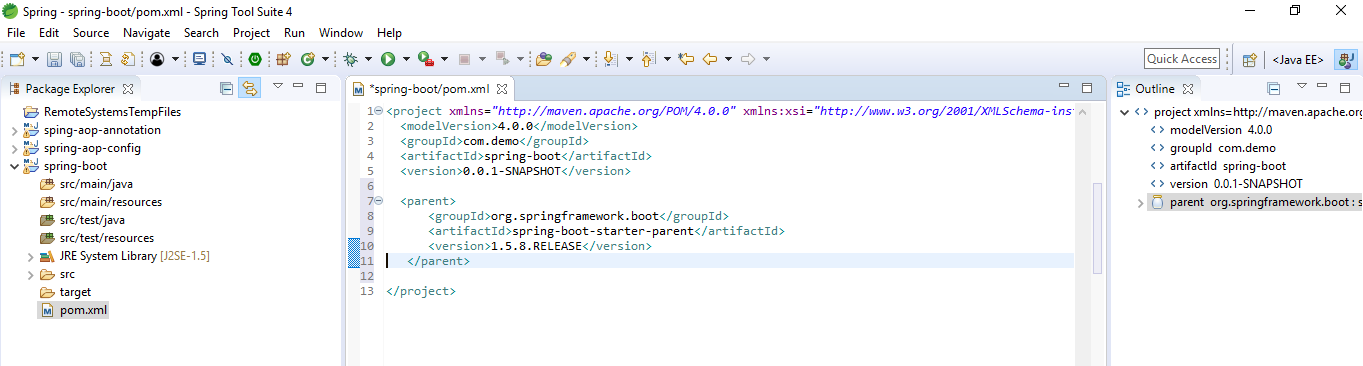
Documentation

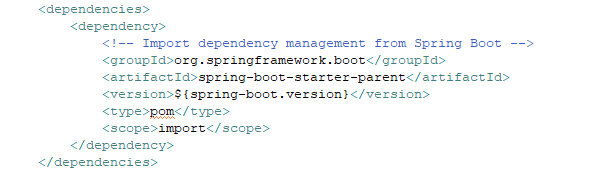
<https://docs.spring.io/spring-boot/docs/current/reference/html/>

Configuration

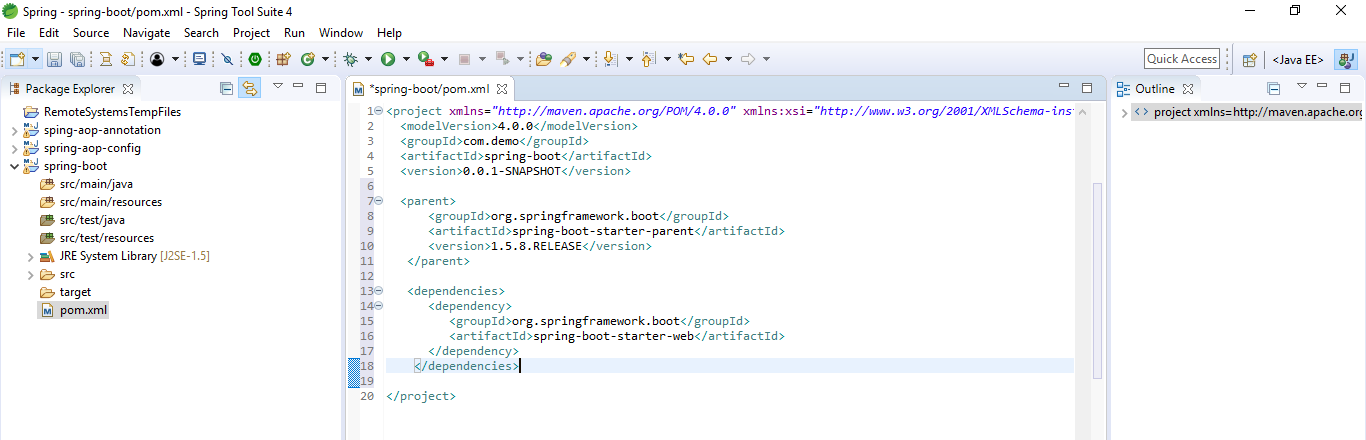
Add Parent proj in pom.xml ( Inherit defaults from Spring boot). It also provides a dependency-management section so that you can omit version tags for “blessed” dependencies.



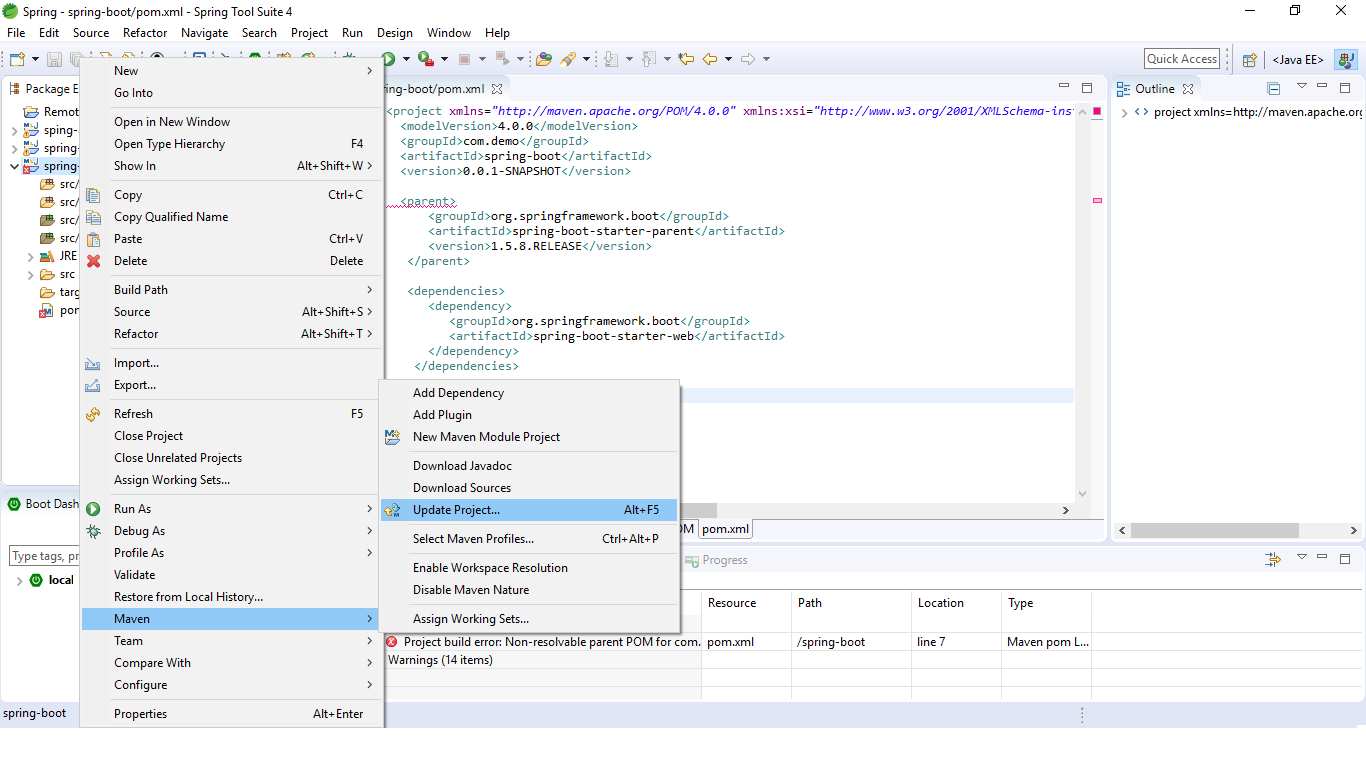
If parent project is something different, configure like below



