Petstore - EJB3 Entity

Le tutorial aborde les différentes étapes pour concevoir et implémenter les EJB3 entity dans l'application Petstore.
L'application est déployée sous JBoss 5.1.0.GA.

I. Analyse et conception

1. **Le modèle Objets**
2. **Modèle physique de données**
3. **Mapping Objets/Tables**

II. Implémentaion

1. **Les classes entity**
2. **Le fichier persistence.xml**
3. **Les DAO**
4. **Les EJB3 session**
5. **Le client JSF**

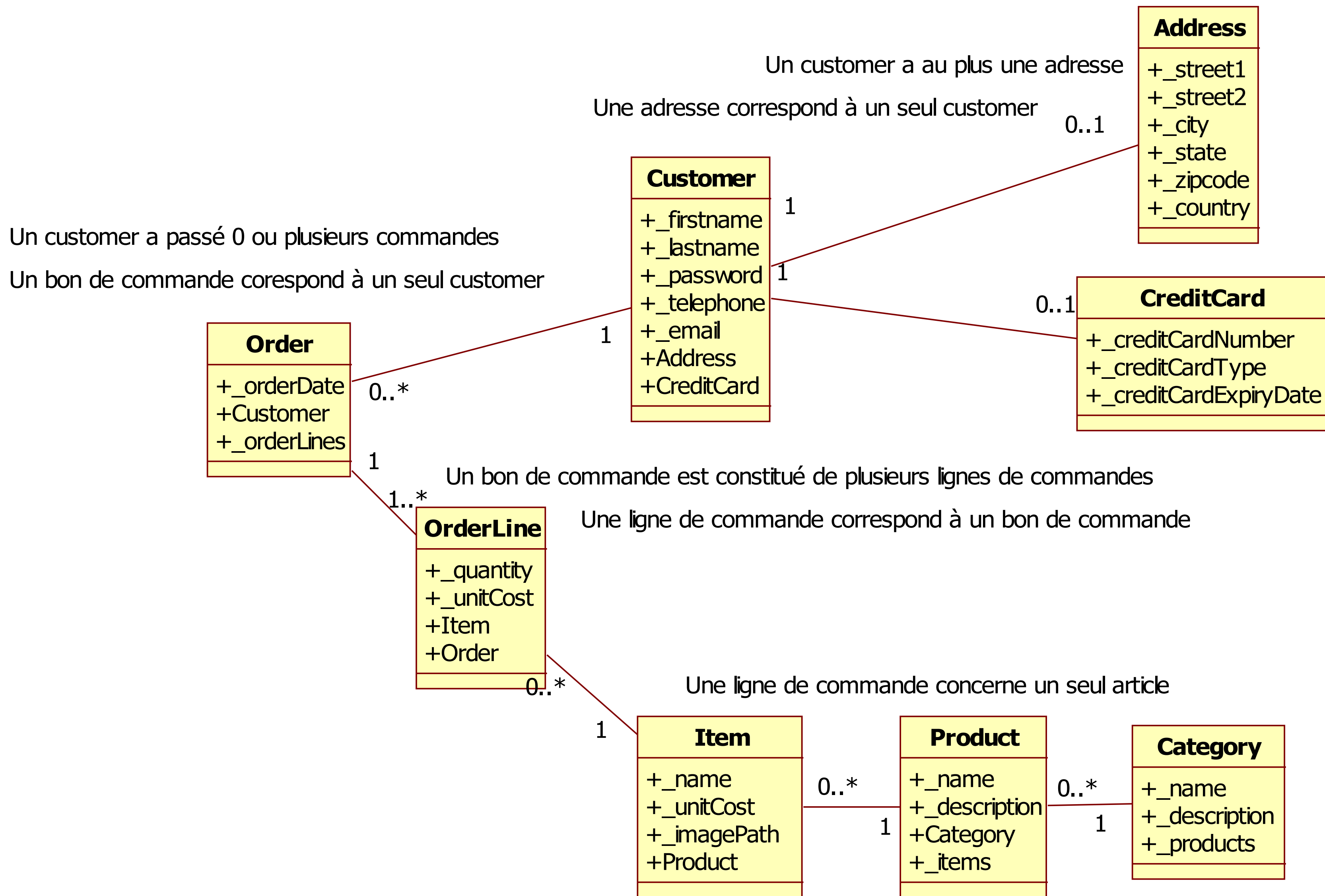
J. HILDEBRAND

I. Analyse et conception

1. Le modèle objets

- **Les objets métier**
- **Les relations entre objets**
- **Les cardinalités**

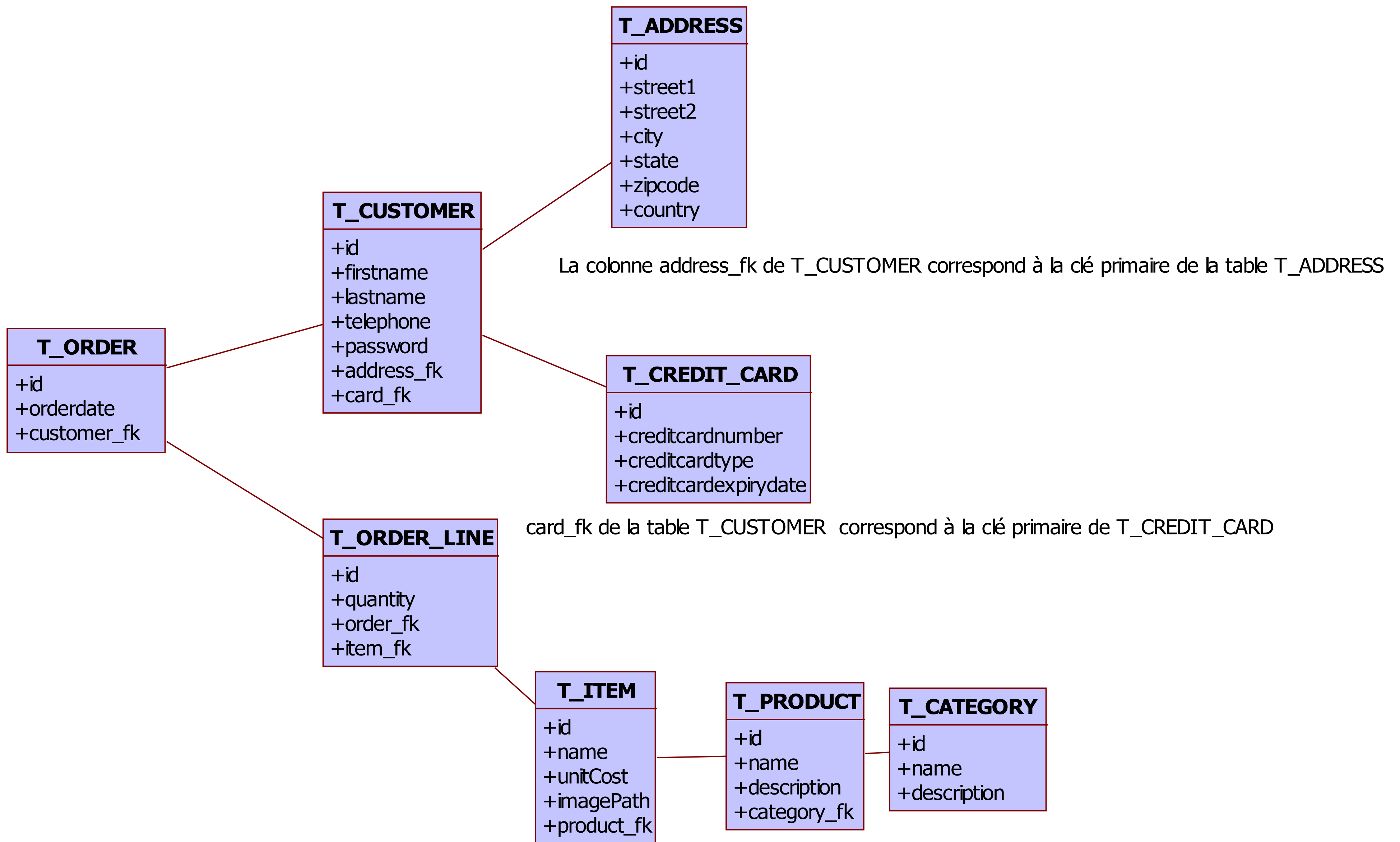
Le **modèle objets** est constitué de **3 domaines** : les clients, les bons de commande, le catalogue.


