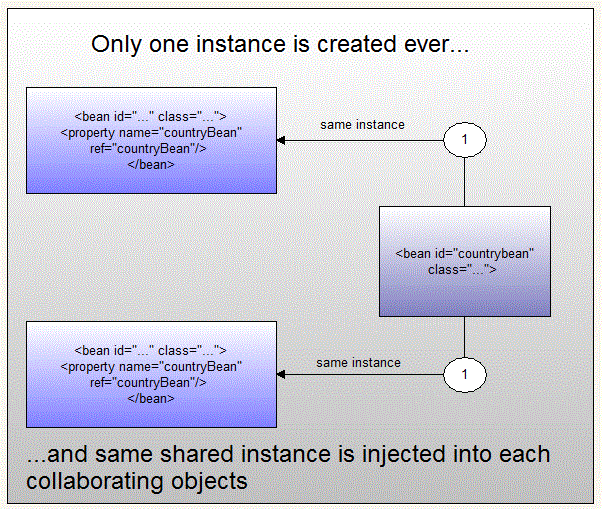
In Spring, bean scope is used to decide which type of bean instance should be return from Spring container back to the caller.  
There are 5 types of bean scopes supported in spring

1. **singleton** – Scopes a single bean definition to a single object instance per Spring IoC container.
2. **prototype** – Return a new bean instance each time when requested
3. **request**– Return a single bean instance per HTTP request.
4. **session**– Return a single bean instance per HTTP session.
5. **globalSession** – Return a single bean instance per global HTTP session.

In many cases,spring’s core scopes i.e.singleton and prototype are used.By default scope of beans is singleton.  
Here we will see singleton and prototype scopes in more details.

**Singleton bean scope**

**[](https://java2blog.com/wp-content/uploads/2012/08/singletonInSpring-1.gif)**

**Example:**

For configuring spring in your eclipse ide please refer  **[hello world example](https://www.java2blog.com/2012/08/spring-hello-world-example-in-eclipse.html" \t "_blank)**

**1.Country.java:**

This is simple pojo class having some attributes so here country has name.

Create Country.java under package **org.arpit.javapostsforlearning**.Copy following content into Country.java.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | package org.arpit.javapostsforlearning;    public class Country {    String countryName;    public String getCountryName() {  return countryName;  }      public void setCountryName(String countryName) {  this.countryName = countryName;  }    } |

**2.ApplicationContext.xml**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <?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:aop="http://www.springframework.org/schema/aop"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">  <bean id="country" class="org.arpit.javapostsforlearning.Country">  </bean>  </beans> |

**3.ScopesInSpringMain.java**

This class contains main function.Create ScopesInSpringMain.java under package **org.arpit.javapostsforlearning**.Copy following content into ScopesInSpringMain.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | package org.arpit.javapostsforlearning;    import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    public class ScopesInSpringMain{    public static void main(String[] args) {    ApplicationContext appContext = new ClassPathXmlApplicationContext("ApplicationContext.xml");  Country countryObj1 = (Country) appContext.getBean("country");  countryObj1.setCountryName("India");  System.out.println("Country Name:"+countryObj1.getCountryName());    //getBean called second time  Country countryObj2 = (Country) appContext.getBean("country");  System.out.println("Country Name:"+countryObj2.getCountryName());  }  } |

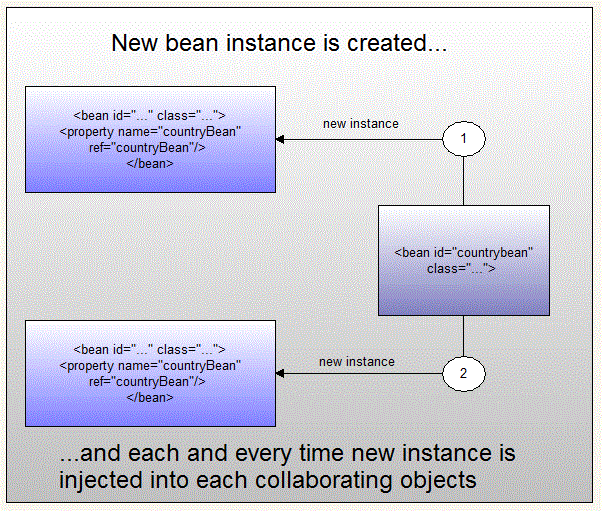
**4.Run it**

When you will run above application,you will get following as output.

|  |  |
| --- | --- |
| 1  2  3  4 | Country Name:India  Country Name:India |

When We firstly called getBean and retrieved country object and set countryName to “india” and when second time we called getBean method it did nothing but returned same object with countryName as “india”.

