IoC Container

1. [IoC Container](https://www.javatpoint.com/ioc-container)
2. [Using BeanFactory](https://www.javatpoint.com/ioc-container)
3. [Using ApplicationContext](https://www.javatpoint.com/ioc-container)

The Spring container is at the core of the Spring Framework.

Spring container uses DI to manage the components that make up an application. These objects are called Spring Beans.

The IoC container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction.

The IoC container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code. The following diagram represents a high-level view of how Spring works. The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.



The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets information’s 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**

### **Using BeanFactory**

The XmlBeanFactory is the implementation class for the BeanFactory interface. To use the BeanFactory, we need to create the instance of XmlBeanFactory class as given below:

|  |
| --- |
| 1. Resource resource = **new** ClassPathResource("applicationContext.xml"); 2. BeanFactory factory = **new** XmlBeanFactory(resource); |

The constructor of XmlBeanFactory class receives the Resource object so we need to pass the resource object to create the object of BeanFactory.

#### **Using ApplicationContext**

#### Types of ApplicationContext

The most commonly used ApplicationContext implementations are:

1. [**FileSystemXmlApplicationContext**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/support/FileSystemXmlApplicationContext.html) – 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.

ApplicationContext context = **new** FileSystemXmlApplicationContext

("C:/Users/ilan/workspace/HelloSpring/src/Beans.xml");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

1. [**ClassPathXmlApplicationContext**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/support/ClassPathXmlApplicationContext.html) – 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.

|  |
| --- |
| 1. ApplicationContext context = 2. **new** ClassPathXmlApplicationContext("applicationContext.xml"); |

1. [**WebXmlApplicationContext**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/context/support/XmlWebApplicationContext.html) – This container loads the XML file with definitions of all beans from within a web application.

***Bean Factory vs Application Context*** [sourced from spring docs](http://docs.spring.io/spring/docs/2.5.x/reference/beans.html#context-introduction-ctx-vs-beanfactory)



