Spring Boot Interview Questions

1.What is the Spring Boot?

2. What are the advantages of Spring Boot?

3. What are the different Spring Boot Components?

4. What are Spring Boot Starters?

5. Name some of the starter provided by Spring Boot?

6. What is Auto-Configuration in Spring Boot?

7. Can we use Spring Boot for non-Spring application?

8. What are the different options for creating the Spring Boot application

9. What is the Spring Boot Initilizr?

10. What are the advantages of Spring Boot Initializr?

11. How can I reload my Spring Boot changes without restarting the server?

12.What are the embedded containers supported by Spring Boot?

13. What is the Spring Boot Actuator?

14. How to run Spring Boot application to custom port?

15. How can we override default properties in Spring Boot?

16. How can we create a custom endpoint in Spring Boot Actuator?

17. What logging support provided by Spring Boot? How can we control logging level in Spring Boot?

18. How to implement security for Spring boot application?

19. How to configure database using Spring Boot?

20. How can we use Jetty instead of tomcat in our web application?

21. Why do we need spring-boot-maven-plugin?

22. How to disable specific auto-configuration in spring boot?

23. What is the use of YAML in Spring Boot?

24. What is new in Spring Boot 2.0?

25. What is @SpringBootApplication annotation?

26. How to include custom static content in Spring Boot application (e.g. custom JS code)?

27. How to use a profile with Spring Boot?

28. How to generate a WAR file with Spring Boot?

29.What is a Spring Bean?

30. What is the use of @Configuration annotation?

31.What is the internal working of @SpringBootApplication annotation?

32.Explain the need of dev-tools dependency.

33. Difference between Spring and Spring Boot.

34.How to create a spring boot project using Spring Initializer.

35. What is component scanning?

36. What is Auto-wiring?

37. What is a Runner and its use?

38. What is ApplicationRunner in SpringBoot?

39.What is CommandLineRunner in SpringBoot?

40.Explain Constructor Injection.

41.Explain Setter Injection.

42.Explain Field Injection.

43. Can you override the default web server in Spring Boot.

44.What is Spring Data JPA?

45. Explain features of Spring Data JPA?

46. How to create a custom Repository class in Spring JPA?

47. Difference between CRUDRepository and JPARepository.

48. What is the purpose of save () method in CrudRepository.

49. Difference between findById() and getOne().

50. Explain @Transactional annotation in Spring.

51. What is the use of @EnableJpaRepositories method?

52. .Explain few CrudRepository methods.

53. Difference between delete () and deleteInBatch() methods.

54. What is the use of @Modifying annotation?

55. What is Spring MVC?

56. Explain the flow of Spring MVC.

57. What is Dispatcher Servlet.

58. What is the use of @Controller annotation.

59. . Difference between @RequestParam and @PathVariable.

60. Explain @Service and @Repository annotations.

61. What is the purpose of @Model Attribute?

62. How does Spring MVC support for validation.

63. Explain @GetMapping and @PostMapping.

64. Difference between @Controller and @RestController annotations.

65. Explain the use of @ResponseEntity annotation.

66. How to handle exceptions in Spring MVC.

67. Explain @ControllerAdvice in Spring Boot.

68.How will you enable actuator in spring boot application.

69. What are the actuator endpoints which are needed to monitor the application.

70. How to get the list of beans available in spring application.

71. How to enable all endpoints in actuator?

72. What is a shutdown in the actuator?

73. . What is Spring Security?

74. What are the features of Spring Security?

75. How to implement JWT?

76. What is DelegatingFilterProxy in Spring Security.

77. What is Spring Security OAuth2.

78. What is the advantage of using JWT Token?

79.What is authentication?

80.What is authorization?

81. What is filter Chain Proxy?

82. What is security context in Spring Security.

83. Difference between has Authority and hasRole?

84. How to enable spring boot security in spring boot project?

85.What is Basic Authentication?

86.What is Digest Authentication?

87. How to get current logged in user in spring security.

88.What is SSL and its use?

89. What is salting?

90.What is hashing in spring security.

91.How to secure passwords in a web application?

92.What is AuthenticationManager in spring security?

93. What are the various ways to implement security in spring boot project?

94. What are the essential components of Spring Boot?

95. What is the use of profiles in Spring Boot?

96. How can you set active profile in Spring Boot.

97. What is AOP?

98. What is YAML?

99. Use of @Profile annotation.

100. How to get current profile in Spring Boot.

100. How does Spring Boot handle dependency management?

101. What is the purpose of the application.properties or application.yml file?

102. How can you override the default properties in Spring Boot?

103. What is the purpose of the @Autowired annotation in Spring Boot?

104. What is the purpose of the @RestController annotation?

105. Explain the difference between @Component , @Service , and @Repository annotations in Spring Boot:

106. Explain the concept of Bean scopes in Spring Boot.

107. What is the purpose of the `@Value` annotation in Spring Boot?

108. How can you enable HTTPS in a Spring Boot application?

109. What is the purpose of the @RestControllerAdvice annotation?

110. Explain the purpose of the @PathVariable annotation in Spring Boot.

111. What is the purpose of the @Scheduled annotation in Spring Boot?

112. What is the purpose of the @EnableAutoConfiguration annotation in Spring Boot?

113. What is the purpose of the @Profile annotation in Spring Boot?

114. What is the purpose of the @Bean annotation in Spring Boot?

115. Explain the concept of a RESTful API in Spring Boot.

116. What is @RestController annotation in Spring boot and how it differs from @controller?

117. What is Spring Cloud?

118. REST and HTTP

119. Annotations used in Spring Boot

Spring Boot Interview Questions and Answers

1.What is the Spring Boot?

Spring Boot is an opinionated framework for building and running Spring applications. Spring Boot is not a framework for writing applications, think of Spring Boot as a tool which can do these initial tasks for us automatically. While working on big enterprise projects involving several frameworks, it is very complex to handle all configurations and making sure required dependencies are in place. Spring Boot focuses on developer productivity by providing smart auto configuration modules and handling all configurations and dependencies for us.

2. What are the advantages of Spring Boot?

- It simplifies Spring dependencies by taking the opinionated view.
- Spring Boot provides a preconfigured set of technologies/framework to reduces error-prone configuration so we as a developer focused on building our business logic and not thinking of project setup.
- It reduces development code by avoiding a lot of boilerplate code.
- Easier to integrate Spring Boot Application with Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
- You really don't need those big XML configurations for your project.
- Embed Tomcat, Jetty or Undertow directly.
- Provide opinionated Maven POM to simplify your configuration.

3. What are the different Spring Boot Components?

- Boot Initializer
- Spring Boot Starter
- Auto Configurator.
- Spring Boot CLI.
- Actuator.

4. What are Spring Boot Starters?

Spring Boot Starters are the set of convenient dependency descriptors which can be easily included in any level of application. These starters work as a bootstrapping process for the Spring related technologies, we no longer need to worry about the dependencies and they will be automatically managed by Spring Boot Starters. The starters contain a lot of the dependencies that you need to get a project up and running quickly and with a consistent, supported a set of managed transitive dependencies. To summarize, Spring Boot Starters are just JAR files used by Spring Boot for auto-dependency.

5. Name some of the starter provided by Spring Boot?

- spring-boot-starter-web – Web and RESTful applications
- spring-boot-starter-security – Spring Security
- spring-boot-starter-data-jpa – Spring Data JPA
- spring-boot-starter-test – Unit testing
- spring-boot-starter-hateoas – Add HATEOAS features
- spring-boot-starter-data-jpa – Spring Data JPA with Hibernate

6. What is Auto-Configuration in Spring Boot?

It takes a lot of configurations and boilerplate code create a simple Spring MVC application without Spring Boot. Spring Boot Auto Configuration provides an opinionated approach to bootstrap your application. Auto-Configuration will attempt to automatically try to set up our application with default behaviour based on the jars in the classpath.

For example, if Spring Boot finds HSQLDB in our classpath, it will automatically configure an in-memory database for us. Think of the auto-configuration as an intelligent system which can provide ready to use the application to us based on the configured jars in our classpath.

7. Can we use Spring Boot for non-Spring application?

No, Spring Boot has limited to Spring based application only. We cannot use Spring Boot for non-Spring applications.

8. What are the different options for creating the Spring Boot application

There are multiple options to create a Spring Boot application. We can use any of the following approaches

● Spring Initializer

● Boot CLI.

● Using Maven

● IDE project wizard Read Building an Application with Spring Boot for detail.

9. What is the Spring Boot Initilizr?

Spring Boot Initilizr is a Spring Boot tool to bootstrap Spring Boot or Spring Applications very easily. Spring Initializr is also integrated with all major Java IDEs along with CLI.

10. What are the advantages of Spring Boot Initializr?

Spring Boot Initializr provides a simple interface to quickly bootstrap a Spring Boot application. Here are some of the benefits or advantages of using Initilizr.

● Spring Initializr provides an extensible API to generate quick start projects.

● Reduce time to create an application setup. Application setup can be created using a few clicks.

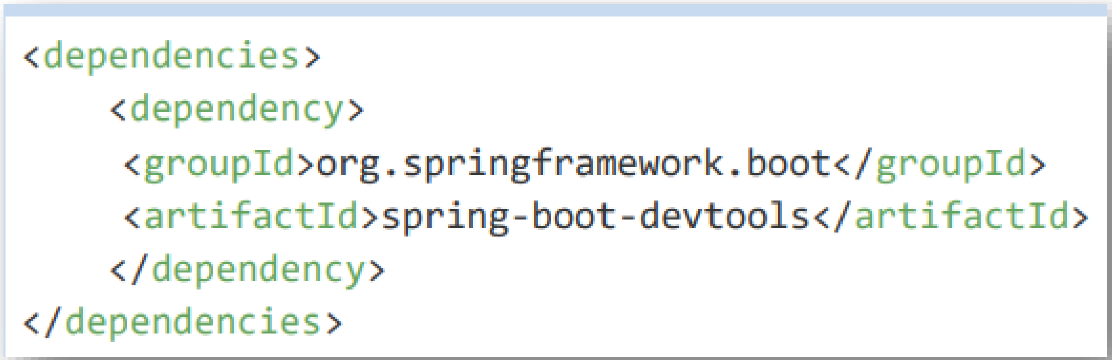
● It increases Productivity

● Initializr offers a configuration structure to define all the aspects related to the project to generate: list of dependencies, supported java and boot versions.

11. How can I reload my Spring Boot changes without restarting the server?

This is achievable by Spring Boot Dev Tools module. It's a powerful tool for development. It helps developers to shorten the development cycle and enable easy deployment and testing during development.

To enable this feature, add the following dependency to the Maven POM file.



```
<dependencies>
    <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-devtools</artifactId>
    </dependency>
</dependencies>
```

12.What are the embedded containers supported by Spring Boot?

Spring Boot includes support for the following embedded containers

- Tomcat
- Jetty
- Undertow.

Use the right "Starter" to configure the embedded container.

13. What is the Spring Boot Actuator?

The actuator provides production-ready features for Spring Boot application. It will help us to check and manage our application in the production environment. We don't need any code to get these features since they are available once the actuator dependency is in the class-path. The actuator provides features like auditing, health, metrics, environment information, thread dump etc. using HTTP endpoints

14. How to run Spring Boot application to custom port?

Use the application.properties file to configure a custom port for Spring Boot application. To change the server port, use server.port property.

```
server.port=9001
```

Read Spring Boot Web Application Configuration for more detail.

15. How can we override default properties in Spring Boot?

Spring Boot advocate convention over configuration. Spring Boot externalize application configurations through application.properties file. These properties work as default values for the Spring Boot application. To override these default values, Spring Boot provides the following options.

● Create an application.properties file in the classpath for overriding specific properties for Spring Boot.

  ○ For Maven based project, this file will be under /src/main/resource.

● application.yml file in the classpath for overriding specific properties for Spring Boot.

  ○ For Maven based project, this file will be under /src/main/resource.

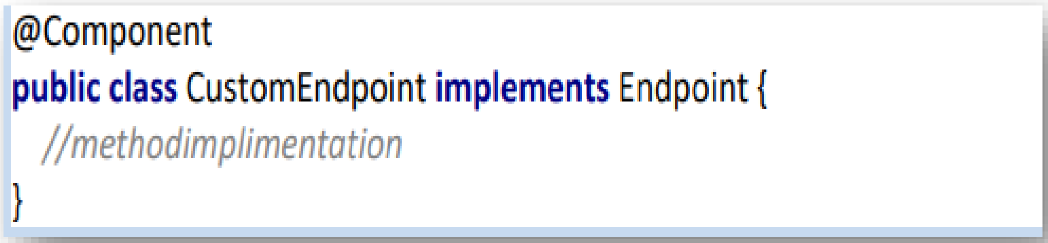
● Through command line switches

e.g. Server HTTP port default to 8080 in the default application.properties file. To change this port to 9090, add below entry in the custom application.properties file

server.port=9090

16. How can we create a custom endpoint in Spring Boot Actuator?

To create a custom endpoint using Spring Boot 1.x, we should expose the instance of the custom endpoint class as a bean. We need to implement Endpoint <T.> interface.



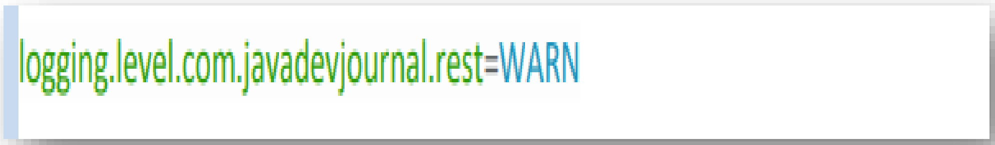
```
@Component
public class CustomEndpoint implements Endpoint {
  //methodimplimentation
}
```

Spring Boot 2.x changed it completely by introducing @Endpoint annotation. Spring Boot expose endpoints with @Endpoint, @WebEndpointor and @WebEndpointExtension over HTTP using Jersey, Spring MVC, or Spring Web Flux. Read Custom Endpoint in Spring Boot Actuator for more detail.

17. What logging support provided by Spring Boot? How can we control logging level in Spring Boot?

Spring Boot provides options to use all popular logging API using the relevant starter, by default Spring Boot use Commons Logging for its internal logging. If we are using Spring Boot Starters for our application, Logback will be used for logging by default unless we want to use any other logging API. To use any other logging API, we need to add the correct starter in our application. In case we like to use Log4j2 for logging configuration, all you have to add the log4j2 starter in your application (You may have to exclude Logback using pom.xml file).

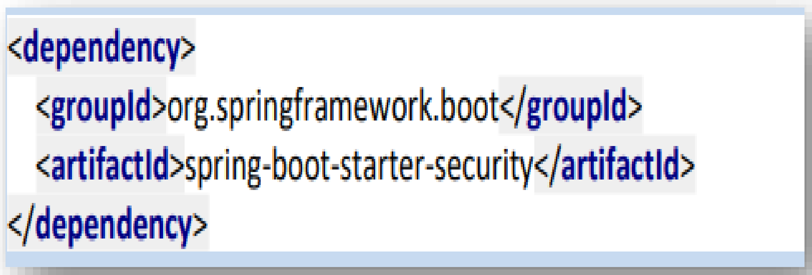
Spring Boot provides an easy way to configure and set logging levels for your application. We can use application.properties file to configure the desired Logging level for our application by using 'logging.level.*=LEVEL'. Here is an example for the same. Read Spring Boot Logging for more detail.



```
logging.level.com.javadevjournal.rest=WARN
```

