# Experiment 3.2

## Java Applications Using Spring and Hibernate for Dependency Injection, CRUD Operations, and Transaction Management

## Part a: Dependency Injection in Spring Using Java-Based Configuration

**Objective:** To create a simple Spring application that demonstrates Dependency Injection (DI) using Java-based configuration.

**Explanation:** This demonstrates how to configure and inject dependencies in Spring using annotations like @Configuration and @Bean without XML files.

**Code:**

```
// Course.java
package com.example;

public class Course {
    private String courseName;

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

    public String getCourseName() {
        return courseName;
    }
}

// Student.java
package com.example;

public class Student {
    private Course course;

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

    public void displayInfo() {
        System.out.println("Student enrolled in: " + course.getCourseName());
    }
}

// AppConfig.java
package com.example;

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");
    }

    @Bean
    public Student student() {
        return new Student(course());
    }
}

// MainApp.java
package com.example;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(
        Student student = context.getBean(Student.class);
        student.displayInfo();
        context.close();
    }
}
```

**Sample Output:**
Student enrolled in: Spring Framework

## Part b: Hibernate Application for Student CRUD Operations

**Objective:** To perform CRUD operations on a Student entity using Hibernate ORM.

**Explanation:** This program demonstrates mapping of a Student class to a database table using Hibernate annotations and performing Create, Read, Update, and Delete operations.

**Code:**
```
// Student.java
package com.hibernate;

import javax.persistence.*;

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

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

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

    // Getters and 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 getMarks() { return marks; }
    public void setMarks(int marks) { this.marks = marks; }
}

// HibernateUtil.java
package com.hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
    private static SessionFactory factory;

    static {
        factory = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Studen
    }

    public static SessionFactory getFactory() {
        return factory;
    }
}

// MainCRUD.java
package com.hibernate;

import org.hibernate.Session;
import org.hibernate.Transaction;
import java.util.List;

public class MainCRUD {
    public static void main(String[] args) {
        Session session = HibernateUtil.getFactory().openSession();
        Transaction tx = session.beginTransaction();

        // Create
        Student s1 = new Student();
        s1.setName("Saksham");
        s1.setMarks(90);
        session.save(s1);

        // Read
        List<Student> students = session.createQuery("from Student", Student.class).list();
        for (Student s : students) {
            System.out.println(s.getId() + " - " + s.getName() + " - " + s.getMarks());
        }

        // Update
        Student student = session.get(Student.class, 1);
        student.setMarks(95);
        session.update(student);

        // Delete
        Student del = session.get(Student.class, 2);
        if (del != null) session.delete(del);

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

**Sample Output:**

```
1 - Saksham - 90
Student with ID 1 updated successfully.
```

## Part c: Transaction Management Using Spring and Hibernate

**Objective:** To create a banking system that demonstrates transaction consistency using Spring and Hibernate integration.

**Explanation:** This example shows how Spring’s @Transactional ensures atomicity in money transfers between accounts, rolling back changes in case of errors.

**Code:**

```
// Account.java
package com.bank;

import javax.persistence.*;

@Entity
@Table(name = "accounts")
public class Account {
    @Id
    private int id;
    private String name;
    private double balance;

    // Getters and Setters
}

// AccountDAO.java
package com.bank;

import org.hibernate.Session;
import org.springframework.stereotype.Repository;

@Repository
public class AccountDAO {
    public void updateAccount(Account acc) {
        Session session = HibernateUtil.getFactory().getCurrentSession();
        session.update(acc);
    }
}

// TransferService.java
package com.bank;

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

@Service
public class TransferService {
    @Autowired
    private AccountDAO dao;

    @Transactional
    public void transfer(Account from, Account to, double amount) {
        from.setBalance(from.getBalance() - amount);
        to.setBalance(to.getBalance() + amount);
        dao.updateAccount(from);
        dao.updateAccount(to);
    }
}

// MainApp.java
package com.bank;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(
        TransferService service = context.getBean(TransferService.class);
        Account a1 = new Account(); a1.setId(1); a1.setName("Saksham"); a1.setBalance(5000);
        Account a2 = new Account(); a2.setId(2); a2.setName("Rahul"); a2.setBalance(3000);
        service.transfer(a1, a2, 1000);
        System.out.println("Transfer successful! Updated balances recorded.");
        context.close();
    }
}
```

**Sample Output:**
Transfer successful! Updated balances recorded.