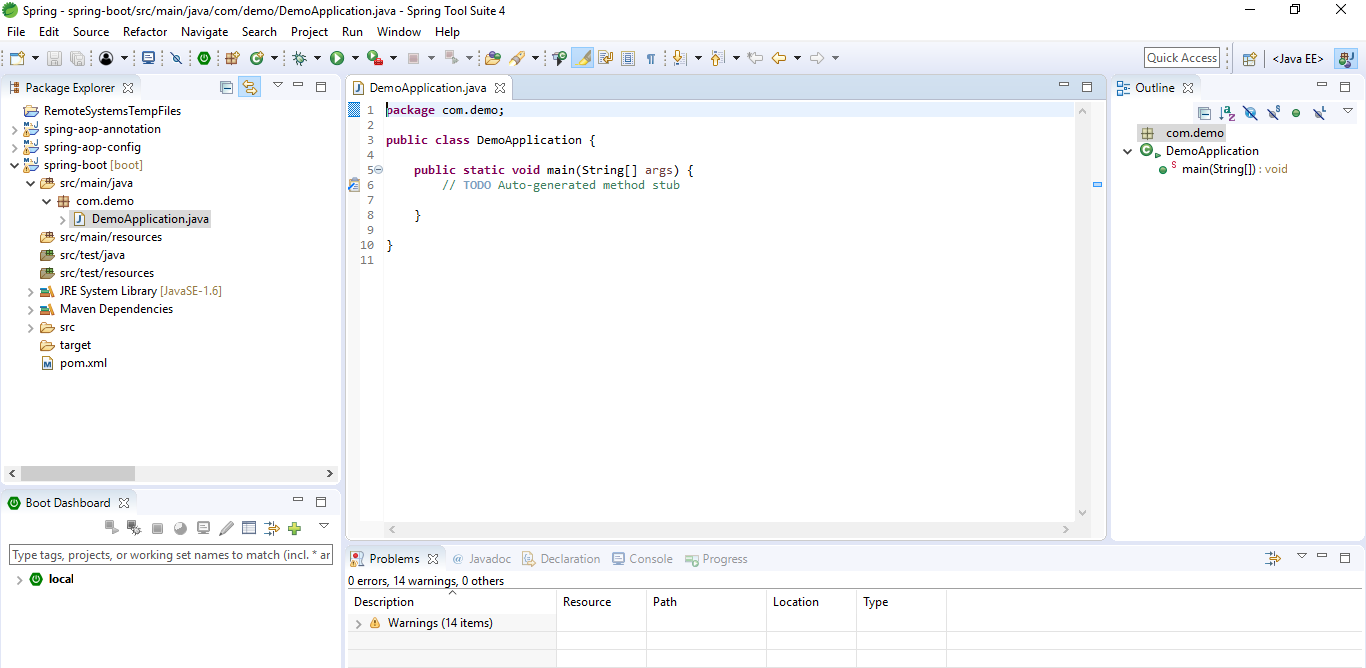
Add the dependency depending on type of application we want to create. For web: starter-web dependency



Once saved it will download jars after updating project



Create a java class with main method – starting point

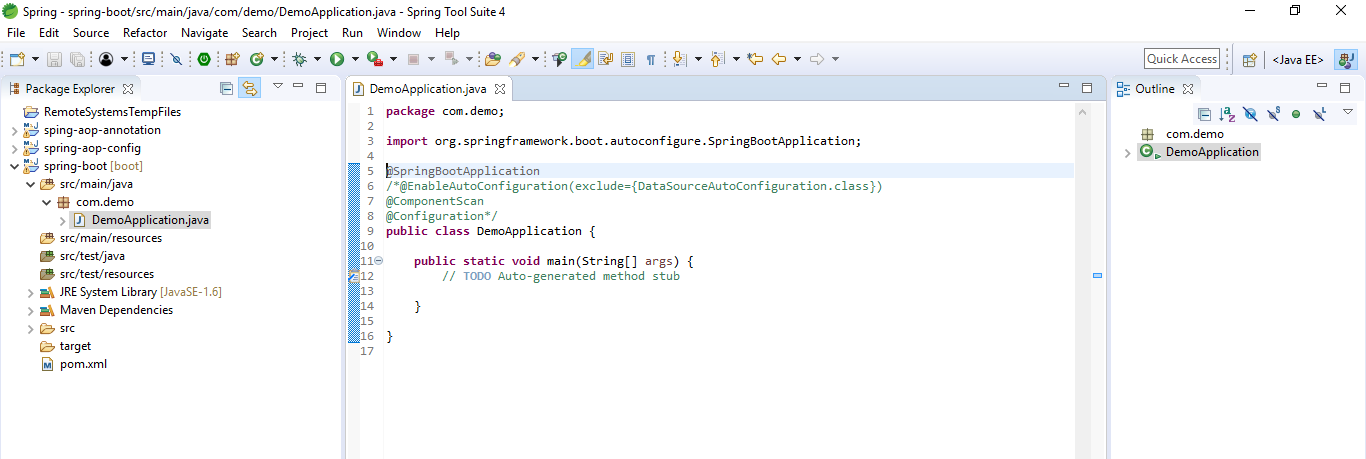


Add @SpringBootApplication – to inform that application is a spring boot application. It is combination of 3 annotations –

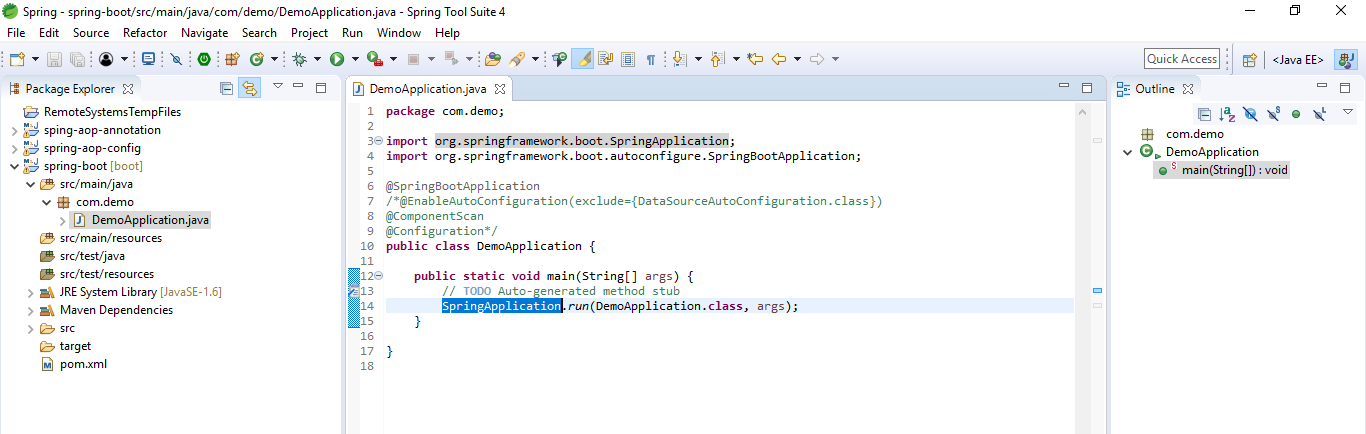
@EnableAutoConfiguration: enable Spring Boot’s auto-configuration mechanism

@ComponentScan: enable @Component scan on the package where the application is located

