**Spring**

**1.What is Loose Coupling?**

**Loose coupling**- When an object gets the object to be used from the outside, then it is a loose coupling situation. As the main object is merely using the object, this object can be changed from the outside world easily marked it as loosely coupled objects.

**2.What is a Dependency?**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.

**3.What is IOC (Inversion of Control)?**

**Inversion of Control** (IoC) is the mechanism to achieve loose-coupling between Objects dependencies. To achieve loose coupling and dynamic binding of the objects at runtime, the objects define their dependencies that are being injected by other assembler objects. Spring IoC container is the program that injects dependencies into an object and makes it ready for our use.

**4.What is Dependency Injection?**

Dependency Injection design pattern allows us to remove the hard-coded dependencies and make our application loosely coupled, extendable and maintainable. We can implement dependency injection pattern to move the dependency resolution from compile-time to runtime.

Some of the benefits of using Dependency Injection are Separation of Concerns, Boilerplate Code reduction, Configurable components, and easy unit testing**.**

**5.Can you give few examples of Dependency Injection?**

1.Suppose you gives 1ltr water to me frequently, you use 10 cups of 100ml for that.

So, every time you come with 10 cups.

Now, suppose you got a jug of 1ltr... What would you do.?

U will use that every time, because it has functionality of doing your work easily...simple...

In technical way, the 1ltr jug is your Dependency Injection, it will make your work much easier

2.House and Rooms

3.Phone and SIM

4.Laptop and Processor

**6.** **What is Auto Wiring?**

@Autowired annotation – We can use Spring @Autowired annotation for spring bean autowiring. @Autowired annotation can be applied on variables and methods for autowiring byType. We can also use @Autowired annotation on constructor for constructor based spring autowiring.

For @Autowired annotation to work, we also need to enable annotation based configuration in spring bean configuration file. This can be done by context:annotation-config element or by defining a bean of type org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.

**7.What are the important roles of an IOC Container?**

**Spring IoC** is the mechanism to achieve loose-coupling between Objects dependencies. To achieve loose coupling and dynamic binding of the objects at runtime, objects dependencies are injected by other assembler objects. Spring IoC container is the program that injects dependencies into an object and make it ready for our use. We have already looked how we can use Spring Dependency Injection to implement IoC in our applications.

**8. What are Bean Factory and Application Context?**

**BeanFactory** is a simplest container providing the basic support for Dependency Injection. **Application Context Application context** is a central interface with in the **spring application** that provide the configuration information to the **application**. It implements the **Bean Factory** Interface.

**9. Can you compare Bean Factory with Application Context?**

a. **One difference between bean factory** and **application context** is that former only instantiate **bean** when **you** call getBean() method while **ApplicationContext** instantiates Singleton **bean** when the container is started, It doesn't wait for getBean to be called.

**10. How do you create an application context with Spring?**

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 like bean configuration XML file in CLASSPATH.
* **WebXmlApplicationContext** − This container loads the XML file with definitions of all beans from within a web application.

**11.** **How does Spring know where to search for Components or Beans?**

@ComponentScan

If you understand Component Scan, you understand Spring.

Spring is a dependency injection framework. It is all about beans and wiring in dependencies.

The first step of defining Spring Beans is by adding the right annotation — @Component or @Service or @Repository.

However, Spring does not know about the bean unless it knows where to search for it.

**This part of “telling Spring where to search” is called a Component Scan.**

You define the packages that have to be scanned.

Once you define a Component Scan for a package, Spring would search the package and all its sub packages for components/beans.

**12.** **What is a Component Scan?**

Using **component scan** is one method of asking Spring to detect Spring-managed **components**. Spring needs the information to locate and register all the Spring **components** with the application context when the application starts. Spring can auto **scan**, detect, and instantiate **components** from pre-defined project packages.

**13. How do you define a component scan in XML and Java Configurations?**

@Configuration

@ComponentScan("com.company") *// search the com.company package for @Component classes*

@ImportXml("classpath:com/company/data-access-config.xml") *// XML with DataSource bean*

**public** **class** Config {

}

**14. How is it done with Spring Boot?**

Executive Summary

* If your other packages hierarchies are below your main app with the @SpringBootApplication annotation, you’re covered by implicit components scan.
* If there are beans/components in other packages which are not sub packages of the main package, you should manually add them as @ComponentScan

####### Detailed Example

Consider the class below:

package com.in28minutes.springboot.basics.springbootin10steps;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

public class SpringbootIn10StepsApplication {

public static void main(String[] args) {

ApplicationContext applicationContext =

SpringApplication.run(SpringbootIn10StepsApplication.class, args);

for (String name : applicationContext.getBeanDefinitionNames()) {

System.out.println(name);

}

}

}

@SpringBootApplication is defined on SpringbootIn10StepsApplication class which is package com.in28minutes.springboot.basics.springbootin10steps.

@SpringBootApplication defines an automatic component scan on package com.in28minutes.springboot.basics.springbootin10steps.

You are fine if all your components are defined in the above package or a sub-package of it.

However, let’s say one of the components is defined in a package com.in28minutes.springboot.somethingelse

In this case, you would need add the new package into component scan.

Two Options

* Define @ComponentScan(“com.in28minutes.springboot”)
  + This would scan the entire parent tree of com.in28minutes.springboot.
* Or Define two specific Component Scans by using an array.
  + @ComponentScan({“com.in28minutes.springboot.basics.springbootin10steps”,”com.in28minutes.springboot.somethingelse”})

Option 1

@ComponentScan(“com.in28minutes.springboot”)

@SpringBootApplication

public class SpringbootIn10StepsApplication {

Option 2

@ComponentScan({"com.in28minutes.springboot.basics.springbootin10steps","com.in28minutes.springboot.somethingelse"})

@SpringBootApplication

public class SpringbootIn10StepsApplication {

**15.What does @Component signify?**

@**Component** annotation **means** that only a single instance of the annotated class gets created. Also in most cases this instance **is** automatically created on application startup, perhaps when some other common Spring annotations **are** used.

**16. What does @Autowired signify?**

The @**Autowired** annotation provides more fine-grained control over where and how **autowiring** should be accomplished. The @**Autowired** annotation **can** be used to **autowire** bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments .

**17. What’s the difference Between @Controller, @Component, @Repository, and @Service Annotations in Spring?**

@**Component** serves as a generic stereotype **for** any **Spring**-managed **component**; whereas, @**Repository**, @**Service**, and @**Controller** serve as specializations **of** @**Component for** more specific use cases (e.g., **in the** persistence, **service**, and presentation layers, respectively).

**18.** **What is the default scope of a bean?**

**Singleton**. **Singleton** is the default scope for a Bean, the one that will be used if nothing else is indicated. This scope implies that Spring container will create an only shared instance of the class designated by this bean, so each time the Bean is required the same object will be injected.

**19. Are Spring beans thread safe?**

**Spring** doesn't guarantee **thread**-**safety**. ... In **Spring**, singleton **beans** will not have any state (stateless). Singleton **bean** scope ensures that single instance per BeanFactory. So in multi **threading** environment it will not assure the single instance even with singleton **bean** scope.

**20. What are the other scopes available?**

The **other scopes**. The **other scopes**, namely request , session , and global session are for use only in web-based applications (and can be used irrespective of which particular web application framework you are using, if indeed any).