EXPERIMENT-9

Student Name: Shivam **UID: 23BCS80044**
Branch: CSE **Section/Group: 642/B**
Semester: 6th **Date of Performance:18-04-2025**
Subject Name: PBLJ **Subject Code: 22CSH-259**

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

9.1.2 Objective: To demonstrate the concept of Dependency Injection (DI) in Spring Framework using Java-based configuration

9.1.3 Code:

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

// Service interface
interface MessageService {
    String getMessage();
}

// Service implementation
class EmailMessageService implements MessageService {
    @Override
    public String getMessage() {
        return "You've got mail!";
    }
}

// Dependent class (client)
class MessagePrinter {
    private final MessageService service;

    // Constructor-based Dependency Injection
    public MessagePrinter(MessageService service) {
        this.service = service;
    }

    public void printMessage() {
```

```
        System.out.println(service.getMessage());
    }
}

// Spring Configuration class
@Configuration
class AppConfig {

    @Bean
    public MessageService messageService() {
        return new EmailMessageService();
    }

    @Bean
    public MessagePrinter messagePrinter() {
        return new MessagePrinter(messageService());
    }
}

// Main application
public class SpringDIApp {
    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
        MessagePrinter printer = context.getBean(MessagePrinter.class);
        printer.printMessage();  // Output: You've got mail!
    }
}
```

9.1.4 Output:

```
You've got mail!
```

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

9.2.2 Objective: To develop a Hibernate-based Java application that demonstrates CRUD (Create, Read, Update, Delete) operations on a Student entity using MySQL as the backend database.

9.2.3 Code:

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

import javax.persistence.*;
import java.util.List;

// Entity class
@Entity
@Table(name = "student")
class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;
    private String email;

    public Student() {}

    public Student(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public int getId() { return id; }
    public String getName() { return name; }
```

```
    public String getEmail() { return email; }

    public void setId(int id) { this.id = id; }
    public void setName(String name) { this.name = name; }
    public void setEmail(String email) { this.email = email; }
}

// Utility class for Hibernate
class HibernateUtil {
    private static SessionFactory sessionFactory;

    static {
        try {
            Configuration config = new Configuration();
            config.configure("hibernate.cfg.xml");
            config.addAnnotatedClass(Student.class);
            sessionFactory = config.buildSessionFactory();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}

// DAO class
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 List<Student> getAllStudents() {
        Session session = HibernateUtil.getSessionFactory().openSession();
        List<Student> students = session.createQuery("from Student", Student.class).list();
        session.close();
        return students;
    }
}

// Main class
public class HibernateStudentApp {
    public static void main(String[] args) {
        StudentDAO dao = new StudentDAO();
```

```
        // CREATE
        Student s1 = new Student("Shivam", "Shivam@gmail.com");
        dao.saveStudent(s1);

        // READ
        Student fetched = dao.getStudent(s1.getId());
        System.out.println("Fetched: " + fetched.getName() + " - " + fetched.getEmail());

        // UPDATE
        fetched.setName("shivam malhotra");
        dao.updateStudent(fetched);

        // LIST ALL
        List<Student> students = dao.getAllStudents();
        for (Student s : students) {
            System.out.println(s.getId() + " - " + s.getName() + " - " + s.getEmail());
        }

        // DELETE
        dao.deleteStudent(fetched.getId());
    }
}
```

Hybernate.cfg.xml:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.org/dtd/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/studentdb</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">@Fghj5678</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="hibernate.show_sql">true</property>
  </session-factory>
</hibernate-configuration>
```

9.2.4 Ouput:

```
Hibernate: insert into student (email, name) values (?, ?)
Hibernate: select student0_.id as id1_0_0_, student0_.email as email2_0_0_, student0_.name as name3_0_0_ from student student0_ where student0_.id=?
Fetched: Shivam - Shivam@gmail.com
Hibernate: update student set email=?, name=? where id=?
Hibernate: select student0_.id as id1_0_, student0_.email as email2_0_, student0_.name as name3_0_ from student student0_
1 - shivam malhotra - Shivam@gmail.com
Hibernate: select student0_.id as id1_0_0_, student0_.email as email2_0_0_, student0_.name as name3_0_0_ from student student0_ where student0_.id=?
Hibernate: delete from student where id=?
```

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

9.3.2 Objective: To create a banking system where the user can:

- Create new accounts.
- Transfer money between accounts.
- Ensure that money transfer operations are managed as transactions using Spring and Hibernate.

9.3.3 Code:

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import javax.persistence.*;

@Entity
@Table(name = "account")
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 void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getBalance() {
        return balance;
    }

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

    public void deposit(double amount) {
        this.balance += amount;
    }

    public void withdraw(double amount) {
        if (this.balance >= amount) {
            this.balance -= amount;
        } else {
            throw new IllegalArgumentException("Insufficient funds");
        }
    }
}

@Service
public class BankingService {

    @Autowired
```

```
    private Session session;

    // Method to transfer money between two accounts
    @Transactional
    public void transferMoney(int fromAccountId, int toAccountId, double amount) {
        Account fromAccount = session.get(Account.class, fromAccountId);
        Account toAccount = session.get(Account.class, toAccountId);

        if (fromAccount == null || toAccount == null) {
            throw new IllegalArgumentException("Account(s) not found");
        }

        fromAccount.withdraw(amount);
        toAccount.deposit(amount);

        session.update(fromAccount);
        session.update(toAccount);
    }

    // Method to create a new account
    public void createAccount(Account account) {
        session.save(account);
    }
}

class HibernateUtil {
    private static SessionFactory sessionFactory;

    static {
        try {
            sessionFactory = new
Configuration().configure().addAnnotatedClass(Account.class).buildSessionFactory();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    public static Session openSession() {
        return sessionFactory.openSession();
    }
}

public class BankingApp {
    public static void main(String[] args) {
```

```
        // Initialize Session
        Session session = HibernateUtil.openSession();

        // Create BankingService Bean
        BankingService bankingService = new BankingService();
        bankingService.setSession(session);

        // 1. Create accounts
        Account account1 = new Account("shivam", 1000);
        Account account2 = new Account("kritika", 500);

        session.beginTransaction();
        bankingService.createAccount(account1);
        bankingService.createAccount(account2);
        session.getTransaction().commit();


        try {
            bankingService.transferMoney(account1.getId(), account2.getId(), 200);
            System.out.println("Money transfer successful!");
        } catch (Exception e) {
            System.out.println("Error during transfer: " + e.getMessage());

        System.out.println("shivam balance: " + account1.getBalance());
        System.out.println("kritika balance: " + account2.getBalance());
    }
}
```

Hibernate.cfg.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.org/dtd/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/bankingdb</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">@Fghj5678</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property name="hibernate.show_sql">true</property>
    </session-factory>
</hibernate-configuration>
```

Pom.xml:

```
<dependencies>
    <dependency>
```

```
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.4.30.Final</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.13</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.25</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>5.3.13</version>
    </dependency>
</dependencies>
```

9.3.4 Output:

```
Hibernate: insert into account (balance, name) values (?, ?)
Hibernate: insert into account (balance, name) values (?, ?)
Hibernate: update account set balance=? where id=?
Hibernate: update account set balance=? where id=?
Money transfer successful!
shivam balance: 800.0
kritika balance: 700.0
```