



Hibernate - VI

Mappings



Agenda

1 Mappings

2 Demo Examples on Associations

Mappings



Objectives

In this module you will learn about:

- Basic O/R mapping
- Collection mapping
- Association mapping
- Inheritance mapping

Basic O/R Mapping

- As discussed earlier ORM is the core idea behind Hibernate which helps in converting objects in domain model to the records in a relational model
- Domain model represents relations in terms of object references and Relational model represents relations in terms of keys (primary & foreign keys)
- Hibernate helps in bridging the gap between the Domain model and Relational model
- Hibernate uses an XML file to do the mappings from object to record
- A separate hibernate mapping (.hbm.xml) file is preferred for each Java persistent class
- We shall take an example and understand some of the basic O/R mapping

Basic O/R Mapping (Contd.).

```
package firstex;

public class Actor {
    private int id;
    private String name;

    public Actor() {
    }

    public int getId() {
        return this.id;
    }

    private void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Basic O/R Mapping (Contd.).

Table schema for the persistent class user is:

```
create table Actor (id number(5), name varchar(15), primary key(id));
```

And the corresponding mapping file `Actor.hbm.xml` file has following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="firstex.Actor" table="ACTOR" schema="SCOTT">

        <id name="id" type="int" unsaved-value="0">
            <column name="ID" precision="5" scale="0" />
            <generator class="increment" />
        </id>

        <property name="name" type="string"/>
            <column name="NAME" length="15" />
        </property>
    </class>
</hibernate-mapping>
```

Basic O/R Mapping (Contd.).

- We shall look into the Actor.hbm.xml file and understand the mapping details. The mapping language is Java-centric, meaning that mappings are constructed around persistent class declarations, not table declarations.

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

- This line mentions the Document Type Definition (DTD) for the XML mapping file. **Doctype** - All XML mappings should declare the doctype shown. The actual DTD may be found at the URL above, in the directory hibernate-x.x.x/src/org/hibernate or in hibernate3.jar. Hibernate will always look for the DTD in its classpath first.

Basic O/R Mapping (Contd.).

- The opening and closing tags for hibernate mapping are written as:
<hibernate-mapping> </hibernate-mapping>
- The details of name of the persistent class and the corresponding table name is provided in the class tag: **<class
name="firstex.Actor" table="ACTOR">.... </class>**
- Each property of the class will be associated with a column in the database table, the details of which is given in the **property** tag.

Basic O/R Mapping (Contd.).

- **id element:** Represents the primary key of the table. Hibernate enforces us to have a primary key for every table. In case where a natural key is not available, we have to provide a key called surrogate key which functions as primary key. Hibernate picks up smart default values if some of the attributes of the tag are ignored by the developer. If we omit the type attribute in the id tag or in the property tag hibernate uses reflection and assigns the proper type for that attribute. If table attribute is omitted in the class tag, the value defaults to the name of the persistent class. The **id element** also describes how the key value is generated.
- Generator element is used to mention the type of key generation strategy. The `<generator>` child element names a Java class used to generate unique identifiers for instances of the persistent class. Hibernate provides a range of built-in implementations. There are shortcut names for the built-in generators. Few of them are given in the next slide.


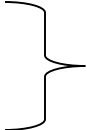
Basic O/R Mapping (Contd.).

- **Assigned identifiers** `<generator class="assigned"/>` - lets the application to assign an identifier to the object before `save()` is called (as opposed to having Hibernate generate them). This special generator will use the identifier value already assigned to the object's identifier property. This is the default strategy if no `<generator>` element is specified.
- **Increment** `<generator class="increment"/>` - generates identifiers of type long, short or int that are unique only when no other process is inserting data into the same table. *Do not use in a cluster.*
- **foreign** - uses the identifier of another associated object. Usually used in conjunction with a `<one-to-one>` primary key association.
- The **unsaved-value attribute** describes the value of the id property for transient instances of class

Collection Mapping

In the examples of the persistent classes considered earlier all the properties were associated with single values

When we need to store a group of values into a property, we use types like:

- List
 - Array
 - Map
- 
- Have indices
-
- Set
 - Bag
- 
- Do not have indices

Collection Mapping

- Since our collections are used to store values (e.g. string, integer, double) but not persistent objects, they are also called “collections of values”. These types listed above are termed as collections. Each one of the type has their own way of storing and retrieving the data back. Some of the types like **Map**, **List**, **Array** have **indices** whereas **Set** and **Bag** does not have **indices**. The collection can hold either primitive types, or objects, or components.
- Hibernate provides different mapping elements for all the listed collections and for their different way of handling the data. We shall understand the different types of mapping elements by looking at some examples.

Collection Mapping - Set

Example for Mapping value **Set**

```
import java.util.*;
public class Album {
    private Integer id;
    private String name;
    private Set pictures = new HashSet();

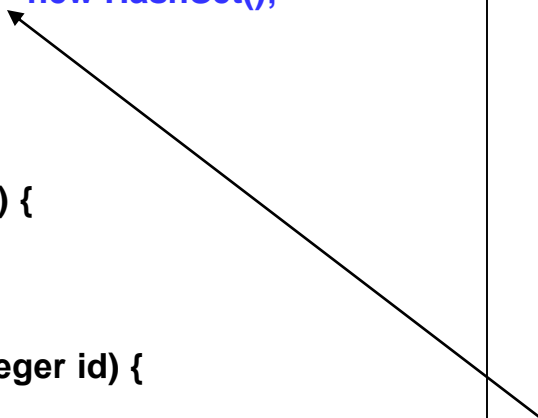
    public Album() {
    }

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```



```
public Set getPictures() {
    return pictures;
}

public void setPictures(Set pictures) {
    this.pictures = pictures;
}
}
```

- Add a collection of value typed objects
- The type is String & collection is Set

Mapping element of java.util.Set is <set>

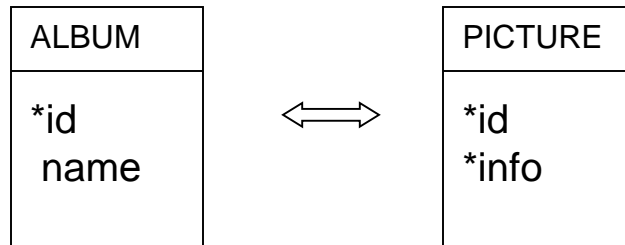
Collection Mapping – Set (Contd.).

- A <set> is an unordered collection which can only store unique objects. It means no duplicated elements can be contained in a set. When you add the same element to a
- set for second time, it will replace the old one. A set which is unordered by default can be asked to be sorted.
- The corresponding type of a <set> in Java is `java.util.Set`.
- We add a collection of value typed objects to the Album entity. We want to store pictures, so the type we use is `String`, and the collection is a `Set`.
- The Album class has two attributes: the name of the album and a `Set` for all the pictures that are part of the album.
- Note: Sets are expected to be the most common kind of collection in Hibernate applications. This is because the "set" semantics are most natural in the relational model.

Collection Mapping – Set (Contd.).

Collection of Values – Set

Example: An Album contains Pictures



Tables:

