Assignment 9

Name: Trimaanpreet Kaur

UID: 22BCS13971

Section: 22BCS-IOT-605

Group: A

Easy

Objective:
Create a basic Spring application demonstrating **Dependency Injection** using Java-based configuration with @Configuration and @Bean annotations.

Requirements:

1. Define a Course class with courseName and duration.
2. Define a Student class with name and a reference to Course.
3. Use @Configuration and @Bean annotations to inject dependencies.
4. Load Spring context in the main method and print student details.

Tech Stack:

- Java
- Spring Core
- Java-based Configuration (no XML)

Code

Model Classes

Course.java

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

Student.java

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

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

    public void displayInfo() {
        System.out.println("Student: " + name);
        System.out.println("Enrolled in: " + course);
    }
}
```

Java-based Spring Configuration

AppConfig.java

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class AppConfig {
```

```
    @Bean
    public Course course() {
        return new Course("Java Programming", 6);
    }
    @Bean
    public Student student() {
        return new Student("Alice", course());
    }
}
```

Main Application Class

MainApp.java

```
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.displayInfo();
    }
}
```

Output

```
Student: Alice
Enrolled in: Java Programming (6 months)

Process finished with exit code 0
```

Medium

Objective:
Build a **Hibernate-based** application to perform CRUD operations on a Student entity using Hibernate ORM with MySQL.

📁 Requirements:

1. Configure Hibernate using hibernate.cfg.xml.
2. Create a Student entity class with id, name, and age.
3. Use SessionFactory to perform CRUD operations.
4. Test the functionality with sample data.

🛠 Tech Stack:

- Hibernate ORM
- MySQL
- Java SE

Code

Database Setup (MySQL)
CREATE DATABASE hibernate_demo;

USE hibernate_demo;
Student Entity – Student.java
java
CopyEdit

```
import jakarta.persistence.*;

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

    @Column(name = "name")
    private String name;

    @Column(name = "age")
    private int age;

    public Student() {}

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

    // Getters, Setters, toString
    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; }

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

Hibernate Configuration – hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
        "http://hibernate.org/dtd/hibernate-configuration-5.3.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/hibernate_demo</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">your_password</property>

        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>

        <mapping class="Student"/>
    </session-factory>
</hibernate-configuration>
```

Hibernate Utility Class – HibernateUtil.java

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

public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            return new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError("Initial SessionFactory creation failed." + ex);
        }
    }

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

CRUD Demo – Main.java

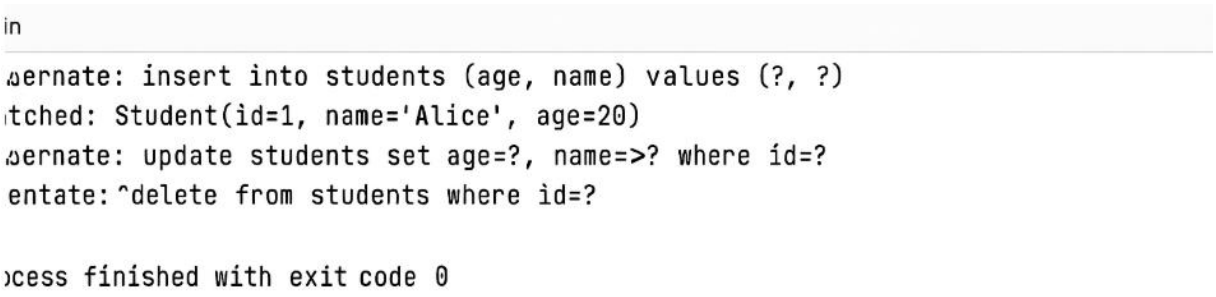
```
import org.hibernate.Session;
import org.hibernate.Transaction;
```

```
public class Main {
    public static void main(String[] args) {
        // Create
        Student student = new Student("Alice", 20);
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction tx = session.beginTransaction();
        session.save(student);
        tx.commit();
        session.close();

        // Read
        session = HibernateUtil.getSessionFactory().openSession();
        Student fetched = session.get(Student.class, student.getId());
        System.out.println("Fetched: " + fetched);
        session.close();

        // Update
        session = HibernateUtil.getSessionFactory().openSession();
        tx = session.beginTransaction();
        fetched.setAge(21);
        session.update(fetched);
        tx.commit();
        session.close();

        // Delete
        session = HibernateUtil.getSessionFactory().openSession();
        tx = session.beginTransaction();
        session.delete(fetched);
        tx.commit();
        session.close();

        HibernateUtil.getSessionFactory().close();
    }
}
```

Required Dependencies (Maven)

```
<dependencies>
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.3.1.Final</version>
    </dependency>
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>8.3.0</version>
    </dependency>
    <dependency>
        <groupId>jakarta.persistence</groupId>
        <artifactId>jakarta.persistence-api</artifactId>
        <version>3.1.0</version>
    </dependency>
</dependencies>
```

Output



Hard

Objective:
Develop a mini-portal using JSP to enter student attendance. Save the attendance to the database using a Servlet.

