**Spring Professional Certification Study**

Table of Contents

[**1. Container, Dependency and IOC** 6](#_Toc536720038)

[1. 1 what is dependency injection and what are the advantages? 6](#_Toc536720039)

[Advantages: 7](#_Toc536720040)

[1.2 What is a pattern? What is an anti-pattern? Is dependency injection a pattern? 7](#_Toc536720041)

[1.3 What is an interface and what are the advantages of making use of them in Java? 7](#_Toc536720042)

[**Why are they recommended for Spring beans?** 7](#_Toc536720043)

[1.4 What is meant by “application-context? 7](#_Toc536720044)

[1.5 What is the concept of a “container” and what is its lifecycle? 7](#_Toc536720045)

[1.6 How are you going to create a new instance of an ApplicationContext? 8](#_Toc536720046)

[1.7 Can you describe the lifecycle of a Spring Bean in an ApplicationContext? 8](#_Toc536720047)

[1.8 How are you going to create an ApplicationContext in an integration test? 8](#_Toc536720048)

[1.9 What is the preferred way to close an application context? Does Spring Boot do this for you? 9](#_Toc536720049)

[1.10 Dependency injection using Java configuration? 9](#_Toc536720050)

[1.11 Dependency injection using annotations (@Component, @Autowired)? 9](#_Toc536720051)

[1.12 Component scanning, Stereotypes and Meta-Annotations? 9](#_Toc536720052)

[1.13 Scopes for Spring beans? What is the default scope? 9](#_Toc536720053)

[Singleton: 10](#_Toc536720054)

[Prototype: 10](#_Toc536720055)

[Request: 10](#_Toc536720056)

[Session: 10](#_Toc536720057)

[GlobalSession: 10](#_Toc536720058)

[1.14 Are beans lazily or eagerly instantiated by default? How do you alter this behaviour? 10](#_Toc536720059)

[1.15 What is a property source? How would you use @PropertySource? 10](#_Toc536720060)

[1.16 What is a BeanFactoryPostProcessor and what is it used for? When is it invoked? 10](#_Toc536720061)

[1.17 Why would you define a static @Bean method? 10](#_Toc536720062)

[1.18 What is a ProperySourcesPlaceholderConfigurer used for? 11](#_Toc536720063)

[1.19 What is a BeanPostProcessor and how is it different to a BeanFactoryPostProcessor? What do they do? When are they called? 11](#_Toc536720064)

[1.21 What is an initialization method and how is it declared on a Spring bean? 11](#_Toc536720065)

[1.22 What does component-scanning do? 11](#_Toc536720066)

[1.23 What is the behaviour of the annotation @Autowired with regards to field injection, constructor injection and method injection? 11](#_Toc536720067)

[1.24 What do you have to do, if you would like to inject something into a private field? How does this impact testing? 12](#_Toc536720068)

[1.25 How does the @Qualifier annotation complement the use of @Autowired? 12](#_Toc536720069)

[1.26 What are the advantages of Java Config? What are the limitations? 12](#_Toc536720070)

[1.27 What does the @Bean annotation do? 12](#_Toc536720071)

[1.28 What is the default bean id if you only use @Bean? How can you override this? 12](#_Toc536720072)

[1.29 Why are you not allowed to annotate a final class with @Configuration 12](#_Toc536720073)

[1.30 Why can’t @Bean methods be final either? 12](#_Toc536720074)

[1.31 How do you configure profiles? What are possible use cases where they might be useful? 13](#_Toc536720075)

[1.32 Can you use @Bean together with @Profile? 13](#_Toc536720076)

[1.33 How many profiles can you have? 13](#_Toc536720077)

[1.34 How do you inject scalar/literal values into Spring beans? 13](#_Toc536720078)

[1.35 What is @Value used for? 13](#_Toc536720079)

[1.36 What is Spring Expression Language (SpEL for short)? 13](#_Toc536720080)

[1.37 What is the Environment abstraction in Spring? 13](#_Toc536720081)

[**1.38 What can you reference using SpEL?** 13](#_Toc536720082)

[1.39 What is the difference between $ and # in @Value expressions? 13](#_Toc536720083)

[2. Aspect Oriented Programming(AOP) 14](#_Toc536720084)

[2.1 What is the concept of AOP? Which problem does it solve? What is a cross cutting concern? 14](#_Toc536720085)

[2.2 What is a pointcut, a join point, an advice, an aspect, weaving? 14](#_Toc536720086)

[Aspect: 14](#_Toc536720087)

[JointPoint: 14](#_Toc536720088)

[Advice: 14](#_Toc536720089)

[Poincut: 15](#_Toc536720090)

[Target: 15](#_Toc536720091)

[Weaving: 15](#_Toc536720092)

[2.3 How does Spring solve (implement) a cross cutting concern? 15](#_Toc536720093)

[2.4 Which are the limitations of the two proxy-types? 15](#_Toc536720094)

[2.5 How many advice types does Spring support. Can you name each one? What are they used for? 15](#_Toc536720095)

[Before Advice (@Before): 15](#_Toc536720096)

[After Advice (@After Returning): 15](#_Toc536720097)

[Around Advice (@Around): 15](#_Toc536720098)

[Throws Advice (@AfterThrowing): 15](#_Toc536720099)

[After-returning: 15](#_Toc536720100)

[2.6 What do you have to do to enable the detection of the @Aspect annotation? What does @EnableAspectJAutoProxy do? 16](#_Toc536720101)

[2.7 If shown pointcut expressions, would you understand them? For example, in the course we matched getter methods on Spring Beans, what would be the correct pointcut expression to match both getter and setter methods? 16](#_Toc536720102)

[2.8 What is the JoinPoint argument used for? 16](#_Toc536720103)

[2.9 What is a ProceedingJoinPoint? When is it used? 16](#_Toc536720104)

[**3. Data management: JDBC, Transaction, JPA,Spring Data** 17](#_Toc536720105)

[3.1 What is the difference between checked and unchecked exceptions? Why does Spring prefer unchecked exceptions? What is the data access exception hierarchy? 17](#_Toc536720106)

[Unchecked: 17](#_Toc536720107)

[Checked: 17](#_Toc536720108)

[3.2 How do you configure a DataSource in Spring? Which bean is very useful for development/test databases? 17](#_Toc536720109)

[JNDI: 17](#_Toc536720110)

[Bean Configuration: 18](#_Toc536720111)

[3.3 What is the Template design pattern and what is the JDBC template? 18](#_Toc536720112)

[3.4 What is a callback? What are the three JdbcTemplate callback interfaces that can be used with queries? What is each used for? (You would not have to remember the interface names in the exam, but you should know what they do if you see them in a code sample). 18](#_Toc536720113)

[RowMapper: 18](#_Toc536720114)

[ResultSetExtractor: 19](#_Toc536720115)

[PreparedStatement: 19](#_Toc536720116)

[3.5 Can you execute a plain SQL statement with the JDBC template? 20](#_Toc536720117)

[3.6 When does the JDBC template acquire (and release) a connection - for every method called or once per template? Why? 20](#_Toc536720118)

[3.7 What is a transaction? What is the difference between a local and a global transaction? 20](#_Toc536720119)

[3.9 Is a transaction a cross cutting concern? How is it implemented by Spring? 20](#_Toc536720120)

[3.10 How are you going to define a transaction in Spring? What does @Transactional do? What is the PlatformTransactionManager? 21](#_Toc536720121)

[3.11 Is the JDBC template able to participate in an existing transaction? 21](#_Toc536720122)

[3.12 What is a transaction isolation level? How many do we have and how are they ordered? 21](#_Toc536720123)

