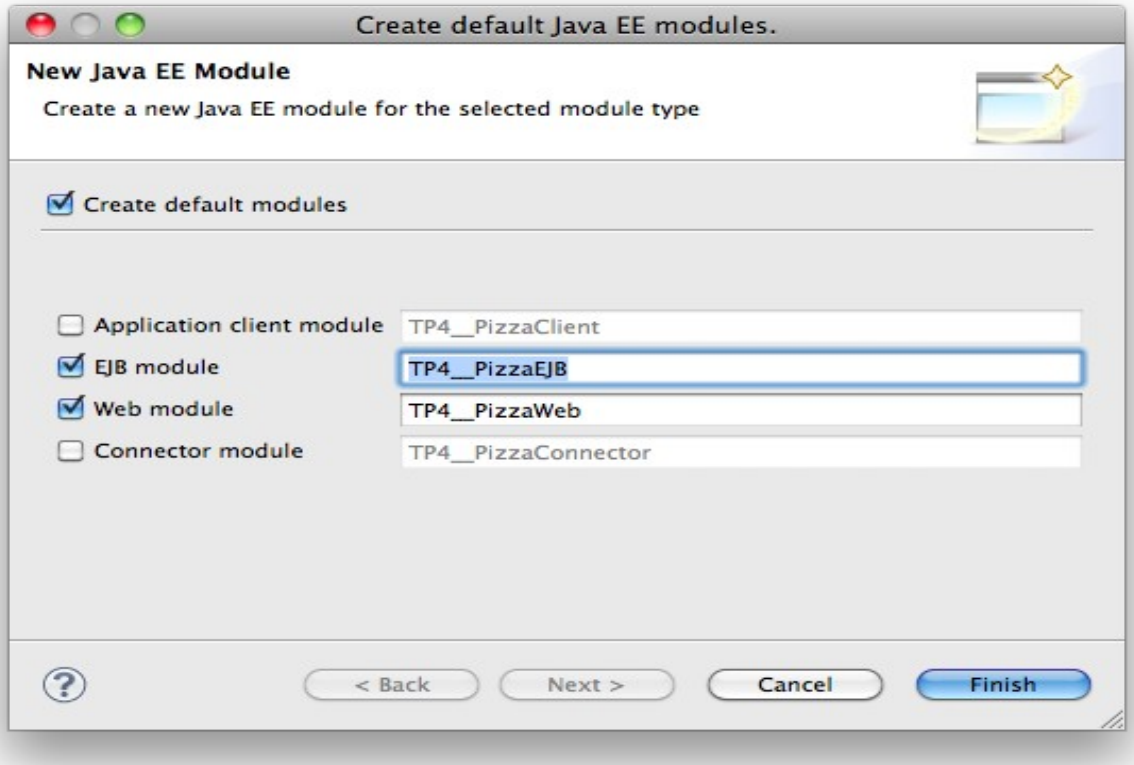
- projet du module WEB : «*TP4_PizzaWeb*» qui va contenir les Servlets et JSPs (.war).
- projet du module EJB : «*TP4_PizzaEJB*» qui va contenir les EJBs (.jar).