**Prototype bean scope**

**[](https://java2blog.com/wp-content/uploads/2012/08/PrototypeInSpring-1.gif)**

Now we will make change in above xml configuration file.We will add scope attribute in tag and set it to “prototype” and then run it again

**ApplicationContext.xml:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?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:aop="http://www.springframework.org/schema/aop"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">    <bean id="country" class="org.arpit.javapostsforlearning.Country" scope="prototype">  </bean>  </beans> |

**Run it again:**

When you will run above application,you will get following as output.

|  |  |
| --- | --- |
| 1  2  3  4 | Country Name:India  Country Name:null |

When We firstly called getBean and retrieved country object and set countryName to “india”and when second time we called getBean method it returned new object with countryName as “null”.

**1.Country.java:**

This is simple pojo class having some attributes so here country has name and list of states.

Create Country.java under package **org.arpit.javapostsforlearning**.Copy following content into Country.java.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | package org.arpit.javapostsforlearning;    import java.util.List;    public class Country {    String countryName;  List listOfStates;  public String getCountryName() {  return countryName;  }  public void setCountryName(String countryName) {  this.countryName = countryName;  }  public List getListOfStates() {  return listOfStates;  }  public void setListOfStates(List listOfStates) {  this.listOfStates = listOfStates;  }    public void printListOfStates()  {  System.out.println("Some of states in india are:");  for(String state:listOfStates)  {  System.out.println(state);  }  }  } |

**2.ApplicationContext.xml**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?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:aop="http://www.springframework.org/schema/aop"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">    <bean id="CountryBean" class="org.arpit.javapostsforlearning.Country">    <property name="listOfStates">     <list>      <value>Maharastra</value>      <value>Madhya Pradesh</value>      <value>Rajasthan</value>     </list>      </property>  </bean>    </beans> |

here for initializing collectios(list) i.e. listofStates attribute of country class, we have used list tag.

In tag ,you can have tag or tag for adding values in list.

**3.InitializingCollectionsMain.java**

This class contains main function.Create InitializingCollectionsMain.java under package **org.arpit.javapostsforlearning**.Copy following content into InitializingCollectionsMain.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package org.arpit.javapostsforlearning;    import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    public class InitializingCollectionsMain{        public static void main(String[] args) {          ApplicationContext appContext = new ClassPathXmlApplicationContext("ApplicationContext.xml");          Country countryObj = (Country) appContext.getBean("CountryBean");          countryObj.printListOfStates();      }  } |

You can note here that we have used ClassPathXmlApplicationContext for getting bean here.There are various ways for getting beans.In [**hello world example**](https://www.java2blog.com/2012/08/spring-hello-world-example-in-eclipse.html) we have used XmlBeanFactory for getting beans.

**4.Run it**

When you will run above application,you will get following as output.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Some of states in india are:  Maharastra  Madhya Pradesh  Rajasthan |

## Autowiring modes:

There are following autowiring modes which can be used to instruct Spring container to use autowiring for dependency injection.

### no:

Default, no auto wiring, set it manually via “ref” attribute as we have done in dependency injection via settor method post.

### byName:

Autowiring by property name. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byName* in the XML configuration file and it tries to match it with name of bean in xml configuration file.

### byType:

Autowiring by property datatype. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byType* in the XML configuration file. It then tries to match and wire a property if its **type** matches with exactly one of the beans name in configuration file. If more than one such beans exists, a fatal exception is thrown.

### contructor:

byType mode in constructor argument.

#### autodetect:

Spring first tries to wire using autowire by *constructor*, if it does not work, Spring tries to autowire by *byType*.