[3.13 What is @EnableTransactionManagement for? 22](#_Toc536720124)

[3.14 What does transaction propagation mean? What happens if one @Transactional annotated method is calling another @Transactional annotated method on the same object instance 22](#_Toc536720125)

[3.15 Where can the @Transactional annotation be used? What is a typical usage if you put it at class level? 22](#_Toc536720126)

[3. 15 what does declarative transaction management mean? 22](#_Toc536720127)

[3.16 What is the default rollback policy? How can you override it? ● What is the default rollback policy in a JUnit test, when you use the @RunWith(SpringJUnit4ClassRunner.class) in JUnit 4 or @ExtendWith(SpringExtension.class) in JUnit 5, and annotate your @Test annotated method with @Transactional? 22](#_Toc536720128)

[3.17 Why is the term “unit of work” so important and why does JDBC AutoCommit violate this pattern? 23](#_Toc536720129)

[3.18 What does JPA stand for - what about ORM? What is the idea behind an ORM? What are benefits/disadvantages or ORM? What is a PersistenceContext and what is an EntityManager. What is the relationship between both? Why do you need the @Entity annotation? Where can it be placed? 23](#_Toc536720130)

[3.19 What is @Query used for? 23](#_Toc536720131)

[**4. Spring Boot** 24](#_Toc536720132)

[4.1 What is Spring Boot? 24](#_Toc536720133)

[4.2 What are the advantages of using Spring Boot? 24](#_Toc536720134)

[4.3 Why is it “opinionated”? 24](#_Toc536720135)

[4.5 What things affect what Spring Boot sets up? 24](#_Toc536720136)

[4.6 How are properties defined? Where is Spring Boot’s default property source? 24](#_Toc536720137)

[4.7 Would you recognize common Spring Boot annotations and configuration properties if you saw them in the exam? 25](#_Toc536720138)

[4.8 What is the difference between an embedded container and a WAR? 25](#_Toc536720139)

[4.9 What embedded containers does Spring Boot support? 25](#_Toc536720140)

[4.10 What does @EnableAutoConfiguration do? What about @SpringBootApplication? 25](#_Toc536720141)

[4.11 Does Spring Boot do component scanning? Where does it look by default? 25](#_Toc536720142)

[4.12 What is a Spring Boot starter POM? Why is it useful? 25](#_Toc536720143)

[4.13 Spring Boot supports both Java properties and YML files. Would you recognize and understand them if you saw them? 26](#_Toc536720144)

[4.14 Can you control logging with Spring Boot? How? 26](#_Toc536720145)

[**5.** **Spring MVC and the Web Layer** 26](#_Toc536720146)

[5.1 MVC is an abbreviation for a design pattern. What does it stand for and what is the idea behind it? 26](#_Toc536720147)

[5.2 Do you need spring-mvc.jar in your classpath or is it part of spring-core? 26](#_Toc536720148)

[5.3 What is the DispatcherServlet and what is it used for? 26](#_Toc536720149)

[5.4 Is the DispatcherServlet instantiated via an application context? 27](#_Toc536720150)

[5.5 What is a web application context? What extra scopes does it offer? 27](#_Toc536720151)

[5.6 What is the @Controller annotation used for? How is an incoming request mapped to a controller and mapped to a method? 27](#_Toc536720152)

[5.7 What is the difference between @RequestMapping and @GetMapping? 28](#_Toc536720153)

[5.8 What is @RequestParam used for? 28](#_Toc536720154)

[5.9 What are the differences between @RequestParam and @PathVariable? 29](#_Toc536720155)

[5.11 What are some of the valid return types of a controller method? 29](#_Toc536720156)

[5.12 What is a View and what's the idea behind supporting different types of View? 29](#_Toc536720157)

[5.13 How is the right View chosen when it comes to the rendering phase? 29](#_Toc536720158)

[5.14 What is the Model? 29](#_Toc536720159)

[5.15 Why do you have access to the model in your View? Where does it come from? 30](#_Toc536720160)

[5.16 What is the purpose of the session scope? 30](#_Toc536720161)

[5.17 What is the default scope in the web context? 30](#_Toc536720162)

[5.18 Why are controllers testable artifacts? 30](#_Toc536720163)

[5.19 What does a ViewResolver do? 30](#_Toc536720164)

[**6. Security** 30](#_Toc536720165)

[6.1 What are authentication and authorization? Which must come first? 30](#_Toc536720166)

[6.2 Is security a cross cutting concern? How is it implemented internally? 31](#_Toc536720167)

[6.3 What is the delegating filter proxy? 31](#_Toc536720168)

[6.4 What is the security filter chain? 31](#_Toc536720169)

[6.5 What is a security context? 31](#_Toc536720170)

[6.6 Why do you need the intercept-url? 31](#_Toc536720171)

[6.7 In which order do you have to write multiple intercept-url's? 32](#_Toc536720172)

[6.8 What does the \*\* pattern in an antMatcher or mvcMatcher do? 32](#_Toc536720173)

[6.9 Why is an mvcMatcher more secure than an antMatcher? 32](#_Toc536720174)

[6.10 Does Spring Security support password hashing? What is salting? 32](#_Toc536720175)

[6.11 Why do you need method security? What type of object is typically secured at the method level (think of its purpose not its Java type). 33](#_Toc536720176)

[6.12 What do @PreAuthorized and @RolesAllowed do? What is the difference between them? What does Spring’s @Secured do? How are these annotations implemented? In which security annotation are you allowed to use SpEL? 33](#_Toc536720177)

[**7. REST** 33](#_Toc536720178)

[7.1 What does REST stand for? 33](#_Toc536720179)

[7.2 What is a resource? 33](#_Toc536720180)

[7.3 What does CRUD mean? 33](#_Toc536720181)

[7.4 Is REST secure? What can you do to secure it? 34](#_Toc536720182)

[7.5 What are safe REST operations? 34](#_Toc536720183)

[7.6 What are idempotent operations? Why is idempotency important? 34](#_Toc536720184)

[7.8 Is REST scalable and/or interoperable? 34](#_Toc536720185)

[7.9 Which HTTP methods does REST use? 34](#_Toc536720186)

[7.10 What is an HttpMessageConverter? 34](#_Toc536720187)

[7.11 Is REST normally stateless? 34](#_Toc536720188)

[7.12 What does @RequestMapping do? 34](#_Toc536720189)

[7.13 Is @Controller a stereotype? Is @RestController a stereotype? What is a stereotype annotation? What does that mean? 35](#_Toc536720190)

[7.14 What is the difference between @Controller and @RestController? 35](#_Toc536720191)

[7.15 When do you need @ResponseBody? 35](#_Toc536720192)

[7.17 What are the HTTP status return codes for a successful GET, POST, PUT or DELETE operation? 35](#_Toc536720193)

[7.18 When do you need @ResponseStatus? 36](#_Toc536720194)

[7.19 Where do you need @ResponseBody? What about @RequestBody? Try not to get these muddled up! 36](#_Toc536720195)

[7.20 If you saw example Controller code, would you understand what it is doing? Could you tell if it was annotated correctly? 36](#_Toc536720196)

[7.21 Do you need Spring MVC in your classpath? 36](#_Toc536720197)

[7.22 What Spring Boot starter would you use for a Spring REST application? 36](#_Toc536720198)

[7.23 What are the advantages of the RestTemplate? 36](#_Toc536720199)

[7.24 If you saw an example using RestTemplate would you understand what it is doing? 36](#_Toc536720200)

# **1. Container, Dependency and IOC**

## 1. 1 what is dependency injection and what are the advantages?

Dependency Injection is a design pattern which takes care of creating dependencies (to create the dependent object required by any other object) from a class thus resulting in a loosely coupled system.

