Hibernate 3.2

Hibernate with MySQL

Hibernate Feature

1) Hibernate system is responsible for taking the connections, creating statements and releasing the resources.
   Example: Customer cust=new Customer("….");
   session.save(cust);

2) Hibernate system is responsible for generating SQL queries required.

3) Hibernate system is responsible for generating the value required for primary key columns. Hibernate provides many built-In primary key generation algorithm and support to implement your own custom primary key generation algorithms.

4) Hibernate support various mapping styles:
   a) Simple Mapping
   b) Collection Mapping
   c) Inheritance Mapping
      1- Table Per Sub Class Mapping
      2- Table Per Class Mapping
      3- Table Per Concrete Class Mapping

   d) Association Mapping
      1- One-to-One Mapping
      2- One-to-Many Mapping
      3- Many-to-Many Mapping

   e) Other Mapping

5) Hibernate Supports two way to manage connections-
   a) DriverManager Connections
   b) DataSource Connections(*)

6) Hibernate supports two ways to manage Transactions-
   a) JDBC Txs (Outside the EJB or Spring Container)
   b) JTA Txs (Inside the EJB or Spring Container)

7) Hibernate has in-built support for Batch updates.

8) Hibernate provides various caching mechanisms.

9) Hibernate provides various Object Oriented Query Language(OOQL)
   a) HQL (*)
   b) QBC
   c) QBE
   d) Native SQL
   e) Named SQL

10) Hibernate systems uses many persistent best practices and forces the developer to use them for better performance.

Hibernate Mapping Types

Simple Mapping

When you map your persistence class fields with simple databases like String, Primitives, Wrappers, Date etc. with the corresponding table column then it is called as Simple Mapping. Refers ALab1

ALab1

Table Required

```
drop database db1;
create database db1;
use db1;

create table customers(
	cid int primary key auto_increment,
	cname char(25),
	email char(30),
	phone long,
	city char(15)
);

select * from customers;
```

Client Class

Lab01Client.java

```
package hibernate;

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

public class Lab01Client {
	public static void main(String[] args) {
		Transaction tx=null;
```

```
            try{
                  SessionFactory sf=AHibernateUtil.getSessionFactory();
                  Session session=sf.openSession();
                  tx=session.beginTransaction();

                  Customer cust=new
Customer("Vishnu","vishnu@hotmail.com",93892999, "Varanasi");
                  session.save(cust);

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

            }catch(Exception e){
                  if(tx!=null){
                        tx.rollback();
                  }
            }
      }
}
```

Persistence Class

Customer.java

```
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="customers")
public class Customer {

      @Id
      @Column(name="cid")
      private int cid;

      @Column(name="cname")
      private String cname;

      @Column(name="email")
      private String email;

      @Column(name="phone")
      private long phone;

      @Column(name="city")
      private String city;

      public Customer() {}

      public Customer(String cname, String email, long phone, String city) {
            super();
```

```
		this.cname = cname;
		this.email = email;
		this.phone = phone;
		this.city = city;
	}
	// Getters and Setters Method
}
```

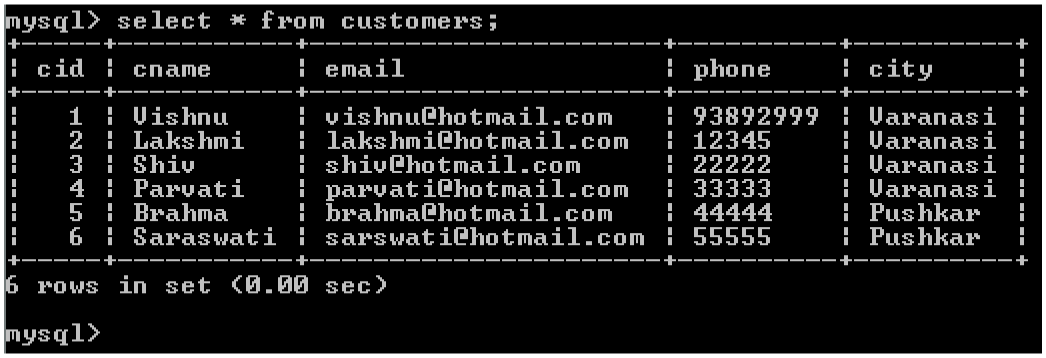
AHibernateUtil.java

```
package hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class AHibernateUtil {
	static SessionFactory sessionFactory;

	static{
		AnnotationConfiguration cfg=new AnnotationConfiguration();
		
	cfg=(AnnotationConfiguration)cfg.configure("resource/hibernate.cfg.xml");
		sessionFactory=cfg.buildSessionFactory();
	}

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

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="connection.driver_class">com.mysql.jdbc.Driver</property>
		<property name="connection.url">jdbc:mysql://localhost/db1</property>
		<property name="connection.username">root</property>
		<property name="connection.password">Westpac1</property>
		<property name="dialect">org.hibernate.dialect.MySQLDialect</property>
		<property name="show_sql">true</property>

		<mapping class="hibernate.Customer"/>
	</session-factory>
</hibernate-configuration>
```

Output:


```
mysql> select * from customers;
+-----+-----------+----------------------+----------+----------+
| cid | cname     | email                | phone    | city     |
+-----+-----------+----------------------+----------+----------+
|   1 | Vishnu    | vishnu@hotmail.com   | 93892999 | Varanasi |
|   2 | Lakshmi   | lakshmi@hotmail.com  | 12345    | Varanasi |
|   3 | Shiv      | shiv@hotmail.com     | 22222    | Varanasi |
|   4 | Parvati   | parvati@hotmail.com  | 33333    | Varanasi |
|   5 | Brahma    | brahma@hotmail.com   | 44444    | Pushkar  |
|   6 | Saraswati | sarswati@hotmail.com | 55555    | Pushkar  |
+-----+-----------+----------------------+----------+----------+
6 rows in set (0.00 sec)

mysql>
```

Collection Mapping

When you map your persistence class with collection data types like Array, List, Set, Map with the corresponding table columns then it is called as Collection Mapping.

Example with Hibernate Annotation (ALab2)

File Required:

a) Student.java
b) Tables (auto creation)
c) ALab2Client.java
d) AHibernateUtil.java
e) Hibernate.cfg.xml

ALab2

Table Required

```
drop database db2;
create database db2;
use db2;

select * from students;
```

Client Class

Lab02Client.java

```
package hibernate;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

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

public class Lab02Client {
  public static void main(String[] args) {

             Transaction tx=null;

             try{
                     SessionFactory sf=AHibernateUtil.getSessionFactory();
                     Session session=sf.openSession();
                     tx=session.beginTransaction();

                     String courses[]={"Java","JDBC","JSP"};

                     List<String> emails=new ArrayList<String>();
                          emails.add("ankur@carecentrix.com");
                          emails.add("basheer@carecentrix.com");
                          emails.add("rajesh@carecentrix.com");
                          emails.add("veera@carecentrix.com");
                          emails.add("sthit@carecentrix.com");

                     List<Integer> marks=new ArrayList<Integer>();
                          marks.add(new Integer(100));
                          marks.add(new Integer(500));
                          marks.add(new Integer(600));
                          marks.add(new Integer(700));
                          marks.add(new Integer(200));

                     Set<Long> phones=new HashSet<Long>();
                          phones.add(new Long(99999999));
                          phones.add(new Long(88888888));
                          phones.add(new Long(77777777));
                          phones.add(new Long(66666666));
                          phones.add(new Long(55555555));

                     Map<String,Long> refs=new HashMap<String,Long>();
                          refs.put("Ram",new Long(77777777));
                          refs.put("Shyam",new Long(88888888));
                          refs.put("Mohan",new Long(33333333));
                          refs.put("Madhav", new Long(56565656));

                     Student student=new Student("Vishnu","30-09-1990","MCA", courses,
emails, marks, phones, refs);

                     session.save(student);
                     tx.commit();
                     session.close();

             }catch(Exception e){
                     if(tx!=null){
                          tx.rollback();
                     }
             }
  }
}
```

Persistence Class

Student.java

```
package hibernate;

import java.util.List;
import java.util.Map;
import java.util.Set;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.Table;

import org.hibernate.annotations.CollectionOfElements;
import org.hibernate.annotations.IndexColumn;

@Entity
@Table(name="students")
public class Student {
	@Id
	@Column(name="sid")
	int sid; // P.K.

	@Column(name="sname")
	String sname;

	@Column(name="dob")
	String dob;

	@Column(name="qualification")
	String qualification;

	@CollectionOfElements
	@JoinTable(name="courses", joinColumns=@JoinColumn(name="sid"))
	@IndexColumn(name="idx")
	@Column(name="courses")
	String[] courses;

	@CollectionOfElements
	@JoinTable(name="emails", joinColumns=@JoinColumn(name="sid"))
	@IndexColumn(name="idx")
	@Column(name="emails")
	List<String> emails;

	@CollectionOfElements
	@JoinTable(name="marks", joinColumns=@JoinColumn(name="sid"))
	@IndexColumn(name="idx")
	@Column(name="marks")
	List<Integer> marks;

	@CollectionOfElements
	@JoinTable(name="phones", joinColumns=@JoinColumn(name="sid"))
	@Column(name="phones")
	Set<Long> phones;

	@CollectionOfElements
	@JoinTable(name="refs", joinColumns=@JoinColumn(name="sid"))
	@Column(name="refs")
	Map<String,Long> refs;

	public Student(String sname, String dob, String qualification,
			String[] courses, List<String> emails, List<Integer> marks,
			Set<Long> phones, Map<String, Long> refs) {
		super();
		this.sname = sname;
		this.dob = dob;
		this.qualification = qualification;
		this.courses = courses;
		this.emails = emails;
		this.marks = marks;
		this.phones = phones;
		this.refs = refs;
	}

	// Setters and Getters Method
}
```

AHibernateUtil.java

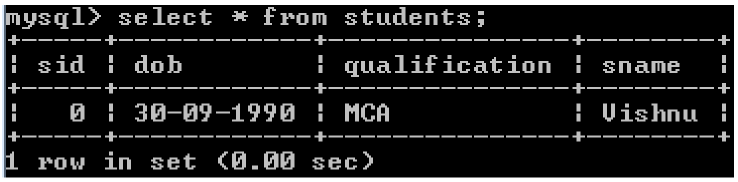
```
package hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class AHibernateUtil {
	static SessionFactory sessionFactory;

	static{
		AnnotationConfiguration cfg=new AnnotationConfiguration();

	cfg=(AnnotationConfiguration)cfg.configure("resource/hibernate.cfg.xml");
		sessionFactory=cfg.buildSessionFactory();
	}

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

hibernate.cfg.xml

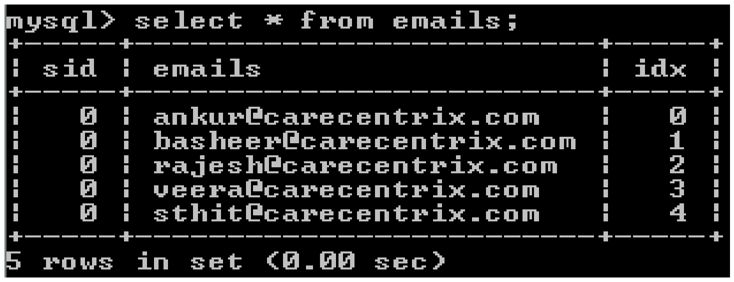
```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
```

```
        <session-factory>
                <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
                <property name="connection.url">jdbc:mysql://localhost/db2</property>
                <property name="connection.username">root</property>
                <property name="connection.password">Westpac1</property>
                <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
                <property name="show_sql">true</property>
                <property name="hbm2ddl.auto">update</property>

