Experiment 9

Student Name: Komaldeep **UID: 22BCS14035**
Branch: CSE **Section:618**
Semester: 6th **DOP:18-4-25**
Subject: Java **Subject Code:22CSH-359**

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

Code:

```
 import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Main {

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

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

      public String getCourseName() {
         return courseName;
      }

      public String getDuration() {
         return duration;
      }

      @Override
      public String toString() {
         return "Course [courseName=" + courseName + ", duration=" + duration +
"]";
      }
   }

   public static class Student {
      private String name;
      private Course course;

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

```
        public String getName() {
            return name;
        }

        public Course getCourse() {
            return course;
        }

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

    @Configuration
    public static class AppConfig {

        @Bean
        public Course course() {
            return new Course("Java Programming", "3 Months");
        }

        @Bean
        public Student student() {
            return new Student("John Doe", course());
        }
    }

    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
        Student student = context.getBean(Student.class);
        System.out.println(student);
        context.close();
    }
}
if (count < n) {
    System.out.print("Enter Employee ID: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // consume newline

    System.out.print("Enter Employee Name: ");
    String name = scanner.nextLine();

    System.out.print("Enter Employee Department: ");
    String department = scanner.nextLine();

    System.out.print("Enter Employee Salary: ");
    double salary = scanner.nextDouble();

    employees[count] = new Employee(id, name, department, salary);
    count++;
    System.out.println("Employee added successfully!");
} else {
```

```
                    System.out.println("Employee array is full!");
                }
                break;

            case 2:
                if (count == 0) {
                    System.out.println("No employees to display.");
                } else {
                    System.out.println("\nEmployee Details:");
                    for (int i = 0; i < count; i++) {
                        employees[i].displayEmployee();
                    }
                }
                break;

            case 3:
                System.out.println("Exiting program. Goodbye!");
                scanner.close();
                return;

            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}
```

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

Code:

```
import jakarta.persistence.*;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class Main {

    @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 setName(String name) { this.name = name; }
    public void setAge(int age) { this.age = age; }

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

public static void main(String[] args) {
    Configuration cfg = new Configuration();
    cfg.setProperty("hibernate.connection.driver_class", "com.mysql.cj.jdbc.Driver");
    cfg.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/testdb");
    cfg.setProperty("hibernate.connection.username", "root");
    cfg.setProperty("hibernate.connection.password", "yourpassword");
    cfg.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL8Dialect");
    cfg.setProperty("hibernate.hbm2ddl.auto", "update");
    cfg.setProperty("hibernate.show_sql", "true");
    cfg.addAnnotatedClass(Student.class);

    SessionFactory factory = cfg.buildSessionFactory();
    Session session = factory.openSession();
    Transaction tx = session.beginTransaction();

    Student s1 = new Student("Alice", 22);
    session.save(s1);

    Student s2 = session.get(Student.class, s1.getId());
    System.out.println("Read: " + s2);

    s2.setAge(23);
    session.update(s2);

    Student updated = session.get(Student.class, s2.getId());
    System.out.println("Updated: " + updated);

    session.delete(updated);

    tx.commit();
    session.close();
    factory.close();
}
}
```

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

Code:

```
import jakarta.persistence.*;
import org.hibernate.*;
import org.hibernate.cfg.Configuration;
import org.springframework.context.annotation.*;
import org.springframework.orm.hibernate5.HibernateTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;
import org.springframework.transaction.annotation.Transactional;

public class BankingApp {

    @Entity
    @Table(name = "accounts")
    public static class Account {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private int id;
        private String name;
        private double balance;

        public Account() {}
        public Account(String name, double balance) {
            this.name = name;
            this.balance = balance;
        }

        public int getId() { return id; }
        public String getName() { return name; }
        public double getBalance() { return balance; }
        public void setBalance(double balance) { this.balance = balance; }

        public String toString() {
            return "Account [id=" + id + ", name=" + name + ", balance=" + balance + "]";
        }
    }

    @Entity
    @Table(name = "bank_transactions")
    public static class BankTransaction {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private int id;
        private String fromAccount;
        private String toAccount;
        private double amount;

        public BankTransaction() {}
        public BankTransaction(String fromAccount, String toAccount, double amount)
{
            this.fromAccount = fromAccount;
            this.toAccount = toAccount;
            this.amount = amount;
        }
```

```
        public String toString() {
            return "Transaction [from=" + fromAccount + ", to=" + toAccount + ",
amount=" + amount + "]";
        }
    }

    public static class BankService {
        private SessionFactory sessionFactory;

        public BankService(SessionFactory sessionFactory) {
            this.sessionFactory = sessionFactory;
        }

        @Transactional
        public void transfer(String fromName, String toName, double amount) {
            Session session = sessionFactory.getCurrentSession();

            Account from = session.bySimpleNaturalId(Account.class).load(fromName);
            Account to = session.bySimpleNaturalId(Account.class).load(toName);

            if (from.getBalance() < amount) {
                throw new RuntimeException("Insufficient funds");
            }

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

            session.save(new BankTransaction(fromName, toName, amount));
        }
    }

    @Configuration
    @EnableTransactionManagement
    public static class AppConfig {
        @Bean
        public SessionFactory sessionFactory() {
            Configuration cfg = new Configuration();
            cfg.setProperty("hibernate.connection.driver_class",
"com.mysql.cj.jdbc.Driver");
            cfg.setProperty("hibernate.connection.url",
"jdbc:mysql://localhost:3306/testdb");
            cfg.setProperty("hibernate.connection.username", "root");
            cfg.setProperty("hibernate.connection.password", "yourpassword");
            cfg.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL8Dialect");
            cfg.setProperty("hibernate.hbm2ddl.auto", "update");
            cfg.setProperty("hibernate.show_sql", "true");
            cfg.addAnnotatedClass(Account.class);
            cfg.addAnnotatedClass(BankTransaction.class);
            return cfg.buildSessionFactory();
        }

        @Bean
```

```
        public HibernateTransactionManager transactionManager() {
            return new HibernateTransactionManager(sessionFactory());
        }

        @Bean
        public BankService bankService() {
            return new BankService(sessionFactory());
        }
    }

    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
        SessionFactory factory = context.getBean(SessionFactory.class);

        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();
        session.save(new Account("Alice", 1000));
        session.save(new Account("Bob", 500));
        tx.commit();
        session.close();

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

        try {
            TransactionManager txMgr =
context.getBean(HibernateTransactionManager.class);
            txMgr.getTransaction(null);
            service.transfer("Alice", "Bob", 200);
            txMgr.commit(null);
            System.out.println("Success: Alice transferred $200 to Bob.");
        } catch (Exception e) {
            System.out.println("Failure: " + e.getMessage());
        }

        try {
            TransactionManager txMgr =
context.getBean(HibernateTransactionManager.class);
            txMgr.getTransaction(null);
            service.transfer("Bob", "Alice", 9999); // force failure
            txMgr.commit(null);
        } catch (Exception e) {
            System.out.println("Rollback triggered: " + e.getMessage());
        }

        context.close();
    }
}
```

Learning Outcomes:

1. Demonstrate: Apply key concepts to real-world scenarios to showcase understanding.
2. Analyze: Critically evaluate information, identify patterns, and draw meaningful conclusions.
3. Create: Develop original work, including presentations, reports, or projects, to exhibit comprehension and skills.
4. Communicate: Convey ideas and findings effectively through oral and written communication.
5. Collaborate: Contribute to group projects and exhibit strong teamwork capabilities in a collaborative environment.