Cloud-Lab6

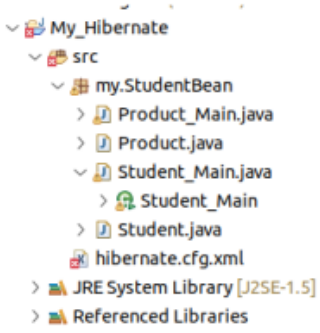
Lab Program - 6

Write a Java application using Hibernate to insert data into Student DATABASE and retrieve info based on particular queries (For example update, delete, search etc...)

create a database as student and database as Student(s is capital in the table name)

```
mysql> desc Student;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| id         | int           | NO   | PRI | NULL    |       |
| usn        | varchar(10)   | YES  |     | NULL    |       |
| address    | varchar(1000) | YES  |     | NULL    |       |
| totalmarks | int           | YES  |     | NULL    |       |
| name       | varchar(100)  | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Project Structure:



Step 1: Set up Maven Project in Eclipse

1. Open Eclipse.
2. Navigate to `File -> New -> Project`.
3. Select `Maven` in the project types and choose `Maven Project`. Click `Next`.
4. Check the option `Create a simple project (skip archetype selection)` and click `Next`.
5. Now, you need to provide the Group Id, Artifact Id, Name, and Description:
   - **Group Id**: This is usually in reverse domain format, e.g., `com.example`.
   - **Artifact Id**: This is the name of your project, e.g., `StudentDatabase`.
   - **Name**: You can give a human-readable name to your project.

- **Description**: Provide a brief description of your project.

For example:

- Group Id: `com.example`
- Artifact Id: `StudentDatabase`
- Name: `StudentDatabaseProject`
- Description: `Java application using Hibernate for Student Database`

6. Click `Finish` to create the Maven project.

I see, let's clarify the sequence for steps 2 and 3:

Step 2: Add dependencies in the pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.cnf.lab</groupId>
  <artifactId>studentDatabase</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>studentDatabaseProject</name>
  <description>lab six hibernate program</description>
  <dependencies>
        <dependency>
                <groupId>com.workshop</groupId>
                <artifactId>demo1</artifactId>
                <version>0.0.1-SNAPSHOT</version>
        </dependency>

        <!--added dependencies from whatsapp(chat and pom.xml)-->
        <dependency>
        <groupId>org.hibernate.javax.persistence</groupId>
        <artifactId>hibernate-jpa-2.1-api</artifactId>
        <version>1.0.2.Final</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>8.3.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-
```

```
validator -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>8.0.1.Final</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.4.2.Final</version>
    </dependency>

  </dependencies>
</project>
```

Step 3: Create `Student.java` under `src/main/java/com/example`

```
package studentDatabase;

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

@Entity
@Table(name = "Student")
public class Student {

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

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

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

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

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

```
    // Getters and setters

    public int getId() {
        return id;
    }

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

    public int getTotalmarks() {
        return totalmarks;
    }

    public void setTotalmarks(int totalmarks) {
        this.totalmarks = totalmarks;
    }

    public String getUsn() {
        return usn;
    }

    public void setUsn(String usn) {
        this.usn = usn;
    }

    public String getName() {
        return name;
    }

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

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

}
```

Step 4: Create `Student_Main.java` under `src/main/java/com/example`

```
package studentDatabase;

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

import java.util.InputMismatchException;
import java.util.List;
import java.util.Scanner;

public class Student_Main {

    private SessionFactory sessionFactory;
    private Session session;
    private Transaction transaction;

    public Student_Main() {
        sessionFactory = new
Configuration().configure("hibernate.cfg.xml").buildSessionFactory();
        session = sessionFactory.openSession();
        transaction = session.beginTransaction();
    }

    public void insert(int id, String usn, String name, String address, int
totalmarks) {
        try {
                if (!transaction.isActive()) {
                transaction = session.beginTransaction();
            }

            Student student = new Student();
            student.setId(id);
            student.setUsn(usn);
            student.setName(name);
            student.setAddress(address);
            student.setTotalmarks(totalmarks);
            session.save(student);
            transaction.commit();
            System.out.println("\nStudent inserted successfully.");
        } catch (Exception e) {
```

```
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }

    public void delete(String usn) {
        try {
                if (!transaction.isActive()) {
                transaction = session.beginTransaction();
            }

            Query query = session.createNativeQuery("delete from Student
where usn = :usn");
            query.setParameter("usn", usn);
            int status = query.executeUpdate();
            if (status > 0) {
                System.out.println(usn + " deleted successfully.");
            } else {
                System.out.println(usn + " not found.");
            }
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }


    public void display() {
        try {
            if (!transaction.isActive()) {
                transaction = session.beginTransaction();
            }

            Query q = session.createQuery("from Student");
            List<Student> students = q.getResultList();

