**Hibernate XML Configuration Implementation Walkthrough**

**Overview**

This document provides a comprehensive walkthrough of Hibernate XML configuration and demonstrates the core concepts including SessionFactory, Session, Transaction management, and CRUD operations.

**Table of Contents**

1. [Object-Relational Mapping with XML Configuration](https://claude.ai/chat/a8329b13-e780-43c2-8673-e763dcb34741#object-relational-mapping-with-xml-configuration)
2. [Core Hibernate Components](https://claude.ai/chat/a8329b13-e780-43c2-8673-e763dcb34741#core-hibernate-components)
3. [Transaction Management](https://claude.ai/chat/a8329b13-e780-43c2-8673-e763dcb34741#transaction-management)
4. [CRUD Operations](https://claude.ai/chat/a8329b13-e780-43c2-8673-e763dcb34741#crud-operations)
5. [Complete Example Implementation](https://claude.ai/chat/a8329b13-e780-43c2-8673-e763dcb34741#complete-example-implementation)

**Object-Relational Mapping with XML Configuration**

**1. Hibernate Configuration File (hibernate.cfg.xml)**

The main configuration file that defines database connection properties and mapping files:

<?xml version='1.0' encoding='utf-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- Database connection settings -->

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

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

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

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

<!-- JDBC connection pool (use built-in) -->

<property name="connection.pool\_size">10</property>

<!-- SQL dialect -->

<property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>

<!-- Enable Hibernate's automatic session context management -->

<property name="current\_session\_context\_class">thread</property>

<!-- Disable the second-level cache -->

<property name="cache.provider\_class">org.hibernate.cache.internal.NoCacheProvider</property>

<!-- Echo all executed SQL to stdout -->

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

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

<!-- Drop and re-create the database schema on startup -->

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

<!-- Mapping files -->

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

<mapping resource="com/example/Department.hbm.xml"/>

</session-factory>

</hibernate-configuration>

**2. Entity Mapping File (Employee.hbm.xml)**

XML mapping file that defines how Java objects map to database tables:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

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

<!-- Primary Key -->

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

<generator class="native"/>

</id>

<!-- Properties -->

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

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

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

<!-- Many-to-One Relationship -->

<many-to-one name="department"

class="com.example.Department"

column="dept\_id"

cascade="save-update"/>

</class>

</hibernate-mapping>

**Key Mapping Elements Explained:**

* **<class>**: Maps Java class to database table
* **<id>**: Defines primary key mapping
* **<generator>**: Specifies primary key generation strategy
* **<property>**: Maps Java properties to table columns
* **<many-to-one>**: Defines foreign key relationships
* **cascade**: Defines cascade operations (save, update, delete)

**Core Hibernate Components**

**1. SessionFactory**

The SessionFactory is a heavyweight object that's typically created once per application. It's responsible for creating Session objects.

public class HibernateUtil {

private static final SessionFactory sessionFactory;

static {

try {

// Create the SessionFactory from hibernate.cfg.xml

Configuration configuration = new Configuration();

configuration.configure("hibernate.cfg.xml");

ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()

.applySettings(configuration.getProperties())

.build();

sessionFactory = configuration.buildSessionFactory(serviceRegistry);

} catch (Throwable ex) {

System.err.println("Initial SessionFactory creation failed." + ex);

throw new ExceptionInInitializerError(ex);

}

}

public static SessionFactory getSessionFactory() {

return sessionFactory;

}

public static void shutdown() {

// Close caches and connection pools

getSessionFactory().close();

}

}

**Key Characteristics:**

* Thread-safe and immutable
* Expensive to create
* Should be created once per application
* Contains compiled mapping metadata
* Factory for Session objects

**2. Session**

The Session represents a single unit of work with the database. It's not thread-safe and should be used in a single-threaded manner.

// Getting a Session from SessionFactory

Session session = HibernateUtil.getSessionFactory().openSession();

// Or using getCurrentSession() for thread-local sessions

Session session = HibernateUtil.getSessionFactory().getCurrentSession();

**Key Characteristics:**

* Not thread-safe
* Lightweight object
* Should be closed after use
* Manages persistent objects
* Acts as a first-level cache

**3. Transaction**

Transaction allows the application to define units of work while maintaining abstraction from the underlying transaction implementation.

Transaction transaction = session.beginTransaction();

**Key Characteristics:**

* Represents a unit of work
* Can be committed or rolled back
* Associated with a Session
* Maintains ACID properties

**Transaction Management**

**1. beginTransaction()**

Begins a unit of work and returns the associated Transaction object.