18. How to implement security for Spring boot application?

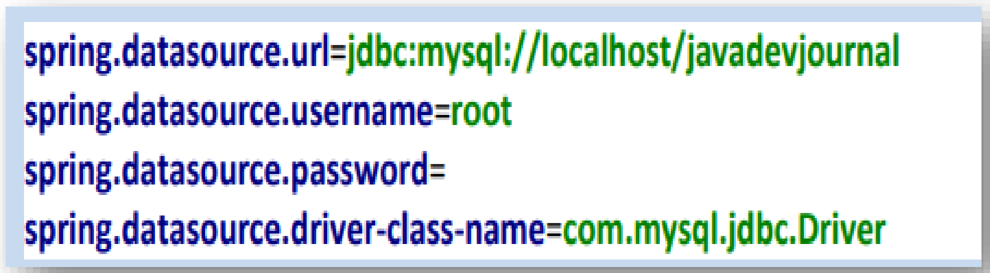
Use the spring-boot-starter-security starter to enable the Spring security support in your Spring Boot application. org.springframework.boot spring-boot-starter-security



```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

19. How to configure database using Spring Boot?

The Spring Framework provides extensive support for working with SQL databases, from direct JDBC access using JdbcTemplate to complete “objectrelational mapping” technologies such as Hibernate. To connect configure the database for your Spring Boot application, use the spring-boot-starter-jdbc or spring-boot-starter-data-jpa starters. For datasource configuration, use the application.properties file in your application.

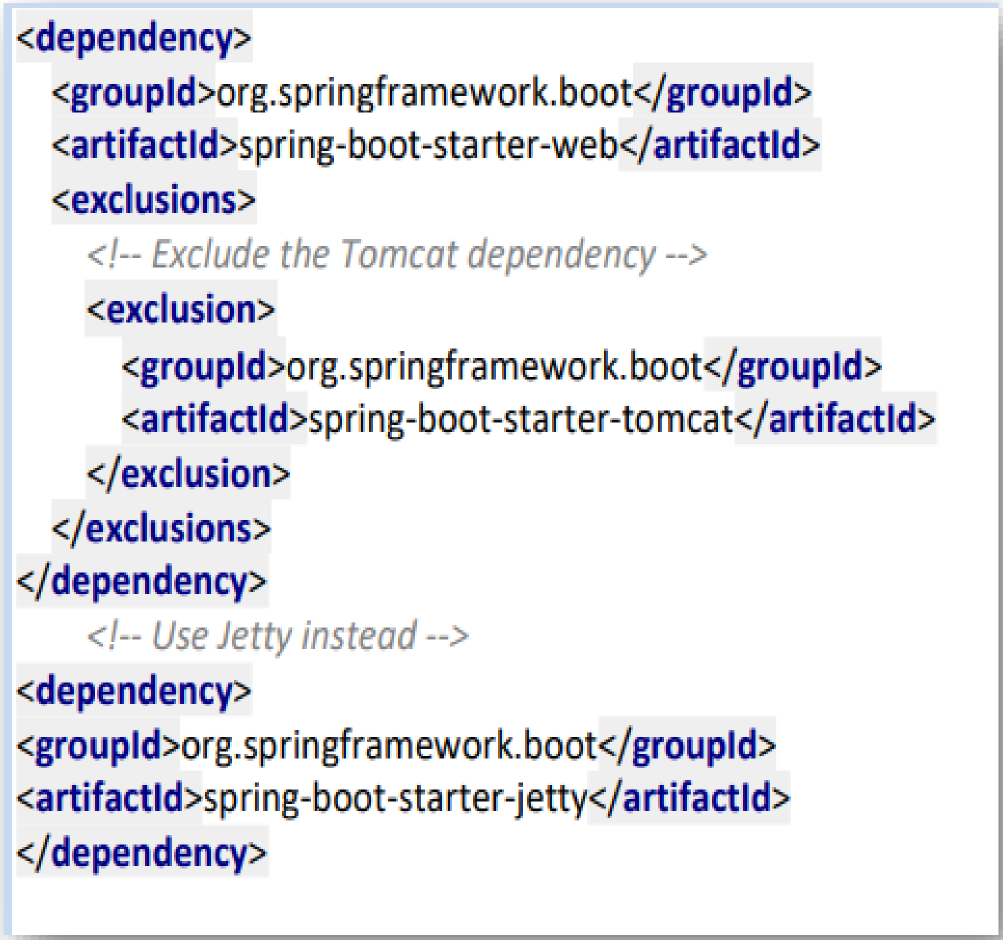


```
spring.datasource.url=jdbc:mysql://localhost/javadevjournal
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

Above example is to configure MySQL in your application. For more information read Configuring MySQL for Spring Boot Application

20. How can we use Jetty instead of tomcat in our web application?

Spring Boot web starters use Tomcat as the default embedded servlet container. When switching to a different HTTP server, we need to exclude the default dependencies in addition to including the one we need. Spring Boot provides separate starters for HTTP servers to help make this process as easy as possible. To use Jetty, we need to exclude Tomcat and include Jetty in our application’s pom.xml file.



```xml
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
   <exclusions>
      <!-- Exclude the Tomcat dependency -->
      <exclusion>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-tomcat</artifactId>
      </exclusion>
   </exclusions>
</dependency>
      <!-- Use Jetty instead -->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

21. Why do we need spring-boot-maven-plugin?

Spring Boot Maven plugin provides Spring Boot support in maven. This plugin provides options to create an executable jar or war files. Here are some of the goals for this plugin.

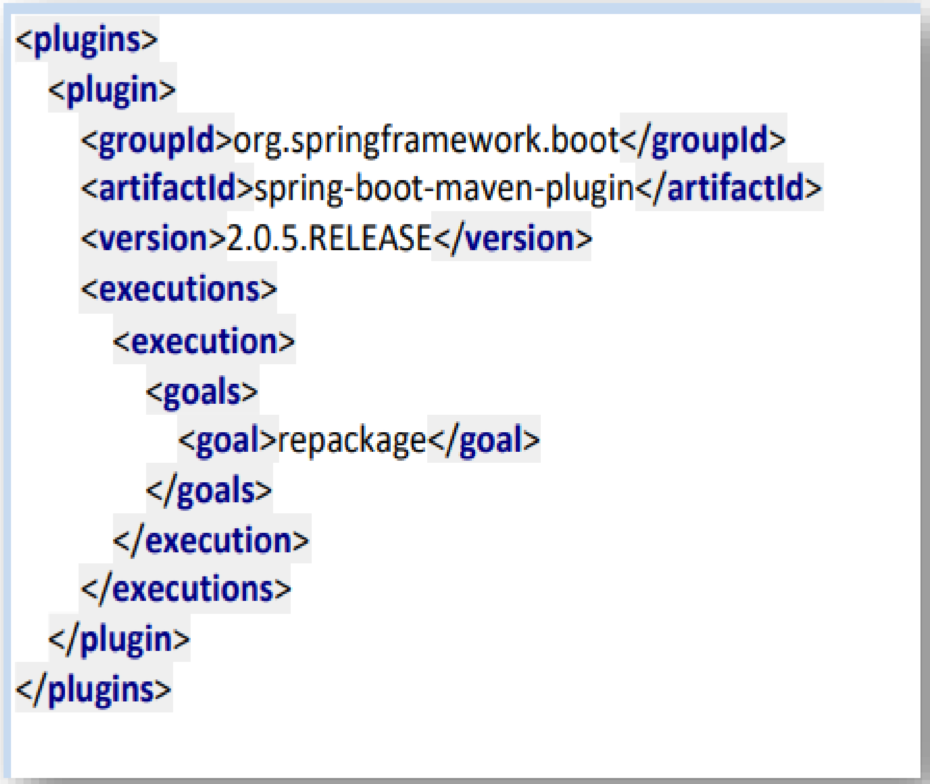
● boot: run runs your Spring Boot application.

● spring-boot:repackage repackages your jar/war to be executable.

● spring-boot:start and spring-boot:stop to manage the lifecycle of your Spring Boot application (i.e. for integration tests).

● spring-boot:build-info generates build information that can be used by the Actuator.

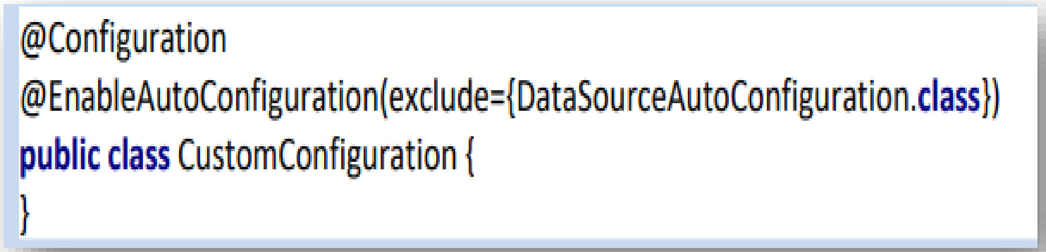
To include this plugin in your project, add XML in the plugins section of your pom.xml



```xml
<plugins>
  <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <version>2.0.5.RELEASE</version>
    <executions>
      <execution>
        <goals>
          <goal>repackage</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
</plugins>
```

22. How to disable specific auto-configuration in spring boot?

To exclude specific auto-configuration classes, use the exclude attribute of @EnableAutoConfiguration to disable them. Here is a sample code for the same.

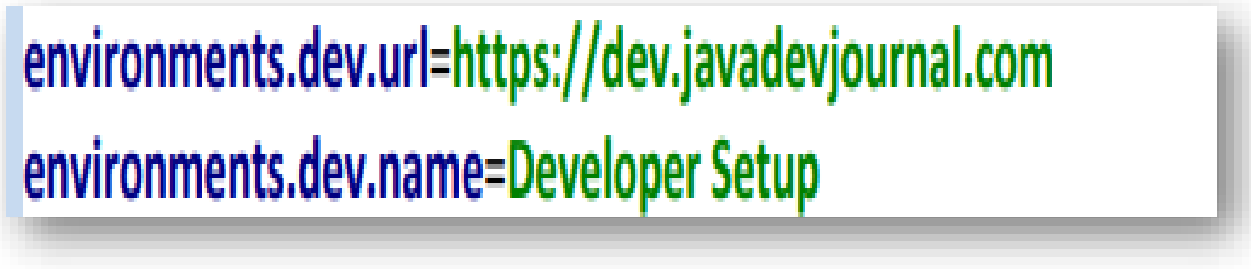


```java
@Configuration
@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})
public class CustomConfiguration {
}
```

23. What is the use of YAML in Spring Boot?

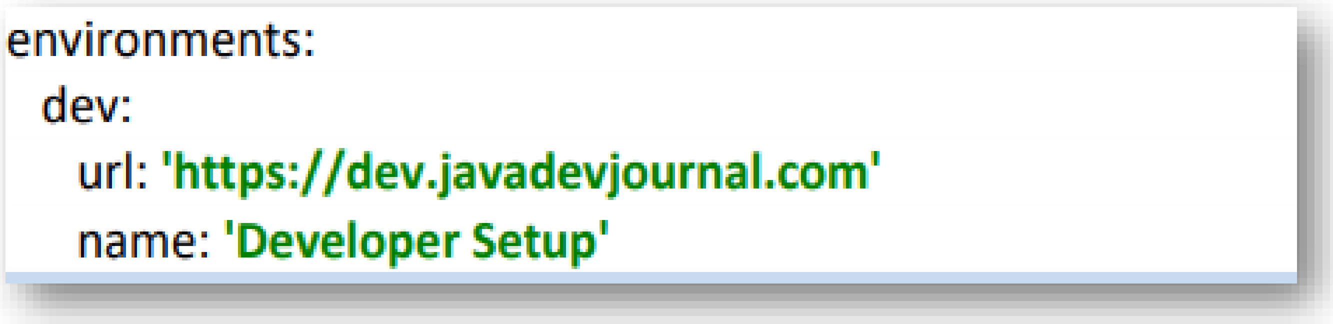
YAML is a superset of JSON.Spring Boot YAML as an alternative to the application.properties file to define your project properties. The SpringApplication class automatically supports YAML as an alternative to properties whenever you have the SnakeYAML library on your classpath.

Let’s take the following example of the application.properties file.



```
environments.dev.url=https://dev.javadevjournal.com
environments.dev.name=Developer Setup
```

This can be represented in the YAML files as follows.



```
environments:
  dev:
    url: 'https://dev.javadevjournal.com'
    name: 'Developer Setup'
```

24. What is new in Spring Boot 2.0?

Spring Boot 2.0 brings a number of features changes to the Spring Boot framework.

● Spring Boot 2.0 is baselined to Java 8. Therefore, Spring Boot 2.0 requires Java 8 or later. Consequently, it doesn't support Java 6 and Java 7 anymore.

● Java 9 is supported with Spring Boot 2.0.

● Spring Boot 2.0 requires Spring Framework 5.0 with Reactive support.

● Embedded servlet containers support got upgraded

  ○ Minimum Tomcat version is 8.5
  ○ Jetty is 9.4

● Spring Boot 2.0 supports HTTP/2 with the help of server.http2.enabledproperty.

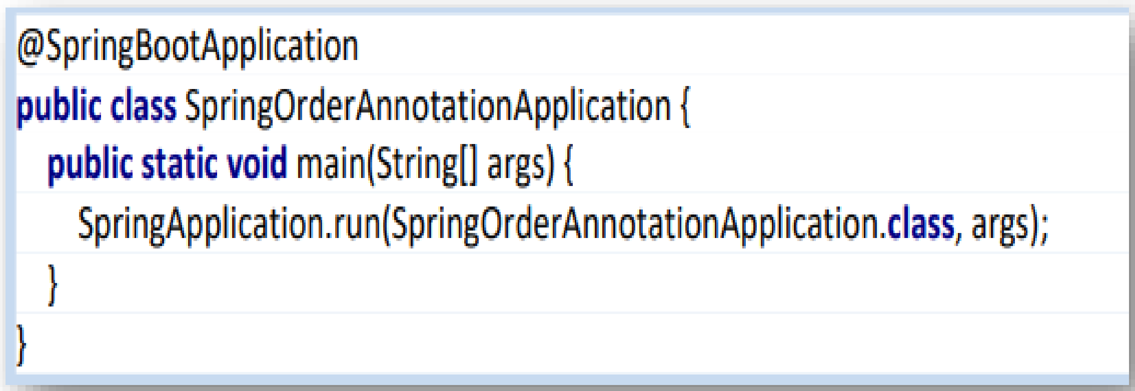
● The framework requires Gradle 4.x in case you are using Gradle as your build tool.

● Security configuration simplified in Spring Boot 2.0.

● A brand-new actuator architecture, with support for Spring MVC, Web Flux and Jersey.

25. What is @SpringBootApplication annotation?

This is one of the most important and core annotations from Spring Boot. We use this annotation to mark the main class of our Spring Boot application.



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

@SpringBootApplication is a convenience annotation that is equal to declaring @Configuration, @EnableAutoConfigurationand @ComponentScan with their default attributes.

You have the option to use @Configuration, @EnableAutoConfiguration, and @ComponentScan individually but the recommendation is to @SpringBootApplication annotation.

26. How to include custom static content in Spring Boot application (e.g. custom JS code)?

Spring Boot search specific location in the project for serving static contents. By default, Spring Boot serves static content from a directory called /static (or /public or /resources or /META-INF/resources) in the classpath or from the root of the ServletContext.

We can put our custom static content in any of the above folders. For example, put the custom.js file under /resources/static/custom.js. To refer to this file in the view, simply use the following code

<script src="/js/test.js"></script>

27. How to use a profile with Spring Boot?

Spring Boot provides multiple ways to active profile. We can pass profile information through the command line or use application.properties, Spring Boot also provide a way to set profile programmatically.

Use profile specific configuration files in out Spring Boot application. We need to the naming convention of application-{profile}.properties where the profile defines the name of the intended profile. Profile specific files will be loaded from the same location as application.properties file.

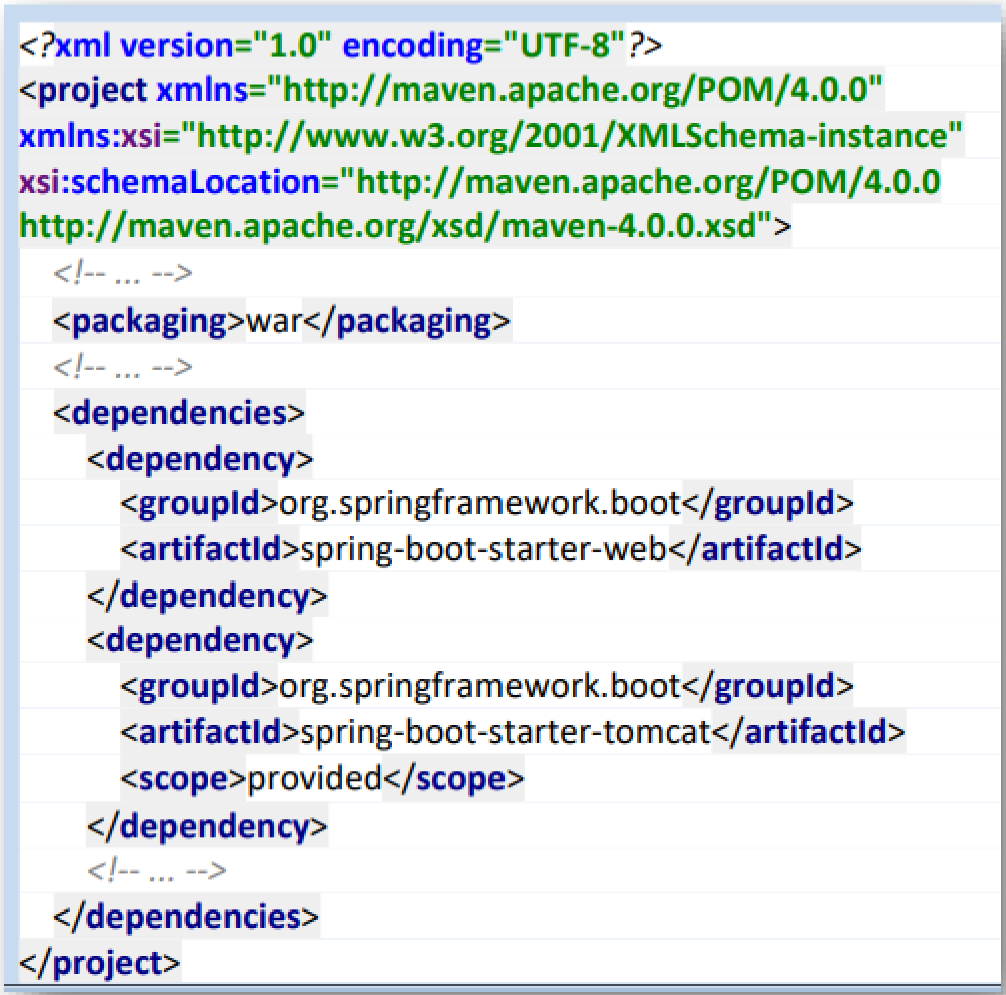
28. How to generate a WAR file with Spring Boot?

We can control the package type generation in our Spring Boot project using spring-boot-maven-plugin to build a war file, **we need to follow these 2 steps.**

1. Set the packaging type as a war in our **pom.xml file.**

2. Mark the embedded container dependencies as "provided" (To build a war file that is both executable and deployable into an external container.)

Here is a snapshot from **pom.xml**



