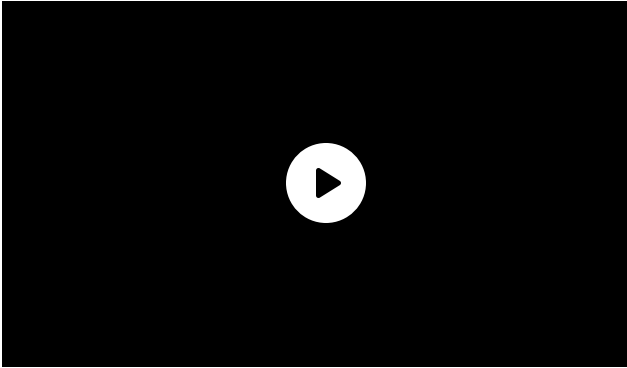


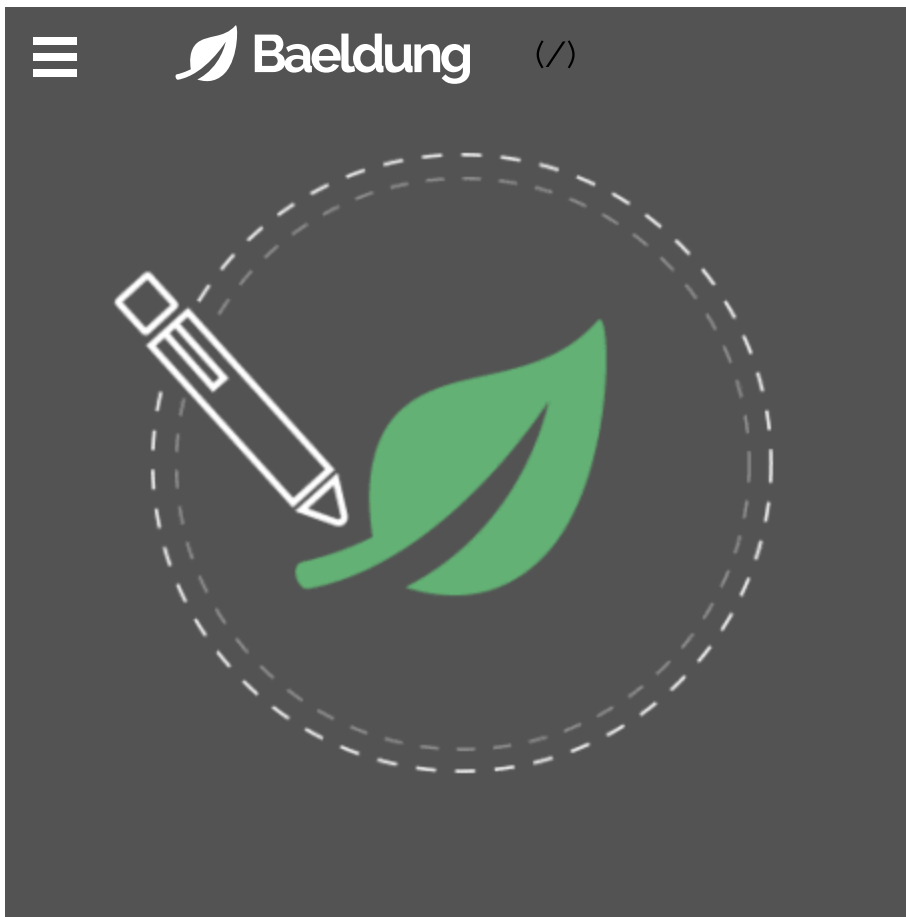
(/)

Top Spring Framework Interview Questions

FEATURED VIDEOS



Last updated: January 11, 2024



Written by: baeldung (<https://www.baeldung.com/author/baeldung>)



Reviewed by: Kevin Gilmore (<https://www.baeldung.com/editor/kevin-author>)

Spring (<https://www.baeldung.com/category/spring>) +

Interview (<https://www.baeldung.com/tag/interview>)

This article is part of a series:

Table of Contents

1. Overview

2. Spring Core

Q1. What Is Spring Framework?