                <mapping class="hibernate.Student"/>
        </session-factory>
</hibernate-configuration>
```

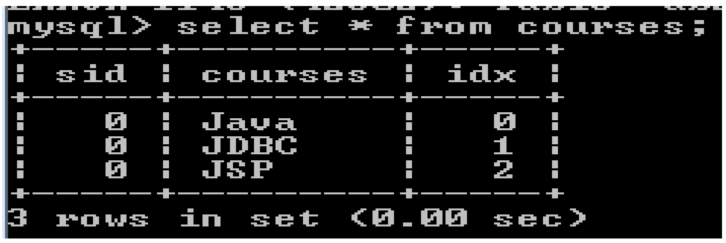
Output:



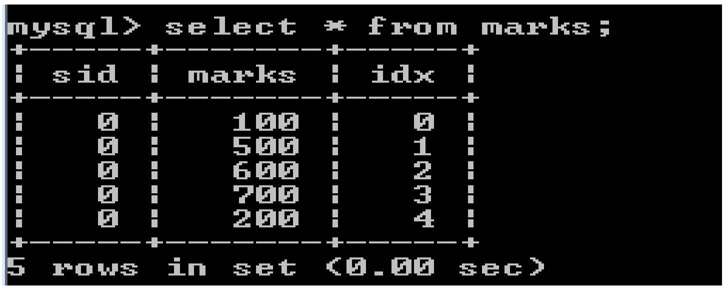
```
mysql> select * from students;
+-----+------------+---------------+--------+
| sid | dob        | qualification | sname  |
+-----+------------+---------------+--------+
|   0 | 30-09-1990 | MCA           | Vishnu |
+-----+------------+---------------+--------+
1 row in set (0.00 sec)
```



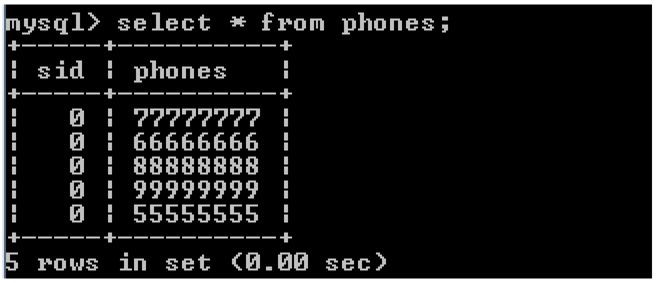
```
mysql> select * from emails;
+-----+---------------------------+-----+
| sid | emails                    | idx |
+-----+---------------------------+-----+
|   0 | ankur@carecentrix.com     |   0 |
|   0 | basheer@carecentrix.com   |   1 |
|   0 | rajesh@carecentrix.com    |   2 |
|   0 | veera@carecentrix.com     |   3 |
|   0 | sthit@carecentrix.com     |   4 |
+-----+---------------------------+-----+
5 rows in set (0.00 sec)
```

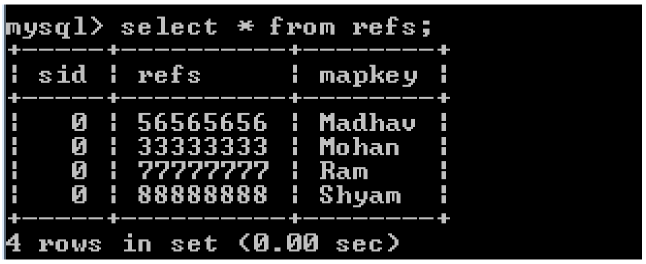


```
mysql> select * from courses;
+-----+---------+-----+
| sid | courses | idx |
+-----+---------+-----+
|   0 | Java    |   0 |
|   0 | JDBC    |   1 |
|   0 | JSP     |   2 |
+-----+---------+-----+
3 rows in set (0.00 sec)
```



```
mysql> select * from marks;
+-----+-------+-----+
| sid | marks | idx |
+-----+-------+-----+
|   0 |   100 |   0 |
|   0 |   500 |   1 |
|   0 |   600 |   2 |
|   0 |   700 |   3 |
|   0 |   200 |   4 |
+-----+-------+-----+
5 rows in set (0.00 sec)
```


```
mysql> select * from phones;
+-----+----------+
| sid | phones   |
+-----+----------+
|   0 | 77777777 |
|   0 | 66666666 |
|   0 | 88888888 |
|   0 | 99999999 |
|   0 | 55555555 |
+-----+----------+
5 rows in set (0.00 sec)
```


```
mysql> select * from refs;
+-----+----------+--------+
| sid | refs     | mapkey |
+-----+----------+--------+
|   0 | 56565656 | Madhav |
|   0 | 33333333 | Mohan  |
|   0 | 77777777 | Ram    |
|   0 | 88888888 | Shyam  |
+-----+----------+--------+
4 rows in set (0.00 sec)
```

Inheritance Mapping

When multiple inheritance classes are in inheritance relationship then use Inheritance Mapping. You can implement inheritance mapping in 3 ways:

1) Table Per Sub Class Mapping
2) Table Per Class Mapping
3) Table Per Concrete Class Mapping

Consider the following persistence classes with inheritance relationship.

***** Draw the Diagram *****

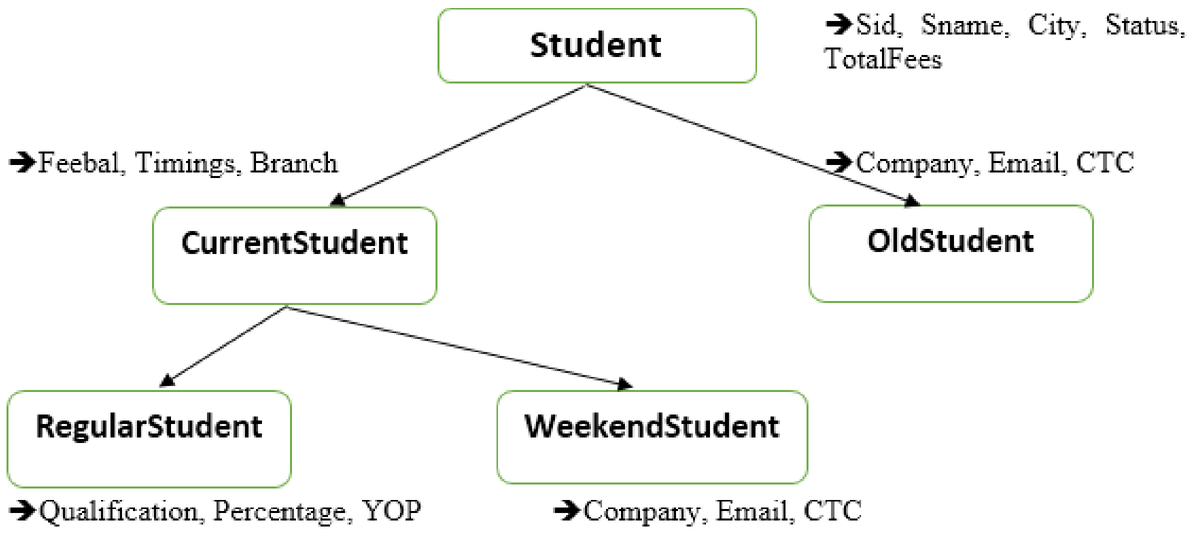


Table per Sub Class Mapping

In this mapping, you need to take one table per one sub class. So **Student** is the main super class which will have the master table called **mystudents**.
Every subclass will have its own table.

- When you save **Student** class object then only one record will be inserted in **mystudents.**
- When you save **CurrentStudent** class object then one record will be inserted in **mystudents** and one record will be inserted in **cstudents.**
- When you save **OldStudent** class object then one record will be inserted in **mystudents** and one record will be inserted in **ostudents**.
- When you save **RegularStudent** class object then one record will be inserted in **mystudents,** one record will be inserted in **cstudents** and one record will be inserted in **rstudents**.
- When you save **WeekendStudent** class object then one record will be inserted in **mystudents,** one record will be inserted in **cstudents** and one record will be inserted in **wstudents**.

1) ***Tables Required.***

a) mystudents:

sid	sname	city	status	totalfee

b) cstudents:

sid	feebal	timings	branch

c) ostudents:

sid	company	cemail	ctc

d) rstudents:

sid	qualification	percentage	yop

e) wstudents:

sid	company	cemail	ctc

```
drop database db3;
create database db3;
use db3;

select * from mystudents;

select * from cstudents;
```

select * from ostudents;

select * from rstudents;

select * from wstudents;

2) ***Client Code.***

Lab03Client.java

```
package hibernate;

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

public class Lab03Client {
	public static void main(String[] args) {
		Integer it=null;
		Transaction tx=null;
		try{
			SessionFactory sf=AHibernateUtil.getSessionFactory();
			Session session=sf.openSession();
			tx=session.beginTransaction();


			// 1. Adding the Student
			Student student=new Student("Ankur 
Shrivastav","Varanasi","Enabled",40000.0);
			it=(Integer)session.save(student);
			System.out.println(it.intValue());



			// 2. Adding the CurrentStudent
			CurrentStudent currentStudent=new CurrentStudent("Ankit 
Shrivastav","Varanasi", "Enabled",35000.0,40000,"4 pm","Cholapur");
			it=(Integer)session.save(currentStudent);
			System.out.println(it.intValue());


			// 3. Adding the OldStudent
			OldStudent oldStudent=new OldStudent("Anshu 
Shrivastav","Delhi","Enabled",20000.0,"TCS","anshu@gmail.com",5000.0);
			it=(Integer)session.save(oldStudent);
			System.out.println(it.intValue());


			// 4. Adding RegularStudent
			RegularStudent regularStudent=new RegularStudent("Akash 
Shrivastav","Kolkata","Enabled",45000.9,4000.8,"20:45 PM","CSA","MS","60 %",2017);
			it=(Integer)session.save(regularStudent);
			System.out.println(it.intValue());
```

```
                    // 5. Adding WeekendStudent
                    WeekendStudent weekendStudent=new
WeekendStudent("Anju","Varanasi","Disabled",2000.9,28000.8,"10:PM","Lalpur","School",
"anju@gmail.com",23000.0);
                    it=(Integer)session.save(weekendStudent);
                    System.out.println(it.intValue());

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

             }catch(Exception e){
                    if(tx!=null){
                           tx.rollback();
                    }
             }
      }
}
```

3) ***Persistence Classes and Entities.***

Student.java

```
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

import javax.persistence.Table;

@Entity
@Table(name="mystudents")
@Inheritance(strategy=InheritanceType.JOINED)
public class Student {
      @Id
      @GeneratedValue(strategy=GenerationType.AUTO)
      @Column(name="sid")
      private int sid;

      @Column(name="sname")
      private String sname;

      @Column(name="city")
      private String city;

      @Column(name="status")
      private String status;

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

```java
	private double totalfee;

	public Student() {
	}

	public Student(String sname, String city, String status, double totalfee) {
		super();
		this.sname = sname;
		this.city = city;
		this.status = status;
		this.totalfee = totalfee;
	}
	// Setters and Getters Method
}
```

CurrentStudent.java

```java
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

@Entity
@Table(name="cstudents")
@PrimaryKeyJoinColumn(name="sid")
public class CurrentStudent extends Student {
	@Column(name="feebal")
	private double feebal;

	@Column(name="timings")
	private String timings;

	@Column(name="branch")
	private String branch;

	public CurrentStudent() {
	}

	public CurrentStudent(String sname, String city, String status,
		double totalfee,double feebal, String timings, String branch) {
		super(sname, city, status, totalfee);
		this.feebal = feebal;
		this.timings = timings;
		this.branch = branch;
	}
	// Setters and Getters Method
}
```

OldStudent.java

```java
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

@Entity
@Table(name="ostudents")
@PrimaryKeyJoinColumn(name="sid")
public class OldStudent extends Student {
	@Column(name="ocompany")
	private String ocompany;

	@Column(name="oemail")
	private String oemail;

	@Column(name="octc")
	private double octc;

	public OldStudent() {
	}

	public OldStudent(String sname, String city, String status, double totalfee,
String ocompany, String oemail, double octc) {
		super(sname, city, status, totalfee);
		this.ocompany = ocompany;
		this.oemail = oemail;
		this.octc = octc;
	}
	// Getters and Setters Method
}
```

RegularStudent.java

```java
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

@Entity
@Table(name="rstudents")
@PrimaryKeyJoinColumn(name="sid")
public class RegularStudent extends CurrentStudent {
	@Column(name="qualification")
	private String qualification;

	@Column(name="percentage")
	private String percentage;

	@Column(name="yoe")
	private int yoe;

	public RegularStudent() {
	}

	RegularStudent(String sname, String city, String status, double totalfee,
                        double feebal, String timings, String branch, String
qualification, String percentage, int yoe) {
                super(sname, city, status, totalfee, feebal, timings, branch);
                this.qualification = qualification;
                this.percentage = percentage;
                this.yoe = yoe;
        }
        // Setters and Getters Method
}
```

WeekendStudent.java

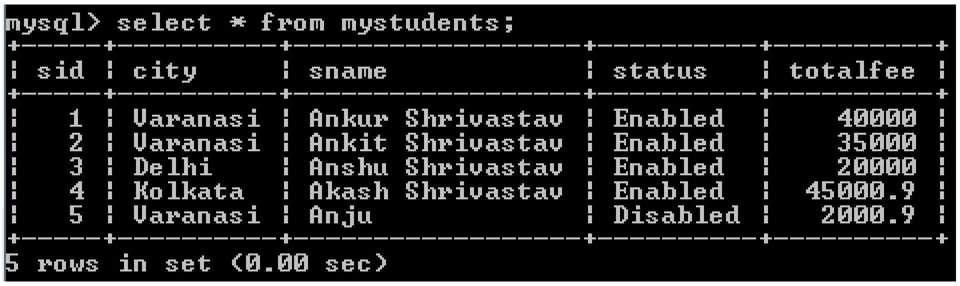
```
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

@Entity
@Table(name="wstudents")
@PrimaryKeyJoinColumn(name="sid")
public class WeekendStudent extends CurrentStudent {
        @Column(name="wcompany")
        private String wcompany;

        @Column(name="wemail")
        private String wemail;

        @Column(name="wctc")
        private double wctc;

        public WeekendStudent() {
        }

        public WeekendStudent(String sname, String city, String status,
                        double totalfee, double feebal, String timings, String branch,
String wcompany, String wemail, double wctc) {
                super(sname, city, status, totalfee, feebal, timings, branch);
                this.wcompany = wcompany;
                this.wemail = wemail;
                this.wctc = wctc;
        }
        // Setters and Getters Method
}
```

hibernate.cfg.xml

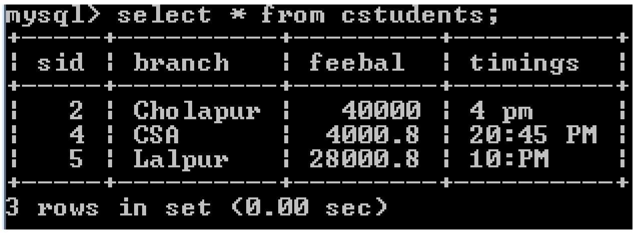
```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
        <session-factory>
```

```
                <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
                <property name="connection.url">jdbc:mysql://localhost/db3</property>
                <property name="connection.username">root</property>
                <property name="connection.password">Westpac1</property>
                <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
                <property name="show_sql">true</property>
                <property name="hbm2ddl.auto">create</property>

                <mapping class="hibernate.Student"/>
                <mapping class="hibernate.CurrentStudent"/>
                <mapping class="hibernate.OldStudent"/>
                <mapping class="hibernate.RegularStudent"/>
                <mapping class="hibernate.WeekendStudent"/>
        </session-factory>
</hibernate-configuration>
```

