Hibernate - III

Working with Objects

Agenda

1 **Working with Objects**

2 **Example - CRUD Operations with hibernate**

Working with Objects

Objectives

In this module, you will be able to :

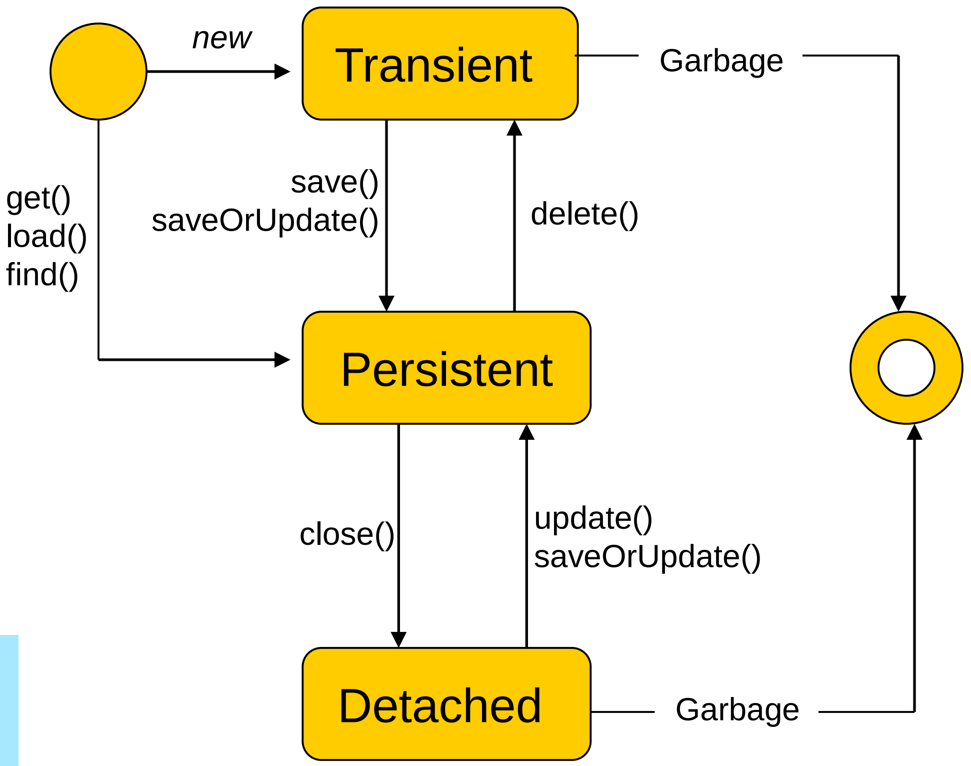
- Understand Hibernate Object States
- Make Objects persistent
- Retrieve a persistent Object
- Modify Objects
- Delete Objects

Hibernate Object States

- An object is transient when instantiated with new operator
- Not associated with Hibernate session
- No persistent representation in database & no identifier value yet assigned

- A persistent object has a database identity & an identifier value
- Associated with Hibernate session – saved or loaded
- Participate in a transaction

- A detached instance is an object that has been persistent, but its Session has been closed
- The state of detached objects is not guaranteed to be synchronized with database state



States of an object and transitions in a Hibernate Application

Hibernate Object States (Contd.).

Use Hibernate Session to make an object persistent . Hibernate will take care of the SQL statements that need to be executed for this transition. An object moves from a transient state to persistent state, after a save method is called. The save method not only saves the object but also any other object it refers to, so that the relationships does not get affected.

Hibernate will detect any change made to an object in persistent state and synchronize the state with the database when the unit of work completes. A persistent instance has a primary key value, set as its database identifier. Persistent instances participate in transactions—their state is synchronized with the database at the end of the transaction.

When a transaction commits, state held in memory is propagated to the database by the execution of SQL INSERT, UPDATE, and DELETE statements. This procedure might also occur at other times.

For example, Hibernate might synchronize with the database before execution of a query. This ensures that queries will be aware of changes made earlier during the transaction.

Hibernate Object States (Contd.).

A detached instance can be reattached to a new Session at a later point in time, making it (and all the modifications) persistent again. This feature enables a programming model for long running units of work that require user think-time. These are called *application transactions*, i.e. a unit of work from the point of view of the user.

Transient instances will be destroyed by garbage collector, if application doesn't hold a reference anymore.

Transient- We create a new instance of a persistence class and we have not called the save method to insert the record in the database. The identifier value will not be provided at this point. There is no record for this object in the database. All these points summarize the transient state of the object.