### Advantages:

* Loosely couples.
* Re-usability of code.
* Reduce dependency to each other

## 1.2 What is a pattern? What is an anti-pattern? Is dependency injection a pattern?

A pattern would be to use of function for code reuse. An anti-pattern is a commonly occurring solution to a problem that generates decidedly negative consequences. Dependency Injection is strategy design pattern.

## 1.3 What is an interface and what are the advantages of making use of them in Java?

### **Why are they recommended for Spring beans?**

Interface is the contract which says to its class that either define all the declared methods or declared itself as an abstract class. In Spring Dependency Injection interface is very powerful to run time injection of various concrete implementations of an interface in the application. By using references to interfaces instead of their concrete implementation classes help to minimize ripple effects, as the user of interface reference doesn’t have to worry about the changes in the underlying concrete implementation.

## 1.4 What is meant by “application-context?

The application-context is core component of spring container in spring framework. “application-context” is loaded by the one of concrete implementation of ApplicationContext interface. The ApplicationContext is the central interface within a Spring Application for providing configuration information to the application.

## 1.5 What is the concept of a “container” and what is its lifecycle?

The Spring container is at the core of the Spring Framework. The container will create the objects (Objects are called Spring Beans), wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses Dependency Injection to manage the components that make up an application. Here is the bean lifecycle.

* Initializing Bean and Disposable Bean callback interfaces
* Bean Name, bean factory and Application Context Aware interfaces for specific behaviour.
* custom init() and destroy() methods in bean configuration file

## 1.6 How are you going to create a new instance of an ApplicationContext?

The most commonly used ApplicationContext implementations are

* ClassPathXmlApplicationContext
* AnnotationConfigApplicationContext
* FileSystemXmlApplicationContext
* WebXmlApplicationContext

|  |
| --- |
| *ApplicationContext context =* ***new*** *ClassPathXmlApplicationContext("application-context.xml");* |
| *EmailService emailService = (EmailService) context.getBean("emailservice");* |
| *emailService.sendMessage();* |
|  |
|  |
| *ApplicationContext context1 =* ***new*** *AnnotationConfigApplicationContext(JavaBasedConfig.****class****);* |
| *SMSService smsService = (SMSService) context1.getBean("smsservice");* |
| *smsService.sendMessage();* |
|  |
|  |
| *ApplicationContext context2 =* ***new*** *FileSystemXmlApplicationContext("E:/Practice/SpringLearning/src/application-context.xml");* |
| *EmailService emailService2 = (EmailService) context2.getBean("emailservice");* |
| *emailService2.sendMessage();* |

Github Link:

<https://github.com/kakanivenkatesh/spring-learning/tree/master/src/com/applicationcontext>

## 1.7 Can you describe the lifecycle of a Spring Bean in an ApplicationContext?

Refer 1.5

## 1.8 How are you going to create an ApplicationContext in an integration test?

First, any Spring enabled test will run with the help of @RunWith(SpringJUnit4ClassRunner.class).

## 1.9 What is the preferred way to close an application context? Does Spring Boot do this for you?

In a Spring web application, the **ContextLoaderListener** class registered in web.xml listens to this event (context destroyed) to properly close the Spring context. When you shut down your server, that ContextLoaderListener's contextDestroyed method is invoked.

## 1.10 Dependency injection using Java configuration?

Refer 1.6

## 1.11 Dependency injection using annotations (@Component, @Autowired)?

The @Component annotation marks a java class as a bean so the component-scanning mechanism of spring can pick it up and pull it into the application context. @Autowired to get an instance either by constructor or by a setter.

|  |
| --- |
| *@Component* |
| ***public******class*** *Message {* |
|  |
| ***private*** *MessgaeService service;* |
|  |
| *@Autowired* |
| ***void*** *Message(MessgaeService service){* |
| ***this****.service= service;* |
| *}* |
|  |
| ***public******void*** *sendMessage(){* |
| *service.sendMessage();* |
| *}* |
| *}* |

Github Link:

<https://github.com/kakanivenkatesh/spring-learning/tree/master/src/com/DI/constructor>

## 1.12 Component scanning, Stereotypes and Meta-Annotations?

The @ComponentScan annotation is used with the @Configuration annotation to tell Spring the packages to scan for annotated components. @ComponentScan also used to specify base packages and base package classes using the basePackageClasses or basePackages attributes of @ComponentScan. A stereotype annotation is an annotation that is used to declare the role that a component plays within the application. A meta-annotation is an annotation that is declared on another annotation. For example, @Service is meta-annotated with @Component.

## 1.13 Scopes for Spring beans? What is the default scope?

In Spring, bean scope is used to decide which type of bean instance should be returned from Spring container back to the caller.

### Singleton:

It returns a single object instance per Spring IOC container.

### Prototype:

It returns a new bean instance each time when requested. It does not store any cache version like singleton.

### Request:

It returns a single bean instance per HTTP request.

### Session:

It returns a single bean instance per HTTP session, its User level session.

### GlobalSession:

It returns a single bean instance per global HTTP session. It is only valid in the context of a web-aware Spring ApplicationContext.

If no bean scope is specified in bean configuration file, **default** to singleton

## 1.14 Are beans lazily or eagerly instantiated by default? How do you alter this behaviour?

As specified above the bean scopes are default to singleton, so default one would be eager instantiation. We can alter this by using lazy-init or @Lazy.

## 1.15 What is a property source? How would you use @PropertySource?

Spring @PropertySource annotation is used to provide properties file to Spring Environment. This annotation is used with @Configuration classes. We read a properties file and display the values with using @Value and Environment.

## 1.16 What is a BeanFactoryPostProcessor and what is it used for? When is it invoked?

BeanFactoryPostProcessor is used to work on bean configuration metadata. BeanFactoryPostProcessor can change bean configuration metadata potentially. It happens before other bean is initialized by spring container. In spring, more than one BeanFactoryPostProcessor can be used and their order of execution can be set by order attribute

## 1.17 Why would you define a static @Bean method?

The benefit of this is to have your bean generated before the instance method beans. You would use this for example with a BeanFactoryPostProcessor.

## 1.18 What is a ProperySourcesPlaceholderConfigurer used for?

PropertyPlaceholderConfigurer is used to resolve ${...} placeholders against a property. It can be local properties or system properties or environment variables. We can use PropertyPlaceholderConfigurer using XML as well as annotation

## 1.19 What is a BeanPostProcessor and how is it different to a BeanFactoryPostProcessor? What do they do? When are they called?

The BeanFactoryPostProcessor executes before bean Object instantiation (i.e. at the time Applicationcontext container is initialized)

BeanPostprocessor is executed after the bean object is created, as it can be executed before init() and after init().

## 1.21 What is an initialization method and how is it declared on a Spring bean?

When a bean is instantiated, it may be required to perform some initialization to get it into a usable state. Similarly, when the bean is no longer required and is removed from the container, some clean-up may be required.

## 1.22 What does component-scanning do?

The @ComponentScan annotation is used with the @Configuration annotation to tell Spring the packages to scan for annotated components. @ComponentScan also used to specify base packages and base package classes using the basePackageClasses or basePackages attributes of @ComponentScan

## 1.23 What is the behaviour of the annotation @Autowired with regards to field injection, constructor injection and method injection?

@Autowired matches the beans by type and inject into the place where we used either it may be constructor, method or field.

## 1.24 What do you have to do, if you would like to inject something into a private field? How does this impact testing?

To assign values into private fields Spring uses either ReflectionUtils or setter methods.