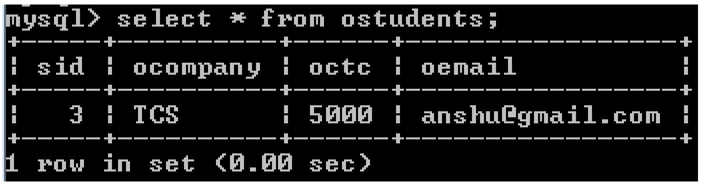
Output:



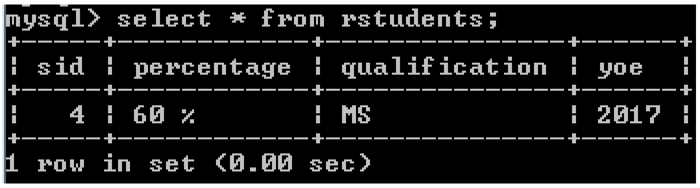
```
mysql> select * from mystudents;
+-----+----------+-----------------+----------+----------+
| sid | city     | sname           | status   | totalfee |
+-----+----------+-----------------+----------+----------+
|   1 | Varanasi | Ankur Shrivastav | Enabled  |    40000 |
|   2 | Varanasi | Ankit Shrivastav | Enabled  |    35000 |
|   3 | Delhi    | Anshu Shrivastav | Enabled  |    20000 |
|   4 | Kolkata  | Akash Shrivastav | Enabled  |  45000.9 |
|   5 | Varanasi | Anju            | Disabled |   2000.9 |
+-----+----------+-----------------+----------+----------+
5 rows in set (0.00 sec)
```



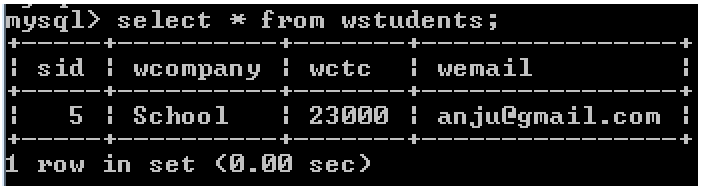
```
mysql> select * from cstudents;
+-----+----------+---------+-----------+
| sid | branch   | feebal  | timings   |
+-----+----------+---------+-----------+
|   2 | Cholapur |   40000 | 4 pm      |
|   4 | CSA      |  4000.8 | 20:45 PM  |
|   5 | Lalpur   | 28000.8 | 10:PM     |
+-----+----------+---------+-----------+
3 rows in set (0.00 sec)
```



```
mysql> select * from ostudents;
+-----+----------+------+------------------+
| sid | ocompany | octc | oemail           |
+-----+----------+------+------------------+
|   3 | TCS      | 5000 | anshu@gmail.com  |
+-----+----------+------+------------------+
1 row in set (0.00 sec)
```



```
mysql> select * from rstudents;
+-----+------------+---------------+------+
| sid | percentage | qualification | yoe  |
+-----+------------+---------------+------+
|   4 | 60 %       | MS            | 2017 |
+-----+------------+---------------+------+
1 row in set (0.00 sec)
```



```
mysql> select * from wstudents;
+-----+----------+-------+----------------+
| sid | wcompany | wctc  | wemail         |
+-----+----------+-------+----------------+
|   5 | School   | 23000 | anju@gmail.com |
+-----+----------+-------+----------------+
1 row in set (0.00 sec)
```

Console:

```
Hibernate: insert into mystudents (city, sname, status, totalfee) values (?, ?, ?, ?)
1
Hibernate: insert into mystudents (city, sname, status, totalfee) values (?, ?, ?, ?)
Hibernate: insert into cstudents (branch, feebal, timings, sid) values (?, ?, ?, ?)
2
Hibernate: insert into mystudents (city, sname, status, totalfee) values (?, ?, ?, ?)
Hibernate: insert into ostudents (ocompany, octc, oemail, sid) values (?, ?, ?, ?)
3
Hibernate: insert into mystudents (city, sname, status, totalfee) values (?, ?, ?, ?)
Hibernate: insert into cstudents (branch, feebal, timings, sid) values (?, ?, ?, ?)
Hibernate: insert into rstudents (percentage, qualification, yoe, sid) values (?, ?,
?, ?)
4
Hibernate: insert into mystudents (city, sname, status, totalfee) values (?, ?, ?, ?)
Hibernate: insert into cstudents (branch, feebal, timings, sid) values (?, ?, ?, ?)
Hibernate: insert into wstudents (wcompany, wctc, wemail, sid) values (?, ?, ?, ?)
5
```

Table per Class Mapping

In this mapping, you need to take only one table for all the super and sub class. It is also called as Single Table Mapping.

4) ***Tables Required.***

a) mystudents1:

sid	stutype	sname	city	status	totalfee
feebal	timings	branch	ocompany	oemail	octc
qualification	percentage	yop	wcompany	wemail	wctc

```
drop database db4;
create database db4;
use db4;

select * from mystudents;
```

Note(For Core Hibernate): In the above table, there is one special column called discriminator column-stutype but there is no variable for this column in any persistence class. You have to specify the discriminator column value in mapping doc.

Note 1: You must specify all the persistence classes in **hibernate.cfg.xml** which are marked with **@Entity**

Note 2: In the case of Table per class Mapping or Single Table Mapping use the following for super class.

@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="stutype", length=4)

And use of the following for super class and all the sub classes-

@DiscriminatorValue(value="XXX")

5) Client Code.

Lab04Client.java

```
package hibernate;

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

public class Lab04Client {
	public static void main(String[] args) {
		Integer it=null;
		Transaction tx=null;
		try{
			SessionFactory sf=AHibernateUtil.getSessionFactory();
			Session session=sf.openSession();
			tx=session.beginTransaction();


			// 1. Adding the Student
			Student student=new Student("Ankur
Shrivastav","Varanasi","Enabled",40000.0);
			it=(Integer)session.save(student);
			System.out.println(it.intValue());



			// 2. Adding the CurrentStudent
			CurrentStudent currentStudent=new CurrentStudent("Ankit
Shrivastav","Varanasi", "Enabled",35000.0,40000,"4 pm","Cholapur");
			it=(Integer)session.save(currentStudent);
			System.out.println(it.intValue());


			// 3. Adding the OldStudent
```

```
                    OldStudent oldStudent=new OldStudent("Anshu 
Shrivastav","Delhi","Enabled",20000.0,"TCS","anshu@gmail.com",5000.0);
                    it=(Integer)session.save(oldStudent);
                    System.out.println(it.intValue());


                    // 4. Adding RegularStudent
                    RegularStudent regularStudent=new RegularStudent("Akash 
Shrivastav","Kolkata","Enabled",45000.9,4000.8,"20:45 PM","CSA","MS","60 %",2017);
                    it=(Integer)session.save(regularStudent);
                    System.out.println(it.intValue());


                    // 5. Adding WeekendStudent
                    WeekendStudent weekendStudent=new 
WeekendStudent("Anju","Varanasi","Disabled",2000.9,28000.8,"10:PM","Lalpur","School",
"anju@gmail.com",23000.0);
                    it=(Integer)session.save(weekendStudent);
                    System.out.println(it.intValue());

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

             }catch(Exception e){
                    if(tx!=null){
                           tx.rollback();
                    }
             }
      }
}
```

6) ***Persistence Classes (or) Entities.***

Student.java

```
package hibernate;

import javax.persistence.Column;
import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

import javax.persistence.Table;

@Entity
@Table(name="mystudents")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="stutype", length=4)
```

```
@DiscriminatorValue(value="STU")
public class Student {
	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	@Column(name="sid")
	private int sid;

	@Column(name="sname")
	private String sname;

	@Column(name="city")
	private String city;

	@Column(name="status")
	private String status;

	@Column(name="totalfee")
	private double totalfee;

	public Student() {
	}

	public Student(String sname, String city, String status, double totalfee) {
		super();
		this.sname = sname;
		this.city = city;
		this.status = status;
		this.totalfee = totalfee;
	}
	// Setters and Getters Method
}
```

CurrentStudent.java

```
package hibernate;

import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue(value="CSTU")
public class CurrentStudent extends Student {
	@Column(name="feebal")
	private double feebal;

	@Column(name="timings")
	private String timings;

	@Column(name="branch")
	private String branch;

	public CurrentStudent() {
	}
	public CurrentStudent(String sname, String city, String status,
		double totalfee,double feebal, String timings, String branch) {
		super(sname, city, status, totalfee);
		this.feebal = feebal;
		this.timings = timings;
		this.branch = branch;
	}
	// Setters and Getters Method
}
```

OldStudent.java

```
package hibernate;

import javax.persistence.Column;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue(value="OSTU")
public class OldStudent extends Student {
	@Column(name="ocompany")
	private String ocompany;

	@Column(name="oemail")
	private String oemail;

	@Column(name="octc")
	private double octc;

	public OldStudent() {
	}

	public OldStudent(String sname, String city, String status, double totalfee,
String ocompany, String oemail, double octc) {
		super(sname, city, status, totalfee);
		this.ocompany = ocompany;
		this.oemail = oemail;
		this.octc = octc;
	}
	// Setters and Getters Method
}
```

RegularStudent.java

```
package hibernate;

import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue(value="RSTU")
public class RegularStudent extends CurrentStudent {
	@Column(name="qualification")
```

```java
	private String qualification;

	@Column(name="percentage")
	private String percentage;

	@Column(name="yoe")
	private int yoe;

	public RegularStudent() {
	}

	RegularStudent(String sname, String city, String status, double totalfee,
			double feebal, String timings, String branch, String
qualification, String percentage, int yoe) {
		super(sname, city, status, totalfee, feebal, timings, branch);
		this.qualification = qualification;
		this.percentage = percentage;
		this.yoe = yoe;
	}
	// Setters and Getters Method
}
```

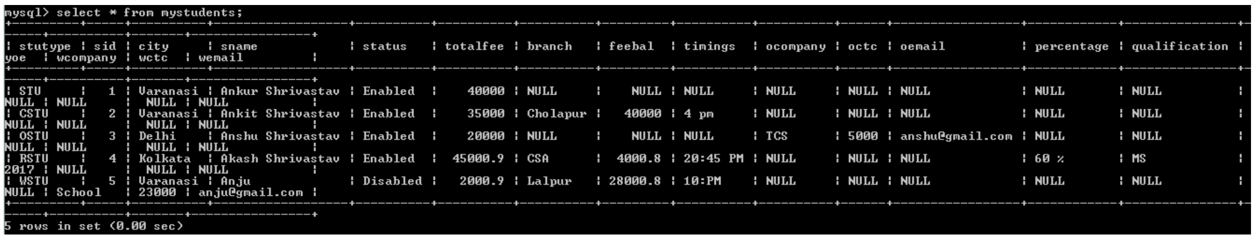
WeekendStudent.java

```java
package hibernate;

import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue(value="WSTU")

public class WeekendStudent extends CurrentStudent {
	@Column(name="wcompany")
	private String wcompany;

	@Column(name="wemail")
	private String wemail;

	@Column(name="wctc")
	private double wctc;

	public WeekendStudent() {
	}

	public WeekendStudent(String sname, String city, String status,
			double totalfee, double feebal, String timings, String branch,
String wcompany, String wemail, double wctc) {
		super(sname, city, status, totalfee, feebal, timings, branch);
		this.wcompany = wcompany;
		this.wemail = wemail;
		this.wctc = wctc;
	}
	// Setters and Getters Method
}
```

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="connection.driver_class">com.mysql.jdbc.Driver</property>
            <property name="connection.url">jdbc:mysql://localhost/db4</property>
            <property name="connection.username">root</property>
            <property name="connection.password">Westpac1</property>
            <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
            <property name="show_sql">true</property>
            <property name="hbm2ddl.auto">create</property>

