Dependency Injection and Spring Annotations

The Spring IoC Container

IoC container is represented by the interface ApplicationContext. The Spring container is responsible for instantiating, configuring and assembling objects known as beans, as well as managing their lifecycle.

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage, develop and test the application. Dependency Injection makes our programming code loosely coupled

Tight Coupling

Example: Assume that you have a task to finish for time being consider it is conducting online class. For this task you depending on android phone So you write a code to perform a online class using android phone. This works well but in future if you upgrade to IPhone your code doesn’t work (need to modify lot of stuffs) because its not android phone this concept is called as tight coupling.

Loose Coupling

Now you write a code for conducting online class using a smartphone, you can give either android phone or iPhone to finish your task this concept is called as loose coupling

Code for Tight Coupling:

```
public class IPhone

	public String display() {
		return "IPhone is best";
	}

}

public class Main {

	public static void main(String[] args) {

		IPhone iphone = new IPhone();
		System.out.println(iphone.display());
	}
}
```

Code for Loose Coupling:

```
public interface Smartphone {

	String display();
}
```

```
public class Android implements Smartphone {

	@Override
	public String display() {
		return "Android is best";
	}
}

public class IPhone implements Smartphone {

	@Override
	public String display() {
		return "IPhone is best";
	}
}
@Getter @Setter
@ToString
@NoArgsConstructor
public class OnlineClass {

	private Smartphone smartphone;

	public OnlineClass(Smartphone smartphone) {
		super();
		this.smartphone = smartphone;
	}
}

public class Main {

	public static void main(String[] args) {

		OnlineClass oc = new OnlineClass(new IPhone());
		System.out.println(oc.getSmartphone().display());
	}
}
```

Spring Annotations

Annotation makes your class managed by the spring.
Controller, RestController [Handling request from a browser],
Component [Generic annotation],
Service [Write Business Logic],
Repository [Persist the data to DB],
Configuration [Provide configuration to your application]
Autowired [This annotation asks spring to give those dependency to whichever the class it is requesting for]

SpringBootApplication [Makes your simple application to Spring Boot Application and lots of configuration like below]

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
```

@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan

ComponentScan [Scans all the packages & sub packages where your SpringBootApplication annotation is declared]
Spring application by default scans only those classes which are resides under the package where your SpringBootApplication annotation is declared.

For Ex:

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

	public static void main(String[] args) {
		SpringApplication.run(Application.class, args);
	}

}
```

Here only annotated classes which resides under the package and sub package com.example.demo are scanned by the spring application.
Other annotated classes are skipped by spring container.
Suppose we have a class Example which is resides under the package com.example.try is skipped by spring application.
If you want to include those external packages then you can mention those package name in component scan something like this
@ComponentScan(“com.example”)

Now annotated Example class will be scanned and managed by the Spring IOC Container.