**Handson2:**

**Hibernate XML Config implementation walk through**

**Explain how object to relational database mapping done in hibernate xml configuration :**

At the heart of Hibernate is the idea of **mapping Java objects to database tables**. This is the crux of what makes Hibernate powerful — it eliminates the boilerplate JDBC code and makes database operations seamless and elegant.

**How does the XML configuration enable mapping?**

**1.Hibernate Configuration File (hibernate.cfg.xml):**  
This file tells Hibernate where your database is, what dialect to use, and where to find your mapping files.  
For example:

<hibernate-configuration

<session-factory>

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

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

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

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

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

<mapping resource="Employee.hbm.xml"/>

</session-factory>

</hibernate-configuration>

**Here:**

* hibernate.dialect specifies the SQL dialect for your DBMS.
* hibernate.connection.\* properties configure the DB connection.
* <mapping resource="Employee.hbm.xml"/> tells Hibernate to use the **Employee.hbm.xml** mapping file.

**2.Mapping File (Employee.hbm.xml)**  
This file defines **how a Java class maps to a table** and **its properties to table columns**.

Example:

<hibernate-mapping>

<class name="Employee" table="EMPLOYEE">

<id name="id" column="ID">

<generator class="native"/>

</id>

<property name="firstName" column="FIRST\_NAME"/>

<property name="lastName" column="LAST\_NAME"/>

<property name="salary" column="SALARY"/>

</class>

</hibernate-mapping>

**In this file:**

* class specifies the fully qualified Java class name and its mapped database table.
* id maps the primary key.
* property maps the fields of the Java class to columns in the table.
* generator decides how the primary key is generated.

By defining these two XML files, Hibernate understands how to **persist and retrieve Java objects from the database without writing SQL queries explicitly**.

**Explain about following aspects of implementing the end to end operations in Hibernate:**

* **SessionFactory**
* **Session**
* **Transaction**
* **beginTransaction()**
* **commit()**
* **rollback()**
* **session.save()**
* **session.createQuery().list()**
* **session.get()**
* **session.delete()**

**SessionFactory:**

* A heavyweight object that holds the configuration and mappings of Hibernate and is responsible for creating Session objects.
* Since creating a SessionFactory is expensive, it is usually created only once per application (singleton).

**Example:**

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.buildSessionFactory();

**Session:**

* A lightweight object obtained from SessionFactory. It represents a connection between your Java application and the database for a single unit of work.
* All interactions with the database (insert, update, delete, query) happen through this object.

**Example:**

Session session = factory.openSession();

**Transaction:**

* A unit of work in Hibernate. Changes made within a transaction are committed as a single atomic operation.
* Transactions ensure that either all operations within the block succeed, or none (rollback on failure).

**Example:**

Transaction tx = session.beginTransaction();

**beginTransaction():**

* Starts a new database transaction.
* Makes sure that the changes are not immediately reflected in the database until you commit them.

Transaction tx = session.beginTransaction();

**commit():**

* Commits the current transaction.
* All changes made during the transaction are permanently saved to the database.

tx.commit();

**rollback():**

* Rolls back the transaction in case of errors or exceptions.
* Any changes made during the transaction are discarded.

tx.rollback();

**CRUD Operations with Hibernate**

**session.save():**

* Saves a transient object (new Java object) into the database and assigns it a unique identifier.
* Converts the transient state of an object to a persistent state by inserting a record in the database.

**Code Example:**

Employee emp = new Employee("John", "Doe", 5000);

session.save(emp);

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

* Executes an HQL (Hibernate Query Language) query and retrieves the result as a list.
* Provides a simple way to fetch multiple records without writing SQL. Uses HQL, which is object-oriented.

**Example:**

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

for (Employee emp : employees) {

System.out.println(emp.getFirstName());

}

**session.get():**

* Fetches an object from the database by its primary key.
* Returns null if the object doesn’t exist.
* Eagerly loads the object immediately**.**

**Example:**

Employee emp = session.get(Employee.class, 1);

**session.delete():**

* Removes a persistent object from the database.
* Deletes a record corresponding to the Java object from the database.

**Example:**

Employee emp = session.get(Employee.class, 1);

session.delete(emp);

**Sample End-to-End Flow**

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.buildSessionFactory();

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

// Create

Employee emp = new Employee("Jane", "Doe", 6000);

session.save(emp);

// Read

Employee retrieved = session.get(Employee.class, emp.getId());

System.out.println("Retrieved: " + retrieved.getFirstName());

// Update

retrieved.setSalary(7000);

session.update(retrieved);

// Delete

session.delete(retrieved);

tx.commit();

} catch (Exception e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

factory.close();

}