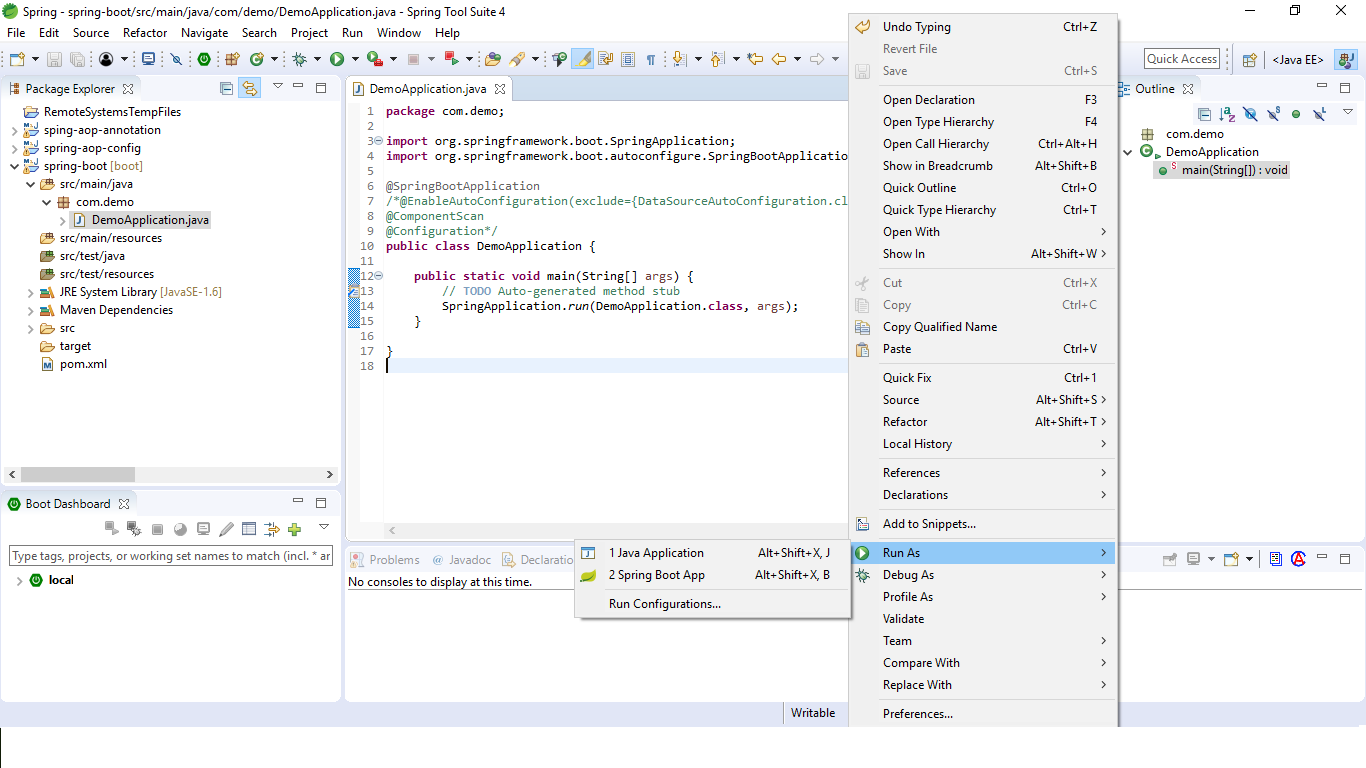
@Configuration: allow to register extra beans in the context or import additional configuration classes



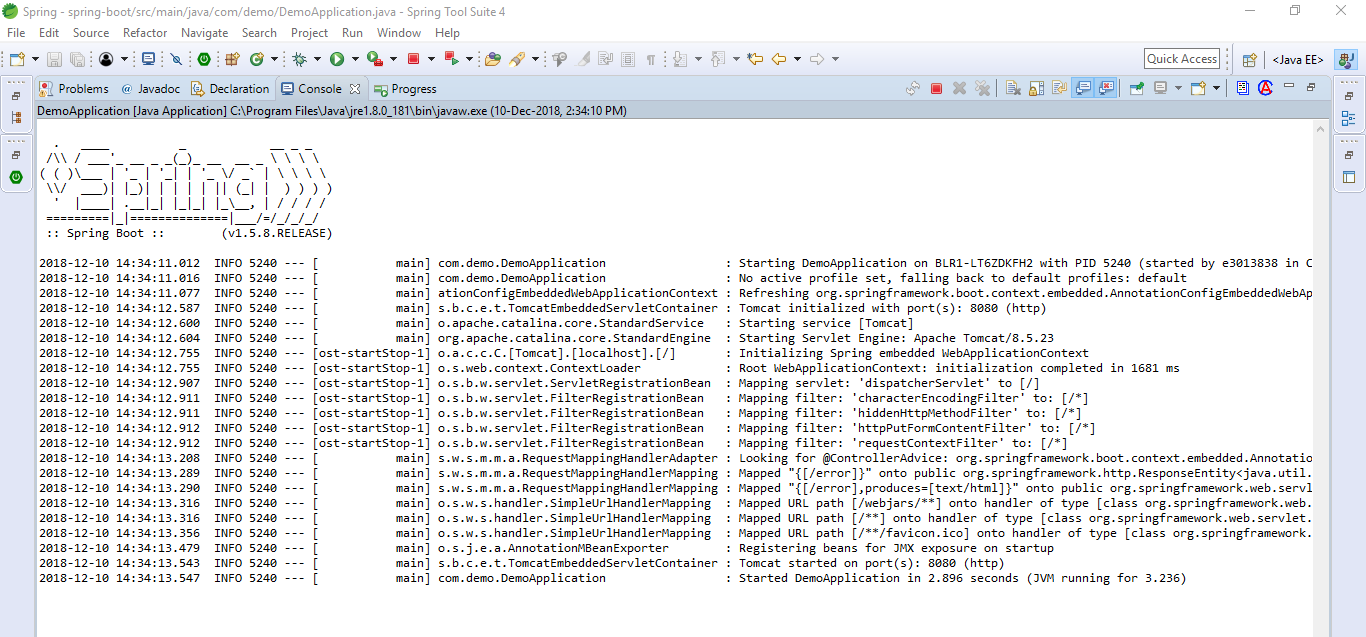
Use SpringApplication.run to run. Args is class that is annoted with @SpringBootApplication