Q2. What Are the Benefits of Using Spring?

Q3. What Spring Sub-Projects Do You Know? Describe Them Briefly.

Q4. What Is Dependency Injection?

Q5. How Can We Inject Beans in Spring?

Q6. Which Is the Best Way of Injecting Beans and Why?

Q7. What Is the Difference Between BeanFactory and ApplicationContext?

Q8. What Is a Spring Bean?

Q9. What Is the Default Bean Scope in Spring Framework?

Q10. How to Define the Scope of a Bean?

Q11. Are Singleton Beans Thread-Safe?

Q12. What Does the Spring Bean Life Cycle Look Like?

Q13. What Is the Spring Java-Based Configuration?

Q14. Can We Have Multiple Spring Configuration Files in One Project?

Q15. What Is Spring Security?

Q16. What Is Spring Boot?

Q17. Name Some of the Design Patterns Used in the Spring Framework?

Q18. How Does the Scope Prototype Work?

3. Spring Web MVC

Q19. How to Get ServletContext and ServletConfig Objects in a Spring Bean?

Q20. What Is a Controller in Spring MVC?

Q21. How Does the @RequestMapping Annotation Work?

4. Spring Data Access

Q22. What Is Spring JdbcTemplate Class and How to Use It?

Q23. How to Enable Transactions in Spring and What Are Their Benefits?

Q24. What Is Spring DAO?


5. Spring Aspect-Oriented Programming (AOP)

Q25. What Is Aspect-Oriented Programming?

Q26. What Are Aspect, Advice, Pointcut, and JoinPoint in AOP?

Q27. What Is Weaving?

(/)

 (https://ads.freestar.com/?
utm_campaign=branding&utm_medium=banner&ut
m_source=baeldung.com&utm_content=baeldung_le
adboard_mid_4)

6. Spring 5

Q28. What Is Reactive Programming?

Q29. What Is Spring WebFlux?

Q30. What Are the Mono and Flux Types?

Q31. What Is the Use of WebClient and WebTestClient?

Q32. What Are the Disadvantages of Using Reactive Streams?

Q33. Is Spring 5 Compatible With Older Versions of Java?

Q34. How Does Spring 5 Integrate With JDK 9 Modularity?

Q35. Can We Use Both Web MVC and WebFlux in the Same Application?

7. Conclusion

1. Overview

In this tutorial, we're going to look at some of the most common Spring-related questions that might pop up during a job interview.

(/)

Further reading:

Java Interview Questions (/java-interview-questions)

Learn the answers to common Java interview questions

Read more (/java-interview-questions) →

Java 8 Interview Questions(+ Answers) (/java-8-interview-questions)

A set of popular Java8-related interview questions and of course answers.

Read more (/java-8-interview-questions) →

Java Collections Interview Questions (/java-collections-interview-questions)

A set of practical Collections-related Java interview questions

Read more (/java-collections-interview-questions) →

2. Spring Core


Q1. What Is Spring Framework?

Spring is the most broadly used framework for the development of Java Enterprise Edition applications. Further, the core features of Spring can be used in developing any Java application.

We use its extensions for building various web applications on top of the Jakarta EE platform. We can also just use its dependency injection provisions in simple standalone applications.

Q2. What Are the Benefits of Using Spring?

Spring targets to make Jakarta EE development easier, so let's look at the advantages:

 (https://ads.freestar.com/?
utm_campaign=branding&utm_medium=banner&utm_source=baeldung.com&utm_content=baeldung_le
adboard_mid_n)

Lightweight – There is a slight overhead of using the framework in development.

Inversion of Control (IoC) – Spring container takes care of wiring dependencies of various objects instead of creating or looking for dependent objects.

Aspect-Oriented Programming (AOP) – Spring supports AOP to separate business logic from system services.

IoC container – manages Spring Bean life cycle and project-specific configurations

MVC framework – used to create web applications or RESTful web services, capable of returning XML/JSON responses

Transaction management – reduces the amount of boilerplate code in JDBC operations, file uploading, etc., either by using Java annotations or by Spring Bean XML configuration file

Exception Handling – Spring provides a convenient API for translating technology-specific exceptions into unchecked exceptions.

