# Hibernate 3.0 - Exercises

January 8, 2007

# Set Up:

## Pre-requisite

1. Copy the participants' content into your system.
2. Use a machine that has about 200 MB of free space in a single drive.

## Setting up JDK

1. Go to \software folder to get all the software's that you will need for this training
2. If you do not have Java 1.5 in your system then install Java 1.5 using the installable provided and ensure that the JDK is installed in C:\Java\jdk1.5.0_04 (It is preferable that you do not install the JDK in under "program files" or any other path that has a 'space' character in it.)

## Adding jdk path to the 'Path' environment variable

1. Navigate to 'My Computer' on your desktop and 'Right click'
2. Navigate to 'Advanced' tab and click on 'Environment Variables'
3. Find the 'Path' variable defined in the 'Systems Variables' and click on 'Edit'
4. Append '**C:\Java\jdk1.5.0_04\bin;**' to the end of the variable (if it is not already present)
5. Click 'Ok' twice to save the 'Path' variable
6. Navigate to 'Start' | 'Run'
7. Enter 'cmd', which should open a DOS command prompt
8. Type 'cd c:\' and hit enter
9. Type 'java –version' and hit enter
10. You should get the following:
    - *java version "1.5.0_06"*
    - *Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_06-b05)*
    - *Java HotSpot(TM) Client VM (build 1.5.0_06-b05, mixed mode, sharing)*
11. This ensures that your jdk is installed and configured correctly

## Set up Eclipse project

Setup the author application as an eclipse project

1. Open eclipse and start a new Java project called "author_app"
2. Let <drive>:\hibernate-training\app\author_app\ be the root folder of the project
3. Ensure that you setup the jars in the lib folder as project class path
4. Setup /bin as the output folder
5. Setup /src as the source folder

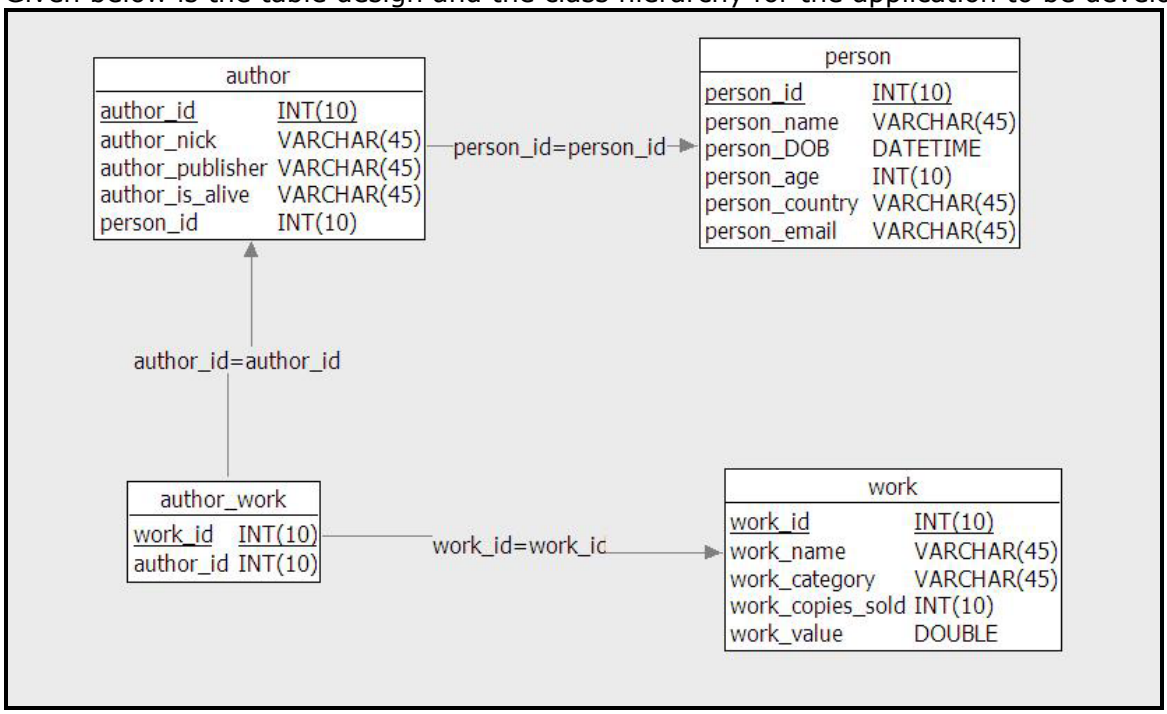
Setup the author web application as an eclipse project