## 1.25 How does the @Qualifier annotation complement the use of @Autowired?

It’s used when you create more than one bean of the same type and want to wire only one of them with a property. In such cases, you can use the @Qualifier annotation along with @Autowired to remove the confusion by specifying which exact bean will be wired

## 1.26 What are the advantages of Java Config? What are the limitations?

The first advantage of the Java config is the compile time checking. You don’t need to wait for your application to start up to fix typos. Use and re-use of constants and enums.

## 1.27 What does the @Bean annotation do?

Spring @Bean Annotation is applied on a method to specify that it returns a bean to be managed by Spring context. Spring Bean annotation is usually declared in Configuration classes methods.

## 1.28 What is the default bean id if you only use @Bean? How can you override this?

The default bean name is the same as the method namewhen we use @Bean. We can override the bean using name attribute.

## 1.29 Why are you not allowed to annotate a final class with @Configuration

Spring creates dynamic proxies for classes annotated with @Configuration classes. Spring uses CGLIB to extend your class to create proxy. Hence, configuration classes cannot be final

## 1.30 Why can’t @Bean methods be final either?

Spring makes CGLib proxies for @Configuration classes.

## 1.31 How do you configure profiles? What are possible use cases where they might be useful?

Spring @Profile allow developers to register beans by condition. For example, register beans based on what operating system (Windows, \*nix) your application is running, or load a database properties file based on the application running in development, test, staging or production environment.

## 1.32 Can you use @Bean together with @Profile?

Yes we can use @Profile at method level.

## 1.33 How many profiles can you have?

We can have multiple profiles.

## 1.34 How do you inject scalar/literal values into Spring beans?

We have annotation @Value supports #{...} expressions and ${...} placeholders as well. @Value annotation is used for injecting values into fields in Spring managed

beans and it can be applied at the field or constructor/method parameter level.

## 1.35 What is @Value used for?

Refer 1.34

## 1.36 What is Spring Expression Language (SpEL for short)?

It's expression language that supports querying and manipulating an object graph at runtime

## 1.37 What is the Environment abstraction in Spring?

Environment abstraction is used for profile and properties, we can use it by @Autowiring where ever needed.

# **1.38 What can you reference using SpEL?**

We can reference variables, beans and system properties.

## 1.39 What is the difference between $ and # in @Value expressions?

${...} is the property placeholder syntax. It can only be used to dereference properties.

# {...} is Spring expression language syntax, which is far more capable and complex. It can also handle property placeholders, and a lot more besides.

# 2. Aspect Oriented Programming(AOP)

## 2.1 What is the concept of AOP? Which problem does it solve? What is a cross cutting concern?

The concept of Aspect Oriented Programming was avoiding redundant code (same functionality) which we are using in multiple places. It’s a programming model that allows you to apply cross cutting logic across various components the application in a decoupled manner.

**○** Name three typical cross cutting concerns.

* Logging
* Email/Notification
* Error Handling
* Transaction Management
* Security
* Caching

**○**What two problems arise if you don't solve a cross cutting concern via AOP?

* Coupling of concerns
* Same concern spread across modules

## 2.2 What is a pointcut, a join point, an advice, an aspect, weaving?

### Aspect:

Aspect is the piece of code that has to apply across various classes of the application.

### JointPoint:

The point at which you want apply te aspect logic, we can apply aspect logic at method execution.

### Advice:

Action taken by an aspect at particular joint point. There are multiple types like before advice, after returning, around advice and throws advice.

### Poincut:

Collection of join point representing on whom you want to advice the aspect.

### Target:

The class on which you want to advice the aspect.

### Weaving:

Process of advising a target class with an aspect based on a pointcut to build proxy.

## 2.3 How does Spring solve (implement) a cross cutting concern?

The **crosscutting concern** is a concern which is applicable throughout the application and it affects the entire application.

## 2.4 Which are the limitations of the two proxy-types?

Spring will create either **JDK**or **CGLib**proxies

## 2.5 How many advice types does Spring support. Can you name each one? What are they used for?

### Before Advice (@Before):

Here aspect method executes before the target class method execute.

### After Advice (@After Returning):

Executes after the target class method finishing execution. Only if the method completes successfully.

### Around Advice (@Around):

Executes the aspect around the target class jointpoint. So the advice can control the target method execution.

### Throws Advice (@AfterThrowing):

The advice method will execute only when the target class method throws an exception.

### After-returning:

Executes after the target class method finishing execution, regardless of its outcome.

Github Link:

<https://github.com/kakanivenkatesh/spring-learning/tree/master/src/com/AOP/Example>

**○** Which two advices can you use if you would like to try and catch exceptions?

@AfterThrowing

@After

## 2.6 What do you have to do to enable the detection of the @Aspect annotation? What does @EnableAspectJAutoProxy do?

It is used with @Configuration. All the classes which are annotated with @Aspect, need to configure in configuration file. It enables @Aspect support with Java @Configuration.

Github Link:

<https://github.com/kakanivenkatesh/spring-learning/tree/master/src/com/AOP/Example>

## 2.7 If shown pointcut expressions, would you understand them? For example, in the course we matched getter methods on Spring Beans, what would be the correct pointcut expression to match both getter and setter methods?

Yes

## 2.8 What is the JoinPoint argument used for?

Context provided by the JoinPoint parameter and Context about the intercepted point

## 2.9 What is a ProceedingJoinPoint? When is it used?

An around advice is a special advice that can control when and if a method (or other join point) is executed. This is true for around advices only, so they require an argument of type ProceedingJoinPoint, whereas other advices just use a plain JoinPoint. ProceedingJoinPoint is used as an argument of the methods which hints for before, after, after throwing and around. ProceedingJoinPoint has the methods like getKind, getTarget, proceed etc.

Github Link:

<https://github.com/kakanivenkatesh/spring-learning/tree/master/src/com/AOP/Example>

# **3. Data management: JDBC, Transaction, JPA,Spring Data**

## 3.1 What is the difference between checked and unchecked exceptions? Why does Spring prefer unchecked exceptions? What is the data access exception hierarchy?

### Unchecked:

Unchecked are the exceptions that are not checked at compiled time. “Error” and its subclasses,”RunTimeException” and its subclasses.

### Checked:

Checked exceptions are exceptions that are checked at compile time. Every exception other than Unchecked exception.

## 3.2 How do you configure a DataSource in Spring? Which bean is very useful for development/test databases?

We can configure DataSource three ways in Spring.

* JDBC-driver defined DataSources.
* JNDI DataSources
* Connection-pooling DataSources

org.springframework.jdbc.datasource.DriverManagerDataSource is usefull for dev or test database.

|  |
| --- |
|  |
| *<bean id = "dataSource" class = "org.springframework.jdbc.datasource.DriverManagerDataSource">* |
| *<property name = "driverClassName" value = "com.mysql.jdbc.Driver"/>* |
| *<property name = "url" value = "jdbc:mysql://localhost:3306/recruitment"/>* |
| *<property name = "username" value = "root"/>* |
| *<property name = "password" value = "root"/>* |
| *</bean>* |

Github Link:

<https://github.com/kakanivenkatesh/spring-learning/tree/master/src/com/spring/JDBC>

### JNDI:

We need to configure JNDI configuration in tomcat containder.to create the JNDI resource.

|  |
| --- |
| *<Resource name="jdbc/TestDB"* |
| *global="jdbc/TestDB"* |
| *auth="Container"* |
| *type="javax.sql.DataSource"* |
| *driverClassName="com.mysql.jdbc.Driver"* |
| *url="jdbc:mysql://localhost:3306/TestDB"* |
| *username="\*\*\*\*\*"* |
| *password="\*\*\*\*\*"* |
|  |
| *maxActive="100"* |
| *maxIdle="20"* |
| *minIdle="5"* |
| *maxWait="10000"/>* |

### Bean Configuration:

|  |
| --- |
| *<beans:bean id="dbDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">* |
| *<beans:property name="jndiName" value="java:comp/env/jdbc/MyLocalDB"/>* |
| *</beans:bean>* |

## 3.3 What is the Template design pattern and what is the JDBC template?

Template pattern is way of defining abstract class to perform an operation as per the requirement. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class.