2.1. Création du projet (EAR):

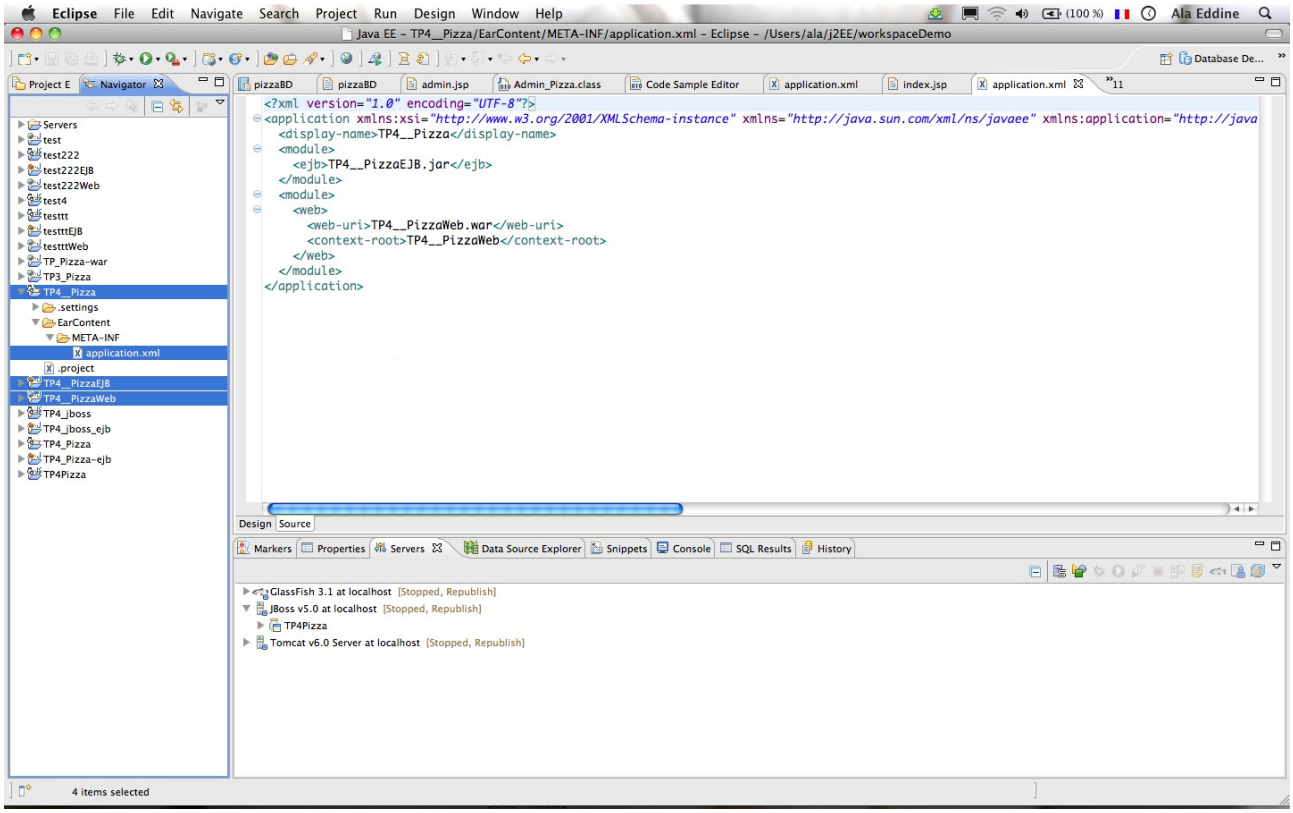
Dans la vue 'Explorateur de projet', cliquer droit puis sélectionner ***Enterprise Application Project***:
Vous obtenez l'écran de création d'un projet java Edition Entreprise (Java EE).
Donnez un nom au projet: *TP4_Pizza*.
Un serveur d'applications cible (Target runtime) est ici associé par défaut : JBoss 5.
Cliquez sur bouton 'Next': vous obtenez l'écran «New EAR Application Project» ci-dessous.
Cocher la case 'Generate Deployment Descriptor': Cette case permet de générer le fichier de configuration de l'EAR: application.xml. Ce fichier de configuration est le descripteur de déploiement pour l'EAR.
Notez le répertoire dans lequel le projet sera créée: EarContent (par défaut).



Eclipse vous demande les modules que vous souhaitez développer. Cliquez sur le bouton 'New Module'.
La spécification J2EE propose 4 modules (voir capture d'écran ci-dessous). Ici nous choisissons les 2 modules EJB et WEB: TP4_PizzaEJB et TP4_PizzaWeb:



Cliquez sur Finish. Les différents projets apparaissent dans la vue ’Project Explorer’ d’Eclipse.



Eclipse a créé le fichier de configuration ***application.xml*** de l’archive EAR dans «*TP4_Pizza/EarContent/META-INF*». C’est ce qu’on appelle le descripteur de déploiement de l’archive EAR. A travers la balise xml ***’<module>’*** du fichier application.xml, l’EAR précise quels sont les différents modules qu’il héberge (ici 2 modules). Chaque module possède un identifiant unique.
Notez la balise xml ***’<contexte-root>’*** du module WEB (balise <*web*>), la valeur du '*contexte-root'* permet de définir l'URL d'accès au module WEB de l'application. Ici, une fois l' EAR déployé dans le serveur d’applications JBoss, l’URL «*http://localhost:8080/**TP4_PizzaWeb**/*» permettra de d'accéder à l’application.

