**What Is a Spring Bean?**

This is a very simple question that is often overcomplicated. **Usually, Spring beans are Java objects that are managed by the Spring container.**

Here is a simple Spring bean example:

3

public class HelloWorld {

4

private String message;

5

​

6

public void setMessage(String message){

7

this.message = message;

8

}

9

public void getMessage(){

10

System.out.println("My Message : " + message);

11

}

12

}

In the XML-based configuration, **beans.xml** supplies the metadata for the Spring container to manage the bean.

**What Is the Spring Container?**

The Spring container is responsible for instantiating, configuring, and assembling the Spring beans. Here is an example of how we configure our HelloWorld POJO for the IoC container:

1

&lt;?xml version = "1.0" encoding = "UTF-8"?&gt;

2

​

3

&lt;beans xmlns = "http://www.springframework.org/schema/beans"

4

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

5

xsi:schemaLocation = "http://www.springframework.org/schema/beans

6

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"&gt;

7

​

8

&lt;bean id = "helloWorld" class = "com.zoltanraffai.HelloWorld"&gt;

9

&lt;property name = "message" value = "Hello World!"/&gt;

10

&lt;/bean&gt;

11

​

12

&lt;/beans&gt;

Now, it managed by the Spring container. The only question is: how we can access it?

**The Difference Between BeanFactory and ApplicationContext**

**The BeanFactory Interface**

This is the root *interface* for accessing the Spring container. To access the Spring container, we will be using Spring's dependency injection functionality using this BeanFactory interface and its sub-interfaces.

Features:

* Bean instantiation/wiring

Usually, the implementations use lazy loading, which means that beans are only instantiating when we directly calling them through the getBean() method.

The most used API that implements the **BeanFactory** is the **XmlBeanFactory**.

Here is an example of how to get a bean through the BeanFactory:

2

​

3

import org.springframework.core.io.ClassPathResource;

4

import org.springframework.beans.factory.InitializingBean;

5

import org.springframework.beans.factory.xml.XmlBeanFactory;

6

​

7

public class HelloWorldApp{

8

public static void main(String[] args) {

9

XmlBeanFactory factory = new XmlBeanFactory (new ClassPathResource("beans.xml"));

10

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

11

obj.getMessage();

12

}

13

}

Refer : **https://docs.spring.io/spring-framework/docs/3.0.0.M3/reference/html/ch04s08.html**

**The ApplicationContext Interface**

The **ApplicationContext**is the central interface within a Spring application that is used for providing configuration information to the application.

It implements the BeanFactory interface. Hence, the ApplicationContext includes all functionality of the BeanFactory and much more! Its main function is to support the creation of big business applications.

Features:

* Bean instantiation/wiring
* Automatic BeanPostProcessor registration
* Automatic BeanFactoryPostProcessor registration
* Convenient MessageSource access (for i18n)
* ApplicationEvent publication

It uses eager loading, so every bean instantiate after the ApplicationContext is started up.

Here is an example of the **ApplicationContext** usage:

2

​

3

import org.springframework.core.io.ClassPathResource;

4

import org.springframework.beans.factory.InitializingBean;

5

import org.springframework.beans.factory.xml.XmlBeanFactory;

6

​

7

public class HelloWorldApp{

8

public static void main(String[] args) {

9

ApplicationContext context=new ClassPathXmlApplicationContext("beans.xml");

10

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

11

obj.getMessage();

12

}

13

}

**Conclusion**

The **ApplicationContext**includes all the functionality of the **BeanFactory.** It is generally recommended to use the former. There are some limited situations, such as in mobile applications, where memory consumption might be critical. In those scenarios, it would be justifiable to use the more lightweight **BeanFactory**. However, in most enterprise applications, the **ApplicationContext** is what you will want to use.