**@Scope(ConfigurableBeanFactory.SCOPE\_SINGLETON):** Every time same instance is returned. This is default value

\*\*\* **Even the singleton bean will be different for different application context**

**@Scope(ConfigurableBeanFactory.SCOPE\_PROTOTYPE):**Every time different instance is created

|  |  |  |
| --- | --- | --- |
| Outer Class | Dependency |  |
| Singleton | Singleton | As Expected |
| Prototype | Prototype | As Expected |
| Prototype | Singleton | As Expected |
| Singleton | Prototype | We must add proxy |

When the outer class is Singleton, but the dependency is of scope Prototype, then the outer

class will consider the dependency also to be singleton even if you have marked it Prototype.

So to avoid this, we must add proxy to the dependency as follows:

**@Scope(value = ConfigurableBeanFactory.SCOPE\_PROTOTYPE,**

**proxyMode = ScopedProxyMode.TARGET\_CLASS)**

**\*\*\*\*We should try to keep the number of instances as low as possible, so use SINGLETON wherever possible**

=>For all scopes except Prototype, both @PostConstruct and @PreDestroy methods are called

=>But **for Prototype scope, @PreDestroy is NOT called**. In contrast to the other scopes, Spring does not manage the complete lifecycle of a prototype bean: the container instantiates, configures, and otherwise assembles a prototype object, and hands it to the client, with no further record of that prototype instance. The client code must clean up prototype-scoped objects and release expensive resources that the prototype bean(s) are holding.