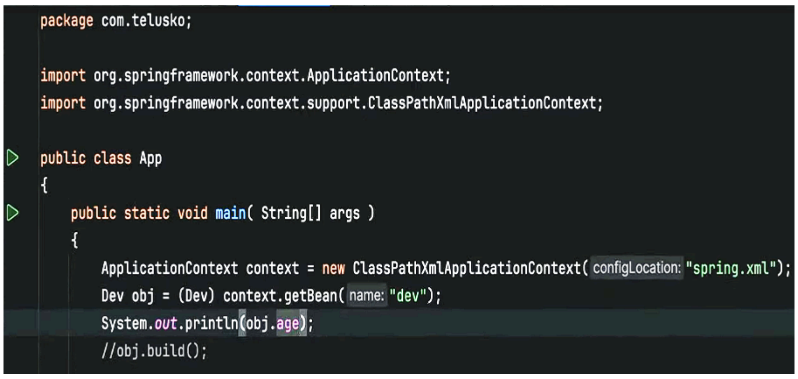
Constructor & Setter Injection in Spring

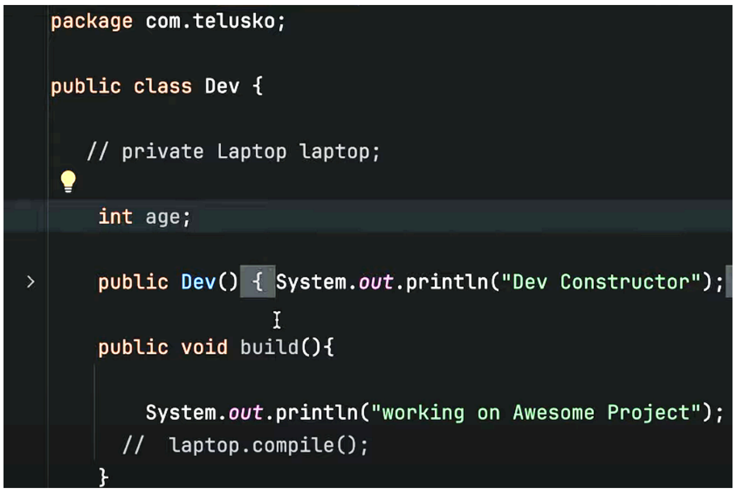
Along with finishing the XML configuration, we also learned the syntax for the tag. Let's now examine how the spring framework handles object reference injection.
If we use the conventional approach, for example, and call the laptop class's reference inside the development class, simply invoking the method through reference, we will eventually receive an error with the code NPE (NullPointerException) because it is declared at the instance level and the object has not yet been assigned memory.

Setter Injection:

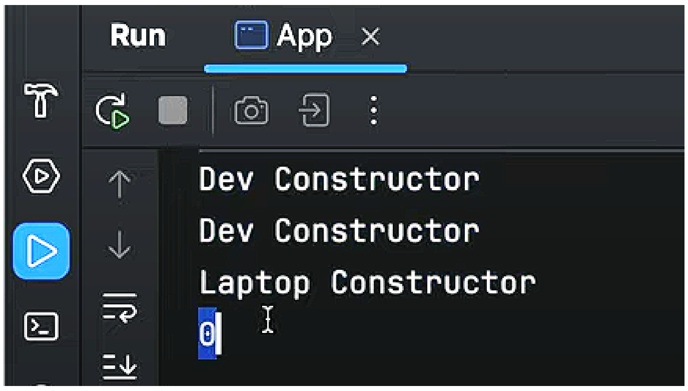


```java
package com.telusko;

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

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context = new ClassPathXmlApplicationContext( configLocation: "spring.xml");
        Dev obj = (Dev) context.getBean( name: "dev");
        System.out.println(obj.age);
        //obj.build();
```

Using setter or getter methods is another method for injecting dependencies. It's crucial to remember that dependencies are only injected when the setter and getter methods are called. To illustrate how setter injection functions in a traditional manner, let's take an example. We have created an int variable called age, and when we print its value in the MainApp, it will have the default value of 0, since we haven't assigned any value to it.



