Refresh özet :

Refresh ile ya çoğunlukla constructor call’ında yeni bir configuration yapıyorduk, AC context objeleri refreshe çağırarak onun geçerli olmasını sağlıyordu.

Birden fazla refresh ettiğimiz zaman yeni configurasyonu çağrılıyordu.

Ya da otomatik olarak yeni configurasyonu bize sunuyordu. Annotation yapılarındaki gibi.

**Close Giris**

• **close()** closes the application context.

• It also releases all resources and locks that the implementation might

hold.

• This includes destroying all cached singleton beans.

• When a context is closed its pre-destruction (or pre-destroy) callback is

also called on singletons.

Close application contexti kapatır.

Bütün beanlari destroy ediyor ve ac’i kapatıyor.

• Once the context is closed, it reaches its end of life and cannot be

refreshed, restarted or it doesn’t serve any bean anymore.

• For example trying to read an object from a closed context causes

**java.lang.IllegalStateException** with a **message** **BeanFactory**

**not initialized or already closed…**

• This method can be called multiple times without side effects,

subsequent **close()** calls on an already closed context will be

ignored.

Contexti kapattığımız zaman artık o contexti bir daha kullanamayız.

• Although IoC container calls post-contruct notifications after refreshing

the configuration on all beans it does not call pre-destroy notifications on prototypes.

• When the context is closed all beans, singleton or prototype, are

destroyed so that the context is not available to serve them anymore.

• What is different is about calling lifecycle destruction methods on

prototypes, they are destroyed but don’t get notification automatically.

Post-contruct’ları refresh ettiğimiz zaman bütün beanlar üzerinde cagırmasına rağmen pre-destroyu sadece singleton üstünde cagırır.

Singleton sadece pre-destroy notificationsları alırlar. Prototype’lar almazlar.

Bunun sebebi singletonun bir tane olması, prototype’n çok fazla olmasıdır.

• The client must clean up prototype beans and release expensive

resources that the prototype beans hold.

• Prototype beans can receive the pre-destruction events via a custom

post processor such as **DestructionAwareBeanPostProcessor**.

• **Spring** **says:** In some respects, the Spring container’s role in regard to

a prototype-scoped bean is a replacement for the **Java** **new** operator. All

lifecycle management past that point must be handled by the client.

Spring olarka prototype objelerde sadece create etmektir. New’dan sonraki bütün lifecycle management kısmını bize bırakıyor.

Bunun icin **DestructionAwareBeanPostProcessor gibi** interfaceler var burada prototypeların hangilerinin cagıracağımıza, cagırmayacağımıza karar verebiliriz.

**Removing Beans From Context**

• Beans can be manually removed from the context by calling

**removeBeanDefinition**() method on

**org.springframework.beans.factory.support.BeanDefinitionsRegistry.**

• When the bean instance is removed from **BeanDefinitionRegistry**

of the context manually then it won’t be available anymore from the

context.

Close cagırmadan Remove ile cagırmadan tek tek silebiliriz.

**Destroying Beans**

• When an **ApplicationContext** object is closed all beans it created

continue to reside in the memory of JVM as initialized but the context is

not available to serve any object or throw event anymore.

• When an **ApplicationContext** object is refreshed in its lifetime all

beans it created before the refresh continue to reside in the memory as

initialized but the context does not contain and manage them anymore.

• After the refresh **ApplicationContext** object starts serving new

beans registered with new configuration.

AC kapatıldığı zaman bütün beanler herhangi bir obje vermez, herhangi bir event vermez, diğer objeler springten bağımsız yaşamaya calısırlar.