**Overview of Spring Bean Life Cycle**

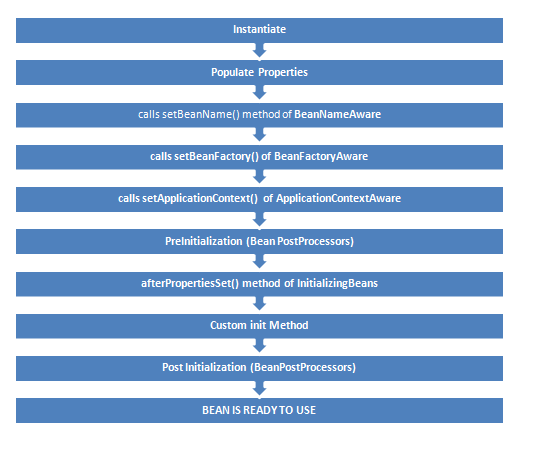
Life of traditional java objects starts on calling new operator which instantiates the object and finalize() method is getting called when the object is eligible for garbage collection. Life cycle of Spring beans are different as compared to traditional java objects.

Spring framework provides the following ways which can be used to control the lifecycle of  bean:

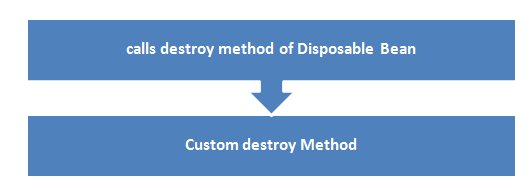
1. InitializingBean and DisposableBean callback interfaces
2. Bean Name, bean factory and Application Context  Aware interfaces for specific behavior
3. custom init() and destroy() methods in bean configuration file

For annotation based configurations -

@PostConstruct and @PreDestroy annotations

Below diagram shows the complete lifecycle methods (from instantiate to Ready To use )

Following diagram shows the method calling  at the time of destruction.



**7.2 InitializingBean and DisposbleBean callback interfaces**

* InitalizingBean interface is defined under org.springframework.beans.factory package and declares a single method where we can  be used to add any initialization related code. Any bean implementing InitalizingBean needs to provide an implementation of afterPropertiesSet() method. Signature of method is:

                  void afterPropertiesSet() throws Exception;

* Similarly DisposableBean interface is defined under the org.springframework.beans.factory and declares a single method which gets executed when bean is destroyed and can be used to add any cleanup related code. Any bean implementing DisposableBean needs to provide an implementation of destroy() method. Signature of method is :

                  void destroy() throws Exception;

         This approach is simple to use but it’s not recommended because it will create tight coupling with the Spring framework in our bean implementations.

**7.2.1 Example-**

Lets write an example to implement InitalizingBean and DisposableBean interface

**Solution:**

a) Write a PersonBean which implements InitializingBean and DisposableBean interface like below

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 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 | import org.springframework.beans.factory.DisposableBean;  import org.springframework.beans.factory.InitializingBean;  public class PersonBean implements InitializingBean,DisposableBean{       private String name;     public PersonBean()     {        System.out.println("Constructor of person bean is called !! ");     }     @Override     public void destroy() throws Exception     {       System.out.println("destroy method of person bean is called !! ");     }     @Override       public void afterPropertiesSet() throws Exception     {     System.out.println("afterPropertiesSet method of person bean is called !! ");     }     public String getName() {      return name;     }     public void setName(String name) {      this.name = name;     }  } |

b) Create a beans.xml file in src directory to define the PersonBean

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?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">         <bean id="personBean" class="PersonBean" >           <property name="name" value="Dummy Person"/>           </bean>  </beans> |

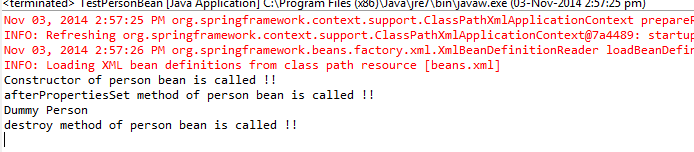
c) Create TestPersonBean class which will just loads the beans.xml and test the person bean life cycle

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | import org.springframework.context.ApplicationContext;  import org.springframework.context.support.AbstractApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;  public class TestPersonBean {      public static void main(String[] args) {           ApplicationContext context =                  new ClassPathXmlApplicationContext("beans.xml");         PersonBean bean = (PersonBean)context.getBean("personBean");             System.out.println(bean.getName());         ((AbstractApplicationContext) context).registerShutdownHook();      }  } |

d)  Run the Program

You will see below output. Initialization and Destroy methods are getting called.