Address
+_street1
+_street2
+_city
+_state
+_zipcode
+_country
Un customer a au plus une adresse
Une adresse correspond à un seul customer
0..1
Customer
+_firstname
+_lastname
+_password
+_telephone
+_email
+Address
+CreditCard
1
1
0..1
CreditCard
+_creditCardNumber
+_creditCardType
+_creditCardExpiryDate
Un customer a passé 0 ou plusieurs commandes
Un bon de commande corespond à un seul customer
Order
+_orderDate
+Customer
+_orderLines
1
0..*
1
1..*
Un bon de commande est constitué de plusieurs lignes de commandes
Une ligne de commande correspond à un bon de commande
OrderLine
+_quantity
+_unitCost
+Item
+Order
0..*
1
Une ligne de commande concerne un seul article
Item
+_name
+_unitCost
+_imagePath
+Product
0..*
1
Product
+_name
+_description
+Category
+_items
0..*
1
Category
+_name
+_description
+_products

2. Modèle physique de données

- **Les tables**

- **Les relations entre tables (contraintes d'intégrité)**



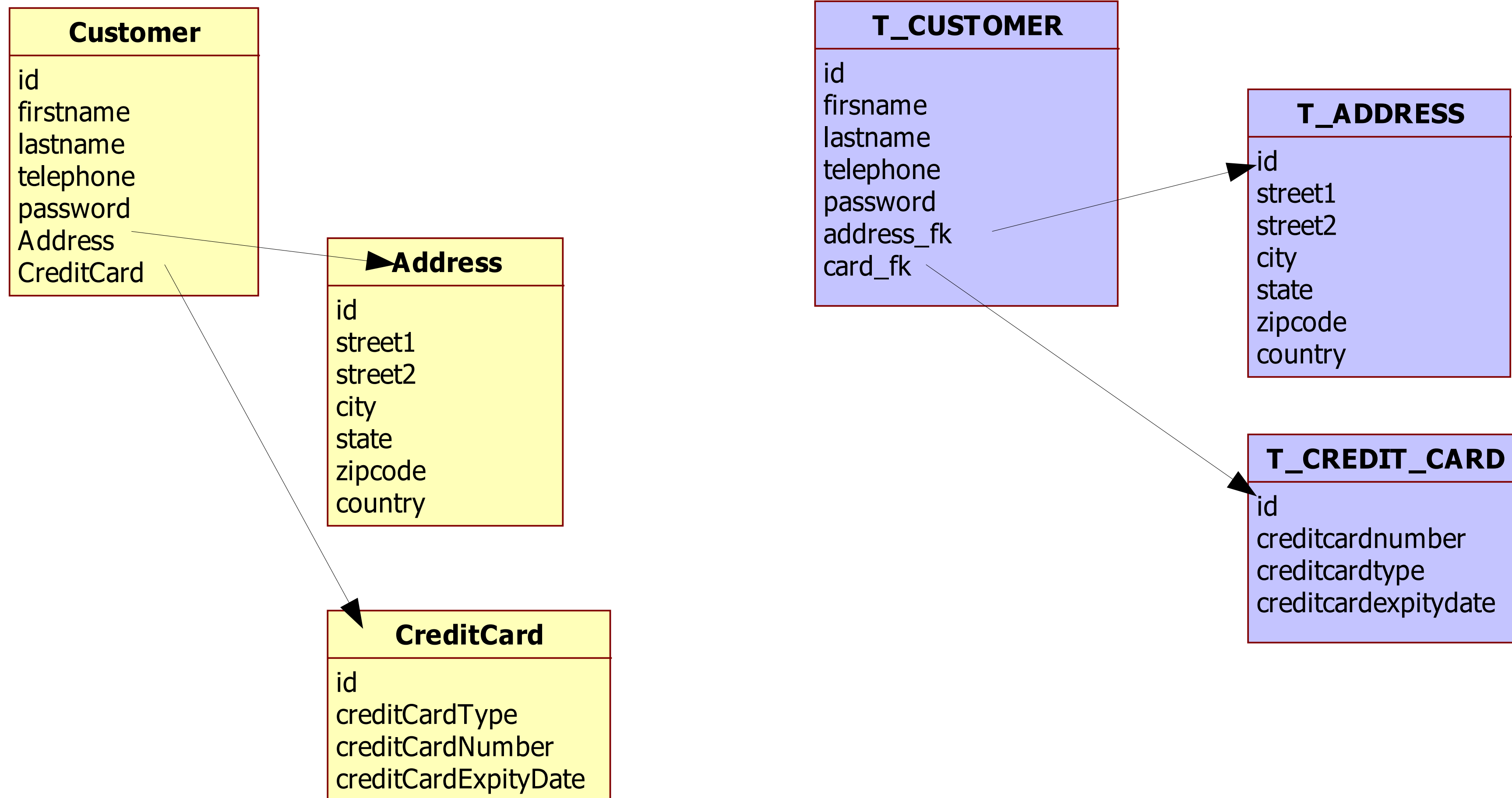
T_ADDRESS
+id
+street1
+street2
+city
+state
+zipcode
+country
T_CUSTOMER
+id
+firstname
+lastname
+telephone
+password
+address_fk
+card_fk
La colonne address_fk de T_CUSTOMER correspond à la clé primaire de la table T_ADDRESS
T_ORDER
+id
+orderdate
+customer_fk
T_CREDIT_CARD
+id
+creditcardnumber
+creditcardtype
+creditcardexpirydate
card_fk de la table T_CUSTOMER correspond à la clé primaire de T_CREDIT_CARD
T_ORDER_LINE
+id
+quantity
+order_fk
+item_fk
T_ITEM
+id
+name
+unitCost
+imagePath
+product_fk
T_PRODUCT
+id
+name
+description
+category_fk
T_CATEGORY
+id
+name
+description