Persistent - An object moves from a transient state to persistent state after a save method is called. The save method not only saves the object but also any other object it refers to, so that the relationships does not get affected. A persistent object will have a database identity and can be used in a transaction.

Detached - An object, which was in persistent state and is released from that state due to a close statement is said to be detached. The object references are still valid and you can update the values in the object and associate the object to a session.

Making Objects Persistent

- Hibernate considers newly instantiated instances of a persistent class as *transient*
- A transient instance can be made *persistent* by associating it with a session and calling save() method
- Hibernate provides save() method to persist object to the database
  - public Object save(Object obj);
- Example:

```
// Create a Student object and save it
Student s1 = new Student();
s1.setName(“Asma");
session.save(s1);
```

Make a transient object persistent

Making Objects Persistent (Contd.).

The save method is used to make data persistent in the database. There are overloaded versions of save() method. This save method used in the example just accepts the object reference. We should be careful to provide unique values for this identifier.

The save method also checks whether the object is saved for the first time or the user has called the save method twice. Save method can detect this by checking the identifier value in the object. If the object is already saved, it will have an identifier generated, else there will not be any identifier. Of course we would have opened a session and started a new database transaction.

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

//Then create a Student object and save it

tx.commit();

session.close();
```

Making Objects Persistent (Contd.).

A call to save() makes the transient instance of Student persistent. It is now associated with the current Session. However, SQL INSERT has not yet been executed. The Hibernate Session never executes any SQL statement until absolutely necessary. The changes made to persistent objects have to be synchronized with the database at some point. This happens when we commit() the Hibernate Transaction in which case, Hibernate obtains a JDBC connection and issues a single SQL INSERT statement. Finally, the Session is closed and the JDBC connection is released.

The object can be modified after calling save(), and the changes will be propagated to the database as an SQL UPDATE.

Everything between session.beginTransaction() and tx.commit() occurs in one database transaction.

Retrieving a persistent object

- You can query the database using Session to retrieve existing persistent objects

- Two special methods are provided for the simplest kind of query: **retrieval by identifier**

  – The get() method
  – The load() method

Retrieving a persistent object (Contd.).

The load() methods of Session gives you a way to retrieve a persistent instance, if you already know its identifier. The load() method takes a class object and will load the state into a newly instantiated instance of that class, in persistent state. You also mention the database identity (id) of the object which has to be loaded from the database.

There are overloaded methods of load() available to load the objects from the database.

Retrieving a persistent object – load()

- The load() method – takes a class object and loads the state into a newly instantiated instance of that class, in persistent state

- Example:

```
//loading Student whose id is already generated
Student s = (Student) session.load(Student.class, s1.getId());
String name = s.getName();
System.out.println("name: " + name);
```

Load an object – if matching row exists

- It throws an unrecoverable exception such as ObjectNotFoundException if there is no database matching row

- No database hit occurs with load()

Retrieving a persistent object – load()

Note that load() will throw an unrecoverable exception if there is no matching database row. If the class is mapped with a proxy, load() just returns an uninitialized proxy and does not actually hit the database until you invoke a method of the proxy.

This behaviour is very useful if you wish to create an association to an object without actually loading it from the database. More important, the load() method may return a *proxy*, a placeholder, without hitting the database.

A consequence of this is that you may get an ObjectNotFoundException later, as soon as you try to access the returned placeholder and force its initialization (this is also called *lazy loading*).

Retrieving a persistent object – get()

- The get() method – Use this method if you are not sure that a matching row exists
- Example:

```
//loading Student object whose id is already generated
Student student = (Student) session.get(Student.class, s1.getId());
System.out.println("Student name = " + student.getName());
```

Load an object – if unsure about matching row

- It hits the database immediately and returns null if there is no matching row

Difference between get() and load()

One of the differences between get() and load() is how they indicate that the instance could not be found. If no row with the given identifier value exists in the database, get() returns null. The load() method throws an ObjectNotFoundException.

It’s one’s choice what error-handling is preferred.

Modifying Objects

Hibernate methods for modifying the database objects:

- public void update(Object object);

```
Session.update(continent);
```

Update an object

- public void saveOrUpdate(Object object);

```
Session.saveOrUpdate(continent);
```

Update or save an object – if you are not sure about the state

- public void flush();