### Example:

I am taking example of autowire “byName” here.It will be almost same as [**Dependency injection via setter method**](https://www.java2blog.com/2012/08/dependency-injection-via-setter-method.html) with some changes in XML configuration file.

#### 1.Country.java:

This is simple pojo class having some attributes so here country has name and object of Capital class.

Create Country.java under package **org.arpit.javapostsforlearning**.Copy following content into Country.java.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | package org.arpit.javapostsforlearning;    public class Country {        String countryName;      Capital capital;      public String getCountryName() {          return countryName;      }      public void setCountryName(String countryName) {          this.countryName = countryName;      }      public Capital getCapital() {          return capital;      }      public void setCapital(Capital capital) {          this.capital = capital;      }  } |

#### 2.Capital.java

This is also simple pojo class having one attribute called “capitalName”.

Create Capital.java under package **org.arpit.javapostsforlearning**.java.Above Country class contains object of this class.Copy following content into Capital.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package org.arpit.javapostsforlearning;    public class Capital {        String capitalName;        public String getCapitalName() {          return capitalName;      }        public void setCapitalName(String capitalName) {          this.capitalName = capitalName;      }  } |

#### 3.BeanAutowirngByNameMain.java

This class contains main function.Create BeanAutowiringByNameMain.java under package **org.arpit.javapostsforlearning**.Copy following content into BeanAutowiringByNameMain.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package org.arpit.javapostsforlearning;    import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    public class BeanAutowiringByNameMain{        public static void main(String[] args) {          ApplicationContext appContext = new ClassPathXmlApplicationContext("ApplicationContext.xml");            Country countryObj = (Country) appContext.getBean("country");          String countryName=countryObj.getCountryName();          String capitalName=countryObj.getCapital().getCapitalName();          System.out.println(capitalName+" is capital of "+countryName);        }  } |

You can note here that we have used ClassPathXmlApplicationContext for getting bean here.There are various ways for getting beans.In [**hello world example**](https://www.java2blog.com/2012/08/spring-hello-world-example-in-eclipse.html) we have used XmlBeanFactory for getting beans.

#### 4.ApplicationContext.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?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:aop="http://www.springframework.org/schema/aop"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">    <bean id="country" class="org.arpit.javapostsforlearning.Country" autowire="byName">  <property name="countryName" value="India"/>  </bean>  <bean id="capital" class="org.arpit.javapostsforlearning.Capital">  <property name="capitalName" value="Delhi" />  </bean>  </beans> |

Here in ,we have used autowire attribute and set it to “byName”.So now spring container will match “capital” reference in Country class with id or name of other beans in XML configuration file. So here you can see we have bean with id as “capital”

#### 5.Run it

When you will run above application,you will get following as output.

|  |  |
| --- | --- |
| 1  2  3 | Delhi is capital of India |

## Limitations of Autowiring :

* **OverRiding possibilities:**You can still define dependencies using or tag which will always override autowiring.
* **Primitive data type**:you have to define primitive data type like String or Interger using or tag.You can not autowire them.
* **Confusing Nature**:If you have lot of dependency in program,then its very hard to find out using autowire attribute of bean.
* In spring,inheritance is supported for reusing already written bean,so that beans can share common attributes and methods among them.
* child bean will have all attributes and methods of parent bean,also child bean can override parent bean’s attributes or methods.
* For configuring spring in your eclipse ide please refer  [**hello world example**](https://www.java2blog.com/2012/08/spring-hello-world-example-in-eclipse.html)
* **Inheritance in Spring example**
* **1.Person.java**
* This is simple pojo class having some attributes so here preson has name .
* Create Person.java under package **org.arpit.javapostsforlearning**.Copy following content into Person.java.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | package org.arpit.javapostsforlearning;    public class Person {    String name;    public String getName() {  return name;  }    public void setName(String name) {  this.name = name;  }    } |

* **2.Employee.java**
* This is also simple pojo class having one attribute called “employeeNumber”.
* Create Employee.java under package **org.arpit.javapostsforlearning**.java..Copy following content into Employee.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | package org.arpit.javapostsforlearning;    public class Employee extends Person{    int employeeNumber;    public int getEmployeeNumber() {  return employeeNumber;  }    public void setEmployeeNumber(int employeeNumber) {  this.employeeNumber = employeeNumber;  }    } |