            System.out.println("\nList of Students:");
```

```
            for (Student student : students) {
                System.out.println("\n Name: " + student.getName());
                System.out.println("  - USN: " + student.getUsn());
                System.out.println("  - Address: " + student.getAddress());
                System.out.println("  - Total Marks: " +
student.getTotalmarks());
                System.out.println();
            }

            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }


    public void search(String usn) {
        try {
                if (!transaction.isActive()) {
                transaction = session.beginTransaction();
            }

            Query<Student> query = session.createQuery("from Student where
usn = :usn", Student.class);
            query.setParameter("usn", usn);
            List<Student> students = query.getResultList();

            if (students.isEmpty()) {
                System.out.println("\nStudent with USN " + usn + " not
found.");
            } else {
                System.out.println("\nStudent Details:");
                for (Student student : students) {
                    System.out.println(" Name: " + student.getName());
                    System.out.println("  - USN: " + student.getUsn());
                    System.out.println("  - Address: " +
student.getAddress());
                    System.out.println("  - Total Marks: " +
student.getTotalmarks());
                }
```

```
            }

            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }

    public Session getSession() {
        return session;
    }

    public static void main(String[] args) {
        Student_Main sm = new Student_Main();
        Scanner sc = new Scanner(System.in);

        try {
            while (true) {
                System.out.println("\nMenu");
                System.out.println("1: Insert");
                System.out.println("2: Delete");
                System.out.println("3: Search");
                System.out.println("4: Display");
                System.out.println("5: Exit");

                System.out.print("\nEnter the choice: ");
                try {
                    int ch = sc.nextInt();

                    switch (ch) {
                        case 1:
                            System.out.println("\nEnter the Student Details
to insert");
                            System.out.print(" - Enter the Student id: ");
                            int id = sc.nextInt();
                            System.out.print(" - Enter the Student usn: ");
                            String usn = sc.next();
                            System.out.print(" - Enter the Student name: ");
                            String name = sc.next();
                            System.out.print(" - Enter the Student address:
```

```
");
                            String add = sc.next();
                            System.out.print(" - Enter the Student
totalmarks: ");
                            int tm = sc.nextInt();
                            sm.insert(id, usn, name, add, tm);
                            break;

                        case 2:
                            System.out.print("\nEnter student usn to delete:
");
                            String usnDelete = sc.next();
                            sm.delete(usnDelete);
                            break;

                        case 3:
                            System.out.print("\nEnter student usn to search:
");
                            String usnSearch = sc.next();
                            sm.search(usnSearch);
                            break;

                        case 4:
                            System.out.println("\nDisplaying all students");
                            sm.display();
                            break;

                        case 5:
                            System.out.println("\nExiting...");
                            return;

                        default:
                            System.out.println("\nInvalid choice! Please
make a valid choice.\n");
                            break;
                    }
                } catch (InputMismatchException e) {
                    System.out.println("\nInvalid input. Please enter a
number.");
                    sc.nextLine();
                }
            }
        } finally {
```

```
            sc.close();
            sm.getSession().close();
        }
    }
}
```

These classes should be created in the specified package (`com.example`). The `Student.java` class represents the entity, and the `Student_Main.java` class contains the main logic for interacting with the database.

Step 5: Create `hibernate.cfg.xml` and `student.hbm.xml` under `src/main/resources`

`hibernate.cfg.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property name="hibernate.connection.password">test</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/student</propert
y>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">sql_zaiba</property>
        <property name="hibernate.connection.pool_size">10</property>
        <property name="show_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
       <mapping resource="student.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

The `hibernate.cfg.xml` file is a configuration file for Hibernate that provides settings and properties to configure Hibernate's behavior. Let's break down the content of this file:

1. **XML Declaration:**

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

   This line declares that the document is an XML file with version 1.0 and encoded in UTF-8.

2. **DOCTYPE Declaration:**

```
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

This declaration specifies the Document Type Definition (DTD) for Hibernate configuration. It defines the structure and the legal elements and attributes of the XML file. In this case, it points to the Hibernate Configuration DTD version 3.0.

3. **Root Element - `<hibernate-configuration>`:**

```
<hibernate-configuration>
```

This is the root element of the Hibernate configuration file.

4. **Session Factory Configuration - `<session-factory>`:**

```
<session-factory>
```

This element contains the configuration settings related to the Hibernate SessionFactory.

5. **Database Connection Properties:**

```
<property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</proper
ty>
<property name="hibernate.connection.password">test</property>
<property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/student</prop
erty>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">sql_zaiba</property>
<property name="hibernate.connection.pool_size">10</property>
```

   - `hibernate.connection.driver_class`: Specifies the JDBC driver class for the database.
   - `hibernate.connection.password`: Specifies the database password.
   - `hibernate.connection.url`: Specifies the JDBC URL for the database.
   - `hibernate.connection.username`: Specifies the database username.
   - `hibernate.connection.pool_size`: Specifies the size of the connection pool.

