Lab 9A: Transactions

In this lab, you will write a program that uses EJB declarative transactions.

Objectives:

- To use EJB declarative transaction demarcation

Part 1: Setup

In this part, you will import an Eclipse project that contains entities, EJBs and a Web application to save you typing.

Steps:

_1. Your first job is to import the completed Eclipse project. Follow these steps:

   a. Switch to the *JEE Perspective* by selecting from the Eclipse menu *Window - Open Perspective - Other - Java EE (default)*.

   b. From the Eclipse menu, choose *File - Import* to start the wizard.

   c. On the first wizard page, select *General - Existing Projects into Workspace*, then press Next.

   d. On the next page, for the *Select archive file*, press the Browse button and navigate to **{Lab Installation Directory}/starters/lab09a** and select the *lab09a.zip* file. If your computer has the standard lab setup, the path to the ZIP file is:

      ```
      c:\j2eeclass\starters\lab09a\lab09a.zip
      ```

      Press Next.

   e. On the next wizard page, put a checkmark next to the *lab09a-Web*, *lab09a-JPA*, *lab09a-EJB* and *lab09a-EAR* entries, then press Finish.

_2. Examine the *lab09a-JPA* and *lab09a-EJB* projects looking in the *Java Resources:src/com.oaktreeair.ffprogram* package. Figure 1 summarizes the application:

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

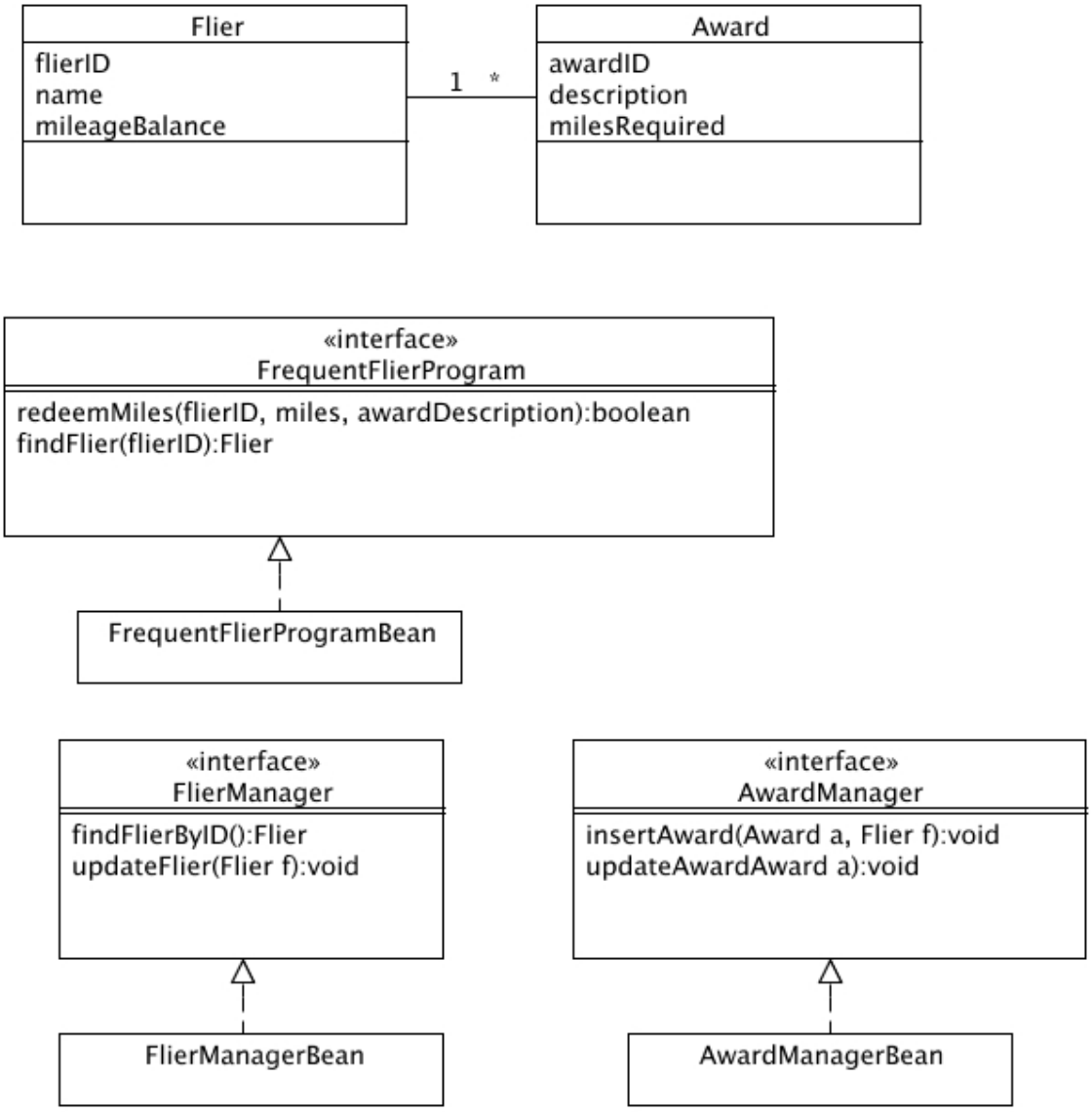


Figure 1: Application Model

We have provided you will all of the interfaces; you will write the implementation classes. In addition, we have provided you with the Flier.java and Award.java classes, but you will need to configure them as entities.

_3. Before you can run the Web application, you need to create the database tables and populate them with sample data. We have provided you with an SQL script for this purpose; follow these steps to run the script:

a. Open a Web browser and enter this URL:

```
http://localhost:8080/jmx-console/
```

This is the JBoss JMX console, which lets you adminster the server.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

b. Find the *jboss* category in the listing, then press the *database=localDB,service=Hypersonic* link.

c. Scroll down to the bottom to find the *startDatabaseManager* operation, then click that operation's *Invoke* button. This will start the administration GUI for the HypersonicSQL database. The GUI will start in a separate window on your desktop - you can open it by clicking its entry in the Windows task bar at the bottom of your screen.

d. Back in Eclipse, in the *lab09a-JPA/src* folder, open the *createfliers.sql* script and examine it and ensure that you understand it.

   Highlight all of the text in the script, then copy it to the clipboard.

   Back in the HypersonicSQL GUI, paste the clipboard contents into the top entry field, then press the *Execute SQL* toolbar button. The script should execute with no errors.

   WARNING: The DefaultDS provided by JBoss uses a memory-based database, so if you stop the server, the tables and their data will be lost. So if you stop/restart JBoss, you will need to run this script again.

Part 2: Configuring the Entities

In this part, you will use JPA annotations to configure the Flier and Award entities.

Steps:

_1. Start by configuring the Flier entity:

   a. Open Flier.java into the editor and familiarize yourself with it. Examining Figure 1 above, note that it has a bidirectional, one-to-many relationship with Award.

   b. Start by adding the required **@Entity** and **@Id** annotations. You should also use an auto-generated key using the IDENTITY strategy.

   c. Annotate the *awards* field as a OneToMany relationship. This entity, Flier, will act as the inverse, so be sure to use the *mappedBy* attribute. For now, you can ignore any error on the OneToMany annotation complaining about "invalid mapping type".

_2. Next, configure the Award entity:

   a. Open Award.java into the editor and familiarize yourself with it. Examining Figure 1 above, note that it has a bidirectional, many-to-one relationship with Flier.

   b. Start by adding the required **@Entity** and **@Id** annotations. You should also use an auto-generated key using the IDENTITY strategy.

   c. Next, annotate the *flier* field as ManyToOne and specify the join-column that the database table uses:

```
@ManyToOne
@JoinColumn(name="flierID")
```

   Save both entities. You should have no errors.

Part 3: Writing the EJBs

In this part, you will write EJBs that will be accessed from the Web tier. We have provided you with the EJB interfaces: your job is to write the implementations.

Steps:

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

_1. In the *lab09a-EJB* project, create a new class named **FlierManagerBean** in the *com.oaktreeair.ffprogram* package that implements the *FlierManager* interface.

Complete the EJB:

a. Annotate the class so that it's a stateless session EJB.

b. Define a field of type EntityManager and inject a reference into it. Be sure to specify the unit name.

c. Annotate the *findFlierById* method so that it's transactional, using a propagation attribute of SUPPORTS. That way, if your program calls this method without a transaction, it will run with no transaction - that's appropriate for a method that doesn't modify any entities.

d. Annotate the *updateFlier* method so that it's transactional, using a propagation attribute of REQUIRED. That way, if your program calls this method without a transaction, the container will start a new transaction - that's appropriate for a method that modifies entities.

e. Implement the *findFlierByID* method. Issue a JPA query that retrieves the Flier with the given flierID and returns it.

f. Implement the *updateFlier* method. Call the EntityManager's *merge* method, passing the given Flier reference. The *merge* method ensures that the entity in memory and the database table are in sync.