```
create table album(  
id number(5) not null primary key,  
name varchar(15)  
);
```

```
create table picture(  
id number(5),  
info varchar(100),  
primary key(id,info)  
);
```

- Primary key of collection table is a composite key of both columns ID & INFO
- Cannot have duplicate pictures per album

Collection Mapping – Set (Contd.).

- Note: The primary key of the collection table is now a composite key using both columns, i.e. composite of ID and INFO. This also implies that there can't be duplicate pictures per album, which is exactly the semantics needed for a set in Java.

- For example if you give these 3 set of statements:

```
Set pictures = new HashSet();
```

```
    pictures.add("Supremacy");
```

```
    pictures.add("Identity");
```

```
    pictures.add("Identity");
```

Then it is just 2 that would be added to the set.

[Identity, Supremacy]

Collection Mapping – Set (Contd.).

Album.hbm.xml file for the persistent class Album

- Hibernate creates a foreign key based on parent's identifier
- It calls foreign key a *collection key*; & is defined using <key> element
- Value of column attribute is the name of identifier column in the parent class

```
<hibernate-mapping>
  <class name="pack1.Album" table="ALBUM" schema="SCOTT">
    <id name="id" type="int" unsaved-value="0">
      <column name="ID" precision="5" scale="0" />
      <generator class="increment" />
    </id>
    <property name="name" type="string">
      <column name="NAME" length="15" />
    </property>
    <set name="pictures" table="PICTURE" order-by="info">
      <key column="ID"/>
      <element type="string" column="INFO"/>
    </set>
  </class>
</hibernate-mapping>
```

The element <element> tells Hibernate that collection of elements is of type String

Column name where String values are stored

Collection Mapping – Set (Contd.).

- The Hibernate mapping element used for mapping a collection depends on the type of the interface. For example, a <set> element is used for mapping properties of type Set. Collection instances are distinguished in the database by the foreign key of the entity that owns the collection. In order to keep track of the parent of the pictures collection, **Hibernate creates a foreign key based on the parent's identifier. Hibernate calls this foreign key a *collection key*; it's defined using a <key> element.** The <key> element contains a single attribute called column. The value of the attribute is the name of the identifier column in the parent class. Hibernate also requires a collection table for the elements of the collection (Pugh & Gradecki , 2004).

Collection Mapping – Set (Contd.).

- Look at the mapping of a set.

The **name** attribute value equal to “**pictures**” must be a reference to the set in the persistent class: i.e., Set **pictures** = new HashSet(). The **table** attribute of the set element determines the table name for the collection. The **key** element defines the foreign-key column name in the collection table, i.e., in Picture table. The **element** part tells Hibernate that the collection does not contain references to another entity, but a collection of elements of type String (the lowercase name tells you it's a Hibernate mapping type/converter). The **column** attribute in the element **element** defines the column name where the String values will actually be stored.

Demo Example - Set

Album.java

```
package com.wipro;
import java.util.HashSet;
import java.util.Set;
public class Album {
    private Integer id;
    private String name;
    private Set pictures=new HashSet();
    public Album() {
    }
    public Album(Integer id,String name){
        this.id=id;
        this.name=name;
    }
    public Integer getId() {
        return this.id;
    }
    public void setId(Integer id) {
        this.id=id;
    }
    // Generate all other getter/setter methods
```

Demo Example - Set (Contd.).

Picture.java

```
package com.wipro;
import com.wipro.PictureId;
import java.io.Serializable;
public class Picture implements Serializable {
    private PictureId id;
    public Picture() {
    }
    public Picture(PictureId id) {
        this.id = id;
    }
    public PictureId getId() {
        return this.id;
    }
    public void setId(PictureId id) {
        this.id = id;
    }
}
```

Demo Example - Set (Contd.).

HibernateUtil.java

```
package com.wipro;
import org.hibernate.cfg.Configuration;
import org.hibernate.SessionFactory;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    static {
        try {
            sessionFactory = new Configuration().configure().buildSessionFactory();
        }
        catch(Exception ex) {
            System.out.println("Initial SEssionFactory creation failed :"+ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Demo Example - Set (Contd.).

Client.java

```
package com.wipro;
import java.util.*;
import org.hibernate.*;

public class Client {
    public static void main(String args[]){
        try{
            Session session=HibernateUtil.getSessionFactory().openSession();
            Transaction tx = session.beginTransaction();
            Album a1 = new Album();
            a1.setName("Marriage");
            session.saveOrUpdate(a1);
            Set<String> pictures = new HashSet<String>();
            pictures.add("Haldi");
            pictures.add("Mehandi");
            pictures.add("Sangeet");
            pictures.add("Phera");
            pictures.add("Reception");
            pictures.add("Bidaai");
```

Contd..

Demo Example - Set (Contd.).

```
a1 = (Album) session.load(Album.class, a1.getId());  
a1.setPictures(pictures);
```

```
session.update(a1);
```

```
tx.commit();  
session.close();
```

```
System.out.println("Album details updated successfully..!");
```

```
}
```

```
catch(HibernateException e){
```

```
    e.printStackTrace();
```

```
    System.out.println("Error in adding");
```

```
}
```

```
}
```

```
}
```

Demo Example - Set (Contd.).

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
    <property name="hibernate.connection.driver_class">oracle.jdbc.OracleDriver</property>
    <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
    <property name="hibernate.connection.username">scott</property>
    <property name="hibernate.connection.password">tiger1</property>
    <property name="hibernate.show.sql">true</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="hibernate.current_session_context_class">thread</property>
    <mapping resource="Album.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Demo Example - Set (Contd.).

Album.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.wipro.Album" table="album">
    <id name="id" type="int" unsaved-value="0">
      <column name="ID" />
      <generator class="increment" />
    </id>
    <property name="name" type="string">
      <column name="name" length="15" />
    </property>

    <set name="pictures" table="picture">
      <key column="ID"/>
      <element type="string" column="info"/>
    </set>
  </class>
</hibernate-mapping>
```

Collection Mapping – Map

Example for Mapping value **Map**

```
import java.util.*;
public class Continent {
    private Integer id;
    private String name;
    private Map countries = new
    HashMap();

    public Continent() {
    }
    public Integer getId() {
        return this.id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return this.name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

```
public Map getCountries() {
    return countries;
}
public void setCountries(Map
countries) {
    this.countries = countries;
}
}
```

- Add a collection of value typed objects
- The type is String & collection is Map

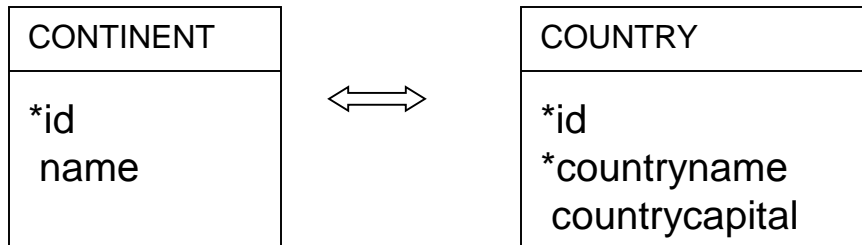
- Mapping element of java.util.Map is <map>
- A map stores its entries in key/value pairs

Collection Mapping – Map

- A <map> uses arbitrary keys to index the collection, not an integer index using in a list. A map stores its entries in key/value pairs. You can lookup the value by its key. The key of a map can be of any data types. The corresponding type of a <map> in Java is `java.util.Map`.