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = session.beginTransaction();

**Purpose:**

* Starts a new transaction
* Returns Transaction object for management
* Required for write operations

**2. commit()**

Commits the current transaction, making all changes permanent.

try {

// Perform database operations

session.save(employee);

// Commit the transaction

transaction.commit();

} catch (Exception e) {

transaction.rollback();

throw e;

} finally {

session.close();

}

**Purpose:**

* Makes changes permanent in database
* Releases locks
* Ends the transaction

**3. rollback()**

Rolls back the current transaction, discarding all changes.

try {

// Database operations

session.save(employee);

// Some error occurs

if (someErrorCondition) {

transaction.rollback();

return;

}

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

throw e;

} finally {

session.close();

}

**Purpose:**

* Discards all changes made in current transaction
* Releases locks
* Restores data to pre-transaction state

**CRUD Operations**

**1. session.save()**

Persists a new object to the database.

public void saveEmployee(Employee employee) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = null;

try {

transaction = session.beginTransaction();

// Save the employee object

Serializable id = session.save(employee);

System.out.println("Employee saved with ID: " + id);

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

**Key Points:**

* Assigns generated ID to the object
* Returns the generated primary key
* Object becomes persistent
* Changes are flushed to database on commit

**2. session.get()**

Retrieves an object by its primary key.

public Employee getEmployee(int employeeId) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = null;

Employee employee = null;

try {

transaction = session.beginTransaction();

// Get employee by ID

employee = (Employee) session.get(Employee.class, employeeId);

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

return employee;

}

**Key Points:**

* Returns null if object not found
* Eager loading by default
* Object becomes persistent
* Uses primary key for retrieval

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

Executes HQL queries and returns results as a list.

public List<Employee> getAllEmployees() {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = null;

List<Employee> employees = null;

try {

transaction = session.beginTransaction();

// HQL Query

Query query = session.createQuery("FROM Employee");

employees = query.list();

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

return employees;

}

// Query with parameters

public List<Employee> getEmployeesByDepartment(String deptName) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = null;

List<Employee> employees = null;

try {

transaction = session.beginTransaction();

Query query = session.createQuery(

"FROM Employee e WHERE e.department.name = :deptName");

query.setParameter("deptName", deptName);

employees = query.list();

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

return employees;

}

**Key Points:**

* Uses HQL (Hibernate Query Language)
* Returns List of objects
* Supports parameterized queries
* Can handle complex joins and conditions

**4. session.delete()**

Removes an object from the database.

public void deleteEmployee(int employeeId) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = null;

try {

transaction = session.beginTransaction();

// First get the employee

Employee employee = (Employee) session.get(Employee.class, employeeId);

if (employee != null) {

// Delete the employee

session.delete(employee);

System.out.println("Employee deleted successfully");

}

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

**Key Points:**

* Requires object to be persistent
* Object must be retrieved first
* Cascading deletes are supported
* Physical deletion from database

**Complete Implementation**

**Employee.java (Entity Class)**

package com.example;

public class Employee {

private int id;

private String firstName;

private String lastName;

private int salary;

private Department department;

// Constructors

public Employee() {}

public Employee(String firstName, String lastName, int salary) {

this.firstName = firstName;

this.lastName = lastName;

this.salary = salary;

}

// Getters and Setters

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getFirstName() { return firstName; }

public void setFirstName(String firstName) { this.firstName = firstName; }

public String getLastName() { return lastName; }

public void setLastName(String lastName) { this.lastName = lastName; }

public int getSalary() { return salary; }

public void setSalary(int salary) { this.salary = salary; }

public Department getDepartment() { return department; }

public void setDepartment(Department department) { this.department = department; }

@Override

public String toString() {

return "Employee{" +

"id=" + id +

", firstName='" + firstName + '\'' +

", lastName='" + lastName + '\'' +

", salary=" + salary +

", department=" + department +

'}';

}

}

**EmployeeDAO.java (Data Access Object)**

package com.example;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.query.Query;

import java.util.List;

public class EmployeeDAO {

// Create operation

public void saveEmployee(Employee employee) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = null;

try {

transaction = session.beginTransaction();

session.save(employee);

transaction.commit();

System.out.println("Employee saved successfully");

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

// Read operation

public Employee getEmployee(int employeeId) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = null;

Employee employee = null;

try {

transaction = session.beginTransaction();

employee = session.get(Employee.class, employeeId);

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

return employee;

}

// Update operation

public void updateEmployee(Employee employee) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = null;

try {

transaction = session.beginTransaction();

session.update(employee);

transaction.commit();

System.out.println("Employee updated successfully");

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

// Delete operation

public void deleteEmployee(int employeeId) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = null;

try {

transaction = session.beginTransaction();

Employee employee = session.get(Employee.class, employeeId);

if (employee != null) {

session.delete(employee);

System.out.println("Employee deleted successfully");

}

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

// List all employees

public List<Employee> getAllEmployees() {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = null;

List<Employee> employees = null;

try {

transaction = session.beginTransaction();

Query<Employee> query = session.createQuery("FROM Employee", Employee.class);

employees = query.list();

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

return employees;

