**Hibernate Annotation Configuration – Hands-on Walkthrough**

In this walkthrough, we explore how Hibernate can be configured using **Java annotations**, which is a cleaner and more modern approach compared to XML mappings. Annotations are written directly in the Java class and help map it to the corresponding database table.

**Object-Relational Mapping with Annotations**

Hibernate allows mapping a Java class to a relational database table using **annotations** instead of external .xml files. This simplifies configuration and improves maintainability.

**Persistence Class: Employee.java**

Here’s how a basic Employee entity is mapped using annotations:

import javax.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "first\_name")

private String firstName;

@Column(name = "last\_name")

private String lastName;

@Column(name = "salary")

private double salary;

// Constructors, Getters, Setters, toString()

}

**Annotation Explanations**

| **Annotation** | **Purpose** |
| --- | --- |
| @Entity | Marks this class as a Hibernate entity. Hibernate will map it to a database table. |
| @Table(name="employee") | Specifies the table name to map with the class. |
| @Id | Specifies the primary key field of the entity. |
| @GeneratedValue(strategy = GenerationType.IDENTITY) | Auto-generates the primary key (commonly used with MySQL auto-increment). |
| @Column(name="first\_name") | Maps a class field to a column in the database. You can customize the column name using this annotation. |

**Hibernate Configuration File: hibernate.cfg.xml**

Even when using annotations, we still need the hibernate.cfg.xml file to configure database connection and Hibernate settings.

**Sample Configuration:**

<hibernate-configuration>

<session-factory>

<!-- Database Connection Settings -->

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

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

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

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

<!-- Hibernate Dialect -->

<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>

<!-- Optional Settings -->

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

<property name="format\_sql">true</property>

<!-- Annotated Class -->

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

</session-factory>

</hibernate-configuration>

**Explanation of Configuration Tags**

| **Tag** | **Description** |
| --- | --- |
| hibernate.dialect | Tells Hibernate which SQL dialect to use based on your DB (e.g., MySQL, Oracle). For MySQL: MySQL5Dialect or MySQL8Dialect. |
| hibernate.connection.driver\_class | Specifies the JDBC driver for the database. For MySQL: com.mysql.cj.jdbc.Driver. |
| hibernate.connection.url | Connection string that includes the database location and name. |
| hibernate.connection.username/password | Database login credentials. |
| <mapping class="..."/> | Registers the annotated entity class with Hibernate. |

**Running Hibernate Operations (Summary)**

Once configuration is complete, you can use the Session and Transaction APIs to perform database operations:

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

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

Employee emp = new Employee("John", "Doe", 50000);

session.save(emp); // Insert

Employee empFromDb = session.get(Employee.class, 1); // Fetch

session.delete(empFromDb); // Delete

tx.commit();

session.close();