```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <!-- ... -->
  <packaging>war</packaging>
  <!-- ... -->
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
      <scope>provided</scope>
    </dependency>
    <!-- ... -->
  </dependencies>
</project>
```

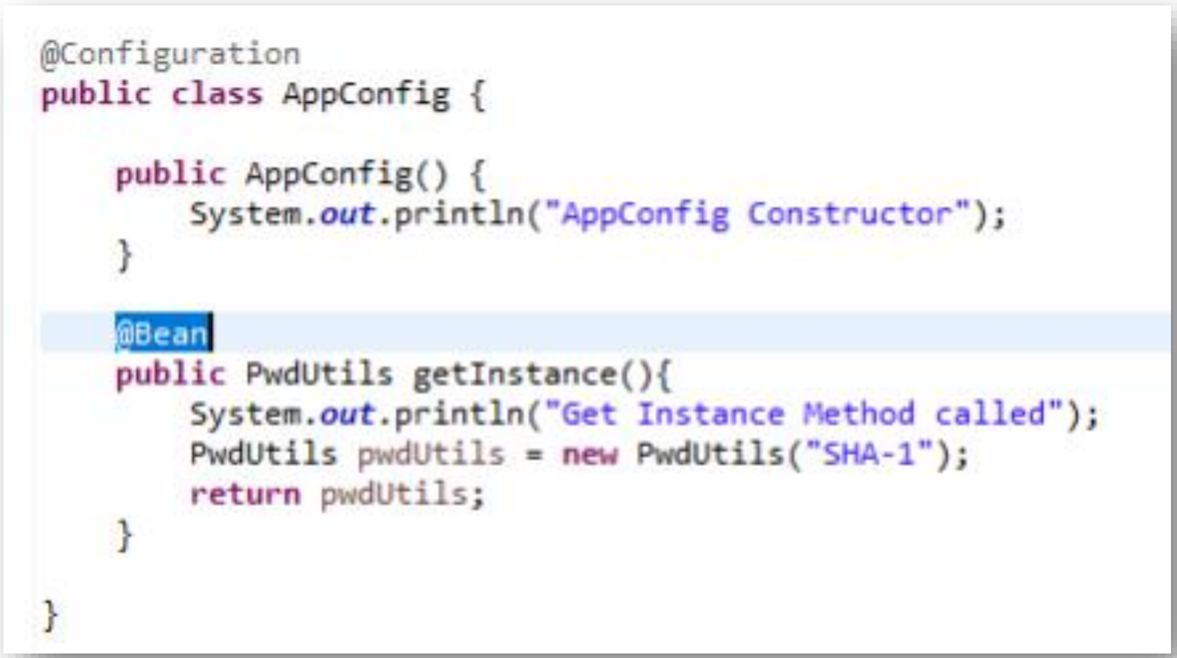
29.What is a Spring Bean?

A java class which is managed by the IOC container is called as Spring Bean.The life cycle of the spring bean are taken care by the IOC container. A spring bean can be represented by using the below annotations.

- @Component
- @Service
- @Repository
- @Configuration
- @Bean
- @Controller
- @RestController

30. What is the use of @Configuration annotation?

A class which is used to provide few configurations such as Swagger configuration, Kafka configuration,etc can be represented using @Configuration annotation. This class contains Bean methods to customize the object creation and returns the object which can be respresented as a Spring Bean by the IOC container.



```
@Configuration
public class AppConfig {

    public AppConfig() {
        System.out.println("AppConfig Constructor");
    }

    @Bean
    public PwdUtils getInstance(){
        System.out.println("Get Instance Method called");
        PwdUtils pwdUtils = new PwdUtils("SHA-1");
        return pwdUtils;
    }

}
```

31.What is the internal working of @SpringBootApplication annotation?

It is a combination of three annotations @ComponentScan, @EnableAutoConfiguration and @Configuration.

@Configuration: It is a class level annotation which indicates that a class can be used by the Spring IOC container as a source of bean definition. The method annotated with @Bean annotation will return an object that will be registered as a Spring Bean in IOC.

@ComponentScan: It is used to scan the packages and all of its sub-packages which registers the classes as spring bean in the IOC container.

@EnableAutoConfiguration: This annotation tells how Spring should configure based on the jars in the classpath. For eg , if H2 database jars are added in classpath , it will create datasource connection with H2 database.

32.Explain the need of dev-tools dependency.

The main aim of adding dev-tools dependency is to improve the development time. When this dependency is included in the project, it automatically restarts the server when there are any modifications made to the source code which helps in reducing the effort of a developer to manually build and restart the server.

33. Difference between Spring and Spring Boot.

Spring – It is an opensource J2EE framework which helps in developing web based and enterprise applications easily. Its main feature is dependency injection by which we can achieve loosely coupling while developing the application. In order to develop a web application, developer needs to write a lot of code for configuring the dispatcher servlet in web.xml, configuring the database,etc. This can be avoided while using Spring Boot.It does not support embedded servers and embedded database integration.

Spring Boot – Spring Boot is built on top of Spring framework which provides flexibility to design applications in a rapid and easier approach. It provides auto configuration feature through which it reduces the developer's effort to write large xml configuration. It provides support for embedded server without the need to install it explicitly.

34.How to create a spring boot project using Spring Initializer.

Spring initializer is a web-based tool developed by Pivotal. With the use of it, we can easily create the project structure needed to develop Spring based application.

- Go to the official Spring Initialize website: https://start.spring.io
- Select the project details such as language, spring boot version, build tool which is needed for¬ application development as:

    - Language: Java
    - Java Version: 1.8
    - Spring Boot: 2.1.4
- Add the required dependencies and click on Generate project. It shall the download the¬ project in your system.
- Import the zip file into the eclipse.

35. What is component scanning?

Component scanning is the process of identifying the Spring beans in the packages and its sub packages. In a spring boot application, the packages which contains the SpringBootApplication class is called as base package and will be scanned implicitly.

@ComponentScan is used to scan the packages and detect the spring beans which will be managed by the IOC container.

If we have more than one base package, then we need to scan the base package using @ComponentScan in Spring start class.

Syntax:

@ComponentScan(basePackages = "in.anmolkit.service")

36. What is Auto-wiring?

Autowiring is the process of injecting one class object into another class. It cannot be implied on primitive types and String type.

Autowiring can be done in 3 ways:

- Constructor Injection
- Setter Injection
- Field Injection

37. What is a Runner and its use?

Runner classes are used to execute the piece of code as soon as the application starts. The code inside the runner classes will execute once on bootstrap of the application. There are mainly used to setup a data source, load the data into cache, etc. These runners will be called from SpringApplication.run() method.

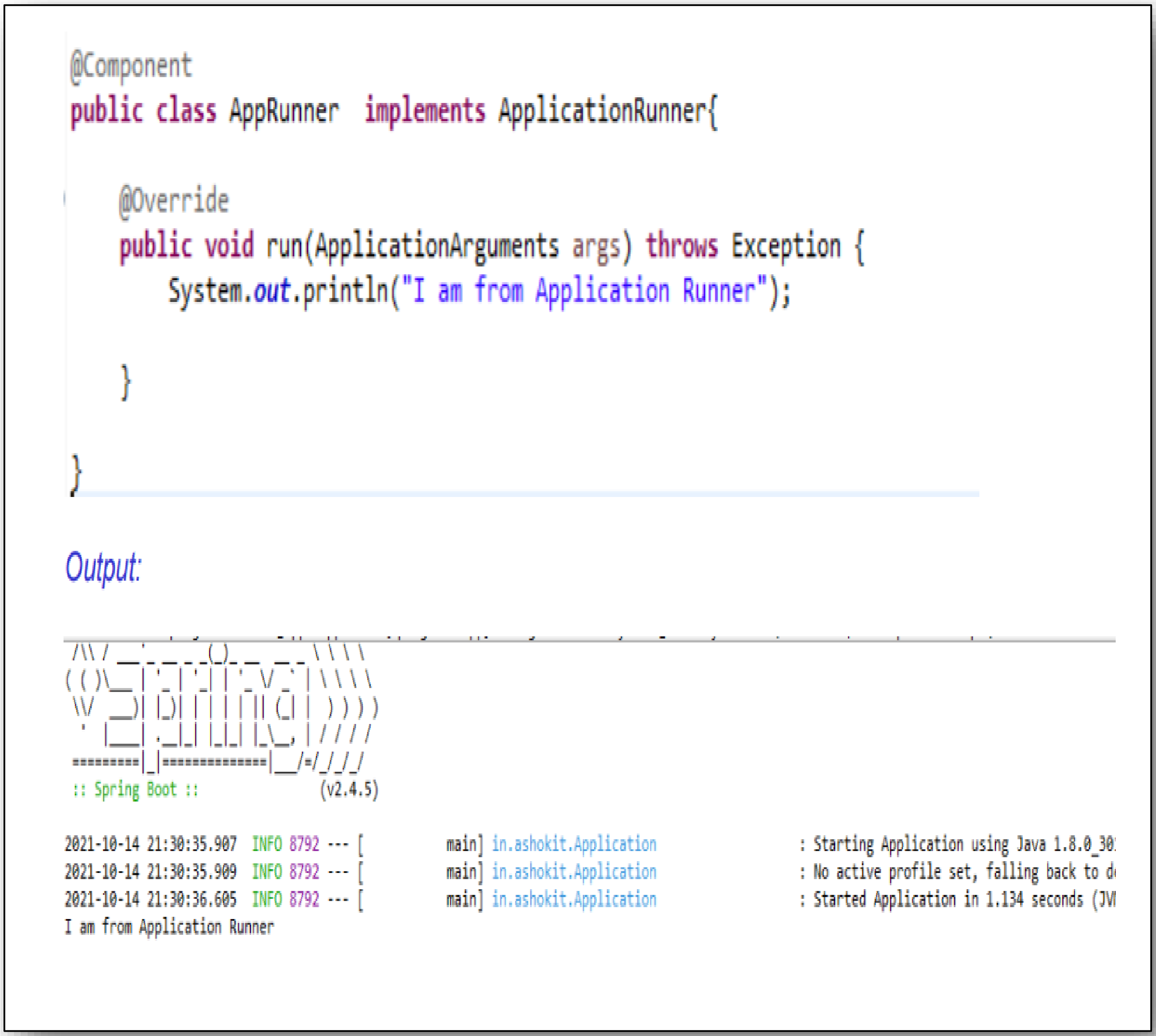
There are two types of Runner Interfaces.

- ApplicationRunner
- CommandLineRunner

38. What is ApplicationRunner in SpringBoot?

Application Runner is a functional interface which contains only one abstract method run(). When there is a need to execute some piece of code during the bootstrap of the spring boot application, then we need to write a Runner class to override the run method and provide the implementation.

Example:



```java
@Component
public class AppRunner  implements ApplicationRunner{

    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("I am from Application Runner");

    }

}
```

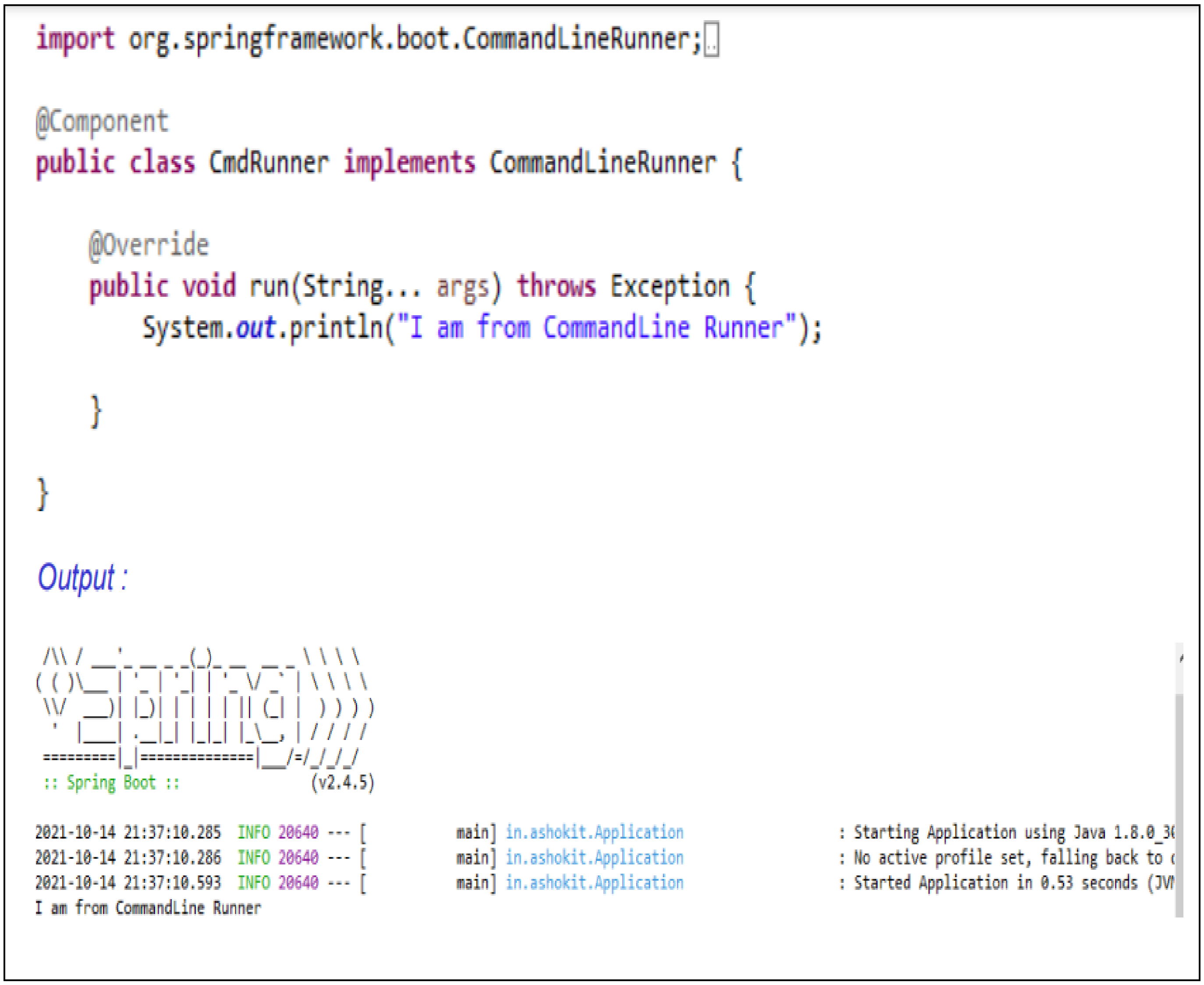
Output:

```
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.4.5)

2021-10-14 21:30:35.907  INFO 8792 --- [           main] in.ashokit.Application                   : Starting Application using Java 1.8.0_30
2021-10-14 21:30:35.909  INFO 8792 --- [           main] in.ashokit.Application                   : No active profile set, falling back to d
2021-10-14 21:30:36.605  INFO 8792 --- [           main] in.ashokit.Application                   : Started Application in 1.134 seconds (JV
I am from Application Runner
```

39.What is CommandLineRunner in SpringBoot?

CommandLineRunner is similar to the ApplicationRunner interface which is also used to execute the logic only once during application startup. The only difference between CommandLineRunner and ApplicationRunner is that ApplicationRunner accepts the arguments in the form of ApplicationArguments where as CommandLineRunner accepts in String[] array.

Example: A class implementing CommandLineRunner interface



```java
import org.springframework.boot.CommandLineRunner;

@Component
public class CmdRunner implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
        System.out.println("I am from CommandLine Runner");

