

## Quiz Questions

### (1) Purpose of @Component Annotation – Spring and Annotations

The screenshot shows a quiz question titled "Purpose of @Component Annotation - Spring and Annotations" with a score of 20/20. The problem statement asks for the purpose of the @Component annotation. The options are: "It marks a class as a component to be automatically detected and configured by Spring's dependency injection framework.", "It creates an instance of the annotated class as a bean in the Spring container.", "Both a and b.", and "None of the above". The correct answer is "Both a and b.". The solution description explains that @Component marks a class as a component and creates a bean instance of that class.

Classroom

Spring and Annotations  
@Component Purpose

Problem Submissions Doubts

✓ Purpose of @Component Annotation - Spring and Annotations  
Easy • Score 20/20

Problem statement [Send feedback](#)  
What is the purpose of the @Component annotation in Spring Framework?

Options: Pick one correct answer from below

☐ It marks a class as a component to be automatically detected and configured by Spring's dependency injection framework.

☐ It creates an instance of the annotated class as a bean in the Spring container.

☒ Both a and b. ✓

☐ None of the above

Solution description

- The @Component annotation marks a class as a component to be automatically detected and configured by Spring's dependency injection framework.
- It creates an instance of the annotated class as a bean in the Spring container. By default, Spring will use the class name with the first letter in lowercase as the bean name, but you can also specify a custom name using the value attribute of the @Component annotation.
- So, the @Component annotation marks a class as a component and creates a bean instance of that class.

### (2) Usage of @Component Annotation – Spring and Annotations

The screenshot shows a quiz question titled "Usage of @Component Annotation - Spring and Annotations" with a score of 20/20. The problem statement asks if the @Component annotation can be used with abstract classes or interfaces. The options are: "The abstract class or interface cannot be instantiated as a bean.", "No, the @Component annotation can only be used with concrete classes.", "Yes, and the abstract class or interface can be instantiated as a bean if a concrete subclass is also annotated with @Component.", and "None of the above". The correct answer is "Yes, and the abstract class or interface can be instantiated as a bean if a concrete subclass is also annotated with @Component.". The solution description explains that @Component can be used with abstract classes or interfaces, but a concrete subclass must be instantiated and annotated with @Component.

Classroom

Spring and Annotations  
@Component Annotation Usage

Problem Submissions Doubts

✓ Usage of @Component Annotation - Spring and Annotations  
Easy • Score 20/20

Problem statement [Send feedback](#)  
Can the @Component annotation be used with abstract classes or interfaces?

Options: Pick one correct answer from below

☐ The abstract class or interface cannot be instantiated as a bean.

☐ No, the @Component annotation can only be used with concrete classes.

☒ Yes, and the abstract class or interface can be instantiated as a bean if a concrete subclass is also annotated with @Component. ✓

☐ None of the above.

Solution description

- Yes, the @Component annotation can be used with abstract classes or interfaces. However, you cannot instantiate an abstract class or interface directly as a bean in the Spring container. Instead, you can create a concrete subclass of the abstract class or implement the interface and annotate it with the @Component annotation. Then, you can use the concrete subclass or implementation as a bean in the Spring container.
- Spring needs a concrete class to instantiate and configure a bean. So, if you have an abstract class or interface that you want to use as a component, you need to provide a concrete implementation of it.

### (3) DI using Annotation

The screenshot shows a quiz question titled "Dependency Injection - Spring and Annotations" with a score of 20/20. The problem statement asks which of the following annotations can be used to inject a Spring Bean into another Bean. The options are: "@Component", "@Autowired", "@Scope", and "@Bean". The correct answer is "@Autowired".

Classroom

Spring and Annotations  
DI using Annotation

Problem Submissions Doubts

✓ Dependency Injection - Spring and Annotations  
Easy • Score 20/20

Problem statement [Send feedback](#)  
Which of the following annotations can be used to inject a Spring Bean into another Bean?

Options: Pick one correct answer from below

☐ @Component

☒ @Autowired ✓

☐ @Scope

☐ @Bean

## (4) Autowired Exception

← Classroom

Spring and Annotations  
@Autowired Exception

?

V

Problem Submissions Doubts

✔ Autowired Exception - Spring and Annotations

Easy • Score 20/20

Send feedback

Problem statement

Which of the following describes the behaviour of Spring's @Autowired annotation when multiple beans of the same type are available?

