SPRING @AUTOWIRED ANNOTATION

The **@Autowired** annotation provides more fine-grained control over where and how autowiring should be accomplished. The @Autowired annotation can be used to autowire bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

@Autowired on Setter Methods:

You can use **@Autowired** annotation on setter methods to get rid of the <property> element in XML configuration file. When Spring finds an @Autowired annotation used with setter methods, it tries to perform **byType** autowiring on the method.

Example

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;

import org.springframework.beans.factory.annotation.Autowired;

public class TextEditor {
   private SpellChecker spellChecker;

   @Autowired
   public void setSpellChecker( SpellChecker spellChecker ){
      this.spellChecker = spellChecker;
   }
   public SpellChecker getSpellChecker( ) {
      return 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**:

```
<?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:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

   <!-- Definition for textEditor bean without constructor-arg  -->
   <bean  >
   </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 checkSpelling.
```

@Autowired on Properties:

You can use **@Autowired** annotation on properties to get rid of the setter methods. When you will pass values of autowired properties using <property> Spring will automatically assign those properties with the passed values or references. So with the usage of @Autowired on properties your **TextEditor.java** file will become as follows:

```
package com.tutorialspoint;
```

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

public class TextEditor {
   @Autowired
   private SpellChecker spellChecker;

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

   public SpellChecker getSpellChecker( ){
      return spellChecker;
   }

   public void spellCheck(){
      spellChecker.checkSpelling();
   }
}
```

Following is the configuration file **Beans.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:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

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

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

</beans>
```

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

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

@Autowired on Constructors:

You can apply @Autowired to constructors as well. A constructor @Autowired annotation indicates that the constructor should be autowired when creating the bean, even if no <constructor-arg> elements are used while configuring the bean in XML file. Let us check the following example.

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

```
package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Autowired;

public class TextEditor {
   private SpellChecker spellChecker;

   @Autowired
   public TextEditor(SpellChecker spellChecker){
```

```
      System.out.println("Inside TextEditor constructor." );
      this.spellChecker = spellChecker;
   }

   public void spellCheck(){
      spellChecker.checkSpelling();
   }
}
```

Following is the configuration file **Beans.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:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

   <!-- Definition for textEditor bean without constructor-arg  -->
   <bean  >
   </bean>

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

</beans>
```

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

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

@Autowired with *required* = *false* option

By default, the @Autowired annotation implies the dependency is required similar to @Required annotation, however, you can turn off the default behavior by using ***required*** = ***false*** option with @Autowired.

The following example will work even if you do not pass any value for age property but still it will demand for name property. You can try this example yourself because this is similar to @Required annotation example except that only **Student.java** file has been changed.

```
package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Autowired;

public class Student {
   private Integer age;
   private String name;

   @Autowired(required=false)
   public void setAge(Integer age) {
      this.age = age;
   }

   public Integer getAge() {
      return age;
   }
```

```
    @Autowired
    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```