Collection Mapping – Map (Contd.).

Collection of Values - Map



Tables:

```
create table continent(  
id number(5) not null primary key,  
name varchar(15)  
);
```

```
create table country(  
id number(5),  
countryname varchar(100),  
countrycapital varchar(100),  
primary key(id,countryname)  
);
```

Primary key of collection table is a composite key of both columns ID & COUNTRYNAME

Collection Mapping – Map (Contd.).

Continent.hbm.xml file for the persistent class `Continent`

```
<hibernate-mapping>
  <class name="collectionmap.Continent"
    table="CONTINENT" schema="SCOTT">
    <id name="id" type="int" unsaved-value="0">
      <column name="ID" precision="5" scale="0" />
      <generator class="increment" />
    </id>
    <property name="name" type="string">
      <column name="NAME" length="15" />
    </property>
    <map name="countries" table="COUNTRY" fetch="join"
      lazy="false">
      <key column="ID"/>
      <map-key column="COUNTRYNAME" type="string"/>
      <element type="string" column="COUNTRYCAPITAL"
        length="100" not-null="true"/>
    </map>
  </class>
</hibernate-mapping>
```

Specifies information for
the key part of the map

Specifies information for
the value part of the map

Collection Mapping – Map (Contd.).

- The attributes in our Continent class include the **identifier**, the **name** of the object, and a **map** consisting of various key/value pairs. For our mapping document, we've included an `<id>` element that supports an increment generator. We also include a `<property>` element for the name attribute.
- **The new element in the mapping document is called `<map>`; it will be used to map the **countries** attribute value to the database. The `<map>` element includes three subelements—`<key>`, `<map-key>`, and `<element>`—and a single name attribute that tells Hibernate the name of the map attribute in the class we are mapping. **Hibernate uses a separate database table to hold the map key/value pairs.****
- **Next we define the `<key>` subelement** to associate the map with its parent. The attribute **column** value is the name of a foreign key column where Hibernate puts the primary key of the parent.
- The `<map-key>` subelement specifies the information for the key part of the map, and
- the `<element>` subelement is the value part of the map. We've supplied both the column name and the type of the key/value pieces of the map.

Collection Mapping – Map Demo

Continent.java

```
package collection.map;
```

```
import java.util.*;
```

```
public class Continent {  
    private Integer id;  
    private String name;  
    private Map<String, String> countries = new HashMap<String, String>();  
  
    public Continent() {  
  
    }  
  
    // Generate Setter/Getter methods for id and name  
  
    public Map<String, String> getCountries() {  
        return countries;  
    }  
  
    public void setCountries(Map<String, String> countries) {  
        this.countries = countries;  
    }  
}
```

Collection Mapping – Map Demo (Contd.).

Country.java

```
package collection.map;
```

```
public class Country {
```

```
    private Integer id;
```

```
    private String countryname;
```

```
    private String countrycapital;
```

```
    public Country() {
```

```
    }
```

```
    public Country(Integer id, String countryname, String countrycapital) {
```

```
        this.id = id;
```

```
        this.countryname = countryname;
```

```
        this.countrycapital = countrycapital;
```

```
    }
```

```
    // Generate Setter/Getter methods for id, countryname and countrycapital
```

```
}
```

Collection Mapping – Map Demo (Contd.).

HibernateUtil.java

```
package collection.map;

import org.hibernate.cfg.Configuration;
import org.hibernate.SessionFactory;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            sessionFactory = new Configuration().configure().buildSessionFactory();
        }
        catch(Exception ex) {
            System.out.println("Initial SEssionFactory creation failed :"+ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

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

Collection Mapping – Map Demo (Contd.).

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
    <property name="hibernate.connection.driver_class">oracle.jdbc.OracleDriver</property>
    <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
    <property name="hibernate.connection.username">scott</property>
    <property name="hibernate.connection.password">tiger1</property>
    <property name="hibernate.show.sql">true</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="hibernate.current_session_context_class">thread</property>
    <mapping resource="Continent.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Collection Mapping – Map Demo (Contd.).

Continent.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="collection.map.Continent" table="CONTINENT" >
    <id name="id" type="int" unsaved-value="0">
      <column name="ID" precision="5" scale="0" />
      <generator class="increment" />
    </id>
    <property name="name" type="string">
      <column name="NAME" length="15" />
    </property>
    <map name="countries" table="COUNTRY" fetch="join" lazy="false">
      <key column="ID" />
      <map-key column="COUNTRYNAME" type="string" length="15" />
      <element type="string" column="COUNTRYCAPITAL" length="15" not-null="true"/>
    </map>
  </class>
</hibernate-mapping>
```

Collection Mapping – Map Demo (Contd.).

Continent_Country_Client.java

```
package collection.map;
```

```
import java.util.*;
```

```
import org.hibernate.*;
```

```
import org.hibernate.Session;
```

```
public class Continent_Country_Client {
```

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

```
        Session session=null;
```

```
        try{
```

```
            session=HibernateUtil.getSessionFactory().openSession();
```

```
            Transaction tx = session.beginTransaction();
```

```
            Continent a1 = new Continent();
```

```
            a1.setName("Africa");
```

```
            session.saveOrUpdate(a1);
```

```
            String country = "South Africa";
```

```
            String capital = "Pretoria";
```

```
            a1 = (Continent) session.get(Continent.class, a1.getId());
```

Contd..