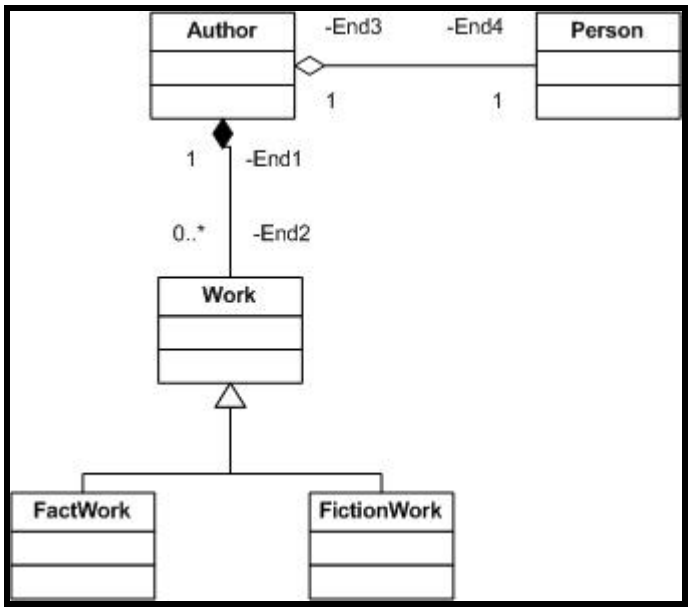
1. Start a new Java project called "author"
2. Let tomcat/webapps/author be the root folder for the shop application
3. Ensure that you setup the WEB-INF/lib in the project class path
4. Setup WEB-INF/classes as the output folder
5. Setup WEB-INF/src as the source folder

Setup the author application as an eclipse project

1. Open eclipse and start a new Java project called "author_annotate"
2. Let <drive>:\hibernate-training\app\author_annotate\ be the root folder of the project
3. Ensure that you setup the jars in the lib folder as project class path
4. Setup /bin as the output folder
5. Setup /src as the source folder

Given below is the table design and the class hierarchy for the application to be developed





# 1.Exercise 1: First Hibernate application

Duration – 30 minutes + 15 minutes

## *Problem statements*

1. Define a Hibernate.cfg.xml to map the application to the Data base.
2. Create a mapping xml to map Person. java to the Person table.
3. Create a Utility class called HIbernateUtil to retrieve a Session as and when required
4. Create an application to insert a new record into the Person table
5. Create an application to list all the records in the Person table

## *Solution tips*

1. Define a hibernate.cfg.xml with the following configurations

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration SYSTEM "hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
<!-- Database connection settings -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/hbn_author</property>
        <property name="connection.username">root</property>
        <property name="connection.password">root</property>
<!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>
<!-- Enable Hibernate's automatic session context management -->
        <property name="current_session_context_class">thread</property>
<!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
<!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>
<!--        <mapping resource="conf/Author.hbm.xml" />
        <mapping resource="conf/Person.hbm.xml" />
        <mapping resource="conf/Work.hbm.xml" />-->
    </session-factory>
</hibernate-configuration>
```

2. Define a java class called person.java in the package com.author.data. The details of the person class is given below

```
package com.author.data;
import java.util.Date;
public class Person {
	private int personId;
	private String personName;
	private Date DOB;
	private int age;
	private String country;
	private String email;
	public int getAge() {
		return age;
	}
	public void setAge(int age) {
		this.age = age;
	}
..
..
}
```

3. Based on the person table in the data base define a mapping file that maps the person class to the person table. Name the file person.hbm.xml and put into the folder conf.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping SYSTEM "hibernate-mapping-3.0.dtd">
<hibernate-mapping>

	<class name="com.author.data.Person" table="person">
		<id name="personId" column="person_id" type="int">
			<generator class="native" />
		</id>
		<property name="personName" column="personName" />
		<property name="DOB" column="person_DOB" />
		<property name="age" column="person_age" />
		<property name="country" column="person_country" />
		<property name="email" column="person_email" />
	</class>

</hibernate-mapping>
```

4. Review the HibernateUtil class provided with the source code

```
package com.author.hbn;
```

```
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    static {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Make sure you log the exception, as it might be swallowed
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

5. Write an application that can add an object of type person. Create an application under the package com.author.app. Name the class AddPerson.java
6. The code to add a person and retrieve a list of persons is given below

```
package com.author.app;

import java.util.GregorianCalendar;
import java.util.List;
import org.hibernate.Session;
import com.author.data.Person;
import com.author.hbn.HibernateUtil;

public class AddPerson {

      public static void main(String[] args) {
            AddPerson ap = new AddPerson();
            ap.addPerson();
            ap.listPersons();
      }

private void listPersons() {
```

```
Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
ses.beginTransaction();
List <Person>lst =  ses.createQuery("from com.author.data.Person").list();
for (Person pers: lst){
      System.out.println(" person is " + pers.getPersonId() + "   " +
pers.getPersonName() + "\n");
}
ses.getTransaction().commit();
}

      private void addPerson() {

            Person p = new Person();
            p.setAge(12);
            p.setCountry("India");
            p.setDOB(new GregorianCalendar(12,12,1977).getTime());
            p.setEmail("manoj@test.com");
            p.setPersonName("Manoj K");

            Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
            ses.beginTransaction();
            ses.save(p);
            ses.getTransaction().commit();
      }
}
```

==========

# 2. Exercise 2: Unidirectional One-2-One mapping

Duration – 45 minutes +15

## *Problem statements*

1. Define a one-2-one mapping using the foreign key relationship between Author and Person
2. Use a Java application to add a few authors and add associated persons details
3. List authors and the associated person details too

## *Solution tips*

1. Create a class called Author.java in the data package. Create fields in the Author class to map onto the Author table.

