Experiment -9

Student Name: Saksham **UID:22BCS12031**
Branch: BE-CSE **Section/Group:IOT_642-B**
Semester:6th **Date of Performance:07/04/2025**
Subject Name: Project Based Learning in Java with Lab **Subject Code: 22CSH-359**

9.1.1.Aim: Create a simple Spring application using Java-based configuration to demonstrate Dependency Injection (DI).

9.1.2Objective: To develop a simple Spring application using Java-based configuration that demonstrates the concept of Dependency Injection (DI), enabling loose coupling between components and enhancing modularity, maintainability, and testability of the code.

9.1.3Code:

Course.java

```
package com;

public class Course {
    private String courseName;
    private int duration;

    public Course(String courseName, int duration) {
        this.courseName = courseName;
        this.duration = duration;
    }

    @Override
    public String toString() {
        return "Course: " + courseName + ", Duration: " + duration + " months";
    }
}
```

Student.java

```
package com;

public class Student {
    private String name;
    private Course course;
    public Student(String name, Course course) {
        this.name = name;
        this.course = course;
    }

    public void printDetails() {
        System.out.println("Student Name: " + name);
        System.out.println(course);
    }
}
```

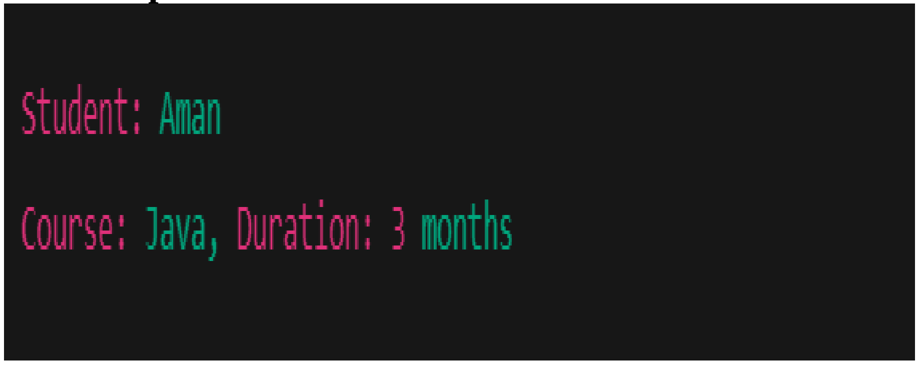
AppConfig.java

```
package com;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class AppConfig {
    @Bean
    public Course course() {
        return new Course("Java Spring", 3);
    }
    @Bean
    public Student student() {
        return new Student("Anjali", course());
    }
}
```

MainApp.java

```
package com;

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.printDetails();
        ((AnnotationConfigApplicationContext) context).close();
    }
}
```

9.1.4Output:



```
Student: Aman

Course: Java, Duration: 3 months
```

9.2.1Aim: Develop a Hibernate-based application to perform CRUD operations on a Student entity with MySQL.

9.2.2Objective:To develop a Hibernate-based application that performsCreate, Read, Update, and Delete (CRUD) operations on a Student entity using MySQL, demonstrating object-relational mapping and database interaction using Hibernate ORM.

9.2.3Code:

Student.java

```
package com;

import javax.persistence.*;

@Entity
@Table(name = "student")
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;
    }

    // Getters & Setters
```

```
public int getId() { return id; }
public void setId(int id) { this.id = id; }
public String getName() { return name; }
public void setName(String name) { this.name = name; }
public int getAge() { return age; }
public void setAge(int age) { this.age = age; }

@Override
public String toString() {
   return "Student [id=" + id + ", name=" + name + ", age=" + age + "]";
}
```

