**Spring Dependency Injection – Introduction**

The most important module in the Spring Framework is Core Container & **Dependency Injection (DI)** acts as the heart of Spring’s Core Container.

In a conventional way, Developers will have the control over the code in creating the objects & injecting them at run time. Here, the Spring framework takes the control of doing the above-mentioned activities at run time, that’s why the term is coined as **‘*Inversion of Control’* (IoC)** i.e., the control is inversed!

**Dependency Injection (DI)** is a design pattern which implements the Inversion of Control for resolving the dependencies at run-time i.e., injecting the dependency object to the dependent object to use it at run time.

**How to Perform Dependency Injection in Spring**

Before we investigate the ways of performing Dependency Injection, it is important to know about the different types of Dependency Injection Configurations.

There are two different ways of Dependency Injection Configurations.

* XML based Configuration
* Java Annotation based Configurations

In this article, we will focus on Java Annotation based Configuration, which is the most advanced, easy to implement and used widely across the software industry.

Also, it is important to know about the below definition before deep diving into implementing Dependency Injection.

* Beans
* Autowiring

In Spring Terminology, Objects which acts as the backbone of your application and are managed i.e., instantiation, configuration & assembled by the IoC Containers are referred as **Beans**.

Spring Framework helps us in providing a way to detect the relationship between the beans either by reading the XML Configuration file or by scanning the Java annotations at the time of booting up the application. This task is taken up by the IoC Container – Bean Factory which will create the objects & wire the dependencies. Since Spring Framework does this process automatically, it is referred as **Autowiring** i.e., Automatic Wiring!

With Java Annotation Configurations, beans are wired by using **@Autowired** annotation.

As per Java Annotation Configuration, Dependency Injection can be performed by three different ways. They are as follows.

* Constructor based Dependency Injection
* Setter based Dependency Injection
* Field or Property based Dependency Injection

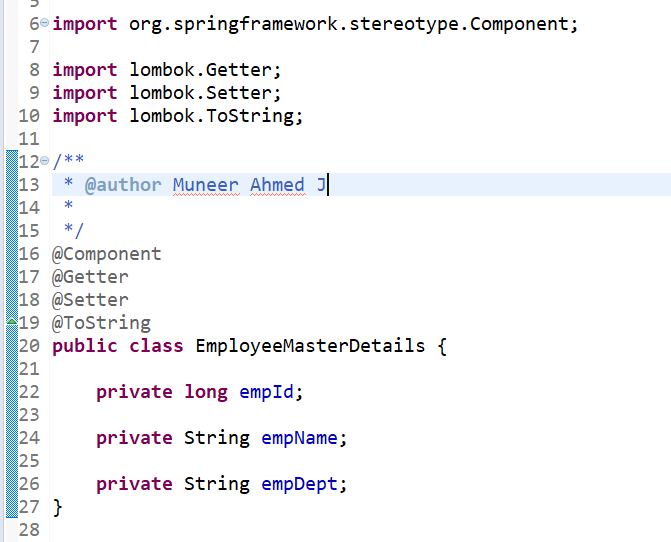
Let us see one by one in details with real time example & code snippets.

**Constructor Based Dependency Injection**

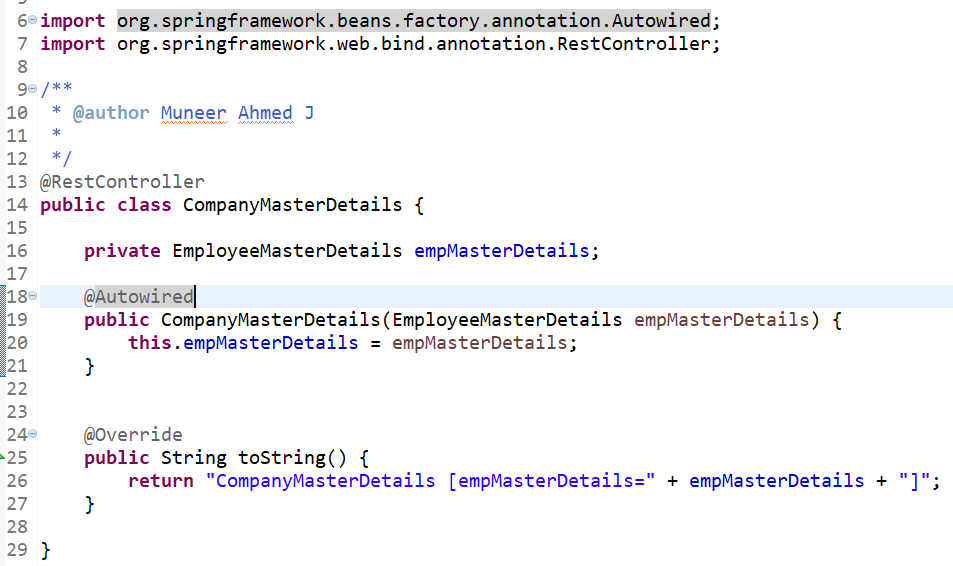
When the annotation **@Autowired** is used on top of the class constructor, it is referred to as Constructor based Dependency Injection.