Run and verify



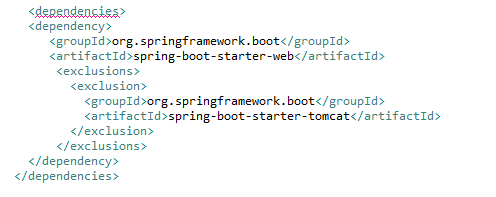
Verify the console



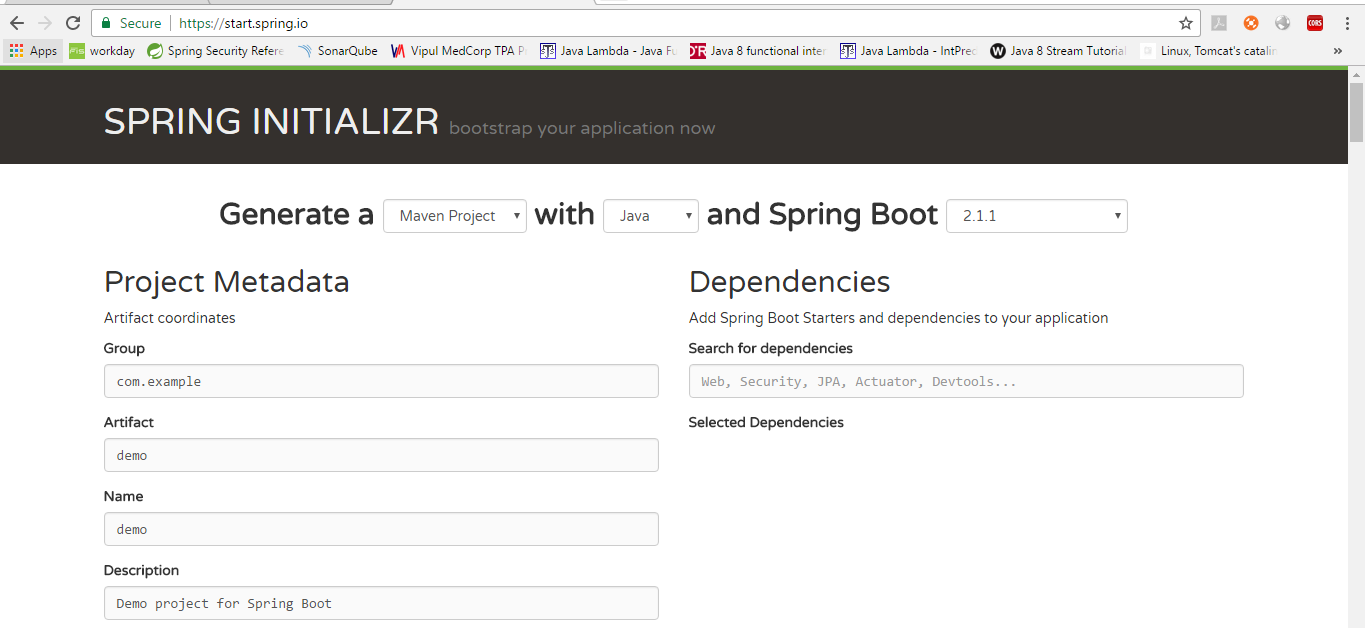
SpringApplication.run

* Create an appropriate ApplicationContext instance (depending on your classpath)
* Register a CommandLinePropertySource to expose command line arguments as Spring properties
* Refresh the application context, loading all singleton beans
* Trigger any CommandLineRunner beans
* Sets up default configuration
* Performs class path scan
* Starts embedded tomcat server

Disable embedded tomcat



Creating Spring boot project



Another option is using Spring boot CLI

You can download the Spring CLI distribution from the Spring software repository:

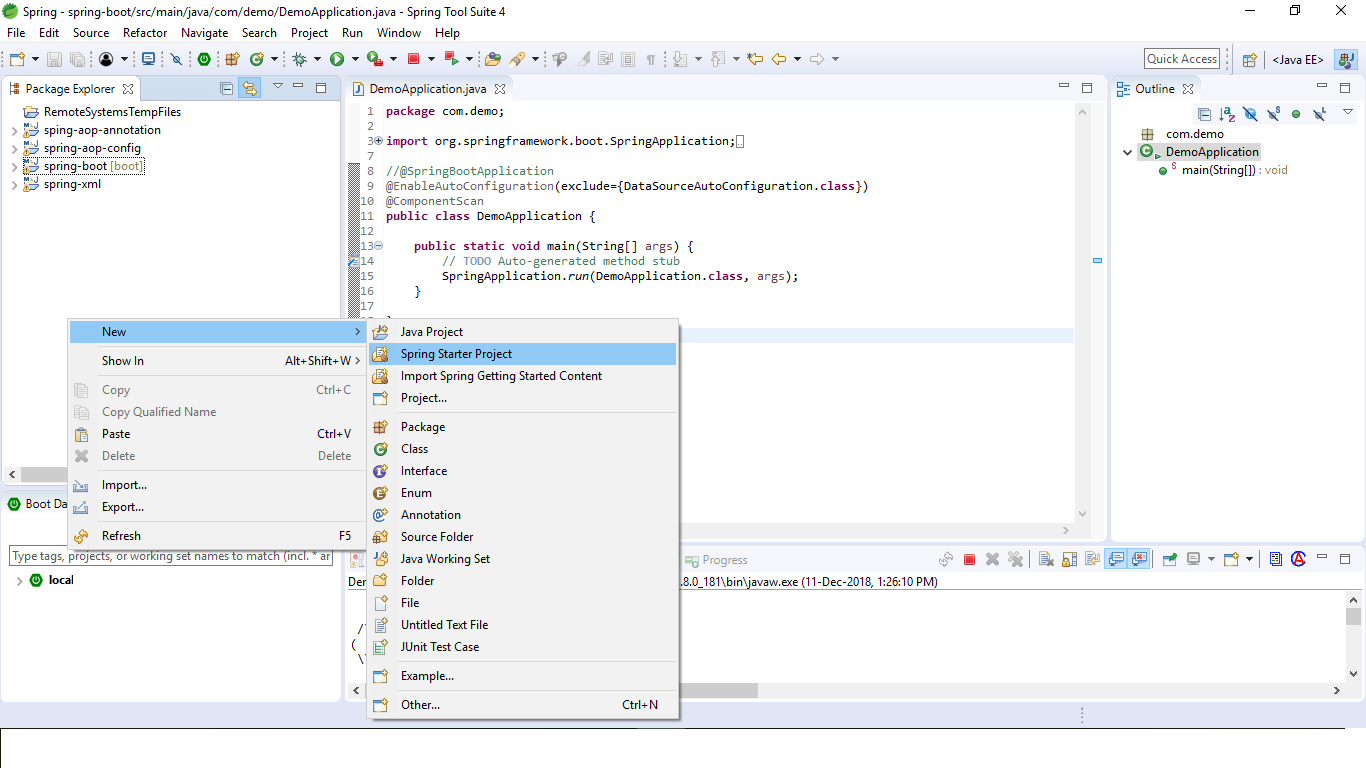
spring-boot-cli-2.1.1.RELEASE-bin.zip

spring-boot-cli-2.1.1.RELEASE-bin.tar.gz

Cutting edge snapshot distributions are also available.

Once downloaded, follow the INSTALL.txt instructions from the unpacked archive. In summary, there is a spring script (spring.bat for Windows) in a bin/ directory in the .zip file. Alternatively, you can use java -jar with the .jar file (the script helps you to be sure that the classpath is set correctly).

Using Spring tool suite



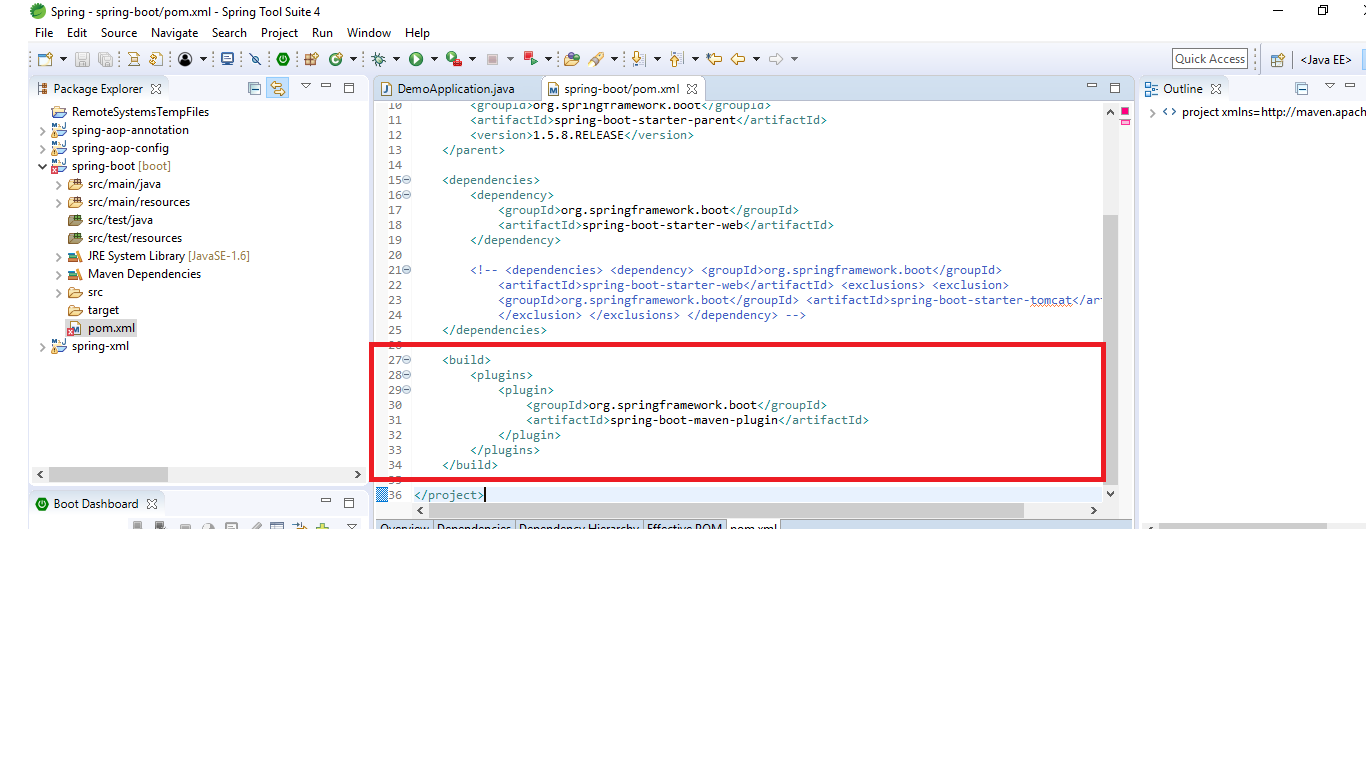
Click next (same as Spring Initializer)

