**Hands on 2**

**Hibernate XML Config implementation walk through**

**1. Object-to-Relational Mapping in Hibernate XML**

Hibernate maps Java classes to database tables using an XML mapping. This file defines how class fields (attributes) are connected to table columns. It allows Hibernate to automatically convert Java objects into database records and vice versa.

**Example:**

**employee.hbm.xml**

<hibernate-mapping>

<class name="com.example.Employee" table="employee">

<id name="id" column="emp\_id">

<generator class="native"/>

</id>

<property name="name" column="emp\_name"/>

<property name="salary" column="emp\_salary"/>

</class>

</hibernate-mapping>

**2.End to End operations in Hibernate:**

**SessionFactory**

SessionFactory is a heavyweight object that reads the Hibernate configuration and mapping files to set up the connection with the database.

It is created only once and used to open sessions.

**Example:**

SessionFactory factory = new Configuration().configure().buildSessionFactory();

**Session**

A Session is a lightweight object used to interact with the database.

It is created from the SessionFactory and is used to perform operations like saving, fetching, and deleting records.

**Example:**

Session session = factory.openSession();

**Transaction**

A Transaction in Hibernate ensures that a group of operations either all succeed or all fail.

It helps maintain data consistency and is needed during insert, update, or delete operations.

**Example:**

Transaction tx = session.beginTransaction();

**beginTransaction()**

beginTransaction() starts a new transaction.

You must call this before making any changes to the database using Hibernate.

**Example:**

Transaction tx = session.beginTransaction();

**commit()**

commit() finalizes the transaction and saves all changes permanently in the database. If not called, changes are not saved.

**Example:**

tx.commit();

**rollback()**

If any error occurs during a transaction, rollback() is used to undo the changes made during that transaction.

**Example:**

tx.rollback();

**session.save()**

**Explanation:**  
sesson.save() is used to store a new object (row) into the database. It inserts the record based on the mapping.

**Example:**

Employee emp = new Employee(101, "Raj", 50000);

session.save(emp);

**session.createQuery().list()**

This is used to retrieve multiple records from the database using HQL (Hibernate Query Language). The result is returned as a list.

**Example:**

List<Employee> list = session.createQuery("from Employee").list();

for (Employee emp : list) {

System.out.println(emp);

}

**session.get()**

session.get() is used to fetch a single record by its primary key. If the record is not found, it returns null.

**Example:**

Employee emp = session.get(Employee.class, 101);

System.out.println(emp);

**session.delete()**

session.delete() is used to remove an object from the database permanently.

**Example:**

Employee emp = session.get(Employee.class, 101);

session.delete(emp);

**Hands on 3**

**Hibernate Annotation Config implementation walk through**

**1. How Object-to-Relational Mapping is Done Using Annotations**

In annotation-based Hibernate configuration, we don't use XML mapping files (.hbm.xml). Instead, we directly annotate the Java class (called the persistence class) using annotations like @Entity, @Table, @Id, etc. These annotations tell Hibernate how to map class fields to database table columns.

**Example: Employee.java**

import jakarta.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "name")

private String name;

@Column(name = "salary")

private double salary;

// Getters and setters

}

**2. Explanation of Each Annotation and Configuration Item**

**@Entity**

Marks the class as an entity, meaning it will be mapped to a table in the database.

**Example:**

@Entity

public class Employee { ... }

**@Table**

Specifies the name of the table in the database that the entity maps to.

**Example:**

@Table(name = "employee")

**@Id**

Marks a field as the **primary key** of the entity.

**Example:**

@Id

private int id;

**@GeneratedValue**

Automatically generates the primary key value. Strategy like IDENTITY or AUTO can be used.

**Example:**

@GeneratedValue(strategy = GenerationType.IDENTITY)

**@Column**

Maps the class field to a specific column in the database table.

**Example:**

@Column(name = "salary")

private double salary;

**3. Hibernate Configuration (hibernate.cfg.xml)**

This file connects the application to the database and tells Hibernate how to work.

**Sample hibernate.cfg.xml:**

<hibernate-configuration>

<session-factory>

**<!-- Database Dialect -->**

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

**<!-- JDBC Driver -->**

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

**<!-- DB Connection URL -->**

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/yourdb</property>

**<!-- Username -->**

<property name="hibernate.connection.username">root</property>

**<!-- Password -->**

<property name="hibernate.connection.password">password</property>

**<!-- Show SQL Queries in console -->**

<property name="hibernate.show\_sql">true</property>

**<!-- Automatically create table -->**

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

**<!-- Register Annotated Class -->**

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

</session-factory>

</hibernate-configuration>

**Dialect**

Specifies the SQL dialect Hibernate should use for the chosen database.

**Example:**

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

**Driver**

The JDBC driver class for the database.

**Example:**

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

**Connection URL**

The full database URL to which Hibernate connects.

**Example:**

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/yourdb</property>

**Username**

Username for the database connection.

**Example:**

<property name="hibernate.connection.username">root</property>

**Password**

Password for the database user.

**Example:**

<property name="hibernate.connection.password">password</property>