Collection Mapping – Map Demo (Contd.).

```
Map<String, String> countries = new LinkedHashMap<String, String>();
Country c1 = new Country();
c1.setId(a1.getId());
c1.setCountryname(country);
c1.setCountrycapital(capital);
countries.put(country, capital);
a1.setCountries(countries);
session.update(a1);
tx.commit();
```

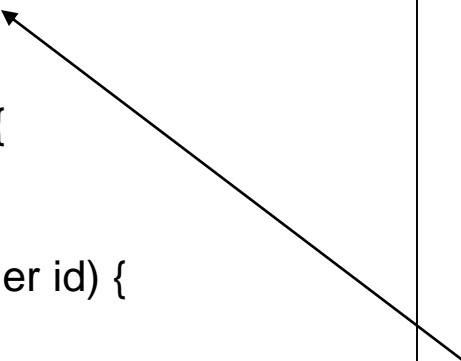
```
System.out.println("Continent details updated successfully..!");
```

```
}
catch(HibernateException e){
    e.printStackTrace();
    System.out.println("Error in adding details");
}
finally{
    session.close();
}
}
}
```

Collection Mapping - Bag

Example for Mapping value **Bag**

```
public class Musiccd {  
    private Integer id;  
    private String name;  
  
    private Collection songs = new ArrayList();  
    public Musiccd() {  
    }  
    public Integer getId() {  
        return this.id;  
    }  
    public void setId(Integer id) {  
        this.id = id;  
    }  
    public String getName() {  
        return this.name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```



```
public Collection getSongs() {  
    return songs;  
}  
  
public void setSongs(Collection songs) {  
    this.songs = songs;  
}
```

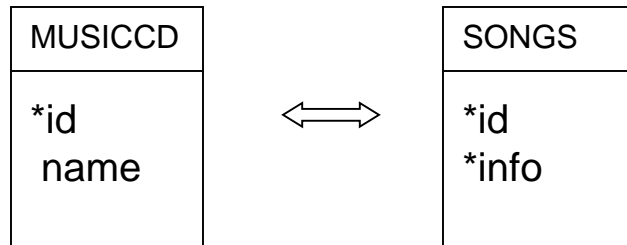
- Add a collection of value typed objects
- The type is String & collection is List

Mapping element of java.util.List can be **<bag>**

Collection Mapping – Bag (Contd.).

Collection of Values – List

Example: A Musiccd contains Songs



Tables:

```
create table musiccd(  
id number(5) not null primary key,  
name varchar(15)  
);
```

```
create table songs(  
id number(5),  
info varchar(100),  
primary key(id,info)  
);
```

- Primary key of collection table is a composite key of both columns ID & INFO

Collection Mapping – Bag (Contd.).

`Musiccd.hbm.xml` file for the persistent class `Musiccd`

```
<hibernate-mapping>
  <class name="collectionbag.Musiccd" table="MUSICCD"
    schema="SCOTT">
    <id name="id" type="int" unsaved-value="0">
      <column name="ID" precision="5" scale="0" />
      <generator class="increment" />
    </id>
    <property name="name" type="string">
      <column name="NAME" length="15" />
    </property>
    <bag name="songs" table="SONG">
      <key column="ID"/>
      <element type="string" column="INFO"/>
    </bag>
  </class>
</hibernate-mapping>
```

- The simplest collection type in Hibernate is a `<bag>`
- It is a list of objects without ordering, and can contain duplicated elements
- The corresponding type in Java is `java.util.List`.

Collection Mapping – Bag Example

MusicCD.java

```
package collection.bag;

import java.util.Collection;
import java.util.ArrayList;
public class MusicCD {
    private Integer id;
    private String name;
    private Collection<String> songs = new ArrayList<String>();

    public MusicCD() {
    }

    // Generate Setter/Getter methods for id and name

    public Collection<String> getSongs() {
        return songs;
    }

    public void setSongs(Collection<String> songs) {
        this.songs = songs;
    }
}
```

Collection Mapping – Bag Example (Contd.).

Song.java

```
package collection.bag;

public class Song {

    private Integer id;
    private String info;

    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getInfo() {
        return info;
    }
    public void setInfo(String info) {
        this.info = info;
    }
}
```

Collection Mapping – Bag Example (Contd.).

MusicClient.java

```
package collection.bag;

import java.util.List;
import java.util.ArrayList;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class MusicClient {
    public static void main(String[] args){
        Session session=null;
        try{
            // Add HibenateUtil.java file to this project before you execute this code
            session=HibernateUtil.getSessionFactory().openSession();
            Transaction tx = session.beginTransaction();
            MusicCD a1 = new MusicCD();
            a1.setName("Devotional");
            session.saveOrUpdate(a1);
        }
    }
}
```

Collection Mapping – Bag Example (Contd.).

```
List<String> movies = new ArrayList<String>();  
movies.add("Jo Bhaje Hari Ko Sada");  
movies.add("Maa Durgaa Bhajan");  
movies.add("Aisi Laagi Lagan");  
movies.add("Hanuman Chalisa");
```

```
a1 = (MusicCD) session.load(MusicCD.class, a1.getId());  
a1.setSongs(movies);  
session.update(a1);
```

```
tx.commit();
```

```
}
```

```
catch(HibernateException e){
```

```
    e.printStackTrace();
```

```
    System.out.println("Error in adding");
```

```
}
```

```
finally {
```

```
    session.close();
```

```
}
```

```
}
```

```
}
```

Collection Mapping – Set Example (Contd.).

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
    <property name="hibernate.connection.driver_class">oracle.jdbc.OracleDriver</property>
    <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
    <property name="hibernate.connection.username">scott</property>
    <property name="hibernate.connection.password">tiger1</property>
    <property name="hibernate.show.sql">true</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="hibernate.current_session_context_class">thread</property>
    <mapping resource="MusicCD.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Collection Mapping – Set Example (Contd.).

MusicCD.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="collection.bag.MusicCD" table="MUSICCD" >
    <id name="id" type="int" unsaved-value="0">
      <column name="ID" precision="5" scale="0" />
      <generator class="increment" />
    </id>
    <property name="name" type="string">
      <column name="NAME" length="25" />
    </property>
    <bag name="songs" table="SONG">
      <key column="ID"/>
      <element type="string" column="INFO" length="30"/>
    </bag>
  </class>
</hibernate-mapping>
```


Association Mappings

Different kinds of relations exist between entities. Relations can be based on cardinalities or the directionality:

- Relations based on cardinality
 - One to One
 - One to Many
 - Many to One
 - Many to Many
- Relations based on directionality
 - Unidirectional
 - Bidirectional

One-to-Many Association

Example: A Writer can write many stories

Tables

```
create table writer(  
  id number(5) not null primary key,  
  name varchar(15)  
);
```

```
create table story(  
  storyid number(5) not null primary key,  
  info varchar(20),  
  id references writer  
);
```

foreign key **id** in Story is a primary key in Writer

One-to-Many Association (Contd.).

Example for one-to-many association: A Writer can write many stories

Writer.hbm.xml

```
<class name="onetomany.Writer" table="WRITER" schema="SCOTT">
  <id name="id" type="int">
    <column name="ID" precision="5" scale="0" />
    <generator class="assigned" />
  </id>
  <property name="name" type="string">
    <column name="NAME" length="15" />
  </property>
  <set name="stories" inverse="true">
    <key>
      <column name="ID" precision="5" scale="0" />
    </key>
    <one-to-many class="onetomany.Story" />
  </set>
</class>
```

One-to-Many Association (Contd.).

Story.hbm.xml

```
<class name="onetomany.Story" table="STORY" schema="SCOTT">
  <id name="storyid" type="int">
    <column name="STORYID" precision="5" scale="0" />
    <generator class="assigned" />
  </id>

  <many-to-one name="writer" class="onetomany.Writer">
    <column name="ID" precision="5" scale="0" />
  </many-to-one>

  <property name="info" type="string">
    <column name="INFO" length="20" />
  </property>
</class>
```

Inheritance Mapping

There are 3 different ways by which we can represent inheritance:

1. Table per concrete class

- **union-subclass** element for each subclass in mapping file
- We just should have a separate class element for each class, no special elements are required.