****

**7.3 Bean Name, Factory, Application context Aware interfaces**

Several times functionality requires infrastructure or we can say application context information in a bean. To achieve such functionalities ,Spring framework  provides  a range of Aware interfaces Each interface requires us to implement a method to inject the dependency in bean. Most commonly used are –

* **BeanFactoryAware** - This interface provides setBeanFactory() method  that supplies the owning bean factory instance to the bean. Signature of the method is

void setBeanFactory(BeanFactory beanFactory) throws BeansException

* **BeanNameAware**- This interface provides setBeanName() method which sets the name of the bean in the bean factory that created this bean. Signature of the method is-

void setBeanName(String name);

* **ApplicationContextAware** -This interface provides setApplicationContext() method  that supplies the owning application context instance to the bean. Signature of the method is

void setApplicationContext(ApplicationContext applicationContext) throws BeansException

**7.3.1 Example**

Lets write an example to implement Aware interfaces

**Solution:**

 a) Create  a class (AwareBean) which implements ApplicationContextAware, BeanNameAware and BeanFactoryAware

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 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 | import java.util.Arrays;  import org.springframework.beans.BeansException;  import org.springframework.beans.factory.BeanFactory;  import org.springframework.beans.factory.BeanFactoryAware;  import org.springframework.beans.factory.BeanNameAware;  import org.springframework.context.ApplicationContext;  import org.springframework.context.ApplicationContextAware;    public class AwareBean implements ApplicationContextAware,BeanNameAware,BeanFactoryAware{  @Override  public void setBeanFactory(BeanFactory beanFactory) throws BeansException {         System.out.println("setBeanFactory method of Aware bean is called");         System.out.println("setBeanFactory:: Aware bean singleton="                  + beanFactory.isSingleton("awareBean"));  }  @Override  public void setBeanName(String beanName) {       System.out.println("setBeanName method of Aware bean is called");       System.out.println("setBeanName:: Bean Name defined in context="                      + beanName);  }  @Override  public void setApplicationContext(ApplicationContext applicationContext)          throws BeansException {          System.out.println("setApplicationContext method of Aware bean is called");          System.out.println("setApplicationContext:: Bean Definition Names="                  + Arrays.toString(applicationContext.getBeanDefinitionNames()));  }  } |

**b)**Create a beans.xml file in src directory to define the AwareBean

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 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>"      xsi:schemaLocation="http://www.springframework.org/schema/beans      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">        <bean id="awareBean" class="AwareBean" >      </bean>  </beans> |

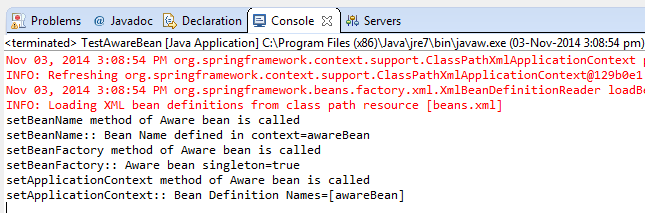
c)  Create TestAwareBean class which will just loads the beans.xml and test the aware life cycle

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | import org.springframework.context.ApplicationContext;  import org.springframework.context.support.AbstractApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    public class TestAwareBean {      public static void main(String[] args) {               ApplicationContext context =                  new ClassPathXmlApplicationContext("beans.xml");             AwareBean bean = (AwareBean)context.getBean("awareBean");             ((AbstractApplicationContext) context).registerShutdownHook();       }  } |

d) Run the Program

You will see below output.



**7.4 Custom init() and destroy() methods in bean configuration file**

Implementing InitalizingBean and DisposableBean interface is simple to use but create tight coupling with the Spring framework in our bean implementations.

Alternatively we can **init-method** and **destroy-method** attribute values for the bean in the spring bean configuration file. This is the recommended approach because of no direct dependency to spring framework and we can create our own methods.

*Note: Both post-init and pre-destroy methods should have no arguments but they can throw Exceptions*

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <beans>      <bean id="bean\_id" class="bean.class"        init-method="customInitmethod"       destroy-method="customDestroymethod">    </bean>  </beans> |

We can configure the default init-method  and destroy-method which will be applied on all the beans .They are useful when we have a pattern of defining common method names such as init() and destroy() for all your beans consistently.

