Core Spring Interview Questions

Description

1) What is Loose Coupling?

Let's say we have a class SortingService, inside the class we have created an instance of BubbleSort Algorithm used to sort the list of employees

```
public class SortingSevice {
         BubbleSort bubbleSort = new BubbleSort();
            bubbleSort(employeeList);
}
```

In future If we want to change the implementation from BubbleSort to QuickSort then we will have to change the code in SortingService class. SortingService is tightly coupled with BubbleSort. Here we only see a small change. But considering the nature of change, impact will be huge in case of larger projects. Hence the above implementation is called tight coupling.

Below code is loosely coupled.

```
public interface SortAlgorithm {
  public List<Employee> sort(List<Employee> emps);
}
```

```
@Component
public class BubbleSort implements SortAlgorithm { //implementation here }
```

```
@Component
public class SortingService {

  @Autowired
  SortAlgorithm sortAlgorithm;

}
```

Since BubbleSort is implementing the SortAlgorithm, that will be injected inside SortingService and the sort method will be called.

2) What is Dependency Injection?

The concept of Dependency injection in Spring is used to create loosely coupled applications by injecting the dependencies when required.
For E.g. If a class A depends on class B to perform any operation, instead of creating an instance of class B inside class A we should inject the instance of class B inside class A. There are two types of dependency injection, setter injection & constructor injection

3) What is IOC (Inversion of Control)?

It means giving the control of creating and instantiating the spring beans to the Spring IOC container and the only work the developer does is configuring the beans. IOC is a technique where you let someone else create the object for you. And someone else in case of spring is an IOC container.

4) What is Auto Wiring?

We use @Autowired annotation in spring for auto wiring purposes. It is a way of telling the container to give an instance on which it has been used.
E.g.

```
public class EmployeeController {
@Autowired
EmployeeService employeeService;
```

The above code states that EmployeeController needs an instance of EmployeeService class.

```
@Component
public class EmployeeService
```

Spring sees that EmployeeService is annotated with @Component, hence it will create an instance and then inject inside EmployeeController class.
So autowiring is wiring the dependencies wherever required and this is automatically done by spring by looking at the annotations.

5) What are the responsibilities of an IOC Container?

The responsibilities of the IOC container is to find the beans, identify the dependency and then wire them and also manage the lifecycle of the beans.

```
public interface SortAlgorithm {
  public List<Integer> sort(List<Integer> emps);
}

@Component
BubbleSort implements SortAlgorithm { //implementation here }

@Component
public class SortingService {

  @Autowired
  SortAlgorithm sortAlgorithm;

}
```


The IOC container will create 2 beans i.e. SortingService & BubbleSort since we have annotated with @Component. Before creating the bean of SortingService, IOC container sees that it has a dependency on BubbleSort (Implementing SortAlgorithm). So Spring will autowire it and then create a bean of SortingService.

6) What are BeanFactory and Applicationcontext?

Bean Factory and ApplicationContext are the two IOC containers. BeanFactory is the most basic version of the IOC container & Application context an advanced version of Bean Factory. Application context provides additional features such as WebApplicationContext that are used for web applications, i18n and various other features.

7) How do you create an application context with Spring?

There are two ways to define the context. One using the XML or we can define using the annotation @Configuration
Method 1 – XML approach

```
<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <bean id = "helloIndia" class = "com.springtutorials.HelloIndia">
```

```
      <property name = "msg" value = "Hello India!"/>
   </bean>

</beans>
```

All the beans are defined in the above xml file.

```
ApplicationContext context = new ClassPathXmlApplicationContext(new String[] {
```

Above code snippet, it looks for all the xml files present in the classpath
Method 2 – Java approach

```
@Configuration
Class MyContext{
//your code here
}

ApplicationContext context = new AnnotationConfigApplicationContext(MyContext.
```


8) What is a Component Scan?

Spring will search for all the classes having @Component so that it can create a bean. Depending on the project use case there can be a lot of packages a project can have. If Spring tries to search in all the packages then it can be a big performance impact. So we can explicitly specify or help spring on which packages to search for. In the below example we are telling spring to search for components present in com.springtutorials package.

```
@Configuration
@ComponentScan(basePackages = {"com.springtutorials"})
class MyContext {
}
```

XML way of defining component scan is as below

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:context="http://www.springframework.org/schema/context"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context-2.5.xsd">

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

</beans>
```

9) What do you mean by @Component and @Autowired?

Whichever class is annotated with @Component, a bean is created and managed by Spring. In the below example 2 beans will be created i.e. BubbleSort and SortingService. Since SortAlogrithm is present in SortingService and annotated with @Autowired, Spring will inject the dependency.


