Name – Pradeep Uid – 22BCS10530 Experiment - 9

Create Java applications using Spring and Hibernate for dependency injection, CRUD operations, and transaction management.

Easy Level: Create a simple Spring application that demonstrates Dependency Injection (DI) using Java-based configuration instead of XML. Define a Student class that depends on a Course class. Use Spring’s @Configuration and @Bean annotations to inject dependencies.

Requirements:

1. Define a Course class with attributes courseName and duration.
2. Define a Student class with attributes name and a reference to Course.
3. Use Java-based configuration (@Configuration and @Bean) to configure the beans.
4. Load the Spring context in the main method and print student details.

Medium Level: Develop a Hibernate-based application to perform CRUD (Create, Read, Update, Delete) operations on a Student entity using Hibernate ORM with MySQL. Requirements:

1. Configure Hibernate using hibernate.cfg.xml.
2. Create an Entity class (Student.java) with attributes: id, name, and age.
3. Implement Hibernate SessionFactory to perform CRUD operations.
4. Test the CRUD functionality with sample data. Hard Level: Develop a Spring-based application integrated with Hibernate to manage transactions. Create a banking system where users can transfer money between accounts, ensuring transaction consistency. Requirements:
5. Use Spring configuration with Hibernate ORM.
6. Implement two entity classes (Account.java and Transaction.java).
7. Use Hibernate Transaction Management to ensure atomic operations.
8. If a transaction fails, rollback should occur.
9. Demonstrate successful and failed transactions.

import org.hibernate.\*;

import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import org.hibernate.cfg.Configuration;

import org.springframework.context.annotation.\*;

import javax.persistence.\*;

import java.util.Properties;

public class FullApp {

// ==== ENTITIES ====

@Entity

@Table(name = "students")

public static class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

private int age;

public Student() {}

public Student(String name, int age) {

this.name = name;

this.age = age;

}

public int getId() { return id; }

public String getName() { return name; }

public int getAge() { return age; }

public void setAge(int age) { this.age = age; }

}

@Entity

@Table(name = "accounts")

public static class Account {

@Id

private int accountNumber;

private String holder;

private double balance;

public Account() {}

public Account(int accountNumber, String holder, double balance) {

this.accountNumber = accountNumber;

this.holder = holder;

this.balance = balance;

}

public int getAccountNumber() { return accountNumber; }

public double getBalance() { return balance; }

public void setBalance(double balance) { this.balance = balance; }

public String getHolder() { return holder; }

}

// ==== SPRING CONFIGURATION ====

@Configuration

public static class AppConfig {

@Bean

public SessionFactory sessionFactory() {

Configuration configuration = new Configuration();

Properties settings = new Properties();

settings.put("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

settings.put("hibernate.connection.url", "jdbc:mysql://localhost:3306/yourdb");

settings.put("hibernate.connection.username", "root");

settings.put("hibernate.connection.password", "password");

settings.put("hibernate.dialect", "org.hibernate.dialect.MySQL8Dialect");

settings.put("hibernate.hbm2ddl.auto", "update");

settings.put("show\_sql", "true");

configuration.setProperties(settings);

configuration.addAnnotatedClass(Student.class);

configuration.addAnnotatedClass(Account.class);

StandardServiceRegistryBuilder builder =

new StandardServiceRegistryBuilder().applySettings(configuration.getProperties());

return configuration.buildSessionFactory(builder.build());

}

@Bean

public StudentService studentService() {

return new StudentService(sessionFactory());

}

@Bean

public BankService bankService() {

return new BankService(sessionFactory());

}

}

// ==== STUDENT SERVICE (CRUD) ====

public static class StudentService {

private final SessionFactory sessionFactory;

public StudentService(SessionFactory sf) {

this.sessionFactory = sf;

}

public void create(Student student) {

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

session.save(student);

tx.commit();

session.close();

}

public Student read(int id) {

Session session = sessionFactory.openSession();

Student student = session.get(Student.class, id);

session.close();

return student;

}

public void update(Student student) {

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

session.update(student);

tx.commit();

session.close();

}

public void delete(int id) {

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

Student s = session.get(Student.class, id);

if (s != null) session.delete(s);

tx.commit();

session.close();

}

}

// ==== BANK SERVICE (Transaction Mgmt) ====

public static class BankService {

private final SessionFactory sessionFactory;

public BankService(SessionFactory sf) {

this.sessionFactory = sf;

}

public void transferMoney(int fromAcc, int toAcc, double amount) {

Session session = sessionFactory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Account from = session.get(Account.class, fromAcc);

Account to = session.get(Account.class, toAcc);

if (from.getBalance() < amount) {

throw new RuntimeException("Insufficient funds");

}

from.setBalance(from.getBalance() - amount);

to.setBalance(to.getBalance() + amount);

session.update(from);

session.update(to);

tx.commit();

System.out.println("✅ Transfer Successful");

} catch (Exception e) {

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

System.out.println("❌ Transfer Failed: " + e.getMessage());

} finally {

session.close();

}

}

}

// ==== MAIN METHOD ====

public static void main(String[] args) {

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

// Student CRUD

StudentService studentService = context.getBean(StudentService.class);

Student s = new Student("Alice", 20);

studentService.create(s);

Student fetched = studentService.read(s.getId());

System.out.println("🎓 Student Retrieved: " + fetched.getName());

// Transaction Test

BankService bankService = context.getBean(BankService.class);

bankService.transferMoney(101, 102, 500); // should succeed

bankService.transferMoney(101, 102, 999999); // should fail

}

}