BY NAGOOR BABU

Beans In Spring Framework

Bean Definition: **Bean is a Software Reusable Component, it is a normal java class contains properties and the corresponding setXXX(-) and getXXX() methods and which are created and managed by IOC Container in Spring Framework.**

Rules and Regulations to write Bean classes:

a) Bean classes must be POJO classes, they must not extend or implement any predefined Library except java.io.Serializable marker interface.

b) Bean must be declared as "public" , "Non-abstract" and "non-final".
----> The main intention of declaring bean class as "public" is to make available bean class scope to IOC Container inorder to create objects.
----> The main intention to declare bean class as "Non-abstract" is to allow to create object .
----> The main intention to declare bean classes as "Non-final" is to extend one bean class to another bean class inorder to improve reusability.

c) In Bean classes, we have to declare all properties as "private" and all behaviours as "public", it will improve "Encapsulation".
d) If we want to provide any constructor in bean class then provide a constructor , it must be 0-arg constructor and "public" constructor, because, IOC Container will search and execute public and 0-arg constructor while instantiating bean.

If we want to use Beans in Spring applications then we must configure that bean classes in spring beans configuration file , because, IOCContainer will recognize and create Bean objects by getting bean class details from beans configuration file only.

CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

BY NAGOOR BABU

There are three ways to provide beans configurations in spring applications.

1. XML Configuration
2. Java Based Configuration
3.Annotations Configuration

1. XML Configuration

To provide beans configurations in beans configuration file we have to use the following xml tags.

```
<beans>

 <bean id="--" name="--" class="--" scope="--">
   <property name="--" value="--"/>

 </bean>

</beans>
```

Where <beans> tag is root tag in beans configuration file.
Where <bean> tag is able to provide configuration details of a particular bean class.

Where "class" attribute in <bean> is able to provide fully qualified name of the bean class.

Where <property> attribute is able to represent a particular property[variable] in bean class and it will set the specified value to the respective bean property by executing setXXX(-) method.

BY NAGOOR BABU

Q)What is the difference between "id" attribute and "name" attribute in <bean> tag?

Ans: 'id' attribute is able to take exactly one identity to the bean object, it will not allow more than one identity.
'name' attribute in <bean> tag is able to allow more than one identity name to the bean object, where in multiple values only first value is treated as th actual bean identity and the remaining names are alias names for the bean object. In this context, while providing alias names to the bean object we have to use either ',' or ';' or [space] as delimiter[seperator].

EX1:

```
<beans>
 <bean id="bean1" class="com.durgasoft.beans.MyBean"/>
</beans>
MyBean mb=(MyBean)ctx.getBean("bean1");
```

Status: Valid.

EX2:

```
<beans>
 <bean id="bean1 bean2 bean3" class="com.durgasoft.beans.MyBean"/>
</beans>
MyBean mb=(MyBean)ctx.getBean("bean1");
```

Status: org.springframework.beans.factory.NoSuchBeanDefinitionException: No bean named 'bean1' is defined
Note: SImilarily the following cases are also be invalid.

```
<bean id="bean1,bean2,bean3" class="MyBean"/>->INvalid
<bean id="bean1;bean2;bean3" class="MyBean"/>->Invalid
<bean id="bean1bean2bean3" class="MyBean"/>-> Valid
```

EX3:

```
<beans>
 <bean name="bean1" class="MyBean"/>
```


```
</beans>
MyBean mb=(MyBean)ctx.getBean("bean1");
Status: Valid
```

EX4:

```
<bean name="bean1 bean2 bean3" class="MyBean"/>
MyBean mb1=ctx.getBean("bean1");
MyBean mb2=ctx.getBean("bean2");
MyBean mb3=ctx.getBean("bean3");
Status: Valid
Similarily the following cases are also be valid.
<bean name="bean1,bean2,bean3" class="MyBean"/>
<bean name="bean1;bean2;bean3" class="MyBean"/>
```

Note: It is possible to use both 'id' attribute and 'name' attribute in single <bean> tag.

EX5:

```
<bean id="bean1" name="bean2" class="MyBean"/>
MyBean mb1=ctx.getBean("bean1");
MyBean mb2=ctx.getBean("bean2");
Status: Valid.
```

Note: It is possible to provide bean alias names explicitly from out side of the bean definition in configuration file by using <alias> tag.

```
<alias name="--" alias="--"/>
```

Where "name" attribuite will take bean logical name which we specified with "id" attribute in beans configuration file.

Where "alias" attribute will take alias name.

