Contents

[Spring Beans and IOC Container 4](#_Toc13531244)

[PROTOTYPE Scope 4](#_Toc13531245)

[During which phase can BeanPostProcessor modify the bean initialization? 4](#_Toc13531246)

[@SpringBootApplication annotation equivalent to @ComponentScan, @Configuration @EnableAutoConfiguration 4](#_Toc13531247)

[What is the default behavior of Spring application context? 4](#_Toc13531248)

[Which is False about Prototype scope? 5](#_Toc13531249)

[ApplicationContext and BeanFactory 5](#_Toc13531250)

[BeanCurrentlyInCreationException 6](#_Toc13531251)

[Valid NameSpaces 6](#_Toc13531252)

[Example of Bean Definition inheritance 6](#_Toc13531253)

[@Autowire 7](#_Toc13531254)

[<context:component-scan> 9](#_Toc13531255)

[@PostConstruct 9](#_Toc13531256)

[@Bean Annotation 10](#_Toc13531257)

[Spring bean lifecycle 12](#_Toc13531258)

[Example of BeanNameAware in Spring. 12](#_Toc13531259)

[Implement BeanNameAware 12](#_Toc13531260)

[Difference between BeanNameAware and BeanFactoryAware interfaces, in the Spring Framework. 12](#_Toc13531261)

[Example of ApplicationContextAware in Spring 13](#_Toc13531262)

[What is Need for Using ApplicationContextAware in Application? 13](#_Toc13531263)

[Bean Creation Life-Cycle Callback. 13](#_Toc13531264)

[@PostConstruct 13](#_Toc13531265)

[Implement InitializingBean interface( @override afterPropertiesSet) 13](#_Toc13531266)

[init-method or @Bean(initMethod=”..”) 13](#_Toc13531267)

[Bean Destruction Life-Cycle Callback 13](#_Toc13531268)

[@PostConstruct 13](#_Toc13531269)

[Implement DisposableBean interface( @override destroy()) 13](#_Toc13531270)

[Destroy-method or @Bean(destroyMethod=”..”) 13](#_Toc13531271)

[I would not recommend to use InitializingBean and DisposableBean interface, because it will tight coupled your code to Spring. A better approach should be specifying the init-method and destroy-method attributes in your bean configuration file. 13](#_Toc13531272)

[What is the first step of the initialization phase of the Spring application lifecycle? 14](#_Toc13531273)

[BeanPostProcessor interface. 14](#_Toc13531274)

[Difference between BeanPostProcessor and @ @PostConstruct, InitializingBean and custom init method. 19](#_Toc13531275)

[Create spring application context? 19](#_Toc13531276)

[Which beans are initialized lazily and eagerly? 20](#_Toc13531277)

[@Profile Usecase 20](#_Toc13531278)

[Is it possible to use a profile expression inside, profile string annotation? Answer : D 20](#_Toc13531279)

[How Many beans will be created? Remember ApplicationConfig is also a spring bean. It is used to create the others. Answer is 4 beans. 21](#_Toc13531280)

[How can you create an application context from multiple files, which annotation can used to combined 21](#_Toc13531281)

[multiple configuration files? 21](#_Toc13531282)

[It is not illegal to define the same bean more than once, but the you get the last bean Spring sees defined. Each bean have a unique id so the previous bean is replaced by config2 class method. 21](#_Toc13531283)

[@Order annotation. 22](#_Toc13531284)

[Spring Security Configuration 22](#_Toc13531285)

[What are the class required to apply spring security as java configuration. 24](#_Toc13531286)

[Create custom login page. 26](#_Toc13531287)

[Spring Boot Intro. 26](#_Toc13531288)

[Logging 26](#_Toc13531289)

[Using the @SpringBootApplication Annotation 26](#_Toc13531290)

[• What is Spring Boot? 27](#_Toc13531291)

[• What are the advantages of using Spring Boot? 27](#_Toc13531292)

[• Why is it “opinionated”? 27](#_Toc13531293)

[• What things affect what Spring Boot sets up? 27](#_Toc13531294)

[• What is a Spring Boot starter POM? Why is it useful? 27](#_Toc13531295)

[• Spring Boot supports both properties and YML files. 27](#_Toc13531296)

[Would you recognize and understand them if you saw them? 27](#_Toc13531297)

[• Can you control logging with Spring Boot? How? 27](#_Toc13531298)

[• Where does Spring Boot look for property file by default? 27](#_Toc13531299)

[• How do you define profile specific property files? 27](#_Toc13531300)

[• How do you access the properties defined in the property files? 27](#_Toc13531301)

[• What properties do you have to define in order to configure external MySQL? 27](#_Toc13531302)

[• How do you configure default schema and initial data? 27](#_Toc13531303)

[• What is a fat far? How is it different from the original jar? 27](#_Toc13531304)

[• What is the difference between an embedded container and a WAR? 28](#_Toc13531305)

[• What embedded containers does Spring Boot support? 28](#_Toc13531306)

[Spring Boot Auto Configuration 28](#_Toc13531307)

[spring-boot-starter-parent 28](#_Toc13531308)

[• How does Spring Boot know what to configure? 28](#_Toc13531309)

[• What does @EnableAutoConfiguration do? 28](#_Toc13531310)

[• What does @SpringBootApplication do? 28](#_Toc13531311)

[• Does Spring Boot do component scanning? Where does it look by default? 28](#_Toc13531312)

[• How are DataSource and JdbcTemplate auto-configured? 28](#_Toc13531313)

[• What is spring.factories file for? 28](#_Toc13531314)

[• How do you customize Spring auto configuration? 28](#_Toc13531315)

[• What are the examples of @Conditional annotations? How are they used? 28](#_Toc13531316)

[Spring Boot Testing 28](#_Toc13531317)

[@RunWith(SpringRunner.class) 28](#_Toc13531318)

[You can use various webEnvironment values based on how you want to run the tests. 30](#_Toc13531319)

[When do you want to use @SpringBootTest annotation? 30](#_Toc13531320)

[What does @SpringBootTest auto-configure? 30](#_Toc13531321)

[What dependencies does spring-boot-starter-test brings to the classpath? 30](#_Toc13531322)

[How do you perform integration testing with @SpringBootTest for a web application? 30](#_Toc13531323)

[When do you want to use @WebMvcTest? What does it auto-configure? 30](#_Toc13531324)

[What are the differences between @MockBean and @Mock? 30](#_Toc13531325)

[When do you want @DataJpaTest for? What does it auto-configure? 30](#_Toc13531326)

[Others 34](#_Toc13531327)

# Spring Beans and IOC Container

## PROTOTYPE Scope



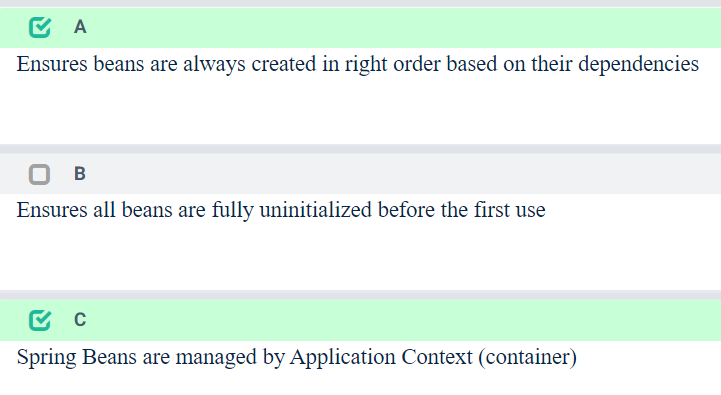


## During which phase can BeanPostProcessor modify the bean initialization?

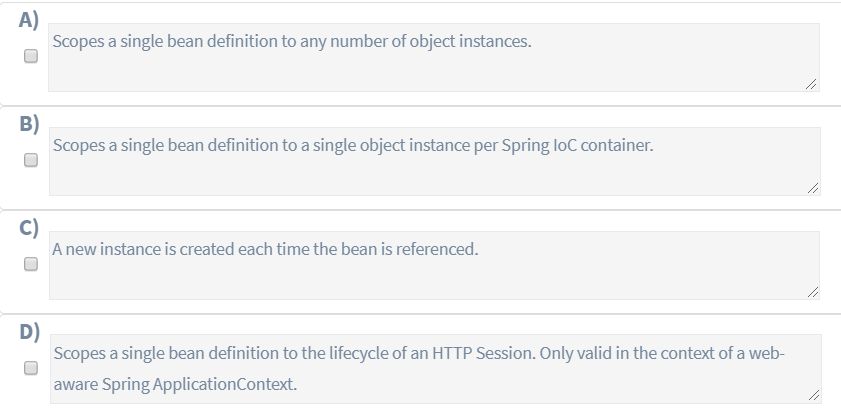
**A. BeforeInit B.AfterInit C.PostConstruct D.A&B**