JdbcTemplate takes care of working with many low-level operations like execute SQL queries, execute update statement, performs stored procedures calls, iterates over ResultSets and extracts returned parameter values. It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

## 3.4 What is a callback? What are the three JdbcTemplate callback interfaces that can be used with queries? What is each used for? (You would not have to remember the interface names in the exam, but you should know what they do if you see them in a code sample).

A callback function is a function which is accessible by another function, and is invoked after the first function if that first function completes.

### RowMapper:

RowMapper interface allows to map a row of the relations with the instance of user-defined class.

|  |
| --- |
| ***public******class*** *CandidateMapper* ***implements*** *RowMapper<Candidate>{* |
| ***public*** *Candidate mapRow(ResultSet rs,* ***int*** *rowNum)* ***throws*** *SQLException {* |
| *Candidate candidate =* ***new*** *Candidate();* |
| *candidate.setName(rs.getString("name"));* |
| *candidate.setEmail(rs.getString("email"));* |
| ***return*** *candidate;* |
| *}* |
| *}* |

### ResultSetExtractor:

ResultSetExtractor is supposed to extract the whole ResultSet (possibly multiple rows), while RowMapper is feeded with row at a time.

|  |
| --- |
| ***public*** *List<Candidate> listCandidatesInfo(){* |
| *String SQL = "select \* from Candidate";* |
| *List <Candidate> candidates = jdbctemplate.query(SQL,* ***new*** *ResultSetExtractor<List<Candidate>>(){* |
| ***public*** *List<Candidate> extractData(ResultSet rs)* ***throws*** *SQLException, DataAccessException {* |
|  |
| *List<Candidate> list =* ***new*** *ArrayList<Candidate>();* |
| ***while****(rs.next()){* |
| *Candidate candidate =* ***new*** *Candidate();* |
| *candidate.setName(rs.getString("name"));* |
| *candidate.setEmail(rs.getString("email"));* |
| *list.add(candidate);* |
| *}* |
| ***return*** *list;* |
| *}* |
| *});* |
| ***return*** *candidates;* |
| *}* |

### PreparedStatement:

To use parameterized query, we pass the instance of PreparedStatementCallback in the execute method.

|  |
| --- |
| ***public******boolean*** *insertCandidateInfoByPreparedStatement(Candidate candidate){* |
| *String query="insert into Candidate values(?,?,?)";* |
| ***return*** *jdbctemplate.execute(query,* ***new*** *PreparedStatementCallback<Boolean>(){* |
|  |
| *@Override* |
| ***public*** *Boolean doInPreparedStatement(PreparedStatement preparestatment)* |
| ***throws*** *SQLException, DataAccessException {* |
|  |
| *preparestatment.setInt(1, candidate.getId());* |
| *preparestatment.setString(2, candidate.getName());* |
| *preparestatment.setString(3, candidate.getEmail());* |
| ***return*** *preparestatment.execute();* |
| *}* |
|  |
| *});* |
| *}* |

## 3.5 Can you execute a plain SQL statement with the JDBC template?

Yes, we can execute.

## 3.6 When does the JDBC template acquire (and release) a connection - for every method called or once per template? Why?

JdbcTemplate supports querying for any type of object assuming you supplied a RowMapper interface implementation defining the way database table should be mapped to some entity type.

|  |
| --- |
| *public List<Candidate> listCandidates() {* |
| *return jdbctemplate.query(SQL, RowMapper<Candidate>(){* |
| *public Candidate mapRow(ResultSet rs, int rowNum) throws SQLException {* |
| *Candidate candidate = new Candidate();* |
| *candidate.setName(rs.getString("name"));* |
| *candidate.setEmail(rs.getString("email"));* |
| *return candidate;* |
| *}* |
| *}* |
| *}* |

## 3.7 What is a transaction? What is the difference between a local and a global transaction?

A database transaction is a sequence of actions that are treated as a single unit of work. Transactions are described in terms of ACID properties, which are as follows Atomic, Consistent, Isolated and Durable.

* **Atomic**: all changes to the database made in a transaction are rolled back if any change fails.
* **Consistent**: the effects of a transaction take the database from one consistent state to another consistent state.
* **Isolated**: the intermediate steps in a transaction are not visible to other users of the database.
* **Durable**: when a transaction is completed (committed or rolled back), its effects persist in the database.

Local transaction is a simple transaction that is about one single database. Global transactions are applications server managed and spread across many components.

## 3.9 Is a transaction a cross cutting concern? How is it implemented by Spring?

Yes it’s a cross cutting concern as it can affect many components, it can be implemented using the AOP (aspect oriented programming) in spring.

3.10 How are you going to define a transaction in Spring? What does @Transactional do? What is the PlatformTransactionManager?

You can define transactions in 2 ways:

* Declarative way
* Programmatic way

**Declarative way** deals with adding some AOP related to transactions to the methods annotated with @Transactional. P**rogrammatic way** is about using either **TransactionTemplate** or directly **PlatformTransactionManager**.

Rather than using XML AOP for matching the methods that should be transactional you can add this (@Transactional) annotation.  PlatformTransactionManager is an interface that defines the transaction strategy through different implementations that match requirements specific to the project they are used in.

## 3.11 Is the JDBC template able to participate in an existing transaction?

JdbcTemplate itself cannot manage transactions that is action of TransactionTemplate. If we define @Transactional on method along with some jdbc template code it ill run that transaction.

## 3.12 What is a transaction isolation level? How many do we have and how are they ordered?

Isolation level defines how the changes made to some data repository by one transaction affect other simultaneous concurrent transactions

DEFAULT: Default isolation level. It uses the isolation of underlying datasource.

READ\_COMMITTED: Dirty reads NOT supported; Non-repeatable reads and Phantom reads can occur.

READ\_UNCOMMITTED: Dirty reads / Non-repeatable reads / Phantom reads all can occur.

REPEATABLE\_READ: Dirty reads and non-repeatable reads are prevented; phantom reads can occur.

SERIALIZABLE: Dirty reads, non-repeatable reads and phantom reads are prevented.

## 3.13 What is @EnableTransactionManagement for?

Enabling Spring's annotation-driven transaction management. It is used on @Configuration classes.

3.14 What does transaction propagation mean? What happens if one @Transactional annotated method is calling another @Transactional annotated method on the same object instance**?**

Transaction propagation defines whether current transaction will be extended or not. Since transaction-management is implemented using AOP @Transactional annotation will have no effect on the method being called as no proxy is created. The same behavior is characteristic for AOP aspects.

## 3.15 Where can the @Transactional annotation be used? What is a typical usage if you put it at class level?

We can be use either on the class level or method level. If you put @Transactional at class-level this is equal to annotating each method of that class. Means it will applicable to all methods.

## 3. 15 what does declarative transaction management mean?

Declarative transaction management is a model build on AOP. Spring has some transactional aspects that may be used to advise methods for them to work in a transactional manner.

## 3.16 What is the default rollback policy? How can you override it? ● What is the default rollback policy in a JUnit test, when you use the @RunWith(SpringJUnit4ClassRunner.class) in JUnit 4 or @ExtendWith(SpringExtension.class) in JUnit 5, and annotate your @Test annotated method with @Transactional?