```
public class Author {

	private int authId;
	private String nickName;

	private String publisherName;
	private String isAlive;
	private Person person;
	private Set work;



	public int getAuthId() {
		return authId;
	}
	public void setAuthId(int authId) {
		this.authId = authId;
	}
..
..
}
```

2. Create a mapping file called author.hbm.xml that maps the author class to the author table. Put the mapping file into the conf folder

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping SYSTEM "hibernate-mapping-3.0.dtd">
<hibernate-mapping>

	<class name="com.author.data.Author" table="author">
		<id name="authId" column="author_id" type="int">
```

```
                    <generator class="native" />
                </id>
                <property name="nickName" column="author_nick" />
                <property name="publisherName" column="author_publisher" />
                <property name="isAlive" column="author_is_alive" />
                <many-to-one name="person" column="person_id" unique="true"
                        not-null="true"/>
        </class>


</hibernate-mapping>
```

3. Create a java application called AddAuthPerson into the app package. Use this class to add rows into the author and person tables.

```
public class AddAuthPerson {
        public static void main(String[] args) {
                AddAuthPerson aap = new AddAuthPerson();
                aap.addAuthPerson();
                aap.getListOfAuthPersons();
        }


private void getListOfAuthPersons() {
Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
ses.beginTransaction();
List <Author> lst = ses.createQuery("from com.author.data.Author").list();
for (Author auth: lst){
        Person p = auth.getPerson();
        System.out.println(" Author is " + auth.getAuthId() + "  nick name is " +
auth.getNickName()  +
        "||person is " + p.getPersonId() + "  person name is " + p.getPersonName() +
"\n\n");
}
ses.getTransaction().commit();
}


private void addAuthPerson() {
                Author auth = getAuth();
                Person p = getPerson();
                auth.setPerson(p);
                Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
                ses.beginTransaction();
                ses.save(p);
                ses.save(auth);
```

```
                ses.getTransaction().commit();
        }
        private Person getPerson(){
                Person p = new Person();
                p.setAge(12);
                p.setCountry("India");
                p.setDOB(new GregorianCalendar(12,12,1977).getTime());
                p.setEmail("james@java.com");
                p.setPersonName("James Gosling");
                return p;
        }
        private Author getAuth(){
                Author auth = new Author();
                auth.setIsAlive("Y");
                auth.setNickName("Techdude");
                auth.setPublisherName("Harper Collins");
                return auth;
        }
}
```

4. Use a separate function to list the various authors and the associate data of these authors in the Person table

# 3.Exercise 3 – one-2-many association with join tables

Duration – 1 hour + 15 minutes

## Problem statement

1. Define a one-2-many relation ship between Author and Work using join table
2. Use a java application to add a few works for an author
3. Use a java application to list all the works of all the authors

## Solution tips

1. Modify the Author.hbm.xml file to ensure that the author class shows the mapping with the Work class.

```
<class name="com.author.data.Author" table="author">
..
..
<set name="work" table="author_work" cascade="save-update,delete">
                        <key column="author_id" unique="false" />
                        <many-to-many column="work_id" unique="true"
                                class="com.author.data.Work" />
                </set>
</class>
```

2. Ensure that the Author class has a variable to link it to a set of 'Work' objects

```
        private Set work;
```

==========

1. In class AddAuthorPerson, modify the add method to add a author with 2 works against his credit. The modified add method is given below. Modify the addAuthPerson method and also add a new method called getWorks(int i) into this application

```
private void addAuthPerson() {
        Author auth = getAuth();
        Person p = getPerson();
        auth.setPerson(p);
        auth.setWork(getWorks(3));
Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
        ses.beginTransaction();
        ses.save(p);
        ses.save(auth);
        ses.getTransaction().commit();
```

```
}
private Set getWorks(int i){
            HashSet hs = new HashSet();
            Work w = null;
            for(int j=0; j<i; j++){
                  w = new Work();
                  w.setWorkName(" Java 1.5 primer");
                  w.setWorkCategory("technical");
                  w.setValue(34.0);
                  w.setCopiesSold(300);
                  hs.add(w);
            }
            return hs;
      }
```

2. modify the list method such that it now lists the works against an author too

```
      private void getListOfAuthPersons() {
            Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
            ses.beginTransaction();
            List <Author> lst = ses.createQuery("from com.author.data.Author").list();
            for (Author auth: lst){
                  Person p = auth.getPerson();
                  System.out.println(" Author is " + auth.getAuthId() + "  nick name is
" + auth.getNickName()  +
                                    "||person is " + p.getPersonId() + "  person name is " +
p.getPersonName() + "\n");
                  Set s = auth.getWork();
                  Iterator iter = s.iterator();
                  while(iter.hasNext()){
                        Work w = (Work)iter.next();
                        System.out.println(" work is " + w.getWorkName() +
"\n\n");
                  }
            }
            ses.getTransaction().commit();

      }
```