```
Session.flush( );
```

Synchronize database with persistence context. Occurs automatically upon transaction.commit( )

Modifying Objects

We will load the objects for manipulation and after the manipulation of the objects based on the business logic, the object state has to be updated in the database.

The methods are self explanatory. But data may or may not be persisted when the methods saveOrUpdate or update methods are called. These operations are transparent to the user. This is what is known as t*ransparent transaction-level write-behind.* The transaction ends when the methods like close or flush are called on the session object or commit on the transaction object.

The flush() call forces persistent objects held in memory to be synchronized to the database. Sessions don't immediately write to the database when an object is saved. Instead, the Session queues a number of database writes to maximize performance.

Deleting Objects

Hibernate methods for deleting Objects:

- public void delete(Object object);

```
Session.delete(student);
```

Delete an object – make it transient again

Summary

In this module, you were able to :

- Understand Hibernate Object States
- Make Objects persistent
- Retrieve a persistent Object
- Modify Objects
- Delete Objects

Example – CRUD operations with hibernate

Objectives

In this module, you will be able to :

– Develop an application that implements CRUD operations through hibernate

CRUD operations using Hibernate

We will now go through an example, where we create an application called SchoolApp, that inserts, updates, deletes and retrieves details of candidates belonging to a School.

This application contains three Java classes, one of which is an entity class called Candidate, objects of which will be persisted in the database.

There are two more classes, SchoolClient and SchoolOperations, details of which are given in the next page.

CRUD operations using Hibernate (Contd.).

SchoolClient class contains the main method, which provides a menu for the user to perform CRUD operations.

SchoolOperations class contains five methods : insert, update, delete, selectOne and selectAll, which contain all the necessary code for performing database CRUD operations.

**** The participants are required to peruse and execute this example. The objective is to understand how to perform basic CRUD operations using hibernate.*

CRUD operations using Hibernate (Contd.).

Candidate.java

```
package school;
import java.sql.Date;

public class Candidate {
        private int rollno;
        private String name;
        private Date jdt;
        private String std;
        private float fees;

       // generate the setter and getter methods
}
```

CRUD operations using Hibernate (Contd.).

SchoolOperations.java

```
package school;

import java.util.Iterator;
import java.util.List;
import java.util.Scanner;

import java.sql.Date;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import school.Candidate;
```

Contd..

CRUD operations using Hibernate (Contd.).

```
public class SchoolOperations {
    public void insert(){
        Scanner x = new Scanner(System.in);
        System.out.println("Enter Roll Number : ");
        int rollno = x.nextInt();
        System.out.println("Enter Candidate Name : ");
        String name = x.next();
        System.out.println("Enter Joining Date (yyyy-mm-dd format only) : ");
        String jdate = x.next();
        Date jdt = Date.valueOf(jdate);
        System.out.println("Enter Standard : ");
        String std = x.next();
        System.out.println("Enter Fees : ");
        float fees = x.nextFloat();
        Candidate i=new Candidate();
        i.setRollno(rollno);  i.setName(name);
        i.setJdt(jdt);  i.setStd(std);  i.setFees(fees);
        SessionFactory sf=new Configuration().configure().buildSessionFactory();
        Session sess=sf.openSession();
        sess.beginTransaction();
        sess.save(i);
        sess.getTransaction().commit();
    }
```

Contd..

CRUD operations using Hibernate (Contd.).

```
public int update() {
        Scanner x = new Scanner(System.in);
        System.out.println("Enter Roll Number : ");
        int rollno = x.nextInt();

        SessionFactory sf=new Configuration().configure().buildSessionFactory();
        Session session=sf.openSession();

        session.beginTransaction();
        Candidate i = (Candidate)session.get(Candidate.class, rollno);

        System.out.println("The Roll no is : "+i.getRollno());
        System.out.println("The candidate name is : "+i.getName());
        System.out.println("The Joining date is : "+i.getJdt());
        System.out.println("Standard : "+i.getStd());
        System.out.println("The Fees : "+i.getFees());

        System.out.println("Enter updatable Fees Amount: ");
        float fees = x.nextFloat();
```

Contd..

CRUD operations using Hibernate (Contd.).

```
        String hql="update Candidate set fees=:newfees where rollno = :varrollno";

        Query query=session.createQuery(hql);
        query.setFloat("newfees", fees);
        query.setInteger("varrollno", rollno);
        int q=query.executeUpdate();

        if (q==1)
                System.out.println("Record is Updated");
        else
                System.out.println("Record not found");

        session.getTransaction().commit();
        return q;
}
```

Contd..

CRUD operations using Hibernate (Contd.).