Any RuntimeException or Error will cause by default a rollback. Test transactions will be automatically rolled back after completion of the test.

## 3.17 Why is the term “unit of work” so important and why does JDBC AutoCommit violate this pattern?

JDBC AutoCommit will treat each individual SQL statement as a transaction. This means that if logically or from business point of view you need to make sure some statement is ok before executing another one, it would fail. Transactions are meant to solve this problem by “grouping” operations in some logical way so that you confirm to ACID principle.

## 3.18 What does JPA stand for - what about ORM? What is the idea behind an ORM? What are benefits/disadvantages or ORM? What is a PersistenceContext and what is an EntityManager. What is the relationship between both? Why do you need the @Entity annotation? Where can it be placed?

JPA stands for Java Persistence API. ORM stands for Object Relational Mapping. The idea behind ORM is that in applications we deal with objects and classes .ORM is meant to help us map our entities to the database tables.

PersistenceContext is a set of entity instances that correspond to persistent entity identity. All the manipulations with the persistent instances are made using EntityManager. EntityManager writes, deletes and searches entities to database.

@Entity annotation is used to mark a class as an entity class that will be mapped to the database using ORM.

## 3.19 What is @Query used for?

Spring @Query annotation is used for customize methods of some instant repository.

# **4. Spring Boot**

## 4.1 What is Spring Boot?

Spring Boot application is a framework which works on top of Spring Framework. Used to create micro service which allows users to develop and deploy services independently. It achieves the lightweight model to support business application. And it will simplifies configuring a Spring application.

## 4.2 What are the advantages of using Spring Boot?

* Easy configuration.
* It reduces lots of development time and increases productivity
* It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
* Spring Boot Starters will manage set of dependencies (spring-boot-starter- \*).

Example:

* spring-boot-starter-actuator
* spring-boot-starter-security
* spring-boot-starter-web….

## 4.3 Why is it “opinionated”?

A Framework makes some assumptions on what you need based on dependencies. Spring Boot just decides on a set of default configured beans which you can override if you want. If these assumptions are then the framework considered as opinionated.

## 4.5 What things affect what Spring Boot sets up?

Starters which are added to dependencies @EnableAutoConfiguration or @SpringBoot

## 4.6 How are properties defined? Where is Spring Boot’s default property source?

Properties are defined in .properties or .yml. We can read properties file using Environment object, @ConfigurationProperties and @Value annotation.

**Github Link:**

<https://github.com/kakanivenkatesh/spring-boot/tree/master/src/main/java/com/sample/sboot>

## 4.7 Would you recognize common Spring Boot annotations and configuration properties if you saw them in the exam?

Yes

## 4.8 What is the difference between an embedded container and a WAR?

Embedded container is a server that comes with the resulting application .WAR is archive which contains dependencies that can be deployed on external container.

## 4.9 What embedded containers does Spring Boot support?

Spring boot is a Java based framework that supports application services. It runs as a standalone jar with an embedded servlet container or as a WAR file inside a container. The spring boot framework supports three different types of embedded servlet containers:

* Tomcat (default),
* Jetty
* Undertow

## 4.10 What does @EnableAutoConfiguration do? What about @SpringBootApplication?

Auto-configuration classes are usually applied based on your classpath and what beans you have defined. You can always manually exclude () any configuration that you never want to apply using excludeName().

If you added @SpringBootApplication annotation to the class, you do not need to add the @EnableAutoConfiguration, @ComponentScan and @SpringBootConfiguration annotation. The @SpringBootApplication annotation includes all other annotations

## 4.11 Does Spring Boot do component scanning? Where does it look by default?

Yes, it scans where main method declared with @SpringBootApplication annotation. And it will scan sub packages too.

## 4.12 What is a Spring Boot starter POM? Why is it useful?

Starter POM is a set of dependencies which are needed for project.

## 4.13 Spring Boot supports both Java properties and YML files. Would you recognize and understand them if you saw them?

Yes. Java properties files come with .properties where as YML comes with .yml.

## 4.14 Can you control logging with Spring Boot? How?

Yes, we can configure logging in Spring Boot using Logback.xml or using application.properties.

# **5.** **Spring MVC and the Web Layer**

## 5.1 MVC is an abbreviation for a design pattern. What does it stand for and what is the idea behind it?

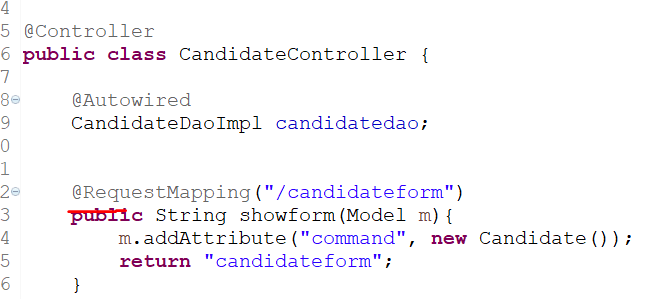
MVC stands for Model-View-Controller. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements. Here model contains the data, View is responsible for rendering the Model data, Controller is responsible to get user request and to build model to pass it to the View for rendering the data.

## 5.2 Do you need spring-mvc.jar in your classpath or is it part of spring-core?

spring-mvc.jar is part of Spring MVC not a part for spring-core. So to use SpringMVC we need to add the jar into application path (in WEB-INF/lib).

## 5.3 What is the DispatcherServlet and what is it used for?

DispatcherServlet is a front controller which handles all the HTTP requests and responses requesting Spring MVC application. After receiving the request it will consult the HandlerMapping (@RequestMapping) to call the appropriate Controller to process the request. It' responsible to pick defined view for the request with the help of View Resolver and then sending the rendered response to the client.



## 5.4 Is the DispatcherServlet instantiated via an application context?

It will instantiated by the servlet container. We must define the DispatcherServlet into the web.xml file. you can see that load-on-startup tag is 1, which means DispatcherServlet is instantiated when you deploy the Spring MVC application into container.

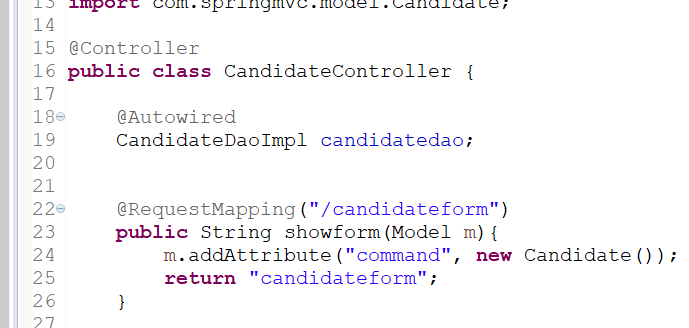


## 5.5 What is a web application context? What extra scopes does it offer?

And "WebApplicationContext" is a child of "ApplicationContext", used to deal with the web- related components such as controllers and view resolvers, which is configured using "DispatcherServlet"

## 5.6 What is the @Controller annotation used for? How is an incoming request mapped to a controller and mapped to a method?

@Controller annotation is used to specify that the class as a controller and it will be used by DispatcherServlet to process the incoming requests. We will use @RequestMapping to handle the requests and map to method.



Github Link:

<https://github.com/kakanivenkatesh/SpringMVC>

## 5.7 What is the difference between @RequestMapping and @GetMapping?

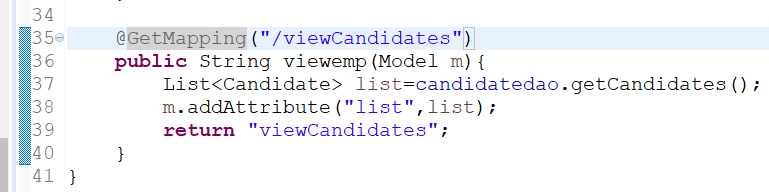
@RequestMapping can be used both at the class and at the method level. GetMapping only applies to method and we use it for mapping HTTP GET requests onto specific handler methods 9@GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET) ).