            <mapping class="hibernate.Student"/>
            <mapping class="hibernate.CurrentStudent"/>
            <mapping class="hibernate.OldStudent"/>
            <mapping class="hibernate.RegularStudent"/>
            <mapping class="hibernate.WeekendStudent"/>
      </session-factory>
</hibernate-configuration>
```

Output:



```
mysql> select * from mystudents;
+---------+-----+----------+-------------------+----------+----------+----------+---------+----------+----------+------+-----------------+------------+---------------+
-----+----------+--------+-----------------+
| stutype | sid | city     | sname             | status   | totalfee | branch   | feebal  | timings  | ocompany | octc | oemail          | percentage | qualification |
yoe  | wcompany | wctc   | wemail          |
+---------+-----+----------+-------------------+----------+----------+----------+---------+----------+----------+------+-----------------+------------+---------------+
-----+----------+--------+-----------------+
| STU     |   1 | Varanasi | Ankur Shrivastav  | Enabled  |    40000 | NULL     |    NULL | NULL     | NULL     | NULL | NULL            | NULL       | NULL          |
NULL | NULL     |   NULL | NULL            |
| CSTU    |   2 | Varanasi | Ankit Shrivastav  | Enabled  |    35000 | Cholapur |   40000 | 4 pm     | NULL     | NULL | NULL            | NULL       | NULL          |
NULL | NULL     |   NULL | NULL            |
| OSTU    |   3 | Delhi    | Anshu Shrivastav  | Enabled  |    20000 | NULL     |    NULL | NULL     | TCS      | 5000 | anshu@gmail.com | NULL       | NULL          |
NULL | NULL     |   NULL | NULL            |
| RSTU    |   4 | Kolkata  | Akash Shrivastav  | Enabled  |  45000.9 | CSA      |  4000.8 | 20:45 PM | NULL     | NULL | NULL            | 60 %       | MS            |
2017 | NULL     |   NULL | NULL            |
| WSTU    |   5 | Varanasi | Anju              | Disabled |   2000.9 | Lalpur   | 28000.8 | 10:PM    | NULL     | NULL | NULL            | NULL       | NULL          |
NULL | School   | 23000  | anju@gmail.com  |
+---------+-----+----------+-------------------+----------+----------+----------+---------+----------+----------+------+-----------------+------------+---------------+
-----+----------+--------+-----------------+
5 rows in set (0.00 sec)
```

Console:

```
Hibernate: insert into mystudents (city, sname, status, totalfee, stutype) values (?,
?, ?, ?, 'STU')
1
Hibernate: insert into mystudents (city, sname, status, totalfee, branch, feebal,
timings, stutype) values (?, ?, ?, ?, ?, ?, ?, 'CSTU')
2
Hibernate: insert into mystudents (city, sname, status, totalfee, ocompany, octc,
oemail, stutype) values (?, ?, ?, ?, ?, ?, ?, 'OSTU')
3
Hibernate: insert into mystudents (city, sname, status, totalfee, branch, feebal,
timings, percentage, qualification, yoe, stutype) values (?, ?, ?, ?, ?, ?, ?, ?, ?,
?, 'RSTU')
4
```

```
Hibernate: insert into mystudents (city, sname, status, totalfee, branch, feebal,
timings, wcompany, wctc, wemail, stutype) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
'WSTU')
5
```

Table per Concrete Class Mapping

In this mapping, you need to take one table for one concrete class.

1) Tables Required.

mystudents1:

sid sname city status totalfee

cstudents1:

sid sname city status totalfee feebal
timings branch

ostudents1:

sid sname city status totalfee ocompany oemail
octc

rstudents:

sid sname city status totalfee feebal
timings branch qualification percentage yop

wstudents:

sid sname city status totalfee feebal
timings branch wcompany wcemail wctc

```
drop database db5;

create database db5;
use db5;
```

I. Client Code.

Lab05Client.java

```
package hibernate;

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

public class Lab05Client {
	public static void main(String[] args) {
		Integer it=null;
		Transaction tx=null;
		try{
			SessionFactory sf=AHibernateUtil.getSessionFactory();
			Session session=sf.openSession();
			tx=session.beginTransaction();

/*
			// 1. Adding the Student
			Student student=new Student("Ankur
Shrivastav","Varanasi","Enabled",40000.0);
			it=(Integer)session.save(student);
			System.out.println(it.intValue());

*/

			// 2. Adding the CurrentStudent
			CurrentStudent currentStudent=new CurrentStudent("Ankit
Shrivastav","Varanasi", "Enabled",35000.0,40000,"4 pm","Cholapur");
			it=(Integer)session.save(currentStudent);
			System.out.println(it.intValue());

/*
			// 3. Adding the OldStudent
			OldStudent oldStudent=new OldStudent("Anshu
Shrivastav","Delhi","Enabled",20000.0,"TCS","anshu@gmail.com",5000.0);
			it=(Integer)session.save(oldStudent);
			System.out.println(it.intValue());


			// 4. Adding RegularStudent
			RegularStudent regularStudent=new RegularStudent("Akash
Shrivastav","Kolkata","Enabled",45000.9,4000.8,"20:45 PM","CSA","MS","60 %",2017);
			it=(Integer)session.save(regularStudent);
			System.out.println(it.intValue());



			// 5. Adding WeekendStudent
			WeekendStudent weekendStudent=new
WeekendStudent("Anju","Varanasi","Disabled",2000.9,28000.8,"10:PM","Lalpur","School",
"anju@gmail.com",23000.0);
			it=(Integer)session.save(weekendStudent);
			System.out.println(it.intValue());
*/
			tx.commit();
			session.close();
		}catch(Exception e){
			if(tx!=null){
				tx.rollback();
			}
		}
	}
}
```

II. ***<u>Persistence Classes (or) Entities.</u>***

Student.java

```java
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

import javax.persistence.Table;

@Entity
@Table(name="mystudents")
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Student {
	@Id
	@Column(name="sid")
	private int sid;

	@Column(name="sname")
	private String sname;

	@Column(name="city")
	private String city;

	@Column(name="status")
	private String status;

	@Column(name="totalfee")
	private double totalfee;

	public Student() {
	}

	public Student(String sname, String city, String status, double totalfee) {
		super();
		this.sname = sname;
		this.city = city;
		this.status = status;
		this.totalfee = totalfee;
	}
	// Setters and Getters Method
}
```

CurrentStudent.java

```java
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name="cstudents")
public class CurrentStudent extends Student {
     @Column(name="feebal")
     private double feebal;

     @Column(name="timings")
     private String timings;

     @Column(name="branch")
     private String branch;

     public CurrentStudent() {
     }

     public CurrentStudent(String sname, String city, String status,
             double totalfee,double feebal, String timings, String branch) {
             super(sname, city, status, totalfee);
             this.feebal = feebal;
             this.timings = timings;
             this.branch = branch;
     }
     // Setters and Getters Method
}
```

OldStudent.java

```
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name="ostudents")
public class OldStudent extends Student {
        @Column(name="ocompany")
        private String ocompany;

        @Column(name="oemail")
        private String oemail;

        @Column(name="octc")
        private double octc;

        public OldStudent() {
        }

        public OldStudent(String sname, String city, String status, double totalfee,
String ocompany, String oemail, double octc) {
                super(sname, city, status, totalfee);
                this.ocompany = ocompany;
                this.oemail = oemail;
                this.octc = octc;
        }
        // Setters and Getters Method
}
```

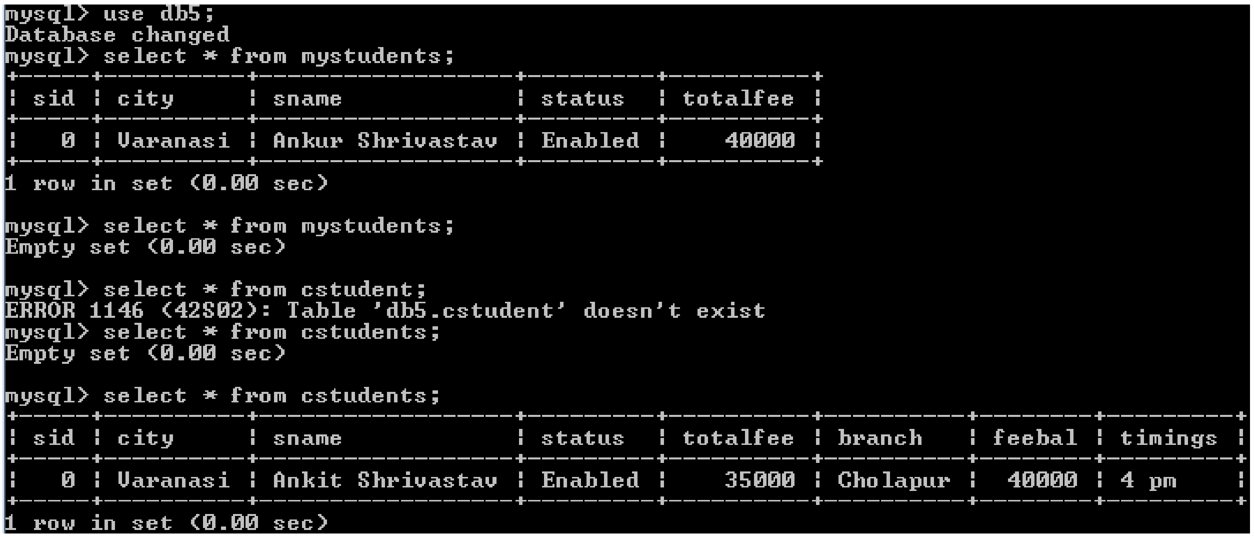
RegularStudent.java
```
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name="rstudents")
public class RegularStudent extends CurrentStudent {
	@Column(name="qualification")
	private String qualification;

	@Column(name="percentage")
	private String percentage;

	@Column(name="yoe")
	private int yoe;

	public RegularStudent() {
	}

	RegularStudent(String sname, String city, String status, double totalfee,
			double feebal, String timings, String branch, String
qualification, String percentage, int yoe) {
		super(sname, city, status, totalfee, feebal, timings, branch);
		this.qualification = qualification;
		this.percentage = percentage;
		this.yoe = yoe;
	}
	// Setters and Getters Method
}
```

WeekendStudent.java
```
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name="wstudents")

public class WeekendStudent extends CurrentStudent {
	@Column(name="wcompany")
	private String wcompany;

	@Column(name="wemail")
	private String wemail;

	@Column(name="wctc")
	private double wctc;

	public WeekendStudent() {
	}

	public WeekendStudent(String sname, String city, String status,
			double totalfee, double feebal, String timings, String branch,
String wcompany, String wemail, double wctc) {
		super(sname, city, status, totalfee, feebal, timings, branch);
		this.wcompany = wcompany;
		this.wemail = wemail;
		this.wctc = wctc;
	}

	// Setters and Getters Method
}
```

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="connection.driver_class">com.mysql.jdbc.Driver</property>
		<property name="connection.url">jdbc:mysql://localhost/db5</property>
		<property name="connection.username">root</property>
		<property name="connection.password">Westpac1</property>
		<property name="dialect">org.hibernate.dialect.MySQLDialect</property>
		<property name="show_sql">true</property>
		<property name="hbm2ddl.auto">create</property>

		<mapping class="hibernate.Student"/>
		<mapping class="hibernate.CurrentStudent"/>
		<mapping class="hibernate.OldStudent"/>
		<mapping class="hibernate.RegularStudent"/>
		<mapping class="hibernate.WeekendStudent"/>
	</session-factory>
</hibernate-configuration>
```

Output:

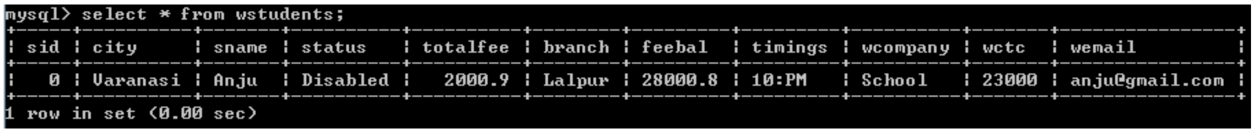


```
mysql> use db5;
Database changed
mysql> select * from mystudents;
+-----+----------+-------------------+----------+----------+
| sid | city     | sname             | status   | totalfee |
+-----+----------+-------------------+----------+----------+
|   0 | Varanasi | Ankur Shrivastav  | Enabled  |    40000 |
+-----+----------+-------------------+----------+----------+
1 row in set (0.00 sec)

mysql> select * from mystudents;
Empty set (0.00 sec)

mysql> select * from cstudent;
ERROR 1146 (42S02): Table 'db5.cstudent' doesn't exist
mysql> select * from cstudents;
Empty set (0.00 sec)

mysql> select * from cstudents;
+-----+----------+-------------------+----------+----------+----------+--------+---------+
| sid | city     | sname             | status   | totalfee | branch   | feebal | timings |
+-----+----------+-------------------+----------+----------+----------+--------+---------+
|   0 | Varanasi | Ankit Shrivastav  | Enabled  |    35000 | Cholapur |  40000 | 4 pm    |
+-----+----------+-------------------+----------+----------+----------+--------+---------+
1 row in set (0.00 sec)
```