Let us witness the usage of Constructor based Dependency Injection with a real time example.

**POJO Class – EmployeeMasterDetails class**



**Controller – CompanyMasterDetails class**



**EmployeeMasterDetails** class has been created as a Component using **@Component** (line 16). Spring will automatically recognize this during initialization/booting up of the application & will create a bean object using Bean Factory for EmployeeMasterDetails class.

After this, when Spring detects that EmployeeMasterDetails bean object has been Autowired (using @Autowired) into CompanyMasterDetails class constructor (line 18), the bean object created for EmployeeMasterDetails will be injected into CompanyMasterDetails class via constructor.

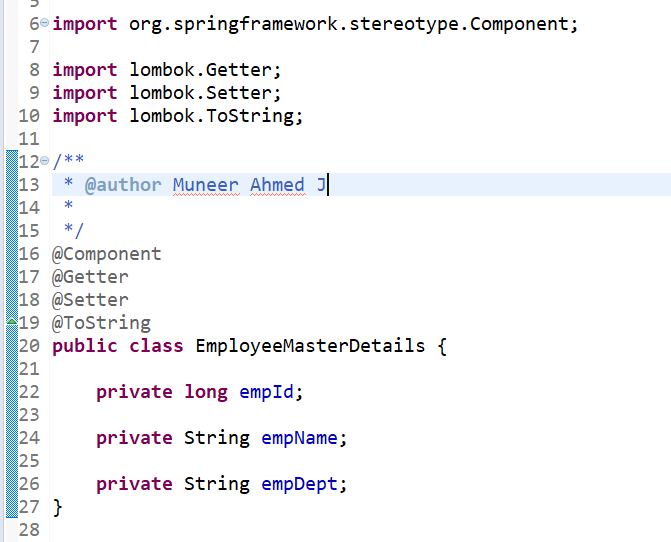
In this way, Spring Framework will perform the Dependency Injection automatically. So, if you notice here, the developer has not created any object by using “new” keyword in CompanyMasterDetails class, thereby removing the tight coupling between the CompanyMasterDetails class & EmployeeMasterDetails object.

**Setter Based Dependency Injection**

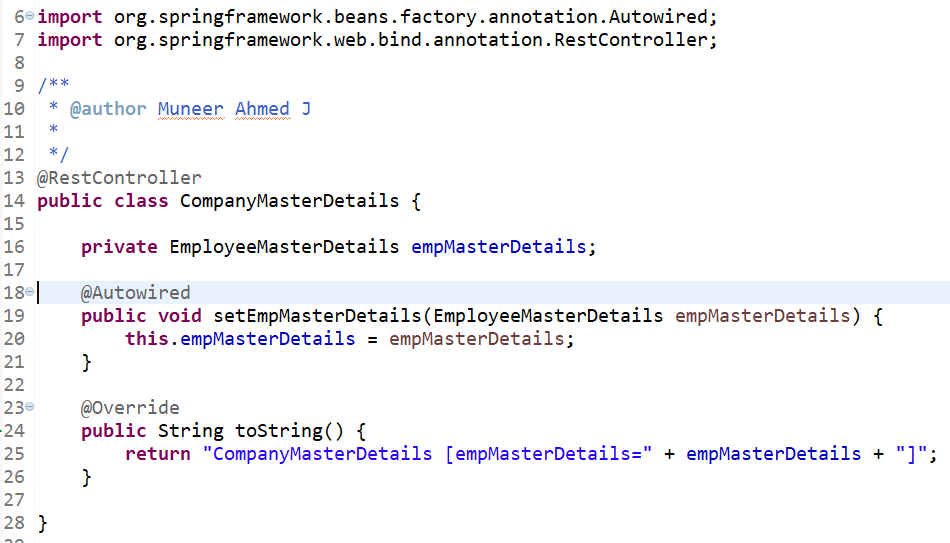
When the annotation **@Autowired** is used on top of the class’s setter method, it is referred to as Setter based Dependency Injection.

Let us witness the usage of Setter based Dependency Injection with a real time example.

