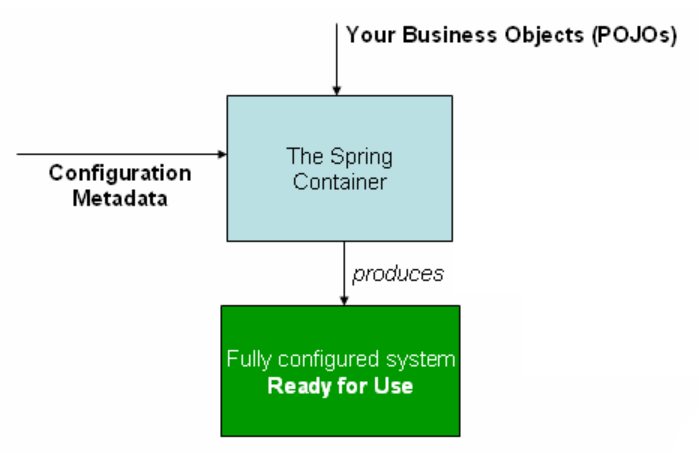
**IoC Containers**

**Introduction on IoC Containers**

One of the most important part of Spring Core Framework is Spring IoC Container. Spring IoC Container is mainly responsible for implementing ***Inversion of Control*** **(IoC)** i.e., to achieve loose coupling between the classes by creating the objects & injecting them dynamically during the run time.

Spring IoC Containers helps in instantiating the class & creating the object, configuring the object, assembling and managing the dependencies between the objects. The information required for creating, configuring & assembling the objects are obtained by reading the XML Configuration file or scanning the Spring Java Annotations used in the project.

**Architecture of IoC Containers**



Reference: <https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html>

From the above architecture diagram, we can interpret that the Spring Container consumes Configuration Metadata which is nothing but the XML file Configuration or Java annotations. These Configuration Metadata contains information which defines,

* What are the Objects to be instantiated?
* What are the Objects to be configured & how to configure?
* How to assemble the Objects?

These objects will be a part of your Project’s Business Objects or DAO Objects. So, IoC Container instantiates, configures, assembles the objects at the time of starting up the application itself as well as manage the object life cycle from creation till destruction.

**Types of IoC Containers**

Spring Framework provides two types of IoC Containers.

* Bean Factory
* Application Context

**Bean Factory**

The Bean Factory, the root interface of IoC container provides basic support for achieving loose coupling between the classes using Dependency Injection.

It is defined under ***org.springframework.beans.factory.BeanFactory*** Interface.

As the name suggest, Bean Factory is a place where the beans are manufactured & injected into the classes.

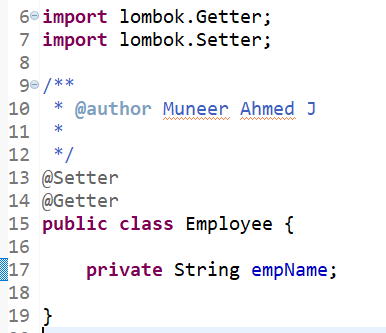
You may be wondering, what are Beans? **Beans** are nothing, but the Java Objects that are initialized by the Spring IoC Containers. So, any POJO classes or DAO classes acts as a Spring Bean if its configured to be instantiated by IoC Containers from the meta information available in the XML Configuration file or as a Java based annotation in the project.