    }

}
```

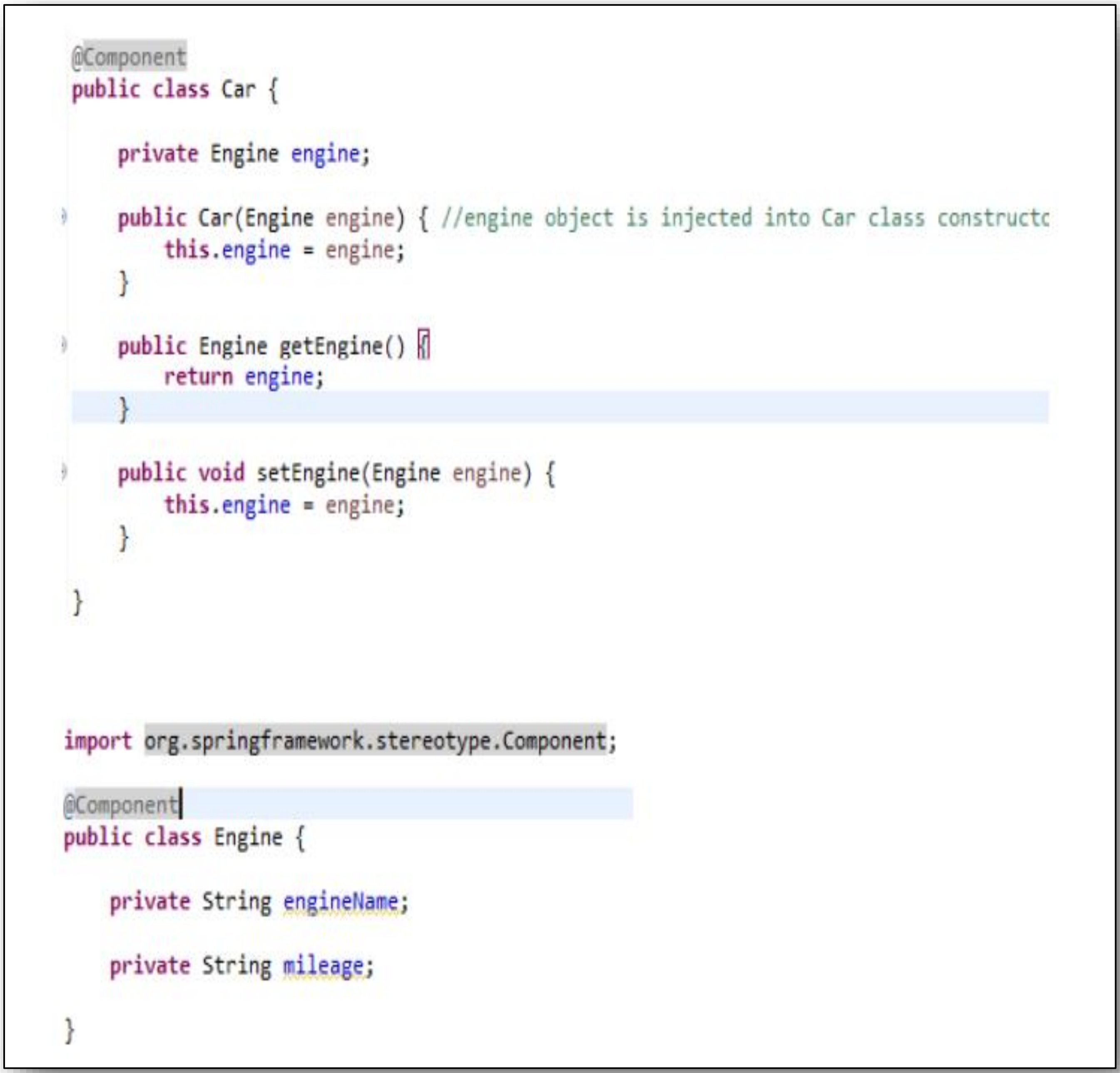
Output :

```
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.4.5)

2021-10-14 21:37:10.285  INFO 20640 --- [           main] in.ashokit.Application                   : Starting Application using Java 1.8.0_3
2021-10-14 21:37:10.286  INFO 20640 --- [           main] in.ashokit.Application                   : No active profile set, falling back to
2021-10-14 21:37:10.593  INFO 20640 --- [           main] in.ashokit.Application                   : Started Application in 0.53 seconds (JV
I am from CommandLine Runner
```

40.Explain Constructor Injection.

Constructor Injection is the process of injecting the dependent bean object into the target bean using the target class construction.

E.g. : If a class Car is dependent on the Engine object which is needed for Car to run, in this case Engine object will be created first and dependency will be injected into Car class. If the dependency is achieved using the target class constructor, it is referred to as Constructor injection.



```
@Component
public class Car {

    private Engine engine;

    public Car(Engine engine) { //engine object is injected into Car class constructc
        this.engine = engine;
    }

    public Engine getEngine() {
        return engine;
    }

    public void setEngine(Engine engine) {
        this.engine = engine;
    }

}



import org.springframework.stereotype.Component;

@Component
public class Engine {

    private String engineName;

    private String mileage;

}
```

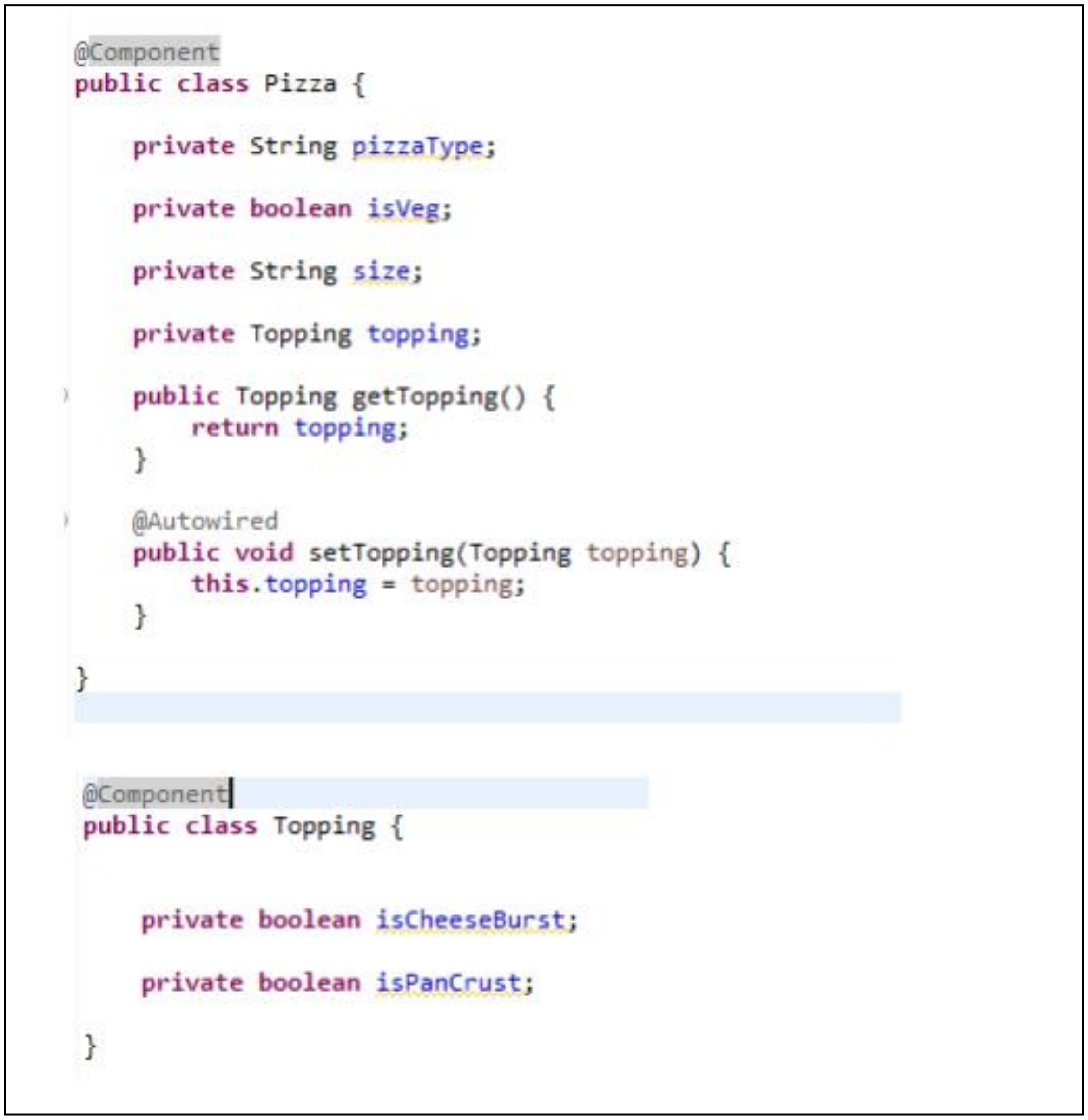
It is not mandatory to give @Autowired annotation if there is only one constructor.

41.Explain Setter Injection.

Setter injection is another mechanism to perform dependency injection. In this approach, the dependent object is injected into target class using target class setter methods. Setter injection can override the constructor injection.

@Autowired annotation is used on the setter methods.

Eg:



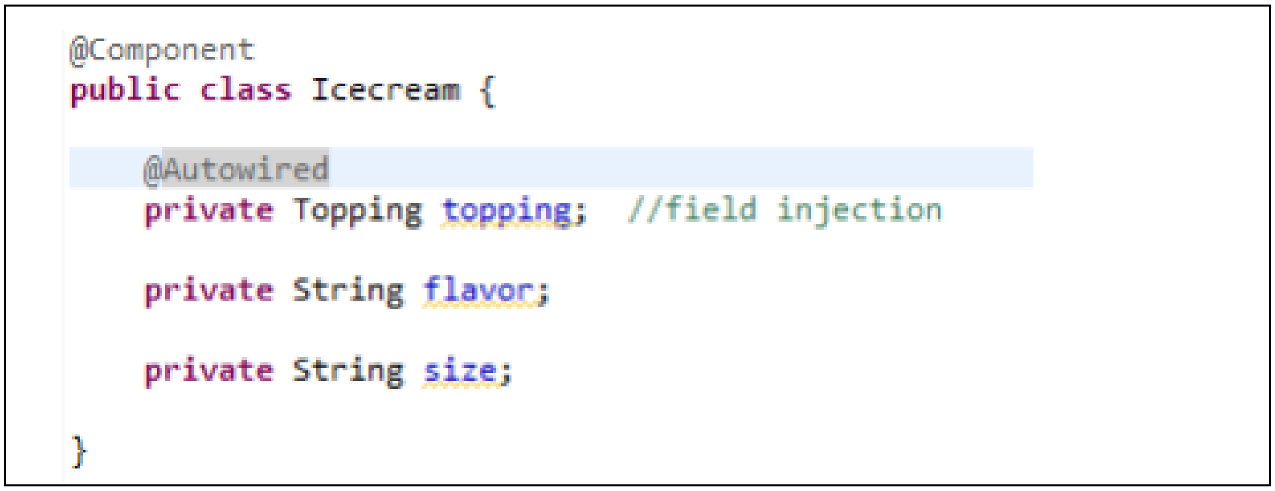
```
@Component
public class Pizza {

    private String pizzaType;

    private boolean isVeg;

    private String size;

    private Topping topping;

    public Topping getTopping() {
        return topping;
    }

    @Autowired
    public void setTopping(Topping topping) {
        this.topping = topping;
    }

}

@Component
public class Topping {


    private boolean isCheeseBurst;

    private boolean isPanCrust;

}
```

In setter injection, target class object should be created first followed by dependent object.

42.Explain Field Injection.

Field injection is a mechanism where the dependent object is injected into the target object using target class variable directly with the use of @Autowired annotation. It internally uses Reflection API to perform field injection

Eg:

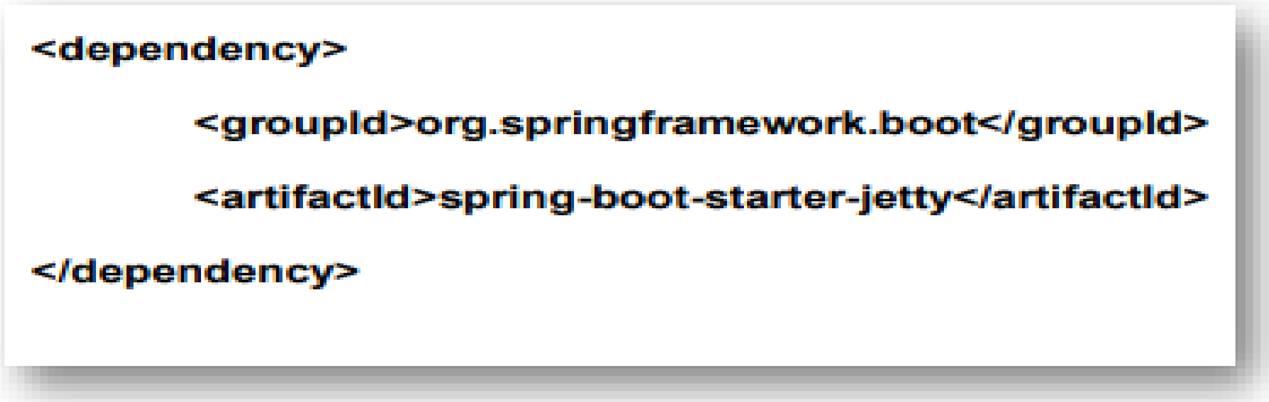


```java
@Component
public class Icecream {

    @Autowired
    private Topping topping;  //field injection

    private String flavor;

    private String size;

}
```

43. Can you override the default web server in Spring Boot.

By default, Spring Boot provides Tomcat as the embedded server. This can be changed, as we can configure Jetty, Netty as embedded servers in Spring boot. This can be done in a convenient way by adding the starter dependencies in the maven pom.xml.

Example: Adding Jetty as dependency to pom.xml



```xml
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

44.What is Spring Data JPA?

Spring data JPA is used for designing the persistence layer of the web application. It is used for managing the relational data in a java application. It acts as an intermediate between the java object and relational database.

Spring data JPA is mainly built on top of JDBC API and it helps in reducing the boilerplate code.

Spring boot provides a starter-POM “spring-boot-starter-data-jpa” which is used to design the DAO layer. All the required jars are added to the classpath after adding the starter pom in dependency management configuration file.

It provides predefined interfaces which has methods to perform CRUD operations.

45. Explain features of Spring Data JPA?

The main advantages of using Spring Data are:

No-code repositories: Spring data provides predefined repository interfaces which should be extended to create a repository for the entity class. It has the built-in methods to perform the CRUD operation.

Reduces Boilerplate code: It reduces a lot of boiler plate such as creating a connection object, creating a statement and executing the query,closing the resources,etc.

Spring data provides predefined methods which are already implemented in the Repository interfaces, and by just calling those methods, we can perform CRUD operations.

Generation of the queries: Another feature is queries are automatically generated based on the method names.

Eg: If there is a method in EmployeeRepository as:

public List<Employee> findByName(String empName);

Spring data jpa will create a query as below :

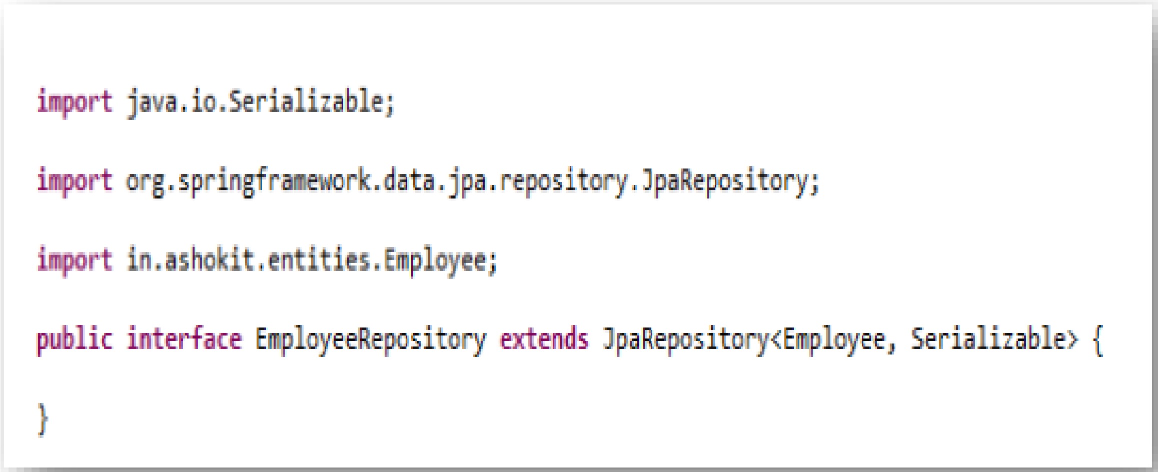
select e.empid,e.empname,e.esalary from employee e where e.name = ? ;

Pagination and Sorting support: It supports pagination and sorting using predefined interface PagingAndSortingRepository.

46. How to create a custom Repository class in Spring JPA?

We can create custom repository by extending one of the interfaces as below:

- CrudRepository
- JpaRepository
- PagingAndSortingRepository



```
import java.io.Serializable;

import org.springframework.data.jpa.repository.JpaRepository;

import in.ashokit.entities.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Serializable> {

}
```

47. Difference between CRUDRepository and JPARepository.

CrudRepository interface provides method to perform only crud operations. It allows to create , read, update and delete records without creating own methods.

JPARepository extends PagingAndSortingRepository which provides methods to retrieve records using pagination and also to sort the records.

