SPRING CONSTRUCTOR-BASED DEPENDENCY INJECTION

Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.

Example:

The following example shows a class *TextEditor* that can only be dependency-injected with constructor injection.

Let us have working Eclipse IDE in place and follow the following steps to create a Spring application:

Step	Description
1	Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project.
2	Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter.
3	Create Java classes *TextEditor*, *SpellChecker* and *MainApp* under the *com.tutorialspoint* package.
4	Create Beans configuration file *Beans.xml* under the **src** folder.
5	The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below.

Here is the content of **TextEditor.java** file:

```
package com.tutorialspoint;

public class TextEditor {
   private SpellChecker spellChecker;

   public TextEditor(SpellChecker spellChecker) {
      System.out.println("Inside TextEditor constructor." );
      this.spellChecker = spellChecker;
   }
   public void spellCheck() {
      spellChecker.checkSpelling();
   }
}
```

Following is the content of another dependent class file **SpellChecker.java**:

```
package com.tutorialspoint;

public class SpellChecker {
   public SpellChecker(){
      System.out.println("Inside SpellChecker constructor." );
   }

   public void checkSpelling() {
      System.out.println("Inside checkSpelling." );
   }

}
```

Following is the content of the **MainApp.java** file:

```
package com.tutorialspoint;

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

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      TextEditor te = (TextEditor) context.getBean("textEditor");

      te.spellCheck();
   }
}
```

Following is the configuration file **Beans.xml** which has configuration for the constructor-based injection:

```
<?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-3.0.xsd">

   <!-- Definition for textEditor bean -->
   <bean  >
      <constructor-arg ref="spellChecker"/>
   </bean>

   <!-- Definition for spellChecker bean -->
   <bean  >
   </bean>

</beans>
```

Once you are done with creating source and bean configuration files, let us run the application. If everything is fine with your application, this will print the following message:

```
Inside SpellChecker constructor.
Inside TextEditor constructor.
Inside checkSpelling.
```

Constructor arguments resolution:

There may be a ambiguity exist while passing arguments to the constructor in case there are more than one parameters. To resolve this ambiguity, the order in which the constructor arguments are defined in a bean definition is the order in which those arguments are supplied to the appropriate constructor. Consider the following class:

```
package x.y;

public class Foo {
   public Foo(Bar bar, Baz baz) {
      // ...
   }
}
```

The following configuration works fine:

```
<beans>
   <bean  >
      <constructor-arg ref="bar"/>
```

```
      <constructor-arg ref="baz"/>
   </bean>

   <bean  />
   <bean  />
</beans>
```

Let us check one more case where we pass different types to the constructor. Consider the following class:

```
package x.y;

public class Foo {
   public Foo(int year, String name) {
      // ...
   }
}
```

The container can also use type matching with simple types if you explicitly specify the type of the constructor argument using the type attribute. For example:

```
<beans>

   <bean  >
      <constructor-arg type="int" value="2001"/>
      <constructor-arg type="java.lang.String" value="Zara"/>
   </bean>

</beans>
```

Finally and the best way to pass constructor arguments, use the index attribute to specify explicitly the index of constructor arguments. Here the index is 0 based. For example:

```
<beans>

   <bean  >
      <constructor-arg index="0" value="2001"/>
      <constructor-arg index="1" value="Zara"/>
   </bean>

</beans>
```

A final note, in case you are passing a reference to an object, you need to use **ref** attribute of <constructor-arg> tag and if you are passing a value directly then you should use **value** attribute as shown above.