Options: Pick one correct answer from below

☒ It will throw a NoUniqueBeanDefinitionException if multiple beans of the same type are found, and no @Qualifier is provided. ✔

☐ It will randomly select one of the available beans of the same type and inject it.

☐ It will inject all of the available beans of the same type.

☐ It will automatically create and inject a new bean of the same type.

Solution description

- When multiple beans of the same type are available in the Spring context, Spring's @Autowired annotation will try to inject a bean of that type into the annotated field, method parameter, or constructor parameter. However, if no @Qualifier annotation is provided to specify which bean to inject, Spring will throw a NoUniqueBeanDefinitionException because it doesn't know which bean to choose.
- To solve this problem, you can provide a @Qualifier annotation to specify which bean to inject or Spring's @Primary annotation to indicate the default bean to be injected when multiple beans of the same type are available.

## (5) Todo Application

Problem Submissions Doubts

Todo Application

Easy • Score 20/20

Problem statement

Below is the implementation of a Todo Application with classes TodoItem, TodoList and TodoApp-

```
public class TodoItem {
    private String description;
    private boolean completed;

    // Getters and setters

    // Other methods
}
```

```
@Component
public class TodoList {
    private List<TodoItem> items;

    public TodoList() {
        items = new ArrayList<>();
    }

    public void addItem(TodoItem item) {
        items.add(item);
    }

    public void removeItem(int index) {
        if (index >= 0 && index < items.size()) {
            items.remove(index);
        }
    }

    public void markItemAsCompleted(int index) {
        if (index >= 0 && index < items.size()) {
            TodoItem item = items.get(index);
            item.setCompleted(true);
        }
    }
}
```

Which is the correct implementation of class TodoApp if we want to inject TodoList in TodoApp

Options: Pick one correct answer from below

☐

```
@SpringBootApplication
public class TodoApp implements CommandLineRunner {

    @Component
    private TodoList todoList;

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

    @Override
    public void run(String... args) throws Exception {
        // Application logic goes here
    }
}
```

☒

```
@SpringBootApplication
public class TodoApp implements CommandLineRunner {

    @Autowired
    private TodoList todoList;

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

    @Override
    public void run(String... args) throws Exception {
        // Application logic goes here
    }
}
```

☐

```
@SpringBootApplication
public class TodoApp implements CommandLineRunner {

    @Autowired("TodoList")
    private TodoList todoList;

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

    @Override
    public void run(String... args) throws Exception {
        // Application logic goes here
    }
}
```

## (6) Purpose of @Qualifier

Classroom

Spring and Annotations  
Purpose of @Qualifier

Problem Submissions Doubts

Purpose of @Qualifier Annotation

Easy • Score 20/20

Problem statement

What is the purpose of the @Qualifier annotation in Spring?

Options: Pick one correct answer from below

☐ It specifies the name of a bean in the Spring container.

☒ It specifies the type of bean in the Spring container.

☐ It specifies the order in which beans are initialised in the Spring container.

☐ It specifies the scope of a bean in the Spring container.

Solution description

The @Qualifier annotation in Spring is used to disambiguate beans when there are multiple beans of the same type are available in the Spring container. It works in conjunction with @Autowired to specify which bean should be injected based on its unique qualifier value. Using @Qualifier, you can specify the exact bean name or qualifier value for injection, ensuring the correct bean is resolved.

## (7) @Qualifier Annotation Usage

←

Classroom

Spring and Annotations  
@Qualifier Annotation Usage

?

V

Problem

Submissions

Doubts

✓ @Qualifier Annotation Usage

Easy • Score 20/20

Problem statement

Send feedback

Can the @Qualifier annotation be used with primitive types in Spring?

Options: Pick one correct answer from below

☐ Yes, but it requires additional configuration in Spring

☐ Yes, the @Qualifier annotation can be used with primitive types in Spring

☒ No, the @Qualifier annotation can only be used with objects in Spring

☐ None of the above

Solution description

The @Qualifier annotation in Spring can only be used with objects, not primitive types. This is because primitive types do not have unique names or identifiers, so Spring cannot use the @Qualifier annotation to differentiate between them.

## (8) @Scope Annotation

←

Classroom

Spring and Annotations  
@Scope Annotation

?

V

Problem

Submissions

Doubts

✓ @Scope Annotation

Easy • Score 20/20

Problem statement

Send feedback

Which of the following statements is true about the @Scope annotation?

Options: Pick one correct answer from below

☐ It can only be used with @Autowired annotation.

☐ It can only be used on interfaces.

