SPRING JSR-250 ANNOTATIONS

Spring also JSR-250 based annotations which include @PostConstruct, @PreDestroy and @Resource annotations. Though these annotations are not really required because you already have other alternates but still let me give a brief idea about them.

@PostConstruct and @PreDestroy Annotations:

To define setup and teardown for a bean, we simply declare the <bean> with **init-method** and/or **destroy-method** parameters. The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation. Similarly, destroy-method specifies a method that is called just before a bean is removed from the container.

You can use **@PostConstruct** annotation as an alternate of initialization callback and **@PreDestroy** annotation as an alternate of destruction callback as explained in the below example.

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 *HelloWorld* 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 **HelloWorld.java** file:

```
package com.tutorialspoint;
import javax.annotation.*;

public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public String getMessage(){
      System.out.println("Your Message : " + message);
      return message;
   }
   @PostConstruct
   public void init(){
      System.out.println("Bean is going through init.");
   }
   @PreDestroy
   public void destroy(){
      System.out.println("Bean will destroy now.");
   }
```

```
}
```

Following is the content of the **MainApp.java** file. Here you need to register a shutdown hook **registerShutdownHook** method that is declared on the AbstractApplicationContext class. This will ensures a graceful shutdown and calls the relevant destroy methods.

```
package com.tutorialspoint;

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

public class MainApp {
   public static void main(String[] args) {

      AbstractApplicationContext context =
                           new ClassPathXmlApplicationContext("Beans.xml");

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
      obj.getMessage();
      context.registerShutdownHook();
   }
}
```

Following is the configuration file **Beans.xml** required for init and destroy methods:

```
<?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/>

   <bean

       init-method="init" destroy-method="destroy">
       <property name="message" value="Hello World!"/>
   </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:

```
Bean is going through init.
Your Message : Hello World!
Bean will destroy now.
```

@Resource Annotation:

You can use **@Resource** annotation on fields or setter methods and it works the same as in Java EE 5. The @Resource annotation takes a 'name' attribute which will be interpreted as the bean name to be injected. You can say, it follows **by-name** autowiring semantics as demonstrated in the below example:

```
package com.tutorialspoint;

import javax.annotation.Resource;

public class TextEditor {
   private SpellChecker spellChecker;
```

```
   @Resource(name= "spellChecker")
   public void setSpellChecker( SpellChecker spellChecker ){
      this.spellChecker = spellChecker;
   }
   public SpellChecker getSpellChecker(){
      return spellChecker;
   }
   public void spellCheck(){
      spellChecker.checkSpelling();
   }
}
```

If no 'name' is specified explicitly, the default name is derived from the field name or setter method. In case of a field, it takes the field name; in case of a setter method, it takes the bean property name.