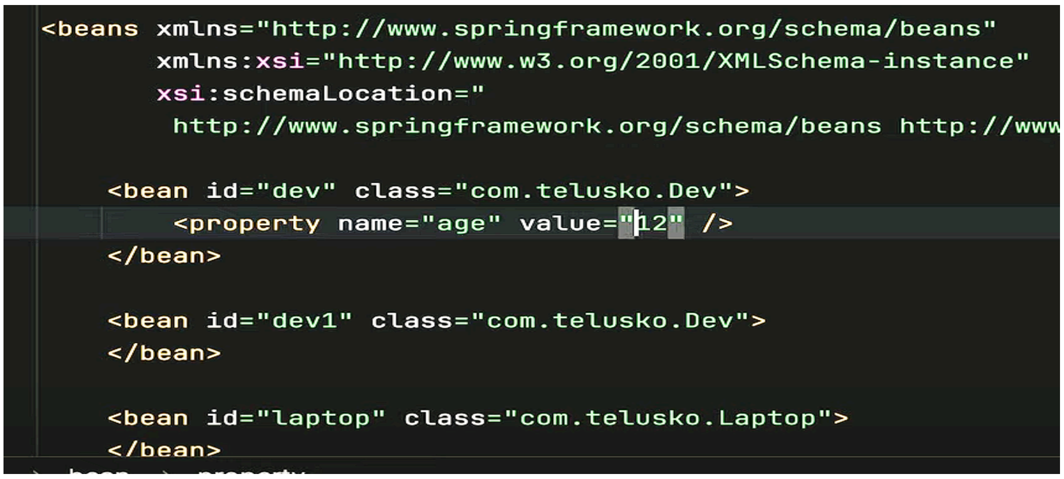
```java
package com.telusko;

public class Dev {

   // private Laptop laptop;

    int age;

    public Dev() { System.out.println("Dev Constructor");

    public void build(){

        System.out.println("working on Awesome Project");
      //  laptop.compile();
    }
```



```
Dev Constructor
Dev Constructor
Laptop Constructor
0
```

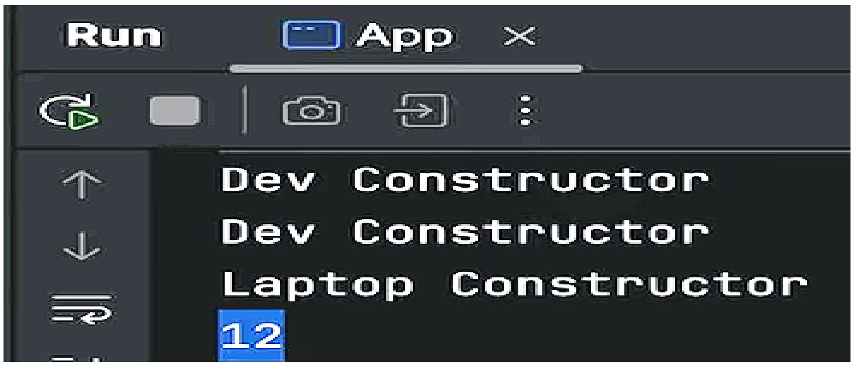
We haven't made the variable private yet, which is not a good idea. Let's do that now. Next, to get to the age value in our main app, we need setters and getters. We can do this in code, but how do we tell Spring through XML? Let's look.
So, there is an inbuilt tag called that tells us which property of that bean we want to change.

