



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Experiment-9

**Student Name:** Komal Sharma

**Branch:** BE-CSE

**Semester:** 6th

**Subject Name:** Project Based Learning in Java

**UID:** 22BCS14520

**Section/Group:** 618-A

**Date of Performance:** 13/04/25

**Subject Code:** 22CSH-359

### EASY:

**Aim:** Create a simple Spring application that demonstrates Dependency Injection (DI) using Java-based configuration instead of XML. Define a Student class that depends on a Course class. Use Spring's `@Configuration` and `@Bean` annotations to inject dependencies.

**Objective:** To develop a simple Spring application that demonstrates Dependency Injection (DI) using Java-based configuration instead of XML. This experiment helps understand the concepts of Spring IoC (Inversion of Control), annotations (`@Configuration` and `@Bean`), and how to inject dependencies between classes using Java code.

### **Implementation/Code:**

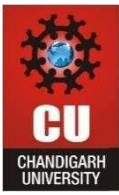
```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Course {

    private String courseName;

    private int duration;


    public Course(String courseName, int duration) {
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        this.courseName = courseName;  
        this.duration = duration;  
    }
```

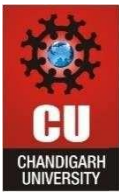
```
    public String getCourseName()  
    { return courseName;  
    }
```

```
    public int getDuration()  
    { return duration;  
    }  
}
```

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

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

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



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        System.out.println("Course: " + course.getCourseName() + ", Duration: " +  
course.getDuration() + " months");
```

```
    }
```

```
}
```

```
@Configuration
```

```
public class AppConfig {
```

```
    @Bean
```

```
    public Course course() {
```

```
        return new Course("Java Development", 6);
```

```
    }
```

```
    @Bean
```

```
    public Student student() {
```

```
        return new Student("Neha", course());
```

```
    }
```

```
}
```

```
public class MainApp {
```

```
    public static void main(String[] args)
```

```
    { ApplicationContext context = new  
AnnotationConfigApplicationContext(AppConfig.class);
```

```
        Student student = context.getBean(Student.class);
```

```
        student.displayDetails();
```

```
    }
```

}

**Output**

```
Student Name: Neha  
Course: Java Development,  
Duration: 6 months
```

**MEDIUM:**

**Aim:** Develop a Hibernate-based application to perform CRUD (Create, Read, Update, Delete) operations on a Student entity using Hibernate ORM with MySQL.

**Objective:** To create a Hibernate-based application that performs basic CRUD operations (Create, Read, Update, Delete) on a Student entity using Hibernate ORM with MySQL database integration. This experiment aims to give hands-on experience with Hibernate configuration, entity mapping, and using SessionFactory for database transactions.

**Code/Implementation:**

```
import javax.persistence.*;  
import org.hibernate.*;  
import org.hibernate.cfg.Configuration;
```

```
@Entity
```

```
@Table(name = "students")
```

```
public class Student {
```

```
    @Id
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
private int id;
```

```
private String name;
```

```
private int age;
```

```
// Constructors
```

```
public Student() {}
```

```
public Student(String name, int age)
```

```
    { this.name = name;
```

```
      this.age = age;
```

```
    }
```

```
// Getters & Setters
```

```
public int getId() { return id; }
```

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

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

```
public int getAge() { return age; }
```

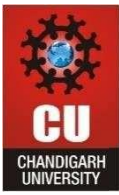
```
public void setAge(int age) { this.age = age; }
```

```
// Main method for CRUD operations
```

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

```
    // Configure SessionFactory
```

```
    SessionFactory factory = new Configuration()
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
.configure("hibernate.cfg.xml")  
.addAnnotatedClass(Student.class)  
.buildSessionFactory();
```

// Create

```
Session session = factory.openSession();  
Transaction tx = session.beginTransaction();  
Student s1 = new Student("Neha", 21);  
session.save(s1);  
tx.commit();  
System.out.println("Student saved: " + s1.getName());
```

// Read

```
session = factory.openSession();  
Student fetched = session.get(Student.class, s1.getId());  
System.out.println("Fetched Student: " + fetched.getName() + ", Age: " +  
fetched.getAge());
```

// Update

```
tx = session.beginTransaction();  
fetched.setAge(22);  
session.update(fetched);  
tx.commit();  
System.out.println("Updated Age to: " + fetched.getAge());
```

```
// Delete

tx = session.beginTransaction();

session.delete(fetched);

tx.commit();

System.out.println("Deleted Student with ID: " + fetched.getId());


session.close();

factory.close();

}

}
```

## HIBERNATE CRUD

```
import javax.persistence.*;

import org.hibernate.*;

import org.hibernate.cfg.Configuration;

@Entity
@Table(name = "students")

public class Student {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private int id;

    private String name;

    private int age;

    // Constructors
```

```
public Student() {}

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

// Getters & Setters

public int getId() { return 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; }

// Main method for CRUD operations

public static void main(String[] args) {

    // Configure SessionFactory

    SessionFactory factory = new Configuration()
        .configure("hibernate.cfg.xml")
        .addAnnotatedClass(Student.class)
        .buildSessionFactory();

    // Create

    Session session = factory.openSession();

    Transaction tx = session.beginTransaction();

    Student s1 = new Student("Neha", 21);

    session.save(s1);

    tx.commit();
}
```



```
System.out.println("Student saved: " + s1.getName());  
  
// Read  
  
session = factory.openSession();  
  
Student fetched = session.get(Student.class, s1.getId());  
  
System.out.println("Fetched Student: " + fetched.getName() + ", Age: " +  
fetched.getAge());  
  
// Update  
  
tx = session.beginTransaction();  
  
fetched.setAge(22);  
  
session.update(fetched);  
  
tx.commit();  
  
System.out.println("Updated Age to: " + fetched.getAge());  
  
// Delete  
  
tx = session.beginTransaction();  
  
session.delete(fetched);  
  
tx.commit();  
  
System.out.println("Deleted Student with ID: " + fetched.getId());  
  
session.close();  
  
factory.close();  
  
}  
  
}
```

**Output:**

```
into  
insert into students  
Student saved: ?  
  