Spring default properties

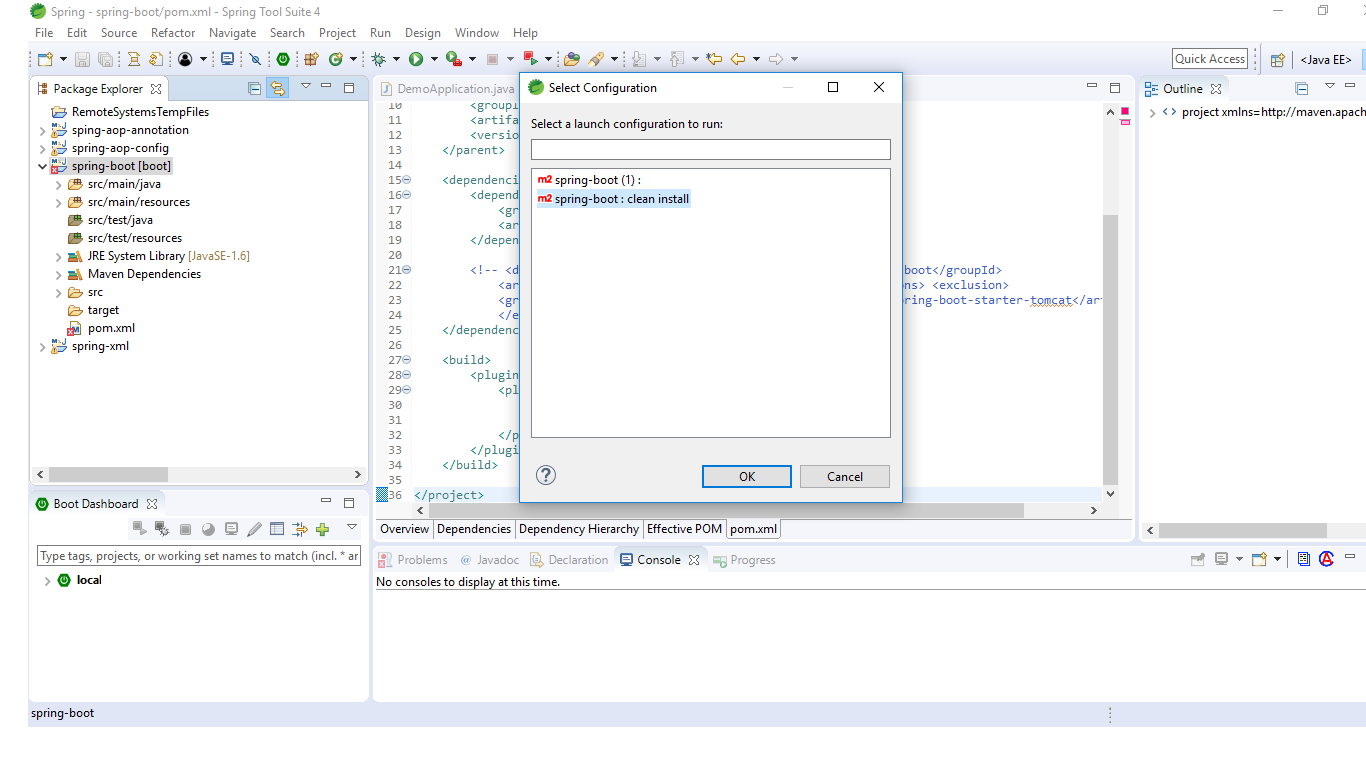
<https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>

Run Spring boot as executable jar

Configure the below in pom.xml



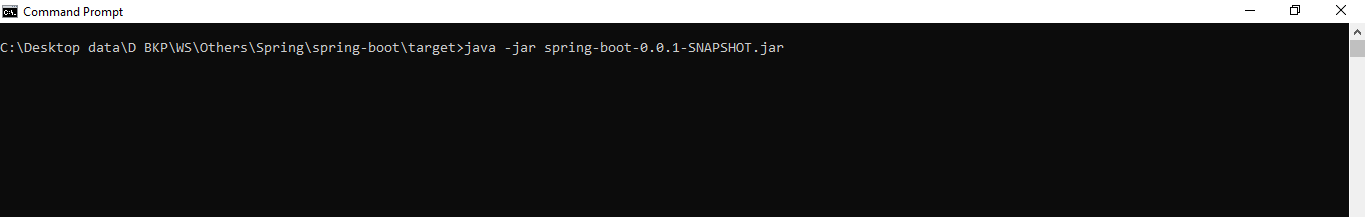
Run it as maven – clean install

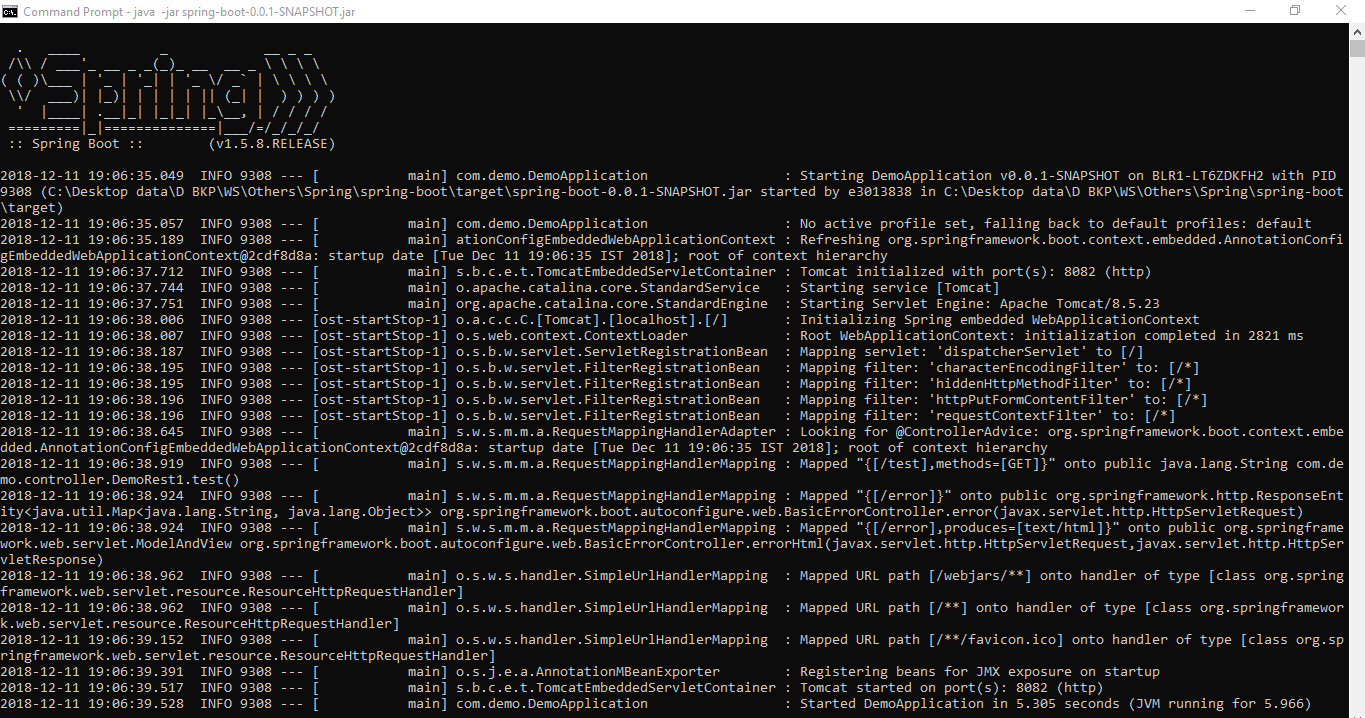


Go to command prompt



Run it as java application using cmd java -jar [jar name]



