Extra Lab 3: Hibernate Entity Relationships

In this lab, you will write a program that uses Hibernate to persist a business object model that features relationships.

Objectives:

- To work with persistent relationships

Part 1: Creating Entities with Relationships

In this part, you will create the entities to represent a business model. The model in question is for an organization that maintains memberships. Each of the several membership lists keeps track of associates, each of which pays dues. A given associate can be part of more than one list. The UML class diagram for your business model is shown in Figure 1:

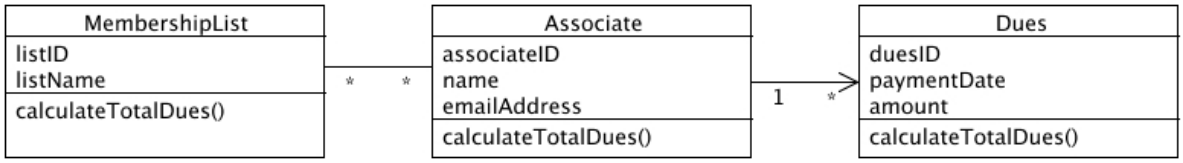


Figure 1: Business Model

Steps:

_1. In the JBoss Developer Studio (this lab assumes the Studio instead of Eclipse), create a new Java project named **ExtraLab3** according to these specifications:

```
Name:           ExtraLab3
Libraries:      Hibernate User Library
```

_2. Copy the following files from *ExtraLab1/src* to the *src* folder in the new project:

```
hibernate.cfg.xml
log4j.properties
```

_3. Edit the copied hibernate.cfg.xml and make the following changes:

a. Change the *mapping resource* element to refer to:

```
org/biketrails/members/MembershipList.hbm.xml
```

Then write additional *mapping* elements for:

Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
org/biketrails/members/Associate.hbm.xml
org/biketrails/members/Dues.hbm.xml
```

You will create these files in a later step.

b. Unlike the other labs you've done so far, this time we will let Hibernate create the tables for the entities you will write.

Ensure that the following property is present and not commented-out:

```
<property name="hibernate.hbm2ddl.auto">create</property>
```

Save and close the configuration file.

_4. As in earlier labs, use the New Class wizard to create the three entity classes in the **org.biketrails.members** package, *MembershipList, Associate* and *Dues*. After you create them, open them into the editor and define each entity's properties, complete with get/set methods as described in Figure 1 and summarized here:

```
MembershipList
--------------

listID        java.lang.Long
listName      java.lang.String

Associate
---------

associateID   java.lang.Long
name          java.lang.String
emailAddress  java.lang.String

Dues
----

duesID        java.lang.Long
paymentDate   java.util.Date
amount        java.math.BigDecimal
```

_5. Edit MembershipList.java and make the following changes and additions:

a. As you can see in Figure 1, MembershipList has a ManyToMany bidirectional relationship with Associate. To implement this, MembershipList needs to maintain a collection of Associates. Define the following field:

```
private Collection<Associate> associateList =
    new HashSet<Associate>();
```

Use *Source - Organize Imports* to import java.util.HashSet.

Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

b. Use *Source - Generate Getters and Setters* to create get/set methods for the associateList.

c. Write a business logic method that calculates the total dues paid by all associates:

```
public BigDecimal calculateTotalDues()
{
    BigDecimal total = new BigDecimal(0);

    for (Associate a : associateList)
    {
        total = total.add(a.calculateTotalDues());
    }

    return total;
}
```

Note: You will have an error on the calculateTotalDues() call - you will fix that in a moment, so ignore it for now.

_6. Edit Associate.java and make the following changes and additions:

a. Since each Associate can be a member of multiple lists, Associate needs to maintain a collection of MembershipLists. Define the following field:

```
private Collection<MembershipList> membershipList =
    new HashSet<MembershipList>();
```

Use *Source - Organize Imports* to import java.util.HashSet.

b. Use *Source - Generate Getters and Setters* to create get/set methods for the membershipList.

c. Associate also has a unidirectional OneToMany relationship with Dues. Define a field named **duesList** to implement that relationship:

```
private Collection<Dues> duesList =
    new HashSet<Dues>();
```

Generate get/set methods for the duesList field.

d. Write a business logic method that calculates the total dues payments by this associate:

Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
public BigDecimal calculateTotalDues()
{
    BigDecimal total = new BigDecimal(0);

    for (Dues d : duesList)
    {
        total = total.add(d.getAmount());
    }

    return total;
}
```

This should fix the error in MembershipList.

_7. Next, you must configure the entities for Hibernate. Start by configuring the MembershipList entity. Follow these steps:

a. Right-click on the *org.biketrails.members* package and choose *New - Other - Hibernate - Hibernate XML Mapping File (hbm.xml)*, then press Next, then Finish - the Developer Studio generates starter mapping files for all of the entities in the package.

b. Open MembershipList.hbm.xml into the editor.

c. In the *class* mapping, change the *table* to **MEMBERS**.

d. In the *id* mapping, change the *generator* to **identity**.

e. After the *listName* property element, configure the many-to-many relationship from MembershipList to Associate, modifying the generated configuration as necessary:

```
<set name="associateList" table="MEMBER_ASSOCIATE">
  <key column="listID"/>
  <many-to-many column="associateID"
    class="org.biketrails.members.Associate"/>
</set>
```

f. Save and close the mapping file.

