**Lifecycle of ApplicationContext**

• **ApplicationContext** is the main object for bean configuration.

• **ConfigurableApplicationContext** is its sub-interface that adds configuration and lifecycle methods.

• Three main lifecycle methods are:

• **refresh**()

• **close**()

• **registerShutdownHook**()



**ApplicationContext** icin spring dokümanına bakmak daha yararlı olabilir. Ayrıca burada bu context sadece read-onlydir. Client’a haber vermek, bilgi gecmek vs gibi amaclar icin bu yeterlidir.

Yukarıdaki hiyerarside amac interface security sağlamak yani her seyi applicationContext üstüne yıkmak değil de belli amaclara hizmet edenleri ve metotlarını ayırmak. Böylece daha security ve anlasılabilir bir durum oluyor. Solid’in I’sına denk geliyor.

**ConfigurableApplicationContext ‘ın bu** ApplicationContexti implement ediyor. İcinde ise configuration lifecycle methodlraı vardır.

Lifecycle metot olarak temelde üc tane vardır.

• **refresh**()

• **close**()

• **registerShutdownHook**()

Bunlar oldugu zaman eventler fırlatılacak. Post-initialization ve pre-destruction eventleri.

**AbstractApplicationContext**

• **AbstractApplicationContext** is the abstract base class for all context objects and implements all lifecycle methods:

• **refresh**() either loads a new persistent representation of the configuration or refreshes new one discarding the old one.

• **close**() closes the context, releasing all resources and locks that the implementation might hold.

**refresh** Configurationu yeniliyor, yeni bir conf yüklüyor,

**close** Contexti kapatıyor, resource release ediyor. Varsa locklar bunları release ediyor.

• **registerShutdownHook**() registers a shutdown hook named

**SpringContextShutdownHook** with the JVM runtime, which is represented

by **java.lang.Runtime** object.

• It closes the context on JVM shutdown unless it has already been closed before.

• **Runtime** class has **addShutdownHook**(**Thread hook**) method that

allows applications register a **Thread** object so that just before shutting down

itself **Runtime** object executes them in **java.lang.Shutdown** class.

• **close**() method removes any registered shutdown hook.

JVM’ e kapanacağın zaman cagır dememizi sağlar.

Dogrudan close yapmak yerine JVM’e kapanırken haber ver, JVM’in kapanacağını görelim çünkü beanlerin temizliği yapılacak.

**addShutdownHook metodunu cagırıyor ve çengel atmıs gibi düsünebiliriz.**

• **AbstractApplicationContext** uses the **Template** design pattern.

• It has three abstract methods and even though it determines the main algorithm of the lifecycle of bean factory, it leaves the implementation of these three methods to concrete sub-classes.

• Those abstract methods that manage bean factory are:

• **ConfigurableListableBeanFactory** **getBeanFactory**()

• **void closeBeanFactory**()

• **void refreshBeanFactory**()

Bir işi yapmanın yolu bellidir. Önce 1.is, sonra 2.is, sonra 3.isi yap gibi. Ama 2.isi yaparken çok fazla is olabilir bu durumda 2.isi abstract olarak bırakabiliriz.

Yani template metot ya da design burada bir şablon bu şablon da nasıl çalışacağı biliniyordur ama bazı adımlar çok fazla işlem vs sebeplerden dolayı sub-classlarda halledilecektir.

• Even though it has an internal bean factory it leaves to its sub-classes

issues regarding the bean factory such as how it is created, closed,

refreshed, and where the storage for bean definitions should exists.

• And it manages all event and notification mechanisms by automatically

registering **BeanFactoryPostProcessor**, **BeanPostProcessor**,

and **ApplicationListener** which are defined as beans in the

context.

• **BeanFactoryPostProcessors**, **BeanPostProcessors**, and

**ApplicationListeners** will be explained soon.

• Have a look at the source code of following methods, which are

template methods, in **AbstractApplicationContext**:

• **refresh**(): Makes calls to abstract **refreshBeanFactory**() and

**getBeanFactory**() methods.

• **close**(): Makes call to abstract **closeBeanFactory**() method.

• **registerShutdownHook**(): Makes call to abstract

**closeBeanFactory**() method.

Bean olusturmaya cagırdıktan sonra event olusturur, registering **BeanFactoryPostProcessor**, **BeanPostProcessor**,and **ApplicationListener** which are defined as beans in the bunları abstract olmayanlarda cagırıyor, ama bazıları abstract.

**refresh**(): Makes calls to abstract **refreshBeanFactory**() and

**getBeanFactory**() methods. Refresh metodu yukarıdaki iki factory metotlarına call yapıyor.

**AbstractRefreshableApplicationContext**

• **AbstractRefreshableApplicationContext** is the abstract base

class that allows multiple **refresh**() calls.

• It creates a new bean factory instance for each **refresh**() call.

• Typically a new **refresh**() call is necessary to load new bean

definitions from new configuration locations.

• It has only one abstract method to load bean definitions:

• **loadBeanDefinitions(DefaultListableBeanFactory**

**beanFactory)**

**Her refreshde yeni bir beanfactory olusturulur bunun nedeni de genelde yeni bir bean definition yükleyeceğiz demektir, eskisini kapatıp.**

**Çünkü programatik olarak yüklemek istediğimizde.**

**Not:** Burada refreshExample örneğinde olduğu gibi kendimiz set edersek burada refreshe ihtiyaç vardır. Refresh etmeyeceğimiz zaman o zaman da beanlar available olmayacaktır.

Classpath ile source verirsek otomatik refresh ediyor.

Destroy edilmek spring anlamında artık takip edilmediğini, onunla ilgilenmediğini event anlamında fırlatmayacağını vs belirtir.

Burada memoryde duruyor, ve lifecycle takip etmiyor ama memoryde olmasına rağmen takip etmiyor.