EX6:

```
<beans>
 <bean name="bean1" class="MyBean"/>
 <alias name="bean1" alias="bean2"/>
 <alias name="bean2" alias="bean3"/>
 </bean>
</beans>
```

MyBean mb1=ctx.getBean(bean1); --> Valid
MyBean mb2=ctx.getBean(bean2); --> Valid
MyBean mb3=ctx.getBean(bean3); --> Valid

Bean Scopes:

In J2SE applications, we are able to define scopes to the data by using the access modifiers like public, protected, <default> and private.
Similarily, in Spring framework to define scopes to the beans spring framework has provided the following scopes.

1.singleton Scope[Default Scope]
2.prototype Scope
3.request Scope
4.session Scope
5.globalSession Scope
6.application Scope
7.webSocket scope

1. singleton Scope:

It is default scope in Spring applications.

If we use this scope to the bean then IOCContainer will create Single Bean object for single bean definition in Spring config file.

This approach will return the same bean object for every time requesting bean object. When we request bean object first time then IOCContainer will create bean object really and it will be stored in Cache memory, then , every time accessing bean object , IOCContainer will return the same bean object reference value with out creating new Bean objects.

EX:

```
beans.xml
<beans>
 <bean id="bean1" class="com.durgasoft.MyBean" scope="singleton"/>
 <bean id="bean2" class="com.durgasoft.MyBean" scope="singleton"/>
</beans>

System.out.println(ctx.getBean("bean1"));//MyBean@a111
System.out.println(ctx.getBean("bean1"));//MyBean@a111
System.out.println(ctx.getBean("bean2"));//MyBean@a222
System.out.println(ctx.getBean("bean2"));//MyBean@a222
```

2. prototype Scope:

It is not default Scope in Spring framework.
In Spring applications, if we provide "prototype" scope in bean configuration file then IOCContainer will ceate a new Bean object at each and every time of calling getBean(--) method.

EX:

```
beans.xml
---------
<beans>
 <bean id="bean1" class="com.durgasoft.MyBean" scope="prototype"/>
 <bean id="bean2" class="com.durgasoft.MyBean" scope="prototype"/>
</beans>
```


```
System.out.println(ctx.getBean("bean1"));//MyBean@a111
System.out.println(ctx.getBean("bean1"));//MyBean@a222
System.out.println(ctx.getBean("bean2"));//MyBean@a333
System.out.println(ctx.getBean("bean2"));//MyBean@a444
```

3. **requestScope:**

This scope is not usefull in Standalone Applications[Spring Core MOdule], it will be used in Web applications which are prepared on the basis of Spring Web module.
RequestScope is able to create a seperate bean object for each and every request object.

4. **sessionScope:**

This Scope will be used web applications which are prepared on the basis of Spring web module and it is not applicable in Standalone Applications.
sessionScope allows to create a seperate bean object for each and every Session object in web applications.

5. **globalSession Scope:**

This scope is not usefull in standard applications, it is usefull in portlet applications which are prepared on the basis of SPring web module.
globalSession scope allows to create a seperate bean object for each and every portlet Session.

6. **application Scope:**
7.

This scope is not usefull in standalone Applications, it is usefull in web applications prepared on the basis of Spring web momdule.
ApplicationScope allows to create a seperate bean object for each and every ServletContext object.

8. webSocketScope:

This scope is usefull in web applications which are prepared on the basis of spring web module.
websocket scope allows to create a seperate bean object for single websocket lifecycle.

If we use the scopes like request, session, globalSession, appplication, webSocket,... in standalone applications which are prepared on the basis of spring core module then Container will rise an exception like "java.lang.IllegalStateException".

Note: Spring Framework has provided environment to customize the existed scopes , but, it is not suggestible. Spring framework has provided environment to create new scopes in spring applications.

To define and use custom scopes in Spring Framework we have to use the following steps.

1.Create User Defined Scope class.
2.Register User defined Scope in Spring beans configuration file.
3.Use User defined Scope to the Beans in beans configuration file.

1. Create User defined Scope class:

a) Declare an user defined class.
b) Implement org.springframework.beans.factory.config.Scope interface to User defined class.
c) Implement the following Scope interface methods in User defined class.
1) get(--): It able to generate a bean object from Scope.
public Object get(String name, ObjectFactory factory)
2) remove(): It able to remove bean object from Scope.
public Object remove(String name)
3) getConversationalId(): It able to provide an id value of the scope if any.
EX: IN Session scope, generating sessionId.
public String getConversationId()

4) registerDestructionCallback(): It will be executed when bean object is destroyed in Scope

```
public void registerDestructionCallback(String name, Runnable r)
```

5) resolveContextualObject(): It will resolve the situation where Multiple Context objects associated with the keys.

```
public Object resolveContextualObject(String name)
```

Note: From the above methods, get() and remove() methods are mandatory to implement and all the remaining methods are optional.

