Spring - Injecting Objects By Constructor Injection

Last Updated : 23 Jul, 2025

Spring IoC (Inversion of Control) Container is the core of Spring Framework. It creates the objects, configures and assembles their dependencies, and manages their entire life cycle. The Container uses Dependency Injection (DI) to manage the components that make up the application. It gets the information about the objects from a configuration file (XML) or Java Code or Java Annotations and Java POJO class. These objects are called Beans. Since the Controlling of Java objects and their lifecycle is not done by the developers, hence the name Inversion Of Control.

Some of the main features of Spring IOC include:

- **Object Creation**: It creates and manages objects (beans) for our application.
- **Object Management**: It handles the lifecycle of objects, including their initialization and destruction.
- **Configuration Management:** It makes our application highly configurable through XML, Java annotations, or Java-based configuration.
- **Dependency Management**: It manages dependencies between objects, ensuring that they are properly injected and decoupled.

Spring Dependency Injection

Dependency Injection is the main functionality provided by Spring IOC(Inversion of Control). The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods. The

design principle of Inversion of Control emphasizes keeping the Java classes independent of each other, and the container frees them from object creation and maintenance. These classes, managed by Spring, must adhere to the standard definition of Java-Bean. Dependency Injection in Spring also ensures loose coupling between the classes. There are two types of Spring Dependency Injection.

- Setter Dependency Injection (SDI)
- Constructor Dependency Injection (CDI)

In this article, we will focus on **Constructor Dependency Injection.**

Constructor Injection

In Constructor Injection, Dependencies are injected using the class constructor. To configure Constructor Dependency Injection in a bean, we use the <constructor-arg> tag in the bean configuration file (beans.xml).

Why Use Constructor Injection?

Constructor injection is a preferred approach in many cases because of the following reasons, which are listed below:

- **Immutability**: Dependencies are set at the time of object creation, making the object immutable.
- **Clear Dependencies:** It makes dependencies explicit and ensures that the object is fully initialized when created.
- **Testability**: It simplifies unit testing since dependencies can be easily injected via the constructor.

Steps to Inject Objects by Constructor Injection in Spring

Let’s consider an example where we have a Student class that depends on a MathCheat class. We will inject the MathCheat object into the Student class using Constructor Injection.

Step 1: Define the Classes

File: Student.java

```
public class Student {

    // Class data members
    private int id;
    private MathCheat mathCheat;

    // Constructor for Dependency Injection
    public Student(int id, MathCheat mathCheat) {
        this.id = id;
        this.mathCheat = mathCheat;
    }

    // Method to demonstrate functionality
    public void cheating() {
        System.out.println("My ID is: " + id);
        mathCheat.mathCheating();
    }
}
```

File: MathCheat.java

```
public class MathCheat {
    public void mathCheating() {
        System.out.println("And I Have Started Math Cheating");
    }
}
```

Step 2: Configure Bean in beans.xml

We need to configure the beans in the beans.xml file. Here, we will define the Student bean and inject the MathCheat dependency using the <constructor-arg> tag.

File: beans.xml

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

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

    <!-- Define MathCheat bean -->
    <bean id="mathCheatObjectValue" class="MathCheat"></bean>

    <!-- Define Student bean with Constructor Injection -->
    <bean id="student" class="Student">
        <constructor-arg name="id" value="101"/>
        <constructor-arg name="mathCheat" ref="mathCheatObjectValue"/>
    </bean>

</beans>
```

Explanation: The MathCheat bean is defined first with the ID mathCheatObjectValue. The Student bean is defined with two <constructor-arg> tags which is listed below:

- <constructor-arg> injects the MathCheat bean using the ref attribute.

Note: The ref attribute is the recommended approach, instead of defining the MathCheat bean inside the Student bean, we define it separately and reference it.

Step 3: Create the Main Application Class

Create a Main class to test the setup. We will load the Spring context, retrieve the Student bean, and then call the cheating() method.

File: Main.java

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

public class Main {

    public static void main(String[] args) {

        // Load the Spring application context
        ApplicationContext context
        = new ClassPathXmlApplicationContext("beans.xml");

        // Retrieve the Student bean
        Student student = context.getBean("student", Student.class);

        // Call the cheating() method
        student.cheating();
```

```
    }
}
```

Output: When we run the Main class, we will see the following output:

```
My ID is: 101
And I Have Started Math Cheating
```

Alternative Approach (Not Recommended)

We can also define the MathCheat bean directly inside the Student bean but it is not the best practice because of the following reasons which are listed below

- The MathCheat bean is tightly coupled to the Student bean.
- We cannot reuse the MathCheat bean elsewhere in your application.
- It makes the configuration less readable and harder to maintain.

```
<bean id="student" class="Student">
    <constructor-arg name="id" value="101"/>
    <constructor-arg name="mathCheat">
        <bean class="MathCheat"></bean>
    </constructor-arg>
</bean>
```

Step 4: Using Annotation-Based Configuration

Modern Spring applications often use annotation-based configuration instead of XML. Let's see how to configure the same example using annotations.

1. Enable Component Scanning: Add the @ComponentScan annotation to a configuration class to enable component scanning.

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

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

@Configuration
@ComponentScan(basePackages = "com.example")
public class AppConfig {
}
```

2. Annotate Classes with @Component: Annotate the Student and MathCheat classes with @Component.

```
import org.springframework.stereotype.Component;

@Component
public class Student {

    private int id;
    private MathCheat mathCheat;

    // Constructor for Dependency Injection
    public Student(int id, MathCheat mathCheat) {
        this.id = id;
        this.mathCheat = mathCheat;
    }

    public void cheating() {
        System.out.println("My ID is: " + id);
        mathCheat.mathCheating();
    }
}

@Component
public class MathCheat {
    public void mathCheating() {
        System.out.println("And I Have Started Math Cheating");
    }
}
```

3. Inject the Dependency Using @Autowired: Update the Student class to use @Autowired for constructor injection.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Student {

    private int id;
    private MathCheat mathCheat;
```

```
    // Constructor for Dependency Injection
    @Autowired
    public Student(int id, MathCheat mathCheat) {
        this.id = id;
        this.mathCheat = mathCheat;
    }

    public void cheating() {
        System.out.println("My ID is: " + id);
        mathCheat.mathCheating();
    }
}
```

4. Create the Main Application Class: Update the Main class to use annotation-based configuration.

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

public class Main {
    public static void main(String[] args) {
        // Load the Spring application context
        // using annotation-based configuration
        ApplicationContext context
        = new AnnotationConfigApplicationContext(AppConfig.class);

        // Retrieve the Student bean
        Student student = context.getBean(Student.class);

        // Call the cheating() method
        student.cheating();
    }
}
```

Output:

```
My ID is: 101
And I Have Started Math Cheating
```

Conclusion

In this article, we explored Constructor Injection in Spring and demonstrated its implementation using:

1. XML Configuration
2. Annotation-Based Configuration

Constructor Dependency Injection in Spring with Example

GeeksforGeeks
Sanchhaya Education Private Limited

Corporate & Communications Address:

A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:

K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305