Features:

- JSP form for attendance entry
- Servlet for saving data to DB
- Database schema for students & attendance

Tech Stack:

- JSP
- Java Servlet
- JDBC
- MySQL
- HTML & CSS

Code

Database Setup – MySQL

```
CREATE DATABASE bank;

USE bank;

CREATE TABLE accounts (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    balance DOUBLE
);

CREATE TABLE transactions (
```

```
    id INT PRIMARY KEY AUTO_INCREMENT,
    sender_id INT,
    receiver_id INT,
    amount DOUBLE,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- Sample Data
INSERT INTO accounts (name, balance) VALUES ('Alice', 1000), ('Bob', 500);
```

Entity Classes

Account.java

```
import jakarta.persistence.*;
@Entity
@Table(name = "accounts")
public class Account {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;
    private double balance;

    // Getters and setters
}
```

TransactionRecord.java

```
import jakarta.persistence.*;
import java.time.LocalDateTime;
@Entity
@Table(name = "transactions")
public class TransactionRecord {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private int senderId;
    private int receiverId;
```

```
    private double amount;
    private LocalDateTime timestamp = LocalDateTime.now();
}
```

Spring Configuration

AppConfig.java

```
import org.springframework.context.annotation.*;
import org.springframework.orm.hibernate5.*;
import org.springframework.transaction.annotation.EnableTransactionManagement;
import javax.sql.DataSource;
import com.zaxxer.hikari.HikariDataSource;
import java.util.Properties;
@Configuration
@EnableTransactionManagement
@ComponentScan("com.example")
public class AppConfig {
    @Bean
    public DataSource dataSource() {
        HikariDataSource ds = new HikariDataSource();
        ds.setJdbcUrl("jdbc:mysql://localhost:3306/bank");
        ds.setUsername("root");
        ds.setPassword("your_password");
        ds.setDriverClassName("com.mysql.cj.jdbc.Driver");
        return ds;
    }
    @Bean
    public LocalSessionFactoryBean sessionFactory() {
        LocalSessionFactoryBean sf = new LocalSessionFactoryBean();
        sf.setDataSource(dataSource());
        sf.setPackagesToScan("com.example");
        Properties props = new Properties();
        props.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");
        props.setProperty("hibernate.hbm2ddl.auto", "update");
```

```
        props.setProperty("hibernate.show_sql", "true");
        sf.setHibernateProperties(props);
        return sf;
    }
    @Bean
    public HibernateTransactionManager txManager() {
        return new HibernateTransactionManager(sessionFactory().getObject());
    }
}
```

Service Layer

BankService.java

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

@Service
public class BankService {

    @Autowired
    private SessionFactory sessionFactory;

    @Transactional
    public void transfer(int senderId, int receiverId, double amount) {
        var session = sessionFactory.getCurrentSession();

        Account sender = session.get(Account.class, senderId);
        Account receiver = session.get(Account.class, receiverId);

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

```
        sender.setBalance(sender.getBalance() - amount);
        receiver.setBalance(receiver.getBalance() + amount);

        TransactionRecord tx = new TransactionRecord();
        tx.setSenderId(senderId);
        tx.setReceiverId(receiverId);
        tx.setAmount(amount);
        session.persist(tx);
    }
}
```

Main App

MainApp.java

```
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
public class MainApp {
    public static void main(String[] args) {
        var context = new AnnotationConfigApplicationContext(AppConfig.class);
        BankService service = context.getBean(BankService.class);

        try {
            service.transfer(1, 2, 300);  // ✅ Should succeed
            service.transfer(2, 1, 1000); // ❌ Should fail and rollback
        } catch (Exception e) {
            System.out.println("Transaction failed: " + e.getMessage());
        }
        context.close();
    }
}
```

Maven Dependencies

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
```

```
        <artifactId>spring-context</artifactId>
        <version>5.3.30</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>5.3.30</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.3.1.Final</version>
    </dependency>
    <dependency>
        <groupId>jakarta.persistence</groupId>
        <artifactId>jakarta.persistence-api</artifactId>
        <version>3.1.0</version>
    </dependency>
    <dependency>
        <groupId>com.zaxxer</groupId>
        <artifactId>HikariCP</artifactId>
        <version>5.1.0</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>8.3.0</version>
    </dependency>
</dependencies>
```

Output

```
inApp

ansfer 1 300 from 1 to 2 account 2 account 1 tron accuant 1
ansaction failed: Insufficient funds

ocess finished with exit code 0
```