Spring is one of the most popular Java EE frameworks. It is an open-source lightweight framework that allows Java EE developers to build simple, reliable, and scalable enterprise applications. It provides Aspect-oriented programming. It provides support for all generic and middleware services and mainly focuses on providing various ways to help you manage your business objects. It is a modularized framework where all the modules are designed on the concept called “**Dependency Injection**”.

**Dependency Injection**: Dependency Injection is a design pattern that allows the spring container to ‘inject’ objects into other objects or dependencies. In simple words, the control of creating objects and managing the spring components is taken care of by the Spring containers.

**Spring Containers**

Spring Framework provides two of the most fundamental and important packages, they are the org.springframework.beans and org.springframework.context packages. Code in these packages provides the basis for Spring’s Inversion of Control/Dependency Injection features. Spring containers are responsible for creating bean objects and injecting them into the classes. The two containers are namely,

* BeanFactory(I) – Available in org.springframework.beans.factory package.
* ApplicationContext(I) – Available in org.springframework.context package.

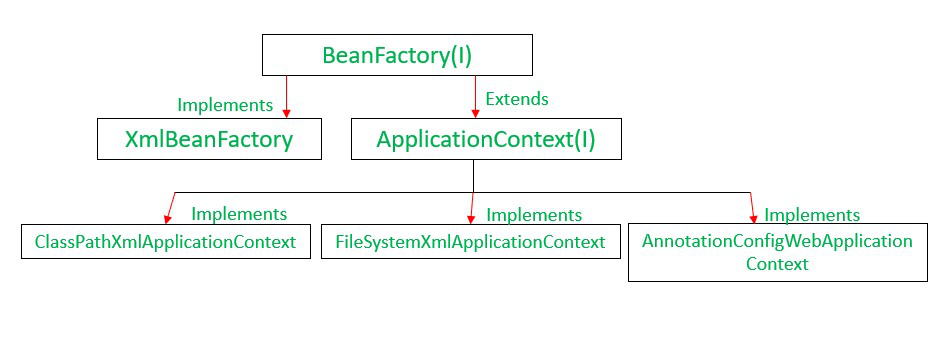
**The BeanFactory Interface**

This is the root interface for accessing a Spring bean container. It is the actual container that instantiates, configures, and manages a number of beans. These beans collaborate with one another and thus have dependencies between themselves. These dependencies are reflected in the configuration data used by the BeanFactory.

**The ApplicationContext Interface**

This interface is designed on top of the BeanFactory interface. The ApplicationContext interface is the advanced container that enhances BeanFactory functionality in a more framework-oriented style. While the BeanFactory provides basic functionality for managing and manipulating beans, often in a programmatic way, the ApplicationContext provides extra functionality like MessageSource, Access to resources, Event propagation to beans, Loading of multiple (hierarchical) contexts etc. There are so many implementation classes that can be used such as

1. **ClassPathXmlApplicationContext**
2. **FileSystemXmlApplicationContext**
3. **AnnotationConfigWebApplicationContext**



|  |  |
| --- | --- |
| **BeanFactory** | **ApplicationContext** |
| It is a fundamental container that provides the basic functionality for managing beans. | It is an advanced container that extends the BeanFactory that provides all basic functionality and adds some advanced features. |
| It is suitable to build standalone applications. | It is suitable to build Web applications, integration with AOP modules, ORM and distributed applications. |
| It supports only Singleton and Prototype bean scopes. | It supports all types of bean scopes such as Singleton, Prototype, Request, Session etc. |
| It does not support Annotations. In Bean Autowiring, we need to configure the properties in XML file only. | It supports Annotation based configuration in Bean Autowiring. |
| This interface does not provides messaging (i18n or internationalization) functionality. | ApplicationContext interface extends MessageSource interface, thus it provides messaging (i18n or internationalization) functionality. |
| BeanFactory does not support Event publication functionality. | Event handling in the ApplicationContext is provided through the ApplicationEvent class and ApplicationListener interface. |
| In BeanFactory, we need to manually register BeanPostProcessors and BeanFactoryPostProcessors. | The ApplicationContext automatically registers BeanFactoryPostProcessor and BeanPostProcessor at startup. |
| BeanFactory will create a bean object when the getBean() method is called thus making it Lazy initialization. | ApplicationContext loads all the beans and creates objects at the time of startup only thus making it Eager initialization. |
| BeanFactory interface provides basic features only thus requires less memory. For standalone applications where the basic features are enough and when memory consumption is critical, we can use BeanFactory. | ApplicationContext provides all the basic features and advanced features, including several that are geared towards enterprise applications thus requires more memory. |

**Important Point from test perspective**

The Spring Framework comes with two IOC containers – BeanFactory and ApplicationContext. The BeanFactory is the most basic version of IOC containers, and the ApplicationContext extends the features of BeanFactory.

**Lazy Loading vs. Eager Loading**

BeanFactory loads beans on-demand, while ApplicationContext loads all beans at startup. Thus, BeanFactory is lightweight as compared to ApplicationContext.

**Bean Factory:-** It is not going to create objects at the time of object creation.

**Application Context:-**  It is going to create objects at the time of object creation.

**Creating Object of BeanFactory**:

ClassPathResource resource = **new** ClassPathResource("com/citiustech/bean.xml");

BeanFactory factory = **new** ~~XmlBeanFactory~~(resource);

or

BeanFactory factory1 = **new** ~~XmlBeanFactory~~(**new** ClassPathResource("com/citiustech/bean.xml"));

**Creating Object of ApplicationContext**:

ApplicationContext context = new ClassPathXmlApplicationContext("com/citiustech/bean.xml");