2. Register User defined Scope in Spring beans configuration File:

a) Configure org.springframework.beans.factory.config.CustomScopeConfigurer class as a bean.
b) Declare "scopes" as property in CustomScopeConfigurer
c) Declare "map" under scopes property.
d) Declare "entry" under the "map".
e) Declare "key" in "entry" with User defined scope name and provide value as USer defined SCope object.

3. Apply User defined Scope to bean definitions:

Use "scope" attribute in <bean> tag to apply user defined scope.

Note: In the following example, we have defined threadScope as user defined scope, that is, it able to create a seperate bean object for each and every thread.

Example:

CustomThreadLocal.java

```
package com.durgasoft.scopes;
import java.util.HashMap;

public class CustomThreadLocal extends ThreadLocal<Object> {
        @Override
```


```
        protected Object initialValue() {
                return new HashMap<String, Object>();
        }
}
```

ThreadScope.java

```
package com.durgasoft.scopes;

import java.util.Map;

import org.springframework.beans.factory.ObjectFactory;
import org.springframework.beans.factory.config.Scope;

public class ThreadScope implements Scope {

        Map<String, Object> scope = null;
        CustomThreadLocal threadLocal = new CustomThreadLocal();
        @Override
        public Object get(String name, ObjectFactory objectFactory) {

                scope = (Map<String, Object>)threadLocal.get();
                Object obj = scope.get(name);
                if(obj == null) {
                        obj = objectFactory.getObject();
                        scope.put(name, obj);
                }
                return obj;
        }

        @Override
        public String getConversationId() {
                // TODO Auto-generated method stub
                return null;
```


```
	}

	@Override
	public void registerDestructionCallback(String arg0, Runnable arg1) {
		// TODO Auto-generated method stub

	}

	@Override
	public Object remove(String name) {
		Object obj = scope.remove(name);
		return obj;
	}

	@Override
	public Object resolveContextualObject(String arg0) {
		// TODO Auto-generated method stub
		return null;
	}

}
```

HelloBean.java

```
package com.durgasoft.beans;

public class HelloBean {
	public HelloBean() {
		System.out.println("HelloBean  Object is created");
	}
	public String sayHello() {
		return "Hello User from "+Thread.currentThread().getName()+" Scope";
	}
}
```

Test.java

```
package com.durgasoft.test;

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

import com.durgasoft.beans.HelloBean;
import com.durgasoft.scopes.ThreadScope;

public class Test {

	public static void main(String[] args) {
		ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
		HelloBean bean1 = (HelloBean)context.getBean("helloBean");
		HelloBean bean2 = (HelloBean)context.getBean("helloBean");
		System.out.println(bean1);
		System.out.println(bean2);
		System.out.println(bean1 == bean2);
		System.out.println(bean1.sayHello());
		System.out.println(bean2.sayHello());

		ThreadScope threadScope = (ThreadScope)context.getBean("threadScope");
		HelloBean bean3 = (HelloBean)threadScope.remove("helloBean");
		System.out.println(bean3);

		HelloBean bean4 = (HelloBean)context.getBean("helloBean");
		HelloBean bean5 = (HelloBean)context.getBean("helloBean");
		System.out.println(bean4);
		System.out.println(bean5);
		System.out.println(bean4 == bean5);
		System.out.println(bean4.sayHello());
```

```
                System.out.println(bean5.sayHello());

        }
}
```

applicationContext.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"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd">
        <bean id="helloBean"  class="com.durgasoft.beans.HelloBean" scope="thread"/>
        <bean id="threadScope" class="com.durgasoft.scopes.ThreadScope"/>
        <bean id="scopeConfigurer"
class="org.springframework.beans.factory.config.CustomScopeConfigurer">
                <property name="scopes">
                        <map>
                                <entry key="thread" value-ref="threadScope"/>
                        </map>
                </property>
        </bean>

</beans>
```

2. Java Based Configuration

In Spring, upto Spring2.4 version Spring beans configuration file is mandatory to configure bean classes and their metadata, but, Right from Spring3.x version Spring beans configuration file is optional, because, SPring3.x version has provided Java Based Configuration as replacement for XML documents.

BY NAGOOR BABU

If we want to use Java Based Configuration as an alternative to Spring beans configuration file in Spring applications then we have to use the following steps.

1. Create Bean classes as per the requirement.
2. Create Beans configuration class with the following annotations.

org.springframework.context.annotation.@Configuration
--> It able to represent a class as configuration class.
org.springframework.context.annotation.@Bean
--> It will be used at method to represent the return object is bean object.

3. In Test class, Create ApplicationContext object with the org.springframework.context.annotation.Annot ationConfigpplicationContext implementation class.

```
ApplicationContext context=new AnnotationConfigApplicationContext(BeanConfig.class);
```

4. Get Bean object from ApplicationContext by using the following method.

