**SPRING**

**Created By**

**Saurav Kumar Jha([321sauravkumar@gmail.com](mailto:321sauravkumar@gmail.com))**

**Note-** You can’t sell or redistributewithout the consent of the Author(except guchhu)

* What is Spring

**🡪** Spring is a dependency injection framework to make java application loosely coupled

* Spring framework makes the easy development of javaEE application
* Dependency Injection
* It is design pattern

Class Ramu{ class Geeta{

Geeta obj; public void doWork(){

public void doWork(){ }

} }  
}

Here Ramu is dependent on Geeta because Ramu object needs Geeta object so like previously when we used to do work in core java then we used to manually create Geeta object but problem with that application will be tightly coupled so now we will give this responsibility to spring and spring will create the object of Geeta at runtime with the help of spring IOC container and inject it into obj so our application will be loosely coupled

**UI Layer**

Spring MVC **ProductController**

Security **Inject**

**Business/Services**

**Layer**

Transaction **ProductService**

Management

**Inject**

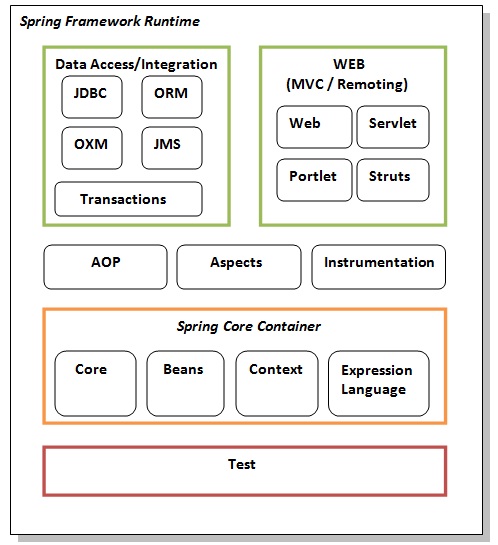
**Data Access Layer**

Spring JDBC **ProductDao**

Spring ORM

**DB**

* Spring Modules



**Core and beans**

These modules provide IOC and Dependency Injection features.

#### Context

This module supports internationalization (I18N), EJB, JMS, Basic Remoting.

**spEl**

### The Spring Expression Language (SpEL) is a powerful expression language that supports querying and manipulating an object graph at runtime

### AOP, Aspects and Instrumentation

It is used to decouple the code

### Data Access / Integration

This group comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database.

### Test

This layer provides support of testing with JUnit and TestNG.

* What is Spring IOC Container
* This is predefined program or component that we get with spring , it will create the object hold them in the memory and inject the object into another object i.e dependency injection work , it means it will manage the life cycle of object from object creation to object destruction
* We will give the Pojos or beans class information and configuration file to IOC container and it will do all the works

**IOC Container**

**Beans**

**Config**

There are two types of IOC Container

1.BeanFactory

2.ApplicationContext

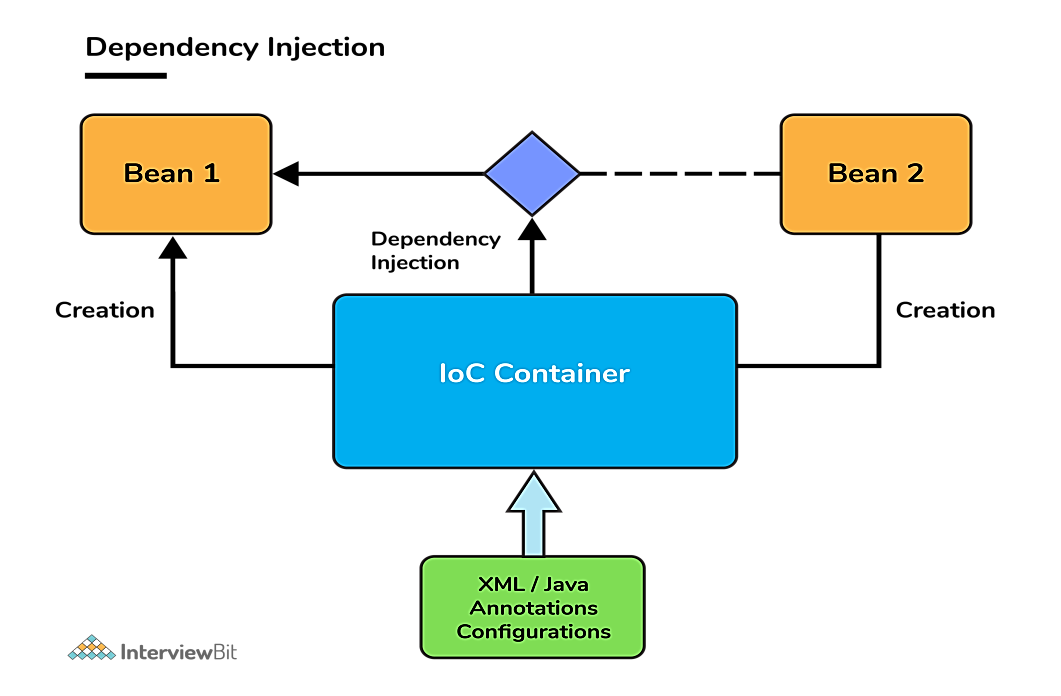
We have ApplicationContext in Spring that represents the IOC Container it is interface which extends the BeanFactory and we can’t create the object of interface so we need implementation