* **3.ApplicationContext.xml**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?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:aop="http://www.springframework.org/schema/aop"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">    <bean id="personBean" class="org.arpit.javapostsforlearning.Person">  <property name="name" value="Arpit"/>  </bean>  <bean id="employeeBean" class="org.arpit.javapostsforlearning.Employee" parent="personBean">  <property name="employeeNumber" value="178230" />  </bean>  </beans> |

* Here We have declared two beans with corresponding ids.  
  1.Class Person with id as “PersonBean”  
  2.Class Employee with id as “EmployeeBean”  
  We have used  **parent** property of to show that Person is parent class of Employee.

|  |  |
| --- | --- |
| 1  2  3 | <bean id="employeeBean" class="org.arpit.javapostsforlearning.Employee" parent="id or name of parent bean"> |

* **4.InheritanceInSpringMain.java**
* This class contains main function.Create InheritanceInSpring.java under package **org.arpit.javapostsforlearning**.Copy following content into InheritanceInSpring.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package org.arpit.javapostsforlearning;    import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    public class InheritenceInSpringMain {    public static void main(String[] args) {  ApplicationContext appContext = new ClassPathXmlApplicationContext("ApplicationContext.xml");  Employee emp=(Employee) appContext.getBean("employeeBean");  System.out.println("Employee name:"+emp.getName());  System.out.println("Employee number:"+emp.getEmployeeNumber());  }  } |

* You can note here that we have used ClassPathXmlApplicationContext for getting bean here.There are various ways for getting beans.In [**hello world example**](https://www.java2blog.com/2012/08/spring-hello-world-example-in-eclipse.html) we have used XmlBeanFactory for getting beans.
* **5.Run it**
* When you will run above application,you will get following as output.

|  |  |
| --- | --- |
| 1  2  3  4 | Employee name:Arpit  Employee number:178230 |

## ApplicationContext:

ApplicationContext is an central interface for providing configuration information to an application.

An ApplicationContext provides the following functionalities:

* Bean factory methods, inherited from ListableBeanFactory. This avoids the need for applications to use singletons.
* The ability to resolve messages, supporting internationalization. Inherited from the MessageSource interface.
* The ability to load file resources in a generic fashion. Inherited from the ResourceLoader interface.
* The ability to publish events. Implementations must provide a means of registering event listeners.
* Inheritance from a parent context. Definitions in a descendant context will always take priority. This means, for example, that a single parent context can be used by an entire web application, while each servlet has its own child context that is independent of that of any other servlet.

### ApplicationContext vs BeanFactory:

We can get our bean from both by applying getBean method.BeanFactory is subset of ApplicationContext and provides less functionalities.So if we want to use full functionalities then we go for ApplicationContext.

## Getting ApplicationContext in Bean Class:

To get access to ApplicationContext we should implement ApplicationContextAware interface in the respective java bean.

### ApplicationContextAware  Interface:

 It has a method,

|  |  |
| --- | --- |
| 1  2  3  4  5 | public void setApplicationContext(ApplicationContext applicationContext)                throws BeansException |

The ApplicationContext implementation which we are using in our application will invoke this method and pass the concrete object for AppplicationContext. Using this we can get access to all the configuration information.

Implementing ApplicationContextAware interface makes sense for example when an object requires access to a set of collaborating beans. Note that configuration via bean references is preferable to implementing this interface just for bean lookup purposes.

This interface can also be implemented if an object needs access to file resources, i.e. wants to call getResource, wants to publish an application event, or requires access to the MessageSource. However, it is preferable to implement the more specific ResourceLoaderAware , ApplicationEventPublisherAware or MessageSourceAware interface in such a specific scenario.

#### Spring Application context example:

For configuring spring in your eclipse ide please refer  **[hello world example](https://www.java2blog.com/2012/08/spring-hello-world-example-in-eclipse.html" \t "_blank)**

## 1.Country.java:

This is simple pojo class having some attributes so here country has name and object of Capital class.

