



### Experiment-9

**Student Name:** Pawan

**UID:** 22BCS10224

**Branch:** BE-CSE

**Section/Group:** KPIT-902(B)

**Semester:** 6<sup>th</sup>

**DOP:** 12/04/25

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

**Subject Code:** 22CSH-359

**9.1.1 Aim:** To demonstrate dependency injection using Spring Framework with Java-based configuration.

#### **9.1.2 Objective:**

Define Course and Student classes.

Use Configuration and Bean annotations to inject dependencies.

Load Spring context and print student details.

**9.1.3 Code:** // Course.java public  
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 + ",  
    Duration: " + duration;  
}
```

```
}
```

// Student.java

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

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

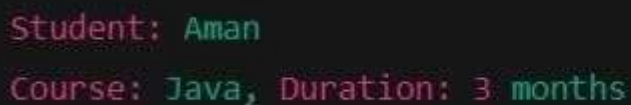
```
        public void showDetails() { System.out.println("Student: " +
            name); System.out.println(course); }
    } // AppConfig.java
import org.springframework.context.annotation.*;

@Configuration
public class AppConfig { @Bean public
    Course course() { return new
        Course("Java", "3 months"); }

    @Bean
    public Student student() { return new
        Student("Aman", course()); }
} // 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.showDetails();
    }
}
```

## Output:



```
Student: Aman
Course: Java, Duration: 3 months
```

**9.2.1 Aim:** To perform CRUD operations on a Student entity using Hibernate ORM with MySQL.

**Objective:** Define Course and Student classes.

Use Configuration and Bean annotations to inject dependencies. Load Spring context and print student details.

## 9.2.2 Code:

```
<hibernate-configuration>
    <session-factory>
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
<property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/testdb</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">password</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="Student"/>
</session-factory>
</hibernate-configuration> import
```

javax.persistence.\*;

Entity

```
public 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;
    }
    // Getters, setters, toString
}
import          org.hibernate.SessionFactory;          import
org.hibernate.cfg.Configuration;
```

```
public class HibernateUtil { private static final
    SessionFactory sessionFactory;

    static {
        sessionFactory = new Configuration().configure().buildSessionFactory();
    }

    public static SessionFactory getSessionFactory() { return
        sessionFactory;
    }
} import  org.hibernate.*;
```

public

```
class MainCRUD {
    public static void main(String[] args) {
        Session session = HibernateUtil.getSessionFactory().openSession();
```

```
// Create
Transaction tx = session.beginTransaction();
Student s1 = new Student("Aman", 22);
session.save(s1);
tx.commit();

// Read
Student student = session.get(Student.class, 1);
System.out.println(student);

// Update
tx = session.beginTransaction(); student.setAge(23);
session.update(student); tx.commit();

// Delete
tx = session.beginTransaction(); session.delete(student);
tx.commit();

session.close();
}
```

### 9.2.3 Output:

```
Student{id=1, name='    ', age=22}
Updated age to 23
Deleted student with id 1
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

9.3.1 Aim: To implement a banking system using Spring and Hibernate that ensures transaction consistency during fund transfers.

Objective:

Integrate Spring + Hibernate.

Handle transactions atomically (rollback on failure). Demonstrate success and failure cases.

Code:

```
import javax.persistence.*;
```

Entity

```
public class Account { @Id private
    int accountId; private String
    holderName; private double
    balance;
```

```
    // Constructors, getters, setters
}
```

```
import javax.persistence.*; import
java.util.Date;
```

@Entity

```
public class BankTransaction { @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY) private int
    txnId; private int fromAcc; private int toAcc; private double
    amount;
    private Date txnDate = new Date();
```

```
    // Constructors, getters, setters
}
```

```
import org.hibernate.*;
import org.springframework.transaction.annotation.Transactional;
```

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



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

@Transactional

```
public void transferMoney(int fromId, int toId, double amount) { Session session =  
    sessionFactory.getCurrentSession();
```

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

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

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

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

```
BankTransaction txn = new BankTransaction(fromId, toId, amount); session.save(txn);
```

```
}
```

```
}
```

@Configuration

@EnableTransactionManagement public

class AppConfig {

@Bean

```
public DataSource dataSource() {
```

```
    DriverManagerDataSource ds = new DriverManagerDataSource();  
    ds.setDriverClassName("com.mysql.cj.jdbc.Driver");  
    ds.setUrl("jdbc:mysql://localhost:3306/testdb"); ds.setUsername("root");  
    ds.setPassword("password");  
    return ds;
```

```
}
```

@Bean

```
public LocalSessionFactoryBean sessionFactory() { LocalSessionFactoryBean lsf = new  
    LocalSessionFactoryBean(); lsf.setDataSource(dataSource());  
    lsf.setPackagesToScan("your.package");
```

```
    Properties props = new Properties();  
    props.put("hibernate.dialect", "org.hibernate.dialect.MySQL8Dialect");  
    props.put("hibernate.hbm2ddl.auto", "update");  
    lsf.setHibernateProperties(props); return lsf;
```

```
}
```

@Bean



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public HibernateTransactionManager transactionManager(SessionFactory sf) { return  
new HibernateTransactionManager(sf); }
```

@Bean

```
public BankService bankService(SessionFactory sf) { return new  
BankService(sf);
```

```
}
```

```
}
```

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

```
AnnotationConfigApplicationContext ctx = new
```

```
AnnotationConfigApplicationContext(AppConfig.class);
```

```
BankService service = ctx.getBean(BankService.class);
```

```
try { service.transferMoney(101, 102, 500);
```

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

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

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

```
}
```

```
ctx.close();
```

```
}
```

```
}
```

## OUTPUT

```
Transaction Successful!
```

```
OR
```

```
Transaction Failed: Insufficient Balance
```

## Learning Outcomes

- Demonstrated Dependency Injection using Spring with Java-based configuration via @Bean and @Configuration.
- Performed CRUD operations on Student entity using Hibernate ORM with MySQL database.
- Integrated Spring + Hibernate for seamless object-relational mapping and dependency management.
- Implemented transaction management using @Transactional to ensure atomicity in fund transfers.



# **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

*Discover. Learn. Empower.*

- Handled transaction failures and rollbacks (e.g., insufficient balance) to maintain data consistency.