Bean factory only provides DI but ApplicationContext provides DI+java EE features

**ApplicationContext**

**AnnotationConfigApplicationContext** **ClasspathXMLApplicationContext**

**FileSystemXMLApplicationContext**



For using the spring first you will have to add the dependencies from the central maven repository in the pom.xml file

@Component

* this we use at the top of the class by using this we are informing to the spring that this is our bean class and create the object of this class so spring will create the object of this class and keep in the spring IOC container
* For example if we have Square class so by default container will store the bean with the name square but if we want different name like for example sq so we can mention using @Component(“sq”)

Now how spring will know where to find these class, so we will create configuration class for example JavaConfig you can name whatever you want

Now how spring will know this class is configuration class so we will use @Configuration annotation on the top of the class

🡪@Configuration is special type of @Component so it means spring will also create the object of Configuration class(ex-JavaConfig class)

Now we will inform to spring where you have to scan so we will use @ComponentScan(“package folder path”) followed by @Configuration so application bean container(ApplicationContext) will scan for the bean classes and it scans recursively it mean it will scan package inside package and so on…

@Autowired🡪 it means we are informing the spring where to inject/set the dependency , it informs to the spring at which method or at which place it has to inject and what type it has to inject

We can use @Autowired on setter,constructor and field also but the most common way is using it on the field

How to use IOC container(ApplicationContext) if we are using annotation based configuration class

Inside the main method

ApplicationContext context=new AnnotationConfogApplicationContext(pass configuration class path);

For example if you have JavaConfig as a Configuration class then,

ApplicationContext context=new AnnotationConfogApplicationContext(JavaConfig.class);

So now spring will create the bean classes and scan the bean classes and do dependency setting/injection

Now if we want the fetch the bean object by type from container then we can use context.getBean(“pass bean class that you want”)

For exam if you want Canvas class

Then ,

Canvas canvas=context.getBean(Canvas.class);

Now if you want to get the value from stored file so we can create app. properties file inside resources directory inside the main directory you can keep any file instead of app. properties

So app. properties basically stores the value in the form of key value

For example we have created app. properties file inside it

square.side=4; (square.side will be key and 4 will be value)

rectangle.length=7;

🡪properties files are a very good way or very common way in java to provide the configuration on the data

The location of app. properties file are totally different at compile time and runtime , at runtime it will go inside java directory it means inside the class path

So for using app. properties file we will mention @PropertiesSource(“app. properties”) inside the configuration class so now spring will read the properties file and load the key and values in the map so then at the later time wherever it is required it will take it from the map itself and set the value

So for example if we have Square class inside it there is a field side and want the set the value of this so we can use @Value(“${square.side}”)

* Difference between @Component and @Bean

🡪till now you are creating the class(ex-Square class) so you have full control whether you want to give the access to spring that create the object of this class or not using @Component but in the cases where source class(ex-Square class) is created by someone else and you have only compiled classes now you can’t change the source class because now you can’t mention the @Component on the top of it