It supports consumes

consumes = "text/plain"

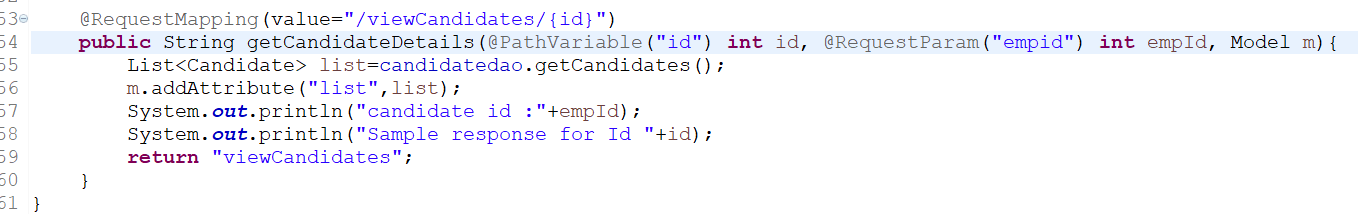
consumes = {"text/plain", "application/\*"}

What is @RequestParam used for?



## 5.8 What is @RequestParam used for?

We use the @RequestParam annotation to accept query parameters in Controller’s handler methods.



## 5.9 What are the differences between @RequestParam and @PathVariable?

Extract data from the incoming request and mapping the request to controller. @RequestParam annotation to accept query parameters in Controller’s handler methods. Where as @PathVariable extracts values from URI.

http://localhost:8081/SpringMVCExample/viewCandidates/12268?empid=12269

From the above we can use @RequestParam to retrieve query parameter empid and @PathVariable to get id (12268)

## 5.11 What are some of the valid return types of a controller method?

ModelAndView, String, void, View, HttpEntity<?> or ResponseEntity<?>, Map are some of the valid return types of a controller method.

## 5.12 What is a View and what's the idea behind supporting different types of View?

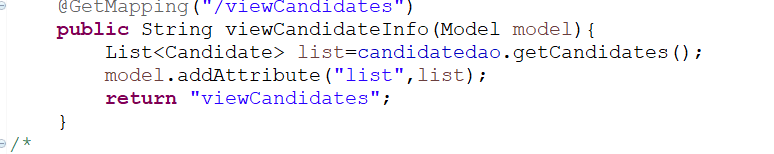
View is the interface in SpringMVC whose implementations are responsible for rendering contest and exposing the model. Or passing the model data from controller to the view a view resolver is required. View can be resolved into different types of View implementation

## 5.13 How is the right View chosen when it comes to the rendering phase?

The DispatcherServlet will take help from ViewResolver to pick up the defined view for the request. Once view is finalized, The DispatcherServlet passes the model data to the view, which is finally rendered, on the browsers

## 5.14 What is the Model?

Model is a reference to encapsulate the data or output for rendering. Model is a map of attributes passed by the controller to the view.



## 5.15 Why do you have access to the model in your View? Where does it come from?

Model is created by the Controller and passed back to the DispatcherServlet. And from DispatcherServlet to ViewResolver where Model should passed and then passes it to respective View. So Model comes from DispatcherServlet.

## 5.16 What is the purpose of the session scope?

The purpose of the session scope is to create an instance of the bean for an HTTP Session. This means the same bean can serve multiple requests if it is scoped in session. You can define the scope of a Spring bean using scope attribute or the @Scope annotation in a Spring MVC application

## 5.17 What is the default scope in the web context?

 The default bean scope is singleton.

## 5.18 Why are controllers testable artifacts?

Controllers are testable artifacts because they are not directly coupled with any View

## 5.19 What does a ViewResolver do?

ViewResolver returns View to handle an output rendering based on the logical view name which is provided by controller. This is how Controller is not coupled with specific view technology. ViewResolver to pick up the defined view for the request based on the view name.

# **6. Security**

## 6.1 What are authentication and authorization? Which must come first?

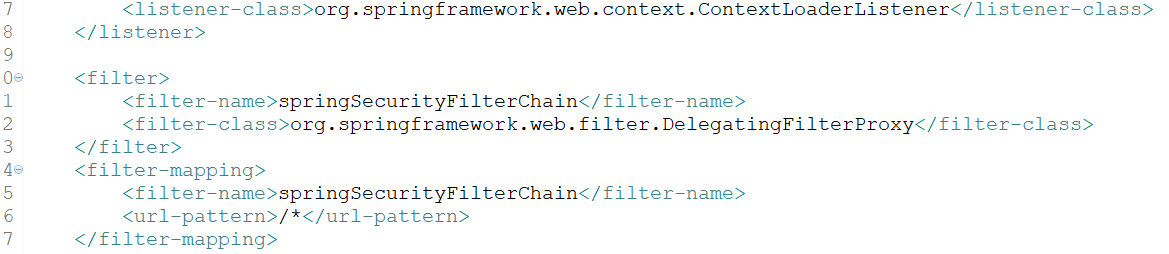
Authentication is validating our credential like username/id and password. Authorization determines your ability to access the system and up to what extent. In simple terms, authentication is the process of verifying who you are, while authorization is the process of verifying what you have access to.

## 6.2 Is security a cross cutting concern? How is it implemented internally?

Yes. Logging, security and data transfer is applicable for all the module of an application. So it's cross-cutting concern. Can implement using AOP.

## 6.3 What is the delegating filter proxy?

Proxy for a standard Servlet Filter, delegating to a Spring-managed bean that implements the Filter interface. Delegating filter proxy will pass requests to the filter whose name is **springSecurityFilterChain.**



## 6.4 What is the security filter chain?

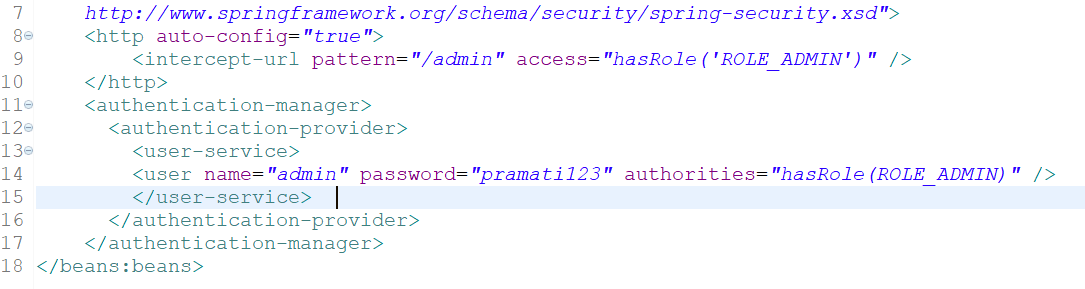
Spring Security maintains a filter chain internally where each of the filters has a particular responsibility and filters are added or removed from the configuration depending on which services are required. The ordering of the filters is important as there are dependencies between them.

## 6.5 What is a security context?

The SecurityContext is used to store the details of the currently authenticated user, also known as a principle. Context is held in the SecurityContextHolder (SecurityContextHolder is a helper class, which provide access to the security context).

## 6.6 Why do you need the intercept-url?

With intercept-url we specifies the urls which we want to have some. This tag has a **pattern** attribute that accepts either ant style paths or regex for matching and **Access** attribute accepts comma-separated roles that will be allowed to access the resource.



