SPRING - BEAN DEFINITION

The objects that form the backbone of your application and that are managed by the Spring IoC container are called **beans**. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container. For example, in the form of XML <bean/> definitions which you have already seen in the previous chapters.

Bean definition contains the information called **configuration metadata**, which is needed for the container to know the following –

- How to create a bean
- Bean's lifecycle details
- Bean's dependencies

All the above configuration metadata translates into a set of the following properties that make up each bean definition.

Sr.No.	Properties & Description
1	**class** This attribute is mandatory and specifies the bean class to be used to create the bean.
2	**name** This attribute specifies the bean identifier uniquely. In XMLbased configuration metadata, you use the id and/or name attributes to specify the bean identifier*s*.
3	**scope** This attribute specifies the scope of the objects created from a particular bean definition and it will be discussed in bean scopes chapter.
4	**constructor-arg** This is used to inject the dependencies and will be discussed in subsequent chapters.

5	**properties** This is used to inject the dependencies and will be discussed in subsequent chapters.
6	**autowiring mode** This is used to inject the dependencies and will be discussed in subsequent chapters.
7	**lazy-initialization mode** A lazy-initialized bean tells the IoC container to create a bean instance when it is first requested, rather than at the startup.
8	**initialization method** A callback to be called just after all necessary properties on the bean have been set by the container. It will be discussed in bean life cycle chapter.
9	**destruction method** A callback to be used when the container containing the bean is destroyed. It will be discussed in bean life cycle chapter.

Spring Configuration Metadata

Spring IoC container is totally decoupled from the format in which this configuration metadata is actually written. Following are the three important methods to provide configuration metadata to the Spring Container –

- XML based configuration file.
- Annotation-based configuration
- Java-based configuration

You already have seen how XML-based configuration metadata is provided to the container, but let us see another sample of XML-based configuration file with different bean definitions including lazy initialization, initialization method, and destruction method –

```
<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"
```

```
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <!-- A simple bean definition -->
   <bean id = "..." class = "...">
      <!-- collaborators and configuration for this bean go here -->
   </bean>

   <!-- A bean definition with lazy init set on -->
   <bean id = "..." class = "..." lazy-init = "true">
      <!-- collaborators and configuration for this bean go here -->
   </bean>

   <!-- A bean definition with initialization method -->
   <bean id = "..." class = "..." init-method = "...">
      <!-- collaborators and configuration for this bean go here -->
   </bean>

   <!-- A bean definition with destruction method -->
   <bean id = "..." class = "..." destroy-method = "...">
      <!-- collaborators and configuration for this bean go here -->
   </bean>

   <!-- more bean definitions go here -->

</beans>
```

You can check Spring Hello World Example to understand how to define, configure and create Spring Beans.

We will discuss about Annotation Based Configuration in a separate chapter. It is intentionally discussed in a separate chapter as we want you to grasp a few other important Spring concepts, before you start programming with Spring Dependency Injection with Annotations.