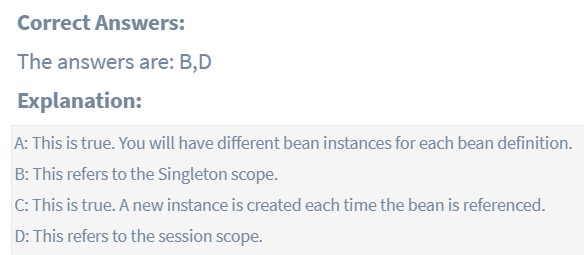
# @SpringBootApplication annotation equivalent to @ComponentScan, @Configuration @EnableAutoConfiguration

# What is the default behavior of Spring application context?



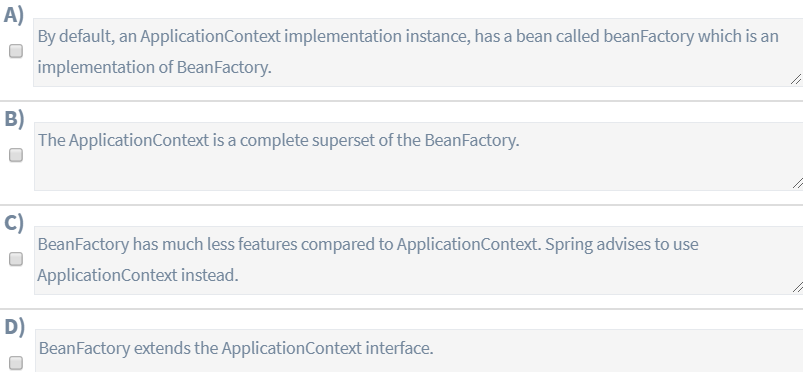
## Which is False about Prototype scope?