Q3. What Spring Sub-Projects Do You Know? Describe Them Briefly.

Core – a key module that provides fundamental parts of the framework, such as IoC or DI

JDBC – enables a JDBC-abstraction layer that removes the need to do JDBC coding for specific vendor databases

ORM integration – provides integration layers for popular object-relational mapping APIs, such as JPA, JDO and Hibernate

Web – a web-oriented integration module that provides multipart file upload, Servlet listeners and web-oriented application context functionalities

MVC framework – a web module implementing the Model View Controller design pattern

AOP module – aspect-oriented programming implementation allowing the definition of clean method-interceptors and pointcuts

Q4. What Is Dependency Injection?

Dependency injection, an aspect of Inversion of Control (IoC), is a general concept stating that we do not create our objects manually but instead describe how they should be created. Then an IoC container will instantiate required classes if needed.

For more details, please look here (<https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html>).

Q5. How Can We Inject Beans in Spring?

A few different options exist in order to inject Spring beans:

- Setter injection

- Constructor injection

- Field injection

The configuration can be done using XML files or annotations.

For more details, check this article ([/inversion-control-and-dependency-injection-in-spring](#)).

(/)

Q6. Which Is the Best Way of Injecting Beans and Why?

The recommended approach is to use constructor arguments for mandatory dependencies and setters for optional ones. This is because constructor injection allows injecting values to immutable fields and makes testing easier.

Q7. What Is the Difference Between BeanFactory and ApplicationContext?

BeanFactory is an interface representing a container that provides and manages bean instances. The default implementation instantiates beans lazily when *getBean()* is called.

In contrast, *ApplicationContext* is an interface representing a container holding all information, metadata and beans in the application. It also extends the *BeanFactory* interface, but the default implementation instantiates beans eagerly when the application starts. However, this behavior can be overridden for individual beans.

For all differences, please refer to the documentation (<https://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html>).

Q8. What Is a Spring Bean?

The Spring Beans are Java Objects that are initialized by the Spring IoC container.

Q9. What Is the Default Bean Scope in Spring Framework?

By default, a Spring Bean is initialized as a *singleton*.

Q10. How to Define the Scope of a Bean?

In order to set Spring Bean's scope, we can use *@Scope* annotation or "scope" attribute in XML configuration files. Note that there are five supported scopes:

Singleton

Prototype

Request

Session

Global-session

For differences, please look here (<https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html>).