==========

# 4. Exercise 4 – Load, Update and delete

Duration – 1 hour + 15

## *Problem statement*

1. Using the Id of a Author, load, modify and update the author details in an application in the context of a single session
2. Perform the above operation using detached objects. Use merge instead of update to perform the same operation
3. For a given author delete any one works by loading the works of an author

## *Solution tips*

1. Create a new java application to perform the load, update and modifications of the author object. Name the class AuthorManage and add it to the app package
2. Create the method loadAndPersistAuthor that loads the author object, modifies it and saves it back in a single session.

```
	public static void main(String[] args) {
		AuthorManage am = new AuthorManage();
		am.loadAndPersistAuthor();
	}
private Author loadAndPersistAuthor() {
Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
		ses.beginTransaction();

		Author auth = (Author) ses.createQuery("select a from Author a where
a.authId = :aid").setParameter("aid", 10).uniqueResult();
		auth.setIsAlive("N");
		auth.setPublisherName("Moon publications");
		ses.save(auth);
		ses.getTransaction().commit();
		return auth;
	}
```

==========

1. Now create 2 methods loadAuthor and persistAuthor.

```
	private void persistAuthor(Author auth) {
		Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
		ses.beginTransaction();
		Author newAuth  = (Author) ses.load(Author.class, new Integer(10));
		newAuth.setIsAlive("Y");
		//ses.update(auth);
		ses.merge(auth);
		ses.getTransaction().commit();
```

```
	}



	private Author loadAuthor() {
		Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
		ses.beginTransaction();
		Author auth = (Author) ses.createQuery("select a from Author a where
a.authId = :aid").setParameter("aid", 10).uniqueResult();
		ses.getTransaction().commit();
		ses = null;
		return auth;
	}
```

2. Invoke loadAuthor first from the main method and then modify it. Then send this modified object to persistAuthor. In new session attempt to update the modified Author object

```
	public static void main(String[] args) {
		AuthorManage am = new AuthorManage();
		Author auth = am.loadAuthor();
		auth.setNickName(auth.getNickName() + "_New");
		auth.setPublisherName("Rambo publications");
		am.persistAuthor(auth);
	}
```

========

1. Create a method call deleteLastWork().

```
	private void deleteLastWork() {
		Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
		ses.beginTransaction();

		Author auth = (Author) ses.load(Author.class, new Integer(9));
		Set s = auth.getWork();

		Iterator iter = s.iterator();
		Work w = null;
		while(iter.hasNext()){
			w = (Work) iter.next();
		}
		ses.delete(w);
		s.remove(w);
		ses.update(auth);
		ses.getTransaction().commit();
```

```
        }
```

2. Invoke this method from the applications main method.

```
        public static void main(String[] args) {
                AuthorManage am = new AuthorManage();
                am.deleteLastWork();
        }
```

==========

# 5. Exercise 5: selective update and cascade

Duration: 30 mts + 15 mts

## *Problem statement*

1. Setup email ID attribute of the person class as a non-updateable field. Test the attribute settings
2. Setup the cascade property of the author to person relationship such that when a author is deleted, the associated person object is not removed

## *Solution tips*

1. In the person.hbm.xml, mapping file modify the property email and setup a new attribute called update with the value as false

```
<property name="email" update="true" column="person_email" />
```

2. Create a new class called PersonManage.java. Define this as an application. In this class create a new method called update person
3. In the updatePeson method, load a new person object based on the person ID. Update the email Id of this object and invoke session.udpate

```
public class PersonManage {
	public static void main(String[] args) {
		PersonManage am = new PersonManage();
		am.updatePerson();
	}
	private void updatePerson() {
		Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
		ses.beginTransaction();
		Person prs = (Person)ses.load(Person.class, new Integer(23));
		prs.setEmail("email@email.com");
		ses.update(prs);
		ses.getTransaction().commit();
	}
}
```

4. Perform this operation with the update attribute in the person.hbm.xml set to true and false

==========

1. Create a new method in the class PersonManage class to delete an particular author

```
public class PersonManage {
	public static void main(String[] args) {
		PersonManage am = new PersonManage();
```

```
                //am.updatePerson();
                am.deleteAuthor();
        }
…

        private void deleteAuthor() {
                Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
                ses.beginTransaction();
                Author auth = (Author)ses.load(Author.class, new Integer(18));
                ses.delete(auth);
                ses.getTransaction().commit();
        }
}
```

2. Modify the author.hbm.xml file and define a new cascade attribute for the <many-to-one.. tag. Setup the vbalue of the attribute as "save-update". Test the application

```
                <many-to-one name="person" column="person_id" unique="true"
                        not-null="true" cascade="save-update "/>
```

