Spring Constructor-Based Dependency Injection

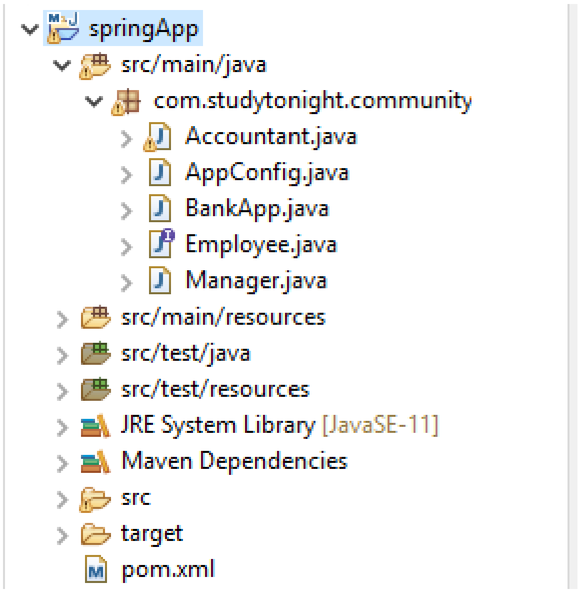
In this topic, we are using the constructor-based dependency injection technique to inject values through the constructor but before moving further let's first understand what is Dependency Injection(DI).

Dependency Injection is a technique by which an object defines its dependencies. The IOC container then injects these dependencies during bean creation. This process is fundamentally the inverse and known as Inversion of Control as well. Dependency Injection makes our code loosely coupled. It is classified into two major categories Constructor-based dependency injection and Setter-based dependency injection. Here, we will discuss Constructor-based DI with an example.

We created a Maven-based Spring Project and that contains the following files.

- BankApp.java
- AppConfig.java
- Employee.java
- Manager.java
- Accountant.java
- pom.xml

And the following is a maven project structure created for the Spring application. Project Structure:



```
springApp
  src/main/java
    com.studytonight.community
      Accountant.java
      AppConfig.java
      BankApp.java
      Employee.java
      Manager.java
  src/main/resources
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-11]
  Maven Dependencies
  src
  target
  pom.xml
```

Spring Constructor-Based Dependency Injection

The files created into the above project contains the following code. See the files below.

Files Source Code:

// BankApp.java

This file contains the code to create an IOC container for our application. The AnnotationConfigApplicationContext class is used to create an object for application context.
package com.studytonight.community;

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

public class BankApp {

	public static void main(String[] args)
	{
		AnnotationConfigApplicationContext context = new 
AnnotationConfigApplicationContext(AppConfig.class);
		Manager manager = context.getBean(Manager.class);
		manager.callMetting();
		context.close();
	}
}
```

Spring Constructor-Based Dependency Injection

// AppConfig.java

This is a configuration file in Java which is an alternate of the applicationContext.xml file that we created for the XML-based configuration example. The @Configuration annotation indicates that this is not a simple class but a configuration class and the @ComponentScan annotation is used to indicate the component location in our spring project.

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

@Configuration
@ComponentScan("com.studytonight.community")
public class AppConfig
{
}
```

// Employee.java

This is an interface Employee that contains a doWork() abstract method. Each class that implements this interface will have to override the doWork() method.

```
package com.studytonight.community;

public interface Employee
{

        void doWork();

}
```

Spring Constructor-Based Dependency Injection

// Accountant.java

This is a component class that is marked using @Component annotation. It implements the Employee interface and overrides its method doWork().

```
import org.springframework.stereotype.Component;
@Component
public class Accountant implements Employee
{

        public Accountant()
        {
                System.out.println("Inside Accountant Constructor");
        }
        public void doWork()
        {
                System.out.println("Audit the accounts...");
        }
}
```

Spring Constructor-Based Dependency Injection

// Manager.java

This is another component class that is marked using the @Component annotation and implements the Employee interface. In this class, we are implementing constructor-based dependency injection. See, the Manager class calls a method of Accountant class by using the Accountant class object which is instantiated inside the Manager class constructor. See the example below.

```
package com.studytonight.community;
import org.springframework.stereotype.Component;

@Component
public class Manager implements Employee
{

	Accountant accountant;

	public Manager(Accountant accountant)
	{
		System.out.println("manager constructor");
		this.accountant = accountant;
	}
	public void doWork()
	{
		System.out.println("Manage the branch office");
	}

	public void callMetting()
	{
		accountant.doWork();
	}
}
```

Spring Constructor-Based Dependency Injection

// pom.xml

This file contains all the dependencies of this project such as spring jars, servlet jars, etc. Put these dependencies into your project to run the application.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.studytonight</groupId>
  <artifactId>springApp</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
		<!--
https://mvnrepository.com/artifact/org.springframework/spring-web -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-core</artifactId>
			<version>${spring.version}</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>${spring.version}</version>
		</dependency>
		<dependency>
			<groupId>javax.annotation</groupId>
			<artifactId>javax.annotation-api</artifactId>
			<version>1.3.2</version>
		</dependency>
	</dependencies>
	<properties>
		<spring.version>5.2.8.RELEASE</spring.version>
	</properties>
	<build>
		<sourceDirectory>src</sourceDirectory>
		<plugins>
			<plugin>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.8.1</version>
				<configuration>
```

```
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>

    </build>
</project>
```

Run the Application

After successfully completing the project and adding the dependencies run the application and you will get the output as below.

```
Inside Accountant Constructor

manager constructor

Audit the accounts...
```

Configuration using XML

The above project is configured using Java code only. No XML configuration did there but we can configure it with XML code as well. We just need to create a file applicationContext.xml and read it into the BankApp class. The applicationContext.xml file contains the following code.

Spring Constructor-Based Dependency Injection

// applicationContext.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"
        xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

        <bean id="accountant"
                class="com.studytonight.community.Accountant" />
        <bean id="manager" class="com.studytonight.community.Manager">
                <constructor-arg>
                        <ref bean="accountant" />
                </constructor-arg>
        </bean>
</beans>
```

Injecting Primitive Values into Constructor

Apart from the reference variable, we can inject primitive values like int, float, etc into the constructor. For example, In the Manager class, we are using the int id and string name inside the constructor and injecting values from the applicationContext.xml file.

Spring Constructor-Based Dependency Injection

// Manager.java

```
package com.studytonight.community;
import org.springframework.stereotype.Component;

@Component
public class Manager implements Employee{
	int id;
	String name;

	public Manager(int id, String name) {
		this.id = id;
		this.name = name;
	}
	public void doWork() {
		System.out.println("Manage the branch office");
	}

	public void managerInfo() {
		System.out.println("Name: "+name+" Id: "+id);
	}
}
```

Spring Constructor-Based Dependency Injection

// applicationContext.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"
        xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

        <bean id="manager" class="com.studytonight.community.Manager">
                <constructor-arg type="int" value="10021" />
                <constructor-arg type="java.lang.String" value="Ramesh" />
        </bean>
</beans>
```

Run the Application

After successfully updating these two files into the project run the application and you will get the output as below.

```
Name: Ramesh Id: 10021
```

Specify Constructor Argument Name

We can also use the constructor parameter name for value disambiguation, as we did in the below example.

```
<bean id="manager" class="com.studytonight.community.Manager">
    <constructor-arg name="id" value="10021"/>
    <constructor-arg name="name" value="Ramesh"/>
</bean>
```