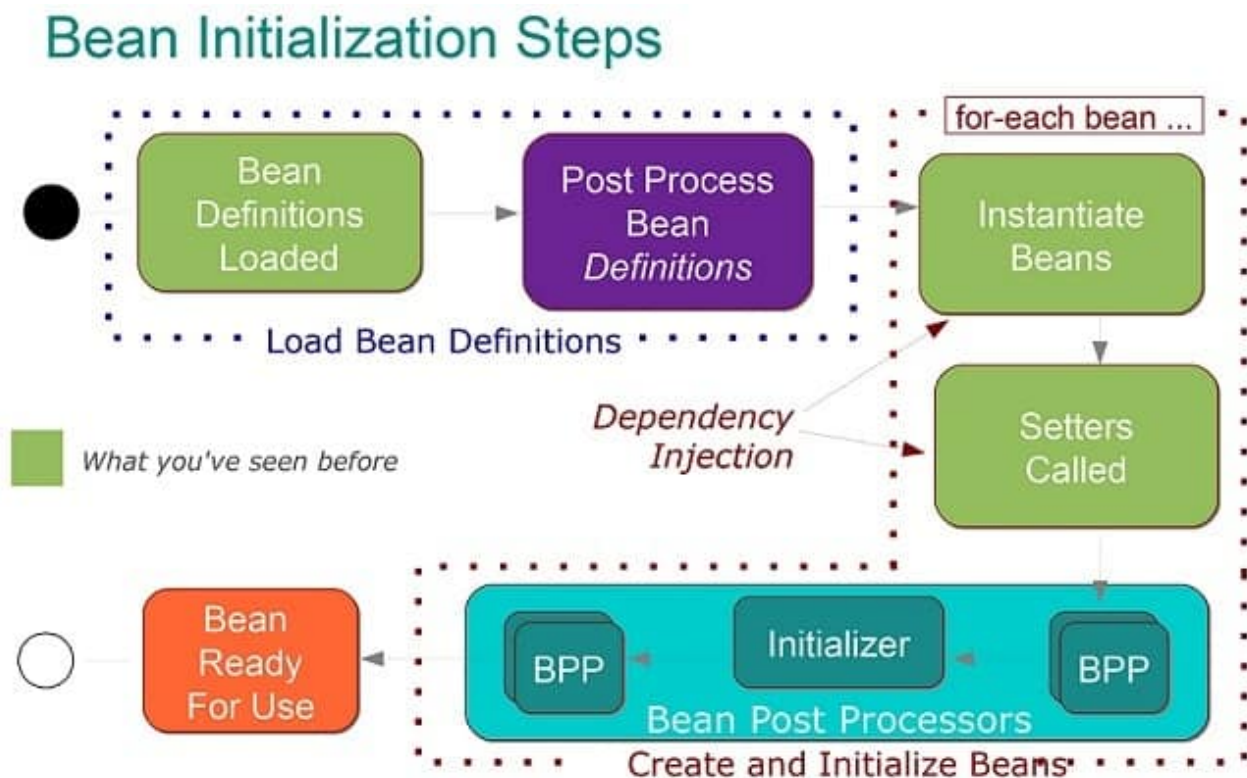
Q11. Are Singleton Beans Thread-Safe?

No, singleton beans are not thread-safe, as thread safety is about execution, whereas the singleton is a design pattern focusing on creation. Thread safety depends only on the bean implementation itself.

Q12. What Does the Spring Bean Life Cycle Look Like?

First, a Spring bean needs to be instantiated based on Java or XML bean definition. It may also be required to perform some initialization to get it into a usable state. After that, when the bean is no longer required, it will be removed from the IoC container.

The whole cycle with all initialization methods is shown in the image (source (<http://www.dineshonjava.com/2012/07/bean-lifecycle-and-callbacks.html>)):



(/wp-content/uploads/2017/06/Spring-Bean-Life-Cycle.jpg)

Q13. What Is the Spring Java-Based Configuration?

It's one of the ways of configuring Spring-based applications in a type-safe manner. It's an alternative to the XML-based configuration.

Also, to migrate a project from XML to Java config, please refer to this article ([/spring-xml-vs-java-config](#)).

Q14. Can We Have Multiple Spring Configuration Files in One Project?

Yes, in large projects, having multiple Spring configurations is recommended to increase maintainability and modularity.

We can load multiple Java-based configuration files:

```
@Configuration
@Import({MainConfig.class, SchedulerConfig.class})
public class AppConfig {
```

Or we can load one XML file that will contain all other configs:

```
ApplicationContext context = new
ClassPathXmlApplicationContext("spring-all.xml");
```

And inside this XML file we'll have the following:

```
<import resource="main.xml"/> (/)  
<import resource="scheduler.xml"/>
```



Q15. What Is Spring Security?

Spring Security is a separate module of the Spring framework that focuses on providing authentication and authorization methods in Java applications. It also takes care of most of the common security vulnerabilities such as CSRF attacks.

To use Spring Security in web applications, we can get started with the simple annotation *@EnableWebSecurity*.

For more information, we have a whole series of articles related to security ([/security-spring](#)).

Q16. What Is Spring Boot?

Spring Boot is a project that provides a pre-configured set of frameworks to reduce boilerplate configuration. This way, we can have a Spring application up and running with the smallest amount of code.

Q17. Name Some of the Design Patterns Used in the Spring Framework?