In this case, we want to change the value of the age property, so this is how it should be written:
:



```
<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

    <bean id="dev" class="com.telusko.Dev">
        <property name="age" value="12" />
    </bean>

    <bean id="dev1" class="com.telusko.Dev">
    </bean>

    <bean id="laptop" class="com.telusko.Laptop">
    </bean>
```

Basically, here in the above xml file, we are setting the value of the property age by using spring as **12.**



```
Dev Constructor
Dev Constructor
Laptop Constructor
12
```

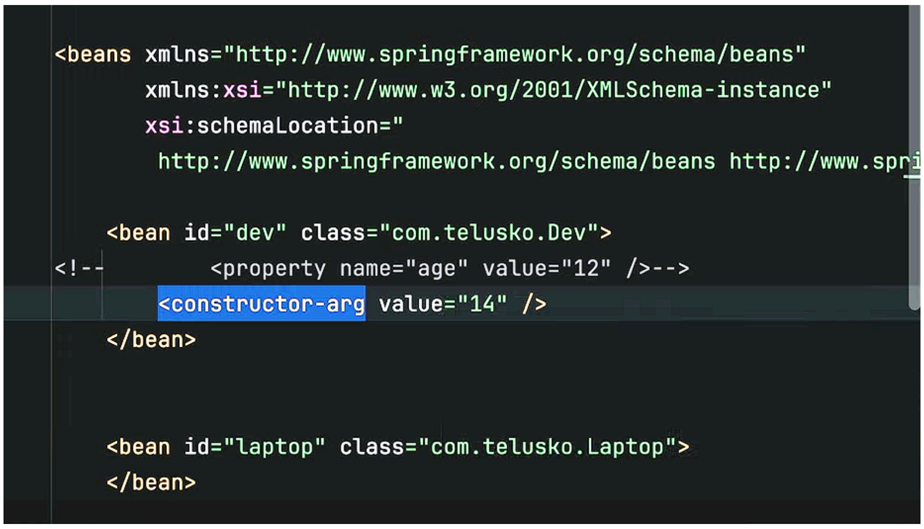
So, in this way, **the setter injection** is performed, and now let’s see the **constructor injection**.

Constructor Injection:

Constructor injection refers to a method of dependency injection where dependencies are provided to a class through its constructor.
In this approach, dependencies are provided as an argument to the constructor of the class. This guarantees that all the dependencies are provided when an object is instantiated.

In the Dev class, let's create a parameterized constructor in which we are passing age as the parameter, and in XML we will be using the **<constructor-arg>** tag for the Dev class <bean> tag.

For the variable, we will assign value '**14 to** it through tag by writing the following code

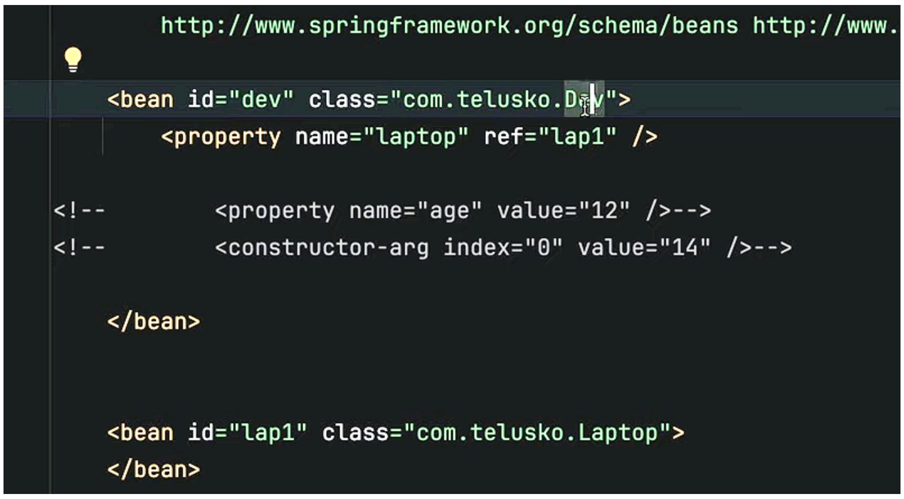


```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.spri

    <bean id="dev" class="com.telusko.Dev">
<!--        <property name="age" value="12" />-->
        <constructor-arg value="14" />
    </bean>


    <bean id="laptop" class="com.telusko.Laptop">
    </bean>
```

inside the xml file:

We used the name and value attributes on the property tag because there is only one parameter in this case. When defining a bean in Spring, the constructor-arg element can take either a value or a reference to another bean. You can also clear up any confusion or ambiguity with the **type** and **index** attributes, especially when a constructor takes more than one argument of the same type**. The name** attribute could also be used to match variables from XML to Java.