Github Link:

<https://github.com/kakanivenkatesh/spring-security>

## 6.7 In which order do you have to write multiple intercept-url's?

The intercept URLs need to be listed from most to least specific. If we put least first like below.

pattern="/test1/test2/\*\*" pattern="/test1/test2/test3".

If anyone accessed /test1/test2/test3, then the permission settings from test1/test2 will get applied as it was matched the first from the list of intercept URLs

## 6.8 What does the \*\* pattern in an antMatcher or mvcMatcher do?

\* matches zero or more characters. \*\* matches zero or more 'directories' in a path.

MvcMatcher() uses Spring MVC's HandlerMappingIntrospector to match the path and extract variables. AntMatcher() is an implementation for Ant-style path patterns

## 6.9 Why is an mvcMatcher more secure than an antMatcher?

mvcMatcher will restrict the URLs by HTTP method.

## 6.10 Does Spring Security support password hashing? What is salting?

Spring supports password hashing, password salting is a form of password encryption which we add extra string to passwords and then hashing the new string of characters which makes hard to trace/break.

## 6.11 Why do you need method security? What type of object is typically secured at the method level (think of its purpose not its Java type).

To secure more we need method level. If we secure web layer there may be a chance of accessing service layer in case if we expose the REST endpoints. So we secure services at method level.

## 6.12 What do @PreAuthorized and @RolesAllowed do? What is the difference between them? What does Spring’s @Secured do? How are these annotations implemented? In which security annotation are you allowed to use SpEL?

@PreAuthorize annotation is for method level security and it’s also known as expression based annotations. That means SPEL (Spring Expression Language) will be used to determine the authorization of the user on the requested resource, @PreAuthorize check the authorization before method execution. @RolesAllowed is a standard java annotation and defined in JSR 250 specification. It works in the same way as @Secured annotation. @Secured is from spring security and can be used for method level security.

# **7. REST**

## 7.1 What does REST stand for?

REST stands for REpresentational State Transfer.

## 7.2 What is a resource?

A resource is a piece of data that can be accessed by the code of an application

## 7.3 What does CRUD mean?

CRUD stands for create, read, update and delete. These are the four basic functions of persistent storage.

## 7.4 Is REST secure? What can you do to secure it?

No it's not. We can implement some method level spring security to make it secure.

## 7.5 What are safe REST operations?

Safe operations are operations that don’t change things GET, HEAD, OPTIONS

## 7.6 What are idempotent operations? Why is idempotency important?

Idempotent operations are operations that always return the same result even though if we applied N number of times. Idempotency is important to maintain consistency and for understanding the limits of effect.

## 7.8 Is REST scalable and/or interoperable?

Because of the stateless nature (does not store any state about the client session on the server side) of REST it is easily scalable and due to its HTTP usage it is highly interoperable because many systems support HTTP

## 7.9 Which HTTP methods does REST use?

It will use GET for read, PUT for update/replace, POST for create, PATCH for update/modify, DELETE for delete

## 7.10 What is an HttpMessageConverter?

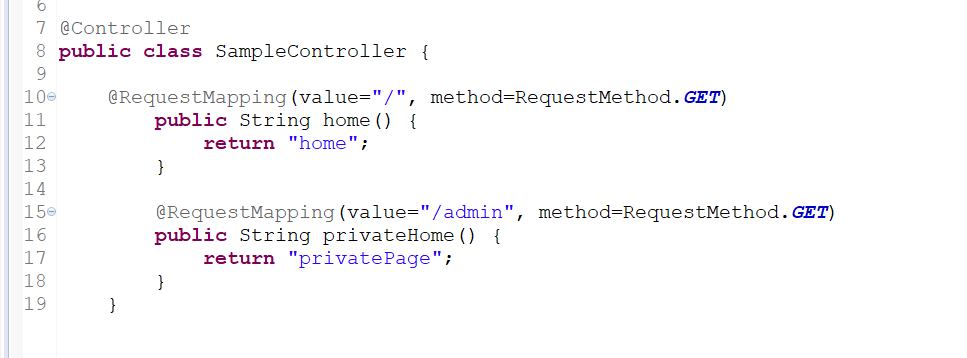
HttpMessageConverters convert HTTP requests to objects and back from objects to HTTP responses. Simply put, message converters are used to marshall and unmarshall Java Objects to and from JSON, XML, etc over HTTP.

## 7.11 Is REST normally stateless?

Yes, REST is stateless.

## 7.12 What does @RequestMapping do?

RequestMapping annotation is used to map web requests onto specific handler classes and/or handler methods. @RequestMapping can be applied to the controller class as well as methods.



## 7.13 Is @Controller a stereotype? Is @RestController a stereotype? What is a stereotype annotation? What does that mean?

Yes, @Controller is stereotype and indicates that an annotated class is a “Controller”. If class annotated with stereotype annotation spring will automatically register them in the application context.

## 7.14 What is the difference between @Controller and @RestController?

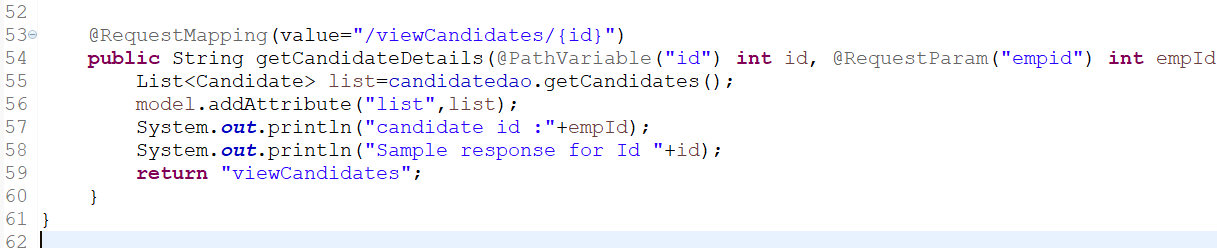
The @RestController annotation in Spring MVC is nothing but a combination of @Controller and @ResponseBody annotation. For a @Controller to act as a @RestController it has to be combined with @ResponseBody.

## 7.15 When do you need @ResponseBody?

@ResponseBody is required when you want a controller result to me passed to a message converter rather than to a view resolver.

**7.16 What does @PathVariable do?**

@PathVariable extracts values from URI.



## 7.17 What are the HTTP status return codes for a successful GET, POST, PUT or DELETE operation?

GET, PUT and DELETE will send 204 n success and 204 for no content. POST get 201 for successful operation. Status-Code is a 3-digit integer where the first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. 2xx represent for on success.

## 7.18 When do you need @ResponseStatus?

@ResponseStatus will prevent DispatcherServlet from trying to find a view for the result and will set a status for the response.

## 7.19 Where do you need @ResponseBody? What about @RequestBody? Try not to get these muddled up!

@ResponseBody, spring will try to convert its return value and write it to the http response automatically. If you annotate a methods parameter with @RequestBody, spring will try to convert the content of the incoming request body to your parameter object on the fly.

## 7.20 If you saw example Controller code, would you understand what it is doing? Could you tell if it was annotated correctly?

Yes

## 7.21 Do you need Spring MVC in your classpath?

Yes. Spring MVC is the core component for REST support.

## 7.22 What Spring Boot starter would you use for a Spring REST application?

Will use spring-boot-starter-web.

## 7.23 What are the advantages of the RestTemplate?

RestTemplate has methods specific to HTTP methods (DELETE, GET, HEAD, OPTIONS, POST and PUT). So it is very convenient for REST calls.

## 7.24 If you saw an example using RestTemplate would you understand what it is doing?

Yes