It means now you have not Square.java you have only Square.class(compiled class)

So now you open the configuration class(ex- JavaConfig) and inside it we will create the method that will return Square object like

@Bean

public Square square(){

//using default constructor

Square square=new Square();

// using parameterized constructor

Square square=new Square(10);

return square;

}

But it will take value 10 only when @Value is not mentioned on the field because this object creation happening too early in the life cycle but setting the value using @Value happens too late in the life cycle so @Value will override the value

So now using @Bean we are informing that bean object for the Square will be returned by this method so object of Square will be taken from here

Previously we were fetching the bean by type now we can fetch by name also

* When the bean object is registered in the container they are registered by the name for example when the Canvas object is registered in the container it is put by name like canvas(c small) that means we can fetch by name also

canvas square

rectangle

For example 🡪 Canvas class

Canvas canvas=(Canvas)context.getBean(“canvas”);

or

Canvas canvas=context.getBean(“canvas”,Canvas.class);

Here we are typecasting to the Canvas because by default it will give Object type

* Scope of Bean
* By default in spring DI the scope of bean is singleton it means for example we have Square class so spring will create only object of Square class ,but we can create multiple object of Square class by ourself

It means for example Canvas class

Canvas canvas1=context.getBean(Canvas.class);

Canvas canvas2=context.getBean(“canvas”,Canvas.class);

So now if we print

System.out. println(canvas1==canvas2)

It will return true because spring will keep only one object of Canvas, so canvas1 and canvas2 will be referencing the same object

canvas1

canvas2

now discussing about prototype scope for example if we want to use for Canvas object then we will use @scope(“prototype”) on the top of the class (for by default @Scope(“singleton”))

most common used scope is singleton

🡪prototype is completely opposite of singleton that is any number of time objects are needed spring bean container will create the object whenever it is required

So,

When we are writing,

Canvas canvas1=context.getBean(Canvas.class);

Canvas canvas2=context.getBean(“canvas”,Canvas.class);

So first time for canvas1 we are writing getBean(Canvas.class) it means we are asking for object so spring the create the object of Canvas similarly for the canvas2 when we are writing getBean(“canvas”,Canvas.class) so spring will again create the object of Canvas and give us new object so now if we are printing

System.out. println(canvas1==canvas2)

It will return false

Similary if we have used @Autowired it means we require object so again spring will create the new object and will us that object

* What is @Qualifier

🡪The @Qualifier annotation in Spring is used to differentiate a bean among the same type of bean objects. If we have more than one bean of the same type and want to wire only one of them then use the @Qualifier annotation along with @Autowired to specify which exact bean will be wired.

Suppose we have two bean object rectangle and square with same type IShape and we are @Autowired like

@Autowired

private IShape shape;

now will spring will confuse which bean should I inject here because rectangle and square has same type

so now we will mention that don’t inject by type ,inject by name for that we will use @Qualifier , for example if we want to inject rectangle object so

@Qualifier(“rectangle”)

@Autowired

private IShape shape;

* Default injection is “by type”

🡪We can use @Qualifier only with @Autowired

As we already know that spring IOC container manages the lifecycle of beans so it provides life cycle hooks what does life cycle hooks mean that when something happens you can have your things executed so for example

-> when object is completely initialized you want something to be executed then you can make use of lifecycle

-> When object is removed from the container you want something to be executed

@PostConstruct

It is used on the top of the method when we want that method to be executed after the bean is initialized of that class

@PreDestroy

It is used on the top of the method when we want that method to be executed before the object is destroyed

But in this case the statement written inside method will not be printed because ApplicationContext is not closed safely because main method finsishes so the ApplicationContext is not getting closes in a safe manner this is the statement is not printed , if the container is closed in the safe manner so obviously before the container is closed it will remove the beans from it and after the beans are removed container will be closed but we are not closing the container it is getting destroyed abruptly that’s why we are not getting logs

So what we will do we will close the container

So we will mention context.close()

But the close method is not in the ApplicationContext so there is a abstract class(AbstractApplicationContext) or ConfigurableApplicationContext interface (that is extending ApplicationContext) that we will use in place of ApplicationContext