Modifier la valeur du ***’contexte-root’*** pour pouvoir utiliser l'URL suivante**:**
*http://localhost:8080/**TP4_Pizza**/ (au lieu de http://localhost:8080/**TP4_PizzaWeb**/).*

Dans la suite du TP, on va développer dans le projet «*TP4_PizzaEJB*» des EJBs de type Entity et Session. Ensuite, pour pouvoir tester ces EJBs, on va créer dans le projet «*TP4_PizzaWeb*» des Servlets qui utilisent les EJBs du projet «*TP4_PizzaEJB*».

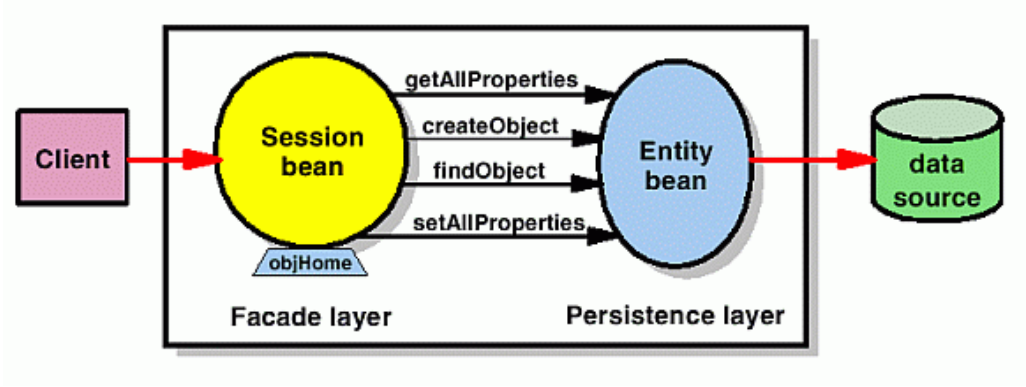
2.2. Partie EJB (projet *TP4_PizzaEJB*):

2.2.1. Développement des EJBs Entity: Créer les EJBs Entity: *Pizza*, *Stock* et *commande* correspondant aux tables *PIZZA, STOCK et COMMANDE* de la base de données *PizzaDB* (créée lors du TP3), pour cela il suffit de copier le code des deux EJBs suivantes dans le répertoire *TP4_PizzaEJB/ejbModule* et de générer les getter et setter. Ensuite, en s'inspirant de l'EJB Entity *Stock*, créer l'EJB Entity *Commande*.

```
package EntityBeans ;
import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "PIZZA")
public class Pizza implements Serializable {

    @Id
    @Basic(optional = false)
    @Column(name = "PIZZA_ID")
    private String pizzaId;

    @Column(name = "PRIX")
    private Integer prix;

    @OneToMany(mappedBy = "pizza")
    private Collection<Commande> commandes;

    @OneToMany(mappedBy = "pizza")
    private Collection<Stock> stocks;

    public Pizza() {}
...
}
```

```
package EntityBeans;
import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "STOCK")
public class Stock implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "STOCK_ID")
    private Integer stockId;

    @Column(name = "QUANTITE")
    private Integer quantite;

    @JoinColumn(name = "PIZZA_ID", referencedColumnName = "PIZZA_ID")
    @ManyToOne
    private Pizza pizza;

    public Stock() {}
    ...
}
```

2.2.2. Déploiement des EJBs Entity:



Il faut d'abord déployer une *DataSource* et une *unité de persistence*, c'est à dire mettre en place un serveur JNDI qui va faire la liaison entre la base de données et les EJBs Entity.