**Scopes of Beans**

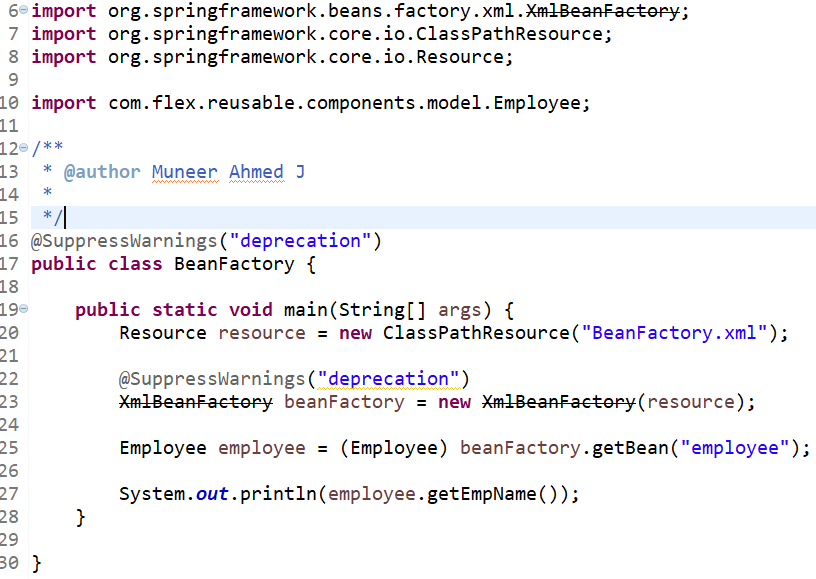
* **Singleton** – This is the default scope. As the name suggests, only one instance of the bean will be created for each container.
* **Prototype** – For each request, a new bean will be created.
* **Request** – It will be used in Web Application. For each HTTP request from client, a new bean will be created.
* **Session** – For each HTTP Session, a new bean will be created.

**How to create a Bean Factory**

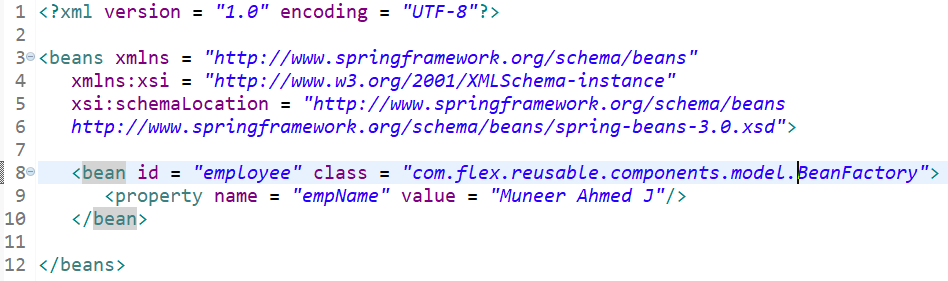
Let us see a real time example on creating Bean Factory using XML based configurations.



Employee POJO class



BeanFactory class for reading the XML Configuration using XmlBeanFactory.



BeanFactory.xml file for configuring the meta data for creating the bean objects by Spring Bean Factory.

**Output:**

Muneer Ahmed J

**Methods of Bean Factory**

There are six primary methods for Bean Factory. They are as follows.

* **boolean containsBean (String)** – This method compare the name of the bean instance available with the name passed as an argument. Returns true if matched else returns false.
* **Object getBean (String)** – This method returns the available bean instance with the name passed as an argument. If the bean instance is not found, it throws NoSuchBeanDefinition Exception.
* **Object getBean (String, Class)** - This method returns the available bean instance with the name passed as an argument & it will be casted to the Class passed as the second argument. If the bean instance is not found, it throws NoSuchBeanDefinition Exception & if the bean cannot be casted to the provided class, it throws BeanNotOfRequiredType Exception.
* **Class getType (String name)** – This method will return the Class of the bean passed as an argument.
* **boolean isSingelton (String) –** This method will check if the bean name passed as argument is in Singleton scope or not. Return true if Singleton else returns false.
* **String[] getAliases (String)** – This method returns a single or array of aliases for the passed bean name, if available.

**Application Context**

Application Context is same as Bean Factory as it is built upon BeanFactory Interface only, but it provides few more extra functionalities such as

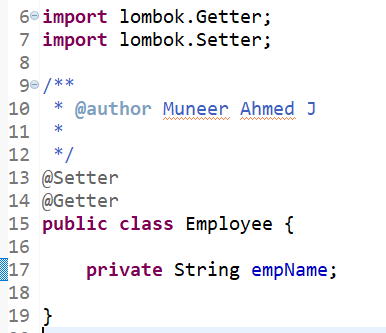
* Simple Integration with Spring Framework’s AOP,
* message resource handling in Internationalization (i18N),
* event propagation &
* most importantly WebApplicationContext for Web Applications.

It is defined under ***org.springframework.context.ApplicationContext*** Interface.

