## ****2. Hibernate XML Configuration: Explained****

Hibernate provides a powerful mechanism called **Object-Relational Mapping (ORM)** that enables Java applications to interact with relational databases seamlessly. When using **XML-based configuration**, the mappings between Java classes and database tables are maintained in external XML files, making the application modular and easier to maintain.

### ****1. XML-Based Object to Relational Mapping (ORM)****

In Hibernate, when XML is used for ORM, all mappings are defined in .hbm.xml files. These files contain information that tells Hibernate how to map Java class fields to database table columns. This configuration is ideal when annotations aren't preferred or when working in environments with strict separation between business and persistence layers.

#### ****Example:**** employee.hbm.xml

xml

CopyEdit

<hibernate-mapping>

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

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

<generator class="increment"/>

</id>

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

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

</class></hibernate-mapping>

This file instructs Hibernate to map the Employee class to the employee table in the database. The id field is auto-generated using the increment strategy, while name and salary fields are mapped to their respective columns.

### ****2. SessionFactory****

The **SessionFactory** is a crucial part of Hibernate, acting as a factory for Session instances.

It is **heavyweight** and **thread-safe**, typically initialized once when the application starts.

The hibernate.cfg.xml file is used to set up database connectivity and mapping files for the SessionFactory.

### ****3. Session****

A **Session** is the main interface between the Java application and the database.

It is **not thread-safe** and used for a single unit of work.

Developers use it to perform database operations like inserting, updating, deleting, and retrieving data.

### ****4. Transaction****

A **Transaction** represents a single atomic operation or a group of operations.

It ensures that all operations within it either **succeed together** or **fail completely**.

Required for making **changes permanent** or reverting them in case of failure.

### ****5. beginTransaction()****

This method is used to **initiate a transaction**.

It returns a Transaction object that allows further control (like commit or rollback).

### ****6. commit()****

Applies all changes made during the transaction to the database.

Once committed, the changes become **final and persistent**.

### ****7. rollback()****

Used to **cancel a transaction** if an error occurs during execution.

It reverts the database back to the state before the transaction began, ensuring **data safety**.

### ****8. session.save(Object entity)****

Persists a new instance of an object into the database.

Internally generates an INSERT SQL query for the specified entity.

### ****9. session.createQuery("from Employee").list()****

Executes an HQL (Hibernate Query Language) command to fetch data.

Returns the result as a List of objects.

Often used for **retrieving multiple records**.

### ****10. session.get(Class, primaryKey)****

Fetches a single entity based on its **primary key value**.

Returns the actual object or null if no corresponding row exists.

Performs the database hit immediately.

### ****11. session.delete(Object entity)****

Removes the provided object from the database.

Requires the object to be in a **persistent state**, which usually means it must be fetched first.

### Summary

This XML-based approach to Hibernate configuration provides a clean and clear separation between Java classes and their database mappings. Although modern applications often use annotations, XML mappings are still relevant for applications with strict layering, external configuration needs, or legacy systems.