2. Table per subclass

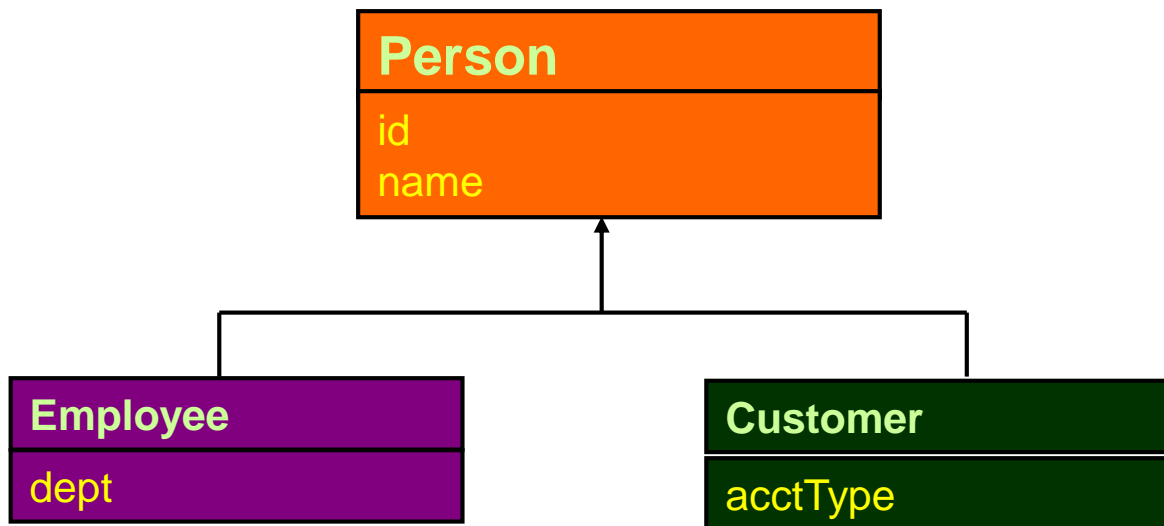
- **joined-subclass** element for each subclass in mapping file
- In this case the entire hierarchy is mapped to a single table, which includes all the attributes of all the classes in a single table as its columns. A discriminator is used to distinguish between different classes.

3. Table per class hierarchy

- **discriminator** element to identify object type
- **subclass** element for each subclass in mapping file
- Here also we have different tables for different classes, but we have only specific attributes of subclasses in their corresponding tables instead of the inherited properties as was the case in table per concrete class.

Inheritance Mapping - Example

Class Model:



Inheritance Mapping - Table per concrete class

- Assign each concrete class an isolated table that duplicates all the columns for inherited properties.

Person Table

Id	Name
1	aaa
2	bbb
3	kkk

Employee Table

Id	Name	Dept
1	aaa	BFSI
2	Bbb	ES

Customer Table

Id	Name	Acct_Type
3	kkk	Savings

- A separate union-subclass element for each subclass in the mapping file
- Limitation - Not efficient for polymorphic queries and associations since several tables need to be navigated for retrieving the objects.

Inheritance Mapping - Table per subclass

- Uses one table for each subclass and non-subclass.
- Only specific attributes of subclasses in their corresponding tables are used instead of the inherited properties as was the case in table per concrete class.
- It defines a foreign key in the table of subclass that references the table of its parent.

Person Table

Id	Name	Person_Type
1	aaa	EMP
2	bbb	EMP
3	kkk	CUST

Employee Table

Id	Dept
1	BFSI
2	ES

Customer Table

Id	Acct_Type
3	Savings

- This strategy has no limitation on not-null constraint but it is less efficient, since several tables need to be joined for retrieving a single object.
- For polymorphic queries and associations, more tables need to be joined.

Inheritance Mapping - Table per subclass

- **Table per subclass**

Here also we have different tables for different classes, but we have only specific attributes of subclasses in their corresponding tables instead of the inherited properties as was the case in table per concrete class.

This strategy uses one table for each subclass and non-subclass. It defines a foreign key in the table of subclass that references the table of its parent. You can imagine there's a one-to-one association from the subclass to its parent. In other words, a different table for each class in the hierarchy is used, but all these tables must share the same primary key. Hibernate will then use this primary key when it inserts new records into the database. It will also make use of this same primary key to perform JOIN operations when accessing the database.

Inheritance Mapping - Table per class hierarchy

- A single table is mapped to store all the properties of all the subclasses/non-subclasses within a class hierarchy.
- In addition, a special column called “**discriminator**” is used to distinguish the type of the object – **Person_Type**

Person Table

Id	Name	Person_Type	Dept	Acct_Type
1	aaa	EMP	BFSI	NULL
2	bbb	EMP	ES	NULL
3	kkk	CUST	NULL	Savings

- **Advantage** - Simple and efficient, especially for polymorphic queries and associations, since one table contains all the data and no table join is required.
- **Limitation** - All the properties in subclasses must not have not-null constraint.

Inheritance Mapping - Table per class hierarchy

In this strategy, we'll look at how to map our Person hierarchy.

- **Table per class hierarchy** - In this case the entire hierarchy is mapped to a single table, which includes all the attributes of all the classes in a single table as its columns. A discriminator is used to distinguish between different classes.

Here Person_Type is the discriminator column.

The main advantage of this strategy is simple and efficient, especially for polymorphic queries and associations, since one table contains all the data and no table join is required. However, this strategy has a fatal limitation that all the properties in subclasses must not have not-null constraint.

Table per class hierarchy - Example

- Steps for setting Web Application to map inheritance in Hibernate
 - Create single table
 - Create a web application project
 - Create HibernateUtil.java file
 - Create 3 separate POJOs
 - Create Mapping files for each POJO
 - Update mapping-resource in hibernate.configuration.xml file
 - Create a helper file
 - Create a jsp page to display saved instances

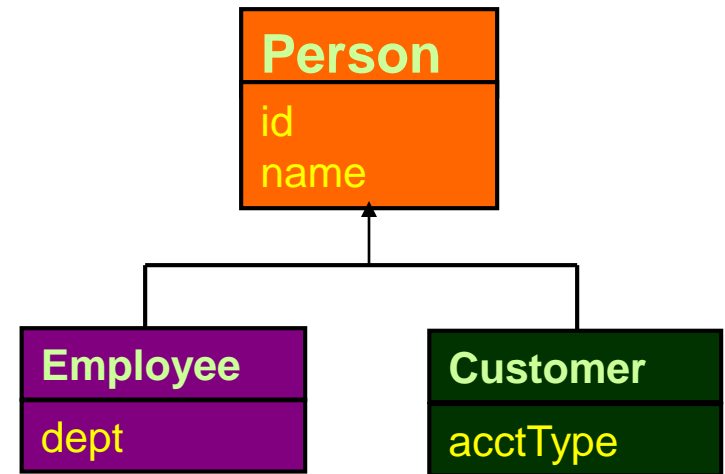
Table per class hierarchy – Example (Contd.).