3. Change the value of the attribute to "save-update, delete" and test the application

```
                <many-to-one name="person" column="person_id" unique="true"
                        not-null="true" cascade="save-update, delete"/>
```

=====

# 6. Exercise 6 – bidirectional one-2-one mapping

Duration – 30 minutes +15

## Problem statement

1. Define a bidirectional mapping using the foreign key relationship between Author and Person
2. List the authors and the associated persons.
3. Test the 2 way linkage between the objects

## Solution tips

1. In order to work on bidirectional mapping recreate the Author, Person and Work java classes. Rename these classes as AuthorBD, WorkBD and PersonBD. Put these new java files into a new package called com.author.bidirection.data.

```
public class AuthorBD {
	private int authId;
	private String nickName;
	private String publisherName;
	private String isAlive;
	private PersonBD person;
	private Set workBD;
	..
	..
}

public class PersonBD {
	private int personId;
	private String personName;
	private Date DOB;
	private int age;
	private String country;
	private String email;
	private AuthorBD auth;
	..
	..
}

public class WorkBD {
	private int workId;
	private String workName;
	private String workCategory;
	private int copiesSold;
	private Double value;
```

```
        private AuthorBD auth;
        ..
        ..
}
```

2. Create new mapping files for the new classes

```
<class name="com.author.bidirection.data.AuthorBD" table="author">
        <id name="authId" column="author_id" type="int">
                <generator class="native" />
        </id>
        <property name="nickName" column="author_nick" />
        <property name="publisherName" column="author_publisher" />
        <property name="isAlive" column="author_is_alive" />
        <many-to-one name="person" column="person_id" unique="true"
                not-null="true"/>
        <set name="workBD" table="author_work" cascade="save-update,delete">
                <key column="author_id" unique="false" />
                <many-to-many column="work_id" unique="true"
                        class="com.author.bidirection.data.WorkBD" />
        </set>
</class>

<class name="com.author.bidirection.data.PersonBD" table="person">
        <id name="personId" column="person_id" type="int">
                <generator class="native" />
        </id>
        <property name="personName" column="person_name" />
        <property name="DOB" column="person_DOB" />
        <property name="age" column="person_age" />
        <property name="country" column="person_country" />
        <property name="email" column="person_email" />
        <one-to-one name="auth"
                        property-ref="person"/>
</class>
```

3. Refer to the new mapping files in the hibernate configuration file.

```
<!-- <mapping resource="conf/Author.hbm.xml" />
        <mapping resource="conf/Person.hbm.xml" />
        <mapping resource="conf/Work.hbm.xml" />-->



        <mapping resource="conf/AuthorBD.hbm.xml" />
```

```
        <mapping resource="conf/PersonBD.hbm.xml" />
```

4. Create a new application called Bidirection.java in the package app. Define a method in the class Bidirection to retireve and author. Use the getter method in AuthorBD and PersonBD classes to navigate between the 2 objects.

```
public class Bidirection {


       public static void main(String[] args) {
              Bidirection bd = new Bidirection();
              bd.listAuthorsAndPersons();
       }


private void listAuthorsAndPersons() {
Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
       ses.beginTransaction();
AuthorBD abd = (AuthorBD)ses.createQuery("select a from AuthorBD a where a.authId =
:aid").setParameter("aid", new Integer(6))
              .uniqueResult();
System.out.println(" author is " + abd + "  auth Id is  " + abd.getAuthId());


PersonBD pbd = abd.getPerson();
System.out.println(" person is " + pbd + "  personID  is " + pbd.getPersonId());


AuthorBD abd1 = (AuthorBD)pbd.getAuth();
System.out.println(" author referd by person is " + abd1 + " auth id is " +
abd1.getAuthId());


ses.getTransaction().commit();
}


}
```

# 7. Exercise 7 – bidirectional one-2-many relationship

Duration – 30 minutes + 15

## *Problem statement*

1. Define a bidirectional one-2-many relationship between a Author and his works.
2. Navigate from a Author to any of his works and backwards

## *Solution tips*

1. Use the same WorkBD and AuthorBD classes alredy defined in the previous exercises for this exercises too.
2. The AuthorBD mapping needs no change. Define a mapping for WorkBD class.

```
	<class name="com.author.bidirection.data.WorkBD" table="work">
		<id name="workId" column="work_id" type="int">
			<generator class="native" />
		</id>
		<property name="workName" column="work_name" />
		<property name="workCategory" column="work_category" />
		<property name="copiesSold" column="work_copies_sold" />
		<property name="value" column="work_value" />
		<join table="author_work" inverse="true" optional="true">
			<key column="work_id" />
			<many-to-one name="auth" column="author_id" not-null="true" />
		</join>
	</class>
```

3. Refer to the new WorkBD mapping in the hibernate configuration

```
<!-- <mapping resource="conf/Author.hbm.xml" />
		<mapping resource="conf/Person.hbm.xml" />
		<mapping resource="conf/Work.hbm.xml" />-->



		<mapping resource="conf/AuthorBD.hbm.xml" />
		<mapping resource="conf/PersonBD.hbm.xml" />
		<mapping resource="conf/WorkBD.hbm.xml" />
```