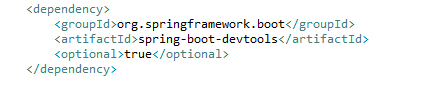


Servlet Containers

Tomcat, Jetty, Reactor Netty and Undertow

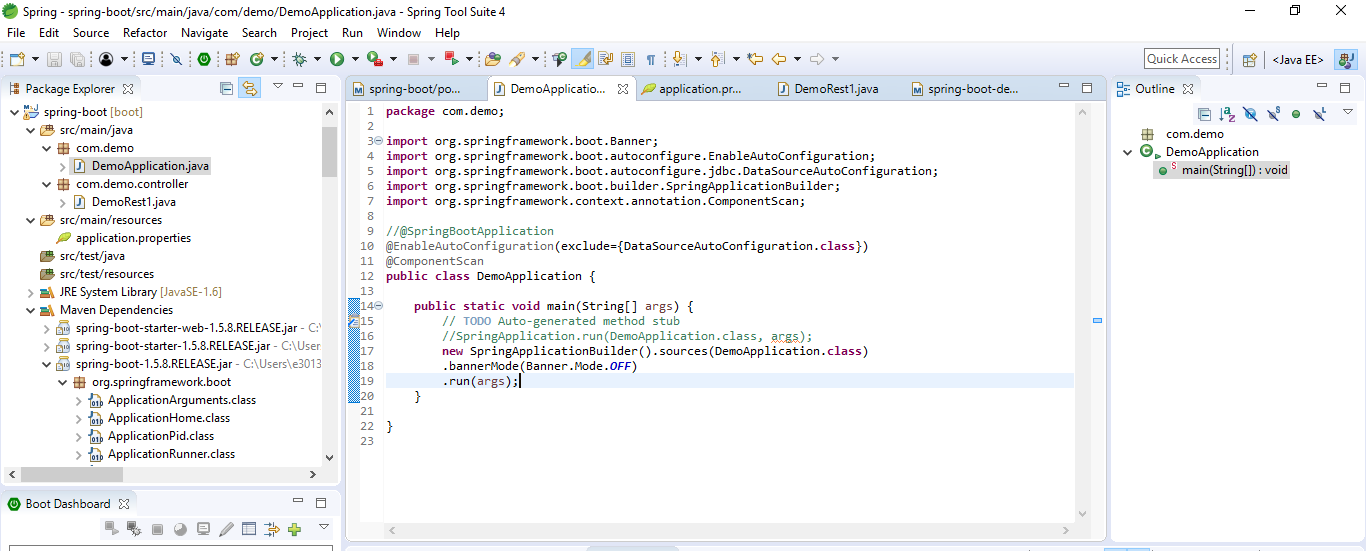
Dev tools

Auto restart when classpath changes, debug options, etc



SpringApplicationBuilder

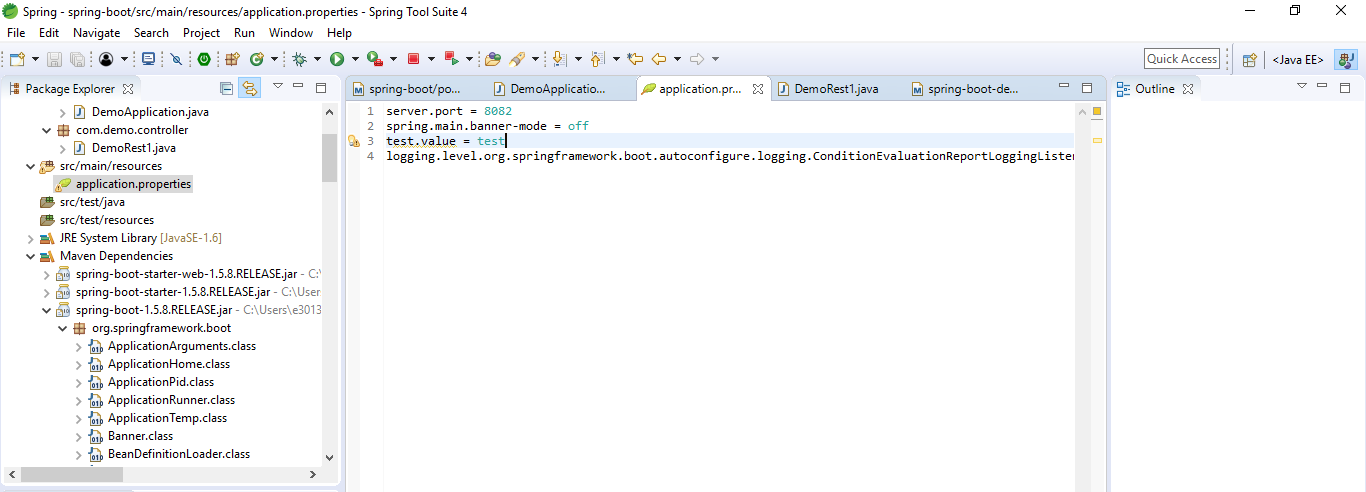
One more way to start spring boot application

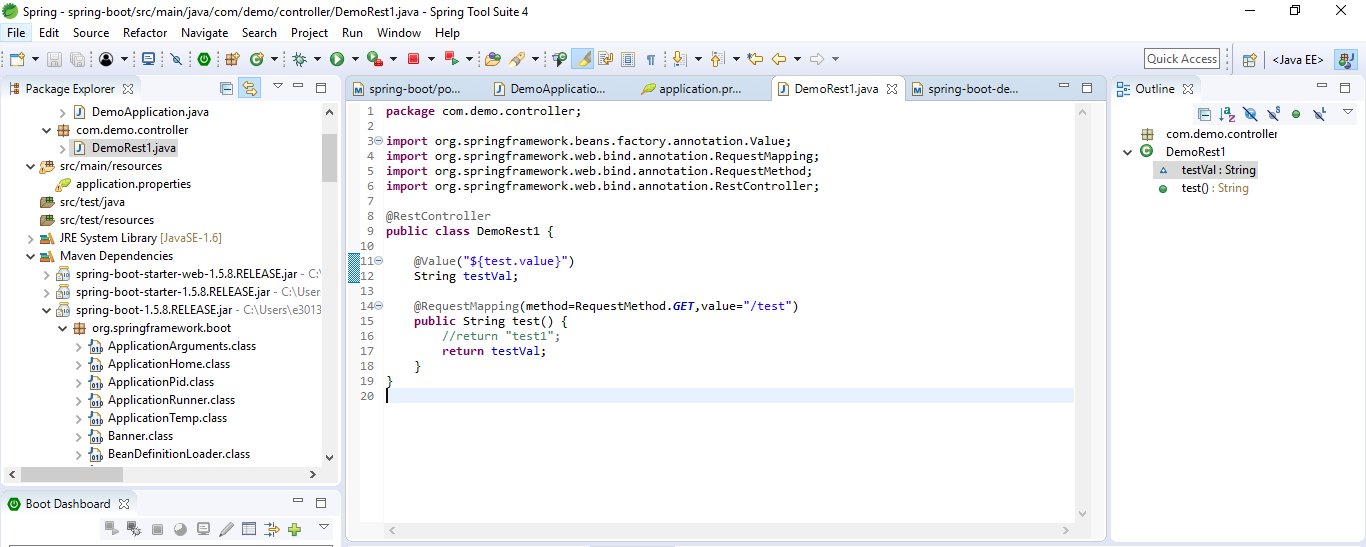


Property file values

We can use .properties, .yml, .yaml files for storing values

Access using @Value annotation. For ex: test.value





The SPRING\_APPLICATION\_JSON properties can be supplied on the command line with an environment variable. For example, you could use the following line in a UN\*X shell:

