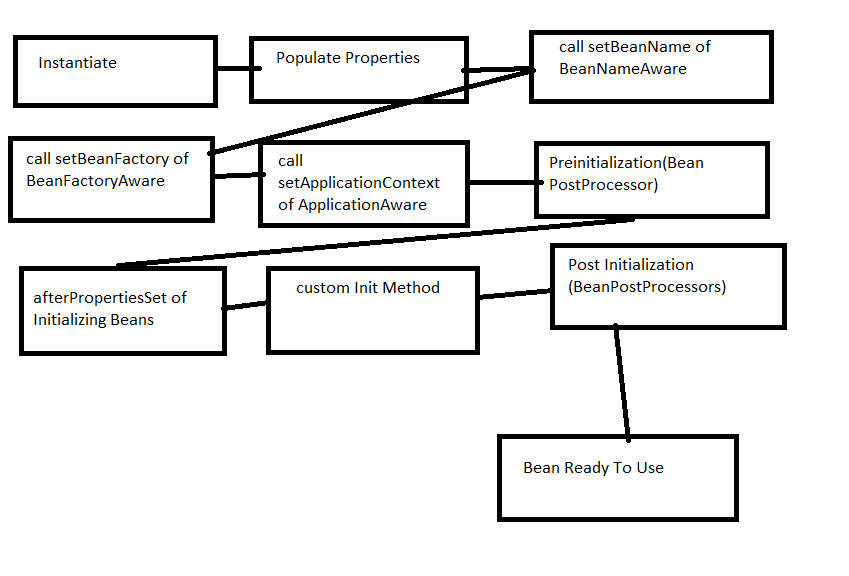
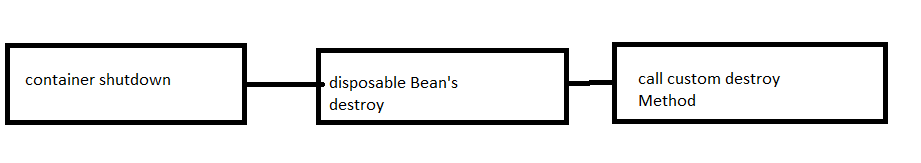
Spring IOC container is responsible for Instantiating, Initializing, and wiring beans.

The container also manages the life cycle of beans





1. Once the bean is instantiated, the spring needs to perform some initialization to get the bean into a usable state and needs to clean up resources before a bean is removed from the container.

Part -1

* Different stages of the bean go through after instantiation until it is ready for use.
* Container instantiates a bean by calling its constructor and then populates its properties.

Part -2

* Shows what happens to bean once the Spring IOC container shutdown.
* When the container shutdown the container calls the bean to enable it to perform any required tasks before the bean is destroyed.

**Aware Interfaces**

This interface is used to access the Spring framework Infrastructure.

It is largely used by Spring and rarely used by Spring programmers.

There are 3 interfaces

* *BeanFactoryAware* 🡪 Provides *setBeanFactory*(), a callback that supplies the owning factory to the bean instance
* *BeanNameAware*: *setBeanAware*() callback of this interface supplies the name of the bean.
* *ApplicationContextAware*: The *setApplicationContext*() method of this interface provides the ApplicationContext object of this bean.

**BeanPostProcessor:**

Spring provides *BeanPostProcessor* interface that gives you to tap into the Spring context life cycle and interact with beans as they are processed.

*BeanPostProcessor* interface contains 2 methods

*postProcessBeforeInitialization()*:

Spring calls this method after methods of aware interfaces and before initialization callbacks like InitializingBean’s *afterPropertiesSet*() or custom init method.

*postProcessAfterInitialization():*

Spring calls this method after any bean initialization callbacks.At the runtime, Spring will inject the new bean instance and the name of the bean to both the methods

***InitializingBean* and *DisposableBean* Callback Interfaces.**

It has 2 callback interfaces:

*InitializingBean*: it declares *afterPropertiesSet()* method which can be used to write the initialization logic. The container calls the method after properties are set.

*Disposablebean*: declares the *destroy*() method which can be used to write any cleanup code. The container calls this method during destruction in shutdown.

Custom Init and Destroy Method

* We can specify init-method and *destroy-method* attributes in the tag.
* Both the attributes specify custom methods in the bean class.
* Method declared in the init-method attribute is called after the spring initializes bean properties through setter or constructor arguments. This method can be used to validate the injected properties or perform any other tasks.
* Spring calls the method declared in the destroy-method attribute just before the bean is destroyed.

Summary:

* Personally, I don’t prefer using InitializingBean and DisposableBean interfaces.
* Primarily because it tight couples your code to Spring.
* A better approach is specifying init-method and destroy-method attributes in your bean configuration file.