```
public int delete(){
        SessionFactory sf=new Configuration().configure().buildSessionFactory();
        Session session=sf.openSession();
        Transaction tx=session.beginTransaction();

        Scanner x = new Scanner(System.in);
        System.out.println("Enter Roll No. : ");
        int rollno = x.nextInt();

        String hql="delete from Candidate where rollno="+rollno;
        Query query=session.createQuery(hql);
        int i=query.executeUpdate();
        if (i==1)
        System.out.println("Record is deleted");
        else
        System.out.println("Record not found");
        tx.commit();
        session.close();
        return i;
}
```

Contd..

CRUD operations using Hibernate (Contd.).

```
public List<Candidate> selectOne() {
        SessionFactory sf=new Configuration().configure().buildSessionFactory();
        Session session=sf.openSession();
        Scanner x = new Scanner(System.in);
        System.out.println("Enter Roll Number : ");
        int rollno = x.nextInt();
        Query query=session.createQuery("from Candidate where rollno = :rno");
        query.setInteger("rno", rollno);
        List<Candidate> l=query.list();
        for (Iterator iterator = l.iterator(); iterator.hasNext();) {
        Candidate cd = (Candidate) iterator.next();
        System.out.println("-----------------------------");
        System.out.println("RollNo :"+cd.getRollno());
        System.out.println("Candidate Name :"+cd.getName());
        System.out.println("Joining Date :"+cd.getJdt());
        System.out.println("Standard :"+cd.getStd());
        System.out.println("Fees :"+cd.getFees());
        System.out.println("*******************************");
        }
        return l;
}
```

Contd..

CRUD operations using Hibernate (Contd.).

```
        public List<Candidate> selectAll() {
                SessionFactory sf=new Configuration().configure().buildSessionFactory();
                Session session=sf.openSession();

                Query query=session.createQuery("from Candidate");
                List<Candidate> c=query.list();

                for (Iterator iterator = c.iterator(); iterator.hasNext();) {
                        Candidate cd = (Candidate) iterator.next();
                        System.out.println("------------------------------");
                        System.out.println("Roll No :"+cd.getRollno());
                        System.out.println("Candidate Name :"+cd.getName());
                        System.out.println("Joining Date :"+cd.getJdt());
                        System.out.println("Standard :"+cd.getStd());
                        System.out.println("Fees :"+cd.getFees());
                        System.out.println("******************************");
                        }
                return c;
        }
}
```

CRUD operations using Hibernate (Contd.).

<u>SchoolClient.java</u>

```
package school;

import java.util.Scanner;
import school.SchoolOperations;

public class SchoolClient {
        public static void main(String[] args) {
                boolean b = true;
                int i=0;
                Scanner x = null;
                SchoolOperations d;
                do {
                        System.out.println("1. Add Record");
                        System.out.println("2. Modify Record");
                        System.out.println("3. Delete Records");
                        System.out.println("4. Display One Record");
                        System.out.println("5. Display All Records");
                        System.out.println("Specify what you want to do..!");
                        x = new Scanner(System.in);
                        d = new SchoolOperations();
                        i = x.nextInt();
```

Contd..

CRUD operations using Hibernate (Contd.).

```
            switch(i) {
                case 1:d.insert();
                break;
                case 2:d.update();
                break;
                case 3:d.delete();
                break;
                case 4:d.selectOne();
                break;
                case 5:d.selectAll();
                break;
                case 6:b=false;
                break;
                default:System.out.println("Invalid Selection : Enter 1 or 2 or 3 or 4 or 5 or 6 for
                exit");
                b=false;
            }
        } while(b);
        System.out.println("Exiting Application");
    }
}
```

CRUD operations using Hibernate (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>
    <!-- Oracle dialect -->
     <property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
    <!-- Database connection settings -->
<property name ="hibernate.connection.driver_class">oracle.jdbc.driver.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>
    <!-- Echo all executed SQL to stdout -->
     <property name="hibernate.show_sql">true</property>
     <property name="hibernate.hbm2ddl.auto">update</property>
    <!-- Enable Hibernate's automatic session context management -->
     <property name="hibernate.current_session_context_class">thread</property>
     <mapping resource="Candidate.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

CRUD operations using Hibernate (Contd.).

Candidate.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="school.Candidate" table="school">
    <id name="rollno" column="Rollno" type="int">
         <generator class="assigned"/>
      </id>
      <property name="name" column="name" type="string" length="25" />
      <property name="jdt" column="joindate" type="java.sql.Date" />
      <property name="std" column="standard" type="string" length="4" />
      <property name="fees" column="fees" type="float" />
   </class>
</hibernate-mapping>
```

Summary

In this module, you were able to :

- Develop an application that implements CRUD operations through hibernate

Thank You