**Spring IOC Container**

Spring IoC Container is a core part of the Spring framework which is used to manage the application bean. It injects dependencies when a bean is created and **manages the bean life cycle** during execution.

The fundamental tasks of Spring IoC are:

* Instantiating
* Configuring, and
* Assembling Bean

The IOC container gets configuration related information from the Spring configuration file. That can be either **XML** or **Java** files.

The container uses [**dependency injection (DI)**](https://www.studytonight.com/spring-framework/spring-constructorbased-dependency-injection) to manage the components that make up an application.

Spring provides two types of IOC containers:

* BeanFactory.
* Application Context.

**Types of ApplicationContext**

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.

**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.

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

**Dependency Injection**

* Dependency Injection is a fundamental aspect of the Spring framework, through which the Spring container “injects” objects into other objects or “dependencies”. Simply put, this allows for loose coupling of components and moves the responsibility of managing components onto the container.
* Dependency injection is a design pattern in which an object or function receives other objects or functions that it depends on. A form of inversion of control, dependency injection aims to separate the concerns of constructing objects and using them, leading to loosely coupled programs.
* Spring IoC is achieved through Dependency Injection. Dependency Injection is the method of providing the dependencies and Inversion of Control is the end result of Dependency Injection. IoC is a design principle where the control flow of the program is inverted.

**Spring supports three types of dependency injection:**

### 1. Constructor-Based Injection:

In a constructor-based injection, Spring will use the matching constructor to resolve and inject the dependency.

We can either configure the beans in applicationContext.xml:

<bean id="address" **class**="com.programmergirl.domain.Address"/>

<bean id="person" **class**="com.programmergirl.domain.Person">

<constructor-arg ref="address"/>

</bean>

Or, we can enable the <component-scan/> in our applicationContext.xml:

<context:component-scan base-package="com.programmergirl.domain" />

**On enabling component scan, we can make the Spring configurations using the annotations. Our classes would then look like**:

**package** *com.programmergirl.domain*;

@Component

**public** **class** Person {

**private** Address address;

@Autowired

**public** Person(Address address) {

**this**.address = address;

}

}

**package** *com.programmergirl.domain*;

@Component

**public** **class** Address {

...

}

Spring, by default, wires the beans by their type.**If there are more than one beans of the same type, we can use @Qualifier annotation to reference a bean by its name:**

@Component

**public** **class** Person {

**private** Address address;

@Autowired

@Qualifier("address1")

**public** **void** setAddress(Address address) {

**this**.address = address;

}

}

Assuming we have two Address beans – address1 and address2, our address1 bean will be injected into Person class while dependency resolution.

### 2. Setter Injection:

Setter-based dependency injection is achieved through the setter method on the bean after instantiating it using a no-arg constructor or no-argument static factory.

We can configure it using XML as:

<bean id="address" **class**="com.programmergirl.domain.Address"/>

<bean id="person" **class**="com.programmergirl.domain.Person">

<property name="address" ref="address"/>

</bean>

**On the other hand, when using annotations, we’ll have:**

@Component

**public** **class** Person {

**private** Address address;

...

@Autowired

**public** **void** setAddress(Address address) {

**this**.address = address;

}

}

### 3. Property-Based Injection:

We can also inject dependencies using fields or properties of a class. To do so, we can simply use the @Autowired annotation over the field:

@Component

**public** **class** Person {

@Autowired

**private** Address address;

...

}

considering we’re using annotation based configurations.

### Noteworthy Points:

As per Spring documentation:

* We should **use constructor injection for mandatory dependencies**
* Setter-based injections should be used for dependencies that are optional in nature
* Spring uses reflection for injecting the field-injected dependencies. So, field-based injection is a costlier approach and we should avoid using it