3. Mapping Objet/Tables

- **Déterminer les liaisons entre les objets pour les opérations CRUD**
- **Actions de JPA (CASCADES)**

	Objet principal		Objets Liés
1	**Customer**	**►1**	**Address**
		►1	**CreditCard**

Lorsqu'un customer est créé, mis-à-jour ou supprimé, jpa peut effectuer conjointement ces opérations sur l'adresse et la carte de crédit.



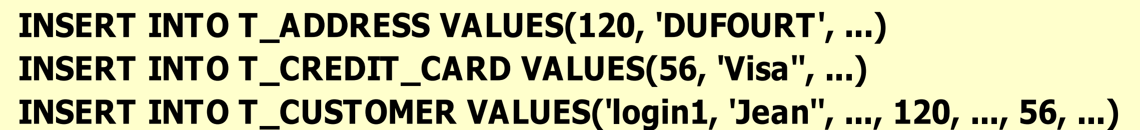
```
Customer customer= new Customer()
customer.setId("login1")
customer.setFirstname("Jean")
customer.setLastname("Dupuis")

Address address =new Address()
address.setId(120)
address.setStreet1("Dufourt")

CreditCard creditCart =new CreditCard()
creditCard.setId(56)
creditCard.setType("Visa")

customer.setAddress(address)
customer.setCreditCard(creditCard)
```

em.persist(customer)



CREATE	Customer	► CREATE	Address
		► CREATE	CreditCard

UPDATE	Customer	► UPDATE	Address
		► UPDATE	CreditCard

DELETE	Customer	► DELETE	Address
		► DELETE	CreditCard

	Objet principal		Objets Liés
1	**Order**	**►n**	**OrderLine**

Il en est de même pour créer, modifier ou supprimer un Order. Jpa peut créer, modifier ou supprimer conjointement l'ensemble des OrderLines.

CREATE	**Order**	**► CREATE**	**OrderLine 1**
		► CREATE	**OrderLine 2**

DELETE	**Order**	**► DELETE**	**OrderLine 1**
		► DELETE	**OrderLine 2**

UPDATE	**Order**	**► UPDATE**	**OrderLine 1**
		► DELETE	**OrderLine 2**
		► CREATE	**OrderLine 3**

En définissant cascade pour l'opération update, il sera nécessaire (dans le code) de passer par Order
pour modifier les OrderLines. Cela demande de redéfinir tous les orderLines pour en modifier ou supprimer un seul.

	Objet principal		Objets Liés		Objets Liés
1	**Category**	**►n**	**Produit**	**►n**	**Item**

CREATE	**Category**	**► CREATE**	**Produit 1**	**► CREATE**	**N Item**
		► CREATE	**Produit 2**	**► CREATE**	**N Item**
		...	**...**	**...**	**...**

DELETE	**Category**	**► DELETE**	**Produit 1**	**► DELETE**	**N Item**
		► DELETE	**Produit 2**	**► DELETE**	**N Item**
		...	**...**	**...**	**...**

Pour l'opération update les mises-à-jour ne se feront pas en cascade pour les produits et les items.
On laisse la possibilité d'accéder aux items ou aux produits sans passer nécessairement par la catégorie.

II. Implémentation

1. Couche métier

- **Créer les classes entity du domain**
- **En fonction des relations identifiées entre objets**

- **"Entity" représentant l'objet Customer**
- **les attributs sont mappés sur la table T_CUSTOMER**

```java
@Entity
@Table(name = "T_CUSTOMER")
public final class Customer implements Serializable {

   // === Attributs
   @Id
   private String id;
   @Column(nullable = false)

   private String firstname;
   @Column(nullable = false)
   private String lastname;
   @Column(nullable = false)
   private String password;
   private String telephone;
   private String email;

   @OneToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
   @JoinColumn(name ="address_fk", nullable = true)
   private Address address;

   @OneToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
   @JoinColumn(name ="card_fk", nullable = true)
   private CreditCard creditCard ;

   // =          Constructors          =
   public Customer() {}
```

T_CUSTOMER
id
firsname
lastname
telephone
password
address_fk
card_fk

- **Customer est la classe parent**

  Elle spécifie les relations avec les entity Adress et CreditCard :

@OneToOne
@JoinColumn

Customer est lié à **Address** par une relation (1, 1) pour les opération CRUD.
Lors de la création d'un Customer, JPA accèdera à la table **T_ADDRESS** pour créer l'adresse,
puis à la table **T_CUSTOMER**.