Student saved: Neha  
update  
  a select - student0_id  
    student0_age as age2_0_0_  
  from students0_student_  
  where student0_id=?  
Updated Age to: 22  
  
Hibernate  
delete students
```

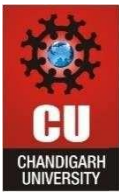
**HARD:**

**Aim:** Develop a Spring-based application integrated with Hibernate to manage transactions. Create a banking system where users can transfer money between accounts, ensuring transaction consistency.

**Objective:** To design and implement a Spring-based application integrated with Hibernate ORM that simulates a basic banking system, enabling users to securely transfer money between accounts. The system ensures data consistency and atomicity of transactions using Hibernate Transaction Management, with rollback mechanisms for failed transactions.

**Code/Implementation:**

```
import org.hibernate.Session;  
  
import org.hibernate.SessionFactory;  
  
import org.hibernate.Transaction;  
  
import org.hibernate.cfg.Configuration;  
  
import javax.persistence.*;  
  
import java.util.Date;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
// Entity: Account
```

```
@Entity
```

```
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;
```

```
    }
```

```
// Getters and Setters
```

```
    public int getId() { return id; }
```

```
    public String getHolderName() { return holderName; }
```

```
    public void setHolderName(String holderName) { this.holderName =  
holderName; }
```

```
    public double getBalance() { return balance; }
```

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

```
}
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
// Entity: Transaction Record
```

```
@Entity
```

```
class TransactionRecord
```

```
    { @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private int id;
```

```
    private int fromAccount;
```

```
    private int toAccount;
```

```
    private double amount;
```

```
    private Date transactionDate;
```

```
    public TransactionRecord() {}
```

```
    public TransactionRecord(int from, int to, double amount)
```

```
    { this.fromAccount = from;
```

```
      this.toAccount = to;
```

```
      this.amount = amount;
```

```
      this.transactionDate = new Date();
```

```
    }
```

```
// Getters
```

```
public int getId() { return id; }
```

```
public int getFromAccount() { return fromAccount; }
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public int getToAccount() { return toAccount; }  
public double getAmount() { return amount; }  
public Date getTransactionDate() { return transactionDate; }  
}  
  
// Main App with Configuration and Service  
public class BankingApp {  
  
    private static SessionFactory sessionFactory;  
  
    public static void main(String[] args)  
    { try {  
        // Configure Hibernate  
        Configuration cfg = new Configuration().configure("hibernate.cfg.xml")  
            .addAnnotatedClass(Account.class)  
            .addAnnotatedClass(TransactionRecord.class);  
        sessionFactory = cfg.buildSessionFactory();  
  
        // Add test data (Optional, only once)  
        //createDummyAccounts();  
  
        // Perform a money transfer  
        transferMoney(1, 2, 500);
```

```
} catch (Exception e)
    { e.printStackTrace();
} finally
    { sessionFactory.close();
}
}

public static void transferMoney(int fromId, int toId, double amount)
{ Session session = sessionFactory.openSession();
Transaction tx = null;

try {
    tx = session.beginTransaction();

    Account from = session.get(Account.class, fromId);
    Account to = session.get(Account.class, toId);

    if (from == null || to == null) {
        throw new RuntimeException("Account not found.");
    }

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

```
from.setBalance(from.getBalance() - amount);
```

```
to.setBalance(to.getBalance() + amount);
```

```
session.update(from);
```

```
session.update(to);
```

```
TransactionRecord txn = new TransactionRecord(fromId, toId, amount);
```

```
session.save(txn);
```

```
tx.commit();
```

```
System.out.println("Transaction Successful.");
```

```
} catch (Exception e) {
```

```
    if (tx != null) tx.rollback();
```

```
    System.out.println("Transaction Failed. Rolled back. Reason: " +  
e.getMessage());
```

```
} finally
```

```
    { session.close();
```

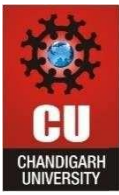
```
}
```

```
}
```

```
// Optional method to add initial accounts
```

```
public static void createDummyAccounts() {
```

```
    Session session = sessionFactory.openSession();
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
Transaction tx = session.beginTransaction();

session.save(new Account("Neha", 1000));
session.save(new Account("Riya", 1000));

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

System.out.println("Dummy accounts created.");
}
}
```

## HIBERNATE FILE

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/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/yourdb</property>
        <property name="hibernate.connection.username">root</property>
```



```
<property name="hibernate.connection.password">yourpassword</property>

<property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.show_sql">true</property>

<property name="hibernate.hbm2ddl.auto">update</property>

</session-factory>

</hibernate-configuration>
```

## Output:

```
Hibernate: update Account set balance=? where id=?
Transaction Successful.
```

## Learning Outcome:

1. We will be able to develop Spring and Hibernate applications with form handling and database connectivity.
2. We will learn to implement dynamic web applications for real-time data processing and display.