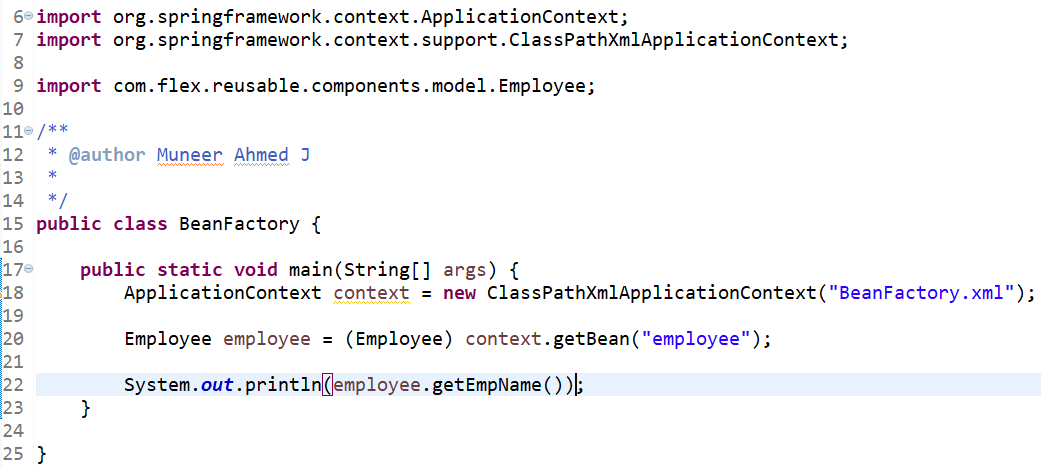
As you can see, Application Context is built over Bean Factory and it also provides few more additional features, it is mostly recommended over Bean Factory.

**How to create Application Context**

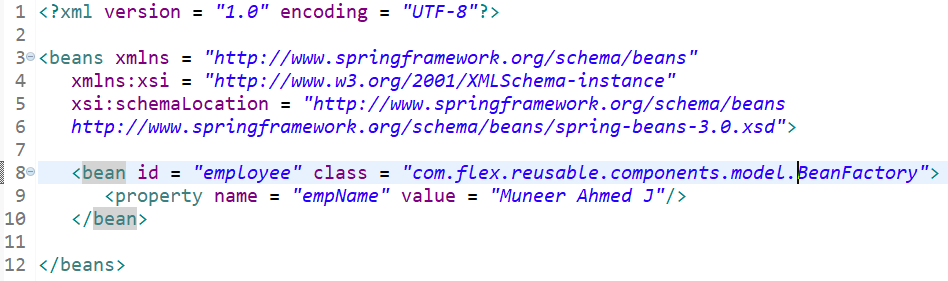
Let us see a real time example on creating Application Context using XML based configurations.



Employee POJO class



BeanFactory class for reading the XML Configuration using Application Context.



BeanFactory.xml file for configuring the meta data for creating the bean objects by Spring Bean Factory.

**Output:**

Muneer Ahmed J

**Types of Application Context**

There are three different types of Application Context. They are as follows.

* FileSystemXmlApplicationContext
* ClassPathXmlApplicationContext
* WebXmlApplicationContext

**Advantages of IoC Containers**

Advantages of IoC Containers are as follows.

* Spring IoC Containers helps in instantiating, configuring & assembling of the Objects provided in the XML Configuration files or Java Annotations.
* Spring IoC Container helps in managing the Spring Bean life Cycle i.e., right from instantiation till destruction.
* Spring IoC Containers provides the support for Inversion of Control (IoC).
* IoC Containers provides a factory of beans for dynamically binding them at the time of starting/booting up the application.
* IoC Container helps in achieving loose coupling.
* IoC Container provides easy support for creating Web Application using WebApplication Context.
* IoC Container minimizes the amount of code by avoiding boiler plate coding.
* IoC Containers provides support for Eager instantiation & lazy loading of Services.
* IoC Containers makes the Unit Testing easy by allowing the developers to inject the Mock Objects.

**Conclusion**

From this article, we have done a deep dive on the architecture, importance and the role of IoC Containers. We have also gone through the different type of IoC Containers and its advantages with real time example on XML based configurations.