- Create a single table:

Person class hierarchy

Person table

```
create table person(  
  id number(5) not null primary key,  
  name varchar(15),  
  person_type varchar(4),  
  dept varchar(10),  
  acct_type varchar(12)  
);
```



- Has all columns to store every attribute of the Person class hierarchy
- Has an additional column called “DISCRIMINATOR”

Table per class hierarchy – Example (Contd.).

The PERSON table possesses all the columns necessary to store every attribute of the Person class hierarchy. Values of the saved instances will then be saved in the table, leaving every unused column filled with NULL values.

The PERSON table contains an additional column called DISCRIMINATOR. Hibernate uses this column to automatically instantiate the appropriate class and populate it accordingly. This column is mapped using the <discriminator> XML element in the Person.hbm.xml mapping files.

Table per class hierarchy – Example (Contd.).

■ Create 3 separate POJOs

```
public class Person {  
    private int id;  
    private String name;  
  
    public Person() {  
    }  
  
    public int getId() {  
        return this.id;  
    }  
  
    public void setId(int id) {  
        this.id = id;  
    }  
  
    public String getName() {  
        return this.name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

```
public class Employee extends Person {  
    private String dept;  
  
    public String getDept() {  
        return this.dept;  
    }  
  
    public void setDept(String dept) {  
        this.dept = dept;  
    }  
}
```

```
public class Customer extends Person {  
    private String acctType;  
  
    public String getAcctType() {  
        return this.acctType;  
    }  
  
    public void setAcctType(String acctType) {  
        this.acctType = acctType;  
    }  
}
```

Table per class hierarchy – Example (Contd.).

- Create Mapping files for each POJO

Person.hbm.xml

```
<hibernate-mapping>
  <class name="Person" table="PERSON" schema="SCOTT">
    <id name="id" type="int" unsaved-value="0">
      <column name="ID" precision="5" scale="0" />
      <generator class="increment" />
    </id>
    <discriminator column="PERSON_TYPE" type="string"/>
    <property name="name" type="string">
      <column name="NAME" length="15" />
    </property>
  </class>
</hibernate-mapping>
```

- Observe that discriminator is not a property of any Java class!!
- It is simply a technical column shared between Hibernate and the database

Table per class hierarchy – Example (Contd.).

Employee.hbm.xml

```
<hibernate-mapping>
  <subclass name="Employee" extends="Person" discriminator-value="EMP">
    <property name="dept" type="string">
      <column name="DEPT" length="10" />
    </property>
  </subclass>
</hibernate-mapping>
```

Customer.hbm.xml

```
<hibernate-mapping>
  <subclass name="Customer" extends="Person" discriminator-value="CUST">
    <property name="acctType" type="string">
      <column name="ACCT_TYPE" length="12" />
    </property>
  </subclass>
</hibernate-mapping>
```

Table per class hierarchy – Example (Contd.).

Note that the `<subclass>` XML tag is used to map the two concrete classes (Employee and Customer). This tag requires the name attribute just as the class tag requires the discriminator-value attribute. Also use `<subclass>` element with *extends* and *discriminator-value* attributes.

Table per class hierarchy – Example (Contd.).

Client code for Table per class Example :

```
import org.hibernate.*;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class Client {
    public static void main(String[] args) {
        Session session=HibernateUtil.getSessionFactory().openSession();
        try {
            Transaction tx=session.beginTransaction();
            Employee e1=new Employee();
            e1.setName("Paritosh");
            e1.setDept("HR");
```

Contd..

Table per class hierarchy – Example (Contd.).

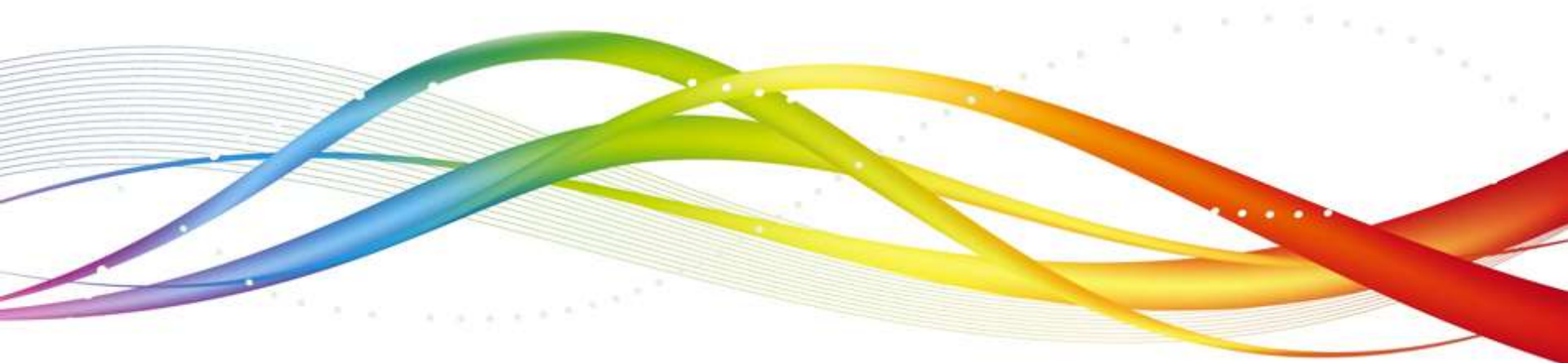
```
Customer c1=new Customer();
c1.setName("Jeevesh");
c1.setAcctType("savings");
session.save(e1);
session.save(c1);
tx.commit();
}
catch (HibernateException e){
    e.printStackTrace();
}
finally {
    session.close();
}
}
```

Summary

In this module you learnt about:

- Basic O/R mapping
- Collection mapping
- Association mapping
- Inheritance mapping

Demo Examples on Associations



Objectives

In this module, you will be able to :

- Develop an application based on One to One Association
- Develop an application based on One to Many Associations
- Develop an Application based on Many to Many Associations

Demo – One to One Association

This example involves two entity classes, Employee and Login. For every Employee, there exists one Login account, thereby establishing One to One Association.

We are going to use the following four Java classes :

- Employee.java – The entity class representing the employee
- Login.java – The entity class representing the Login details for every employee
- HibernateUtil.java – The utility class containing a method that returns a SessionFactory object
- Client.java – This class contains the main method through which we execute this code

As usual, we will have the hibernate.cfg.xml file, which will contain the configuration details.

Demo – One to One Association (Contd.).

Employee.java

```
package com.wipro;
import java.sql.Date;
import javax.persistence.GenerationType;
import javax.persistence.SequenceGenerator;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;
import javax.persistence.Table;
@Entity
@Table(name="EMPLOYEE121")
public class Employee {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO, generator="my_seq_gen")
    @SequenceGenerator(name="my_seq_gen", sequenceName="ENTITY_SEQ")
    @Column(name="employee_id")
    private Long employeeId;
```

Contd...

Demo – One to One Association (Contd.).