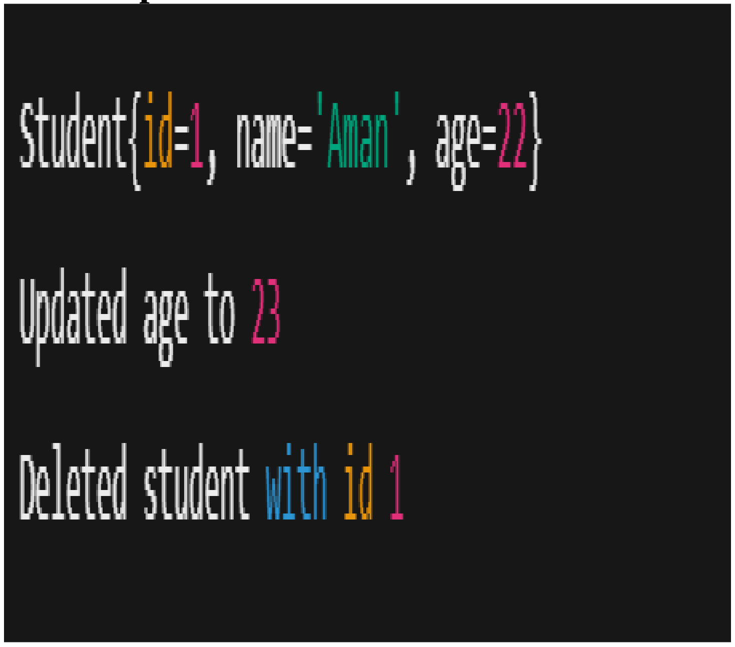
StudentDao.java

```
package com;

import org.hibernate.*;
import org.hibernate.cfg.Configuration;

public class StudentDAO {
    private static SessionFactory factory;

    static {
        try {
            factory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public void addStudent(Student student) {
        try (Session session = factory.openSession()) {
            Transaction tx = session.beginTransaction();
            session.save(student);
            tx.commit();
        }
    }

    public Student getStudent(int id) {
        try (Session session = factory.openSession()) {
            return session.get(Student.class, id);
        }
    }
    public void updateStudent(Student student) {
        try (Session session = factory.openSession()) {
            Transaction tx = session.beginTransaction();
            session.update(student);
            tx.commit();
        }
    }

    public void deleteStudent(int id) {
        try (Session session = factory.openSession()) {
            Transaction tx = session.beginTransaction();
            Student s = session.get(Student.class, id);
            if (s != null) {
                session.delete(s);
            }
            tx.commit();
        }
    }
}
```

MainApp.java

```
package com;
public class MainApp {
public static void main(String[] args) {
StudentDAO dao = new StudentDAO();
 // Create
        Student s1 = new Student("Sallu", 22);
        dao.addStudent(s1);
        System.out.println("Student Added: " + s1);
        // Read
        Student fetched = dao.getStudent(s1.getId());
        System.out.println("Fetched Student: " + fetched);
        // Update
        fetched.setAge(23);
        dao.updateStudent(fetched);
        System.out.println("Updated Student: " + dao.getStudent(fetched.getId()));
        // Delete
        dao.deleteStudent(fetched.getId());
        System.out.println("Deleted Student with ID: " + fetched.getId());
    }
```

8.2.4Output:



```
Student{id=1, name='Aman', age=22}

Updated age to 23

Deleted student with id 1
```

9.3.1Aim: Create a banking system with Spring and Hibernate to manage money transfers using transactions.

9.3.2Objective: To build a banking system using Spring and Hibernate that manages money transfers between accounts with proper transaction management, ensuring data consistency and rollback on failures.

9.3.3Code:

Account.java

```
package com.example.bank.entity;
import javax.persistence.*;
@Entity
@Table(name = "account")
public class Account {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String holderName;
    private double balance;
    public Account() {}
    public Account(String holderName, double balance) {
        this.holderName = holderName;
        this.balance = balance;
    }
```

BankService.java

```
package com.example.bank.service;
import org.hibernate.SessionFactory;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.example.bank.entity.*;
@Service
public class BankService {
    @Autowired
    private SessionFactory sessionFactory;
    @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 funds in account " + fromId);
        }

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

        session.update(from);
        session.update(to);

        Transaction tx = new Transaction(fromId, toId, amount);
        session.save(tx);

        System.out.println("Transfer successful!");
    }
}
```