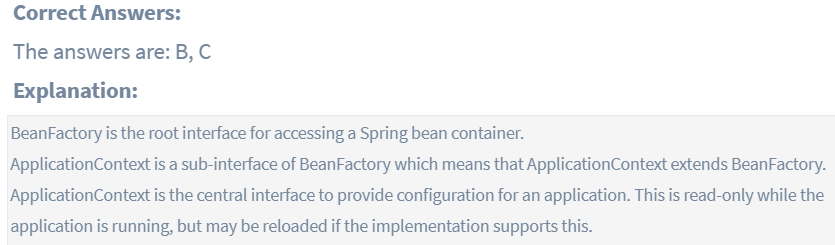




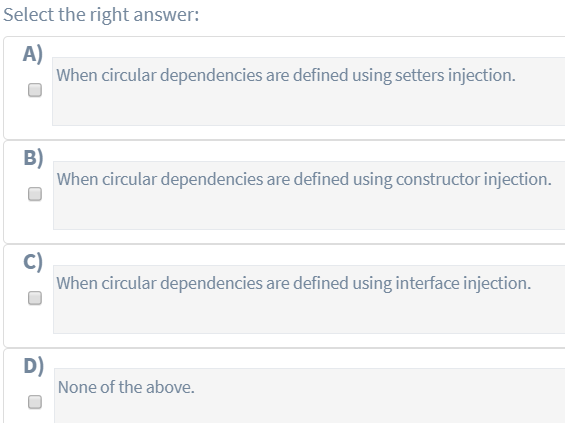
## ApplicationContext and BeanFactory

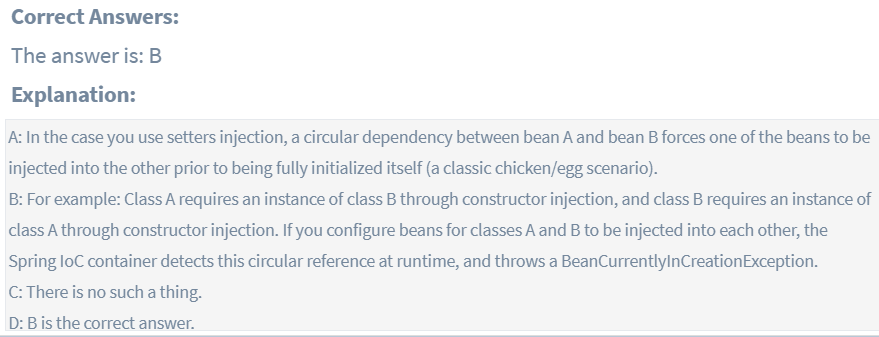






## BeanCurrentlyInCreationException

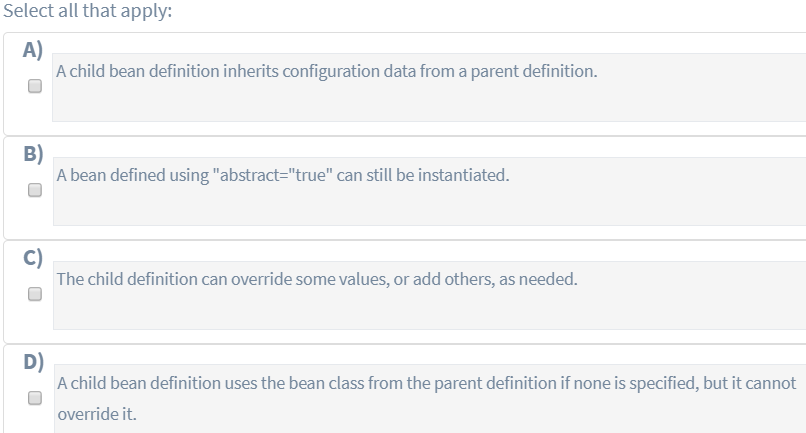


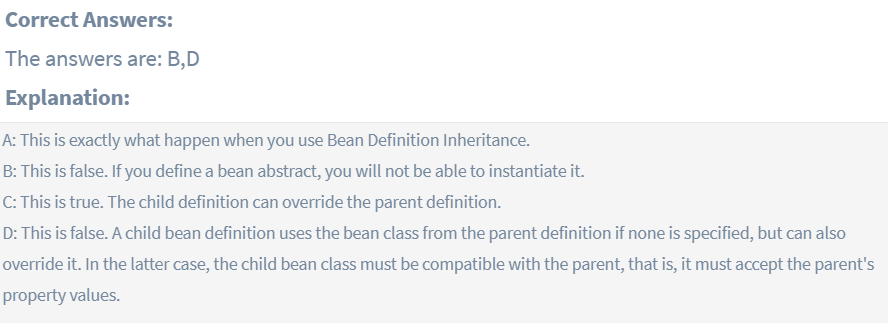


## Valid NameSpaces

## Example of Bean Definition inheritance

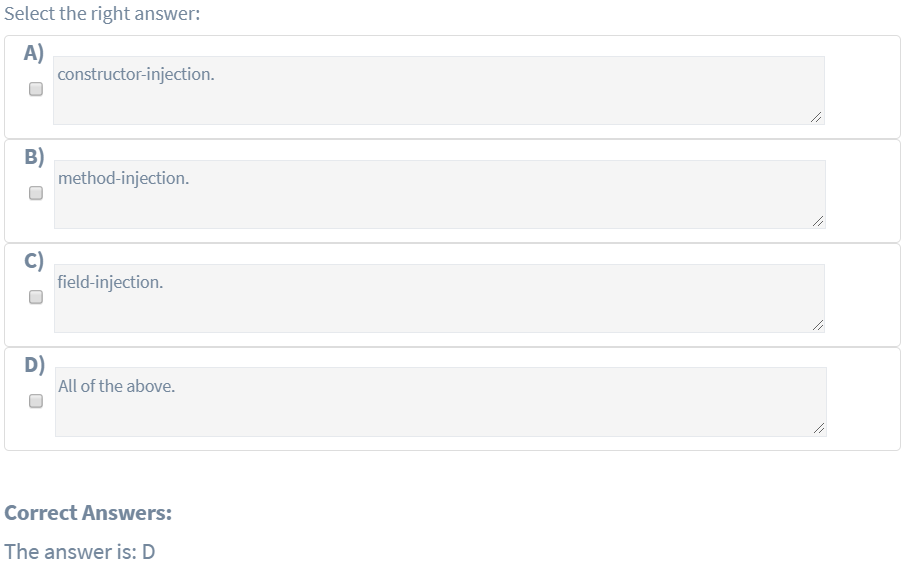


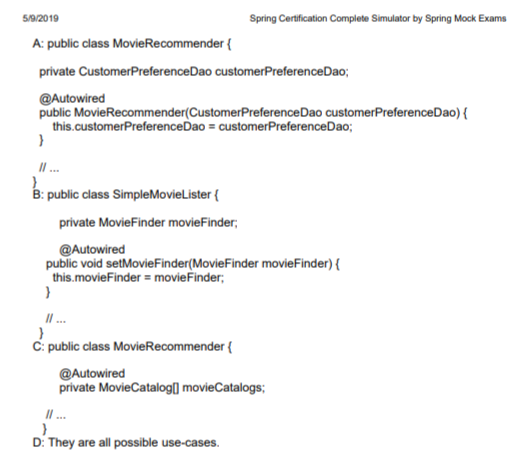




## @Autowire

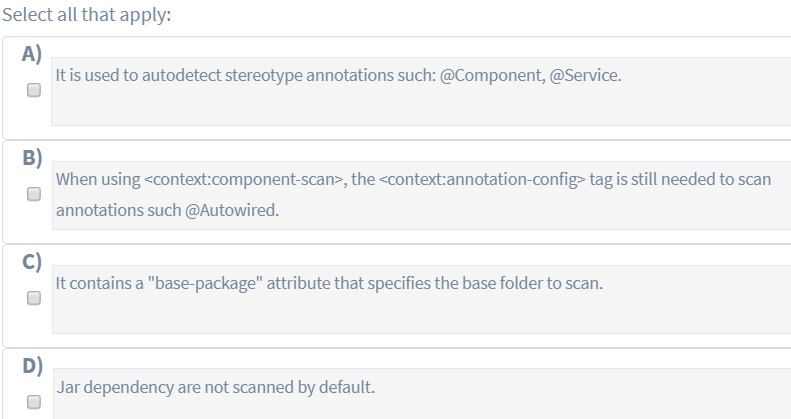


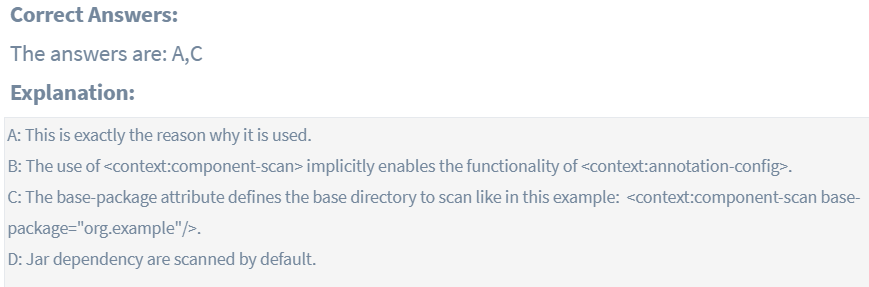




## <context:component-scan>

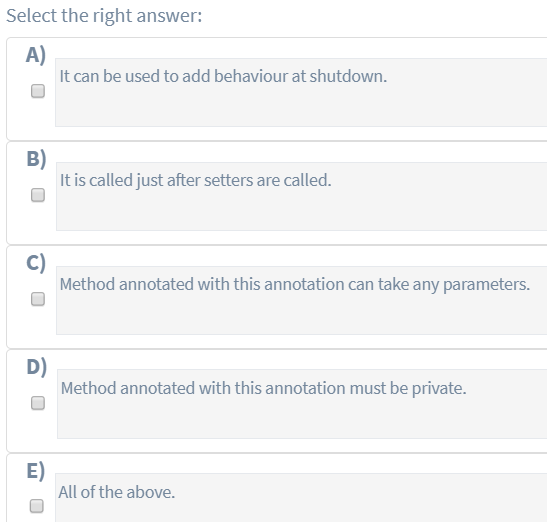


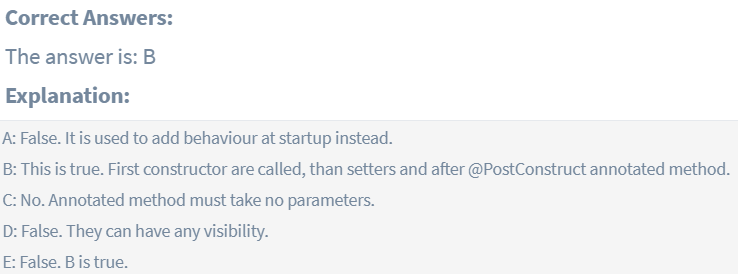




## @PostConstruct

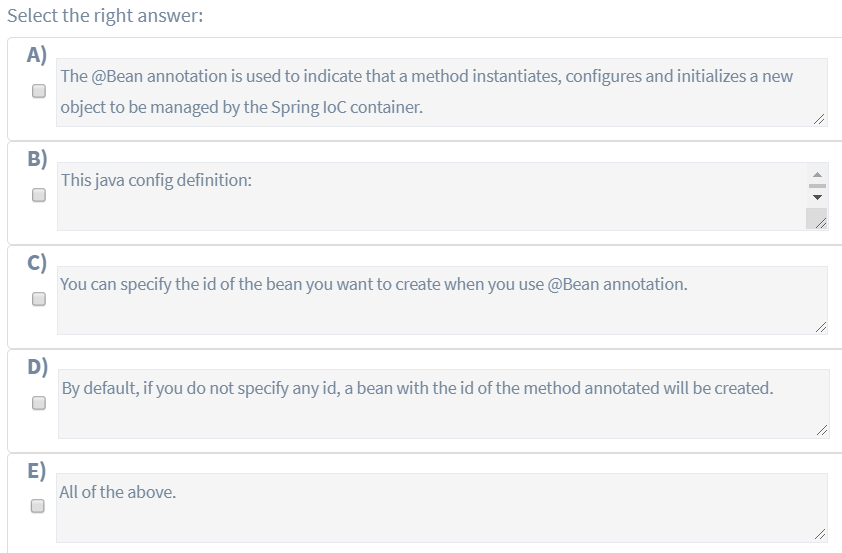


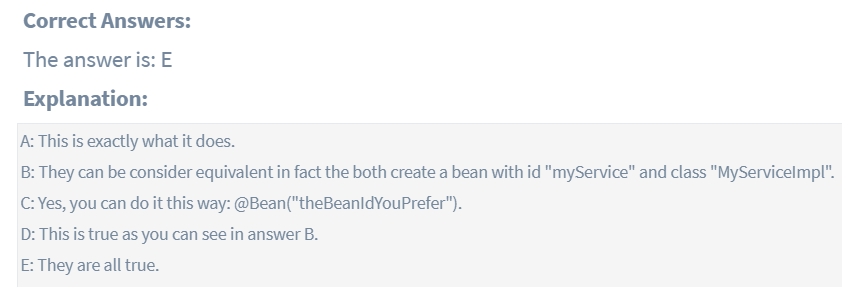




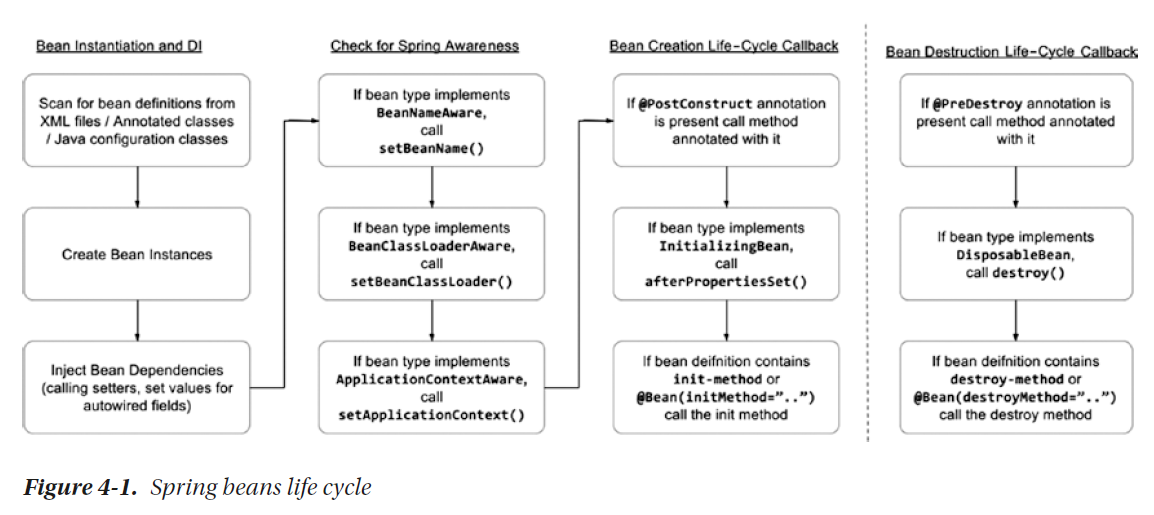
## @Bean Annotation







## Spring bean lifecycle



# Example of BeanNameAware in Spring.

On this page we will provide example of BeanNameAware in spring. BeanNameAware belongs to the package org.springframework.beans.factory. BeanNameAware can be used to set bean name. The class has to implement interface BeanNameAware and override its method setBeanName(String beanName). When the bean is loaded by spring container, the bean name is set to this method. Find the example below.

## Implement BeanNameAware

We are creating a bean that is implementing BeanNameAware interface and overriding method setBeanName().

## Difference between BeanNameAware****and****BeanFactoryAware****interfaces, in the Spring Framework****.

BeanFactoryAware is used to inject the BeanFactory object. This way we get access to the BeanFactory which created the object. Here’s an example of a MyBeanFactory class:

public class MyBeanFactory implements BeanFactoryAware {

private BeanFactory beanFactory;

@Override

public void setBeanFactory(BeanFactory beanFactory) throws BeansException {

this.beanFactory = beanFactory;

}

public void getMyBeanName() {

MyBeanName myBeanName = beanFactory.getBean(MyBeanName.class);

System.out.println(beanFactory.isSingleton("myCustomBeanName"));

}

}

With the help of the setBeanFactory() method, we assign the BeanFactory reference from the IoC container to the beanFactory property.