4. Define a new method called listAuthorsAndworks in the application Bidirection. Invoke this new method from the main method

```
public class Bidirection {


	public static void main(String[] args) {
		Bidirection bd = new Bidirection();
```

```
            bd.listAuthorsAndworks();
      }

      private void listAuthorsAndworks() {
Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
      ses.beginTransaction();

AuthorBD abd = (AuthorBD)ses.createQuery("select a from AuthorBD a where a.authId =
:aid").setParameter("aid", new Integer(12))
.uniqueResult();
System.out.println(" author is " + abd + "  auth Id is  " + abd.getAuthId());

Set wset = abd.getWorkBD();
Object[] oarr= wset.toArray();
WorkBD wobj1 = (WorkBD)oarr[0];
WorkBD wobj2 = (WorkBD)oarr[1];

System.out.println(" work object is " + wobj1 + "   work Id is " + wobj1.getWorkId());
System.out.println(" work object is " + wobj2 + "   work Id is " + wobj2.getWorkId());

AuthorBD abd1 = wobj1.getAuth();
AuthorBD abd2 = wobj2.getAuth();

System.out.println(" author referred by person is " + abd1 + " auth id is " +
abd1.getAuthId());
System.out.println(" author referred by person is " + abd2 + " auth id is " +
abd2.getAuthId());
}
```

# 8. Exercise 8 – inheritance mapping

Duration: 1 hour + 15 mts

## *Problem statements*

1. Define an Inheritance mapping for the works table.
2. A work can be of type fact or fiction. Map each work to Fictionwork.java or FactWork.java based on the type of work.
3. Use a java application to load all the works in the the DB and shows the count of fact and fiction works.

## *Solution tips*

1. Define 2 new classes calles Factwork and fictionWork in a package called com.author.data.subclass. The 2 classes should extend from the class Work.

```
public class FictionWork extends Work {
}


public class FactWork extends Work {
}
```

2. Modify the Work.hbm.xml. Map the data in the table to Fact or Fiction work based on the discriminatory column work_category. Use the class factwork for rows that have the value 'fact' and FictionWork for the rows that have the value 'Fiction'

```
	<class name="com.author.data.Work" table="work">
		<id name="workId" column="work_id" type="int">
			<generator class="native" />
		</id>
		<discriminator column="work_category" type="string"/>
		<property name="workName" column="work_name" />
		<property name="copiesSold" column="work_copies_sold" />
		<property name="value" column="work_value" />

		<subclass discriminator-value="Fact"
			name="com.author.data.subclass.FactWork">
			</subclass>
		<subclass discriminator-value="Fiction"
			name="com.author.data.subclass.FictionWork">
			</subclass>
	</class>
```

3. Create a new application to load all the rows from work and check their data type. Increment counter to determine the various types of work in the DB

```
public class TypesOfWork {
	public static void main(String[] args) {
```

```
		TypesOfWork tow = new TypesOfWork();
		tow.getWorks();
	}
	private void getWorks() {
		Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
		ses.beginTransaction();
		int factworkcnt = 0;
		int fictionworkcnt = 0;
		List <Work>lst = ses.createQuery("from Work").list();
		for(Work wk: lst){
			System.out.println(" work is " + wk);
			if (wk instanceof FactWork) {
				factworkcnt++;
			}
			else if(wk instanceof FictionWork){
				fictionworkcnt ++;
			}
		}
	System.out.println(" fact work count is " + factworkcnt);
	System.out.println(" fiction work count is " + fictionworkcnt);
		ses.getTransaction().commit();
	}
}
```

# 9. Exercise 9 – Use HQL to perform queries

Duration – 1 hour + 15 minutes

## *Problem statements*

1. Create a java application and define separate methods to perform the following query actions
    a. Obtain a list of authors who have more than 3 books to their credit
    b. Obtain the list of the books that is priced less than the average price of all books
    c. Obtain the average age of an author
    d. Obtain the details of an author given his full name

## *Solution tips*

1. Define a new class called HQLQueries in the app package
2. define a method each for the above requirements and invoke those methods from the application's main method
3. The code for the application is given below

```
public class HQLQueries {
	public static void main(String[] args) {
		HQLQueries hql = new HQLQueries();
		hql.getAuthorList(3);
		hql.getMaxWorksAuthor();
		hql.getAuthorAvgAge();
		hql.searchAuthor();
	}
public void getAuthorList(int bookCount){
Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
	ses.beginTransaction();
	List lst = ses.createQuery("from Author a where size(a.work) > 3").list();
	System.out.println(" No. of authors with more than 3 works is " + lst.size());
	ses.getTransaction().commit();
}
public void getMaxWorksAuthor(){
Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
	ses.beginTransaction();
	List lst = ses.createQuery("from Work where value < (select avg(value) from
Work)").list();
	System.out.println(" the no. of books that are cheaper than the average price is "
+ lst.size());
	ses.getTransaction().commit();
	}
public void getAuthorAvgAge(){
Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
	ses.beginTransaction();
```

```
        Float f = (Float)ses.createQuery("select avg(a.person.age) from Author a
").uniqueResult();

        System.out.println(" The average age of all authors is " + f.floatValue());

        ses.getTransaction().commit();

}

public void searchAuthor(){

Session ses = HibernateUtil.getSessionFactory().getCurrentSession();

        ses.beginTransaction();

        Author auth  = (Author) ses.createQuery("select a from Author a where
a.person.personName = :name")

        .setParameter("name", "Steve Jobs")

        .uniqueResult();

        System.out.println(" The author you are searching for has the ID  " +
auth.getAuthId());

        ses.getTransaction().commit();

        }

}
```