**POJO Class – EmployeeMasterDetails class**



**Controller – CompanyMasterDetails class**



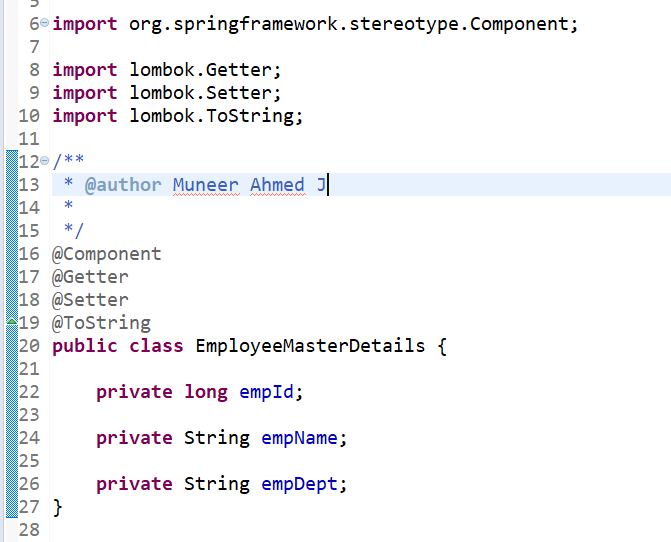
Similar to Constructor based Dependency Injection, here the Autowiring takes places by means of setter method at line no.18. The bean object created for EmployeeMasterDetails class will be Autowired & injected via setter method in CompanyMasterDetails class.

**Field or Property Based Dependency Injection**

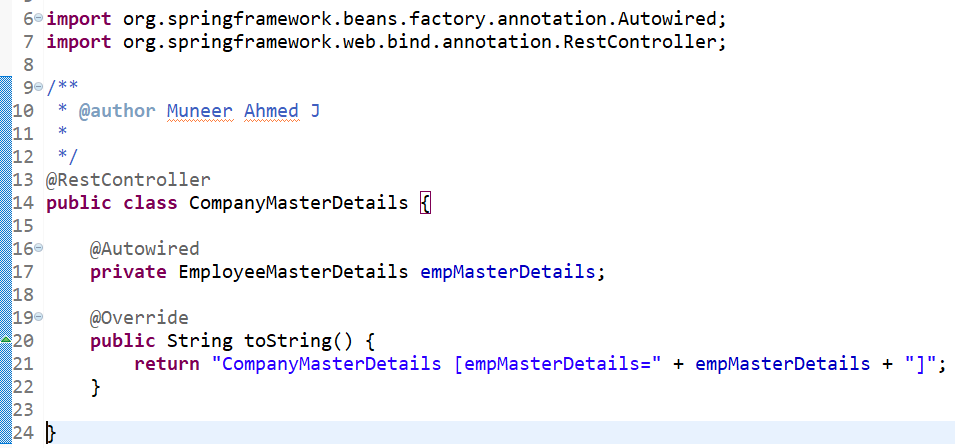
When the annotation **@Autowired** is used on top of the field or property in the class, it is referred to as Field based Dependency Injection.

Let us witness the usage of Field based Dependency Injection with a real time example.

**POJO Class – EmployeeMasterDetails class**



**Controller – CompanyMasterDetails class**



Similar to Constructor based Dependency Injection, here the Autowiring takes places by means of field ***‘empMasterDetails’*** at line no.16. The bean object created for EmployeeMasterDetails class will be Autowired & injected via the field, ***‘empMasterDetails’*** in CompanyMasterDetails class.

**Fall back or Error Handling cases**

Suppose if there are more than one bean which are of same type, the Spring Framework will get confused to inject the appropriate dependency. At this point, we will be running into **NoUniqueBeanDefinitionException** exception.

This should be handled by making use of **@Primary** or **@Qualifier** annotations along with **@Autowired** annotation.

* **@Primary** annotation lets you to give priority to the other beans of the same type.
* **@Qualifier** annotation lets you to differentiate between the other beans by providing a unique qualifier name.

**Advantages of Dependency Injection**

Some of the advantages of Spring’s Dependency Injection are as follows.

* Primarily Dependency Injection helps in achieving loose coupled architecture by removing the tight coupling/dependency between a class & its dependency.
* As the dependency between objects are loosely coupled, it helps the developers to test the module by injecting the dependent Mock Objects (for example, making use of Spring Mockito).
* Dependency Injection removes unnecessary dependencies between the classes.
* As the modules are independent of each other & can be injected, the scope of making the component reusable is very high.
* Maintenance of the System becomes easy.

**Disadvantages of Dependency Injection**

Some of the disadvantages of Spring’s Dependency Injection are as follows.

* Dependency Injection makes it difficult to trace the code as the developer needs to refer to more files (such as XML Configurations) to understand how the system behaves.
* As the Spring Framework takes care of the control rather than the developer, it will be difficult for the developers to understand how things work in the background & also to have customization.
* As Dependency Injection allows loose coupling, it ends up in increasing the number of Interfaces & classes.

**Conclusion**

So, in a nutshell, Dependency Injection helps the developers to achieve loose couplings between the classes. From this article, we have covered in depth of Dependency Injection, its advantages & disadvantages & different ways of performing dependency injection with real time examples & code snippets.