After that, we can use it directly like in the getMyBeanName() function.

Let’s initialize the MyBeanFactory and call the getMyBeanName() method:

MyBeanFactory myBeanFactory = context.getBean(MyBeanFactory.class);

myBeanFactory.getMyBeanName();

As we have already instantiated the MyBeanName class in the previous example, Spring will invoke the existing instance here. The beanFactory.isSingleton(“myCustomBeanName”) line verifies that.

<https://www.baeldung.com/spring-bean-name-factory-aware>

# Example of ApplicationContextAware in Spring

<https://www.concretepage.com/spring/example_applicationcontextaware_spring>

# What is Need for Using ApplicationContextAware in Application?

<https://www.dineshonjava.com/using-applicationcontextaware-in-spring/>

# Bean Creation Life-Cycle Callback.

## @PostConstruct

## Implement InitializingBean interface( @override afterPropertiesSet)

## init-method or @Bean(initMethod=”..”)

# Bean Destruction Life-Cycle Callback

## @PostConstruct

## Implement DisposableBean interface( @override destroy())

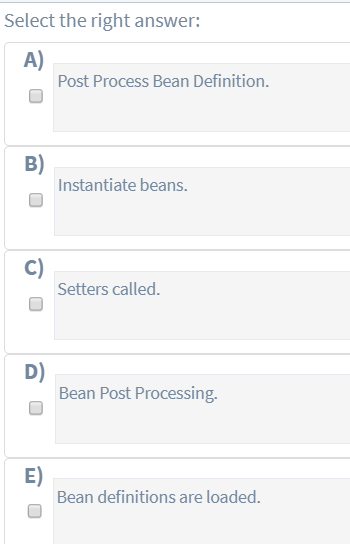
## Destroy-method or @Bean(destroyMethod=”..”)

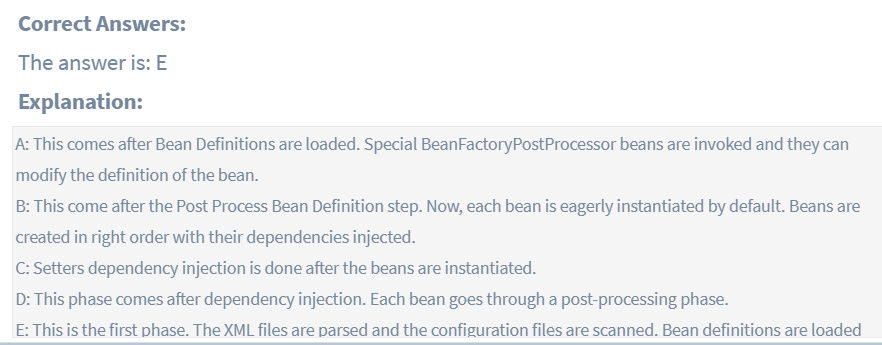
<https://www.baeldung.com/spring-postconstruct-predestroy>

# I would not recommend to use InitializingBean and DisposableBean interface, because it will tight coupled your code to Spring. A better approach should be specifying the [init-method and destroy-method](http://www.mkyong.com/spring/spring-init-method-and-destroy-method-example/) attributes in your bean configuration file.

<https://www.mkyong.com/spring/spring-initializingbean-and-disposablebean-example/>

# What is the first step of the initialization phase of the Spring application lifecycle?





# BeanPostProcessor interface.

**BeanPostProcessor** is used to perform some operations before and after creating a bean,this allows us to add some code before and after creating the bean.

**BeanPostProcessor** is applicable for all the beans, which means its methods will be executed for each bean we define in the xml.

We can use the **BeanPostProcessor** to execute some logic for all the beans in the application context before and after their initialization

**BeanPostProcessor** interface has 2 methods **postProcessBeforeInitialization()** and **postProcessAfterInitialization()** where former is called after the bean is created and before it is initialized And the latter is called after the bean initialization

**Create a class which implements BeanPostProcessor**

1. **package** com.kb.beans;
3. **import** org.springframework.beans.BeansException;
4. **import** org.springframework.beans.factory.config.BeanPostProcessor;
6. **public** **class** MyBeanPostProcessor **implements** BeanPostProcessor {
8. **public** Object postProcessAfterInitialization(Object bean, String beanName) **throws** BeansException {
9. System.out.println("Post Process After Initialization for the bean "+beanName);
10. **return** bean;
11. }
13. **public** Object postProcessBeforeInitialization(Object bean, String beanName) **throws** BeansException {
14. System.out.println("Post Process Before Initialization for the bean "+beanName);
15. **return** bean;
16. }
17. }

Now create 2 classes and make them as spring beans so that we can see how the **BeanPostProcessor** is executed for both beans

**Create a class called Person**

[Copy this code](javascript:;)

1. **package** com.kb.beans;
3. **import** org.springframework.beans.BeansException;
4. **import** org.springframework.beans.factory.config.BeanPostProcessor;
6. **public** **class** Person {
8. **public** Person() {
9. System.out.println("Person - no argument constructor");
10. }
11. **private** String name;
13. **public** **void** setName(String name) {
14. **this**.name = name;
15. }
17. **public** String getName() {
18. **return** name;
19. }
20. }

**Create the User class**

[Copy this code](javascript:;)

1. **package** com.kb.beans;
3. **import** org.springframework.beans.BeansException;
4. **import** org.springframework.beans.factory.config.BeanPostProcessor;
6. **public** **class** User {
8. **public** User() {
9. System.out.println("User - no argument constructor");
10. }
12. **private** String name;
14. **public** String getName() {
15. **return** name;
16. }
18. **public** **void** setName(String name) {
19. **this**.name = name;
20. }
21. }

**Create the spring configuration file**

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="<http://www.springframework.org/schema/beans>"
3. xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"
4. xmlns:context="<http://www.springframework.org/schema/context>"
5. xmlns:p="<http://www.springframework.org/schema/p>"
6. xsi:schemaLocation="<http://www.springframework.org/schema/beans>
7. <http://www.springframework.org/schema/beans/spring-beans-3.2.xsd>
8. <http://www.springframework.org/schema/context>
9. <http://www.springframework.org/schema/context/spring-context-3.2.xsd>">
11. <bean id="person" **class**="com.kb.beans.Person">
12. <property name="name" value="Raj"/>
13. </bean>
15. <bean id="user" **class**="com.kb.beans.User">
16. <property name="name" value="Ram"/>
17. </bean>
19. <!-- **This** post processor methods are called **for** every bean -->
20. <bean **class**="com.kb.beans.MyBeanPostProcessor"/>
21. </beans>

**Create pom.xml file as below**

1. <project xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"
2. xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <http://maven.apache.org/xsd/maven-4.0.0.xsd>">
3. <modelVersion>4.0.0</modelVersion>
5. <groupId>Spring</groupId>
6. <artifactId>SpringCore</artifactId>
7. <version>0.0.1-SNAPSHOT</version>
8. <packaging>jar</packaging>
10. <name>SpringCore</name>
11. <url>http:*//maven.apache.org</url>*
13. <properties>
14. <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15. </properties>
17. <dependencies>
18. <dependency>
19. <groupId>org.springframework</groupId>
20. <artifactId>spring-core</artifactId>
21. <version>4.2.4.RELEASE</version>
22. </dependency>
23. <dependency>
24. <groupId>org.springframework</groupId>
25. <artifactId>spring-context</artifactId>
26. <version>4.2.4.RELEASE</version>
27. </dependency>
29. <dependency>
30. <groupId>junit</groupId>
31. <artifactId>junit</artifactId>
32. <version>3.8.1</version>
33. <scope>test</scope>
34. </dependency>
35. </dependencies>
36. </project>

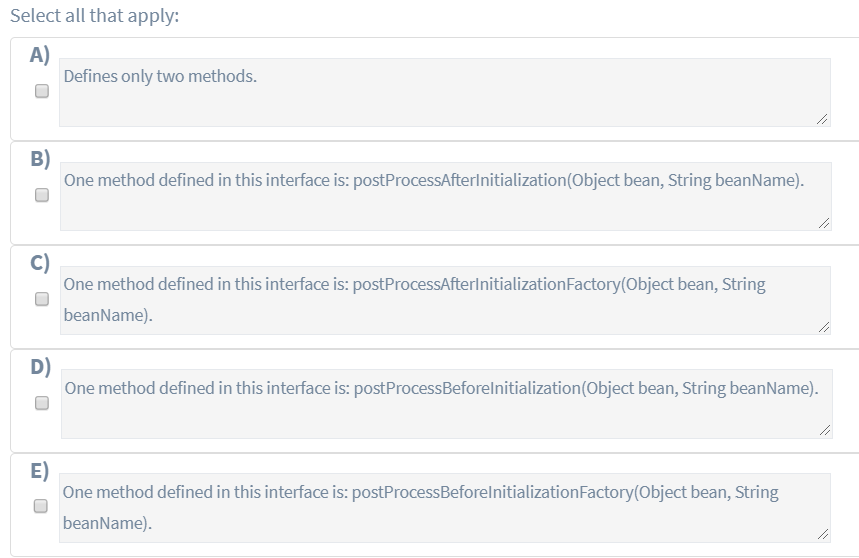
**Create a BeansLifeCycle class as below**