PagingAndSortingRepository extends CrudRepository which allows to do CRUD operations.

48. What is the purpose of save () method in CrudRepository.

An entity can be saved into the database using save () method of CrudRepository. It will persist or merge the entity by using JPA Entity Manager. If the primary id is empty, it will call entityManager.persist(…) method, else it will merge the existing record by making a call to entityManager.merge(…) method.

49. Difference between findById() and getOne().

The findById() method is available in CrudRepository while getOne() is available in JpaRepository.

The findById() returns null if record does not exist while the getOne() will throw an exception called EntityNotFoundException.

getOne() is a lazy operation where it will return a proxy without even hitting the database.

findById() – will retrieve the row record by directly hitting the database.

50. Explain @Transactional annotation in Spring.

A database transaction is a sequence of statements/actions which are treated as a single unit of work. These operations should execute completely without any exception or should show no changes at all. The method on which the @Transactional annotation is declared, should execute the statements sequentially and if any error occurs, the transaction should be rolled back to its previous state. If there is no error, all the operations need to be committed to the database. By using @Transactional, we can comply with ACID principles.

E.g.: If in a transaction, we are saving entity1, entity2 and entity3 and if any exception occurs while saving entity3, then as enitiy1 and entity2 comes in same transaction so entity1 and entity2 should be rolledback with entity3.

A transaction is mainly implied on non-select operations (INSERT/UPDATE/DELETE).

51. What is the use of @EnableJpaRepositories method?

If the repositories classes belong to the sub package of the Spring Boot Main class, then @SpringBootApplication is enough as it scans the package using @EnableAutoConfiguration. If the repository classes are not part of the sub package of the Main class, in that case, it will not scan the repository classes, we need to use @EnableJpaRepositories. This needs to be provided in Configuration class or SpringBootApplication class.

52. .Explain few CrudRepository methods.

Some of the methods to perform DML operations are :

- findById – to retrieve record based on the primary key.
- findAll – to retrieve all records from the database.
- existsById – to check if the record exists by passing primary key
- count – to check the total number of records.
- Save – to insert a record into the database
- deleteById – to delete a record using primaryKey

- deleteAll – to delete all records from the table

53. Difference between delete () and deleteInBatch() methods.

- delete() – It is used to delete a single record at a time. It internally uses remove method of entitymanager.
- deleteInBatch() – it can delete multiple records at a time, it internally calls executeUpdate() method.It is much faster than delete method.

54. What is the use of @Modifying annotation?

It indicates that a query method should be considered as a modifying query. It can be implied only on non-select queries (INSERT, UPDATE, DELETE). This annotation can be used only on the query methods which are defined by @Query annotation.

55. What is Spring MVC?

Spring MVC is one of the modules in Spring framework which helps in building web and distributed applications. It supports two design patterns

a. MVC Design Pattern

b. Front Controller Design Pattern

MVC stands for Model, View and Controller.

The major role is played by DispatcherServlet in Spring MVC which acts as a front controller which receives the incoming request and maps it to the right resource.

The main advantage of Spring MVC is that it helps in the separation of the presentation and business layer.

The components of Spring MVC are:

- Model – A model represents the data which can be an object or a group of objects.
- View – A view represents an UI to display the data. It can be a JSP, or a Thymeleaf page.
- Controller – It acts as an intermediate between model and view components and is responsible to handle the incoming requests.
- Front Controller – Dispatcher servlet serves the main purpose of redirecting the request to the respective controller methods.

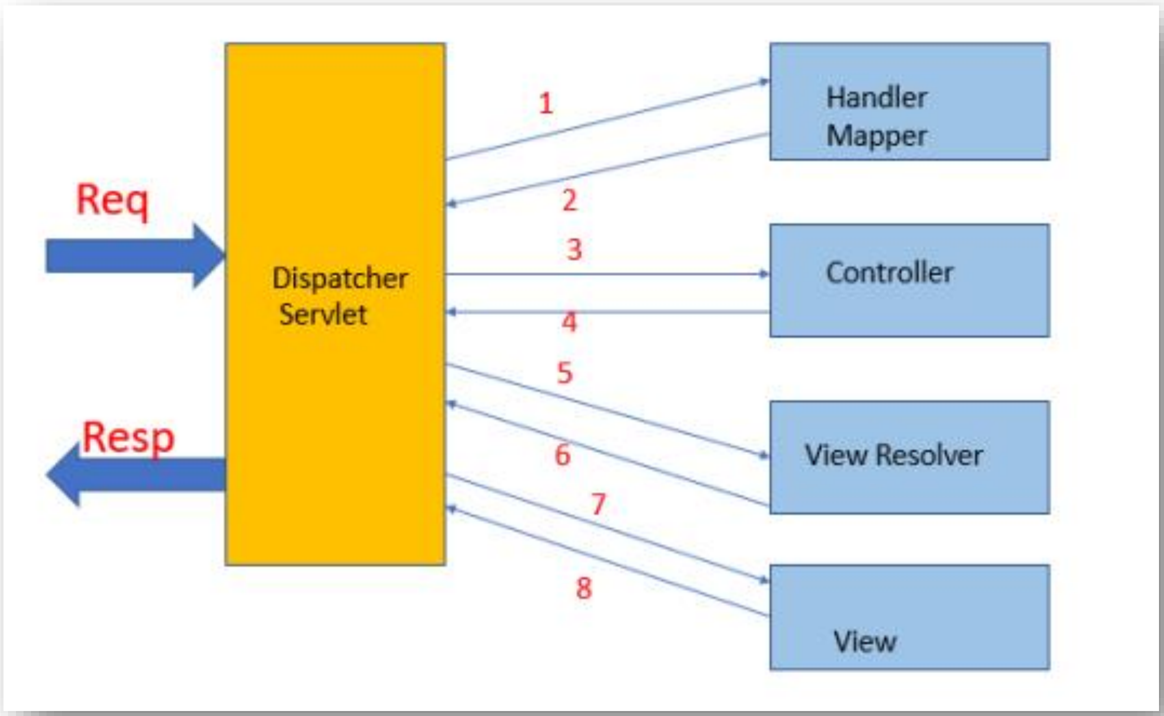
56. Explain the flow of Spring MVC.

When a client request comes in, it is intercepted by the Dispatcher Servlet which acts as a Front Controller. The dispatcher servlet is responsible for pre – processing of the request and calls the handler methods to decide which controller should handle the request. It uses

BeanNameUrlHandlerMapping and SimpleUrlHandlerMapping to map the request to the corresponding controller method.

The controller then processes the request and sends the response as ModelAndView back to the DispatcherServlet. Model represents the data to be displayed and view represents the component in which data should be rendered on the browser.

Front Controller is also responsible in manipulating the response data(post-processing) before sending back to client.

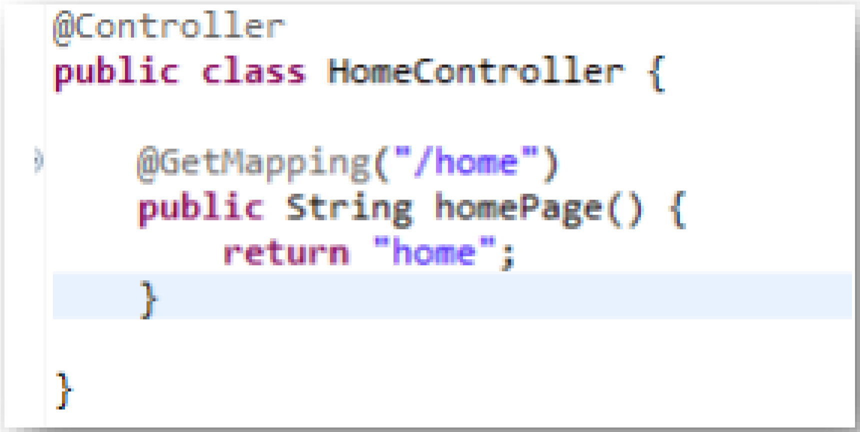


57. What is Dispatcher Servlet.

Dispatcher Servlet acts as a central servlet which handles all the incoming HTTP Requests and Responses. Once a client request is sent, it is received by the DispatcherServlet and it forwards the request to handler mapper to identify the corresponding controller class to handle the request. The controller performs all the business logic and hands over the response back to DispatcherServlet. The servlet then prepares the view component by looking for the view resolver in properties file and sends the data to be rendered on view page.

58. What is the use of @Controller annotation.

A class can be represented as a Controller class which is used to handle one or more HTTP requests. It is represented as a controller class using @Controller annotation. It is one of the stereotype annotations.



```java
@Controller
public class HomeController {

    @GetMapping("/home")
    public String homePage() {
        return "home";
    }

}
```

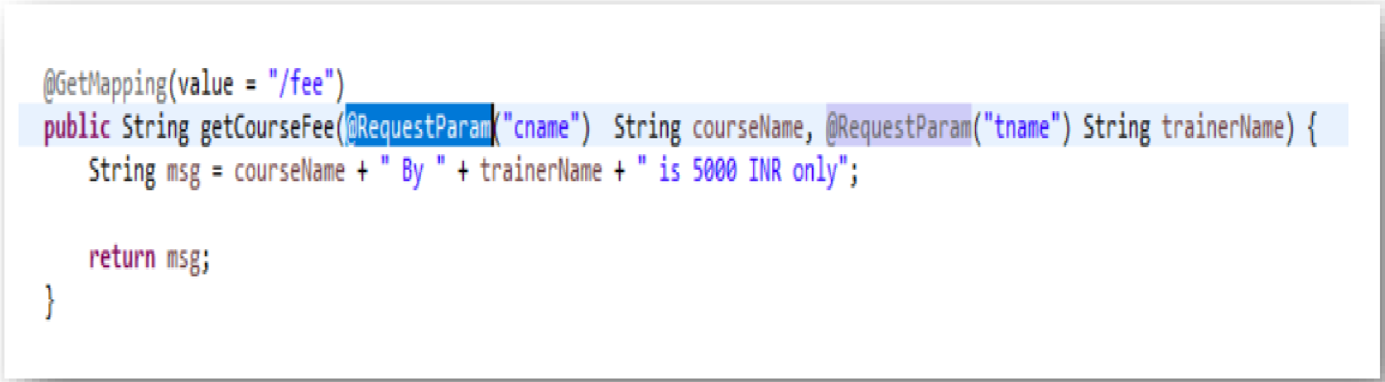
In the example above, whenever a client sends a request with the url as localhost:8090/home, the homepage method is invoked and view is returned from the method.

59. . Difference between @RequestParam and @PathVariable.

@RequestParam is used to access the parameter values which are passed as part of request URL.

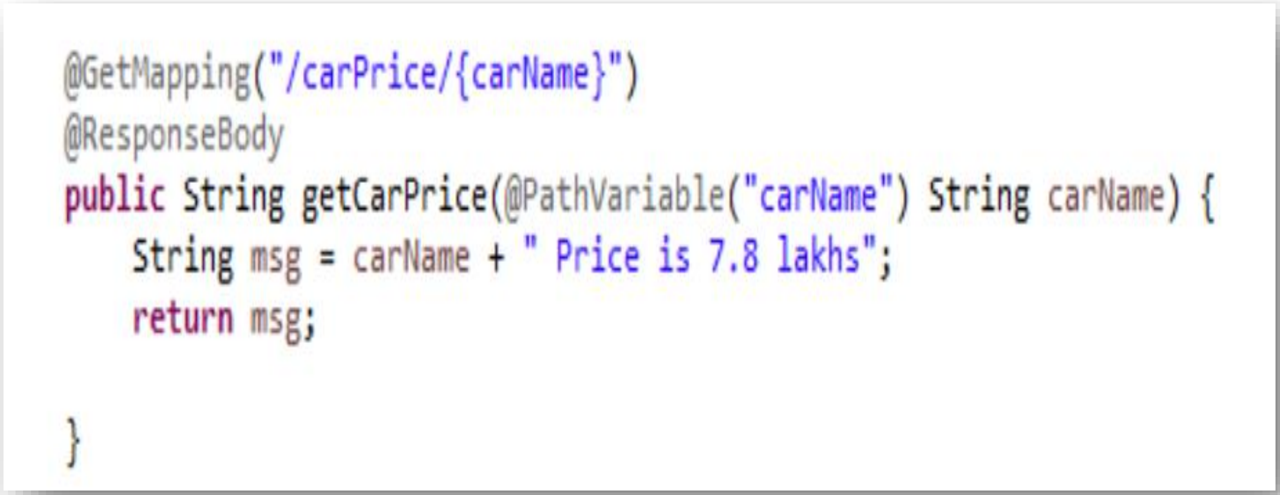
URL: http://localhost:8090/fee?cname=SBMS&tname=Savitha

In the above example, we can access the parameter values of courseName and trainerName using @RequestParam annotation. In case of @RequestParam, if the parameter value is empty , it can take default value using attribute defaultValue=XXXXX



```java
@GetMapping(value = "/fee")
public String getCourseFee(@RequestParam("cname")  String courseName, @RequestParam("tname") String trainerName) {
    String msg = courseName + " By " + trainerName + " is 5000 INR only";

    return msg;
}
```

@PathVariable – It is used to extract data from the request URI. Eg : if the URL is as : http://localhost:8090/carPrice/{carName} – the value for the placeholder {carName} can be accessed using @PathVariable annotation. In order to access the carName, we need to write code as below:



```
@GetMapping("/carPrice/{carName}")
@ResponseBody
public String getCarPrice(@PathVariable("carName") String carName) {
    String msg = carName + " Price is 7.8 lakhs";
    return msg;

}
```

60. Explain @Service and @Repository annotations.

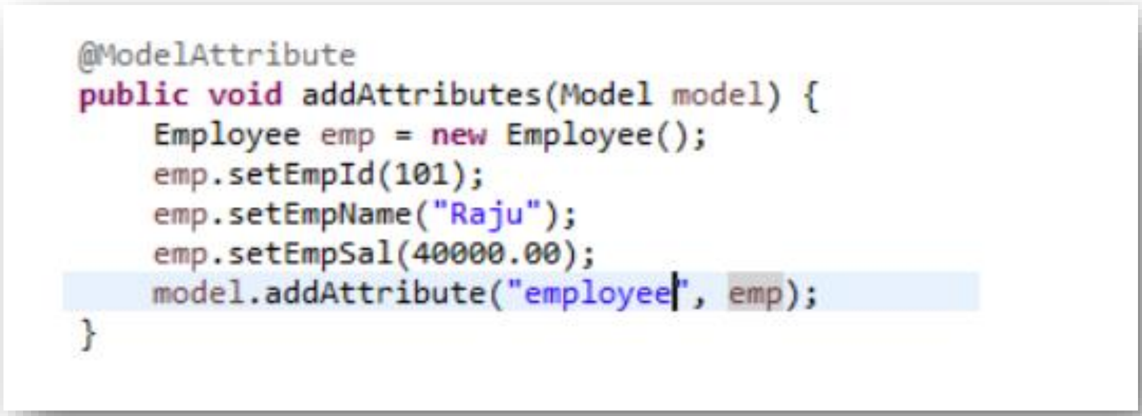
A class which is annotated with **@Repository** annotation is where the data is stored. It is a stereotype annotation for the persistence layer.

@Service – It indicates that a java class contains the business logic. It is also a part of @Component stereotype annotation.