```
@Component
BubbleSort implements SortAlgorithm { //implementation here }

@Component
public class SortingService {

  @Autowired
  SortAlgorithm sortAlgorithm;

}
```

10. Whatâ€™s the difference between @Controller, @Component, @Repository, and @Service Annotations in Spring?

Since we follow a layered architecture while developing any enterprise application, we add a specific annotation to that specific layer.

@**Controller –** This annotation is most widely used in spring MVC pattern. This annotation is used on a class which belongs to a controller layer & which is the starting point that gets the request.
@**Service –** Usually the controller layer will give a call to the service layer where the business logic is written. @Service is used to annotate all the service or business layer classes.
@**Repository –** The layer that talks to the database or that is responsible to give us the expected data is annotated with @Repository. Spring will automatically add the exception translation for JDBC exceptions. Whenever a JDBC exception happens then it will be translated to a specific Spring exception.
@**Component –** When we add this annotation to a class, a bean will be created and registered. This is the most generic annotation.

11. What are different scopes of a bean?

Singleton – Only one instance per spring context will be created. When you create an application context there is only one instance of that particular bean per container.
Prototype – New bean will be created whenever it is requested. To be more precise, in an application context if 10 requests to that bean then 10 different instances will be created.
Request – This scope is applicable only in web application context. A new bean will be created on every HTTP request. This scope is more suitable if we want to create beans as and when the user performs that specific operation.
Session – This scope is applicable only in web application context. Only one bean will exist per HTTP session. If some data is required across the session then this scope is more suitable. Example if a user is logged in and we want to maintain the user data and share it till the session is alive then we can make use of session scope.

12. What is the default scope of a bean?

The default scope of a bean is singleton scope.

13. Are Spring beans thread safe?

By default spring beans are not thread safe.

14. What are the different types of dependency injections?

There are two types of dependency injection. Setter and Constructor injection.

15. What is setter injection?

Setter injection is used for Optional dependencies.

```
@Component
public class EmployeeService {

 EmployeDao employeeDao;

 @Autowired
 public void setemployeeDao(EmployeDao employeeDao) {
  this.employeeDao = employeeDao;
 }
}
```

By using the @Autowired annotation on top of setemployeeDao method, we are telling spring to wire the dependency i.e. employeeDao using setter injection.

16. What is constructor injection?

Constructor injection is used for Mandatory dependencies.

```
@Component
public class EmployeeService {

 EmployeDao employeeDao;

 @Autowired
 public EmployeeService(EmployeDao employeeDao) {
  super();
  this.employeeDao = employeeDao;
 }
}
```

By using the @Autowired annotation on top of EmployeeService constructor, we are telling spring to wire the dependency i.e. employeeDao using constructor injection.

17. How does Spring do Autowiring?

There are 2 different ways Spring does the autowiring.

Autowiring byType

```
public interface SortAlgorithm {
  public List<Employee> sort(List<Employee> emps);
}

@Component
public class BubbleSort implements SortAlgorithm { //implementation here }

@Component
public class SortingService {

  @Autowired
  SortAlgorithm sortAlgorithm;
}
```

In the above example, SortAlgorithm is the interface and BubbleSort class is implementing the SortAlgorithm interface. When Spring sees that SortingService class needs SortAlgorithm, it searches for any implementations of SortAlgorithm interface i.e. any classes that are implementing SortAlgorithm.
If SortAlgorithm would have been a class instead of interface, Spring would search for classes which are of that specific type.
When spring sees that BubbleSort class is implementing SortAlgorithm, spring will autowire

BubbleSort inside SortingService.

Autowiring byName

```
public interface SortAlgorithm {
  public List<Employee> sort(List<Employee> emps);
}

@Component
public class BubbleSort implements SortAlgorithm { //implementation here }

@Component
public class InsertionSort implements SortAlgorithm { //implementation here }

@Component
public class SortingService {

  @Autowired
  SortAlgorithm bubbleSort;
}
```

In the above example 2 classes i.e. BubbleSort and InsertionSort are implementing SortAlgorithm. And we are autowiring the SortAlgorithm interface inside the SortingService class. How will spring come to know which implementing class to autowire?