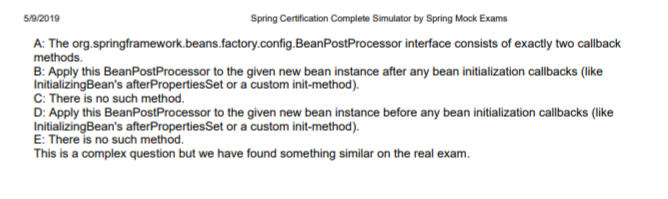
1. **package** com.kb.beans;
3. **import** org.springframework.context.ApplicationContext;
4. **import** org.springframework.context.support.ClassPathXmlApplicationContext;

7. **public** **class** BeansLifeCycle {
9. **public** **static** **void** main(String[] args) {
11. ClassPathXmlApplicationContext applicationContext=**new** ClassPathXmlApplicationContext("beans.xml");
13. *//Load Person object*
14. Person person = (Person)applicationContext.getBean("person");
15. System.out.println("Name : "+person.getName());
17. *//Load User object*
18. User user = (User)applicationContext.getBean("user");
19. System.out.println("Name : "+user.getName());
21. applicationContext.registerShutdownHook();
23. }
24. }

**Run this class and see the output as below**  
  
INFO: Loading XML bean definitions from class path resource [beans.xml]  
Person – no argument constructor  
Post Process Before Initialization for the bean person  
Post Process After Initialization for the bean person  
User – no argument constructor  
Post Process Before Initialization for the bean user  
Post Process After Initialization for the bean user  
Name : Raj  
Name : Ram







# Difference between BeanPostProcessor and @PostConstruct, InitializingBean and custom init method.

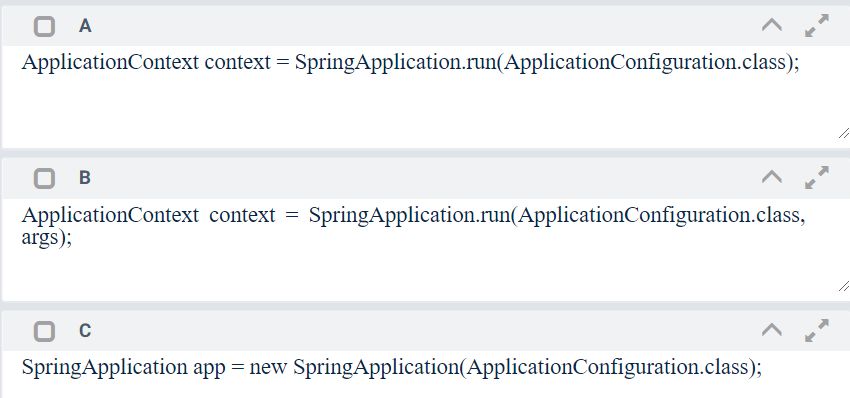
The BeanPostProcessor interface defines callback methods that you can implement to provide your own (or override the container's default) instantiation logic, dependency-resolution logic, and so forth. If you want to implement some custom logic after the Spring container finishes instantiating, configuring, and initializing a bean, you can plug in one or more BeanPostProcessor implementations.

So in essence the method postProcessBeforeInitialization defined in the BeanPostProcessor gets called (as the name indicates) before the initialization of beans and likewise the postProcessAfterInitialization gets called after the initialization of the bean.

The difference to the @PostConstruct, InitializingBean and custom init method is that these are defined on the bean itself. Their ordering can be found in the [Combining lifecycle mechanisms](http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/htmlsingle/spring-framework-reference.html#beans-factory-lifecycle-combined-effects)section of the spring documentation.

So basically the BeanPostProcessor can be used to do custom instantiation logic for several beans where as the others are defined on a per bean basis.

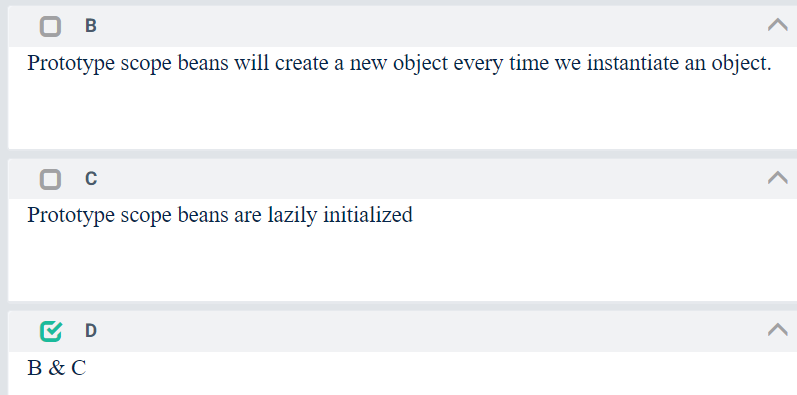
# Create spring application context?



ApplicationContext appContext = new AnnotationConfigApplicationContext(AnnotationJavaConfig.class);

ApplicationContext appContext = new FileSystemXmlApplicationContext();

# Which beans are initialized lazily and eagerly?

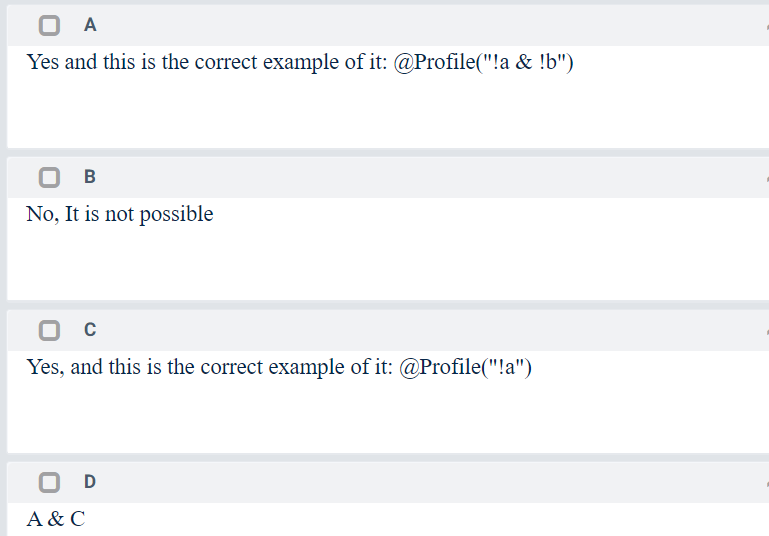


# @Profile Usecase

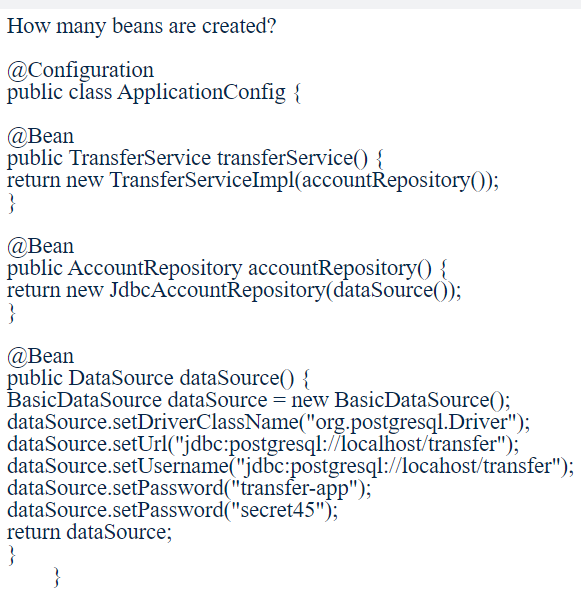
## Is it possible to use a profile expression inside, profile string annotation? Answer : D

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-profiles.html>

<https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/context/annotation/Profile.html>

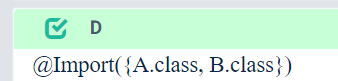


## How Many beans will be created? Remember ApplicationConfig is also a spring bean. It is used to create the others. Answer is 4 beans.