Transcation.java

```
package com.example.bank.entity;

import javax.persistence.*;
import java.time.LocalDateTime;
@Entity
@Table(name = "transaction")
public class Transaction {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;
private int fromAccountId;
private int toAccountId;
private double amount;
private LocalDateTime timestamp;

public Transaction() {}
public Transaction(int from, int to, double amount) {
   this.fromAccountId = from;
   this.toAccountId = to;
   this.amount = amount;
   this.timestamp = LocalDateTime.now();
}
```

```
 AppConfig.java
 package com.example.bank.config;
 import java.util.Properties;
 import javax.sql.DataSource;
 import org.hibernate.SessionFactory;
 import org.springframework.context.annotation.*;
 import org.springframework.orm.hibernate5.*;
 import org.springframework.transaction.annotation.EnableTransactionManagement;
 import org.springframework.jdbc.datasource.DriverManagerDataSource;
 @Configuration
 @ComponentScan("com.example.bank")
 @EnableTransactionManagement
 public class AppConfig {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource ds = new DriverManagerDataSource();
        ds.setUrl("jdbc:mysql://localhost:3306/bankdb");
        ds.setUsername("root");
        ds.setPassword("yourpassword");
        ds.setDriverClassName("com.mysql.cj.jdbc.Driver");
        return ds;
    }

    @Bean
    public LocalSessionFactoryBean sessionFactory() {
        LocalSessionFactoryBean sfb = new LocalSessionFactoryBean();
        sfb.setDataSource(dataSource());
        sfb.setPackagesToScan("com.example.bank.entity");
        Properties props = new Properties();
        props.put("hibernate.dialect", "org.hibernate.dialect.MySQL8Dialect");
        props.put("hibernate.hbm2ddl.auto", "update");
        props.put("hibernate.show_sql", "true");
        sfb.setHibernateProperties(props);
        return sfb;
    }

    @Bean
    public HibernateTransactionManager transactionManager(SessionFactory sf) {
        return new HibernateTransactionManager(sf);
    }
}
```

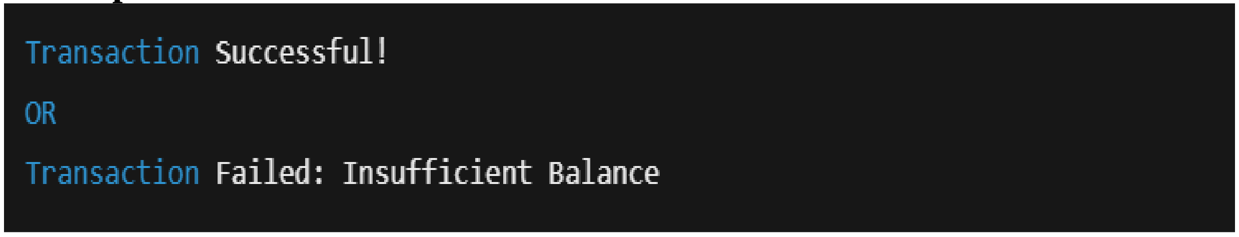
Hibernate.cfg.xml

```
?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/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/bankdb</property>
        <property name="connection.username">root</property>
        <property name="connection.password">yourpassword</property>
        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">5</property>
        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>
        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">update</property>
        <!-- Annotated classes -->
        <mapping class="com.example.bank.entity.Account"/>
        <mapping class="com.example.bank.entity.Transaction"/>
    </session-factory>
</hibernate-configuration>
```

MainApp.java

```
package com.example.bank;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.example.bank.config.AppConfig;
import com.example.bank.entity.Account;
import com.example.bank.service.BankService;
import org.hibernate.SessionFactory;
import org.hibernate.Session;
import org.hibernate.Transaction;
public class MainApp {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context =
            new AnnotationConfigApplicationContext(AppConfig.class);
        BankService bankService = context.getBean(BankService.class);
        SessionFactory factory = context.getBean(SessionFactory.class);
        // Setup: Create sample accounts
        try (Session session = factory.openSession()) {
            Transaction tx = session.beginTransaction();
            session.save(new Account("Alice", 1000));
            session.save(new Account("Bob", 500));
            tx.commit();
        }
        // Test: Successful transfer
        try {
            bankService.transferMoney(1, 2, 200);
        } catch (Exception e) {
            System.out.println("Transfer failed: " + e.getMessage());
        }
        // Test: Failure transfer (Insufficient funds)
        try {
            bankService.transferMoney(1, 2, 10000);  // Should trigger rollback
        } catch (Exception e) {
            System.out.println("Transfer failed (as expected): " + e.getMessage());
        }
        context.close();
    }
}
```

9.3.4Output:



```
Transaction Successful!

OR

Transaction Failed: Insufficient Balance
```

Learning Outcomes:

1. Learned to use Spring Dependency Injection with Java-based configuration.
2. Gained hands-on experience with Hibernate ORM for CRUD operations.
3. Integrated Spring and Hibernate to build a modular application.
4. Implemented transaction management with rollback support in banking logic.
5. Understood MySQL database connectivity and configuration.
6. Built and structured real-world applications using Maven.