Console for only cstudents:

```
0
Hibernate: insert into cstudents (city, sname, status, totalfee, branch, feebal,
timings, sid) values (?, ?, ?, ?, ?, ?, ?, ?)
```

Output for only wstudents:



```
mysql> select * from wstudents;
+-----+----------+-------+----------+----------+--------+---------+---------+----------+-------+----------------+
| sid | city     | sname | status   | totalfee | branch | feebal  | timings | wcompany | wctc  | wemail         |
+-----+----------+-------+----------+----------+--------+---------+---------+----------+-------+----------------+
|   0 | Varanasi | Anju  | Disabled |   2000.9 | Lalpur | 28000.8 | 10:PM   | School   | 23000 | anju@gmail.com |
+-----+----------+-------+----------+----------+--------+---------+---------+----------+-------+----------------+
1 row in set (0.00 sec)
```

Console for only cstudents:

```
0
Hibernate: insert into wstudents (city, sname, status, totalfee, branch, feebal,
timings, wcompany, wctc, wemail, sid) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

Like that we will get all the value regarding the value putting in the Client code.

Note:

- ❖ You must specify all the persistence classes in hibernate.cfg.xml which are marked with @Entity.
- ❖ Don't specify the primary key generation for sid.
- ❖ You set the sid value to all the objects in the client code.
- ❖ Don't save all types of Student class object in one session.

In the case of Table per concrete class mapping, use the following for super class-
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS) and no need to use anything the sub class.

DAO & Hibernate Annotation

Tables

```
drop database db6;
create database db6;
use db6;

select * from customers;
```

Customer.java

```
package hibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="customers")
public class Customer {

	@Column(name="cid")
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int cid;

	@Column(name="cname")
	private String cname;

	@Column(name="email")
	private String email;

	@Column(name="phone")
	private long phone;

	@Column(name="city")
	private String city;

	@Column(name="status")
	private String status;


	public Customer() {}

	public Customer(String cname, String email, long phone, String city,
			String status) {
		super();
		this.cname = cname;
		this.email = email;
		this.phone = phone;
		this.city = city;
		this.status = status;
	}

	public Customer(int cid, String cname, String email, long phone,
			String city, String status) {
		super();
		this.cid = cid;
		this.cname = cname;
		this.email = email;
		this.phone = phone;
		this.city = city;
		this.status = status;
	}
	// Setters and Getters Method
}
```

CustomerDAO.java

```
package hibernate;

public interface CustomerDAO {
	public int addCustomer(CustomerTO customer);
	public void updateCustomer(CustomerTO cust);
	public void  deleteCustomer(int cid);
	public CustomerTO getCustomerByCid(int cid);
}
```

AHibernateUtil.java

```
package hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class AHibernateUtil {
	static SessionFactory sessionFactory=null;

	static{
		AnnotationConfiguration cfg=new AnnotationConfiguration();

	cfg=(AnnotationConfiguration)cfg.configure("resource/hibernate.cfg.xml");
		sessionFactory=cfg.buildSessionFactory();
	}

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

CustomerTO.java

```
package hibernate;

public class CustomerTO {
```

```java
	private int cid;
	private String cname;
	private String email;
	private long phone;
	private String city;
	private String status;

	public CustomerTO(String cname, String email, long phone, String city,
			String status) {
		super();
		this.cname = cname;
		this.email = email;
		this.phone = phone;
		this.city = city;
		this.status = status;
	}

	public CustomerTO(int cid, String cname, String email, long phone,
			String city, String status) {
		super();
		this.cid = cid;
		this.cname = cname;
		this.email = email;
		this.phone = phone;
		this.city = city;
		this.status = status;
	}
	// Setters and Getters Method
}
```

DAOFactory.java

```java
package hibernate;

public class DAOFactory {
	static CustomerDAO customerDAO;

	static{
		customerDAO=new HibernateCustomerDAO();
	}

	public static CustomerDAO getCustomerDAO() {
		return customerDAO;
	}
}
```

HibernateCustomerDAO.java

```java
package hibernate;

public class HibernateCustomerDAO implements CustomerDAO {
	@Override
	public int addCustomer(CustomerTO customerTO) {
		Customer customer=new 
Customer(customerTO.getCname(),customerTO.getEmail(),customerTO.getPhone(),customerTO
.getCity(), customerTO.getStatus());
		Integer it=(Integer) HibernateTemplate.saveObject(customer);
		return it.intValue();
	}

	@Override
	public void updateCustomer(CustomerTO customerTO) {
		Customer customer=new 
Customer(customerTO.getCid(),customerTO.getCname(),customerTO.getEmail(),customerTO.g
etPhone(),customerTO.getCity(),customerTO.getStatus());
		HibernateTemplate.updateObject(customer);
	}

	@Override
	public void deleteCustomer(int cid) {
		HibernateTemplate.deleteObject(Customer.class,cid);
	}

	@Override
	public CustomerTO getCustomerByCid(int cid) {
		Customer customer=(Customer) 
HibernateTemplate.loadObject(Customer.class,cid);
		CustomerTO customerTO=new 
CustomerTO(customer.getCid(),customer.getCname(),customer.getEmail(),customer.getPhon
e(),customer.getCity(),customer.getStatus());
		return customerTO;
	}
}
```

HibernateTemplate.java

```
package hibernate;

import java.io.Serializable;

import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.classic.Session;

public class HibernateTemplate {

	public static Object saveObject(Object object) {
		Object id=null;
		try{
			SessionFactory sf=AHibernateUtil.getSessionFactory();
			Session session=sf.openSession();
			Transaction tx=session.beginTransaction();
			id=session.save(object);
			tx.commit();
			session.close();
		}catch(Exception e){
```

```
                e.printStackTrace();
            }
            return id;
        }


        public static void updateObject(Object object) {
            try{
                SessionFactory sf=AHibernateUtil.getSessionFactory();
                Session session=sf.openSession();
                Transaction tx=session.beginTransaction();
                session.update(object);
                tx.commit();
                session.close();
            }catch(Exception e){
                e.printStackTrace();
            }
        }


        public static void deleteObject(Class<Customer> cls, Serializable s) {
            try{
                SessionFactory sf=AHibernateUtil.getSessionFactory();
                Session session=sf.openSession();
                Transaction tx=session.beginTransaction();
                Object o=session.load(cls,s);
                session.delete(o);
                tx.commit();
                session.close();
            }catch(Exception e){
                e.printStackTrace();
            }
        }


        public static Object loadObject(Class<Customer> cls, Serializable s) {
            Object o=null;
            try{
                SessionFactory sf=AHibernateUtil.getSessionFactory();
                Session session=sf.openSession();
                Transaction tx=session.beginTransaction();
                o=session.load(cls,s);
                tx.commit();
                session.close();
            }catch(Exception e){
                e.printStackTrace();
            }
            return o;
        }

}
```

hibernate.cfg.xml

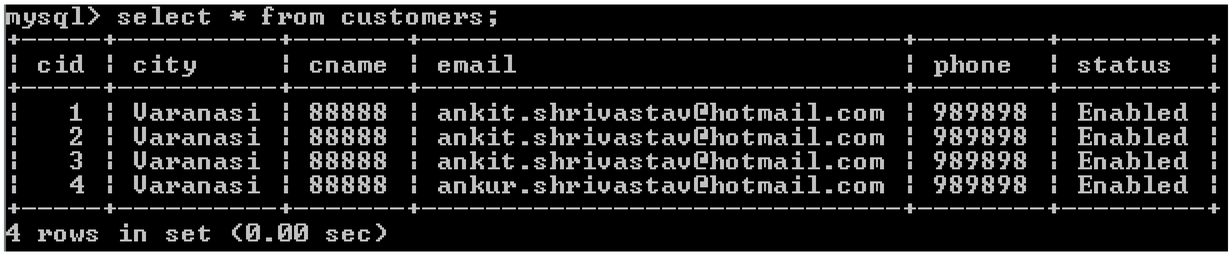
```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<!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="connection.driver_class">com.mysql.jdbc.Driver</property>
              <property name="connection.url">jdbc:mysql://localhost/db6</property>
              <property name="connection.username">root</property>
              <property name="connection.password">Westpac1</property>
              <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
              <property name="show_sql">true</property>
              <property name="hbm2ddl.auto">update</property>

              <mapping class="hibernate.Customer"/>
       </session-factory>
</hibernate-configuration>
```

Lab01Client.java

```
package hibernate;

public class Lab01Client {
       public static void main(String[] args) {

              CustomerDAO cdao=DAOFactory.getCustomerDAO();
              cdao.deleteCustomer(9);

              // 1. Add Customer
              CustomerTO customerTO=new
CustomerTO("88888","ankur.shrivastav@hotmail.com",989898,"Varanasi","Enabled");
              cdao.addCustomer(customerTO);


/*
              // 2. get Customer
              CustomerTO c1=cdao.getCustomerByCid(3);

       System.out.println(c1.getCid()+"\t"+c1.getCname()+"\t"+c1.getEmail()+"\t"+c1.g
etPhone()+"\t"+c1.getCity()+"\t"+c1.getStatus());

              // 3. delete Customer
              cdao.deleteCustomer(4);

              // 4. update Customer
              CustomerTO c2=cdao.getCustomerByCid(2);
              c2.setCname("Lakshminarayan");
              c2.setEmail("lakshminarayan@gmail.com");
              c2.setPhone(888888);
              cdao.updateCustomer(c2);
*/
              }
}
```

Output:



```
mysql> select * from customers;
+-----+----------+-------+------------------------------+--------+---------+
| cid | city     | cname | email                        | phone  | status  |
+-----+----------+-------+------------------------------+--------+---------+
|   1 | Varanasi | 88888 | ankit.shrivastav@hotmail.com | 989898 | Enabled |
|   2 | Varanasi | 88888 | ankit.shrivastav@hotmail.com | 989898 | Enabled |
|   3 | Varanasi | 88888 | ankit.shrivastav@hotmail.com | 989898 | Enabled |
|   4 | Varanasi | 88888 | ankur.shrivastav@hotmail.com | 989898 | Enabled |
+-----+----------+-------+------------------------------+--------+---------+
4 rows in set (0.00 sec)
```

DAO and Collection & Inheritance Mapping

Tables Required

```
drop database DAO_db;
create database DAO_db;
use DAO_db;

select * from customers;
```

Candidate.java

```java
package hibernate;

import java.util.List;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.Table;

import org.hibernate.annotations.CollectionOfElements;
import org.hibernate.annotations.IndexColumn;
import org.hibernate.annotations.LazyCollection;
import org.hibernate.annotations.LazyCollectionOption;
import org.hibernate.annotations.Proxy;

@Entity
@Table(name="candidates")
@Proxy(lazy=false)
@Inheritance(strategy=InheritanceType.JOINED)
public class Candidate {

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

	@Column(name="fname")
	private String fname;

	@Column(name="lname")
	private String lname;

	@CollectionOfElements
	@JoinTable(name="emails", joinColumns=@JoinColumn(name="cid"))
	@IndexColumn(name="idx")
	@Column(name="emailId")
	@LazyCollection(LazyCollectionOption.FALSE)
	private List<String> emails;
```

```
        @Column(name="qualification")
        private String qualification;

        @Column(name="dob")
        private String dob;

        public Candidate() {}

        public Candidate(String fname, String lname, List<String> emails,
                     String qualification, String dob) {
                super();
                this.fname = fname;
                this.lname = lname;
                this.emails = emails;
                this.qualification = qualification;
                this.dob = dob;
        }

        // Setters and Getters Method
}
```

FreshCandidate.java

```
package hibernate;

import java.util.List;
import java.util.Map;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

import org.hibernate.annotations.CollectionOfElements;
import org.hibernate.annotations.LazyCollection;
import org.hibernate.annotations.LazyCollectionOption;
import org.hibernate.annotations.Proxy;

@Entity
@Table(name="freshcandidates")
@Proxy(lazy=false)
@PrimaryKeyJoinColumn(name="cid")
public class FreshCandidate extends Candidate {
        @Column(name="yop")
        int yop;

        @CollectionOfElements
        @JoinTable(name="percentages", joinColumns=@JoinColumn(name="cid"))
        @Column(name="percentage")
        @LazyCollection(LazyCollectionOption.FALSE)
        Map<String,Double> percentages;
```

```
	@Column(name="yearGaps")
	String yearGaps;

	public FreshCandidate(){

	}

	public FreshCandidate(String fname, String lname, List<String> emails, String 
qualification, String dob, int yop, 
			Map<String, Double> percentages, String yearGaps) {
		super(fname, lname, emails, qualification, dob);
		this.yop = yop;
		this.percentages = percentages;
		this.yearGaps = yearGaps;
	}
	// Setters and Getters Method
}
```