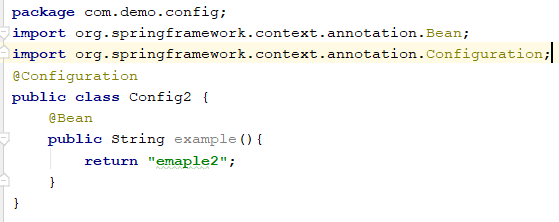
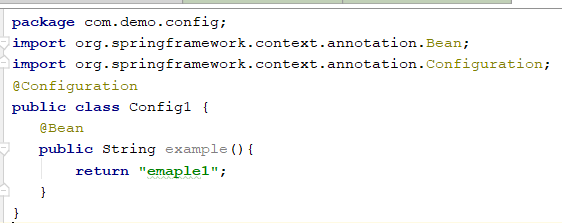


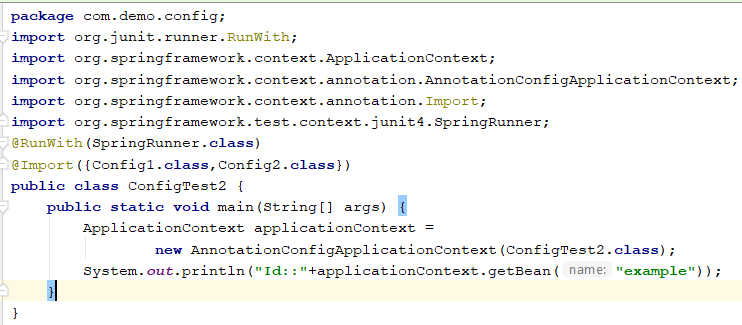
## How can you create an application context from multiple files, which annotation can used to combined

## multiple configuration files?



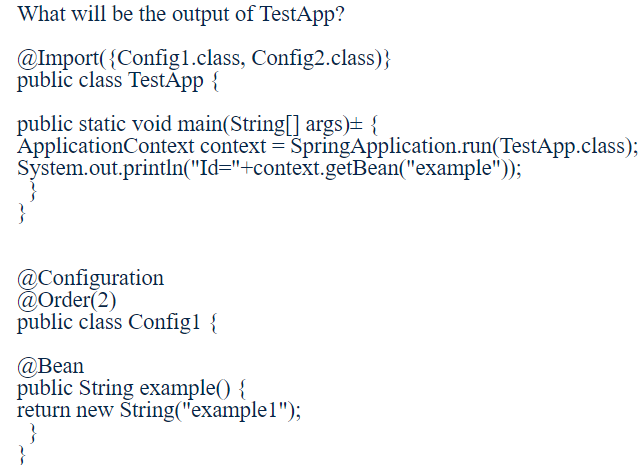
## It is not illegal to define the same bean more than once, but the you get the last bean Spring sees defined. Each bean have a unique id so the previous bean is replaced by config2 class method.





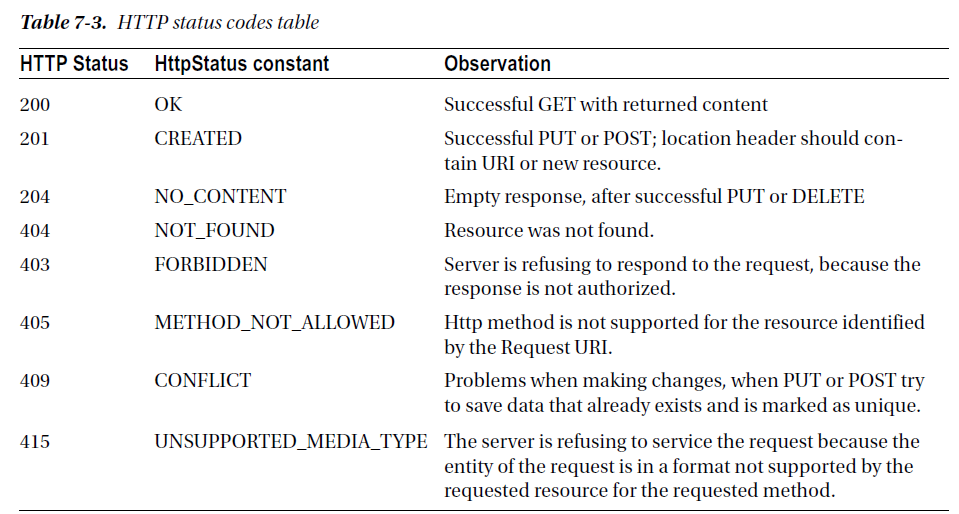


## @Order annotation.

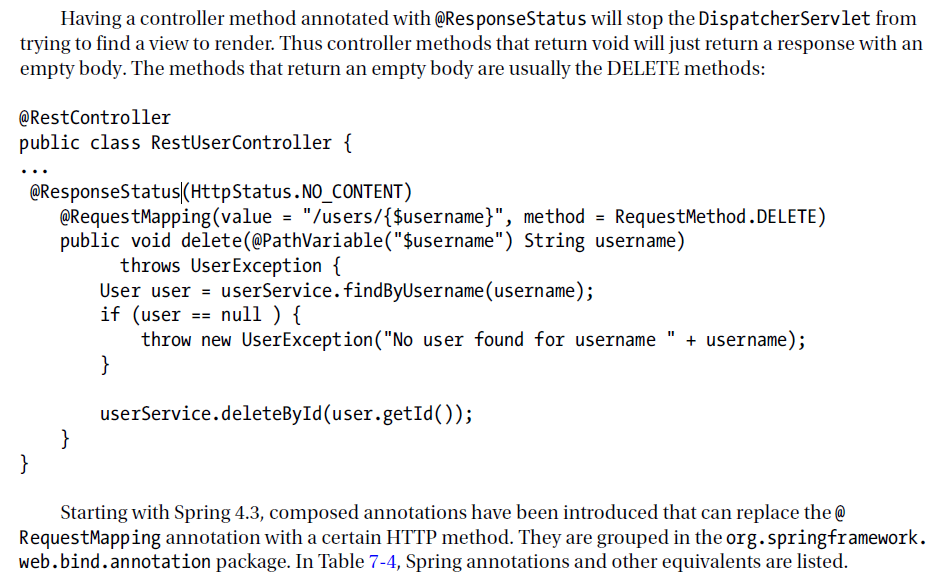


# Rest

## HTTP Status Code



## Having a controller method annotated with @ResponseStatus will stop the DispatcherServlet from trying to find a view to render.



## HATEOAS (Hyper media as the engine of application state)

When you need to provide additional resources on response use HATEOAS.

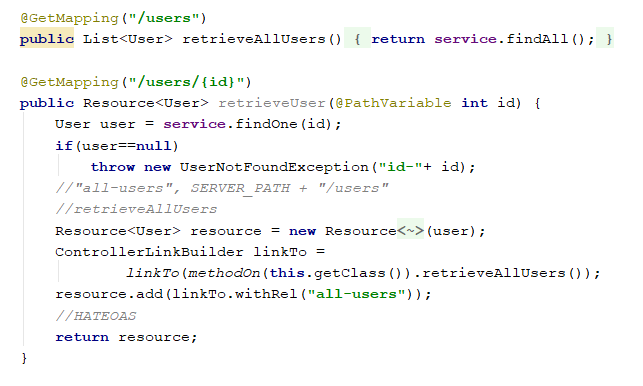
Step 1- Add dependency – spring-boot-starter-hateoas

This dependency will help to add link on resources.

Step 2- Let say we want to add the link of all users in get particular users

### Resource<User> resource = new Resource<User>(user); ControllerLinkBuilder linkTo = linkTo(methodOn(this.getClass()).retrieveAllUsers()); resource.add(linkTo.withRel("all-users")); //HATEOAS return resource;

**import static** org.springframework.hateoas.mvc.ControllerLinkBuilder.*linkTo*;  
**import static** org.springframework.hateoas.mvc.ControllerLinkBuilder.*methodOn*;  
**import** org.springframework.hateoas.Resource;  
**import** org.springframework.hateoas.mvc.ControllerLinkBuilder;

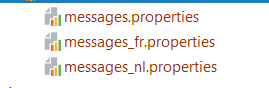


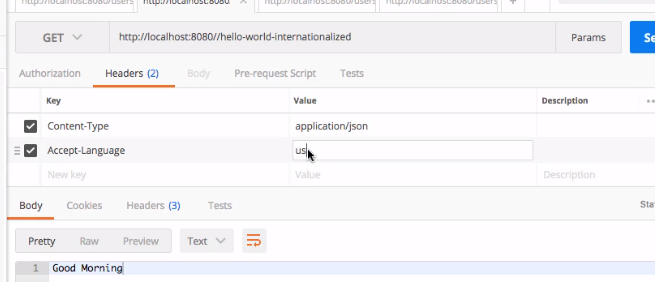
## How to implement i18n

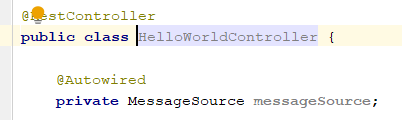
Step 1- Configure 2 beans LocaleResolver bean and ResouceBundleMessageSource

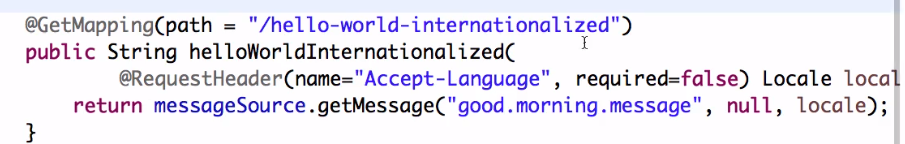


Create msg properties like below



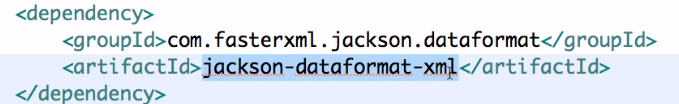






## Content Negotiation

### Add jackson-dataformat-xml to convert json to xml.

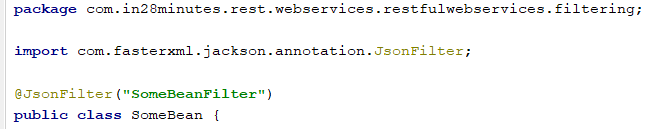


## What is Static filtering and what is dynamic filtering?

For Static filtering we annotated particular field as @JasonIgnore but in Dynamic filtering we can use SimpleBeanPropertyFilter, FilterProvider and MappingJacksonValue.