6. **Show SQL Property:**

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

This property is set to `true` to enable the display of SQL statements in the console. It's useful for debugging and understanding what queries Hibernate is executing.

7. **DDL Generation Strategy:**

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

- `hibernate.hbm2ddl.auto`: Specifies the action Hibernate should take with the database schema. In this case, it's set to `update`, which means Hibernate will automatically update the database schema based on the entity mappings.

8. **Mapping Configuration - `<mapping>`:**

   ```
   <mapping resource="student.hbm.xml"/>
   ```

   This element specifies the location of the Hibernate mapping file (`student.hbm.xml`). It tells Hibernate where to find the mapping information for Java entities.

9. **Closing Tags:**

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

   These tags close the `<session-factory>` and `<hibernate-configuration>` elements, indicating the end of the Hibernate configuration file.

In summary, the `hibernate.cfg.xml` file provides essential configuration settings for Hibernate, including database connection properties, SQL display settings, DDL generation strategy, and the location of the Hibernate mapping file. It is used to configure the `SessionFactory` and, subsequently, Hibernate's behavior in a Java application.

`student.hbm.xml`

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

<hibernate-mapping>
    <class name="studentDatabase.Student" table="Student">
        <id name="id" type="int" column="id">
            <generator class="native" />
        </id>
        <property name="totalmarks" column="totalmarks" type="int" />
        <property name="usn" column="usn" type="string" />
        <property name="name" column="name" type="string" />
        <property name="address" column="address" type="string" />
    </class>
</hibernate-mapping>
```

The `student.hbm.xml` file is a Hibernate mapping file, which defines how the Java objects (in this case, the `Student` class) are mapped to database tables. Let's break down the content of this file:

1. **XML Declaration:**

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

This line declares that the document is an XML file with version 1.0 and encoded in UTF-8.

2. **DOCTYPE Declaration:**

```
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
```

This declaration specifies the Document Type Definition (DTD) for Hibernate mapping. It defines the structure and the legal elements and attributes of the XML file. In this case, it points to the Hibernate Mapping DTD version 3.0.

3. **Root Element - `<hibernate-mapping>`:**

```
<hibernate-mapping>
```

This is the root element of the Hibernate mapping file.

4. **Class Mapping - `<class>`:**

```
<class name="studentDatabase.Student" table="Student">
```

- `name`: Specifies the fully qualified class name of the Java class (`studentDatabase.Student`).
- `table`: Specifies the name of the database table associated with this class (`Student`).

5. **ID Mapping - `<id>`:**

```
<id name="id" type="int" column="id">
    <generator class="native" />
</id>
```

- `name`: Specifies the name of the Java property that represents the primary key (`id`).
- `type`: Specifies the data type of the primary key (`int`).
- `column`: Specifies the corresponding column name in the database table (`id`).
- `<generator>`: Specifies the strategy used for generating primary key values. In this case, it uses the native strategy, which typically relies on an auto-incrementing column in the database.

6. **Property Mappings - `<property>`:**

```
<property name="totalmarks" column="totalmarks" type="int" />
<property name="usn" column="usn" type="string" />
<property name="name" column="name" type="string" />
<property name="address" column="address" type="string" />
```

- `name`: Specifies the name of the Java property.
- `column`: Specifies the corresponding column name in the database table.
- `type`: Specifies the data type of the property.

7. **Closing Tags:**

```
</class>
</hibernate-mapping>
```

These tags close the `<class>` and `<hibernate-mapping>` elements, indicating the end of the Hibernate mapping file.

In summary, this Hibernate mapping file defines the mapping between the `Student` Java class and the `Student` database table. It specifies how the attributes of the Java class (`id`, `totalmarks`, `usn`, `name`, `address`) are mapped to columns in the database table.

Step 5: Set Up MySQL Database

(On Terminal type - `mysql -u root -p`)

1. **Create MySQL Database:**
   - Open your MySQL command-line client.
   - Create a new database named "student" :

```
CREATE DATABASE student;

(Verify databases - SHOW DATABASES;)
```

   - Switch to the newly created database:

```
USE student;
```

2. **Create Table:**
   - Run the following SQL script to create a table named "Student" within the "student" database:

```
CREATE TABLE Student (
    id INT PRIMARY KEY AUTO_INCREMENT,
    usn VARCHAR(255),
    name VARCHAR(255),
    address VARCHAR(255),
    totalmarks INT
);

(Verify databases - SHOW TABLES;)
```

Step 7: Run Hibernate Application

- Run Hibernate Application:
  Right-click on the Student_Main class in Eclipse.
  Choose Run As > Java Application.

Step 8: Verify Database Changes

1. **Check Database:**
    - Go back to your MySQL command-line client.
    - Query the "Student" table to verify that records are being inserted, updated, or deleted based on your application's operations:

      ```
      SELECT * FROM Student;
      ```

    - Ensure that the data in the database corresponds to the operations you performed using the Hibernate application.