<u>ExpCandidate.java</u>

```
package hibernate;

import java.util.List;
import java.util.Set;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

import org.hibernate.annotations.CollectionOfElements;
import org.hibernate.annotations.IndexColumn;
import org.hibernate.annotations.LazyCollection;
import org.hibernate.annotations.LazyCollectionOption;
import org.hibernate.annotations.Proxy;

@Entity
@Table(name="expcandidates")
@Proxy(lazy=false)
@PrimaryKeyJoinColumn(name="cid")
public class ExpCandidate extends Candidate {
	@Column(name="yoe")
	int yoe;

	@CollectionOfElements
	@JoinTable(name="skills", joinColumns=@JoinColumn(name="cid"))
	@Column(name="skillName")
	@LazyCollection(LazyCollectionOption.FALSE)
	Set<String> skills;

	@CollectionOfElements
	@JoinTable(name="companies", joinColumns=@JoinColumn(name="cid"))
	@IndexColumn(name="idx")
	@Column(name="companyName")
	@LazyCollection(LazyCollectionOption.FALSE)
	List<String> companies;

	@Column(name="ctc")
	double ctc;

	public ExpCandidate(){

	}

	public ExpCandidate(String fname, String lname, List<String> emails, String
qualification, String dob, int yoe,
			Set<String> skills, List<String> companies, double ctc) {
		super(fname, lname, emails, qualification, dob);
		this.yoe = yoe;
		this.skills = skills;
		this.companies = companies;
		this.ctc = ctc;
	}

	// Setters and Getters Method
}
```

CandidateDAO.java

```
package hibernate;

import java.util.List;

public interface CandidateDAO {
	public void addCandidate(Candidate can);
	public void updateCandidate(Candidate can);
	public void deleteCandidate(int cid);
	public Candidate getCandidateByCid(int cid);
	public List<Candidate> getAllCandidates();
}
```

AHibernateUtil.java

```
package hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class AHibernateUtil {
	static SessionFactory sessionFactory=null;

	static{
		AnnotationConfiguration cfg=new AnnotationConfiguration();
		cfg=(AnnotationConfiguration)cfg.configure("hibernate.cfg.xml");
		sessionFactory=cfg.buildSessionFactory();
	}

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

AHibernateTemplate.java

```
package hibernate;

import java.io.Serializable;

import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.classic.Session;

public class AHibernateTemplate {

	public static Object save(Object object) {
		Object id=null;
		try{
			SessionFactory sf=AHibernateUtil.getSessionFactory();
			Session session=sf.openSession();
			Transaction tx=session.beginTransaction();
			id=session.save(object);
			tx.commit();
			session.close();
		}catch(Exception e){
			e.printStackTrace();
		}
		return id;
	}


	public static void update(Object object) {
		try{
			SessionFactory sf=AHibernateUtil.getSessionFactory();
			Session session=sf.openSession();
			Transaction tx=session.beginTransaction();
			session.update(object);
			tx.commit();
			session.close();
		}catch(Exception e){
			e.printStackTrace();
		}
	}

	public static void delete(Class<Candidate> cls, Serializable s) {
		try{
			SessionFactory sf=AHibernateUtil.getSessionFactory();
			Session session=sf.openSession();
			Transaction tx=session.beginTransaction();
			Object o=session.load(cls,s);
			session.delete(o);
```

```
				tx.commit();
				session.close();
			}catch(Exception e){
				e.printStackTrace();
			}
		}


		public static Object load(Class<Candidate> cls, Serializable s) {
			Object o=null;
			try{
				SessionFactory sf=AHibernateUtil.getSessionFactory();
				Session session=sf.openSession();
				Transaction tx=session.beginTransaction();
				o=session.load(cls,s);
				tx.commit();
				session.close();
			}catch(Exception e){
				e.printStackTrace();
			}
			return o;
		}
}
```

DAOFactory.java

```
package hibernate;

public class DAOFactory {
	static CandidateDAO cdao=null;

	static{
		cdao=new HibernateCandidateDAO();
	}

	public static CandidateDAO getCandidateDAO() {
		return cdao;
	}
}
```

Hibernate.hbm.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="connection.driver_class">com.mysql.jdbc.Driver</property>
		<property name="connection.url">jdbc:mysql://localhost/DAO_db</property>
		<property name="connection.username">root</property>
		<property name="connection.password">Westpac1</property>
		<property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

```
			<property name="show_sql">true</property>
			<property name="hbm2ddl.auto">update</property>

			<mapping class="hibernate.Candidate"/>
			<mapping class="hibernate.FreshCandidate"/>
			<mapping class="hibernate.ExpCandidate"/>

		</session-factory>
</hibernate-configuration>
```

HibernateCandidateDAO.java

```
package hibernate;

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

import javax.transaction.Transaction;

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

public class HibernateCandidateDAO implements CandidateDAO {

	@Override
	public void addCandidate(Candidate can) {
		AHibernateTemplate.save(can);
	}

	@Override
	public void updateCandidate(Candidate can) {
		AHibernateTemplate.update(can);
	}

	@Override
	public void deleteCandidate(int cid) {
		AHibernateTemplate.delete(Candidate.class,cid);
	}

	@Override
	public Candidate getCandidateByCid(int cid) {
		Candidate can=(Candidate)AHibernateTemplate.load(Candidate.class,cid);
		return can;
	}

	@Override
	public List<Candidate> getAllCandidates() {
		List<Candidate> cans=new ArrayList<Candidate>();
		try{
			SessionFactory sf=AHibernateUtil.getSessionFactory();
			Session session=sf.openSession();
			Transaction tx=(Transaction) session.beginTransaction();
			String HQL="from Candidate can";
			Query q=session.createQuery(HQL);
```

```
				cans=q.list();
				tx.commit();
				session.close();
			}catch(Exception e){
				e.printStackTrace();
			}
			return cans;
		}

}
```

Lab07Client.java

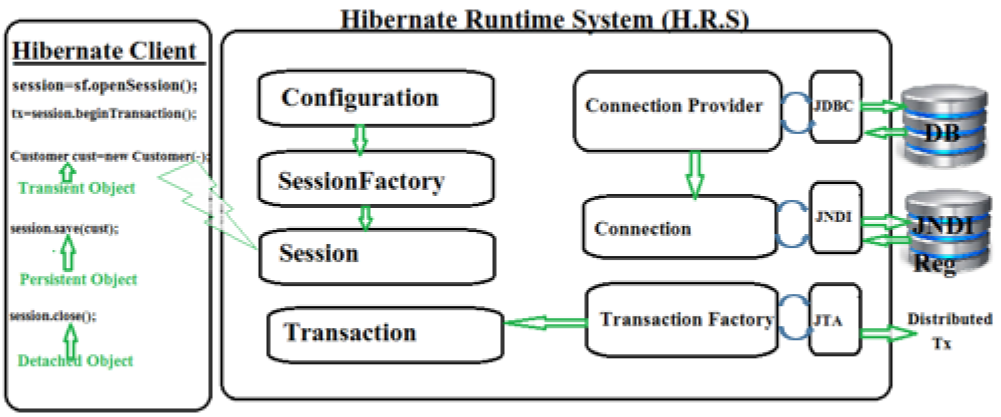
```
package hibernate;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class Lab07Client {
	public static void main(String[] args) {
		CandidateDAO candidateDAO=DAOFactory.getCandidateDAO();

		//1. Add Fresh Candidate
		List<String> emails=new ArrayList<String>();
			emails.add("Ankur@hotmail.com");
			emails.add("ankit@gmail.com");
			emails.add("Anshu@yahoo.com");

		Map<String, Double> pers=new HashMap<String, Double>();
			pers.put("MCA",new Double(99.99));
			pers.put("B.Sc", 88.90);

		FreshCandidate freshCandidate=new FreshCandidate("Ankur", "Shrivastav",
emails, "MCA", "15-Sep-1988", 2013, pers,"Yes");
		candidateDAO.addCandidate(freshCandidate);

		// 2. Adding ExpCandidate
		Set<String> skills=new HashSet<String>();
			skills.add("Java");
			skills.add("EJB");
			skills.add("Spring");
			skills.add("Hibernate");

		List<String> companies=new ArrayList<String>();
			companies.add("IBM");
			companies.add("TCS");
			companies.add("Wipro");

		ExpCandidate expCandidate=new ExpCandidate("Anshu","Shrivastava",
emails, "M.A.","15-Jun-1986",2010, skills, companies,9.0);
		candidateDAO.addCandidate(expCandidate);
```

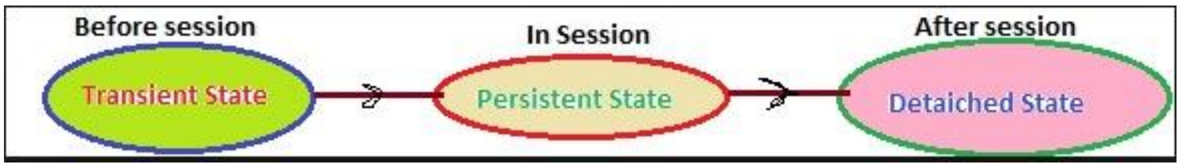
```
/*
             // 3. Loading FreshCandidate
             FreshCandidate freshCandidate1=(FreshCandidate)
candidateDAO.getCandidateByCid(1);
                   System.out.println(freshCandidate1.getCid());
                   System.out.println(freshCandidate1.getFname());
                   System.out.println(freshCandidate1.getLname());
                   System.out.println(freshCandidate1.getQualification());
                   System.out.println(freshCandidate1.getEmails());
                   System.out.println(freshCandidate1.getDob());
                   System.out.println(freshCandidate1.getYop());
                   System.out.println(freshCandidate1.getYearGaps());
                   System.out.println(freshCandidate1.getPercentages());

             // 4. Loading ExpCandidate
             ExpCandidate expCandidate1=(ExpCandidate)
candidateDAO.getCandidateByCid(2);
                   System.out.println(expCandidate1.getCid());
                   System.out.println(expCandidate1.getFname());
                   System.out.println(expCandidate1.getLname());
                   System.out.println(expCandidate1.getQualification());
                   System.out.println(expCandidate1.getEmails());
                   System.out.println(expCandidate1.getDob());
                   System.out.println(expCandidate1.getYoe());
                   System.out.println(expCandidate1.getCompanies());
                   System.out.println(expCandidate1.getCtc());
                   System.out.println(expCandidate1.getSkills());

             // 5. Deleting FreshCandidate
                   candidateDAO.deleteCandidate(7);

             //6. Deleting ExpCandidate
                   candidateDAO.deleteCandidate(8);

             // 7. Updating FreshCandidate
             FreshCandidate freshCandidate2=(FreshCandidate)
candidateDAO.getCandidateByCid(1);
                   ((Candidate) candidateDAO).setFname("Vishnu");
                   ((Candidate) candidateDAO).setLname("Lakshmi");

                   // Similarly Call other setter methods to update any Value
                   candidateDAO.updateCandidate(freshCandidate2);

             // 8. Updating ExpCandidate
             ExpCandidate expCandidate2=(ExpCandidate)
candidateDAO.getCandidateByCid(2);
                   ((Candidate) candidateDAO).setFname("Komal");
                   ((Candidate) candidateDAO).setFname("Kali");

                   // Similarly Call other setter methods to update any Value
                   candidateDAO.updateCandidate(expCandidate2);

             // 9. Loading All Candidate
             List<Candidate> candidates=candidateDAO.getAllCandidates();
             for(Candidate candidate:candidates){

      System.out.println(candidate.getCid()+"\t"+candidate.getFname()+"\t"+candidate
.getLname()+"\t"+candidate.getQualification());
             }

*/
      }
}
```

Hibernate Architecture

Client Code:

```
             org.hibernate.Transaction tx=null;
             try{
                1.  Configuration cfg=new Configuration();
                2.  cfg.configure();
                3.  SessionFactory factory=cfg.buildSessionFactory();
                4.  Session session=factory.openSession();
                5.  tx=session.beginTransaction();
                6.  Customer cust=new Customer("Ankur","ankur.as88#gmail.com",88888,"Pune");
                    session.save(cust);
                7.  tx.commit();
                8.  session.close();
             }catch(Exception e){
                    e.printStackTrace();
                9.  tx.rollback();