You need to provide a filter name and then annotated it

***@JsonFilter("SomeBeanFilter")***



### SimpleBeanPropertyFilter filter = SimpleBeanPropertyFilter.filterOutAllExcept("field1", "field2"); FilterProvider filters = new SimpleFilterProvider().addFilter("SomeBeanFilter", filter); MappingJacksonValue mapping = new MappingJacksonValue(someBean); mapping.setFilters(filters); return mapping;

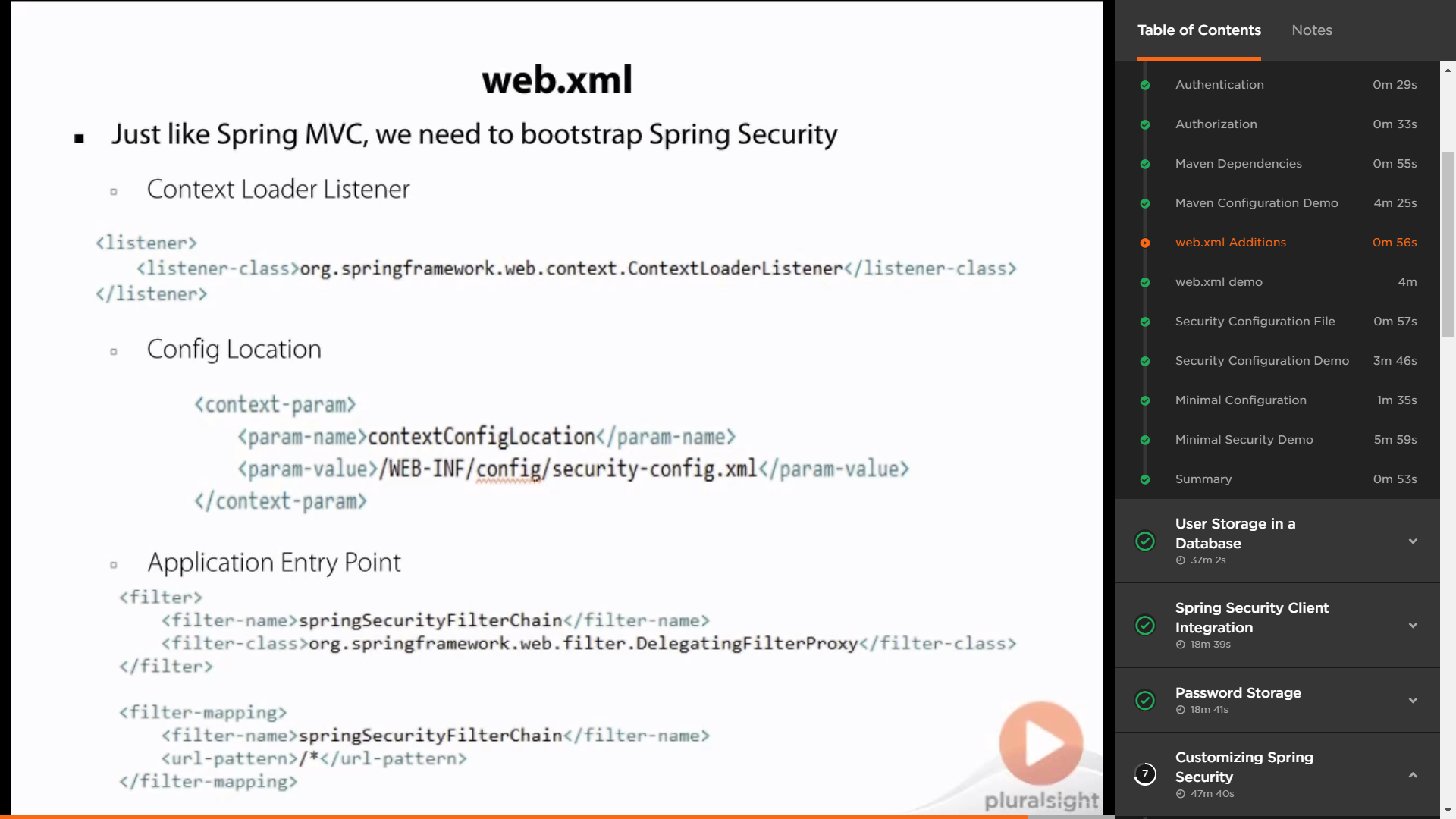
# Spring Security Configuration

To configure Spring security, the developer must take care of three things:

declare the security filter for the application

define the Spring Security context

configure authentication and authorization



web.xml Additions

Just like in Spring MVC, we need to bootstrap our context. There are a couple of things that we need to configure when integrating security into our application. The first is a context loader listener, which is defined by adding this XML snippet to your web. xml. The second is the config location, and the config location tells your app where the security configuration file is located at. So from this XML snippet you can see that we have a security-config file that's located underneath our WEB-INF config directory. Lastly is the application entry point, which simply defines a filter that routes all requests through Spring Security so that they can then be evaluated in our security configuration.

web.xml demo

Let's go ahead and open up our src, main, webapp, WEB-INF, web. xml. I'm going to make it full screen so you can see everything that's going on in here. And we need to add those three pieces to our web. xml that we just talked about. First thing we want to add is our filter. So we're going to define a filter, and this filter is going to have a filter-name associated with it. And we can name this whatever we want, but it has to just be the same between our filter definition and our filter mapping. So we're going to do springSecurityFilterChain, and then we want to add a class. So I'll do filter-class, and this is org. springframework. web. filter. DelegatingFilterProxy. I can go ahead and save that. So that's the first part of the filter, now we need to add the filter mapping. And the filter mapping just routes all URLs to go through our Spring Security filter. So inside of here we're going to do filter-mapping, and make sure it's below your filter and not inside the filter element, and we need two things in here. We need the filter-name and that will just be the same as up above here, the springSecurityFilterChain, and then we want a URL pattern. And our URL pattern is going to be very straightforward, it's just /\*. So, we are going to send every request through this filter mapping. Now we need to add two more pieces. We need a listener, and the listener is what's going to bootstrap our Spring Security configuration for us. And this has a listener class associated with it. And this class is org. springframework. web. context. ContextLoaderListener. And then we want to add, last but not least, the context-params. And these are used, these context-params are used by this listener to find the locations. So we want to do param-name, and that is going to be, and this has to be named contextConfigLocation because that's what this ContextLoaderListener is looking for, a context-param name contextConfigLocation. And then we want to give this a param-value of /WEB-INF, and make sure you spell it, it is case sensitive, we want config/security-config. xml. Now if you try to run the app right now, it'll break because we have not created this file yet, that's the next step that we're going to discuss about what that file is and what goes into that file. So now our web. xml is configured. So we've got all of our JARs downloaded, we have our web. xml configured to where it's going to load our security and point it to our security configuration file. We're just about done with what we need to actually do as far as configuring our application to start security and start routing this through our application. So let's save that and in the next steps we'll go through and build the security config file.

Security Configuration File

To keep with a separation of concerns design pattern, we define our security configuration in a file separate of our web configuration. We want to add another XML application context, and that can be under our src/main/resources or our src/main/webapp/WEB-INF/config directory. Like other Spring configuration files, this has a namespace associated with it. This is the namespace declaration that you should add to the top of this config file for easier XML configuration. Now two little secrets here. If you are using Spring STS, it makes adding that namespace definition a lot simpler. You can just click on the tab at the bottom and it will automatically put these headers in our file for us.

## What are the class required to apply spring security as java configuration.

Java Configuration

To develop a working configuration for a Spring Security web application, the XML configuration must be

transformed into a security configuration class. The class that replaces the Spring XML configuration should

extend WebSecurityConfigurerAdapter, so the amount of code needed to be written for a valid security

configuration should be minimal. Thus, the XML configuration given as an example so far becomes:

package com.pr.config;

...

