

TASK 11

HIBERNATE FRAMEWORK

11.1

Aim:-

To Demonstrate the use of Hibernate to provide Java Persistence in a Standalone application for User Details Database by creating a UserDetails POJO class, userTable table, a Mapping Configuration File, and an application class with the main() method to run the application. Also to develop the application class to be used to save few Users' records and then apply CRUD (Create, read, update and delete) operations.

Algorithm:-

- Step 1. Start
- Step 2. Create the Persistent class - UserDetails.java
- Step 3. Create the mapping file for Persistent class - UserDetails.hbm.xml
- Step 4. Create the Configuration file - hibernate.cfg.xml
- Step 5. Create the class that retrieves or stores the persistent object - HibernateTestDemo.java
- Step 6. Load the required jar files for hibernate
- Step 7. Run the Hibernate Application by right-clicking on the project > Run
As > Java Application and run the application
- Step 8. Stop

Program:-

UserDetails.java

```
package com.hibernate;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class UserDetails
```

```

{
    @Id
    private int    userId;
    private String userName;
    private String email;

    public UserDetails() {
    }

    public UserDetails(String userName, String email) {
        this.userName = userName;
        this.email = email;
    }

    public int getUserId() {
        return userId;
    }

    public void setUserId(int userId) {
        this.userId = userId;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}

```

UserDetails.hbm.xml:

```

import java.util.Scanner;
class InvestmentCalculator
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter present value: ");
        double p=sc.nextInt();
        System.out.print("Enter the interest rate: ");
        double r=sc.nextInt();
        System.out.print("Enter the time period in years: ");
        double y=sc.nextInt();
        double f=p*Math.pow((1+r/100),y);
        System.out.print("Future Investment value is: "+f);
    }
}

```

HibernateTestDemo.java

```

package com.hibernate;

import java.io.Serializable;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

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

        //apply configuration property settings to
        StandardServiceRegistryBuilder
        SessionFactory sessionFactory = new
        Configuration().configure().buildSessionFactory();

        // opens a new session from the session factory
        Session session = sessionFactory.openSession();
        System.out.println("Hibernate Configuration loaded");
        System.out.println("Hibernate serviceRegistry created");

        session.beginTransaction();

        /*
        persist() -> This method is used to save an entity object into database
        and return a void.
        *if an entity already exists in the database the It will throw an
        exception
        */
        UserDetails user1 = new UserDetails("John","John@gmail.com");
        session.persist(user1);
        System.out.println("User 1 is created");
        /*
        save() -> This method is used to save an entity object into database and
        return generated primary key.
        If an entity already exists in the database then It will throw an
        exception.
        */
        UserDetails user2 = new UserDetails("Mark","Mark@gmail.com");
        Serializable id1 = session.save(user2);
        System.out.println("User is created with Id::"+id1);

        //saveOrUpdate()->This method is basically used to either save or update an
        entity in the database.
        UserDetails user3 = new UserDetails("Tom","Tom@gmail.com");
        session.saveOrUpdate(user3);

        try {

            UserDetails user4 = (UserDetails) session.load(UserDetails.class,
2);
            //new UserDetails("Ann","Ann@gmail.com");
            if(user4 != null)
            {
                user4.setEmail("USER@gmail.com");

```

```

        session.update(user4);
        System.out.println("User updated with email:
"+user4.getEmail());
    }
    else{
        System.out.println("User doesn't exist!!");
    }

} catch (HibernateException e) {
    e.printStackTrace();
}

// Delete an object from database
//first create the object in database and then delete it.
UserDetails userForDelete = new
UserDetails("Mary", "Mary@gmail.com");
Serializable id2 = session.save(userForDelete);
System.out.println("User is created with Id:"+id2);
System.out.println("This User "+userForDelete.getUserName()+" will be
deleted from Database now");
session.delete(userForDelete);
//Use the session to save model objects
session.getTransaction().commit();
session.close();
}
}

```

hibernate.cfg.xml

```

<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="connection.url">jdbc:mysql://localhost:3306/UserDb1</property>
        <property name="connection.username">root</property>
        <property name="connection.password">admin</property>
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <property name="show_sql">true</property>

        <mapping resource="com/hibernate/UserDetails.hbm.xml"/>

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

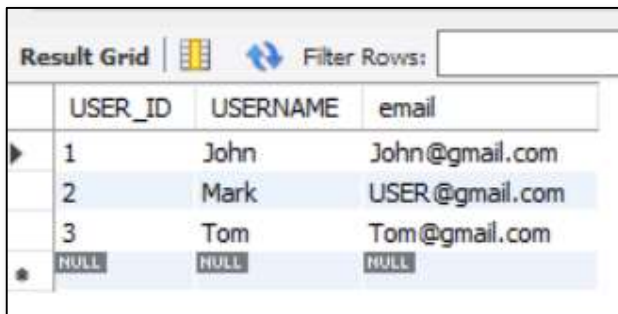
```

Output:-

```

Hibernate Configuration loaded
Hibernate serviceRegistry created
Hibernate: select max(USER_ID) from UserTable1
User 1 is created
User is created with Id::2
User updated with email: USER@gmail.com
User is created with Id:4
This User Mary will be deleted from Database now
Hibernate: insert into UserTable1 (USERNAME, email, USER_ID) values (?, ?, ?)
Hibernate: insert into UserTable1 (USERNAME, email, USER_ID) values (?, ?, ?)
Hibernate: insert into UserTable1 (USERNAME, email, USER_ID) values (?, ?, ?)
Hibernate: insert into UserTable1 (USERNAME, email, USER_ID) values (?, ?, ?)
Hibernate: update UserTable1 set USERNAME=?, email=? where USER_ID=?
Hibernate: delete from UserTable1 where USER_ID=?

