***Hands On : 2 Hibernate XML Config implementation***

**1. Object to Relational Mapping (ORM) in Hibernate XML Configuration**

Hibernate uses an XML mapping file (like Employee.hbm.xml) to map Java class attributes to database table columns.

**Example Mapping File: Employee.hbm.xml**

<hibernate-mapping>

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

<id name="id" column="id" type="int">

<generator class="native"/>

</id>

<property name="firstName" column="first\_name" type="string"/>

<property name="lastName" column="last\_name" type="string"/>

<property name="salary" column="salary" type="int"/>

</class>

</hibernate-mapping>

**Key Points:**

| **XML Element** | **Description** |
| --- | --- |
| <class> | Maps Java class (Employee) to DB table (EMPLOYEE) |
| <id> | Maps primary key field (id) to DB column |
| <property> | Maps each field to a column (firstName, lastName, etc.) |
| type | Specifies Hibernate data type, e.g., string, int |
| <generator> | Auto-generates ID (e.g., using DB auto-increment) |

This is how Hibernate **automatically understands how to persist or retrieve** objects from a relational database.

**2. End-to-End Hibernate Workflow with Key Concepts**

Let's go through each of the components involved in the ManageEmployee class.

**🔷 a. SessionFactory**

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

* **Definition**: Singleton object, initialized once per application.
* **Role**: Reads hibernate.cfg.xml and mapping files (\*.hbm.xml) and sets up DB connection, dialect, etc.
* **Usage**: Used to create Session objects.

**🔷 b. Session**

Session session = factory.openSession();

* **Definition**: Lightweight object for a single unit of work (DB operations).
* **Role**: Interface between Java code and database.
* **Lifecycle**: Open a session → Do DB work → Close session.

**🔷 c. Transaction**

Transaction tx = session.beginTransaction();

tx.commit();

* **Definition**: Wraps a set of DB operations into a single unit.
* **Role**: Ensures atomicity — all or none of the operations succeed.

**🔷 d. beginTransaction()**

tx = session.beginTransaction();

* Starts a new database transaction.

**🔷 e. commit()**

tx.commit();

* Finalizes and saves all DB operations made in the transaction.

**🔷 f. rollback()**tx.rollback();

* Reverses all changes in case of an exception or error.

**3. Hibernate CRUD Operation Methods**

Let's break down the CRUD methods implemented in ManageEmployee.java.

**🔷 g. session.save()**

employeeID = (Integer) session.save(employee);

* **Purpose**: Inserts the Java object into the database.
* **Returns**: The generated primary key (ID).

**🔷 h. session.createQuery().list()**

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

* **Purpose**: Executes an HQL query (FROM Employee) to fetch all records.
* **Returns**: A list of Employee objects.

**🔷 i. session.get()**

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

* **Purpose**: Fetches a single object by its primary key.
* **Returns**: The object or null if not found.

**🔷 j. session.delete()**

session.delete(employee);

* **Purpose**: Deletes the object from the database.

**Example: How it all connects in code**

Here’s a short snippet using these elements:

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction(); // beginTransaction()

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

session.save(e); // save()

tx.commit(); // commit()

} catch (Exception ex) {

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

} finally {

session.close(); // close session

}

**✅ Summary Table**

| **Concept** | **Description** |
| --- | --- |
| Mapping XML | Maps class → table and fields → columns |
| SessionFactory | Sets up DB connection and ORM config |
| Session | Opens a link to perform DB actions |
| Transaction | Begins and manages DB changes |
| beginTransaction() | Starts transaction |
| commit() | Finalizes DB changes |
| rollback() | Cancels changes if errors |
| save() | Inserts record |
| createQuery().list() | Fetches all records |
| get() | Retrieves one record by ID |
| delete() | Removes record from DB |