2.2.2.1. Déploiement d'une DataSource dans JBoss:

Une DataSource (source de données) est une fabrique de connexions vers une base de donnée, on va voir dans la suite qu'une DataSource est déployée dans un serveur J2EE pour faire le lien entre la base de données et les EJBs Entity. Dans ce TP, on va créer une DataSource nommé «*PizzaDS*» qui pointe vers la base de données «*PizzaDB*» (du TP3). Pour configurer une DataSource dans un serveur JBoss, il faut suivre les étapes suivantes:

- Copiez les 2 jars *derby.jar* et «*jboss-5.0.0.GA/docs/examples/varia/derby-plugin.jar*» dans le répertoire contenant les jars du serveur JBoss: «*jboss-5.0.0.GA/server/default/lib*»,

- Dans JBoss, le répertoire d'exemples d'applications «*jboss-5.0.0.GA/docs/examples/jca/*» contient des fichiers «Template» pour aider les développeurs à écrire les fichiers de configuration des DataSources dans JBoss (un fichier «Template» pour chaque type de base de données). Puisque «*PizzaDB*» est une base de données de type Derby, il faut d'abord copier le fichier «Template» *derby-ds.xml* dans le répertoire de déploiement du serveur JBoss: «*jboss-5.0.0.GA/server/default/deploy*». Ensuite, afin de créer une nouvelle DataSource «*PizzaDS*» qui pointe vers la base de données «*PizzaDB*», il faut modifier le fichier *derby-ds.xml* copié avec la configuration de base «*PizzaDB*» de la façon suivante:

```
...
<jndi-name>PizzaDS</jndi-name>
<connection-url>jdbc:derby:[à remplacer avec votre URL]/PizzaDB;create=true</connection-
url>
<driver-class>org.apache.derby.jdbc.EmbeddedDriver</driver-class>
<user-name>pizza</user-name>
<password>pizza</password>
...
```

2.2.2.2. Création d'une unité de persistance (fichier persistence.xml):

Dans le répertoire «*TP4_PizzaEJB/ejbModule/META-INF*» ajoutez le fichier *persistence.xml* ci-dessous. Ce fichier permet de créer une unité de persistance (balise <*persistence-unit*>) nommé "*Pizza-ejbPU*". Une unité de persistance permet de faire le lien entre les EJBs Entity et la DataSource «*PizzaDS*» (balise <*jta-data-source*>) et donc de mettre en place un mapping objet-relationnel entre les EJBs Entity et les tables de la base de données (l'accès à la base de données «*PizzaDB*» étant effectué à travers la DataSource «*PizzaDS*» créée dans 2.2.2.1).

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" 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">
  <persistence-unit name="Pizza-ejbPU" transaction-type="JTA">
    <jta-data-source>java:/PizzaDS</jta-data-source>
    <properties/>
  </persistence-unit>
</persistence>
```

2.2.2.3. Déploiement des EJBs Entity dans JBoss:

Cliquer droit sur le projet (EAR) *TP4_Pizza*, puis sélectionner *Run As → Run on Server*. Ensuite, vérifier dans l'onglet «Console » qu'il n' y a pas d'erreurs lors de l'instanciation des EJBs Entity par le serveur JBoss.

2.2.3. Développement des EJBs Session (pour gérer les EJBs Entity):

Dans ce TP, les EJBs Session vont jouer le même rôle que les Classes «Façades» développées en JDBC lors des TPs précédents. En effet, dans la suite, on va créer un EJB Session pour gérer la persistance (en base de donnée) de chaque EJB Entity.

Par exemple, pour gérer la persistance de l'EJB Entity *Pizza*, il faut développer un EJB Session *PizzaFacade*. Le développement d'un EJB Session *PizzaFacade* nécessite la création d'au moins une interface «Local» *PizzaFacadeLocal.java* et d'une classe *PizzaFacade.java* qui implémente cette interface.

Voici le code de *PizzaFacadeLocal.java* et *PizzaFacade.java* :

```
package sessionBeans;
import java.util.List;
import javax.ejb.Local;

@Local
public interface PizzaFacadeLocal {
    void create(Pizza pizza);
    void update(Pizza pizza);
    void remove(Pizza pizza);
    Pizza find(Object id);
    List<Pizza> findAll();
}
```

```java
package sessionBeans;