Singleton Pattern – singleton-scoped beans

Factory Pattern – Bean Factory classes

Prototype Pattern – prototype-scoped beans

Adapter Pattern – Spring Web and Spring MVC

Proxy Pattern – Spring Aspect-Oriented Programming support

Template Method Pattern – *JdbcTemplate*, *HibernateTemplate*, etc.

Front Controller – Spring MVC *DispatcherServlet*

Data Access Object – Spring DAO support

Model View Controller – Spring MVC

Q18. How Does the Scope Prototype Work?

Scope *prototype* means that every time we call for an instance of the Bean, Spring will create a new instance and return it. This differs from the default *singleton* scope, where a single object instance is instantiated once per Spring IoC container.

3. Spring Web MVC

Q19. How to Get *ServletContext* and *ServletConfig* Objects in a Spring Bean?

We can do either by implementing Spring-aware interfaces. The complete list is available here (<http://www.buggybread.com/2015/03/spring-framework-list-of-aware.html>).

We could also use *@Autowired* annotation on those beans:

```
@Autowired .           (/)
ServletContext servletContext;

@Autowired
ServletConfig servletConfig;
```



Q20. What Is a Controller in Spring MVC?

Simply put, all the requests processed by the *DispatcherServlet* are directed to classes annotated with *@Controller*. Each controller class maps one or more requests to methods that process and execute the requests with provided inputs.

To take a step back, we recommend having a look at the concept of the Front Controller in the typical Spring MVC architecture ([/spring-controllers](#)).

Q21. How Does the *@RequestMapping* Annotation Work?

The *@RequestMapping* annotation is used to map web requests to Spring Controller methods. In addition to simple use cases, we can use it for mapping of HTTP headers, binding parts of the URI with *@PathVariable*, and working with URI parameters and the *@RequestParam* annotation.

More details on *@RequestMapping* are available here ([/spring-requestmapping](#)).

For more Spring MVC questions, please check out our article on Spring MVC interview questions ([/spring-mvc-interview-questions](#)).

4. Spring Data Access

Q22. What Is Spring *JdbcTemplate* Class and How to Use It?

The Spring JDBC template is the primary API through which we can access database operations logic that we're interested in:

- Creation and closing of connections
- Executing statements and stored procedure calls
- Iterating over the *ResultSet* and returning results

In order to use it, we'll need to define the simple configuration of *DataSource*:

```
@Configuration
@ComponentScan("org.baeldung.jdbc")
public class SpringJdbcConfig {
    @Bean
    public DataSource mysqlDataSource() {
        DriverManagerDataSource dataSource = new
        DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://localhost:3306/springjdbc");
        dataSource.setUsername("guest_user");
        dataSource.setPassword("guest_password");

        return dataSource;
    }
}
```

For further explanation, check out this quick article (</spring-jdbc-jdbctemplate>).

(/)

Q23. How to Enable Transactions in Spring and What Are Their Benefits?

There are two distinct ways to configure *Transactions* — with annotations or by using Aspect-Oriented Programming (AOP) — each with their advantages.

Here are the benefits of using Spring Transactions, according to the official docs (<http://docs.spring.io/spring/docs/current/spring-framework-reference/html/transaction.html>):

- Provide a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA and JDO

- Support declarative transaction management

- Provide a simpler API for programmatic transaction management than some complex transaction APIs such as JTA

- Integrate very well with Spring's various data access abstractions

Q24. What Is Spring DAO?

Spring Data Access Object (DAO) is Spring's support provided to work with data access technologies like JDBC, Hibernate and JPA in a consistent and easy way.

There is an entire series (/persistence-with-spring-series) discussing persistence in Spring that provides a more in-depth look.

5. Spring Aspect-Oriented Programming

Q25. What Is Aspect-Oriented Programming (AOP)?

Aspects enable the modularization of cross-cutting concerns such as transaction management that span multiple types and objects by adding extra behavior to already existing code without modifying affected classes.

Here is the example of aspect-based execution time logging (/spring-aop-annotation).

Q26. What Are Aspect, Advice, Pointcut and JoinPoint in AOP?

