JAVA SPRING FRAMEWORK

Lab Guides

Document Code	25e-BM/HR/HDCV/FSOFT
Version	1.0
Effective Date	01/05/2022

Hanoi, 08/2024

RECORD OF CHANGES

No	Effective Date	Change Description	Reason	Reviewer	Approver
1	06/08/2024	Create a new Lab	Create new		VinhNV

Contents

FRESHER ACADEMY	CODE:	JSFW_Lab_01_Opt4
	TYPE:	SHORT
	LOC:	200
	DURATION:	60 MINUTES

Java Spring Framework Introduction

Objectives:

- Use XML-based configuration for Spring beans.
- Understand dependency injection and autowiring in XML-based configuration.
- Configure and use beans with dependencies.

Lab Specifications:

Develop a simple Employee and Department Management System where departments and employees can be managed. This system will use XML-based configuration for Spring beans.

Problem Description:

- Trainees must write scripts to test the methods they have developed.

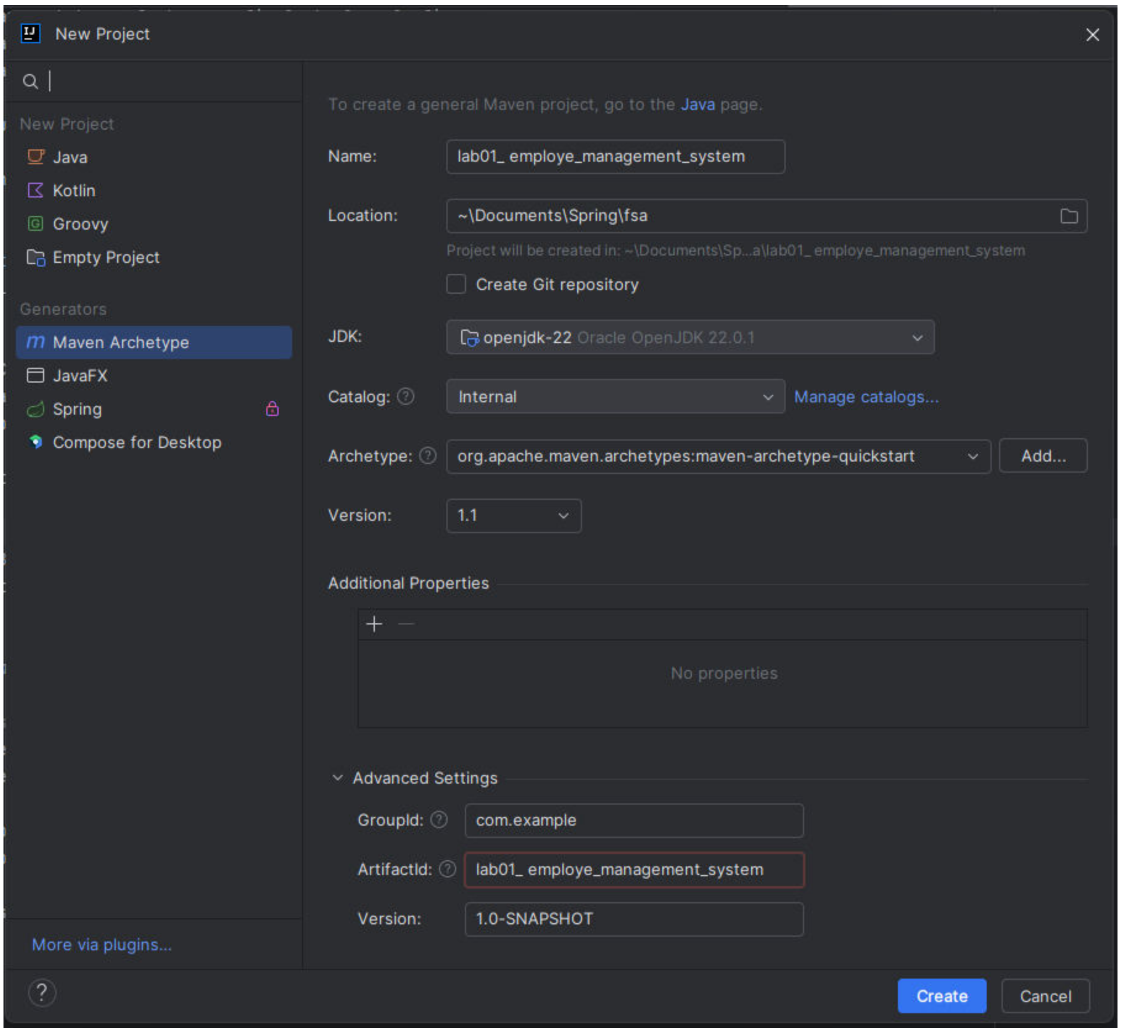
Prerequisites:

- Using Java SDK version 8.0 at least.
- Using Maven.
- Using Spring Framework 5.0 or higher version.

Guidelines:

Step 1: Extend the previous project to include dependency injection:

- Open IntelliJ IDEA.
- Click on File -> New -> Project....
- Select Maven from the project types.
- Click Next and set the project name to **lab01_employe_management_system**.
- Set the groupId to com.example, and artifactId to **lab01_ employe_management_system**.
- Click **Create**.



To create a general Maven project, go to the Java page.

