[Java Spring Bean Lifecycle](http://javabeginnerstutorial.com/spring-framework-tutorial/java-spring-bean-lifecycle/)

* 27. Jun
* /
* [Spring](http://javabeginnerstutorial.com/category/spring-framework-tutorial/)
* /
* [6 Comments](http://javabeginnerstutorial.com/spring-framework-tutorial/java-spring-bean-lifecycle/#comments)

Spring Beans are Instantiated / Managed by [**Spring IoC Container**](http://javabeginnerstutorial.com/spring-framework-tutorial/spring-ioc-container-overview/).  These beans can be created by providing bean specific *configuration metadata* to container. Configuration Metadata can be provided in any of below formats.

1. *XML*
2. *Annotation*
3. *Java Code*

Bullet Points

* Container will contain beans as long as they are required by Application.
* Beans created outside Spring container can also be registered with AC(*Application Context*).
* *BeanFactory* is root interface for accessing the *bean container*. Other interfaces are also available for specific purpose.
* *BeanFactory* is a central registry of application components(Beans).
* These component(Beans) have **lifecycle interfaces and methods** which will be invoked in some order before Bean can be *handed over to application* and before Bean is getting *destroyed*.

Bean LifeCycle :

When bean is initialized it might require to perform some activity before it can come into use able state(State in which application can use it) and when bean is getting destroyed there might be some cleanup activity required for given bean. These activities are known as bean Lifecycle.

Standard bean lifecycle interfaces & there standard order of execution are given below..  
**1-**   IoC container will look for the *configuration metadata* of given Bean.  
**2-**   Once find, container will create the instance of Bean(Using reflection API).  
**3-**   After instance creation dependency will be injected(DI).

If Bean Class  implements any of the below interface then corresponding method will be invoked in below order(**Point 4 – 13**).

**4-**   setBeanName method of *BeanNameAware* class. It sets the name of the bean in the bean factory that created this bean.  
**5-**   setBeanClassLoader method of *BeanClassLoaderAware* class. Callback that supplies the bean to a bean instance.  
**6-**   setBeanFactory  method of *BeanFactoryAware* class. Callback that supplies the owning factory to a bean instance.

Below method execution will be applicable when running in an application context. (**Points 7 – 11**)

**7-**   setResourceLoader  method of *ResourceLoaderAware* class. It set the ResourceLoader that this object runs in.  
**8-** setApplicationEventPublisher  method of *ApplicationEventPublisherAware* class. Set the ApplicationEventPublisher that this object runs in.  
**9-**   setMessageSource method of *MessageSourceAware* class. Set the MessageSource that this object runs in.  
**10-**   setApplicationContext method of *ApplicationContextAware* class. Set the ApplicationContext that this object runs in.  
**11-**   setServletContext method of *ServletContextAware*. Set the ServletContext that this object runs in.

**12-**   postProcessBeforeInitialization method of *BeanPostProcessor*. Apply this BeanPostProcessor to the given new bean instance before any bean initialization callbacks.  
**13-**   afterPropertiesSet method of *InitializingBean*. Invoked by a BeanFactory after it has set all bean properties supplied.

In case Bean class has custom init method defined(via init-method attribute)

**14-**   Custom **init method** will be invoked.  
**15-**   postProcessAfterInitialization methods of *BeanPostProcessors*. Apply this BeanPostProcessor to the given new bean instance after any bean initialization callbacks

When Bean Factory is getting shut down following lifecycle methods will be executed.

**1-**   DisposableBean’s **destroy** method. Invoked by a BeanFactory on destruction of a singleton.  
**2-**   **Custome destroy** method will be executed if there is any defined via destroy-method attributes