_8. Next, configure the Associate entity:

a. Open Associate.hbm.xml into the editor.

b. Change the *generator* to **identity**.

c. Configure a *set* element so it represents the bidirectional, one-to-many relationship between MembershipList and Associate. Associate will be the *inverse*, or non-owning side (modify the generated configuration):

Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
<set name="membershipList"
    table="MEMBER_ASSOCIATE"
    inverse="true">
  <key column="associateID"/>
  <many-to-many column="listID"
    class="org.biketrails.members.MembershipList"/>
</set>
```

d. Configure the unidirectional, one-to-many relationship between Associate and Dues, modifying the generated configuration:

```
<set name="duesList"
    table="ASSOCIATE_DUES">
  <key column="associateID"/>
  <many-to-many column="duesID"
    class="org.biketrails.members.Dues"
    unique="true"/>
</set>
```

_9. Next, you need to configure the Dues entity, which will be simpler, since it has no relationships to configure:

a. Open Dues.hbm.xml into the editor.

b. Change the *generator* to **identity**.

_10. Create a new class named **CreateMembers** in the **org.biketrails.test** package with a *main* method.

Complete the CreateMembers.java class:

a. Open a Hibernate session and start a transaction.

b. Create thee Dues entities and persist them:

```
Dues d1 = new Dues();
d1.setAmount(new BigDecimal(45.00));
d1.setPaymentDate(new Date());
session.save(d1);

Dues d2 = new Dues();
d2.setAmount(new BigDecimal(75.00));
d2.setPaymentDate(new Date());
session.save(d2);

Dues d3 = new Dues();
d3.setAmount(new BigDecimal(500.00));
d3.setPaymentDate(new Date());
session.save(d3);
```


Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

c. Create a couple of Associate entities and persist them:

```
Associate john = new Associate();
john.setEmailAddress("john@biketrails.org");
john.setName("John Jefferson");
session.save(john);

Associate sue = new Associate();
sue.setEmailAddress("sue@goliath.com");
sue.setName("Sue Karmelo");
session.save(sue);
```

d. Create two MembershipList entities and persist them:

```
MembershipList blazers = new MembershipList();
blazers.setListName("Trailblazers");
session.save(blazers);

MembershipList patrons = new MembershipList();
patrons.setListName("Patrons");
session.save(patrons);
```

e. Now you need to establish the relationships. Here's the scenario:

- John has two Dues payments and is on both "Trailblazers" and "Patrons" lists
- Sue has one Dues payment and is only on the "Trailblazers" list

Write this code to establish the relationships:

Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
// John has two Dues, Sue has 1
Collection<Dues> johnDuesList = john.getDuesList();
johnDuesList.add(d1);
johnDuesList.add(d2);
Collection<Dues> sueDuesList = sue.getDuesList();
sueDuesList.add(d3);

// John is on "Trailblazer" and "Patron"
// Sue is on "Trailblazer"
Collection<MembershipList> johnMemberList =
    john.getMembershipList();
johnMemberList.add(blazers);
johnMemberList.add(patrons);
Collection<MembershipList> sueMemberList =
    sue.getMembershipList();
sueMemberList.add(blazers);

Collection<Associate> blazersAssociates =
    blazers.getAssociateList();
blazersAssociates.add(john);
blazersAssociates.add(sue);
Collection<Associate> patronsAssociates =
    patrons.getAssociateList();
patronsAssociates.add(john);
```

f. Display the IDs of the MembershipList entities to the console.

g. Commit the transaction and clean up the session and factory.

_11. Run CreateMembers.java as a Java Application. There should be no errors, but not much interesting output, either.

_12. To verify that Hibernate inserted into the database, follow this procedure:

a. Start the Derby interactive command processor (ij) by running the dbSQL.cmd file.

b. In the ij window, enter:

```
SELECT * FROM Members;
```

You should see a row for each entity your program created. You should see a row for each MembershipList.

c. Then enter:

```
SELECT * FROM Associate;
```

Run the query.

You should see a row for each Associate.

d. Then enter:

Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
SELECT * FROM MEMBER_ASSOCIATE;
```

Run the query.

You should see the *join* table that relates MembershipList and Associate.

e. Then enter:

```
SELECT * FROM Dues;
```

Run the query.

You should see a row for each Dues entity.

f. Then enter:

```
SELECT * FROM ASSOCIATE_DUES;
```

Run the query.

You should see the *join* table that relates Associate and Dues.

_13. Next, write a simple program that will query the entities:

a. Create a new class in the *org.biketrails.test* package named **QueryMembers** with a *main* method.

b. In the *main* method, open a Hibernate session. You can copy and paste this code from CreateMembers.java.

c. Execute a Hibernate query to retrieve all MembershipList entities.

d. Write a loop that displays each list's name the total dues paid by members of the list.

e. Clean up. You can copy and paste this code from CreateMembers.java.

_14. Before you run, open *hibernate.cfg.xml* into the editor and comment-out the **hibernate.hbm2ddl.auto** property. If you left it in, Hibernate would create a new empty table - there would be no entities to query!

_15. Run QueryMembers.java as a Java Application. If you used the values suggested above, the "Trailblazers" total should be 620.00 and the "Patrons" total should be 120.00.

Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.