```
@Column(name="firstname", length=15)
```

```
private String firstname;
```

```
@Column(name="lastname", length=15)
```

```
private String lastname;
```

```
@Column(name="birth_date")
```

```
private Date birthDate;
```

```
@Column(name="cell_phone", length=15)
```

```
private String cellphone;
```

```
@OneToOne
```

```
@JoinColumn(name="LOGIN_ID")
```

```
private Login login;
```

```
public Employee() {
```

```
}
```

```
public Employee(String firstname, String lastname, String phone, String birthDate) {
```

```
    this.firstname = firstname;
```

```
    this.lastname = lastname;
```

```
    this.birthDate = Date.valueOf(birthDate);
```

```
    this.cellphone = phone;
```

```
}
```

```
//generate all the setter/getter methods
```

```
}
```

Demo – One to One Association (Contd.).

Login.java

```
package com.wipro;  
  
import javax.persistence.CascadeType;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
import javax.persistence.OneToOne;  
import javax.persistence.Table;
```

```
@Entity
```

```
@Table(name="LoginUserDetails")
```

```
public class Login {
```

```
    @Id
```

```
    @GeneratedValue
```

```
    @Column(name="LOGIN_ID")
```

```
    private Long loginId;
```

```
    @Column(name="password", length=20)
```

```
    private String password;
```

Contd...

Demo – One to One Association (Contd.).

```
@Column(name="type", length=15)
```

```
private String usertype;
```

```
@OneToOne(cascade=CascadeType.ALL, mappedBy="login")
```

```
private Employee employee;
```

```
public Long getLoginId() {
```

```
    return loginId;
```

```
}
```

```
public void setLoginId(Long loginId) {
```

```
    this.loginId = loginId;
```

```
}
```

```
public Employee getEmployee() {
```

```
    return employee;
```

```
}
```

```
public void setEmployee(Employee employee) {
```

```
    this.employee = employee;
```

```
}
```

```
//Generate all the remaining setter/getter methods
```

```
}
```

Demo – One to One Association (Contd.).

HibernateUtil.java

```
package com.wipro;
import org.hibernate.cfg.Configuration;
import org.hibernate.SessionFactory;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    static {
        try {
            sessionFactory = new Configuration().configure().buildSessionFactory();
        }
        catch(Exception ex) {
            System.out.println("Initial SEssionFactory creation failed :"+ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Demo – One to One Association (Contd.).

Client.java

```
package com.wipro;

import org.hibernate.Session;
import org.hibernate.SessionFactory;

public class Client {
    public static void main(String[] args) {
        try {
            SessionFactory sf = HibernateUtil.getSessionFactory();
            Session session = sf.openSession();
            session.beginTransaction();

            Login login = new Login();
            login.setPassword("123456");
            login.setUserType("admin");
```

Contd..

Demo – One to One Association (Contd.).

```
Employee emp1 = new Employee("Harish", "Rao", "9876543210", "1999-09-09");

emp1.setLogin(login);
login.setEmployee(emp1);

session.save(login);
session.getTransaction().commit();
session.close();
}
catch(Exception e) {
    System.out.println(e);
}
}
}
```

Demo – One to One Association (Contd.).

hibernate.cfg.xml

<hibernate-configuration>

.....

.....

<mapping class="com.wipro.Login"/>

<mapping class="com.wipro.Employee"/>

.....

</hibernate-configuration>

Code for creating Sequence (SQL):

**CREATE SEQUENCE ENTITY_SEQ MINVALUE 1 MAXVALUE 99999 INCREMENT BY 1 START
WITH 1001 NOCACHE NOCYCLE ;**

Demo – One to Many Association

In this example, we are dealing with two entity classes, Employee and Department. Every Department can have as many employees as possible, thereby establishing One to Many Association.

We are going to use the following four Java classes :

- Department.java – The entity class that represents the Department
- Employee.java – The entity class that represents the employees in a department
- HibernateUtil.java – The utility class containing a method that returns a SessionFactory object
- Client.java – This class contains the main method through which we execute this code

As usual, we will have the hibernate.cfg.xml file, which will contain the configuration details.

Demo – One to Many Association (Contd.).

Department.java

```
package com.wipro;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
@Entity
@Table(name="DEPARTMENT12M")
public class Department {

    @Id
    @GeneratedValue
    @Column(name="DEPARTMENT_ID")
    private Long departmentId;

    @Column(name="DEPT_NAME", length=15)
    private String departmentName;
```

Contd..

Demo – One to Many Association (Contd.).

```
@OneToMany(cascade=CascadeType.ALL, mappedBy="department")
```

```
private Set<Employee> employees;
```

```
public Long getDepartmentId() {  
    return departmentId;  
}
```

```
public void setDepartmentId(Long departmentId) {  
    this.departmentId = departmentId;  
}
```

```
public String getDepartmentName() {  
    return departmentName;  
}
```

```
public void setDepartmentName(String departmentName) {  
    this.departmentName = departmentName;  
}
```

```
public Set<Employee> getEmployees() {  
    return employees;  
}
```

```
public void setEmployees(Set<Employee> employees) {  
    this.employees = employees;  
}
```

```
}
```

Demo – One to Many Association (Contd.).

Employee.java

```
package com.wipro;  
import java.sql.Date;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;
```

@Entity

@Table(name="EMPLOYEE12M")

public class Employee {

@Id

@GeneratedValue(strategy=GenerationType.AUTO, generator="my_seq_gen")

@SequenceGenerator(name="my_seq_gen", sequenceName="ENTITY_SEQ")

@Column(name="employee_id")

private Long employeeId;

@Column(name="firstname", length=15)

private String firstname;

Contd..

Demo – One to Many Association (Contd.).

```
@Column(name="lastname", length=15)
```

```
private String lastname;
```

```
@Column(name="join_date")
```

```
private Date joinDate;
```

```
@Column(name="cell_phone", length=15)
```

```
private String cellphone;
```

```
@ManyToOne
```

```
@JoinColumn(name="department_id")
```

```
private Department department;
```

```
public Employee() {
```

```
}
```

```
public Employee(String firstname, String lastname, String phone) {
```

```
    this.firstname = firstname;
```

```
    this.lastname = lastname;
```

```
    this.joinDate = new Date(System.currentTimeMillis());
```

```
    this.cellphone = phone;
```

```
}
```

```
}
```

Demo – One to Many Association (Contd.).

Client.java

```
package com.wipro;
```

```
import org.hibernate.Session;
```

```
import org.hibernate.SessionFactory;
```

```
import java.util.*;
```

```
public class Client {
```

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

```
        //Hibernate.Util class remains same as given in the previous one to one example
```

```
        SessionFactory sf = HibernateUtil.getSessionFactory();
```

```
        Session session = sf.openSession();
```

```
        session.beginTransaction();
```

```
        Department department = new Department();
```

```
        department.setDepartmentName("Research");
```

Contd..