Name: lab01_ employe_management_system
Location: ~\Documents\Spring\fsa
Project will be created in: ~\Documents\Sp...a\lab01_ employe_management_system
Create Git repository
JDK: openjdk-22 Oracle OpenJDK 22.0.1
Catalog: Internal
Archetype: org.apache.maven.archetypes:maven-archetype-quickstart
Version: 1.1
Additional Properties
No properties
Advanced Settings
GroupId: com.example
ArtifactId: lab01_ employe_management_system
Version: 1.0-SNAPSHOT

Step 2: Add dependencies and configuration into pom.xml file: Add the Spring Core dependency to your pom.xml file.

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.3.20</version>
</dependency>
```

Step 3: Create Entity Classes

1. Create Department class:

```
package com.example;


public class Department {
    private String deptName;

    // Getter and Setter
    public String getDeptName() {
        return deptName;
    }

    public void setDeptName(String deptName) {
        this.deptName = deptName;
    }
    @Override
    public String toString() {
        return "Department{deptName='" + deptName + "'}";
    }
}
```

2. Create **Address** class:

```
package com.example;

public class Address {
    private String street;
    private String city;
    private String zipCode;

    // Getters and Setters
    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getZipCode() {
        return zipCode;
    }

    public void setZipCode(String zipCode) {
        this.zipCode = zipCode;
    }

    @Override
    public String toString() {
        return "Address{street='" + street + "', city='" + city + "',
zipCode='" + zipCode + "'}";
    }
}
```

3. Create a Project class:

```
package com.example;

public class Project {
    private String projectName;

    // Getter and Setter
    public String getProjectName() {
        return projectName;
    }

    public void setProjectName(String projectName) {
        this.projectName = projectName;
    }

    @Override
    public String toString() {
        return "Project{projectName='" + projectName + "'}";
    }
}
```

4. Update **Employee** class with dependencies on **Department**, **Address**, and **Project**:

```
package com.example;

public class Employee {
    private int eid;
    private String ename;
    private Department department;
    private Address address;
    private Project project;

    // Getters and Setters
    public int getEid() {
        return eid;
    }

    public void setEid(int eid) {
        this.eid = eid;
    }

    public String getEname() {
        return ename;
    }

    public void setEname(String ename) {
        this.ename = ename;
    }

    public Department getDepartment() {
        return department;
    }

    public void setDepartment(Department department) {
        this.department = department;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    public Project getProject() {
        return project;
    }

    public void setProject(Project project) {
        this.project = project;
    }

    @Override
    public String toString() {
        return "Employee{eid=" + eid + ", ename='" + ename + "',
department=" + department + ", address=" + address + ", project=" +
project + "}";
    }
}
```

Step 4: Configure Beans in beans.xml

Create **beans.xml** configuration file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- Department bean -->
    <bean id="department" class="com.example.Department">
        <property name="deptName" value="Information Technology"/>
    </bean>

    <!-- Address bean -->
    <bean id="address" class="com.example.Address">
        <property name="street" value="123 Main St"/>
        <property name="city" value="Springfield"/>
        <property name="zipCode" value="12345"/>
    </bean>

    <!-- Project bean -->
    <bean id="project" class="com.example.Project">
        <property name="projectName" value="Spring Framework Project"/>
    </bean>

    <!-- Employee bean with autowiring by name -->
    <bean id="employee" class="com.example.Employee" autowire="byName">
        <property name="eid" value="100"/>
        <property name="ename" value="John Doe"/>
        <!-- No need to explicitly set department, address, and project
here, they will be autowired by name -->
    </bean>
</beans>
```

Step 5: The Main Class

The App class to load the context and use the Employee bean:

```java
package com.example;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");
        Employee employee = (Employee) context.getBean("employee");
        System.out.println(employee);
    }
}
```

Step 6: Write a JUnit Test Case

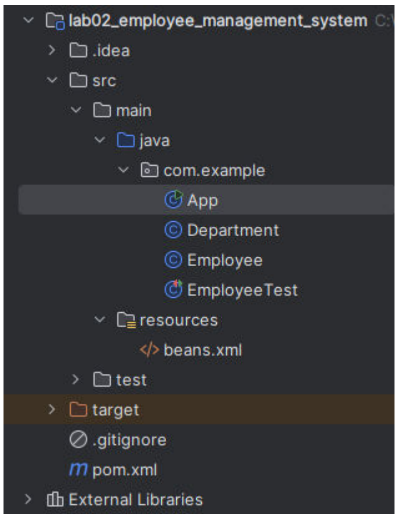
1. **Create EmployeeTest class:**

```
package com.example;


import org.junit.jupiter.api.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;

public class EmployeeTest {

    @Test
    public void testEmployeeBean() {
        ApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");
        Employee employee = (Employee) context.getBean("employee");
        assertNotNull(employee);
        assertEquals(100, employee.getEid());
        assertEquals("John Doe", employee.getEname());
        assertNotNull(employee.getDepartment());
        assertEquals("Information Technology",
employee.getDepartment().getDeptName());
    }
}
```

2. **Run the test and verify it passes.**

Notes:

- The autowire="byName" attribute in the employee bean configuration will automatically inject the Department, Address, and Project beans into the Employee bean based on the property names.
- Ensure that all Java files are located within the com.example package.



```
lab02_employee_management_system
  .idea
  src
    main
      java
        com.example
          App
          Department
          Employee
          EmployeeTest
      resources
        beans.xml
    test
  target
  .gitignore
  pom.xml
External Libraries
```

- Make sure the XML configuration file is correctly placed in the classpath so that it can be loaded by Spring.

This extended exercise will help you understand how to use XML-based configuration for beans, dependency injection, and autowiring in a Spring application.

----oOo-----

THE END