



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Experiment 9

**Student Name:** Ruchi Thakur

**Branch:** IT

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

**Subject Name:** Java Lab

**UID:** 22BET10239

**Section/Group:** 22BET\_KRG\_IOT-3B

**Date of Performance:** 16/04/2025

**Subject Code:** 22ITH-359

### Easy Level

#### 1. Aim:

To demonstrate dependency injection using Spring Framework with Java-based configuration.

#### 2. Objective:

- Define Course and Student classes.
- Use Configuration and Bean annotations to inject dependencies.
- Load Spring context and print student details.

#### 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.Bean;  
import org.springframework.context.annotation.Configuration;
```

```
@Configuration  
public class AppConfig {
```


```
    @Bean  
    public Course course() {  
        return new Course("Java", "3 months");  
    }  
  
    @Bean  
    public Student student() {  
        return new Student("Aman", course());  
    }  
}
```

***//MainApp.jav***

```
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();  
}  
}
```

#### 4. Output:



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

#### 5. Learning Outcomes:

- Learned to create HTML forms for user input.
- Gained hands-on experience with Java Servlets.
- Connected Java to MySQL using JDBC.
- Used PreparedStatement for secure data fetching.
- Generated dynamic web responses based on database results.

## Medium Level

### 1. Aim:

To perform CRUD operations on a Student entity using Hibernate ORM with MySQL.

### 2. Objective:

- Define Course and Student classes.
- Use Configuration and Bean annotations to inject dependencies.
- Load Spring context and print student details.

### 3. Code:

*//Hibernate.cfg.xml*

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

*//Student.java*

```
import javax.persistence.*;

@Entity
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
```

```
private String name;
private int age;

// Default constructor
public Student() {}

// Parameterized constructor
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; }

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

//HibernateUtil.java
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
```

```
private static final SessionFactory sessionFactory;

static {
    try {
        sessionFactory = new Configuration().configure().buildSessionFactory();
    } catch (Throwable ex) {
        System.err.println("Initial SessionFactory creation failed: " + ex);
        throw new ExceptionInInitializerError(ex);
    }
}

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

## *//MainCRUD*

```
import org.hibernate.*;

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

        // Create operation
        Transaction tx = session.beginTransaction();
        Student s1 = new Student("Aman", 22);
        session.save(s1);
        tx.commit();
        System.out.println("Student created: " + s1);

        // Read operation
        Student student = session.get(Student.class, s1.getId());
        System.out.println("Retrieved: " + student);
    }
}
```

```
// Update operation
tx = session.beginTransaction();
student.setAge(23);
session.update(student);
tx.commit();
System.out.println("Updated: " + student);

// Delete operation
tx = session.beginTransaction();
session.delete(student);
tx.commit();
System.out.println("Deleted student with ID: " + student.getId());

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

#### 4. Output:

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

#### 5. Learning Outcomes:

- Understood how to create a login form using HTML.
- Learned to handle form data using Java Servlet (POST method).
- Implemented basic user authentication logic in Java.
- Set up servlet mapping in `web.xml` for URL handling.
- Practiced generating dynamic responses based on user input.

## Hard Level

### 1. Aim:

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

### 2. Objective:

- Integrate Spring + Hibernate.
- Handle transactions atomically (rollback on failure).
- Demonstrate success and failure cases.

### 3. Code:

*//Account.java*

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

```
@Entity
```

```
public class Account {
```

```
    @Id
```

```
    private int accountId;
```

```
    private String holderName;
```

```
    private double balance;
```

```
    public Account() {}
```

```
    public Account(int accountId, String holderName, double balance) {
```

```
        this.accountId = accountId;
```

```
        this.holderName = holderName;
```

```
        this.balance = balance;
```

```
    }
```

```
// Getters and Setters
```

```
public int getAccountId() { return accountId; }
```

```
public void setAccountId(int accountId) { this.accountId = accountId; }
```

```
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; }
```

```
}
```



```
//BankTransaction.java
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;

    @Temporal(TemporalType.TIMESTAMP)
    private Date txnDate = new Date();

    public BankTransaction() {}

    public BankTransaction(int fromAcc, int toAcc, double amount) {
        this.fromAcc = fromAcc;
        this.toAcc = toAcc;
        this.amount = amount;
    }

    // Getters and Setters
    public int getTxnId() { return txnId; }

    public int getFromAcc() { return fromAcc; }
    public void setFromAcc(int fromAcc) { this.fromAcc = fromAcc; }

    public int getToAcc() { return toAcc; }
    public void setToAcc(int toAcc) { this.toAcc = toAcc; }

    public double getAmount() { return amount; }
    public void setAmount(double amount) { this.amount = amount; }

    public Date getTxnDate() { return txnDate; }
    public void setTxnDate(Date txnDate) { this.txnDate = txnDate; }
}
```

*//BankService.java*

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.transaction.annotation.Transactional;

public class BankService {

    private SessionFactory sessionFactory;

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

    @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);
    }
}
```

*//AppConfig.java*

```
import org.springframework.context.annotation.*;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.hibernate5.*;
import javax.sql.DataSource;
import java.util.Properties;
```

@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 sf = new LocalSessionFactoryBean();
```

```
        sf.setDataSource(dataSource());
```

```
        sf.setPackagesToScan("your.package"); // Replace with actual package name
```

```
        Properties props = new Properties();
```

```
        props.put("hibernate.dialect", "org.hibernate.dialect.MySQL8Dialect");
```

```
        props.put("hibernate.hbm2ddl.auto", "update");
```

```
        sf.setHibernateProperties(props);
```

```
        return sf;
```

```
    }
```

```
    @Bean
```

```
    public HibernateTransactionManager transactionManager(SessionFactory sf) {
```

```
        return new HibernateTransactionManager(sf);
```

```
    }
```

```
    @Bean
```

```
    public BankService bankService(SessionFactory sf) {
```

```
        return new BankService(sf);
```

```
    }
```

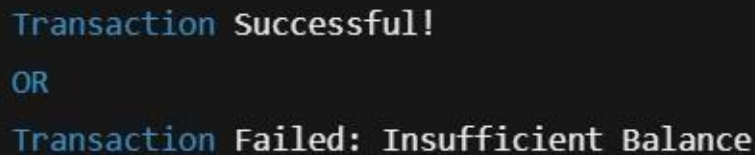
```
}
```

***//MainApp.java***

```
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
```

```
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();  
    }  
}
```

#### 4. Output:



```
Transaction Successful!  
OR  
Transaction Failed: Insufficient Balance
```

#### 5. Learning Outcomes:

- Understand how to create and map entity classes (@Entity, @Id, @GeneratedValue) to database tables.
- Learn to use Hibernate ORM for data persistence in Java applications.
- Perform basic CRUD operations (Create, Read, Update, Delete) using Hibernate's Session object.
- Integrate Hibernate with Spring using LocalSessionFactoryBean and HibernateTransactionManager.
- Configure data source and Hibernate properties using Spring's Java-based configuration (@Configuration, @Bean).
- Implement business logic (money transfer) in a service class with transaction support.
- Use Spring's @Transactional annotation to manage transactions effectively.