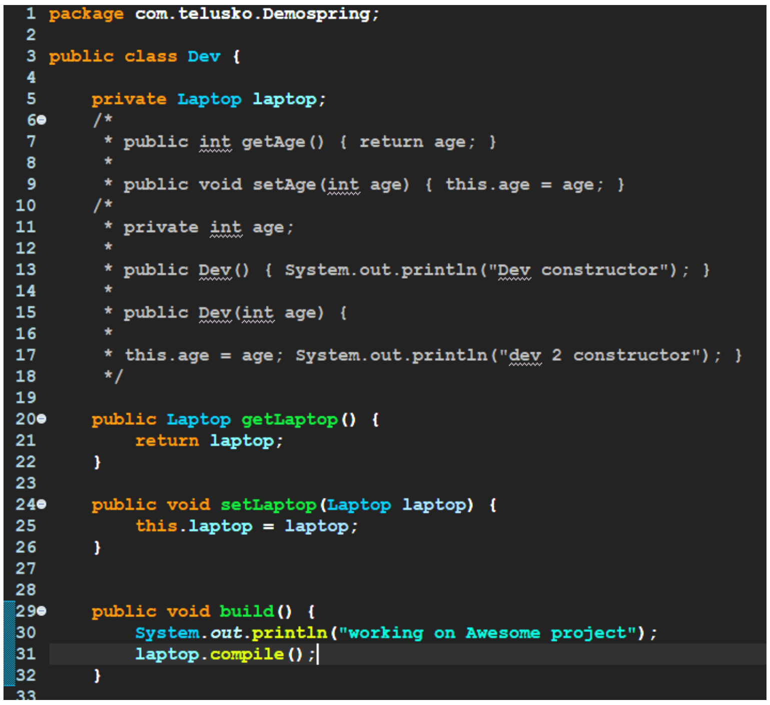
One more thing that can be used is references to other beans. For example, in the Dev class, we are calling the laptop reference. Since laptop is a reference variable, we can't directly assign a value to it. Instead, we'll use a different attribute in the property tag called "ref" to show that it is a reference, with bean id lap1. This way, we can connect the laptop reference to the dev class so that the build method of Dev can call the method compile.



```
        http://www.springframework.org/schema/beans http://www.

    <bean id="dev" class="com.telusko.Dev">
        <property name="laptop" ref="lap1" />

<!--        <property name="age" value="12" />-->
<!--        <constructor-arg index="0" value="14" />-->

    </bean>


    <bean id="lap1" class="com.telusko.Laptop">
    </bean>
```

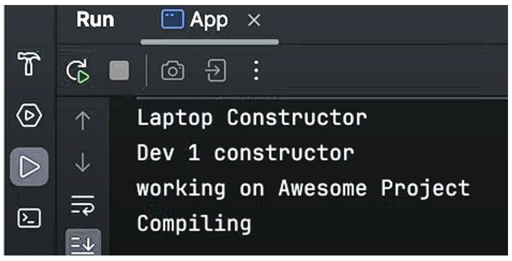
.

Now we will be injecting the laptop object, which is already inside the IOC container, by using setter injection. For that, we have used property tags, and now let's see the constructor injection for the reference variable.



```java
package com.telusko.Demospring;

public class Dev {

    private Laptop laptop;
    /*
     * public int getAge() { return age; }
     *
     * public void setAge(int age) { this.age = age; }
    /*
     * private int age;
     *
     * public Dev() { System.out.println("Dev constructor"); }
     *
     * public Dev(int age) {
     *
     * this.age = age; System.out.println("dev 2 constructor"); }
     */

    public Laptop getLaptop() {
        return laptop;
    }

    public void setLaptop(Laptop laptop) {
        this.laptop = laptop;
    }


    public void build() {
        System.out.println("working on Awesome project");
        laptop.compile();
    }
```