La colonne **addresse_fk** de T_CUSTOMER a la même valeur que la colonne **id** de T_ADDRESS.

- "Entity" représentant l'objet Address
- les attributs sont mappés sur la table T_ADDRESS

```java
@Entity
@Table(name = "T_ADDRESS")
public final class Address implements Serializable {

    // ======================================
    // =            Attributes             =
    // ======================================
        @Id
        private String id;
        private String street1;
        private String street2;
        private String city;
        private String state;
        private String zipcode;
        private String country;


    // ======================================
        // =            Constructors            =
        // ======================================
        public Address() {
        }
...
```

- "Entity" représentant l'objet CreditCard d'une customer

- les attributs sont mappés sur la table T_CREDIT_CARD

```
@Entity
@Table(name = "T_CREDIT_CARD")
public final class CreditCard implements Serializable {

   // ======================================
   // =            Attributes          =
   // ======================================
      @Id
      private String id;
      private String creditCardNumber;
      private String creditCardType;
      private String creditCardExpiryDate;

      // ======================================
      // =           Constructors          =
      // ======================================
      public CreditCard() {
      }
...
```

2. Le fichier persistence.xml

- **Déclarer les Entity gérer par JPA** dans le fichier perstistence.xml
- Pour une **exécution en dehors du conteneur** (RESOURCE_LOCAL) on utilise l'implémentation de **toplink**.

```
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence">
<persistence-unit name="petstorePU" transaction-type="RESOURCE_LOCAL">
   <provider>
        oracle.toplink.essentials.PersistenceProvider
   </provider>
   <class>com.yaps.petstore.server.domain.customer.Customer</class>
   <class>com.yaps.petstore.server.domain.customer.Address</class>
   <class>com.yaps.petstore.server.domain.customer.CreditCard</class>
   <properties>
     <property name="toplink.logging.level" value="FINE"/>
     <property name="toplink.jdbc.url"    value="jdbc:mysql://localhost:3306/petstoreDB"/>
     <property name="toplink.jdbc.user" value="root"/>
     <property name="toplink.jdbc.driver" value="com.mysql.jdbc.Driver"/>
     <property name="toplink.jdbc.password" value=""/>
     <property name="toplink.target-database" value="MySQL4"/>
     <property name="toplink.ddl-generation" value="none"/>
   </properties>
  </persistence-unit>
</persistence>
```

- **Tester à partir d'une classe JUnit**

```
public class CustomerJpaTest {

    private static String PERSISTENCE_UNIT_NAME = "petstorePU";

    private EntityManagerFactory emf;
    private EntityManager em;
    private EntityTransaction trans;

   @Before
  public void init() {
  emf = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);
        em = emf.createEntityManager();
        trans = em.getTransaction();
    }

    @Test
    public void testJPACreateCustomer() throws Exception {

      // Création du customer
         Customer customer =new Customer();
         customer.setId("customer1"); customer.setFirstname("Jean");    customer.setLastname("Durand");
         customer.setPassword("password");        customer.setTelephone("0154675464");
         customer.setEmail("mail@yahoo.fr");

         Address address =new Address();
```

```
        address.setId(customer.getId()); address.setStreet1("street1");     address.setState("France");


        customer.setAddress(address);

        trans.begin();
                em.persist(customer);
        trans.commit();
    }
}
```

- **La console d'Eclipse affiche**

```
INSERT INTO T_ADDRESS (ID, CITY, STATE, STREET2, ZIPCODE, STREET1, COUNTRY) VALUES (?, ?, ?, ?, ?, ?, ?)
INSERT INTO T_CREDIT_CARD (ID, CREDITCARDNUMBER, CREDITCARDTYPE, CREDITCARDEXPIRYDATE)
VALUES (?, ?, ?, ?)
INSERT INTO T_CUSTOMER (ID, PASSWORD, TELEPHONE, FIRSTNAME, EMAIL, LASTNAME, card_fk, address_fk)
VALUES (?, ?, ?, ?, ?, ?, ?, ?)
```

3. Les classes DAO

- **Créer les classes DAO du domain**

Classes ayant la responsabilité de déclencher les méthodes qui accèdent à la base de données.

