SPRING AUTOWIRING 'BYNAME'

This mode specifies autowiring by property name. Spring container looks at the beans on which *auto-wire* attribute is set to *byName* in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file. If matches are found, it will inject those beans otherwise, it will throw exceptions.

For example, if a bean definition is set to autowire *byName* in configuration file, and it contains a *spellChecker* property (that is, it has a *setSpellChecker*. . . method), Spring looks for a bean definition named *spellChecker*, and uses it to set the property. Still you can wire remaining properties using <property> tags. Following example will illustrate the concept.

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;
   private String name;

   public void setSpellChecker( SpellChecker spellChecker ){
      this.spellChecker = spellChecker;
   }
   public SpellChecker getSpellChecker() {
      return spellChecker;
   }

   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      return name;
   }

   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** in normal condition:

```
<?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  >
      <property name="spellChecker" ref="spellChecker" />
      <property name="name" value="Generic Text Editor" />
   </bean>

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

</beans>
```

But if you are going to use autowiring 'byName', then your XML configuration file will become as follows:

```
<?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
      autowire="byName">
      <property name="name" value="Generic Text Editor" />
   </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.
```