Output:

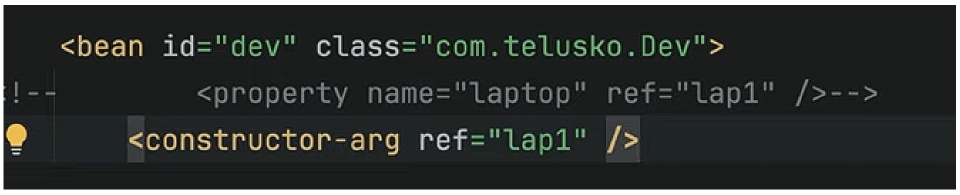


```
Laptop Constructor
Dev 1 constructor
working on Awesome Project
Compiling
```

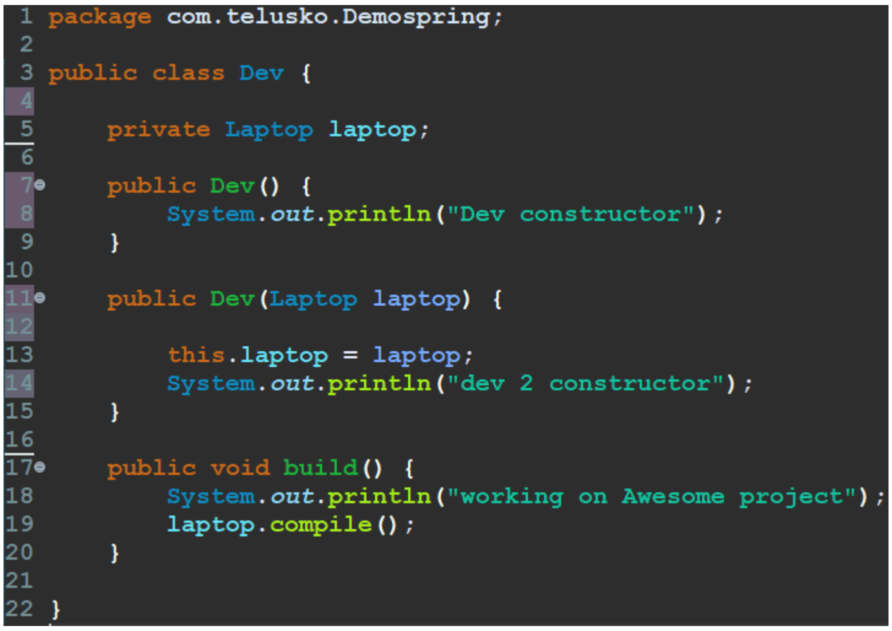
Constructor Injection for References:

For constructor injection, we will use the **<constructor-arg>** tag and also create the parameterized constructor by passing laptop as an argument to it, same as we did in the case of the age example.

Syntax for the xml file:



```xml
<bean id="dev" class="com.telusko.Dev">
<!--        <property name="laptop" ref="lap1" />-->
        <constructor-arg ref="lap1" />
```



```java
package com.telusko.Demospring;

public class Dev {

    private Laptop laptop;

    public Dev() {
        System.out.println("Dev constructor");
    }

    public Dev(Laptop laptop) {

        this.laptop = laptop;
        System.out.println("dev 2 constructor");
    }

    public void build() {
        System.out.println("working on Awesome project");
        laptop.compile();
    }

}
```

When to use constructor injection and setter injection?

It depends upon when we need an object with all of its dependencies along with it that its better to go with the **constructor injection,** and if that object has dependencies that are optional and not required for the moment but may need in the future for this kind of dependency, we use **setter injection.**

By understanding these injection methods, you can effectively manage dependencies in your Spring applications, ensuring robust and maintainable code.

In the next chapter, we will learn about autowiring in the Spring Framework.

Happy learning 😊