```

Usertable1 Table:


| | USER_ID | USERNAME | email |
|---|---------|----------|----------------|
| ▶ | 1 | John | John@gmail.com |
| | 2 | Mark | USER@gmail.com |
| | 3 | Tom | Tom@gmail.com |
| ★ | NULL | NULL | NULL |

Result:-

Demonstrate the use of Hibernate to provide Java Persistence in a Standalone application for User Details Database and apply CRUD (Create, read, update and delete) operations.

11.2

Aim:-

To Demonstrate the use of Hibernate to provide Java Persistence in a Web application for UserDB Database by creating Java Server Pages for presentation logic of User Registration, Bean class for representing data and DAO class for database codes.

Algorithm:-

- Step 1. Start
- Step 2. Create index.jsp page to obtain input from the user and sends it to the register.jsp file using post method
- Step 3. Create register.jsp page to obtain all request parameters and stores this information into an object of User class
- Step 4. Create bean class User.java representing the Persistent class in hibernate
- Step 5. Create user.hbm.xml which maps the User class with the Userdetails1 table of the database
- Step 6. Create UserDao.java that contains method to store the instance of User class
- Step 7. Create hibernate.cfg.xml which is a configuration file, containing information about the UserDB2 database and mapping file
- Step 8. Run the application as 'Run on Server'
- Step 9. Stop

Program:-

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

<form action="register.jsp" method="post">
```

```

Name:<input type="text" name="name"/><br><br>
Email:<input type="text" name="email"/><br><br>
Password:<input type="password" name="password"/><br><br>
Phone:<input type="text" name="phone"/><br><br>
<input type="submit" value="register"/>

</form>
</body>
</html>

```

register.jsp

```

<%@page import="com.mypack.UserDao"%>
<jsp:useBean id="obj" class="com.mypack.User">
</jsp:useBean>
<jsp:setProperty property="*" name="obj"/>

<%
UserDao.register(obj);

out.print("You are successfully registered");

%>
</br>
</br>
<a href="index.jsp">Please click to register another user</a>

```

User.java

```

package com.mypack;

public class User {
    private int id;
    private String name,email,password, phone;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
}

```

```

    public void setEmail(String email) {
        this.email = email;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getPassword() {
        return password;
    }
}

```

UserDao.java

```

package com.mypack;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class UserDao {

    public static void register(User u){

        StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure().build();

        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

        SessionFactory factory = meta.getSessionFactoryBuilder().build();
        Session session = factory.openSession();
        Transaction t = session.beginTransaction();

        session.persist(u);

        t.commit();
        session.close();

    }
}

```

user.hbm.xml

```

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

<hibernate-mapping>
    <class name="com.mypack.User" table="userDetails1">
        <id name="id" type="int" column="Id">
            <generator class="increment" />

```



```

    </id>
    <property name="name">
        <column name="name" />
    </property>
    <property name="email">
        <column name="email" />
    </property>
    <property name="password">
        <column name="password" />
    </property>
    <property name="phone">
        <column name="phone" />
    </property>
</class>
</hibernate-mapping>

```

hibernate.cfg.xml

```

<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="connection.url">jdbc:mysql://localhost:3306/UserDb2</property>
        <property name="connection.username">root</property>
        <property name="connection.password">admin</property>
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <property name="show_sql">true</property>

        <mapping resource="com/mypack/user.hbm.xml"/>

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

```

Output:-

User Registration

http://localhost:8080/HibernateWebProject/

Name:

Email:

Password:

Phone:



Console Output:

Hibernate: select max(Id) from userDetails1
 Hibernate: insert into userDetails1 (name, email, password, phone, Id) values (?, ?, ?, ?, ?)

UserDetails1 table:

A screenshot of a database table viewer interface. It shows a table with the following columns: id, name, email, password, and phone. There are two rows of data displayed.

| | id | name | email | password | phone |
|---|----|------------|----------------------|----------|--------|
| ▶ | 1 | Gayetri | user@gmail.com | abcde | 123456 |
| | 2 | Programmer | programmer@gmail.com | aaaaa | 13579 |



Result:-

Demonstrated a Spring Standalone application by creating a basic simple Java Project, then including Spring Framework and common logging API libraries, actual source files that displays a console-based text to the user. Also, the Spring Beans Configuration is performed in a Bean class with class-level annotation - @Configuration in order to display the console text to the user..

11.3

Aim:-

To develop a simple and basic Java program to calculate the Execution time or the CPU time of a program using the method - public static long nanoTime()..

Algorithm:-

- Step 1. Start
- Step 2. Obtain current time before calling the method and assign to 'start' variable.
- Step 3. Invoke () of the RunTimeCalculator display object .
- Step 4. Obtain current time after calling the display() method and assign to 'end' variable
- Step 5. Calculate execution time as: executionTime = end - start;
- Step 6. Print the result to the user.
- Step 7. Stop

Program:-

```
public class RunTimeCalculator {

    public void display() {
        System.out.println("RunTimeCalculator.display()");
    }

    public static void main(String[] args) {
        RunTimeCalculator obj = new RunTimeCalculator();
        // get current time before calling the method
        long start = System.nanoTime();

        obj.display();

        // get current time after calling the method
        long end = System.nanoTime();

        long executionTime = end - start;
        System.out.println("The display() method execution"+
            " time is: " + executionTime + " nanoseconds");
    }
}
```

Output:-

```
RunTimeCalculator.display()
The display() method execution time is: 188700 nanoseconds
```

Result:-

Developed a simple and basic Java program to calculate the Execution time or the CPU time of a program using the method - public static long nanoTime().