```
public Object getaBean(Class c)
EX: Bean b=context.getBean(Bean.class);
```

5. Access business methods from Bean.

Example:

HelloBean.java

```
package com.durgasoft.beans;

public class HelloBean {
        static {
                System.out.println("Bean Loading.....");
        }
```


CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,


```
        public HelloBean() {
                System.out.println("Bean Created....");
        }
        public String sayHello() {
                return "Hello User";
        }
}
```

HelloBeanConfig.java

```
package com.durgasoft.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.durgasoft.beans.HelloBean;

@Configuration
public class HelloBeanConfig {
        @Bean
        public HelloBean helloBean() {
                return new HelloBean();
        }
}
```

Test.java

```
package com.durgasoft.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.durgasoft.beans.HelloBean;
import com.durgasoft.config.HelloBeanConfig;
```

BY NAGOOR BABU

```
public class Test {

        public static void main(String[] args) throws Exception{

                ApplicationContext context = new
AnnotationConfigApplicationContext(HelloBeanConfig.class);
                HelloBean bean = (HelloBean)context.getBean("helloBean");
                System.out.println(bean.sayHello());
        }

}
```

Bean Lifecycle:

In spring framework applications, when IOC Container recognizes all the beans definitions in beans configuration file then IOC Container will execute that bean by using the following lifecycle actions.

1. **Bean Class Loading**
2. **Bean Instantiation**
3. **Bean Initialization**
4. **Bean Destruction**

1. Bean Class Loading:

When IOC Container recognized fully qualified names of the bean classes in beans configuration file then IOC Container will load the specified bean class byte code to the memory. To load bean class bytecode to the memory, IOC Container will use the following method.

```
public static Class forName(String class_Name)throws ClassNotFoundException
EX: Class c=Class.forName("com.durgasoft.beans.Welcom
eBean");
```

2. Bean Instantiation

In Spring applications, after loading bean class bytecode to the memory, IOC Container will create object for the bean class.

IN Spring applications we are able to use the following three approaches to create Bean objects.

1.By using Constructor directly.
2.By Using Static Factory Method
3.By Using Instance Factory Method

1. Bean Instantiation through Constructors:

If we want to create bean objects by using constructors then we must provide 0-arg costructor in bean class irrespective of bean class constructor scopes.

Note: In Bean class , if we provide parameterized constructor then IOC Container will rise an exception.

EX:

HelloBean.java

```
pbulic class HelloBean{
public HelloBean(){
System.out.println("Bean Instantiation");
}
public String sayHello(){
System.out.println("Hello User!";
}
}
```

applicationContext.xml

```
<beans>
 <bean id="hello" class="HelloBean"/>
</beans>
```

Test.java

```
class Test{
public static void main(String[] args){
ApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
HelloBean bean=(HelloBean)context.getBean("hello");
System.out.println(bean.sayaHello());
}
}
```

1. Bean Instantiation through Static Factory Method:

In this approach, first we have to define static factory method in Bean class and we have to configure that static factory method in bean defination in beans configuration file.

EX:

HelloBean.java

```
public class HelloBean{
public static HelloBean getInstance(){
 System.out.println("Static Factory Method");
 return new HelloBean();
}
public String sayHello(){
System.out.println("Hello User!";
}
}
```

applicationContext.xml

```
<beans>
 <bean id="hello" class="HelloBean"
   factory-method="getInstance"/>
</beans>
```

Test.java

```
class Test{
public static void main(String[] args){
ApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
HelloBean bean=(HelloBean)context.getBean("hello");
System.out.println(bean.sayaHello());
}
}
```

2. Bean Instantiation through INstance Factory Method:

In this approach, we have to define a seperate factory class with instance factory method and we have to configure factory class as bean in beans configuration file then we have to configure factory class and factory method in the original bean class definition by using "factory-bean" and
"factory-method" attributes.

EX:

HelloBeanFactory.java

```
public class HelloBeanFactory{
public HelloBean getBeanInstance(){
SYstem.out.println("Instance Factory Method");
```


```
return new HelloBean();
}
```

HelloBean.java

```
public class HelloBean{
public String sayHello(){
System.out.println("Hello User!";
}
}
```

applicationContext.xml

```
<beans>
 <bean id="hello" class="HelloBean"
   factory-method="getBeanInstance"
   factory-bean="factory" />
 <bean id="factory" class="HelloBeanFactory"/>
</beans>
```

Test.java