```
@Autowired
SortAlgorithm bubbleSort;
```

If you look at the above line, we have given bubbleSort as the name and it matches with BubbleSort class name and hence it injects BubbleSort class. This is called autowiring byName

18) What do you mean by NoSuchBeanDefinitionException and when do we get this exception?

We get this exception when spring does not find the bean which it has to inject.

```
public interface SortAlgorithm {
  public List<Employee> sort(List<Employee> emps);
}

public class BubbleSort implements SortAlgorithm { //implementation here }

public class InsertionSort implements SortAlgorithm { //implementation here }

@Component
public class SortingService {
```

```
  @Autowired
  SortAlgorithm sortAlgorithm;
}
```

In the above example BubbleSort and InsertionSort classes are implementing the SortAlgorithm interface, but the classes have not been annotated with @Component. Hence Spring will not be able to find the class that it has to inject in the SortingService class where we have SortAlgorithm autowired. Sometimes even after adding the @Component annotation, you will see the exception occurring. One of the possible reasons can be you are component scanning the wrong package whereas the classes you want to autowired must be present in a different package structure.

19) What do you mean by NoUniqueBeanDefinitionException & when do we get this exception?

If Spring sees that it has to inject some dependency in a particular class but if the spring finds that there are two beans, then it gets confused as to which bean to inject. In that case it throws a NoUniqueBeanDefinitionException.


```
public interface SortAlgorithm {
  public List<Employee> sort(List<Employee> emps);
}

@Component
public class BubbleSort implements SortAlgorithm { //implementation here }

@Component
public class InsertionSort implements SortAlgorithm { //implementation here }

@Component
public class SortingService {

  @Autowired
  SortAlgorithm sortAlgorithm;
}
```

In the above example Spring has to autowire SortAlgorithm inside the SortingService class. But Spring sees that two classes are implementing SortAlgorithm so it gets confused which one to autowire.

20) What is @Primary?

In the previous question we saw why we get NoUniqueBeanDefinitionException.
Method 1 – One way to solve the above exception is autowire byName. We can write below code in SortingService.
Spring will autowire BubbleSort class.

```
@Autowired
SortAlgorithm bubbleSort;
```

Method 2 – Another way is to make use of @Primary.
Since two classes are implementing the same interface, and to make Spring aware which one to use while injecting the dependency, we have to add one more annotation in one of the classes.

```
@Component
@Primary
public class BubbleSort implements SortAlgorithm { //implementation here }
```

21) What is @Qualifier?

In the previous question we saw two methods on how we can avoid NoUniqueBeanDefinitionException. @Qualifier is the third way. We have to add this above the class we want to inject and give it a name as below.


```
@Component
@Qualifier(â€œbubbleAlgoâ€•)
public class BubbleSort implements SortAlgorithm { //implementation here }
```

Also add the @Qualifier annotation along with the name which we used in the implementing class.

```
@Autowired
@Qualifier(â€œbubbleAlgoâ€•)
SortAlgorithm bubbleSort;
```

22) Different Spring versions and its features?

Spring 2.5 – From 2.5 onwards, Spring provided support for annotations. Before 2.5 we had to make extensive use of XML.
Spring 4 – From version 4 onwards, Spring provided support for Java 8 features. Also introduced @RestController annotation which is used in Spring REST.
Spring 5 – From version 5 onwards, Spring provided support for reactive programming, Functional web framework & support for Kotlin.

23) What are important Spring Projects?

Apart from providing just dependency injection features, Spring has evolved very much. Spring has various projects which we can make use of for various purposes. Below are some of the projects
Spring Data – Provides consistent data access.
Spring Batch – To develop batch applications.
Spring Security –

To secure the applications.
Spring Boot – To quickly build production ready applications.
Spring Cloud – To enable projects built in spring boot, we can use Spring cloud to cloud enable the applications.
Spring REST – To develop REST web services.

Category

1. Interview

Date Created
March 15, 2021
Author
kk-ravi144gmail-com