# 10. Exercise 10 – Use Criteria Queries and Native SQL

Duration – 1 hour + 15 minutes

## *Problem statements*

1. Create a java application and obtain the names of authors who are below the age of 30 and have atleast 3 books to their credit
2. Perform a similar operation using native SQL. Get names of authors whose age is less than 30

## *Solution tips*

1. Define a new java class called Critquery under the package app to write the new queries
2. Define 2 functions has shown below to execute the criteria query and the SQL query
3. invoke the methods from the application's main method

```
public class Critquery {
	public static void main(String[] args) {
		Critquery cq = new Critquery();
		cq.executeCriteriaQuery();
		cq.executeNativeQuery();
	}
private void executeCriteriaQuery() {
Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
	ses.beginTransaction();
	Criteria ct = ses.createCriteria(Author.class)
	.createAlias("person", "p")
	.add(Restrictions.le("p.age", new Integer(20)))
	.add(Expression.sizeGt("work", new Integer(3)));
	List lst = ct.list();
	System.out.println(" No. of authors who fit the criteria is " + lst.size());
	ses.getTransaction().commit();
}
private void executeNativeQuery() {
Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
	ses.beginTransaction();

	List lst = (List) ses.createSQLQuery(" select * from author a, person p  where
a.person_id = p.person_id and p.person_age < 20").list();
	System.out.println(" No. of authors who fit the criteria is " + lst.size());
	ses.getTransaction().commit();

}
}
```

# 11. Exercise 11 – Events and interceptors

Duration: 30 mts + 15 mts

## *Problem statements*

1. Define an interceptor scoped at the session level and intercept the save event of the author Print out the ID of the row created by the save operation.
2. Define a event listener for the pre-update operation of the person object and log the complete details of the object

## *Solution tips*

1. In the method "addAuthPerson' in class AddAuthPerson.java, modify the line of code where the session is initialized and add a new SaveInterceptor to the session

```
Session ses = HibernateUtil.getSessionFactory().openSession(new SaveInterceptor());
```

2. Create a new class called SaveInterceptor in the package "com.author.listener"

```
public class SaveInterceptor extends EmptyInterceptor {
}
```

3. Over ride the method onSave() in this class. Access the object to be saved and modify the

```
	public boolean onSave(Object entity,
		Serializable id,
		Object[] state,
		String[] propertyNames,
		Type[] types) {

		 if (entity instanceof Author) {
			System.out.println(" inside the if loop ");
			for ( int i=0; i<propertyNames.length; i++ ) {
		    if ( "nickName".equals( propertyNames[i] ) ) {
			state[i] = state[i] + "X";
			return true;
		    }
		}
		 }
		 return true;
	}
```

==========

1. Edit the hibernate.cfg.xml and add the event listener for the pre-update event

```
	<event type="pre-update">
		<listener class="com.author.listener.UpdateListener"/>
	</event>
```

2. Create a class called UpdateListener in the package "com.author.listener". Over ride the method onPreUpdate()

```
public class UpdateListener implements PreUpdateEventListener {


	public boolean onPreUpdate(PreUpdateEvent evnt) {
		Object obj = evnt.getEntity();
		System.out.println(" here in preUpdate " + obj);
		if(obj instanceof Person){
			Person p = (Person)obj;
			System.out.println(" Id is " + p.getEmail() );
		}
		return false;
	}
}
```

3. From this method modify the return value as a true and a false and test the outcome

==========

# 12. Exercise 12 - Hibernate with a web application

Duration – 45 minutes + 15 minutes

## Problem statements

1. Define a web application with hibernate configuration and appropriate data classes
2. Use a JSP page that submits data to another JSP and adds a new book into the DB and assigns it to an existing author