```
class Test{
public static void main(String[] args){
ApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
HelloBean bean=(HelloBean)context.getBean("hello");
System.out.println(bean.sayaHello());
}
}
```

BY NAGOOR BABU

Bean Initialization and Bean Destruction:

As part of Beans lifecycle, IOC Container has to perform Beans initialization after the Bean Instantiation and IOC Container has to perform Bean destruction after executing the business logic or at the time of shutdown the IOC Container.

There are three ways to perform Beans initialization and destruction in Spring Framework.

1. By using Custom initialization and destruction methods.
2. By using InitializingBean and DesposableBean callback interfaces.
3. By using @PostConstruct and @Predestroy annotations

1. By using Custom initialization and destruction methods:

In this approach, we have to define user defined initialization and destruction methods with any name and we have to configure that user defined methods in beans definitions in beans configuration file by using "init-method" and "destroy-method" attributes in <bean> tag.

EX:

WelcomeBean.java

```
package com.durgasoft.beans;

public class WelcomeBean {
public void init(){
System.out.println("User defined init() method");
}
public String sayWelcome(){
return "Welcome To Durga Software Solutions";
}
```


CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,


```
public void destroy(){
System.out.println("User defined destroy() Method");
}
}
```

applicationContext.xml

```
<beans>
 <bean id="wel"          class="com.durgasoft.beans.WelcomeBean"
    init-method="init" destroy-method="destroy" />
</beans>
```

Test.java

```
package com.durgasoft.test;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.durgasoft.beans.WelcomeBean;
public class Test {
public static void main(String[] args)throws Exception {
AbstractApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
WelcomeBean bean=(WelcomeBean)context.getBean("wel");
System.out.println(bean.sayWelcome());
context.registerShutdownHook();
}
}
```

2. By using InitializingBean and DesposableBean callback interfaces:

IN Spring framework, InitializingBean is a callback interface , it provides the following method to execute while performing bean initialization by IOC Container.

public void afterPropertiesSet()throws Exception

Note: This method will be executed by the container after executing all the setXXX() methods of bean class by ApplicationContext.

IN Spring framework, DisposableBean is a callback interface, it provides the following method to execute while performing Bean Destruction by IOC Container.

public void destroy()

EX:

WelcomeBean.java

```
package com.durgasoft.beans;
import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;

public class WelcomeBean implements InitializingBean,DisposableBean{
private String message;
public String getMessage() {
return message;
}
public void setMessage(String message) {
this.message = message;
System.out.println("setMessage()");
}
public String sayWelcome(){
return message;
}
@Override
public void afterPropertiesSet() throws Exception {
message="Welcome To Durga Software Solutions";
System.out.println("afterPropertiesSet()");
}
```


```
@Override
public void destroy() throws Exception {
System.out.println("destroy()");
}
}
```

applicationContext.xml

```
<beans>
  <bean id="wel" class="com.durgasoft.beans.WelcomeBean">
        <property name="message" value="Welcome To Durgasoft"/>
  </bean>
</beans>
```

Test.java

```
package com.durgasoft.test;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.durgasoft.beans.WelcomeBean;
public class Test {
public static void main(String[] args)throws Exception {
AbstractApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
WelcomeBean bean=(WelcomeBean)context.getBean("wel");
System.out.println(bean.sayWelcome());
context.registerShutdownHook();
}
}
```

3. By using @PostConstruct and @Predestroy annotations

@PostConstruct annotation will make a method to execute by the IOC Container while performing Beans iniitalization.

BY NAGOOR BABU

@Predestroy annotation will make a method to execute by the IOC Container while performing Bean Destruction.

EX:

Welcome.java

```
package com.durgasoft.beans;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

public class WelcomeBean {
	private String message;

	public String getMessage() {
		return message;
	}
	public void setMessage(String message) {
		this.message = message;
		System.out.println("setMessage()");
	}
	public String sayWelcome(){
		return message;
	}
	@PostConstruct
	public void init(){
		System.out.println("postConstruct method");
	}
	@PreDestroy
	public void destroy(){
		System.out.println("preDestroy method");
	}
```

```
}
```

applicationContext.xml

```
<beans >
        <context:annotation-config/>
        <bean id="wel" class="com.durgasoft.beans.WelcomeBean">
                <property name="message" value="Welcome To Durgasoft"/>
        </bean>
</beans>
```

Note: We need to provide "context" name space along with beans name space in spring beans configuration file.

Test.java