☒ It can be used on any class managed by the Spring container.

☐ It can only be used on primitive data types.

Solution description

- The @Scope annotation in Spring defines the scope of a bean, specifying how the container manages the lifecycle and availability of bean instances.
- The @Scope annotation can be applied to a class or a method. When applied to a class, it defines the scope for all instances of that class.

## (9) Annotation Based Configuration

Problem Submissions Doubts

### Annotation Based Implementation of XML Configuration

Easy • Score 20/20

#### Problem statement

[Send feedback](#)

Which is the correct annotation-based implementation of the below XML configuration?

```

    ...
    <bean id="Ninja" class="com.example.Student" scope="prototype"/>
    ...
  
```

Options: Pick one correct answer from below

☐

```

    ...
    @Component
    @Scope("prototype")
    public class Ninja {
        // ...
    }
    ...
  
```

☐

```

    ...
    @Component("ninja")
    @Scope("prototype")
    public class Ninja {
        // ...
    }
    ...
  
```

☐

```


    ...
    @Component("Ninja")
    @Scope(value="prototype")
    public class Student {
        // ...
    }
    ...
  
```

☒

```

    ...
    @Component("Ninja")
    @Scope("prototype")
    public class Student {
        // ...
    }
    ...
  
```

## (10) Find Bean Scope

 Classroom

Spring and Annotations  
[Find Bean Scope](#)

Problem Submissions Doubts

Find Bean Scope

Easy • Score 20/20

Problem statement

[Send feedback](#)

Below are the Employee Class codes and the main method.

```
***
// Here is the Employee.java

@Component("empl")
public class Employee {
    private String name;

    @PostConstruct
    public void init(){
        System.out.println("Employee bean created");
    }
    @PreDestroy
    public void destroy(){
        System.out.println("Employee bean destroyed");
    }
}

***

// Following is the MyApplication.java file -
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

@SpringBootApplication
public class MyApplication{

    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
        AnnotationConfigApplicationContext("com.example");
        Employee employee = (Employee) context.getBean("empl");
        context.close();
    }
}
```

Options: Pick one correct answer from below

☐ Singleton

☐ Archetype

☒ Prototype

☐ Session

Solution description

In the above output, only the init() method is called, which happens when the bean's scope is a prototype.

## (11) Code Analysis

```
//Student Class
@Component
public class Student {
    private final Result result;

    @Autowired
    public Student(Result result) {
        this.result = result;
    }

    public void showResult() {
        String result = result.generateResult();
        System.out.println(result);
    }
}
```

```
//Result Class
@Component
@Scope("prototype")
public class Result {

    public String generateResult() {
        return "This is a sample result";
    }
}
```

Based on the codes given above what modification is required to change the scope of the Result to a singleton?

Problem

Submissions

Doubts

Code Analysis - Spring and Annotations

Easy • Score 20/20

Problem statement

Send feedback

Below are the Student and Result classes along with the main method

```
// Main Method
@SpringBootApplication
public class MyApp {

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

Options: Pick one correct answer from below

☒ Change @Scope("prototype") to @Scope("singleton")

☐ Add @Component("prototype") to @Component to Result class

☐ Remove the @Scope annotation from MyService

☐ Add @Component("myResult") annotation to Result

Solution description

Changing @Scope("prototype") to @Scope("singleton") modifies the scope of the class Result to singleton. In Spring, the @Scope annotation is used to specify the lifecycle or scope of a Spring bean, determining how long the bean should exist in the Spring application context. Common scopes include singleton and prototype.

## (12) Code Analysis Part II

← Classroom

Spring and Annotations  
Code Analysis Part II

?

V

Problem

Submissions

Doubts

Code Analysis Part II

Easy • Score 20/20

Problem statement

Send feedback

In the previous question what modification is required to make Result a dependency of the Student without using constructor injection?

Options: Pick one correct answer from below

☐ Add @Autowired annotation to a setter method for Result

☒ Add @Autowired annotation the field Result

☐ Use the @Resource annotation on a field for Result

☐ Use the @Inject annotation on a setter method for Result

### (13) Code Analysis Part 3

← Classroom

Spring and Annotations  
Code Analysis Part III

?

V

Problem Submissions Doubts

Problem statement [Send feedback](#)

Continuing the previous problem, Now there is a StudentResult interface with two implementations UndergradResult and PostGradResult. The StudentResult has been injected into Student using @Autowired annotation on the studentResult field. But the application gives a NoUniqueBeanDefinitionException exception at runtime. What changes should be made in the Student class to rectify the error?

