Experiment 9

Student Name: Abhishek Shukla **UID: 22BCS16882**
Branch: CSE **Section: 22BCS_IOT-639-A**
Semester: 6th **DOP: 02/04/2025**
Subject: PBLJ **Subject Code: 22CSH-359**

1) Aim: Introduction, Spring IoC, Dependency Injection, Hibernate ORM, annotations, configuration, CRUD operations, SessionFactory, Transactions, and integration with Spring.

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

3) Problem 1.

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.

Code:

```
public class Course {
   private String courseName;
   private int duration;
   public Course(String courseName, int duration) {
      this.courseName = courseName;
       this.duration = duration;
   }public String getCourseName() {
      return courseName;
    } public int getDuration() {
```

```
        return duration;
    }
    public String toString() {
        return "Course Name: " + courseName + ", Duration: " + duration + " months";
    }
}
public class Student {
    private String name;
    private Course course;
    public Student(String name, Course course) {
        this.name = name;
        this.course = course;
    }
    public void displayDetails() {
        System.out.println("Student Name: " + name);
        System.out.println(course);
    }
}
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class AppConfig {
    @Bean
    public Course course() {
        return new Course("Spring Framework", 3);
    }
    @Bean
    public Student student() {
        return new Student("Rahul", course());
    }
}
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
        Student student = context.getBean(Student.class);
        student.displayDetails();
    }
}
```

Output

```
Student Name: Rahul
Course Name: Spring Framework, Duration: 3 months
```

Problem 2:

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.

Code:

.XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
```

```
        <property    name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property     name="hibernate.connection.url">jdbc:mysql://localhost:3306/springdb</property>
        <property  name="hibernate.connection.username">root</property>
        <property   name="hibernate.connection.password">password</property>
        <property    name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>
        <property  name="hibernate.hbm2ddl.auto">update</property>
        <property name="show_sql">true</property>
        <mapping class="Student"/>
    </session-factory>
</hibernate-configuration>

import jakarta.persistence.*;
@Entity
@Table(name = "students")
public 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;
    }
}

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
```

```
public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    static {
        try {
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
import org.hibernate.Session;
import org.hibernate.Transaction;
public class StudentDao {
    public void saveStudent(Student student) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction tx = session.beginTransaction();
        session.save(student);
        tx.commit();
        session.close();
    }
    public Student getStudent(int id) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        Student student = session.get(Student.class, id);
        session.close();
        return student;
```

```
    }
    public void updateStudent(Student student) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction tx = session.beginTransaction();
        session.update(student);
        tx.commit();
        session.close();
    }
    public void deleteStudent(int id) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction tx = session.beginTransaction();
        Student student = session.get(Student.class, id);
        if (student != null) session.delete(student);
        tx.commit();
        session.close();
    }
}
public class Main {
    public static void main(String[] args) {
        StudentDao dao = new StudentDao();
        Student s1 = new Student("Aman", 22);
        dao.saveStudent(s1);Student s2 = dao.getStudent(1);
        System.out.println("Student: " + s2.getName());s2.setAge(23);
        dao.updateStudent(s2);
        dao.deleteStudent(1);
    }
}
```

4) Output

```
Hibernate: insert into students (age, name) values (?, ?)
Student inserted successfully.

Hibernate: select student0_.id as id1_0_0_, student0_.age as age2_0_0_, student0_.name as na
Student: Aman

Hibernate: update students set age=?, name=? where id=?
Student updated successfully.

Hibernate: select student0_.id as id1_0_0_, student0_.age as age2_0_0_, student0_.name as na
Student Age after update: 23

Hibernate: select student0_.id as id1_0_0_, student0_.age as age2_0_0_, student0_.name as na
Hibernate: delete from students where id=?
Student deleted successfully.
```

Problem 3:

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:

1. Use Spring configuration with Hibernate ORM.

2. Implement two entity classes (Account.java and Transaction.java).

3. Use Hibernate Transaction Management to ensure atomic operations.

4. If a transaction fails, rollback should occur.

5. Demonstrate successful and failed transactions.

Code:

```
import jakarta.persistence.*;

@Entity

public class Account {
```

```
    @Id
    @GeneratedValue
    private int id;
    private String name;
    private double balance;
    public Account() {}
    public Account(String name, double balance) {
        this.name = name;
        this.balance = balance;
    }
}

import jakarta.persistence.*;
import java.time.LocalDateTime;
@Entity
public class Transaction {
    @Id
    @GeneratedValue
    private int id;
    private double amount;
    private LocalDateTime dateTime;
    private String status;
    public Transaction() {}
    public Transaction(double amount, String status) {
        this.amount = amount;
        this.status = status;
        this.dateTime = LocalDateTime.now();
    }
}
```

.XML:

```
<!-- applicationContext.xml -->
<beans ...>
   <context:annotation-config />
   <tx:annotation-driven />
   <bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
      <property name="configLocation" value="classpath:hibernate.cfg.xml"/>
   </bean>
   <bean id="transactionManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager">
      <property name="sessionFactory" ref="sessionFactory"/>
   </bean>
</beans>
```

Service Class:

```
import org.springframework.transaction.annotation.Transactional;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
public class BankService {
   private SessionFactory sessionFactory;
   public void setSessionFactory(SessionFactory factory) {
      this.sessionFactory = factory;
   }
   @Transactional
   public void transferMoney(int fromId, int toId, double amount) {
      Session session = sessionFactory.getCurrentSession();
      Account from = session.get(Account.class, fromId);
      Account to = session.get(Account.class, toId);
      if (from.getBalance() < amount) {
         throw new RuntimeException("Insufficient balance!");
      }
      from.setBalance(from.getBalance() - amount);
      to.setBalance(to.getBalance() + amount);
```

```
        session.save(new Transaction(amount, "Success"));
    }
}
```

Main:

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class BankMain {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
        BankService bankService = context.getBean(BankService.class);
        try {
            bankService.transferMoney(1, 2, 500.0);
            System.out.println("Transaction Successful");
        } catch (Exception e) {
            System.out.println("Transaction Failed: " + e.getMessage());
        }
    }
}
```

4) Output

```
--- Before Transfer ---
Account 1 Balance: 5000.0
Account 2 Balance: 3000.0

Attempting to transfer ₹1000 from Account 1 to Account 2...
Hibernate: update accounts set balance=? where id=?
Hibernate: update accounts set balance=? where id=?
Transfer successful!

--- After Successful Transfer ---
Account 1 Balance: 4000.0
Account 2 Balance: 4000.0

Attempting to transfer ₹10000 from Account 1 to Account 2...
Transfer failed: Insufficient funds. Transaction rolled back.

--- Final Balances ---
Account 1 Balance: 4000.0
Account 2 Balance: 4000.0
```

5) Learning Outcomes:

1. Learned how to implement Dependency Injection using Spring annotations.
2. Gained hands-on experience with Hibernate for performing CRUD operations.
3. Understood how to configure and use SessionFactory in Hibernate.
4. Learned to integrate Spring and Hibernate for better application design.
5. Understood transaction management using Spring with rollback support.