cnfsad-2 script for lab2

Lab Program - 2

Demonstrate Dependency Injection using constructor based using Spring Boot

Glossary:

- **Bean:** In Spring, a bean is an object that is managed by the Spring IoC (Inversion of Control) container. It is an instance of a Java class.
- **ID:** An identifier assigned to a bean within the Spring IoC container. It provides a unique reference for retrieving the bean.
- **Class:** The fully qualified name of the Java class that the Spring IoC container will instantiate to create a bean.
- **Scope:** Defines the lifecycle and visibility of a bean. Common scopes include "singleton" (one instance per container) and "prototype" (a new instance for each request).
- **Constructor-arg:** In Spring XML configuration, it is used to inject dependencies through a constructor. It specifies the arguments to be passed to the bean's constructor.
- **Name:** An attribute used in the Spring XML configuration to specify the name of a property or constructor argument.
- **Ref:** Stands for reference. It's used in bean configuration to refer to another bean by its ID or name.
- **Property:** A characteristic or attribute of a bean that is set during its instantiation. In Spring XML configuration, it is used to inject values into bean properties.

Step 1: Setting Up the Project on Spring Initializer:

Navigate to 'start.spring.io', choose Java, Spring Boot 3.2.2, and configure the project details. Hit generate and download the project zip. Import it into Eclipse.

```
# Example configuration:
Project: Maven
Language: Java
Spring Boot: 3.2.2 (SNAPSHOT)
Group: com.lab2
Artifact: my_lab2
Name: my_lab2
Description: Lab Program 2
Package name: com.lab2.my_lab2
```

```
Packaging: Jar
Java: 17
```

Step 2: Creating College and Department Classes:

In Eclipse, create two classes - `College` and `Department`.

```
// College.java
package com.lab2.my_lab2;

public class College {
    String college_name, college_address;

    // Getters and setters...
}

// Department.java
package com.lab2.my_lab2;

public class Department {
    String dept_name, dept_description;
    int dept_id;
    College college_instance;

    // Constructor-based dependency injection
    public Department(College college_instance) {
        this.college_instance = college_instance;
    }

    // Getters and setters...
}
```

Step 3: Writing XML Configuration for Spring Container:

Create or update `testBoot.xml` to define Spring beans for `Department` and `College` classes.

```
<?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
        https://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="department" class="com.lab2.my_lab2.Department"
scope="prototype">
```

```
        <constructor-arg name="college_instance" ref="college" />
    </bean>

    <bean id="college" class="com.lab2.my_lab2.College" scope="singleton">
        <property name="college_name" value="RV College of Engineering" />
        <property name="college_address" value="Mysuru Road, Bengaluru" />
    </bean>

</beans>
```

Step 4: Using Dependency Injection in the Main Class:

Initialize the Spring application context using `ClassPathXmlApplicationContext` and specify the name of the XML file.

```
// MyLab2Application.java
package com.lab2.my_lab2;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.util.Scanner;

@SpringBootApplication
public class MyLab2Application {

    public static void main(String[] args) {
        SpringApplication.run(MyLab2Application.class, args);

        Scanner sc = new Scanner(System.in);
        ApplicationContext ac = new
ClassPathXmlApplicationContext("testBoot.xml");
        Department d = (Department) ac.getBean("department");

        while (true) {

System.out.println("\n*****************************************");
            System.out.println("1. Insert Department details\n" +
                    "2. Display Department with College details \n" +
                    "3. Exit");

System.out.println("\n*****************************************");
```

```
            System.out.print("Enter your choice: ");
            int choice = sc.nextInt();

            switch (choice) {
                case 1:
                    System.out.println("\nInsert Department Details");
                    System.out.print(" - Enter Department Name: ");
                    d.setDept_name(sc.next());
                    System.out.print(" - Enter Department ID: ");
                    d.setDept_id(sc.nextInt());
                    System.out.print(" - Enter Department Description: ");
                    d.setDept_description(sc.next());
                    System.out.println("\nDetails inserted successfully");
                    break;
                case 2:
                    System.out.println("\nDepartment Details:");
                    System.out.println(" - Name: " + d.getDept_name());
                    System.out.println(" - ID: " + d.getDept_id());
                    System.out.println(" - Description: " +
d.getDept_description());
                    System.out.println("\nCollege Details");
                    College c = d.getCollege_instance();
                    System.out.println(" - College Name: " +
c.getCollege_name());
                    System.out.println(" - College Address: " +
c.getCollege_address());
                    break;
                case 3:
                    System.out.println("\nExiting...");
                    System.exit(0);
                default:
                    System.out.println("\nInvalid Choice");
                    break;
            }
        }
    }
}
```

Step 5: Property Injection and Scope Definition in XML Configuration:

Utilize the `<property>` element within each bean definition to inject properties. Set the scope of both beans to "prototype."

```xml
<bean id="department" class="com.lab2.my_lab2.Department" scope="prototype">
    <constructor-arg name="college_instance" ref="college" />
</bean>

<bean id="college" class="com.lab2.my_lab2.College" scope="singleton">
    <property name="college_name" value="RV College of Engineering" />
    <property name="college_address" value="Mysuru Road, Bengaluru" />
</bean>
```

Ensure that the property names specified in `<property>` match the actual property names in `College.java` and `Department.java`. The scope definition is crucial for Spring to manage dependency injection correctly.

Step 6: Property Injection:

Utilize the `<property>` element within each bean definition to inject properties. In this case, `College` has properties called `college_name` and `college_address`, which are set using property injection.

Step 7: Scope Definition:

Consider the scope of your beans. In this example, the `department` bean has a scope of "prototype," meaning a new instance is created each time it is requested. The `college` bean has a scope of "singleton," meaning a single instance is created and shared.

Ensure that:

- The id in the `<bean>` definition for `college_instance` should match the `ref` attribute in the `<constructor-arg>` definition inside the `department` bean.
- The name attribute in `<constructor-arg>` should match the variable name in the `Department.java` class.

Here, `college_instance` is the constructor argument name that we are referring to in the XML configuration and is crucial for Spring to correctly manage the constructor-based dependency injection.

Bean Retrieval:

Retrieve beans from the application context using their IDs.

```xml
Department d = (Department) ac.getBean("department");
College c = (College) ac.getBean("college");
```

The Spring application context is used to get instances of the `Department` and `College` beans.

The injected `Department` bean has a reference to the `College` bean, establishing the dependency.

Conclusion:

The provided program demonstrates Dependency Injection using constructor-based Spring Boot. It involves setting up the project, creating classes, configuring a Spring Container using XML, and utilizing Dependency Injection in the main class. Ensure consistency in IDs, package names, and class attributes for successful execution.