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

@Stateless
public class PizzaFacade implements PizzaFacadeLocal {
   @PersistenceContext(unitName="Pizza-ejbPU")
    private EntityManager entityMgr;

    public void create(Pizza pizza) {
        entityMgr.persist(pizza);
    }

    public void update(Pizza pizza) {
        entityMgr.merge(pizza);
    }

    public void remove(Pizza pizza) {
        entityMgr.remove(entityMgr.merge(pizza));
    }

    public Pizza find(Object id) {
        return entityMgr.find(Pizza.class, id);
    }

    public List<Pizza> findAll() {
        return entityMgr.createQuery("select object(o) from Pizza as o").getResultList();
    }
}
```

Pendant le déploiement de l'EAR contenant l'EJB Session *PizzaFacade*, l'utilisation de l'annotation `@PersistenceContext` dans la classe *PizzaFacade.java* permet au conteneur d'EJB d'injecter une référence de l'instance de l'*EntityManager* dans la variable *entityMgr*. C'est cette instance de l'*EntityManager* qui permet à un EJB Session de gérer la persistance des EJBs Entity.

En suivant la même démarche, créer les Deux autre EJBs Session: *StockFacade* et *CommandeFacade.*

2.2.4. Déploiement des EJBs Session:

Cliquer droit sur le projet EAR *TP4_Pizza*, puis sélectionner Run As → Run on Server. Vérifier dans l'onglet «Console» qu'il n' y a pas d'erreurs affichées lors de l'instanciation des EJBs Session par le serveur JBoss.

2.3. Partie WEB (Servlets, JSPs) (projet *TP4_PizzaWeb*):

Pour tester les EJBs déployés, on va utiliser les Servlets développés lors des TPs précédents. En effet, dans ce TP, les Servlets vont jouer le rôle des clients des EJB, ce sont des clients de type «Client Locale» (interface «*XXXLocal*») car ici les Servlets sont déployées dans le même serveur que les EJBs.

Dans une Servlet, pour utiliser une référence (locale) sur un EJB Session, il faut déclarer une «variable» de type l'interface «Local» de l'EJB annoté par l'annotation `@EJB` de la façon suivante:

```
@EJB
private PizzaFacadeLocal pizzaFacade;
```

Grace à l'annotation `@EJB`, pendant le déploiement de l'EAR (contenant EJBs + Servlets), le conteneur d'EJB qui va se charger d'instancier les EJBs Session, puis le conteneur WEB va injecter la référence de chaque EJB Session dans les Servlets qui l'utilisent. Voici des exemples d'appels des trois «EJBs Session» à partir de la Servlet `Commande_Pizza`:

```
import sessionBeans.CommandeFacadeLocal;
import sessionBeans.PizzaFacadeLocal;
import sessionBeans.StockFacadeLocal;
import EntityBeans.Commande;
import EntityBeans.Pizza;
import EntityBeans.Stock;

public class Commande_Pizza extends HttpServlet {
    @EJB
    private CommandeFacadeLocal commandeFacade;
    @EJB
    private StockFacadeLocal stockFacade;
    @EJB
    private PizzaFacadeLocal pizzaFacade;

        protected void processRequest(HttpServletRequest request, HttpServletResponse response)...
        {
                List lStock = stockFacade.findAll();
                ...
                Pizza pizza = pizzaFacade.find(type);
                ...
                commandeFacade.create(commande);

        }

}
```

- Créer les Servlets *Admin_Pizza* et *Commande_Pizza* dans le (sous)-projet *TP4_PizzaWeb*. Dans un premier temps, copier/coller le contenu de ces deux Servlets à partir du TP3. Ensuite, adapter le code de chaque Servlet pour utiliser les EJBs déployés précédemment.

- Déployer l'EAR de l'application *TP4_Pizza*, puis tester les Servlets *Admin_Pizza* et *Commande_Pizza.*

- Compléter **uniquement** le code de la Servlet ***Commande_Pizza (sans rien ajouter aux EJBs)*** pour implémenter les fonctionnalités suivantes :

1. Refuser les commandes pour un type de pizza inexistant et refuser les commandes pour un stock insuffisant.
2. *Mettre à jour le stock après chaque commande.*

- Enfin, en utilisant l'EJB Session *CommandeFacade*, créer la JSP qui affiche la liste de commandes effectuées (sous la forme d'un tableau HTML).