## Solution tips

1. Set up a web project that refers to the webapp **authors** under tomcat. The project setup information is provided in the setup section in this document
2. The AuthorAddForm.jsp has been developed with various fields required to create a new author, the associated book and the person details.
3. The mapping files and the data objects have been copied from the previous project
4. The data submitted to the AuthorAdd.jsp should be used to compose the hierarchy of author, person and work. Invoke the create method of AuthorDao to save this to the DB

```
<%@ page contentType="text/html; charset=ISO-8859-1" %>
<%@ page import="com.author.data.*, com.author.dao.*" %>
<%@ page import="java.util.*" %>
<%
try{

        Person p = new Person();
        Author auth = new Author();
        Work w = new Work();
        AuthorDao adao = new AuthorDao();

        p.setPersonName(request.getParameter("pname"));
        p.setCountry(request.getParameter("pcntry"));
        p.setAge(new Integer(request.getParameter("page")).intValue());
        p.setEmail(request.getParameter("pemail"));
        p.setDOB(new Date());

        auth.setPerson(p);
        auth.setNickName(request.getParameter("nname"));
        auth.setIsAlive(request.getParameter("isalive"));
        auth.setPublisherName(request.getParameter("pubname"));

        w.setWorkName(request.getParameter("wname"));
        w.setWorkCategory(request.getParameter("wcat"));
        w.setValue(new Double(request.getParameter("wvalue")));
        w.setCopiesSold(new Integer(request.getParameter("wsold")).intValue());
```

```
        Set s = new HashSet();
        s.add(w);
        auth.setWork(s);
        adao.create(auth);
}
catch(Exception e){
        e.printStackTrace();
%>
        <jsp:forward page="failure.jsp"/>
<%
}
%>
        <jsp:forward page="success.jsp"/>
```

5. Develop the AuthorDao.create method to save the data into the DB

```
public class AuthorDao {

public void create(Author auth) {
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
session.save(auth.getPerson());
session.save(auth);
session.getTransaction().commit();
}
}
```

6. Redirect user to success / failure page based on the result from the create method

# 13. Exercise 13: Use Hibernate Annotations

Duration: 1 hour + 15 mts

## Problem statement

1. use hibernate annotations to map author, person and work to their appropriate tables
2. create a simple application to create an author and his works

## Solutions tips

1. Use the project called "author_annotate" for this exercise
2. Define the hibernate.cfg.xml and ensure that the mapping is pointing to the classes

```
<!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>
              <mapping class="com.author.data.Author"/>
              <mapping class="com.author.data.Person"/>
              <mapping class="com.author.data.Work"/>
    </session-factory>
</hibernate-configuration>
```

3. Define a new HibernateUtil such that it uses the AnnotationSessionFactory

```
    static {
        try {
            sessionFactory = new
AnnotationConfiguration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            ex.printStackTrace();
            throw new ExceptionInInitializerError(ex);
        }
    }
```

4. Map the Author, Person and work class to their appropriate tables

```
@Entity
@Table (name="author")

public class Author {

       private int authId;
       private String nickName;
       private String publisherName;
       private String isAlive;
       private Person person;
       private Set work;
```

```
   @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
   @Column(name="author_id")
   public int getAuthId() {
          return authId;
   }

   @Column(name="author_is_alive")
   public String getIsAlive() {
          return isAlive;
   }
   @Column(name="author_nick")
   public String getNickName() {
          return nickName;
   }
   @Column(name="author_publisher")
   public String getPublisherName() {
          return publisherName;
   }

   @OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name="person_id")
   public Person getPerson() {
          return person;
   }

   @OneToMany(cascade = CascadeType.ALL)
@JoinTable(
        name="author_work",
        joinColumns = { @JoinColumn( name="author_id") },
        inverseJoinColumns = @JoinColumn( name="work_id")
)
   public Set<Work> getWork() {
          return work;
   }
```

1. Define a new class called AddPerson. Define a main method in the class
2. Instantiate the author, person and work objects. Setup the associations between these objects

3. Uses session.save to add these objects to the DB

```
public class AddPerson {

	public static void main(String[] args) {
		AddPerson ap = new AddPerson();
		ap.addAuthPerson();
	}
	private void addAuthPerson() {
		Author auth = getAuth();
		Person p = getPerson();
		auth.setPerson(p);
		auth.setWork(getWorks(2));
		Session ses = HibernateUtil.getSessionFactory().getCurrentSession();
		ses.beginTransaction();
		ses.save(p);
		ses.save(auth);
		ses.getTransaction().commit();
	}

	private Person getPerson(){
		Person p = new Person();
		p.setAge(12);
		p.setCountry("India");
		p.setDOB(new GregorianCalendar(12,12,1977).getTime());
		p.setEmail("gavin@annotation.com");
		p.setPersonName("Gavin King");
		return p;
	}

	private Set getWorks(int i){
		HashSet hs = new HashSet();
		Work w = null;
		for(int j=0; j<i; j++){
			//w = new FactWork();
			w = new Work();
			w.setWorkName(" Java 1.5 primer");
			w.setWorkCategory("Fact");
			w.setValue(34.0);
			w.setCopiesSold(300);
			hs.add(w);
```

```
		}
		return hs;
	}
	private Author getAuth(){
		Author auth = new Author();
		auth.setIsAlive("Y");
		auth.setNickName("Techdude");
		auth.setPublisherName("Harper Collins");
		return auth;
	}


}
```

====