```
public class Test {
public static void main(String[] args)throws Exception {
AbstractApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
WelcomeBean bean=(WelcomeBean)context.getBean("wel");
System.out.println(bean.sayWelcome());
context.registerShutdownHook();
}
}
```

Note: IN general, in spring framework applications, we are able to use either of the above three approaches to perform beans initialization and destruction, we are not using all three approaches at a time. If we use all the above three approaches in single bean theni IOC Container will execute the above three approaches provided intialization methods and destruction methods in the following order.

BY NAGOOR BABU

Initialiazation order:

a) an initialization method marked with @Postconstruct annotation.
b) afterPropertiesSet() method provided by InnitializingBean callback interface.
c) an initialization method configured with "init-method" in <bean> tag in beans configuration file

Destruction Order:

a) A Destruction method marked with @Predestroy annotation
b) destroy() method provided by DisposableBean callback interface.
c) A destruction method configured with "destroy-method" annotation in <bean> tag in beans configuration file.

Note: If we have same initialization method and destruction method in more than one bean class in spring application then we are able to provide common init method and destroy method configuration by using
"default-init-method" attribute and "default-destroy-method" attribute in <beans> tag[not in <bean> tag] with out providing "init-method" and "destroy-method" attributes in <bean> tag at each and every bean configuration.

EX:

Welcome.java

```
public class Welcome {
public void init(){
System.out.println("init()-Welcome");
}
public void destroy(){
System.out.println("destroy()-Welcome");
}
}
```


CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

Hello.java

```
public class Hello{
public void init(){
System.out.println("init()-Hello");
}
public void destroy(){
System.out.println("destroy()-Hello");
}
}
```

spring_beans_config.xml

```
<beans default-init-method="init" default-destroy-method="destroy">
 <bean id="welcome" class="com.durgasoft.beans.Welcome"/>
 <bean id="hello" class="com.durgasoft.beans.Hello"/>
</beans>
```

Test.java

```
public class Test{
public static void main(String[] args){
AbstractApplicationContext context=new ClassPathXmlApplicationContext("
/com/durgasoft/cfgs/spring_beans_config.xml");
Welcome bean1=(Welcome)context.getBean("welcome");
Hello bean2=(Hello)context.getBean("hello");
context.registerShutdownHook();
}
}
```

4. Beans Inheritance:

In general, in Spring framework, we are able to provide more and more no of configuration details in beans definitions under beans configuration file like properties

configurations, dependency injection related configurations, initialization destruction methods configurations,.....

In the above context, we are able to reuse one bean configuration details in another bean configurations like normal java classes inheritance inorder to improve reusability and inorder to reduce no of configurations in spring configuration file by declaring parent and chaild beans configurations.

To declare a particular bean configuration as parent to the present bean configuration then we have to use "parent" attribute in "<bean>" tag in beans configuration file, where "parent" attribute will take identity of the respective bean definition.

EX:

WishBean.java

```
package com.durgasoft.beans;
public class WishBean {
        private String wish_Message;
        private String name;

        public void init(){
                System.out.println("WishBean Initialization");
        }
        public void destroy(){
                System.out.println("WishBean Destruction");
        }

        setXXX()
      getXXX()
}
```

HelloBean.java

```
package com.durgasoft.beans;
```


```
public class HelloBean {
        private String wish_Message;
        private String name;

        public void init(){
                System.out.println("HelloBean Initialization");
        }
        public void destroy(){
                System.out.println("HelloBean Destruction");
        }

        setXXX()
      getXXX()

        public String sayHello(){
                return wish_Message+name;
        }
}
```

WelcomeBean.java

```
package com.durgasoft.beans;
public class WelcomeBean {
        private String wish_Message;
        private String name;

        public void init(){
        System.out.println("WelcomeBean Initialization");
        }
        public void destroy(){
                System.out.println("WelcomeBean Destruction");
        }

        setXXX()
```


```
    getXXX()

      public String sayWelcome(){
            return wish_Message+name;
      }
}
```

spring_beans_config.xml

```
<beans>
 <bean id="wishBean" class="com.durgasoft.beans.WishBean" init-method="init"
destroy-method="destroy">
   <property name="wish_Message" value="Durga Software  Solutions"/>
   <property name="name" value="Anil"/>
 </bean>
 <bean id="helloBean" class="com.durgasoft.beans.HelloBean" parent="wishBean">
      <property name="wish_Message" value="Hello "/>
 </bean>
 <bean id="welcomeBean" class="com.durgasoft.beans.WelcomeBean"
parent="wishBean">
      <property name="wish_Message" value="Welcome "/>
 </bean>
</beans>
```

Test.java

```
package com.durgasoft.test;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.durgasoft.beans.HelloBean;
import com.durgasoft.beans.WelcomeBean;
public class Test {
public static void main(String[] args)throws Exception {
```