Les transactions sont gérées au niveau de la couche service.

L'entity manager est instancié dans la couche service. C'est un attribut (valué par le service appelant) dans la classe DAO.

```
public class CustomerDAO extends CommonDomain  {

   // =====           Attributes           =====

   private EntityManager em;

   // =====          Business methods        =====

   public CustomerDTO createCustomer(final CustomerDTO customerDTO) {
      final String mname = "createCustomer";

      if (customerDTO == null)
         throw new CheckException("Customer object is null");

      // Transforms DTO into domain object
      Customer customer =transformCustomerDTO2Customer(customerDTO);
      customer.checkData();

      // test if the user exists
      Customer customer2 =em.find(Customer.class, customer.getId());
       if (customer2 !=null) throw new DuplicateKeyException("Customer     already exists.");
      // Creates the object
      em.persist(customer);

      // Transforms domain object into DTO
      final CustomerDTO result = transformCustomer2DTO(customer);
      return result;
   }
```

4. Couche service

- **créer les EJB3 session**

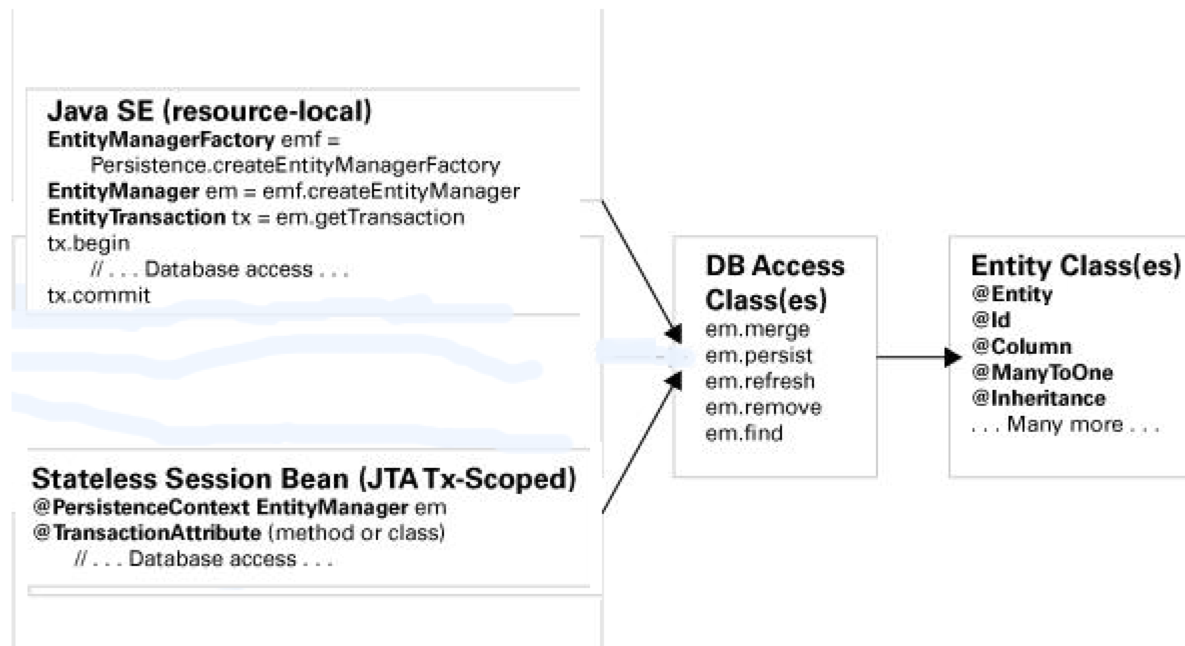
Le **contexte de persitence** délimite le périmètre auquel l'**EntityManager** a accès : les classes déclarées dans le fichier persistence.xml pour la base de données mentionnée. Il est instancié par le serveur d'application.