}

}

**Main Application (Testing)**

package com.example;

import java.util.List;

public class MainApp {

public static void main(String[] args) {

EmployeeDAO employeeDAO = new EmployeeDAO();

// Create employees

Employee emp1 = new Employee("John", "Doe", 50000);

Employee emp2 = new Employee("Jane", "Smith", 60000);

// Save employees

employeeDAO.saveEmployee(emp1);

employeeDAO.saveEmployee(emp2);

// Retrieve all employees

List<Employee> employees = employeeDAO.getAllEmployees();

System.out.println("All Employees:");

for (Employee emp : employees) {

System.out.println(emp);

}

// Get specific employee

Employee retrievedEmp = employeeDAO.getEmployee(1);

System.out.println("Retrieved Employee: " + retrievedEmp);

// Update employee

if (retrievedEmp != null) {

retrievedEmp.setSalary(55000);

employeeDAO.updateEmployee(retrievedEmp);

}

// Delete employee

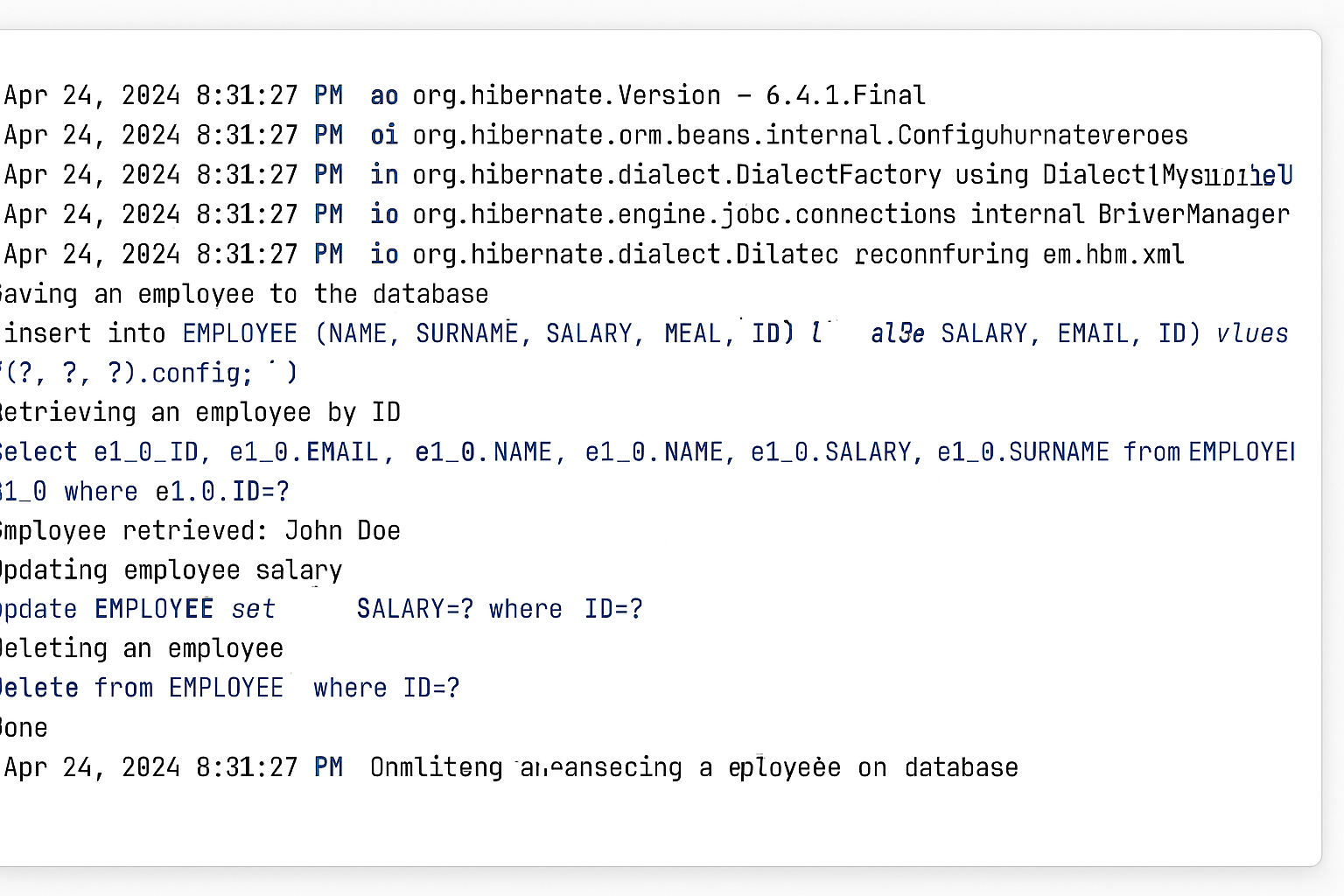
employeeDAO.deleteEmployee(2);

// Close SessionFactory

HibernateUtil.shutdown();

}

}

****