

Exercise

To dive into the whole EJB topic, we want you to develop a small and simple application for a fictitious music store working with Enterprise Java Bean technology. The customers should be able to search the music cd database and add and remove cds to/from a shopping cart. The employees should maintain the database. To keep it simple, you should make it possible to add Artists and MusicCds to the database, no remove, no edit!

An artist should have the attributes:

- id
- name
- nationality

A music cd should have:

- id
- title
- year
- price
- songs
- artist

The business logic provided by the application server for the client front end should have the following features:

```
* get a list of all artists
* add a new artist
* add a new music cd
* find music cds by artist's search pattern
* get a list of music cds by its artist
* add a music cd to the shopping cart
* remove music cd from the shopping cart
* get the content of the shopping cart
```

Implement the server using EJB and the client with JSP (Java Server Pages).

Create the Entities

The first step is to write the according Entities which will be mapped to the database. The Entities must have the `@Entity` annotation and the table to which it should be mapped (`@Table(name="tablename")`). All the attributes of the music cd must be private and require the appropriate getter and setter methods. Remember the naming conventions of getters and setters:

```
private int myattribute;
public int getMyattribute()
public void setMyattribute(int myattribute)
```

Furthermore the Entity needs a primary key which will be inserted by the container automatically.

The getter-methods requires additional annotations. To let the container know from which column it should look for the requested value, every getter-methods needs the `@Column(name="column_name")` annotation. The getter for the primary key also needs a

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
```

annotation.

Create entities for Artist, MusciCD and Song.

Insert namedQueries

NamedQueries are nothing else than a simple SQL Statement in EJBSql. They have to be unique in the whole application. To keep track of them we are going to define them on top of an Entity.

```
@javax.persistence.NamedQueries({
    @javax.persistence.NamedQuery(name="findAllMusicCD",
        query="SELECT OBJECT(o) FROM MusicCD o")
    })
```

Create namedQueries for:

- find all music cds in the database
- find all artists

Create the StoreBrowser SessionBean

The StoreBrowser SessionBean (SB) contains the business logic to browse the music cd database. It is a stateless SB. You need to create an interface and a SB implementing the interface. For simplification, the interface should be named as *StoreBrowser* and the SB *StoreBrowserBean*. The SB needs an EntityManager injected with the *@PersistenceContext* annotation. The following methods should be implemented:

```
public MusicCd getMusicCd(int id);
public Collection<MusicCd> getMusicCdsByArtist(Artist artist);
public Collection<MusicCd> getMusicCdsByArtistPattern(String pattern);
public Collection<Artist> getAllArtists();
```

Create the ShopAdmin SessionBean

The ShopAdmin SessionBean contains methods to store and read artists as well as music cds. The following methods from the interface *ShopAdminSession* should be implemented:

```
public void addArtist(Artist artist);
public void removeArtist(Integer id);
public Artist getArtist(int id);
public Collection<Artist> getAllArtists();
public void saveMusicCd(MusicCd m);
```

Short EntityManager HowTo

EntityManager *persist()*

The EntityManager *persist(Object entity)* API is used to mark a new instance for insertion into the Database. It must only be called on **new** Entities. The value returned from the *persist(Object entity)* call is the same instance that was passed in.

```
@Stateless
public class TravelAgentBean implements TravelAgentSession {
    ...
    public void createTrip(String tName) {
        Trip trip = new Trip();
        trip.setFirstName(fName);
        em.persist(employee);
    }
}
```

```
}
...

```

EntityManager *merge()*

To integrate a bean, that was read in a different transaction or provided by the client into the current transaction use the *merge* (*Object entity*) API. The result will be an instance of the provided entity. The state of that entity will also be available in the returned instance.

```
@Stateless
public class TravelAgentBean implements TravelAgentSession {
    ...
    public void updateAddress(Address addressExample) {
        em.merge(addressExample);
    }
    ...
}
```

EntityManager *remove()*

To delete a bean from the database use the *remove*(Object entityBean) API.

```
@Stateless
public class TravelAgentBean implements TravelAgentSession {
    ...
    public void removeTrip(Integer tripId) {
        Trip trip = (Trip)em.find(Trip.class, tripId);
        ...
        em.remove(trip);
    }
    ...
}
```

EntityManager *find()*

When a primary key query is required the *find*(String entityName, Object primaryKey) or *find*(Class entityClass, Object primaryKey) API can be used.

```
@Stateless
public class TravelAgentBean implements TravelAgentSession {
    ...
    public void removeTrip(Integer tripId) {
        Trip trip = (Trip)em.find(Trip.class, tripId);
        ...
        em.remove(trip);
    }
    ...
}
```

Call NamedQueries

The NamedQueries must be called using the EntityManager.

```
@PersistenceContext
public EntityManager em;
...
customers = em.createNamedQuery("findAllCustomersWithName")
    .setParameter("custName", "Smith")
    .getResultList(); // <- returns a List
```

```
@PersistenceContext
public EntityManager em;
...
customer = (Customer) em.createNamedQuery("findAllCustomersWithName")
    .setParameter("custName", "Smith")
    .getSingleResult(); // <- returns a single Object!! Must be casted!
```

Create the ShoppingSession Stateful SessionBean

The ShoppingSession stateful SessionBean should act as a shopping cart. It must have a collection of the ordered music cds. There must be a method which initialises the collection of the cds. Because of the instance pooling there must not be any constructor for any bean type. But for initialising we can use a life cycle method called *PostConstruct*. Just put the annotation *@PostConstruct* above your init method and the container will handle this. The interface should contain the following methods:

```
public void addMusicCd(MusicCd musicCd);
public void removeMusicCd(MusicCd musicCd);
public Collection<MusicCd> getCart();
public void submitOrder();
```

The JSPs

Have a look at the package

```
lu.linuxdays.musicstore.web
```

There are some jsp as well as html files. The **browser.jsp** is the main website that contains the search field and the shopping cart. **addArtist.jsp** and **addMusicCd.jsp** contain the logic and html fields to add either a new artist and a new music cd. The detailed exercises are explained in the code.

Packaging and Deployment

- Build a jar file as described in chapter 2.7 of the documentation.
- Additionally we need another file that contains the web application, a *.war* file. As the Jboss AS contains a Tomcat Webcontainer we need this file to be deployed by Tomcat. It must contain the jsps and html file that are in the package *lu.linuxdays.musicstore.web*. Do not enter a Prefix for that!
- Deploy both files to the deploy directory of JBoss (*JBOSS_HOME/server/default/deploy*)

The Client

Point your browser to

```
http://127.0.0.1:8080/NAME_OF_THE_WAR_FILE/browser.jsp
```

and start using your first webshop written with Enterprise Java Beans.

Thats all.