Les **transactions** sont gérées par le conteneur et leurs caractéristiques sont annotées sur les méthodes.

```
@TransactionManagement(value=TransactionManagementType.CONTAINER)
@Stateless
public class CustomerServiceBean extends CommonRemoteService implements CustomerServiceRemote,
CustomerServiceLocal {

   // ====          Attributes          ====
   @PersistenceContext(unitName = "petstorePU")
   private EntityManager em;

   // ====         Business methods        ====
   @TransactionAttribute(value = TransactionAttributeType.REQUIRED)
   public CustomerDTO createCustomer(final CustomerDTO customerDTO) {
      // Appel du DAO pour enregistrement en base
      CustomerDAO  customerDAO =new CustomerDAO();
      customerDAO.setEm(em);
      CustomerDTO result = customerDAO.createCustomer(customerDTO);
     // Appel de barbank pour vérification de la carte
     ...
   return  result    } ...
```


Java SE (resource-local)
EntityManagerFactory emf =
Persistence.createEntityManagerFactory
EntityManager em = emf.createEntityManager
EntityTransaction tx = em.getTransaction
tx.begin
// . . . Database access . . .
tx.commit
Stateless Session Bean (JTA Tx-Scoped)
@PersistenceContext EntityManager em
@TransactionAttribute (method or class)
// . . . Database access . . .
DB Access Class(es)
em.merge
em.persist
em.refresh
em.remove
em.find
Entity Class(es)
@Entity
@Id
@Column
@ManyToOne
@Inheritance
. . . Many more . . .

- **Tester à partir d'une classe JUnit**

Le serveur JBoss doit être démarré. Le fichier de persistence utilise **hybernate** implémenté nativement par Jboss. Les transactions sont gérées dans le conteneur à partir de **JTA** et de sa **data source**.

```
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" ...>
    <persistence-unit name="petstorePU" transaction-type="JTA">
   <provider>org.hibernate.ejb.HibernatePersistence</provider>
   <jta-data-source>java:/PetstoreDS</jta-data-source>
   <class>com.yaps.petstore.server.domain.customer.Customer</class>
   <class>com.yaps.petstore.server.domain.customer.Address</class>
   <class>com.yaps.petstore.server.domain.customer.CreditCard</class>
        <properties>
                  <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/petstoreDB" />
                <property name="hibernate.connection.driver_class"        value="com.mysql.jdbc.Driver" />
                 <property name="hibernate.connection.username" value="root" />
                 <property name="hibernate.connection.password" value="" />
                <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLInnoDBDialect" />
                 <property name="hibernate.hbm2ddl.auto" value="update"/>
                <property name="hibernate.cache.provider_class" value="org.hibernate.cache.HashtableCacheProvider"/>
                 <property name="hibernate.show_sql" value="true" />
        </properties>
  </persistence-unit>
</persistence>
```

```
public class CustomerServiceTest extends AbstractTestCase {

   // Référence sur l'EJB session
   CustomerServiceRemote customerBean;
   Context ctx;

   @BeforeClass
   public void setUp() throws Exception {
       ctx = new InitialContext();
       customerBean = (CustomerServiceRemote)  ctx.lookup("petstore/CustomerServiceBean/remote");
   }

   @Test
   public void testServiceCreateCustomer() throws Exception {
        Customer custo =new Customer();
        CustomerDTO customer =new CustomerDTO();
        customer.setId("customer7"); customer.setFirstname("Jean");    customer.setLastname("Durand");
        customer.setPassword("password");        customer.setTelephone("0154675464");
        customer.setEmail("mail@yahoo.fr");
        customer.setStreet1("street1"); customer.setState("France");

        customer.setCreditCardExpiryDate("05/10");
        customer.setCreditCardType("Visa"); customer.setCreditCardNumber("12345");

        CustomerDTO customerDTO =customerBean.createCustomer(customer);

        System.out.println(" after create customerDTO : "+customerDTO);
   }
```

5. Le client JSF

Le bean managé utilise l'injection de dépendance pour accéder à l'EJB.

- **Bean associé à la facelet createcustomer.xhtml**

```
public class CustomerControllor extends Controllor {

    // = ====        Constructeurs        =====

    public CustomerControllor() {
        setListState(); // Initialisation des listes
        setListCountry();
        setListCreditCardType();
    }

    // =        Attributs          =  Champs de la jsp

    private CustomerDTO customerDTO;

    private SelectItem[] listState ;
    private SelectItem[] listCountry ;
    private SelectItem[] listCreditCardType ;

    @EJB (name ="petstore/CustomerServiceBean/remote")
    CustomerServiceLocal customerBean;
```

```
// =        Methodes                    =

    /** Action du <commandButton> de la facelet : Appel du service createCustomer de l'EJB*/
     public String create(){
        String response =null;

        try {
            // Creates the customer
            customerDTO =customerBean.createCustomer(customerDTO);

            // NAVIGATION RULE - RENDER RESPONSE
            response ="index";

         }
        catch (Exception e) {
            // RENDER RESPONSE
                  ...
         }

         return response;
     }
....
```