61. What is the purpose of @Model Attribute?

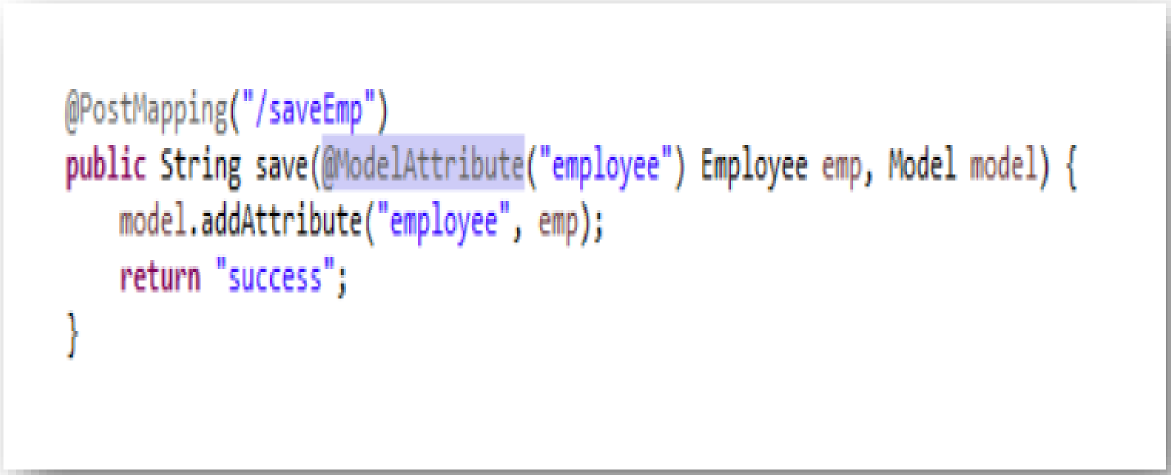
It is part of Spring MVC module and can be used in two scenarios:

@ModelAttribute at method level: When used at method level, it indicates that a method will return one or more model attributes.



```java
@ModelAttribute
public void addAttributes(Model model) {
    Employee emp = new Employee();
    emp.setEmpId(101);
    emp.setEmpName("Raju");
    emp.setEmpSal(40000.00);
    model.addAttribute("employee", emp);
}
```

@ModelAttribute at method argument: When it is used at method argument, it indicates that the argument should be retrieved from the form data. It binds the form data to the bean object.



```java
@PostMapping("/saveEmp")
public String save(@ModelAttribute("employee") Employee emp, Model model) {
    model.addAttribute("employee", emp);
    return "success";
}
```

62. How does Spring MVC support for validation.

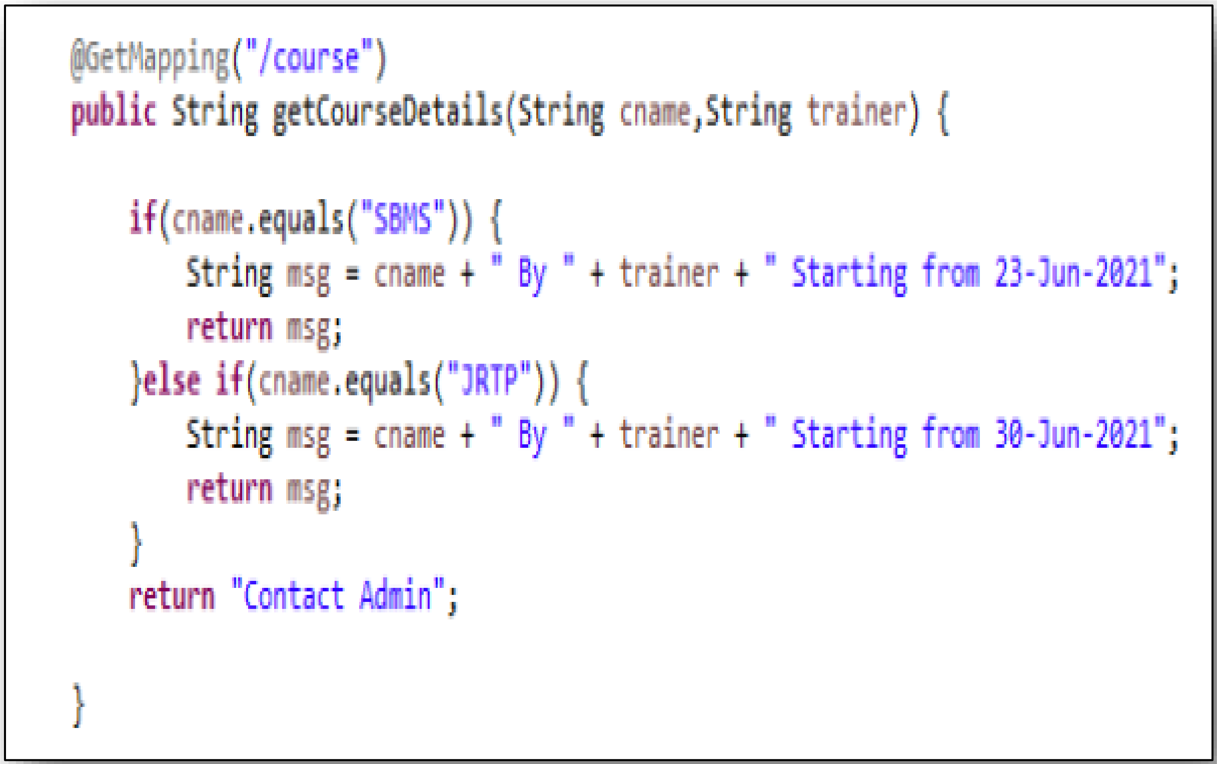
It is used to restrict the user input provided by the user. Spring provides the validation API where the BindingResult class is used to capture the errors raised while validating the form.

We need to add spring-boot-starter-validation in pom.xml.

63. Explain @GetMapping and @PostMapping.

@GetMapping is an alternative for @RequestMapping (method = RequestMethod.GET)

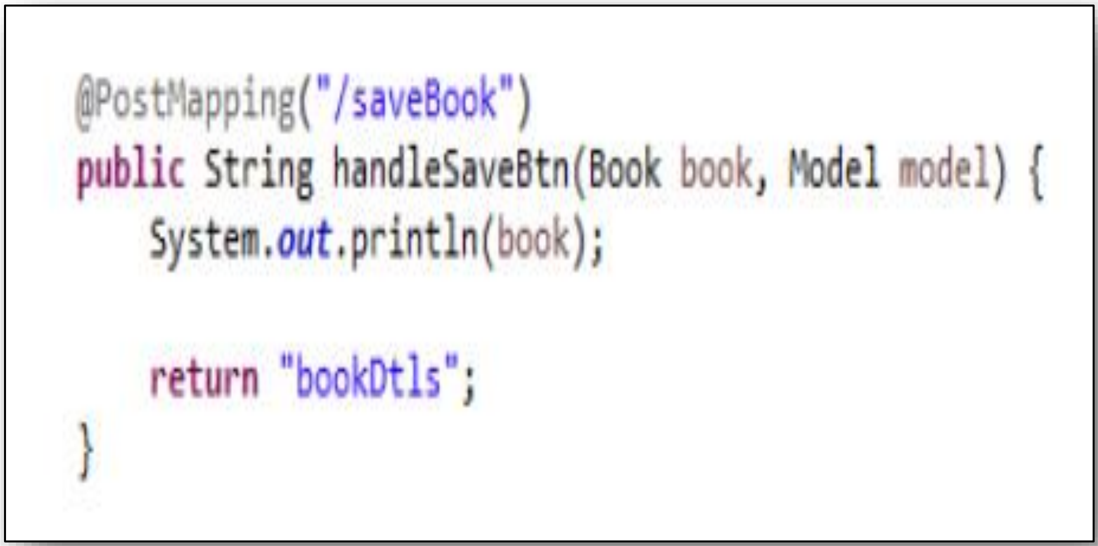
It handles the HTTP Get methods matching with the given URI.



```
@GetMapping("/course")
public String getCourseDetails(String cname,String trainer) {

    if(cname.equals("SBMS")) {
        String msg = cname + " By " + trainer + " Starting from 23-Jun-2021";
        return msg;
    }else if(cname.equals("JRTP")) {
        String msg = cname + " By " + trainer + " Starting from 30-Jun-2021";
        return msg;
    }
    return "Contact Admin";

}
```

@ PostMapping is an alternative for @RequestMapping (method = RequestMethod.POST)

It handles the HTTP Post methods matching with the given URI.



```
@PostMapping("/saveBook")
public String handleSaveBtn(Book book, Model model) {
    System.out.println(book);

    return "bookDtls";
}
```

64.Difference between @Controller and @RestController annotations.

A class which is annotated with **@Controller** indicates that it is a controller class which is responsible to handle the requests and forwards the request to perform business logic. It returns a view which is then resolved by ViewResolver after performing business operations.

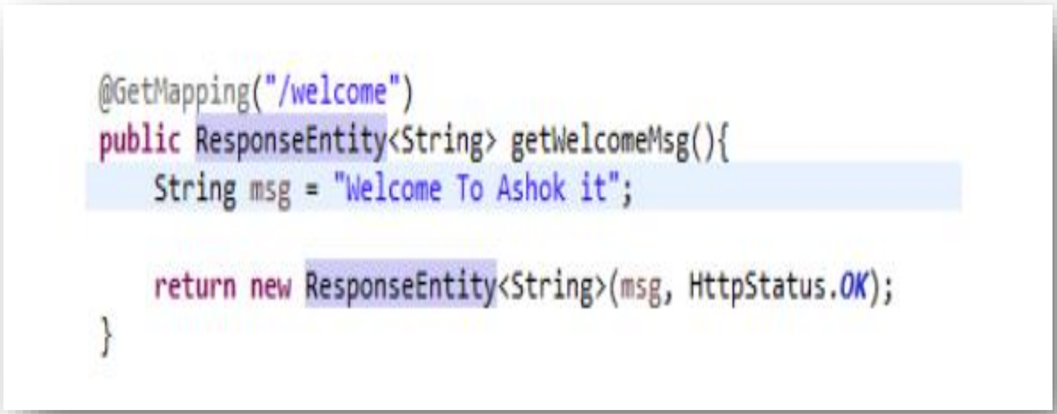
@RestController is used in REST Webservices and is combination of @Controller and @ResponseBody.It returns the object data directly to HTTP Response as a JSON or XML.

65. Explain the use of @ResponseEntity annotation.

@ResponseEntity is used to represent the entire HTTP response such as status code, headers and response body.

We can return the entire response from the endpoint. When using @ResponseBody , it returns the value into the body of the http response.

Example:



```
@GetMapping("/welcome")
public ResponseEntity<String> getWelcomeMsg(){
    String msg = "Welcome To Ashok it";

    return new ResponseEntity<String>(msg, HttpStatus.OK);
}
```

66. How to handle exceptions in Spring MVC.

In Spring Boot, exceptions can be handled using below two annotations:

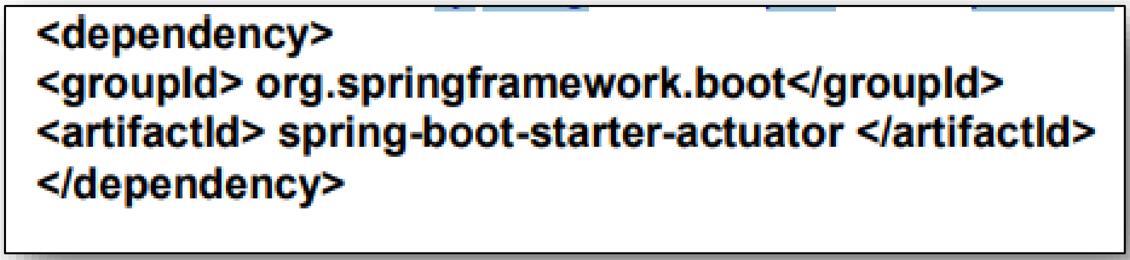
- @ExceptionHandler -specific to a controller class
- @ControllerAdvice – common to all controllers.

67. Explain @ControllerAdvice in Spring Boot.

@ControllerAdvice is a specialization of @Component annotation which is used to handle the exceptions across the whole application by providing a global code which can be applied to multiple controllers.

68.How will you enable actuator in spring boot application.

An actuator can be enabled by adding the starter pom into the pom.xml.



```
<dependency>
<groupId> org.springframework.boot</groupId>
<artifactId> spring-boot-starter-actuator </artifactId>
</dependency>
```

69. What are the actuator endpoints which are needed to monitor the application.

Actuators provide below pre-defined endpoints to monitor our application.

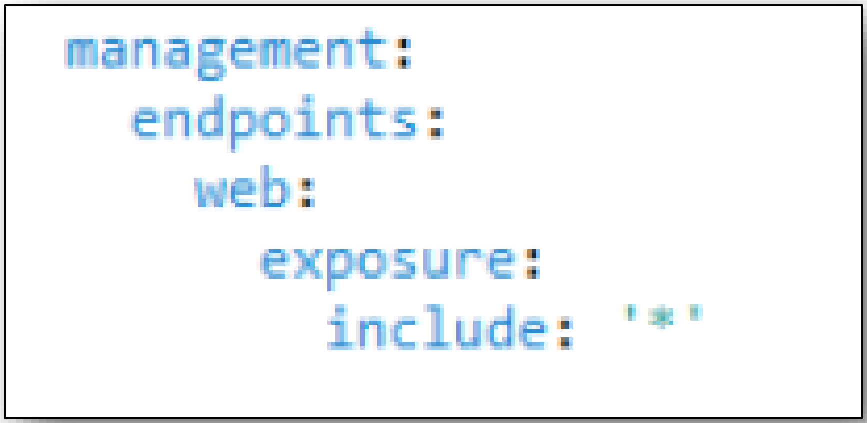
- Health
- Info
- Beans
- Mappings
- Configprops
- Httptrace
- Heapdump
- Threaddump
- Shutdown

70. How to get the list of beans available in spring application.

Spring Boot Actuator provides an endpoint url /beans to load all the spring beans of the application.

71. How to enable all endpoints in actuator?

In order to expose all endpoints of actuator, we need to configure it in application properties/yml file as:


```yaml
management:
  endpoints:
    web:
      exposure:
        include: '*'
```

72. What is a shutdown in the actuator?

A shutdown is an endpoint that helps application to shut down properly. This feature is not enabled by default. We can enable it by giving the below command in properties file.

management.endpoint.shutdown.enabled=true

73. . What is Spring Security?

Spring security is a powerful access control framework. It aims at providing authentication and authorization to java applications. It enables the developer to impose security restrictions to save from common attacks.

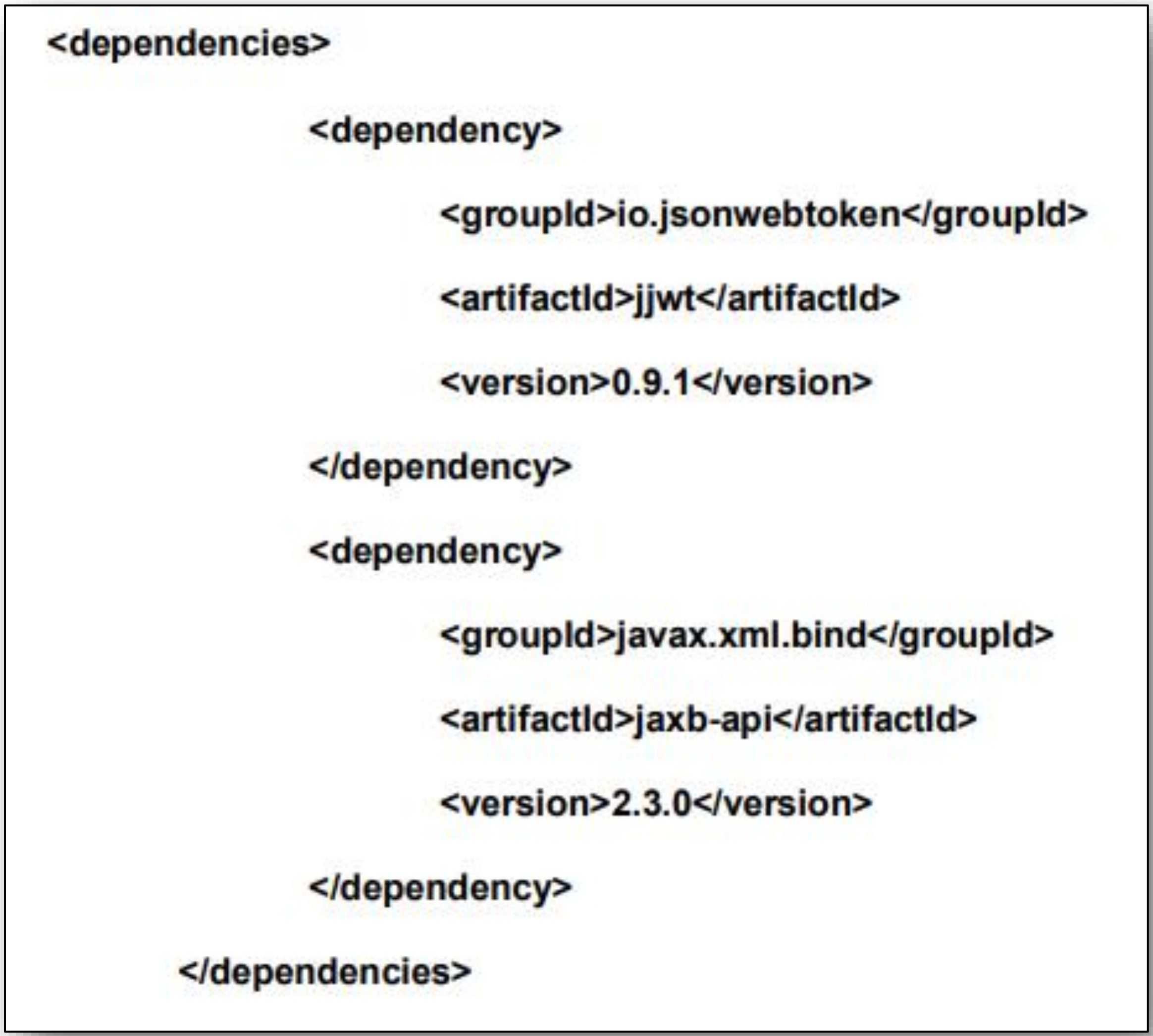
74. What are the features of Spring Security?

Spring security provides many features as below:

- Authentication and Authorization.
- Supports Basic and Digest Authentication.
- Supports CSRF Implementation.
- Supports Single Sign-on.