```
public interface StudentResult {  
    String generateResult();  
}  
  
@Component  
@Scope("prototype")  
public class UndergradResult implements StudentResult {  
    public String generateResult() {  
        return "This is a UG result";  
    }  
}  
  
@Component  
@Scope("prototype")  
public class PostgradResult implements StudentResult {  
    public String generateResult() {  
        return "This is a PG result";  
    }  
}
```

Options: Pick one correct answer from below

☐ Add @Required annotation to the constructor parameter.

☒ Add @Qualifier("UnderGradResult") annotation to Studentresult field

☐ Use the @Resource annotation on a field for Studentresult

☐ Use the @Inject annotation on a setter method for Studentresult

### (14) @SpringBootApplication Annotation

← Classroom

Spring and Annotations  
@SpringBootApplication Annotation

?

V

Problem Submissions Doubts

✓ @SpringBootApplication Annotation - Spring and Annotations

Easy • Score 20/20

Problem statement [Send feedback](#)

Which of the following statements regarding the @SpringBootApplication annotation in Spring Boot is true?

Options: Pick one correct answer from below

☐ It is used to enable only component scanning.

☐ It is used to enable only auto-configuration.

☐ It is not a valid annotation in Spring Boot.

☒ It is used to enable auto-configuration and component scanning.

Solution description

- The @SpringBootApplication annotation enables auto-configuration and component scanning. It combines three annotations: @EnableAutoConfiguration, @ComponentScan, and @Configuration.
- Option b) and d) are incorrect as they refer to enabling only one of the two features, whereas the @SpringBootApplication annotation enables both features.
- Option d) is incorrect as @SpringBootApplication is a valid annotation in Spring Boot.



## (15) Accessing Context

← Classroom

Spring and Annotations  
Accessing Context

?

V

Problem Submissions Doubts

✓ Accessing Context - Spring and Annotations

Easy • Score 20/20

Problem statement

Which instruction is used to access ApplicationContext using SpringApplication.run?

Send feedback

Options: Pick one correct answer from below

☐ ApplicationContext context = new SpringApplication.run;

☒ ApplicationContext context = SpringApplication.run(MainApplication.class, args);

☐ ApplicationContext context = new ApplicationContext(SpringApplication.run);

☐ ApplicationContext new context = SpringApplication.run(MainApplication.class, args);

Solution description

The SpringApplication.run() bootstraps a spring application as a stand-alone application from the main method. It creates an appropriate ApplicationContext instance and load beans.We can fetch that applicationContext by passing the main Class as an argument in the run() method. Therefore, Option B is correct.

## (16) What is YAML?

← Classroom

Spring and Annotations  
What is YAML

?

V

Problem Submissions Doubts

✓ What is YAML - Spring and Annotations

Easy • Score 20/20

Problem statement

What is YAML?

Send feedback

Options: Pick one correct answer from below

☐ A programming language

☐ A markup language

☒ A data serialisation format

☐ A network protocol

Solution description

YAML is a data serialisation format, meaning it is used to convert data from one format to another.

## (17) YAML Uses

← Classroom

Spring and Annotations  
YAML Uses

?

V

Problem Submissions Doubts

✓ Uses of YAML - Spring and Annotations

Easy • Score 20/20

Problem statement

What are the common uses of YAML?

Send feedback

Options: Pick one correct answer from below

☐ Configuring files

☐ Data serialisation

☐ A markup language for web pages


☒ All of the above

Solution description

YAML has multiple uses, including configuration files, data serialisation, and as a markup language for web pages. It is a universal language that can be applied in various contexts.

## (18) Indentation in YAML

←

 Classroom

Spring and Annotations  
Indentation in YAML ▾

?

V

Problem Submissions Doubts

✔ Indentation in YAML - Spring and Annotations

Easy • Score: 20/20

Problem statement

What is the significance of indentation in YAML?

Send feedback

Options: Pick one correct answer from below

☐ Indentation is optional and doesn't affect the YAML structure.

☒ Indentation is required for proper syntax and readability. ✔

☐ Indentation is used for code comments.

☐ Indentation is used to indicate data types in YAML.

Solution description

In YAML, indentation is crucial as it defines the structure and hierarchy of the data. Proper indentation helps visually represent the relationships between elements such as lists, objects, and key-value pairs. It ensures that the YAML file is well-formed and easily understandable.