_2. Create a new class named **AwardManagerBean** in the *com.oaktreeair.ffprogram* package that implements the *AwardManager* interface.

Complete the EJB:

a. Annotate the class so that it's a stateless session EJB.

b. Define a field of type EntityManager and inject a reference into it. Be sure to specify the unit name.

c. Annotate the *insertAward* method so that it's transactional with a propagation attribute of MANDATORY. This method is designed to only be called from another EJB, FrequentFlierProgramBean, that starts a transaction (you will write that EJB in a moment). So if called from that EJB, it will propagate the transaction, but if called without any transaction in place, the container will throw an exception. This guarantees that this method is only called via the FrequentFlierProgramBean EJB.

d. Annotate the *updateAward* method so that it's transactional, using a propagation attribute of REQUIRED. That way, if your program calls this method without a transaction, the container will start a new transaction - that's appropriate for a method that modifies entities.

e. Implement the *insertAward* method (this code assumes you named your EntityManager field *em*:

```
award.setFlier(flier);
em.persist(award);
Collection<Award> awards = flier.getAwards();
awards.add(award);
em.merge(flier);
```

f. Implement the *updateAward* method:

```
em.merge(award);
```

_3. Next, write the FrequentFlierProgramBean EJB class:

a. Create a new class named **FrequentFlierProgramBean** in the *com.oaktreeair.ffprogram* package that implements the *FrequentFlierProgram* interface.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

b. Annotate the class so that it's a stateless session EJB.

c. Annotate the class so all of its methods are transactional with *REQUIRED* propagation.

d. Define fields for the AwardManager and the FlierManager EJBs and use @EJB to inject a reference to those EJBs.

e. Implement the *findFlierByID* method:

```
Long id = new Long(flierID);
return flierDAO.findFlierByID(id);
```

f. Implement the *redeemMiles* method (assuming that you named your AwardManager reference *am* and your FlierManager reference *fm*):

```
boolean retVal = false;

Flier f = fm.findFlierByID((long)flierID);
int balance = f.getMileageBalance();

if (balance - miles >= 0)
{
    // create a new award and add
    // to flier's award collection
    Award award = new Award();
    award.setMilesRequired(miles);
    award.setDescription(awardDescription);

    am.insertAward(f, award);

    balance -= miles;
    f.setMileageBalance(balance);
}

return retVal;
```

Part 4: Testing

In this part, you will run a provided servlet that redeems miles for an Award on a Flier's account.

Steps:

_1. In the *lab09a-Web* project, open the *Java Resources: src/servlets/AwardServlet.java* servlet and ensure that you understand it. Note how it uses the Spring context to retrieve a reference to the FrequentFlierProgram bean, then attempts to redeem a mileage award, displaying the results to a simple Web page.

_2. Test the program by right-clicking on AwardServlet.java and choosing *Run As - Run on Server*. After the server starts, you should see the "before" and "after" of the attempt to redeem miles.

_3. Insert the following code into the servlet immediately following the first *out.println* that displays the mileage balance:

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
Collection<Award> awards = f.getAwards();
out.println("<table border='1'>");
for(Award a : awards)
{
    out.println("<tr>");
    out.println("<td>" + a.getDescription() + "</td>");
    out.println("<td>" + a.getMilesRequired() + "</td>");
    out.println("</tr>");
}
out.println("</table>");
```

Also inert this code after the second *out.println* that displays the mileage balance, modifying it as necessary to get it to compile.

Try running the servlet again. What happens?

_4. The problem is that the EJB/JPA code returns a Flier object to the servlet that has a lazily instantiated relationship. Since the transaction and session start and end in the FrequentFlierProgram bean, the servlet itself is not within the transaction - the Hibernate session is closed. So when the code you just pasted attempts to follow the relationship, it gets the lazy instantiation error.

To fix, modify the Flier entity so that the @OneToMany annotation specifies a fetch type of EAGER.

Then run the servlet again - it should work. **Note:**You may need to right-click on the server and choose **Publish** so that JBoss picks up the changes.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.