Aspect – a class that implements cross-cutting concerns, such as transaction management

Advice – the methods that get executed when a specific *JoinPoint* with matching *Pointcut* is reached in the application

Pointcut – a set of regular expressions that are matched with *JoinPoint* to determine whether *Advice* needs to be executed or not

JoinPoint – a point during the execution of a program, such as the execution of a method or the handling of an exception

Q27. What Is Weaving?

According to the official docs (<https://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html>), *weaving* is a process that links aspects with other

application types or objects to create an advised object. This can be done at compile time, load time, or runtime. Spring AOP, like other pure Java AOP frameworks, performs *weaving* at runtime.

6. Spring 5

Q28. What Is Reactive Programming?

Reactive programming is about non-blocking, event-driven applications that scale with a small number of threads, with back pressure being a key ingredient that aims to ensure producers don't overwhelm consumers.

These are the primary benefits of reactive programming:

- Increased utilization of computing resources on multicore and multi-CPU hardware
- Increased performance by reducing serialization

Reactive programming is generally event-driven, in contrast to reactive systems, which are message-driven. So, using reactive programming does not mean we're building a reactive system, which is an architectural style.

However, reactive programming may be used as a means to implement reactive systems if we follow the Reactive Manifesto (<https://www.reactivemanifesto.org/>), which is quite vital to understand.

Based on this, reactive systems have four important characteristics:

Responsive – The system should respond in a timely manner.

Resilient – In case the system faces any failure, it should stay responsive.

Elastic – Reactive systems can react to changes and stay responsive under varying workload.

Message-driven – Reactive systems need to establish a boundary between components by relying on asynchronous message passing.

Q29. What Is Spring WebFlux?

Spring WebFlux (<https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html#webflux>) is Spring's reactive-stack web framework, and it's an alternative to Spring MVC.

In order to achieve this reactive model and be highly scalable, the entire stack is non-blocking. Check out our tutorial on Spring 5 WebFlux (</spring-webflux>) for additional details.

Q30. What Are the Mono and Flux Types?

The WebFlux framework in Spring Framework 5 uses Reactor (<https://projectreactor.io>) as its async foundation.

This project provides two core types: *Mono* to represent a single async value and *Flux* to represent a stream of async values. They both also implement the *Publisher* interface defined in the Reactive Streams (<http://www.reactive-streams.org>) specification.

(/)

Mono implements *Publisher* and returns 0 or 1 elements:

```
public abstract class Mono<T> implements Publisher<T> {...}
```



And *Flux* implements *Publisher* and returns *N* elements:

```
public abstract class Flux<T> implements Publisher<T> {...}
```



By definition, the two types represent streams, and so they're both lazy. This means nothing is executed until we consume the stream using the *subscribe()* method. Both types are also immutable, so calling any method will return a new instance of *Flux* or *Mono* ([/reactor-core#streams](#)).

Q31. What Is the Use of *WebClient* and *WebTestClient*?

WebClient ([/spring-5-webclient](#)) is a component in the new Web Reactive framework that can act as a reactive client for performing non-blocking HTTP requests. Since it's reactive client, it can handle reactive streams with back

pressure, and it can take full advantage of Java 8 lambdas. It can also handle both sync and async scenarios.

On the other hand, the *WebTestClient* is a similar class that we can use in tests. Basically, it's a thin shell around the *WebClient*. It can connect to any server over an HTTP connection. It can also bind directly to WebFlux applications using mock request and response objects, without the need for an HTTP server.

Q32. What Are the Disadvantages of Using Reactive Streams?

There are some major disadvantages to using reactive streams:

Troubleshooting a Reactive application is a bit difficult, so be sure to check out our **tutorial on debugging reactive streams (/spring-debugging-reactive-streams)** for some handy debugging tips.

There is limited support for reactive data stores since traditional relational data stores have yet to embrace the reactive paradigm.

There's an extra learning curve when implementing.

Q33. Is Spring 5 Compatible With Older Versions of Java?

In order to take advantage of Java 8 features, the Spring codebase has been revamped. This means older versions of Java cannot be used. So, the framework requires a minimum of Java 8.

Q34. How Does Spring 5 Integrate With JDK 9 Modularity?

