The **Inversion of Control** is a process by which application defines the dependency and **Spring IoC Container** manages these processes as well life cycle of beans. These dependencies are then satisfied in **runtime**by the Spring Framework. The **IoC**container is the main component of the Spring framework. It provides the main **IoC**container and **AOP**framework. The core container of the Spring Framework provides important functionality including [**dependency injection**](https://www.dineshonjava.com/dependency-injection-in-spring/) and **bean life cycle**management.

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are:

* to instantiate the application class
* to configure the object
* to assemble the dependencies between the objects

There are two types of IoC containers. They are:

1. **BeanFactory**
2. **ApplicationContext**

Difference between BeanFactory and the ApplicationContext

The org.springframework.beans.factory.**BeanFactory** and the org.springframework.context.**ApplicationContext** interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

## Spring IoC Container

The Spring IoC Container actually comes with two distinct containers:  
**1.**[**Bean Factories**](https://www.dineshonjava.com/what-is-bean-factory-in-spring/)  
**2.**[**Application Context**](https://www.dineshonjava.com/application-context-in-spring/)

### [Bean Factories](https://www.dineshonjava.com/what-is-bean-factory-in-spring/)

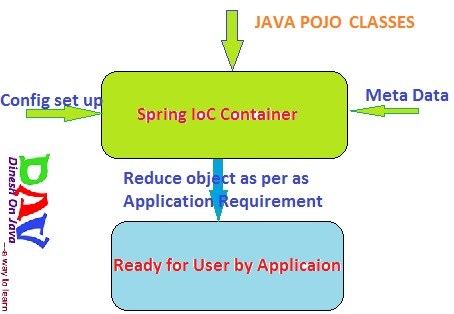
The **[BeanFactory](https://www.dineshonjava.com/what-is-bean-factory-in-spring/)**provides the configuration framework and basic functionality and support for [**Dependency Injection**](https://www.dineshonjava.com/dependency-injection-in-spring/). The **org.springframework.beans.factory.BeanFactory** is the actual representation of the Spring **IoC***container* that is responsible for containing and otherwise managing the aforementioned beans.

The **BeanFactory**interface is the central **IoC**container interface in Spring. Its responsibilities include instantiating or sourcing application objects, configuring such objects, and assembling the dependencies between these objects.

A Spring BeanFactory is like a factory class that contains a collection of beans. The Spring BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients.

* Spring ***BeanFactory***is able to create associations between collaborating objects as they are instantiated. This removes the burden of configuration from bean itself and the beans client.
* Spring ***BeanFactory***also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

This is the simplest container providing basic support for **DI** and defined by the **org.springframework.beans.factory.BeanFactory** interface. The ***BeanFactory***and related interfaces, such as ***BeanFactoryAware***, ***InitializingBean***, ***DisposableBean***, are still present in Spring for the purposes of backward compatibility with the large number of third-party frameworks that integrate with Spring.



### [Spring ApplicationContext Container](https://www.dineshonjava.com/application-context-in-spring/):

This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. This container is defined by the **org.springframework.context.ApplicationContext interface**.

 we used [Bean Factory](https://www.dineshonjava.com/what-is-bean-factory-in-spring/) container and discussed with example. Now we will discuss about the ApplicationContext and Using with in Example. ApplicationContext like [Bean Factory](https://www.dineshonjava.com/what-is-bean-factory-in-spring/)‘s Big Brother with some additional functionality such as **[AOP](https://www.dineshonjava.com/introduction-to-aop-in-spring/)concept**, **event notification** and it adds more enterprise-specific functionality such as the ability to resolve **textual messages from a properties file** and the ability to **publish application events to interested event listeners**. This container is defined by the **org.springframework.context.ApplicationContext** interface.

## ApplicationContext Implementations

The most commonly used **ApplicationContext** implementations are:

### FileSystemXmlApplicationContext

This container loads the definitions of the beans from an XML file. Here you need to provide the**full path of the XML bean configuration file** to the constructor.

FileSystemXmlApplicationContext context = new FileSystemXmlApplicationContext("F:/my workspace/springAppDemo/src/spring.xml");

**Using wildcard for file system:**

FileSystemXmlApplicationContext context = new FileSystemXmlApplicationContext("F:/\*my workspace\*/\*\*/src/\*-spring.xml");

### ClassPathXmlApplicationContext

This container loads the definitions of the beans from an XML file. Here you do not need to provide the full path of the XML file but you need to set **CLASSPATH** properly because this container will look bean configuration XML file in CLASSPATH.

ApplicationContext context = new ClassPathXmlApplicationContext("classpath:com/dineshonjava/sdnext/springConfig/spring.xml");

**Using wildcard for claspath:**

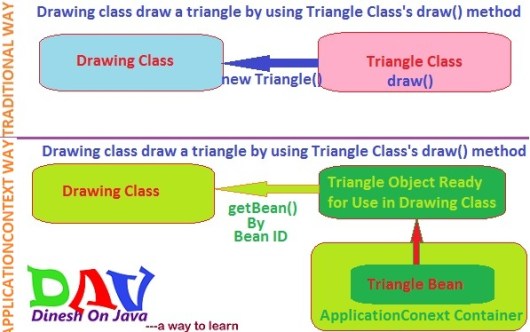
ApplicationContext context = new ClassPathXmlApplicationContext("classpath\*:com/dineshonjava/\*\*/springConfig/spring.xml");

ApplicationContext context = new ClassPathXmlApplicationContext("classpath\*:com/\*/\*\*/springConfig/\*-spring.xml");

### WebXmlApplicationContext

This container loads the XML file with definitions of all beans from within a web application.

ApplicationContext appContext = WebApplicationContextUtils.getWebApplicationContext(servletContext);



These are two ways to define the dependency injection in the spring application. Spring Dependency Injection (DI) design pattern is used to define the object dependencies between each other. There are following two types in [**dependency-injection**](https://www.dineshonjava.com/dependency-injection-in-spring/):