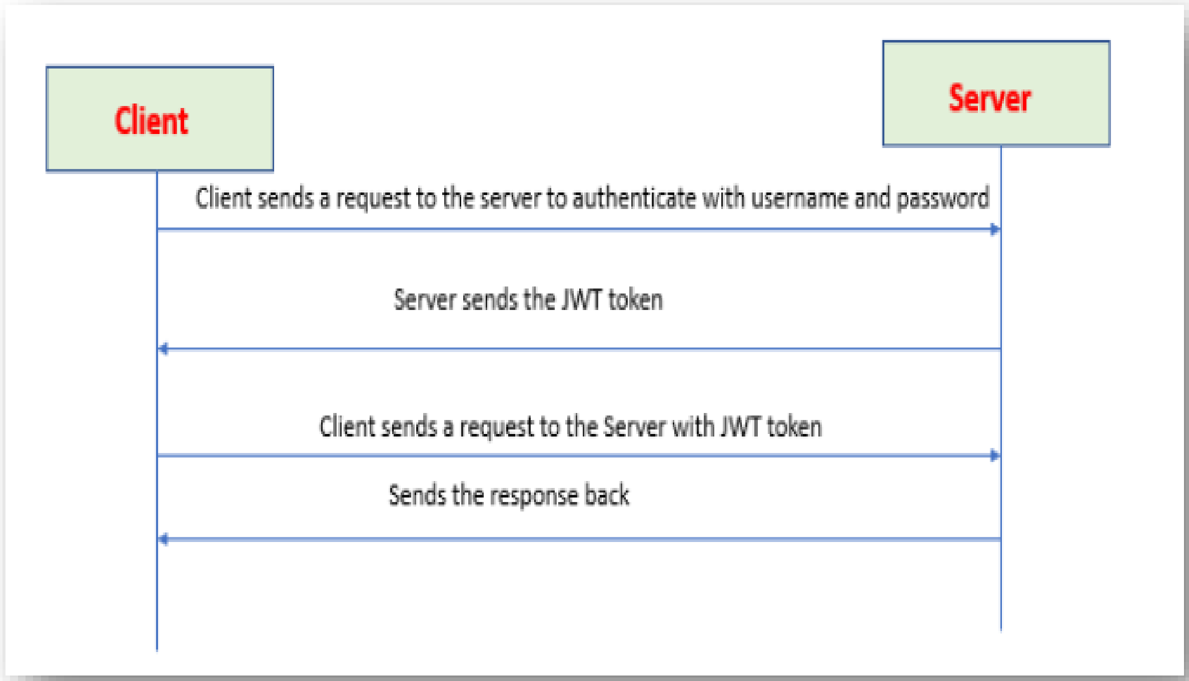
75. How to implement JWT?

JWT stands for Json Web Token which helps in implementing token-based security. Token is generated using the secret key. We need to add below dependency in pom.xml.



```
<dependencies>
                <dependency>
                        <groupId>io.jsonwebtoken</groupId>
                        <artifactId>jjwt</artifactId>
                        <version>0.9.1</version>
                </dependency>
                <dependency>
                        <groupId>javax.xml.bind</groupId>
                        <artifactId>jaxb-api</artifactId>
                        <version>2.3.0</version>
                </dependency>
        </dependencies>
```

The following diagram depicts the working of JWT.


Client
Server
Client sends a request to the server to authenticate with username and password
Server sends the JWT token
Client sends a request to the Server with JWT token
Sends the response back

76. What is DelegatingFilterProxy in Spring Security.

It is a predefined filter class which helps in performing pre-processing of the request.It supports for both authentication and authorization. It is a proxy servlet filter which acts as an intermediator before redirecting the request to dispatcher servlet

77. What is Spring Security OAuth2.

OAuth2 is an authorization framework, granting clients access to protected resources via authorization server. It allows end user's account information to be used by third party services(eg.facebook) without exposing user's password.

The oAuth token are random strings generated by the authorization server.

There are 2 types of token.

Access token – It is sent with each request, usually valid for about an hour only.

Refresh token – It is used to get a new access token, not sent with each request. It lives longer than access token.

78. What is the advantage of using JWT Token?

Advantages of using JWT Token:

- The jwt token has authentication details and expire time information.
- It is one of the approaches to secure the application data because the parties which are interacting are digitally signed.
- It is very small token and is better than SAML token.
- It is used at internet scale level, so it is very easy to process on user's device.

79.What is authentication?

Authentication is the mechanism to identify whether user can access the application or not.

80.What is authorization?

Authorization is the process to know what the user can access inside the application and what it cannot i.e which functionality it can access and which it cannot.

.81. What is filter Chain Proxy?

The filter Chain Proxy contains multiple security filter chains and a task is delegated to the filter chain based on the URI mapping. It is not executed directly but started by DelegatingFilterProxy.

.82. What is security context in Spring Security.

It is used to store the details of the current authenticated user, which is known as principle. So if we want to get the current username, we need to get the SecurityContext.

83. Difference between has Authority and hasRole?

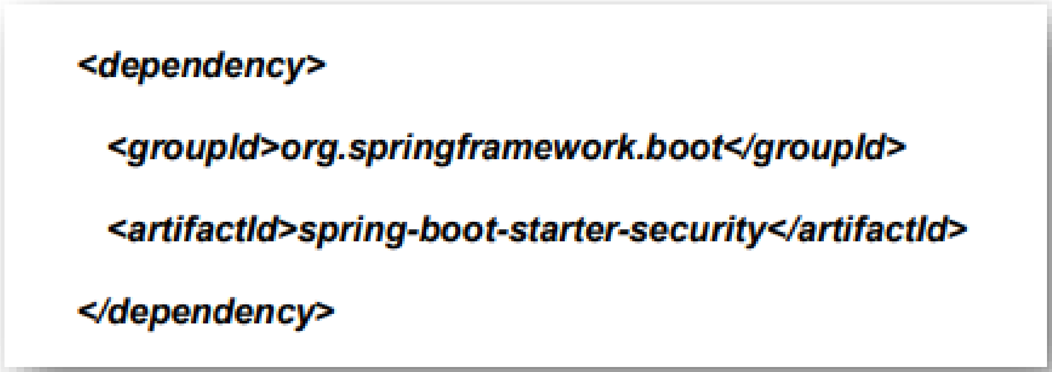
hasRole – defines the role of the user.It does not use the ROLE_prefix but it will automatically added by spring security as hasRole(ADMIN);

has Authority – defines the rights of the user. It uses ROLE prefix while using has Authority method as has Authority (ROLE_ADMIN)

84. How to enable spring boot security in spring boot project?

If Spring boot Security dependency is added on class path, it automatically adds basic authentication to all endpoints. The Endpoint "/" and "/home" does not require any authentication. All other Endpoints require authentication.

The following dependency needs to be added.



```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

85.What is Basic Authentication?

In Basic Authentication, we send username and password as part of the request to allow user to access the resource. The user credentials are sent as authorization request headers. This approach is used to authenticate the client requests.

86.What is Digest Authentication?

Digest Authentication is more preferable when compared to basic authentication as the credentials are in encrypted format by applying hash function to username,password,etc. It does not send the actual password to the server. Spring security provides digest authentication filter to authenticate the user using digest authentication header.

87. How to get current logged in user in spring security.

We can get the current logged in user by using the following code snippet.

User user = (User)SecurityContextHolder.getContext().getAuthentication().getPrincipal();

String name = user.getUsername();

88.What is SSL and its use?

SSL stands for secure socket layer which is an encryption- based internet security protocol.

It is mainly used to secure client information (such as credit card number /password/ssn) to a web server.

SSL provides a secure channel between two machines or devices running over the internet. A common example is when SSL is used to secure communication between a web browser and a web server. This changes the address of the website from HTTP to HTTPS, basically 'S' stands for 'Secure'.

89. What is salting?

Salting is used to generate random bytes that is hashed along with the password. Using salt , we can add extra string to password , so that hackers finds difficult to break the password. The salt is stored as it is, and need not be protected.

90.What is hashing in spring security.

Hashing is an approach where a string is converted into encoded format using hashing algorithm. The hashing algorithm takes input as password and returns hashed string as output. This hashed data is stored in the database instead of plain text which is easily vulnerable to hacker attacks.

91.How to secure passwords in a web application?

Generally, passwords should not be stored as plain text into the storage as it can easily be accessed by the hackers.

We need to use encryption techniques before storing the password.

We need to use hashing and salting techniques to prevent from security breaches.

92.What is AuthenticationManager in spring security?

Authentication manager is the interface that provides the authentication mechanism for any object. The most common implementation of it is the AuthenticationProvider.

If the principal of the input is valid , and authenticated , it returns an authentication instance if successful. It checks whether the username and password is authenticated to access a specific resource.

93. What are the various ways to implement security in spring boot project?

There are 3 ways to implement security.

a) In Memory Credential Security
b) Using Database
c) Using UserDetailsService

In Memory Credential – In this mechanism, we configure the user credentials in the application itself, and use it when there is a request to validate the user.

Using JDBC Credentials – Here, the user credentials are stored into the database and when the client request comes, it is validated against it.

Using UserDetailsService – It is an interface provided by Spring framework. After entering the username in the form and clicking on Login button invokes a call to this service. It locates the user based on the username provided. It contains a method loadUserByUsername(String username) which returns UserDetails object.

94. What are the essential components of Spring Boot?

Some of the components of Spring Boot are:

- Spring Boot Starter
- Spring Boot autoconfiguration
- Spring Boot Actuator
- Spring Boot CLI

95. What is the use of profiles in Spring Boot?

Spring Profiles provide a way to segregate parts of your application configuration and make it only available in certain environments. For eg, if we want to enable swagger configuration only in QA environment, it can be done using spring profiles.

96. How can you set active profile in Spring Boot.

We can set the active profile by using configuration properties as:
spring.profiles.active=production

97. What is AOP?

Aspect Oriented Programming aims at separating the cross-cutting logics from the main business logic.

98. What is YAML?

YAML is mainly used for configuration purpose. It is similar to properties file and provides more readability.

99. Use of @Profile annotation.

The @Profile annotation indicates that a component is eligible for registration when the specified profile is active. The default profile is called default, all the beans that do not have a profile set belong to this profile.

100. How to get current profile in Spring Boot.

String[] activeProfiles = environment. getActiveProfiles();

100. How does Spring Boot handle dependency management?

Spring Boot simplifies dependency management by providing a curated set of starter dependencies. These starter dependencies are opinionated, preconfigured dependencies that include all the necessary libraries and configurations for specific functionalities or technologies. By including a starter dependency, developers can easily leverage the features and capabilities provided by that technology without worrying about manually managing the dependencies.

Spring Boot manages dependencies through its dependency management plugin, which ensures that all dependencies within a specific version range are compatible with each other. Developers can simply declare the starters they need in their project's build configuration (e.g., Maven or Gradle), and Spring Boot will resolve and manage the transitive dependencies automatically.

101. What is the purpose of the application.properties or application.yml file?

The **application.properties** or **application.yml** file is used in Spring Boot for externalized configuration. It provides a convenient way to configure various properties of the application without modifying the source code. These properties 100+ Spring Boot Questions 4 include database connection details, server port, logging configuration, application-specific settings, and more.

The **application.properties** file uses a key-value format, while **application.yml** uses YAML (YAML Ain't Markup Language) format. Both formats allow developers to specify configuration properties and their corresponding values. Spring Boot automatically loads and applies the properties defined in these files during application startup. It provides sensible default values, and developers can override these defaults by specifying the properties in the **application.properties** or **application.yml** file.

102. How can you override the default properties in Spring Boot?

To override default properties in Spring Boot, you can provide custom values in the **application.properties** or **application.yml** file. By specifying the properties in these files, Spring Boot will use the custom values instead of the default values.

When defining custom properties, you need to use the same property keys as the ones used in the default configuration. By setting a property with the same key in the custom configuration file, Spring Boot will override the default value with the custom value.

Additionally, you can also override properties using command-line arguments or environment variables. By providing a command-line argument or setting an environment variable with the same key as a property, you can override the default value during application startup.

The precedence order for property resolution in Spring Boot is: command-line arguments or environment variables > **application.properties** or **application.yml** > default values.

103. What is the purpose of the @Autowired annotation in Spring Boot?

The **@Autowired** annotation in Spring Boot is used for automatic dependency injection. It allows Spring to automatically wire and inject dependencies into a class or a bean without the need for manual configuration.

By using **@Autowired** , you can declare a dependency within a class, and Spring will resolve and provide the appropriate instance of that dependency at runtime. It enables loose coupling and promotes the use of interfaces, making the code more maintainable and testable.

The **@Autowired** annotation can be used with constructor injection, setter injection, or field injection, depending on the preference and design of the application. When applied, Spring Boot will scan the classpath for suitable dependencies and inject them accordingly

104. What is the purpose of the @RestController annotation?

The **@RestController** annotation is used in Spring Boot to indicate that a class is a specialized version of the @Controller annotation, specifically for building RESTful web services. It combines the @Controller and @ResponseBody annotations into a single, convenient annotation.

When you annotate a class with @RestController , Spring Boot treats all the handler methods within that class as being responsible for generating the response body for RESTful requests. It eliminates the need to annotate individual methods with @ResponseBody because @RestController implicitly adds it to all the methods.

The **@RestController** annotation simplifies the development of RESTful web services by eliminating the need to explicitly annotate every method with @ResponseBody and makes it more concise and expressive.

105. Explain the difference between @Component , @Service , and @Repository annotations in Spring Boot:

In Spring Boot, @Component , @Service , and @Repository are three stereotype annotations used to categorize and define the roles of classes within an application.

@Component is a generic stereotype annotation that indicates a class is a component or a bean. It is the base annotation for all other stereotypes. It is typically used for classes that don't fall into the specific categories of @Service or @Repository .

@Service is a specialization of @Component and represents a service layer component. It is typically used to annotate classes that provide business logic, perform operations, or coordinate activities within the application. Services are often used to encapsulate complex operations and make them available to other parts of the application.

@Repository is also a specialization of @Component and represents a repository or data access component. It is commonly used for classes that interact with the database or any other data storage mechanism. Repositories typically provide CRUD (Create, Read, Update, Delete) operations for accessing and managing persistent data.

The main difference between these annotations lies in their intended roles and responsibilities. While all three are essentially used to define Spring beans, using the appropriate annotation helps convey the purpose and intent of the class to other developers. It also allows for potential optimizations and clarity in the codebase.

106. Explain the concept of Bean scopes in Spring Boot.

Bean scopes in Spring Boot define the lifecycle and visibility of Spring-managed beans. They determine how instances of a particular bean are created, managed, and shared within the application context.

Spring Boot provides several bean scopes, including:

- **Singleton:** The default scope. A singleton bean is created only once, and the same instance is shared across all requests. It remains in the application context until the application shuts down.
- **Prototype:** A new instance of the prototype bean is created each time it is requested. It allows multiple instances of the same bean to coexist within the application.
- **Request:** A new instance of the request-scoped bean is created for each HTTP request. It is useful when you want to store data specific to a single request, such as user session information.
- **Session:** A new instance of the session-scoped bean is created for each user session. It maintains the state specific to a user's session across multiple requests.
- **Application**: A new instance of the application-scoped bean is created once for the entire application. It is shared across all requests and sessions.
- **WebSocket**: A new instance of the web socket-scoped bean is created for each WebSocket session.
- By selecting the appropriate bean scope, you can control the lifecycle and visibility of beans, ensuring that they are managed and shared according to your application's requirements.

107. What is the purpose of the `@Value` annotation in Spring Boot?

The **@Value** annotation in Spring Boot is used to inject values from properties files, environment variables, command-line arguments, or other sources into Spring-managed beans.

By annotating a field, setter method, or constructor parameter with **@Value** , you can specify the placeholder expression to retrieve the value from the configured sources.

For example, you can use **@Value("${my.property}")** to inject the value of the my.property property defined in the **application.properties** or **application.yml** file. The @Value annotation is versatile and can be used in various scenarios, such as injecting simple values, configuring object properties, setting default values, or retrieving values from different sources. It allows you to externalize and configure properties for your application in a flexible and dynamic manner.

108. How can you enable HTTPS in a Spring Boot application?

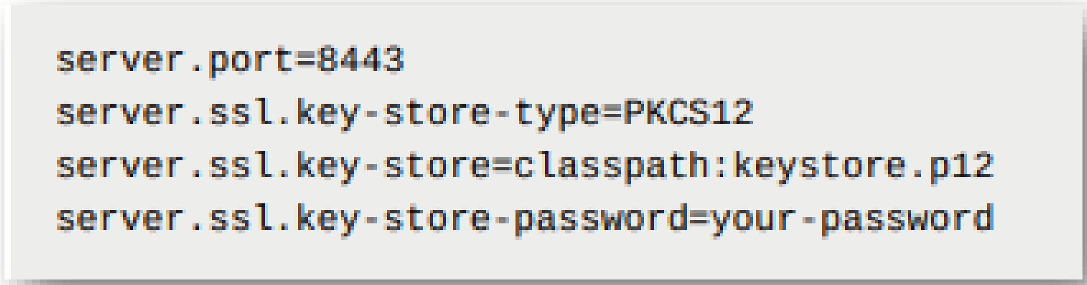
To enable HTTPS in a Spring Boot application, you need to configure an SSL/TLS certificate and update the application's configuration.

Here are the steps to enable HTTPS in a Spring Boot application:

1. Obtain or generate an SSL/TLS certificate for your domain. This can be done through a trusted certificate authority or by creating a self-signed certificate.

2. Store the certificate and private key in a keystore file, typically in PKCS12 format.

3. Update the application's configuration in the application.properties or application.yml file to enable HTTPS. Set the following properties:



```
server.port=8443
server.ssl.key-store-type=PKCS12
server.ssl.key-store=classpath:keystore.p12
server.ssl.key-store-password=your-password
```

In this example, the server will listen on port 8443 and use the specified keystore file with the provided password.

4. Restart the application, and it will now be accessible over HTTPS.

Enabling HTTPS in a Spring Boot application ensures secure communication between the server and clients by encrypting the data transmitted over the network. It is particularly important when dealing with sensitive information, such as user credentials or financial data.

109. What is the purpose of the @RestControllerAdvice annotation?

The **@RestControllerAdvice** annotation in Spring Boot combines the functionalities of @ControllerAdvice and @ResponseBody annotations. It is used to handle exceptions and provide global, centralized exception handling for RESTful APIs.

When you annotate a class with @RestControllerAdvice , it becomes a specialized version of @ControllerAdvice , specifically for RESTful APIs. It allows you to define exception handling methods that are applied globally to all @RestController or @RequestMapping annotated classes in your application.

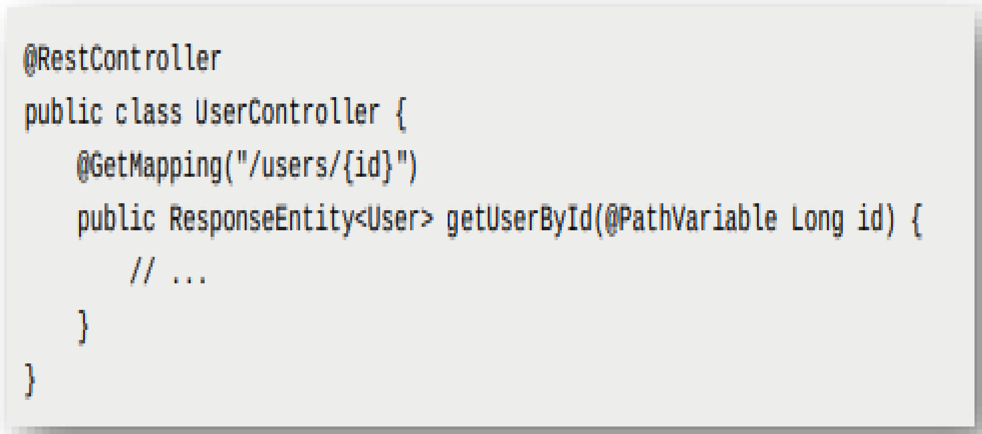
Exception handling methods within a @RestControllerAdvice class can return custom error responses, handle specific exceptions, perform logging, or execute any necessary actions when an exception occurs during API request processing.

The @RestControllerAdvice annotation simplifies the process of handling exceptions in RESTful APIs by providing a centralized mechanism for consistent error handling and response generation.

110. Explain the purpose of the @PathVariable annotation in Spring Boot.

The **@PathVariable** annotation in Spring Boot is used to extract and bind path variables from a URL to method parameters in a controller handler method. When you define a mapping for a URL with variables, such as /users/{id} , the @PathVariable annotation allows you to retrieve the value of the id variable and use it in your method logic.

Here's an example:



```
@RestController
public class UserController {
    @GetMapping("/users/{id}")
    public ResponseEntity<User> getUserById(@PathVariable Long id) {
        // ...
    }
}
```

In this example, the getUserById method maps to the /users/{id} URL. The @PathVariable annotation is used to bind the value of id in the URL to the id parameter of the method.

When a request is made to /users/123 , Spring Boot automatically maps the value 123 to the id parameter, allowing you to retrieve and process the requested user with the ID 123 .

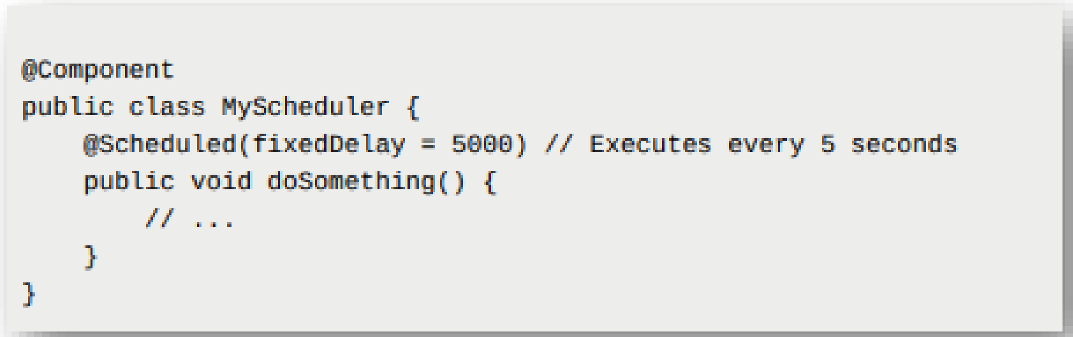
The @PathVariable annotation simplifies the handling of path variables in Spring Boot and allows you to create dynamic and parameterized mappings in your RESTful APIs.

111. What is the purpose of the @Scheduled annotation in Spring Boot?

The **@Scheduled annotation** in Spring Boot is used to schedule the execution of a method at fixed intervals or based on a cron expression.

By annotating a method with @Scheduled , you can specify when and how often the method should be executed. The annotated method will be automatically invoked by Spring Boot's scheduling infrastructure according to the configured schedule.

Here's an example:



```
@Component
public class MyScheduler {
    @Scheduled(fixedDelay = 5000) // Executes every 5 seconds
    public void doSomething() {
        // ...
    }
}
```

In this example, the doSomething method is annotated with @Scheduled and configured to execute every 5 seconds (fixedDelay = 5000).

The @Scheduled annotation is commonly used for tasks such as periodic data synchronization, cache refresh, report generation, or any other repetitive tasks.

Spring Boot's scheduling support provides a convenient way to automate such tasks, allowing you to focus on the business logic instead of manually managing timers or background threads.

112. What is the purpose of the @EnableAutoConfiguration annotation in Spring Boot?

The @EnableAutoConfiguration annotation in Spring Boot is used to enable the auto-configuration feature, which automatically configures the Spring application context based on the classpath dependencies, beans, and properties present in the application.

By adding @EnableAutoConfiguration to a configuration class or the main application class, Spring Boot performs classpath scanning and automatically configures various aspects of the application, such as data sources, JPA, messaging, web, security, and more. It leverages the Spring Boot starters, which provide pre-packaged configurations and dependencies for common use cases.

The auto-configuration feature reduces the need for manual configuration, boilerplate code, and tedious bean wiring. It promotes convention over configuration and allows developers to focus on the business logic rather than the low-level setup details.

The @EnableAutoConfiguration annotation is typically used in the main application class of a Spring Boot application.

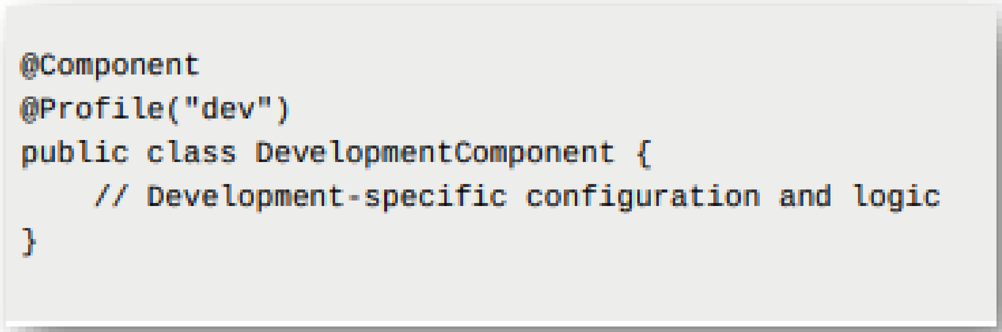
113. What is the purpose of the @Profile annotation in Spring Boot?

The **@Profile** annotation in Spring

Boot is used to activate or deactivate specific configurations, beans, or components based on the specified profile(s).

By associating the @Profile annotation with a class or method, you can control when that class or method should be considered for configuration and bean creation based on the active profiles.

For example, consider the following code snippet:



```
@Component
@Profile("dev")
public class DevelopmentComponent {
    // Development-specific configuration and logic
}
```

In this example, the DevelopmentComponent is annotated with @Profile("dev") , indicating that it should only be considered for configuration and bean creation when the "dev" profile is active.

Profiles can be activated through various means, such as environment variables, system properties, or the spring.profiles.active property in the application.properties or application.yml file..

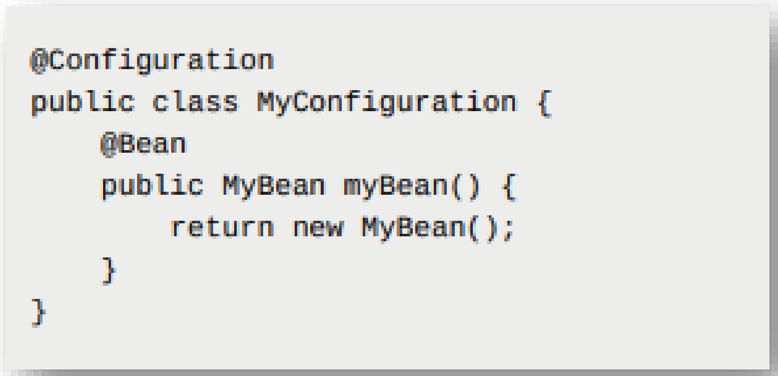
The @Profile annotation provides a powerful mechanism for creating different configurations or customizing behavior based on different runtime environments, such as development, testing, staging, or production.

114. What is the purpose of the @Bean annotation in Spring Boot?

The **@Bean** annotation in Spring Boot is used to indicate that a method produces a bean to be managed by the Spring application context.

By annotating a method with @Bean , you define a bean creation method that Spring Boot will invoke to instantiate and configure the object. The returned object from the method will be managed by the Spring container, and its lifecycle will be handled by Spring.

Here's an example:



```
@Configuration
public class MyConfiguration {
    @Bean
    public MyBean myBean() {
        return new MyBean();
    }
}
```

In this example, the myBean() method is annotated with @Bean , indicating that it produces a bean of type MyBean . When the Spring container is initialized, it will invoke this method and register the returned MyBean instance as a managed bean.

The @Bean annotation allows you to control the instantiation, configuration, and lifecycle management of objects in the Spring container. It's commonly used to define custom beans, third-party integrations, or to customize the behavior of existing beans.

115. Explain the concept of a RESTful API in Spring Boot.

A RESTful API (Representational State Transfer) in Spring Boot is an architectural style for building web services that follow the principles of REST. It provides a standard set of guidelines and constraints for designing, implementing, and interacting with web services.

In a RESTful API, resources are represented as URLs (Uniform Resource Locators), and clients interact with these resources using HTTP methods (GET, POST, PUT, DELETE) and standard HTTP status codes.

Spring Boot simplifies the development of RESTful APIs by providing features and integrations that align with the RESTful principles:

- **HTTP Verbs and Endpoints**: Spring Boot allows you to define REST endpoints using @RequestMapping or more specialized annotations like @GetMapping , @PostMapping , @PutMapping , and @DeleteMapping . These annotations map the endpoints to corresponding HTTP methods.

- **Request and Response Serialization**: Spring Boot integrates with libraries like Jackson or Gson to automatically serialize and deserialize request and response payloads in JSON or other formats.
- **Error Handling**: Spring Boot provides mechanisms for handling and returning appropriate HTTP status codes and error messages for different scenarios, such as validation errors, resource not found, or server errors.
- **Content Negotiation:** Spring Boot supports content negotiation, allowing clients to request different media types (JSON, XML, etc.) based on their preferences. It can automatically marshal and unmarshal data in the requested format.
- **HATEOAS:** Spring HATEOAS, a part of the Spring Boot ecosystem, enables the implementation of HATEOAS (Hypermedia as the Engine of Application State) by adding hypermedia links to API responses, allowing clients to navigate through the API dynamically.

By following RESTful principles and utilizing the features provided by Spring Boot, you can design and develop robust, scalable, and interoperable APIs that provide a uniform interface for clients to interact with your application.

116. What is @RestController annotation in Spring boot and how it differs from @controller?

@Controller

The @Controller annotation is a fundamental annotation in the Spring Framework used to mark a class as a controller in a web application.

Controllers are responsible for processing user requests and returning appropriate responses.

In traditional Spring MVC applications, controllers are used to handle requests and return views (HTML pages) that are rendered by a view resolver.

@RestController

The @RestController annotation is a specialized version of @Controller introduced in Spring 4.0. (@Controller + @ResponseBody)

It is used to define a RESTful web service controller that returns data in a serialized format, such as JSON or XML, instead of rendering views.

When you use @RestController, every method in the class is automatically annotated with @ResponseBody, which means the return values of those methods are directly serialized and sent as the HTTP response body.

117. What is Spring Cloud?

1. It is building blocks for Cloud and Microservices

2. It provides microservices infrastructure like provide use services such as Service Discovery, Configuration server and Monitoring.

3. It provides several other open source projects like Netflix OSS.

4. It provides PaaS like Cloud Foundry, AWS and Heroku.

5. It uses Spring Boot style starters

There are many use-cases supported by Spring Cloud like Cloud Integration, Dynamic Reconfiguration, Service Discovery, Security, Client side Load Balancing etc. But in this post we concentrate on following microservices support

118. REST and HTTP

REST builds on top of HTTP (Hypertext Transfer Protocol). HTTP is the language of the web.

HTTP has a few important verbs.

- POST - Create a new resource
- GET - Read a resource
- PUT - Update an existing resource
- DELETE - Delete a resource

HTTP also defines standard response codes.

- 200 – SUCCESS
- 404 - RESOURCE NOT FOUND
- 400 - BAD REQUEST
- 201 – CREATED
- 401 – UNAUTHORIZED
- 415 - UNSUPPORTED TYPE Representation not supported for the resource
- 500 - SERVER ERROR

Restful Service Constraints

- Client – Server : There should be a• service producer and a service consumer.
- The interface (URL) is uniform and• exposing resources. Interface uses nouns (not actions)

- The service is stateless. Even if the• service is called 10 times, the result must be the same. The service result should be Cacheable. HTTP cache, for example.
- Service should assume a Layered• architecture. Client should not assume direct connection to server - it might be getting info from a middle layer - cache.

119. Annotations used in Spring Boot

- **@SpringBootApplication**:
  This annotation is used to mark a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning.
- **@Bean**:
  It is applied on a method to specify that it returns a bean to be managed by Spring context.
- **@ RestController**:
  It is used to build REST API in a declarative way. @RestController annotation is applied to a class to mark it as a request handler, and Spring will do the building and provide the RESTful web service at runtime.
- **@ Service:**
  It is used at class level. It tells the Spring that class contains the business logic.
- **@ Repository**:
  It is a class-level annotation. The repository is a DAOs (Data Access Object) that access the database directly. The repository does all the operations related to the database.
- **@Configuration**
  It indicates that a class is a configuration class that may contain bean definitions.
- **@Autowired**
  Marks a constructor, field, or setter method to be auto wired by Spring dependency injection.
- **@ PostMapping**
  It maps the HTTP POST requests on the specific handler method. It is used to create a web service endpoint that creates. It is used instead of using:
  @RequestMapping(method = RequestMethod.POST)
- **@ GetMapping**
  It maps the HTTP GET requests on the specific handler method. It is used to create a web service endpoint that fetches. It is used instead of using:
  @RequestMapping(method = RequestMethod.GET)
- **@ DeleteMapping**
  It maps the HTTP DELETE requests on the specific handler method. It is used to create a web service endpoint that deletes a resource. It is used instead of using:
  @RequestMapping(method = RequestMethod.DELETE)
- **@ PutMapping**

It maps the HTTP PUT requests on the specific handler method. It is used to create a web service endpoint that creates or updates. It is used instead of using: @RequestMapping(method = RequestMethod.PUT)

- **@ RequestBody**
  It is used to bind HTTP request with an object in a method parameter. Internally it uses HTTP MessageConverters to convert the body of the request. When we annotate a method parameter with @RequestBody, the Spring framework binds the incoming HTTP request body to that parameter.
- **@PathVariable**
  It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method. @RequestParam: It is used to extract the query parameters form the URL. It is also known as a query parameter. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.
- **@EnableSwagger2**
  It is used to enable the Swagger2 for your Spring Boot application.
- **@Controller**
  It indicates that a particular class serves the role of a controller. Spring Controller annotation is typically used in combination with annotated handler methods based on the @RequestMapping annotation. It can be applied to classes only. It’s used to mark a class as a web request handler.