```
AbstractApplicationContext context=new
ClassPathXmlApplicationContext("/com/durgasoft/cfgs/    spring_beans_config.xml");
HelloBean hello=(HelloBean)context.getBean("helloBean");
System.out.println(hello.sayHello());
System.out.println();
WelcomeBean welcome=(WelcomeBean)context.getBean("welcomeBean");
System.out.println(welcome.sayWelcome());
context.registerShutdownHook();
}
}
```

In the above Beans Inheritance, it is possible to declare parent bean definition as a template, that is, a set of configurations which we want to apply to chaild definintions, not to have its own bean class and not to apply to its own bean class.

To declare a bean definition as template we have to use "abstract" attribute with "true" value in <bean> tag , in this context, it is not required to use "class" attribute in bean tag.

EX:

```
<beans>
 <bean id="wishBean" abstract="true" .... >

 </bean>
 <bean id="helloBean" class="HelloBean" parent="wishBean">

 </bean>
 <bean id="welBean" class="WelcomeBean" parent="wishBean">

 </bean>
</beans>
```

Example:

WishBean.java


```
package com.durgasoft.beans;


public class WishBean {
	private String wish_Message;
	private String name;

	public void init(){
		System.out.println("WishBean Initialization");
	}
	public void destroy(){
		System.out.println("WishBean Destruction");
	}
	public String getWish_Message() {
		return wish_Message;
	}
	public void setWish_Message(String wish_Message) {
		this.wish_Message = wish_Message;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

HelloBean.java

```
package com.durgasoft.beans;

public class HelloBean {
	private String wish_Message;
	private String name;
```


```
	public void init(){
		System.out.println("HelloBean Initialization");
	}
	public void destroy(){
		System.out.println("HelloBean Destruction");
	}



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

	public String sayHello(){
		return wish_Message+name;
	}
}
```

WelcomeBean.java

```
package com.durgasoft.beans;

public class WelcomeBean {
	private String wish_Message;
```


```
	private String name;

	public void init(){
	System.out.println("WelcomeBean Initialization");
	}
	public void destroy(){
		System.out.println("WelcomeBean Destruction");
	}


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

	public String sayWelcome(){
		return wish_Message+name;
	}
}
```

Test.java

```
package com.durgasoft.test;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.durgasoft.beans.HelloBean;
```

BY NAGOOR BABU

```
import com.durgasoft.beans.WelcomeBean;
import com.durgasoft.beans.WishBean;
public class Test {
public static void main(String[] args)throws Exception {
AbstractApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");

//WishBean wish = (WishBean)context.getBean("wishBean");---> Exception
//System.out.println(wish);
HelloBean hello=(HelloBean)context.getBean("helloBean");
System.out.println(hello.sayHello());
System.out.println();
WelcomeBean welcome=(WelcomeBean)context.getBean("welcomeBean");
System.out.println(welcome.sayWelcome());
context.registerShutdownHook();
}
}
```

applicationContext.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"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd">

 <bean id="wishBean" abstract="true" init-method="init" destroy-method="destroy">
    <property name="wish_Message" value="Durga Software  Solutions"/>
    <property name="name" value="Anil"/>
 </bean>
 <bean id="helloBean" class="com.durgasoft.beans.HelloBean" parent="wishBean">
        <property name="wish_Message" value="Hello "/>
 </bean>
```


CONTACT US:

Mobile: +91- **8885 25 26 27**

+91- **7207 21 24 27/28**

US NUM: **4433326786**

Mail ID: **durgasoftonlinetraining@gmail.com**

WEBSITE: **www.durgasoftonline.com**

FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,


```
 <bean id="welcomeBean" class="com.durgasoft.beans.WelcomeBean"
parent="wishBean">
        <property name="wish_Message" value="Welcome "/>
 </bean>
</beans>
```

Nested Beans:

Declaring a bean configuration in another bean configuration is called as Inner Beans or Nested Beans.

IN Bean class, if we declare any property of User defined class type then we have to use <bean> tag for the user defined class as value to the respective property under <property> tag.

```
<beans>

 <bean .... >

   <property name="variable of USer defined class type">
     <bean id="--" class="---"/>
   </property>

 </bean>

</beans>
```

EX:

Account.java

```
package com.durgasoft.beans;
public class Account {
        private String accNo;
        private String accName;
        private String accType;
```


```
        private long balance;
    setXXX()
    getXXX()
}
```

Employee.java

```
package com.durgasoft.beans;
public class Employee {
        private String eid;
        private String ename;
        private float esal;
        private String eaddr;
        private Account acc;

    setXXX()
    getXXX()

        public void displayEmpDetails(){
                System.out.println("Employee Details");
                System.out.println("---------------------");
                System.out.println("Employee Id     :"+eid);
                System.out.println("Employee Name   :"+ename);
                System.out.println("Employee Salary :"+esal);
                System.out.println("Employee Address:"+eaddr);
                System.out.println();
                System.out.println("Account Details");
                System.out.println("---------------------");
                System.out.println("Account Number  :"+acc.getAccNo());
                System.out.println("Account Name    :"+acc.getAccName());
                System.out.println("Account Type    :"+acc.getAccType());
                System.out.println("Account Balance :"+acc.getBalance());
        }
}
```

applicationContext.xml