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation

.authentication.builders.AuthenticationManagerBuilder;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) {

try {

auth.inMemoryAuthentication()

.withUser("john").password("doe").roles("USER").and()

.withUser("jane").password("doe").roles("USER","ADMIN").and()

.withUser("admin").password("admin").roles("ADMIN");

} catch (Exception e) {

e.printStackTrace();

}

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/user/edit").hasRole("ADMIN")

.antMatchers("/\*\*").hasAnyRole("ADMIN","USER")

.anyRequest()

.authenticated()

.and()

.formLogin()

.usernameParameter("username") // customizable

.passwordParameter("password") // customizable

.loginProcessingUrl("/login") // customizable

.loginPage("/auth")

.failureUrl("/auth?auth\_error=1")

.defaultSuccessUrl("/home")

.permitAll()

.and()

.logout()

.logoutUrl("/logout")

.logoutSuccessUrl("/")

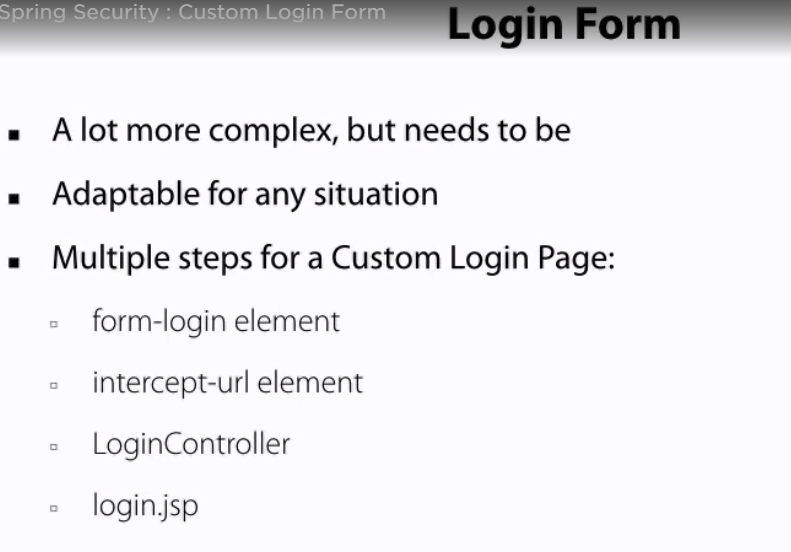
.and()

.csrf().disable();

}

}

## Create custom login page.



# Spring Boot Intro.

The primary goal of Spring Boot is to make it easy to develop Spring-based applications. Spring Boot

provides several features to implement commonly used features, such as logging and externalizing

configuration properties in a much easier way.

## Logging

## Using the @SpringBootApplication Annotation

## • What is Spring Boot?

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

## • Why is it “opinionated”?

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

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

## • Spring Boot supports both properties and YML files.

## Would you recognize and understand them if you saw them?

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

## • Where does Spring Boot look for property file by default?

<https://docs.spring.io/spring-boot/docs/current/reference/html/howto-properties-and-configuration.html>

## • How do you define profile specific property files?

## • How do you access the properties defined in the property files?

## • What properties do you have to define in order to configure external MySQL?

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-sql.html>

## • How do you configure default schema and initial data?

<https://www.baeldung.com/spring-boot-data-sql-and-schema-sql>

## • What is a fat far? How is it different from the original jar?

Fat JAR Using the Spring Boot Maven Plugin

You can run your application directly from the IDE or use Maven spring-boot:run during development,

but ultimately you need to create a deployment unit that can be run in the production environment without

any IDE support. You can use spring-boot-maven-plugin to create a single deployment unit (a fat JAR) by

executing the following Maven goals.

mvn clean package

Now there are two interesting files in the target directory—springboot-basic-1.0-SNAPSHOT.jar and

springboot-basic-1.0-SNAPSHOT.jar.original. The springboot-basic-1.0-SNAPSHOT.jar.original file

will contain only the compiled classes and classpath resources.

But if you look at springboot-basic-1.0-SNAPSHOT.jar, you find the following:

• Compiled classes of your own source code in src/main/java and static resources

from src/main/resources will be in the BOOT-INF/classes directory

• All the dependent JARs in the BOOT-INF/lib directory

• Classes in the org.springframework.boot.loader package that do the Spring Boot

magic of running the Spring Boot application

You can create self-contained deployment units for JAR-type modules using plugins like mavenshade-

plugin, which packages all the dependent JAR classes into a single JAR file. But Spring Boot

follows a different approach and it allows you to nest JARs directly within your Spring Boot application

JAR file. You can read more about at: http://docs.spring.io/spring-boot/docs/current/reference/

htmlsingle/#executable-jar.

You can run the application using the following command:

java -jar springboot-basic-1.0-SNAPSHOT.jar

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

## • What embedded containers does Spring Boot support?

# Spring Boot Auto Configuration

<https://docs.spring.io/spring-boot/docs/current/reference/html/>

## spring-boot-starter-parent

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.0.BUILD-SNAPSHOT</version>

<relativePath/>

</parent>

## • How does Spring Boot know what to configure?

## • What does @EnableAutoConfiguration do?

## • What does @SpringBootApplication do?

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

## • How are DataSource and JdbcTemplate auto-configured?

## • What is spring.factories file for?

## • How do you customize Spring auto configuration?

## • What are the examples of @Conditional annotations? How are they used?

# Spring Boot Testing

## @RunWith(SpringRunner.class)

One of the key reasons for the popularity of the Spring framework is its great support for testing. Spring

provides SpringRunner, which is a custom JUnit runner helping to load the Spring ApplicationContext by using @ContextConfiguration(classes=AppConfig.class).

A typical Spring unit/integration test is shown in Listing below.

Listing 15-1. Typical Spring JUnit Test

@RunWith(SpringRunner.class)

@ContextConfiguration(classes=AppConfig.class)

public class UserServiceTests

{

@Autowired

UserService userService;

@Test

public void should\_load\_all\_users()

{

List<User> users = userService.getAllUsers();

assertNotNull(users);

assertEquals(10, users.size());

}

}

The Spring Boot application is also nothing but a Spring application, so you can use all of Spring’s

testing features in your Spring Boot applications as well.

However, some of the Spring Boot features, like loading external properties and logging, are available only if you create ApplicationContext using the SpringApplication class, which you’ll typically use in your

entry point class. These additional Spring Boot features won’t be available if you use @ContextConfiguration.

@SpringBootApplication

public class SpringbootTestingDemoApplication

{

public static void main(String[] args)

{

SpringApplication.run(SpringbootTestingDemoApplication.class, args);

}

}

Spring Boot provides the @SpringBootTest annotation, which uses SpringApplication behind the

scenes to load ApplicationContext so that all the Spring Boot features will be available. See Listing 15-2.

***Listing 15-2.*** Typical Spring Boot JUnit Test

@RunWith(SpringRunner.class)

@SpringBootTest

public class SpringbootTestingDemoApplicationTests

{

@Autowired

UserService userService;

@Test

public void should\_load\_all\_users()

{

...

...

}

}

For @SpringBootTest, you can pass Spring configuration classes, Spring bean definition XML files, and

more, but in Spring Boot applications, you’ll typically use the entry point class.

The Spring Boot Test starter spring-boot-starter-test pulls in the JUnit, Spring Test, and Spring Boot

Test modules, along with the following most commonly used mocking and asserting libraries:

• Mockito—A Java mocking framework found at http://site.mockito.org/.

• Hamcrest—A matcher/predicates library for data assertion found at

<http://hamcrest.org/JavaHamcrest/>.

• AssertJ—A fluent assertion library found at https://joel-costigliola.github.io/

assertj/.

• JSONassert—An assertion library for JSON found at https://github.com/

skyscreamer/JSONassert.

• JsonPath—XPath for JSON found at <https://github.com/json-path/JsonPath>.

## You can use various webEnvironment values based on how you want to run the tests.

You can use various webEnvironment values based on how you want to run the tests.

• MOCK (default)—Loads a WebApplicationContext and provides a mock servlet

environment. It will not start an embedded servlet container. If servlet APIs are

not on your classpath, this mode will fall back to creating a regular non-web ApplicationContext.

• RANDOM\_PORT— Loads a ServletWebServerApplicationContext and starts an

embedded servlet container listening on a random available port.

• DEFINED\_PORT—Loads a ServletWebServerApplicationContext and starts an

embedded servlet container listening on a defined port (server.port).

• NONE—Loads an ApplicationContext using SpringApplication but does not

provide a servlet environment.

## When do you want to use @SpringBootTest annotation?

## What does @SpringBootTest auto-configure?

## What dependencies does spring-boot-starter-test brings to the classpath?

## How do you perform integration testing with @SpringBootTest for a web application?

## When do you want to use @WebMvcTest? What does it auto-configure?

## What are the differences between @MockBean and @Mock?

## When do you want @DataJpaTest for? What does it auto-configure?

# Others