Configuration du projet

1 Déclarer les librairies sous eclipse

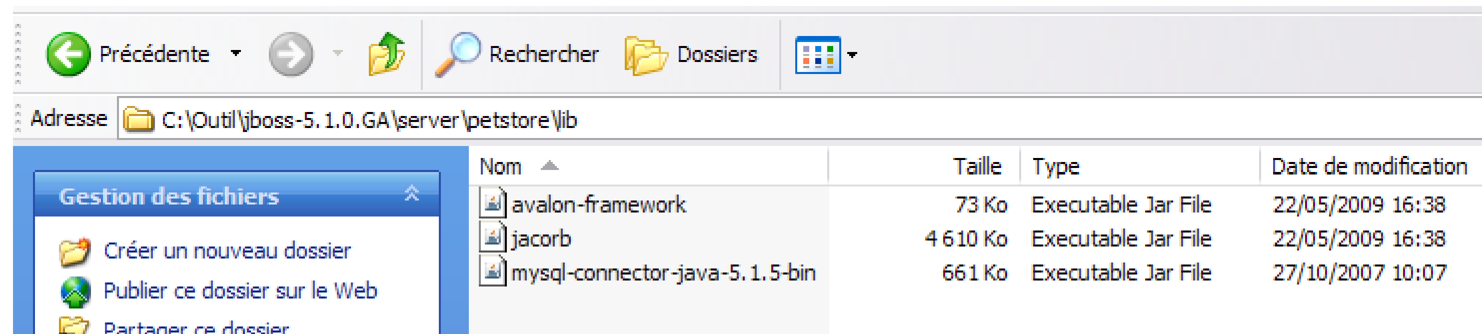
Ouvrir le projet sous eclipse :
Projet, Java Build Path, Add External JARs

Ajouter la liste de librairies.
Les jars se trouvent sous le répertoire lib du projet.

Pour faire tourner les tests Junit sous eclipse, ajouter dans les librairies du projet les fichiers jars du répertoire C:\Outil\jboss-5.1.0.GA\client.

2. Déclarer les librairies sous Jboss

Ajouter la **librairie de mysql** dans le répertoire lib de petstore :



3. Copier dans le répertoire deploy de petstore le fichier mysql-ds.xml qui définit la data source :

Précédente Rechercher Dossiers

Adresse C:\Outil\jboss-5.1.0.GA\server\petstore\deploy

Gestion des fichiers

Autres emplacements

Détails

Nom	Taille	Type
jbossweb.sar		Dossier de fichiers
jbossws.sar		Dossier de fichiers
jmx-console.war		Dossier de fichiers
juddi-service.sar		Dossier de fichiers
messaging		Dossier de fichiers
profileservice-secured.jar		Dossier de fichiers
ROOT.war		Dossier de fichiers
security		Dossier de fichiers
uuid-key-generator.sar		Dossier de fichiers
xnio-provider.jar		Dossier de fichiers
adminps	91 Ko	Fichier WAR
barkbank	18 Ko	Fichier WAR
ejb2-container-jboss-beans	1 Ko	Document XML
ejb2-timer-service	3 Ko	Document XML
ejb3-connectors-jboss-beans	2 Ko	Document XML
ejb3-container-jboss-beans	1 Ko	Document XML
ejb3-interceptors-aop	27 Ko	Document XML
ejb3-timerservice-jboss-beans	1 Ko	Document XML
hdscanner-jboss-beans	2 Ko	Document XML
hsqldb-ds	6 Ko	Document XML
iiop-service	7 Ko	Document XML
jboss-local-jdbc	15 Ko	Archive WinRAR
jboss-xa-jdbc	15 Ko	Archive WinRAR
jca-jboss-beans	3 Ko	Document XML
jms-ra	85 Ko	Archive WinRAR
jmx-invoker-service	6 Ko	Document XML
jsr88-service	2 Ko	Document XML
legacy-invokers-service	3 Ko	Document XML
mail-service	2 Ko	Document XML
mysql-ds	2 Ko	Document XML

TP13

A partir des sources fournies du TP13_JBOSS 5.1.0 :

- **implémenter les classes métier du domaine Customer**
- **Faire une classe JUnit pour tester les classes métier en local**
- **implémenter le DAO (CustomerDAO)**

- **implémenter l'ejb CustomerServiceBean :**
  - politique transactionnelle du container, des méthodes
  - injection du contexte de persistence
  - utilisation des dao pour les accès base
- **Faire une classe JUnit pour tester le service dans le conteneur**

- **modifier le bean managé CustomerControllor pour utiliser l'injection de dépendance (`@EJB`) pour accéder à l'ejb.**