```
<beans>
 <bean id="emp" class="com.durgasoft.beans.Employee">
    <property name="eid" value="E-111"/>
       <property name="ename" value="Durga"/>
       <property name="esal" value="10000"/>
       <property name="eaddr" value="Hyd"/>
       <property name="acc">
             <bean id="account" class="com.durgasoft.beans.Account">
                   <property name="accNo" value="abc123"/>
                   <property name="accName" value="Durga"/>
                   <property name="accType" value="Savings"/>
                   <property name="balance" value="20000"/>
             </bean>
       </property>
 </bean>
</beans>
```

Test.java

```
package com.durgasoft.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.durgasoft.beans.Employee;
public class Test {
public static void main(String[] args)throws Exception {
ApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
Employee emp=(Employee)context.getBean("emp");
emp.displayEmpDetails();
}
}
```

BY NAGOOR BABU

BeanPostProcessor:

In SPring Framework, when we activate ApplicationContext container then ApplicationContext container will perform the following actions automatically with out having developers interaction.

1. ApplicatinContext will read beans definitions from beans configuration file.
2. ApplicationContext will recognize beans classes and creates objecs for bean classes
3. ApplicationContext will perform initialization by executing initialization methods.
4. When IOCContainer is shutdown then bean objects are destroyed.

IN the above Bean lifecycle, if we want to provide customizations over beans initializations then we have to use a predefined interface provided by Spring framework like "org.springframework.beans.factory.config.BeanPostProcessor".

BeanPostProcessor contains the following two methods .
1.public void postProcessBeforeInitialization(Object bean, String name) throws BeansException
--> This method will be executed just before performing bean initialization and immediatly after bean instantiation, that is, before executing init() method if we provide custom initialization method init().

2. public void postProcessAfterInitialization(Object bean, String name) throws BeansException
--> This method will be executed immediatly after performing beans initialization, that is, after executing init() method if we provide custom initialization method init().

To use BeanPostPrcessor in Spring applications we have to declare an user defined class as an implementation class to BeanPostProcessor interface , we must implement the above specified methods and we must configure implementation class in bean configuration file. If we provide like this then container will detect BeanPostProcessor implementation automatically and IOCContainer will execute the BeanPostProcessor methods in the respective locations of the beans lifecycle.

CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

EX:

BeanPostProcessorImpl.java

```
package com.durgasoft.processor;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
import com.durgasoft.beans.Customer;
public class BeanPostProcessorImpl implements BeanPostProcessor {
@Override
public Object postProcessAfterInitialization(Object bean, String name) throws
BeansException {
System.out.println("postProcessAfterInitialization()");
Customer cust=(Customer)bean;
cust.setCmobile("91-9988776655");
return cust;
}
@Override
public Object postProcessBeforeInitialization(Object bean, String name) throws
BeansException {
System.out.println("postProcessBeforeInitialization()");
Customer cust=(Customer)bean;
cust.setCemail("durga@durgasoft.com");
return cust;
}
}
```

Customer.java

```
package com.durgasoft.beans;
public class Customer {
        private String cid;
        private String cname;
```


```
        private String caddr;
        private String cemail;
        private String cmobile;

        public Customer(){
                System.out.println("Customer Bean Object Creating");
        }
        public void init(){
                System.out.println("Customer Bean Object Initialization through init()
method");
        }
        public void destroy(){
                System.out.println("Customer Object Destroying through destroy()
method");
        }

        setXXX()
        getXXX()

        public void getCustomerDetails(){
                System.out.println("Customer Details");
                System.out.println("-------------------");
                System.out.println("Customer Id     :"+cid);
                System.out.println("Customer Name   :"+cname);
                System.out.println("Customer Address:"+caddr);
                System.out.println("Customer Email  :"+cemail);
                System.out.println("Customer Mobile :"+cmobile);
        }
}
```

applicationContext.xml

```
<beans>
```


```
 <bean id="cust" class="com.durgasoft.beans.Customer" init-method="init"      destroy-
method="destroy">
                <property name="cid" value="C-111"/>
                <property name="cname" value="Durga"/>
                <property name="caddr" value="Hyd"/>
 </bean>
 <bean class="com.durgasoft.processor.BeanPostProcessorImpl"/>
</beans>
```

Test.java

```
package com.durgasoft.test;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.durgasoft.beans.Customer;
public class Test {
public static void main(String[] args)throws Exception {
AbstractApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
Customer cust=(Customer)context.getBean("cust");
cust.getCustomerDetails();
context.registerShutdownHook();
}
}
```