             }
```





1. *Configuration or AnnotationConfiguration*

1) These are available in **org.hibernate.cfg** package.
2) This is the first class that will be instantiated by the hibernate application.
3) You can use configuration or AnnotationConfiguration class Object for 2 tasks-
   a. Calling configure() or configure(String) method.
   b. Calling buildSessionFactory() method.
4) configure() method is responsible for-
   a) Identifying cfg doc.
   b) Reading the data from Hibernate Configuration Document.
   c) Identifies all the mapping Resources or Mapping classes.
   d) Reading the data from all the Hibernate Mapping documents or Annotations specified in persistence.
   e) Initializing Configuration object with the data taken from all the xml's or all the annotations.

5) buildSessionFactory() method is responsible for creating SessionFactoryObject.
6) Configuration or AnnotationConfiguration object is Single Threaded and Short lived.
7) Once SessionFactory object is created then there is no use with Configuration or AnnotationConfiguration object.

2. *SessionFactory*

1) This interface is available in **org.hibenate** package.
2) Session Factory object Multithreaded and long lived.
3) When you call buildSessionFactory() method on configuration object then following tasks will happen:
   a) Set the default to many parameters like Batch size, fetch size, auto commit, etc.
   b) Generates the caches the SQL Queries required.
   c) Generates and execute the Table creation statement.
   d) Select the connection provider.
   e) Select the TransactionFactory etc.
4) SessionFactory is-
   a) Factory of Session objects.
   b) Client for Connection Provider.
   c) Client for TransactionFactory.
5) You need to create One SessionFactory per database.
6) When you are using multiple databases then you need to write multiple hibernate configuration docs.

7) Hibernate for one database:

a) 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="connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="connection.url">jdbc:mysql://localhost/db1</property>
      <property name="connection.username">root</property>
      <property name="connection.password">Westpac1</property>
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
      <property name="show_sql">true</property>
      <property name="hbm2ddl.auto">update</property>
      <mapping class="hibernate.Student"/>
   </session-factory>
</hibernate-configuration>
```

b) HibernateUtil.java

```
package com.jlcindia.hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class AHibernateUtil {
        static SessionFactory sessionFactory;
        static{
                AnnotationConfiguration cfg=new AnnotationConfiguration();
                cfg=(AnnotationConfiguration) cfg.configure("hibernate.cfg.xml");
                sessionFactory=cfg.buildSessionFactory();
        }
        public static SessionFactory getSessionFactory(){
                return sessionFactory;
        }
}
```

8) HibernateUtil for 2 database:

Oracle.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="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
      <property name="connection.url">jdbc:oracle:thin:@localhost:1521:XE</property>
      <property name="connection.username">system</property>
      <property name="connection.password">Westpac1</property>
      <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
      <property name="show_sql">true</property>
      <property name="hbm2ddl.auto">update</property>
      <mapping class="hibernate.Student"/>
</session-factory>
</hibernate-configuration>
```

mysql.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="connection.driver_class">com.mysql.jdbc.Driver</property>
       <property name="connection.url">jdbc:mysql://localhost/db1</property>
       <property name="connection.username">root</property>
       <property name="connection.password">Westpac1</property>
       <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
       <property name="show_sql">true</property>
       <property name="hbm2ddl.auto">update</property>
       <mapping class="hibernate.Student"/>
   </session-factory>
</hibernate-configuration>
```

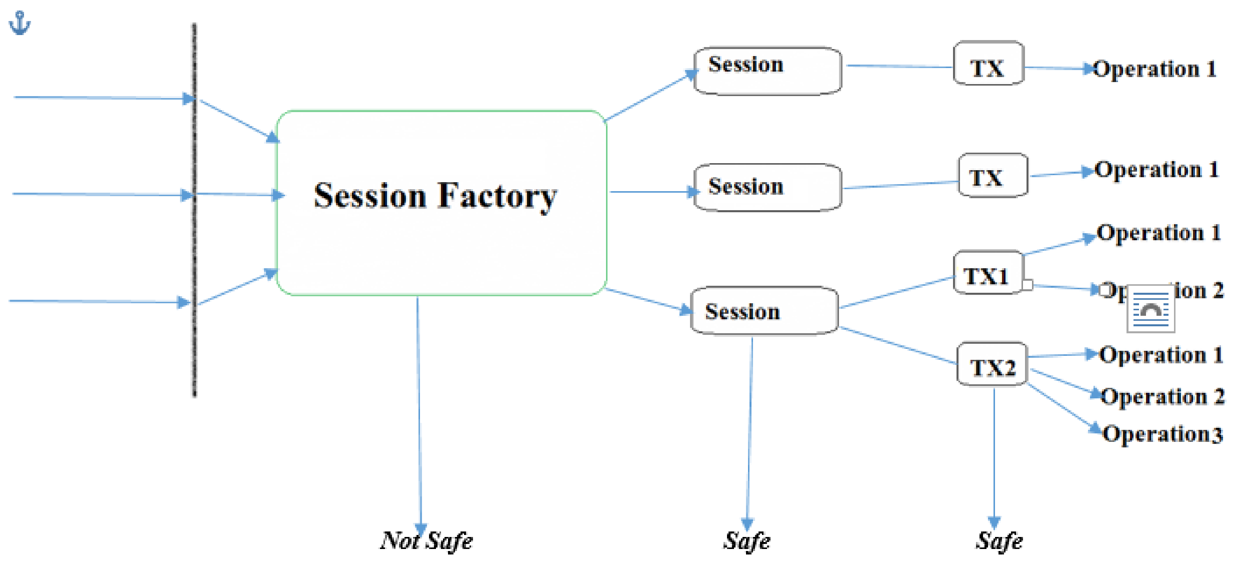
AHibernateUtil.java:
```
package com.jlcindia.hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class AHibernateUtil {
        static SessionFactory oraFactory=null;
        static SessionFactory myFactory=null;
        static{
                AnnotationConfiguration cfg1=new AnnotationConfiguration();
                cfg1=(AnnotationConfiguration) cfg1.configure("oracle.cfg.xml");
                oraFactory=cfg1.buildSessionFactory();

                AnnotationConfiguration cfg2=new AnnotationConfiguration();
                cfg2=(AnnotationConfiguration) cfg2.configure("mysql.cfg.xml");
                myFactory=cfg2.buildSessionFactory();

        }
        public static SessionFactory getSessionFactory(int x){
                if(x==1){
                        return oraFactory;
                }else {
                        return myFactory;
                }
        }
}
```

Client.java:
```
                org.hibernate.Transaction tx=null;
                try{
                        SessionFactory sf=AHibernateUtil.getSessionFactory(1);
                        Session session=sf.openSession();
                        tx=session.beginTransaction();
                        Student stu=new Student("Ankur","MCA", courses, emails, marks, phones, refs);
```

```
            session.save(stu);
            tx.commit();
            session.close();
    }catch(Exception e){
            e.printStackTrace();
    }
```

Usage:-

```
AHibernateUtil.getSessionFactory(1);
AHibernateUtil.getSessionFactory(0);
```



3. Session

1) This interface is available in **org.hibernate** package.
2) Session object is single Threaded and short lived.
3) Session represents period of time where you can do multiple database operation.
4) Session Object-
    a) Uses TransactionFactory to get the connection.
    b) Get the connection from Connection Provider.

4. Transaction

When Transaction is started, then following tasks will happen:

1) Session cache will be created.
2) Connection will be taken and will be associated with the current session.
3) While transaction is running, following tasks will happen.
    1. When any object is participated in Session Operation that will be placed in session Cache.
    2. When Transaction is committed then following tasks will happen-
        a) Session will be flushed.
        b) Session cache will be destroyed.

c) Commit will be issues to database.
d) Connection will be released.

Types of Object States

Persistence class object can be found in 3 states-

1. Transient State
2. Persistent State
3. Detached State

Transient State

- ❖ When Persistence class object is newly created and not participated any session operations then that object is called as Transient Object.
- ❖ State of that object is called Transient State.
- ❖ Transient Object does not contain any identity or P.K. (Primary Key)

Persistent State

- ❖ When persistent class object is participated any session operations then that Object is called as Persistent Object.
- ❖ State of that object is called Persistent State.
- ❖ Persistent Object contains any identity or P.K.

*** Any modification happened on persistent object will be reflected to Database ***

Detached State

- ❖ When persistent class Object is participated any session operation and removed from session cache then that object is called as Detached Object.
- ❖ State of that object contains any identity or P.K.

*** Any modification happened on Detached object will not be reflected to Database ***

Hibernate Eager vs. Lazy Fetch Type

Joining two tables in SQL is the foundation of a relational database, as joins allow you to actually define relationships between tables (objects).

Having said that, relationships are important to understand when talking about fetch types in Hibernate. This is the case because whenever you define a relationship in Hibernate, you'll also need to define the fetch type. The fetch type essentially decides whether or not to load all of the relationships of a particular object/table as soon as the object/table is initially fetched.

An example of this would be as follows, consider this `User` object:

```
public class User{
  import javax.persistence.OneToOne;
```

```
 import javax.persistence.JoinColumn;

 private String username;
 private String password;
 private Profile userProfile;

 // omitting code for getters and setters for username, password
 @OneToOne
 @JoinColumn(name="user_profile_id")
 private Profile getUserProfile() {
   return userProfile;
 }

 private void setUserProfile(Profile userProfile) {
   this.userProfile = userProfile;
 }
}
```

Any time you see a `@OneToOne`, `@OneToMany` or `@ManyToMany` annotations, you've got a relationship. What's important to note is that the fetch type should be specified within those annotations, if you don't specify one then it defaults to `FetchType.LAZY`.

Fetch type Eager is essentially the opposite of Lazy, Eager will by default load ALL of the relationships related to a particular object loaded by Hibernate. This means that if you change the relationship to be this:

```
 import javax.persistence.FetchType;
 //....
 //....
 //....

 @OneToOne(fetch=FetchType.EAGER)
 @JoinColumn(name="user_profile_id")
 private Profile getUserProfile() {
           return userProfile;
 }
```

Hibernate will now load the user profile into the user object by default. This means that if you access `user.getUserProfile()` it won't be `NULL` (unless the joining table actually doesn't contain a matching row by the "user_profile_id" join column).

So, the long story short here is:

`FetchType.LAZY` = Doesn't load the relationships unless explicitly "asked for" via getter

`FetchType.EAGER` = Loads ALL relationships

Let's assume we had a `User` object that has 10 different relationships to other objects. Let's also say that you have a user that's trying to login and you just want to check to see if their username and password is correct and matches what's stored in the database. Would you really want to load ALL 10 relationships and NOT make use of ANY of those loaded relationships? Remember, loading things from a database is fairly "expensive" in terms of the time it takes to load all that information into a Java objects AND keep them in memory. And what happens if one of the relationships points to an object that ALSO has 10 relationships (and all of those are set up as fetch type EAGER). Think of the cascading massacre of objects loaded from the database into memory!
The way I like to put it is like this:

- **EAGER**: Convenient, but slow
- **LAZY**: More coding, but much more efficient

view plaincopy to clipboardprint?

```java
/**
* This UserDao object is being constructed under the assumption that the relationship
*  to the Profile object is being Lazily loaded using FetchType.LAZY
*/
@Repository
@Transactional
public class UserDao
{
  @Autowired
  private SessionFactory sessionFactory;

  public User getUserById(Long userId)
  {
    Session session = sessionFactory.getCurrentSession();

    // this will invoke the
    return (User) session.createCriteria(User.class).add(Restrictions.idEq(userId)).uniqueResult();
  }

  public User getUserWithProfileById(Long userId)
  {
    Session session = sessionFactory.getCurrentSession();

    User user = (Users) session.createCriteria(User.class).add(Restrictions.idEq(userId)).uniqueResult();
    // this will force SQL to execute the query that will join with the user's profile and populate
    //  the appropriate information into the user object.
    Hibernate.initialize(user.getUserProfile());

    return user;
  }
}
```

In the code above you see that we have created two getter methods to load the `User` object from our database via Hibernate.
This means that when you want to load a `User` object without any relationships (i.e. when you're checking to see if the login information is valid) then you can call `getUserById` and this will bypass the unnecessary join to the `Profile` object/table. But if you are in a situation where you DO need to refer to the `User`'s profile, then you can call the `getUserWithProfileById`.

Lazy/Eager Loading Using Hibernate by Example

You are a parent who has a kid with a lot of toys. But the current issue is whenever you call him (we assume you have a boy), he comes to you with all his toys as well. Now this is an issue since you do not want him carrying around his toys all the time.

So being the rationale parent, you go right ahead and define the toys of the child as LAZY. Now whenever you call him, he just comes to you without his toys.