In Spring 5, everything has been modularized. This way, we won't be forced to import jars that may not have the functionalities we're looking for.

(/)

Please have a look at our guide to Java 9 modularity ([/java-9-modularity](#)) for an in-depth understanding of how this technology works.

Let's see an example to understand the new module functionality in Java 9 and how to organize a Spring 5 project based on this concept.

We'll first create a new class that contains a single method to return a *String* "HelloWorld". We'll place this within a new Java project — *HelloWorldModule*:

```
package com.hello;
public class HelloWorld {
    public String sayHello(){
        return "HelloWorld";
    }
}
```

Then we create a new module:

```
module com.hello {
    export com.hello;
}
```

Now let's create a new Java Project, *HelloWorldClient*, to consume the above module by defining a module:

```
module com.helloclient {  
    requires com.hello;  
}
```



The above module will be available for testing now:

```
public class HelloWorldClient {  
    public static void main(String[] args){  
        HelloWorld helloWorld = new HelloWorld();  
        log.info(helloWorld.sayHello());  
    }  
}
```



Q35. Can We Use Both Web MVC and WebFlux in the Same Application?

As of now, Spring Boot will only allow either Spring MVC or Spring WebFlux, as Spring Boot tries to auto-configure the context depending on the dependencies that exist in its classpath.

Also, Spring MVC cannot run on Netty. Moreover, MVC is a blocking paradigm and WebFlux is a non-blocking style. So, we shouldn't be mixing both together because they serve different purposes.

(/)

7. Conclusion

In this extensive article, we've explored some of the most important questions for a technical interview all about Spring.

We hope that this article will help in upcoming Spring interviews. Good luck!

« **Previous**

Java Annotations Interview Questions (+ Answers)
(/java-annotations-interview-questions)

Get started with Spring and Spring Boot, through

the *Learn Spring* course:

>> THE COURSE (/ls-course-end)



Learning to build your API
with Spring?

Download the E-book (/rest-api-spring-guide)

Comments are open for 30 days after publishing a post. For any issues past this date, use the Contact form on the site.

(/)

COURSES

[ALL COURSES \(/ALL-COURSES\)](#)

[ALL BULK COURSES \(/ALL-BULK-COURSES\)](#)

[ALL BULK TEAM COURSES \(/ALL-BULK-TEAM-COURSES\)](#)

[THE COURSES PLATFORM \(HTTPS://COURSES.BAELDUNG.COM\)](https://courses.baeldung.com)

SERIES

[JAVA "BACK TO BASICS" TUTORIAL \(/JAVA-TUTORIAL\)](#)

[JACKSON JSON TUTORIAL \(/JACKSON\)](#)

[APACHE HTTPCLIENT TUTORIAL \(/HTTPCLIENT-GUIDE\)](#)

[REST WITH SPRING TUTORIAL \(/REST-WITH-SPRING-SERIES\)](#)

[SPRING PERSISTENCE TUTORIAL \(/PERSISTENCE-WITH-SPRING-SERIES\)](#)

[SECURITY WITH SPRING \(/SECURITY-SPRING\)](#)

[SPRING REACTIVE TUTORIALS \(/SPRING-REACTIVE-GUIDE\)](#)

[JAVA ARRAY \(HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/JAVA-ARRAY\)](https://www.baeldung.com/category/java/java-array)

ABOUT

[ABOUT BAELDUNG \(/ABOUT\)](#)
[THE FULL ARCHIVE \(/FULL-ARCHIVE\)](#) (/)
[EDITORS \(/EDITORS\)](#)
[JOBS \(/TAG/ACTIVE-JOB/\)](#)
[OUR PARTNERS \(/PARTNERS\)](#)
[PARTNER WITH BAELDUNG \(/ADVERTISE\)](#)

[TERMS OF SERVICE \(/TERMS-OF-SERVICE\)](#)
[PRIVACY POLICY \(/PRIVACY-POLICY\)](#)
[COMPANY INFO \(/BAELDUNG-COMPANY-INFO\)](#)
[CONTACT \(/CONTACT\)](#)