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 1  2  3  4 | <beans default-init-method=”customDefaultInitMethod” default-destroy-method=”customDefaultDestroyMethod” >      <bean id="bean\_id" class="bean.class" >  </bean>  </beans> |

**7.4.1 Example**

Write and example to show the init-method and destroy-method

**Solution**

 a)Write a class CustomLifeCycleMehodBean

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 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 | public class CustomLifeCycleMethodBean {  private String name;        public CustomLifeCycleMethodBean()      {          System.out.println("Constructor of  bean is called !! ");      }        public void customDestroy() throws Exception {            System.out.println("custom destroy method of  bean is called !! ");      }        public void customInit() throws Exception {          System.out.println("custom Init  method of  bean is called !! ");      }        public String getName() {          return name;      }        public void setName(String name) {          this.name = name;      }    } |

 b) Create a beans.xml file in src directory to define the CustomMethodLifeCycleBean

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 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>"      xsi:schemaLocation="http://www.springframework.org/schema/beans      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">            <bean id="customLifeCycleMethodBean"                 class="CustomLifeCycleMethodBean"                 init-method="customInit"                 destroy-method="customDestroy">          <property name="name" value="custom methods bean" ></property>          </bean>  </beans> |

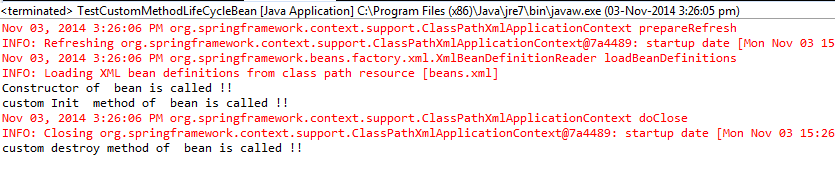
c) Create TestCustomMethodLifeCycleBean class which will just loads the beans.xml and test the custom methods life cycle

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | import org.springframework.context.ApplicationContext;  import org.springframework.context.support.AbstractApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    public class TestCustomMethodLifeCycleBean {       public static void main(String[] args) {            ApplicationContext context =              new ClassPathXmlApplicationContext("beans.xml");            CustomLifeCycleMethodBean bean = (CustomLifeCycleMethodBean)context.getBean("customLifeCycleMethodBean");             ((AbstractApplicationContext) context).registerShutdownHook();        }  } |

d)Run the Program

 You will see below output and custom life cycle methods are getting called

****

**7.4.2 – Example**

Write an example to demonstrate global init and destroy methods

**Solution**

a)Write a class CustomGlobalLifeCycleMehodBean

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class CustomGlobalLifeCycleMehodBean {      public CustomGlobalLifeCycleMehodBean()     {      System.out.println("Constructor of  bean is called !! ");     }     public void globalCustomDestroy() throws Exception {      System.out.println("global custom destroy method of  bean is called !! ");     }     public void globalCustomInit() throws Exception {      System.out.println("global custom Init  method of  bean is called !! ");     }  } |

**b)**Create a beans.xml file in src directory to define the CustomGlobalMethodLifeCycleBean

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?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"      default-init-method="globalCustomInit"      default-destroy-method="globalCustomDestroy">        <bean id="customGlobalLifeCycleMethodBean"            class="CustomGlobalLifeCycleMehodBean" />  </beans> |

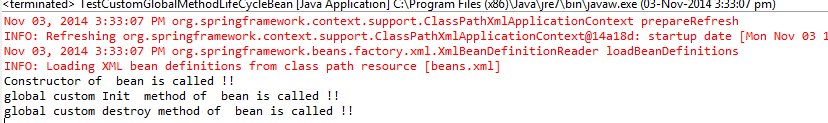
 c) Create TestCustomMethodLifeCycleBean class which will just loads the beans.xml and test the custom methods life cycle

[?](http://wideskills.com/spring/spring-bean-lifecycle)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | import org.springframework.context.ApplicationContext;  import org.springframework.context.support.AbstractApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    public class TestCustomGlobalMethodLifeCycleBean {      public static void main(String[] args) {               ApplicationContext context =                  new ClassPathXmlApplicationContext("beans.xml");             CustomGlobalLifeCycleMehodBean bean = (CustomGlobalLifeCycleMehodBean)context.getBean("customGlobalLifeCycleMethodBean");             ((AbstractApplicationContext) context).registerShutdownHook();       }  } |

d)Run the Program

You will see below output and global custom life cycle methods are getting called

****

**7.5 @PostConstruct and @PreDestroy annotations**

Spring 2.5 onwards we can use annotations to specify life cycle methods using @PostConstruct and @PreDestroy annotations.