Create Country.java under package **org.arpit.javapostsforlearning**.Copy following content into Country.java.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | package org.arpit.javapostsforlearning;    import org.springframework.beans.BeansException;  import org.springframework.context.ApplicationContext;  import org.springframework.context.ApplicationContextAware;    public class Country implements ApplicationContextAware{    String countryName ;  ApplicationContext applicationContext;  Capital capital;  public String getCountryName() {  return countryName;  }      public void setCountryName(String countryName) {  this.countryName = countryName;  }      public void setApplicationContext(ApplicationContext applicationContext)  throws BeansException {  this.applicationContext=applicationContext;  }    public String getCapitalName(String capitalBeanName)  {  capital=(Capital) applicationContext.getBean(capitalBeanName);  String capitalName=capital.getCapitalName();  return capitalName;  }  } |

## 2.Capital.java

This is also simple pojo class having one attribute called “capitalName”.

Create Capital.java under package **org.arpit.javapostsforlearning**.java.Above Country class contains object of this class.Copy following content into Capital.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package org.arpit.javapostsforlearning;    public class Capital {    String capitalName;    public String getCapitalName() {  return capitalName;  }    public void setCapitalName(String capitalName) {  this.capitalName = capitalName;  }  } |

## 4.ApplicationContext.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.springframework.org/schema/aop"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">    <bean id="country" class="org.arpit.javapostsforlearning.Country" >  <property name="countryName" value="India"/>  </bean>    <bean id="capital" class="org.arpit.javapostsforlearning.Capital" >  <property name="capitalName" value="Delhi"/>  </bean>  </beans> |

As you can see, there is no connection between above two beans in this XML file.We are getting object of country class with help of getBean method and then passing id of  capital class to getCapitalName method of country class.In getCapital method,we have ApplicationContext object which is initialized by setApplicationContext method by spring container ,so with help of ApplicationContext object,we are calling getBean method for initializing capital object and getting capitalName from that object.

## 4.ApplicationContextMain.java

This class contains main function.Create ApplicationContextMain.java under package **org.arpit.javapostsforlearning**.Copy following content into ApplicationContextMain.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package org.arpit.javapostsforlearning;    import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    public class ApplicationContextMain{    public static void main(String[] args) {    ApplicationContext appContext = new ClassPathXmlApplicationContext("ApplicationContext.xml");  Country countryObj = (Country) appContext.getBean("country");  System.out.println("Capital Name:"+countryObj.getCapitalName("capital"));  }  } |

You can note here that we have used ClassPathXmlApplicationContext for getting bean here.There are various ways for getting beans.In [**hello world example**](https://www.java2blog.com/2012/08/spring-hello-world-example-in-eclipse.html) we have used XmlBeanFactory for getting beans.

## 5.Run it

When you will run above application,you will get following as output.

|  |  |
| --- | --- |
| 1  2  3 | India's Capital Name:Delhi |

There are two ways via which you can inject dependency in spring

1. By configuring XML.
2. By using annotation.

In all our previous posts,we have injected dependency by configuring XML file but instead of doing this,we can move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

You might think what if you have done both i.e.used annotations and XML both.In that case,XML configuration will override annotations because XML configuration will be injected after annotations.

Now annotations based configuration is turned off by default so you have to turn it on by entering into spring XML file.

### ApplicationContext.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <?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/>  <!-- beans declaration goes here -->  </beans> |

Now you are ready to use annotations in your code.Let us discuss few important annotations in spring

#### [@Required:](https://www.java2blog.com/2012/09/required-annotation-in-spring.html)

 The @Required annotation applies to bean property setter methods.

#### [@Autowired:](https://www.java2blog.com/2012/09/autowired-annotation-in-spring.html)

The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties.

#### [@Qualifier:](https://www.java2blog.com/2012/09/qualifier-annotation-in-spring.html)

The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired.

#### [JSR 250 Annotations:](https://www.java2blog.com/2012/09/spring-jsr-250-annotations.html)

Spring supports JSR-250 based annotations which include @Resource, @PostConstruct and @PreDestroy annotations.