But you are faced with another issue. When the time comes for a family trip, you want him to bring along his toys because the kid will be bored with the trip otherwise. But since you strictly enforced LAZY on the child's toy, you are unable to ask him to bring along the toys. This is where EAGER fetching comes into play. Let us first see our domain classes.

```
package com.fetchsample.example.domain;
import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;
/**
 * Holds information about the child
 *
 * @author dinuka.arseculeratne
 *
 */
@Entity
@Table(name = "CHILD")
@NamedQuery(name = "findChildByName", query = "select DISTINCT(chd) from Chil
d chd left join fetch chd.toyList where chd.childName=:chdName")
public class Child {
 public static interface Constants {
  public static final String FIND_CHILD_BY_NAME_QUERY = "findChildByName";
  public static final String CHILD_NAME_PARAM = "chdName";
 }
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 /**
  * The primary key of the CHILD table
  */
 private Long childId;
 @Column(name = "CHILD_NAME")
 /**
  * The name of the child
  */
 private String childName;
 @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
 /**
  * The toys the child has. We do not want the child to have the same toy mor
e than
  * once, so we have used a set here.
  */
 private Set<Toy> toyList = new HashSet<Toy>();
 public Long getChildId() {
  return childId;
 }
 public void setChildId(Long childId) {
  this.childId = childId;
 }
 public String getChildName() {
  return childName;
 }
 public void setChildName(String childName) {
  this.childName = childName;
 }
 public Set<Toy> getToyList() {
  return toyList;
 }
 public void addToy(Toy toy) {
  toyList.add(toy);
 }
}
```

```
package com.fetchsample.example.domain;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

/**

 * Hols data related to the toys a child possess

 *

 * @author dinuka.arseculeratne

 *

 */

@Entity

@Table(name = "TOYS")

public class Toy {

 @Id

 @GeneratedValue(strategy = GenerationType.AUTO)

 @Column(name = "TOY_ID")

 /**

  * The primary key of the TOYS table

  */

 private Long toyId;

 @Column(name = "TOY_NAME")

 /**

  * The name of the toy
  */
 private String toyName;
 public Long getToyId() {
  return toyId;
 }
 public void setToyId(Long toyId) {
  this.toyId = toyId;
 }
 public String getToyName() {
  return toyName;
 }
 public void setToyName(String toyName) {
  this.toyName = toyName;
 }
 @Override
 public int hashCode() {
  final int prime = 31;
  int result = 1;
  result = prime * result + ((toyName == null) ? 0 : toyName.hashCode());
  return result;
 }
 @Override
 /**
  * Overriden because within the child class we use a Set to
  * hold all unique toys
  */
 public boolean equals(Object obj) {
  if (this == obj)
   return true;
  if (obj == null)
   return false;
  if (getClass() != obj.getClass())
   return false;
  Toy other = (Toy) obj;
  if (toyName == null) {
   if (other.toyName != null)
    return false;
  } else if (!toyName.equals(other.toyName))
   return false;
  return true;
 }
 @Override
 public String toString() {
  return "Toy [toyId=" + toyId + ", toyName=" + toyName + "]";
 }
}
```

So as you can see, we have two simple entities representing the child and toy. The child has a one-to-many relationship with the toys which means one child can have many toys (oh how I miss my childhood days). Afterwards we need to interact with the data, so let us go ahead and define out DAO(Data access object) interface and implementation.

```
package com.fetchsample.example.dao;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;
import com.fetchsample.example.domain.Child;
/**
 * The basic contract for dealing with the {@link Child} entity
 *
 * @author dinuka.arseculeratne
 *
 */
@Transactional(propagation = Propagation.REQUIRED, readOnly = false)
public interface ChildDAO {
 /**
  * This method will create a new instance of a child in the child table
  *
  * @param child
  *            the entity to be persisted
  */
 public void persistChild(Child child);
 /**
  * Retrieves a child without his/her toys
  *
  * @param childId
  *            the primary key of the child table
  * @return the child with the ID passed in if found
  */
 public Child getChildByIdWithoutToys(Long childId);
 /**
  * Retrieves the child with his/her toys
  *
  * @param childId
  *            the primary key of the child table
  * @return the child with the ID passed in if found
  */
 public Child getChildByIdWithToys(Long childId);
 /**
  * Retrieves the child by the name and with his/her toys
  *
  * @param childName
  *            the name of the child
  * @return the child entity that matches the name passed in
  */
 public Child getChildByNameWithToys(String childName);
}
```

```
package com.fetchsample.example.dao.hibernate;

import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

import com.fetchsample.example.dao.ChildDAO;

import com.fetchsample.example.domain.Child;

/**

 * The hibernate implementation of our {@link ChildDAO} interface

 *

 * @author dinuka.arseculeratne

 *
 */
public class ChildDAOHibernateImpl extends HibernateDaoSupport implements
  ChildDAO {
 /**
  * {@inheritDoc}
  */
 public void persistChild(Child child) {
  getHibernateTemplate().persist(child);
 }
 /**
  * {@inheritDoc}
  */
 public Child getChildByIdWithoutToys(Long childId) {
  return getHibernateTemplate().get(Child.class, childId);
 }
 /**
  * {@inheritDoc}
  */
 public Child getChildByIdWithToys(Long childId) {
  Child child = getChildByIdWithoutToys(childId);
  /**
   * Since by default the toys are not loaded, we call the hibernate
   * template's initialize method to populate the toys list of that
   * respective child.
   */
  getHibernateTemplate().initialize(child.getToyList());
  return child;
 }
 /**
  * {@inheritDoc}
  */
 public Child getChildByNameWithToys(String childName) {

  return (Child) getHibernateTemplate().findByNamedQueryAndNamedParam(

    Child.Constants.FIND_CHILD_BY_NAME_QUERY,

    Child.Constants.CHILD_NAME_PARAM, childName).get(0);

 }

}
```

A simple contract. I have four main methods. The first one of course just persists a child entity to the database.

The second method retrieves the Child by the primary key passed in, but does not fetch the toys.

The third method first fetches the Child and then retrieves the Child's toys using the Hibernate template's initialize() method. Note that when you call the initialize() method, hibernate will fetch you LAZY defined collection to the Child proxy you retrieved.

The final method also retrieves the Child's toys, but this time using a named query. If we go back to the Child entity's Named query, you can see that we have used "left join **fetch**". It is the keyword**fetch** that actually will initialize the toys collection as well when returning the Child entity that qualifies. Finally i have my main class to test our functionality;

```
package com.fetchsample.example;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.fetchsample.example.dao.ChildDAO;
import com.fetchsample.example.domain.Child;
import com.fetchsample.example.domain.Toy;
/**
 * A test class
 *
 * @author dinuka.arseculeratne
 *
 */
```

```java
public class ChildTest {
 public static void main(String[] args) {
  ApplicationContext context = new ClassPathXmlApplicationContext(
    "com/fetchsample/example/spring-context.xml");
  /**
   * First we initialize a child
   */
  Child child = new Child();
  /**
   * A cool ben 10 action figure
   */
  Toy ben10 = new Toy();
  ben10.setToyName("Ben 10 figure");
  /**
   * A even more cooler spider man action figure
   */
  Toy spiderMan = new Toy();
  spiderMan.setToyName("Spider man figure");
  child.setChildName("John");
  /**
   * Add the toys to the collection
   */
  child.addToy(ben10);
  child.addToy(spiderMan);
  ChildDAO childDAO = (ChildDAO) context.getBean("childDAO");
  childDAO.persistChild(child);
  Child childWithoutToys = childDAO.getChildByIdWithoutToys(1L);
  // The following line will throw a lazy initialization error since we have
  // defined fetch type as LAZY in the Child domain class.
  // System.out.println(childWithToys.getToyList().size());
  Child childWithToys = childDAO.getChildByIdWithToys(1L);
  System.out.println(childWithToys.getToyList().size());
  Child childByNameWithToys = childDAO.getChildByNameWithToys("John");
  System.out.println(childByNameWithToys.getToyList().size());
 }
}
```

Working with Primary Key

Hibernate Supports to configure both the types of Primary Keys-

1. Single Primary Keys
2. Composite Primary Keys

Hibernate Query Language

4 methods of session for doing CURD Operations:

```
session.save();
session.update();
session.load();
session.delete();
```

4 methods of EntityManager for doing CURD Operations:

```
entityManager.persist();
entityManager.merge();
entityManager.find();
entityManager.remove();
```

Hibernate supports various Query languages to select the records based on various criteria's-

1. HQL (Hibernate Query Language) *****
2. QBC (Query by Criteria) *****
3. Native Queries.
4. Named Queries.

Consider the table called **students** which is mapped with persistence class called **Student**.

Questions:

1. **Display all the students?**

   SQL:-
   ```
   select sid, sname, email from students stu;
   ```

   HQL:-
   ```
   String hql=”from Student stu”;
   Query q=session.createQuery(hql);
   List<Student> list=q.list();
   ```

2. **Display all the students belongs to Bangalore?**

   SQL:-
   ```
   select * from students where branch=?;
   ```

HQL:-

```
String hql="from Student stu where stu.branch=?" ;
Query q=session.createQuery(hql);
q.setString(0, "Bangalore");
List<Student> list=q.list();
```

1) **Display all the Customers?**
   HQL:-

```
String hql="from Customer c";
Query q=session.createQuery(hql);
List<Customer> list=q.list();
for(Customer c:list){
        System.out.println(c);
}
```

2) **Display all the Customers staying in Blore.**
3) **HQL:-**

```
String hql="from Customer c where c.city=?";
Query q=session.createQuery(hql);
q=q.setString(0,"Blore");
List<Customer> list=q.list();
for(Customer c:list){
        System.out.println(c);
}
```

4) **Display all the Customers staying in Blore with bal>10,000**
   HQL:-

```
String hql="from Customer c where c.city=? and c.cardBal>?";
Query q=session.createQuery(hql);
q=q.setparameter(0,"Blore");
q=q.setparameter(1,10000.0);
list=q.list();
```

5) **Display all the Customers with Visa card, bal between 10k and 30k and status- true.**
   HQL:-

```
        String hql="from Customer c where c.cardType=? and c.cardBal between ? and ?
and c.status=?";
        Query q=session.createQuery(hql);
                q.setparameter(0,"Visa");
                q.setparameter(1,10000.0);
                q.setparameter(2,30000.0);
                q.setparameter(3,true);
                list=q.list();
```

6) **Display all the Customers who are in Blore, Pune, Hyd**
   HQL:-

```
String hql=”from Customer c where c.city in (?,?,?)”;
Query q=session.createQuery(hql);
        q.setparameter(0,”Blore”);
        q.setparameter(1,”Pune”);
        q.setparameter(2,”Hyd”);
        list=q.list();
```

7) **Display all the Customers who are in Blore in desc order by cname**
   HQL:

```
String hql=”from Customer c where c.city=? order by c.name desc”;
Query q=session.createQuery(hql);
q=q.setparameter(0,”Blore”);
list=q.list();
```

8) **Display all the Customers whose email is CareCentrix email ascending order by email**
   HQL:

```
String hql=”from Customer c where c.email like ? order by c.email”;
Query q=session.createQuery(hql);
q=q.setparameter(0,”%CareCentrix.com”);
list=q.list();
```

9) **Display Nth highest Customer.**
   HQL:

```
Object getNthHighestCustomer(int n){
        String hql=”select distinct * from Customer c where order by c.cardBal ”;
        Query q=session.createQuery(hql);
        list=q.list();
        Object o=list.get(n-1);
        return o;
}
```

Pagination

10) **Display all the Customers staying in Blore.**
   HQL:

```
getCustomersByCity(String ci, int start, int total){
        String hql=”from Customer c where c.city=?”;
        Query q=session.createQuery(hql);
        q=q.setString(0,ci);
        q=q.setFirstResult(start);
        q=q.setMaxResult(total);
        List<Customer> list=q.list();
    }
```

Polymorphic Queries

1) **Display the Regular Students**
   HQL:

```
String hql="from RegularStudent stu";
Query q=session.createQuery(hql);
List<RegularStudent> list=q.list();
```

2) **Display all types of Students**
   HQL:

```
String hql="from RegularStudent stu";
Query q=session.createQuery(hql);
List<Student> list=q.list();
for(Student s:list){
        if(s instance of WeekendStudent)
                RegularStudent rs=( RegularStudent)s;
        else if(s instance of WeekendStudent)
                WeekendStudent ws=( WeekendStudent)s;
}
```

3) **Display all records from all the tables**
   HQL:

```
String hql="from Object obj";
```

4) **Display all Serializable**
   HQL:

```
String hql="from Serializable s";
```

Parameter Types

1- Positional Parameters
2- Named Parameters

1. Positional Parameters

Example 1:

```
String hql="from Customer c where c.city=?";
Query q=session.createQuery(hql);
q=q.setString(0,"Blore");
```

Example 2:

```
String hql="from Customer c where c.city=? and c.cardBal>?";
Query q=session.createQuery(hql);
q=q.setParameter(0,"Blore");
```

```
q=q.setParameter(1,10000.0);
list=q.list();
```

2. Positional Parameters

Example 1:

```
String hql=”from Customer c where c.city=:cit”;
Query q=session.createQuery(hql);
q=q.setString(0,”Blore”);
```

Example 2:

```
String hql=”from Customer c where c.city=:cit and c.cardBal>:cbal”;
Query q=session.createQuery(hql);
q=q.setParameter(0,”Blore”);
q=q.setParameter(1,10000.0);
list=q.list();
```

Hibernate Transaction Management

Transaction is an interface available in org.hibernate package and has the following concrete implementations.

1) JDBCTransaction
2) JTATransaction
3) CMTTransaction

TransactionFactory is an interface available in org.hibernate.transaction package and has the following concrete implementation.

1) JDBCTransactionFactory
2) JTATransactionFactory
3) CMTTransactionFactory

These TransactionFactory implementation classes are responsible for providing the corresponding Transaction depending on the properties written by you in **hibernate.cfg.xml**.
i.e.

1) JDBCTransactionFactory provides the JDBCTransaction
2) JTATransactionFactory provides the JTATransaction
3) CMTTransactionFactory provides the CMTTransaction

JDBCTransactions

JTATransactions

CMTTransactions