Demo – One to Many Association (Contd.).

```
Employee emp1 = new Employee("Shreyas", "Kansal", "555");  
Employee emp2 = new Employee("Ayush", "Pathak", "666");
```

```
emp1.setDepartment(department);  
emp2.setDepartment(department);
```

```
Set<Employee> set = new HashSet<Employee>();
```

```
set.add(emp1);  
set.add(emp2);
```

```
department.setEmployees(set);  
session.save(department);
```

```
session.getTransaction().commit();  
session.close();
```

```
}
```

```
}
```

// Please add hibernate.cfg.xml file, before you execute this code

Demo – Many to Many Association

In this example, we are dealing again with two entity classes, Employee and Certificate. An Employee can have one or more Certificates and A Certificate can be possessed by one or more employees, thereby establishing Many to Many Association.

We are going to use the following four Java classes :

- Employee.java – The entity class that represents an Employee
- Certificate.java – The entity class that represents the Certificates
- HibernateUtil.java – The utility class containing a method that returns a SessionFactory object
- Employee_Certificate_Client.java – This class contains the main method through which we execute this code

As usual, we will have the hibernate.cfg.xml file, which will contain the configuration details. The participants are expected to add this file to the project, before executing this code.

Demo – Many to Many Association (Contd.).

Employee.java

```
package com.wipro;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.Table;
import javax.persistence.ManyToMany;

@Entity
@Table(name="EMPLOYEEM2M")
public class Employee implements java.io.Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name="id")
    private int id;
```

Demo – Many to Many Association (Contd.).

```
@Column(name="first_name", length=15)
private String firstName;

@Column(name="last_name", length=15)
private String lastName;

@ManyToMany(
    targetEntity=com.wipro.Certificate.class,
    cascade={CascadeType.ALL}
)
@JoinTable(name = "EMP_CERT",
    joinColumns = @JoinColumn(name = "EMPLOYEE_ID", referencedColumnName="ID"),
    inverseJoinColumns = @JoinColumn(name = "CERTIFICATE_ID",
        referencedColumnName="ID"))
private Set<Certificate> certificates;

public Employee() {
}

public Employee(String fname, String lname) {
    this.firstName = fname;
    this.lastName = lname;
}

//Generate all Setter/Getter methods
```

Demo – Many to Many Association (Contd.).

Certificate.java

```
package com.wipro;

import javax.persistence.CascadeType;
import javax.persistence.Table;
import javax.persistence.Id;
import javax.persistence.Entity;
import javax.persistence.Column;
import javax.persistence.ManyToMany;
import java.util.Set;

@Entity
@Table(name="CERTIFICATEM2M")
public class Certificate implements java.io.Serializable {

    @Id
    @Column(name = "ID")
    private int certid;

    @Column(name="certificate_name", length=15)
    private String name;
```

Contd..

Demo – Many to Many Association (Contd.).

```
@ManyToMany(mappedBy = "certificates", cascade = {CascadeType.ALL},  
targetEntity=com.wipro.Employee.class)  
private Set<Employee> employees;
```

```
public Certificate() {}  
public Certificate(String name) {  
    this.name = name;  
}  
public Certificate(String name, int certid) {  
    this.certid = certid;  
    this.name = name;  
}
```

Demo – Many to Many Association (Contd.).

HibernateUtil.java

```
package com.wipro;
import org.hibernate.cfg.Configuration;
import org.hibernate.SessionFactory;
public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    static {
        try {
            sessionFactory = new Configuration().configure().buildSessionFactory();
        }
        catch(Exception ex) {
            System.out.println("Initial SessionFactory creation failed :"+ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Demo – Many to Many Association (Contd.).

Employee Certificate Client.java

```
package com.wipro;
```

```
import java.util.HashSet;
```

```
import java.util.Iterator;
```

```
import java.util.List;
```

```
import java.util.Set;
```

```
import org.hibernate.HibernateException;
```

```
import org.hibernate.Session;
```

```
import org.hibernate.SessionFactory;
```

```
import org.hibernate.Transaction;
```

```
public class Employee_Certificate_Client {
```

```
    private static SessionFactory sf;
```

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

```
        try{
```

```
            sf = HibernateUtil.getSessionFactory();
```

```
        }
```

```
        catch (Throwable ex) {
```

```
            System.err.println("Failed to create sessionFactory object." + ex);
```

```
            throw new ExceptionInInitializerError(ex);
```

```
        }
```

Demo – Many to Many Association (Contd.).

```
Employee_Certificate_Client ecc = new Employee_Certificate_Client();
HashSet<Certificate> certificates = new HashSet<Certificate>();
certificates.add(new Certificate("MTech", 1001));
certificates.add(new Certificate("PhD", 1002));
ecc.addEmployeeDetails("Rikesh", "Singh", certificates);
ecc.addEmployeeDetails("Shivarshi", "Bajpai", certificates);
System.out.println("Employee details added");
ecc.retrieveEmployeesData();
}
```

```
public Integer addEmployeeDetails(String fname, String lname, Set cert){
    Session session = sf.openSession();
    Transaction tx = null;
    Integer employeeID = null;
    try{
        tx = session.beginTransaction();
        Employee employee = new Employee(fname, lname);
        employee.setCertificates(cert);
        employeeID = (Integer) session.save(employee);
        tx.commit();
    }
```

Demo – Many to Many Association (Contd.).

```
        catch (HibernateException e) {
            if (tx!=null) tx.rollback();
            e.printStackTrace();
        }
        finally {
            session.close();
        }
        return employeeID;
    }

    public void retrieveEmployeesData( ){
        Session session = sf.openSession();
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            List<Employee> employees = session.createQuery("FROM Employee").list();
            for (Iterator<Employee> it1 = employees.iterator(); it1.hasNext(); ) {
                Employee employee = (Employee) it1.next();
                System.out.print("First Name: " + employee.getFirstName());
                System.out.print(" Last Name: " + employee.getLastName());
                Set<Certificate> certificates = employee.getCertificates();
            }
        }
    }
```


Demo – Many to Many Association (Contd.).

```
        int flag=1;
        for (Iterator<Certificate> it2 = certificates.iterator(); it2.hasNext();) {
            Certificate cName = (Certificate) it2.next();
            if(flag==1) {
                System.out.print(" Certificates: " );
                flag++;
            }
            System.out.print(cName.getName() + " ");
        }
        System.out.println();
        System.out.println("-----");
    }
    tx.commit();
}
catch (HibernateException e) {
    if (tx!=null) tx.rollback();
    e.printStackTrace();
}
finally {
    session.close();
}
}
}
```

Summary

In this module, you were able to :

- Develop an application based on One to One Association
- Develop an application based on One to Many Associations
- Develop an Application based on Many to Many Associations



Thank You