$ SPRING\_APPLICATION\_JSON='{"acme":{"name":"test"}}' java -jar myapp.jar

In the preceding example, you end up with acme.name=test in the Spring Environment. You can also supply the JSON as spring.application.json in a System property, as shown in the following example:

$ java -Dspring.application.json='{"name":"test"}' -jar myapp.jar

You can also supply the JSON by using a command line argument, as shown in the following example:

$ java -jar myapp.jar --spring.application.json='{"name":"test"}'

Random values

my.secret=${random.value}

my.number=${random.int}

my.bignumber=${random.long}

my.uuid=${random.uuid}

my.number.less.than.ten=${random.int(10)}

my.number.in.range=${random.int[1024,65536]}

The random.int\* syntax is OPEN value (,max) CLOSE where the OPEN,CLOSE are any character and value,max are integers. If max is provided, then value is the minimum value and max is the maximum value (exclusive).

Command Line Properties

By default, SpringApplication converts any command line option arguments (that is, arguments starting with --, such as --server.port=9000) to a property and adds them to the Spring Environment. As mentioned previously, command line properties always take precedence over other property sources.

If you do not like application.properties as the configuration file name, you can switch to another file name by specifying a spring.config.name environment property. You can also refer to an explicit location by using the spring.config.location environment property (which is a comma-separated list of directory locations or file paths). The following example shows how to specify a different file name:

$ java -jar myproject.jar --spring.config.name=myproject

The following example shows how to specify two locations:

$ java -jar myproject.jar --spring.config.location=classpath:/default.properties,classpath:/override.properties

[Warning]

spring.config.name and spring.config.location are used very early to determine which files have to be loaded, so they must be defined as an environment property (typically an OS environment variable, a system property, or a command-line argument).

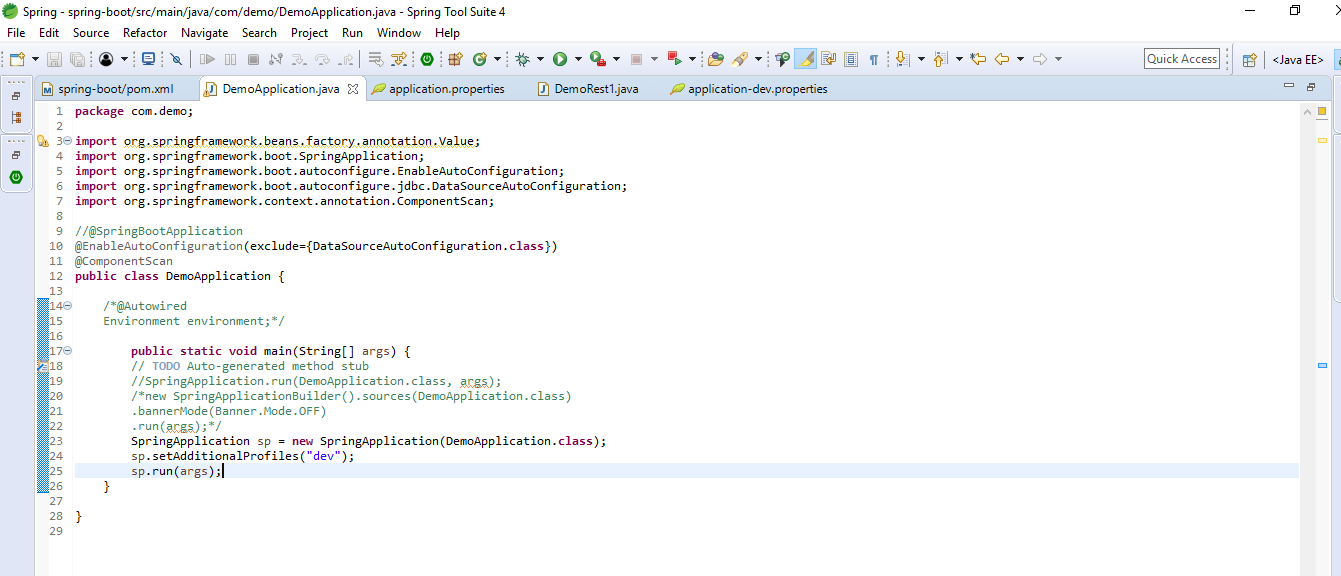
By default, the configured locations are classpath:/,classpath:/config/,file:./,file:./config/

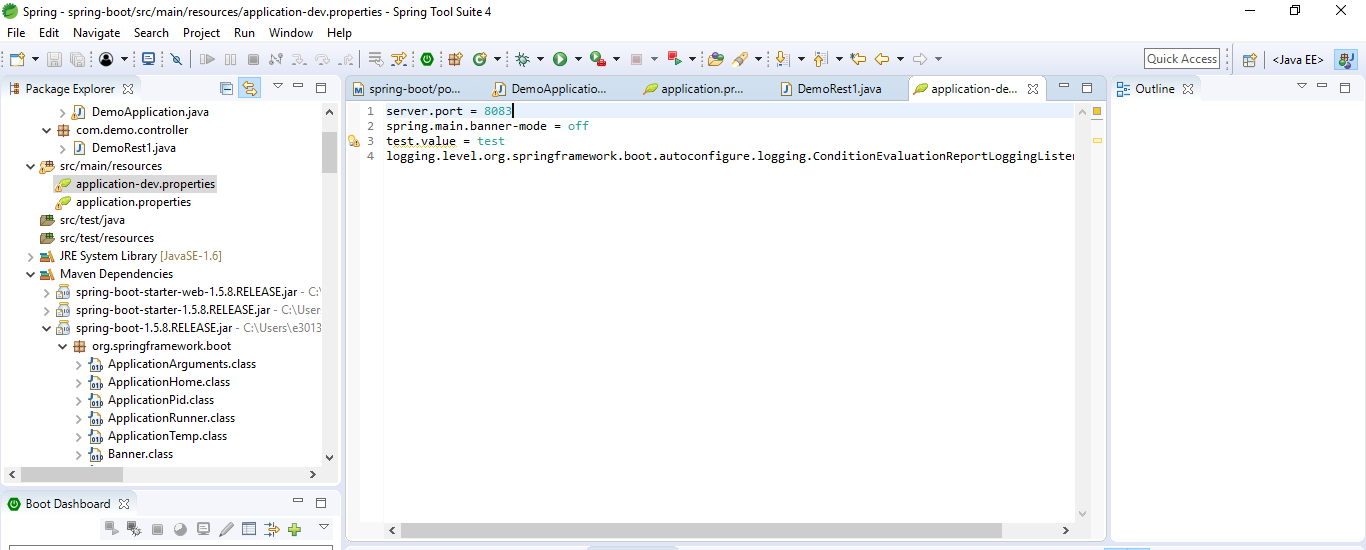
Ex:

file:./custom-config/

classpath:custom-config/

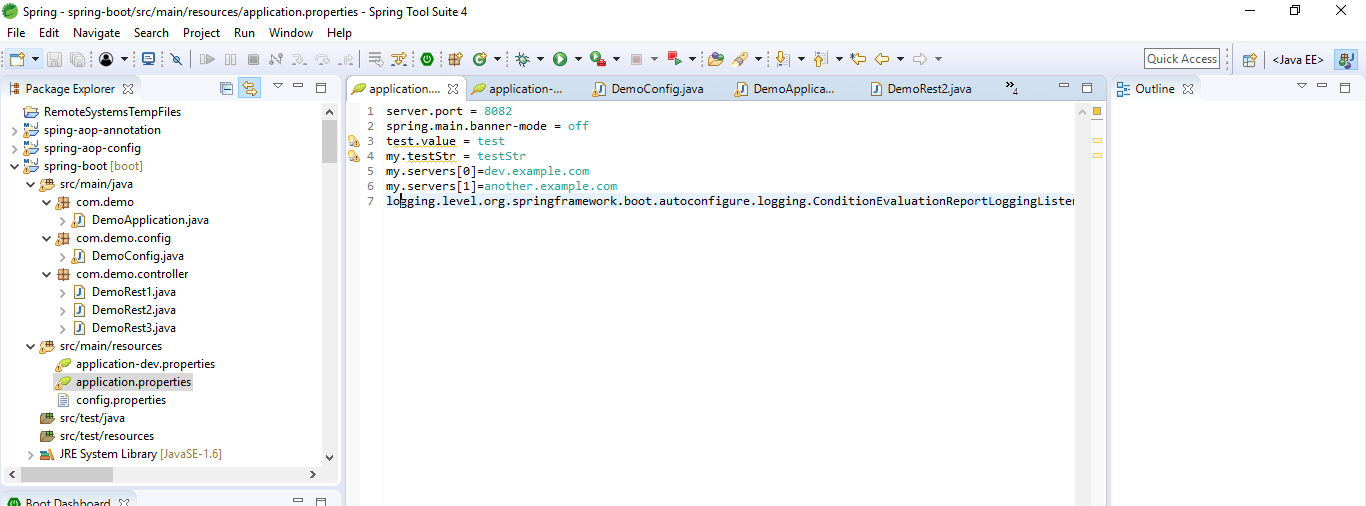
Add profile

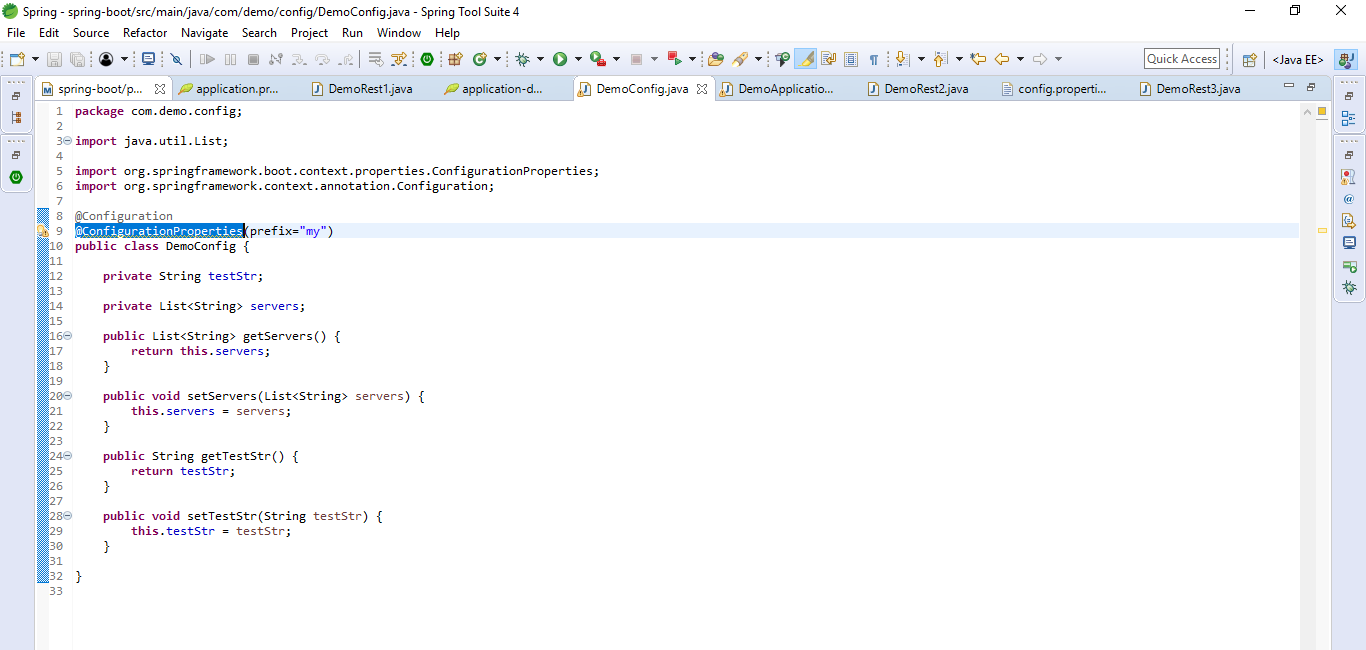




@ConfigurationProperties

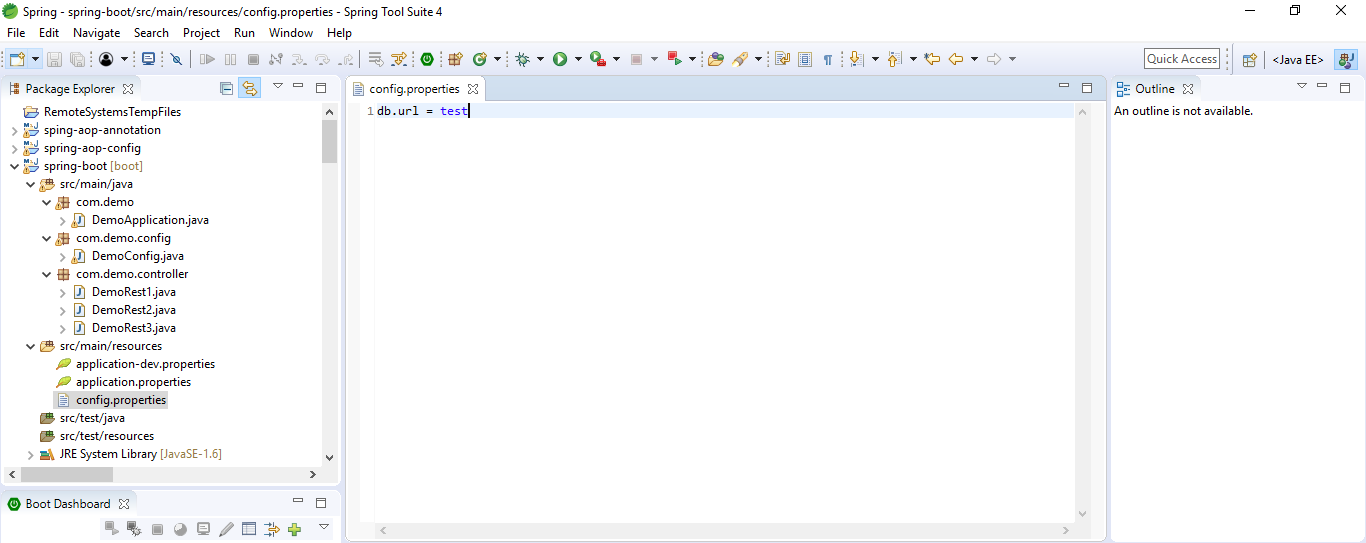
This is to directly read from property file with prefix

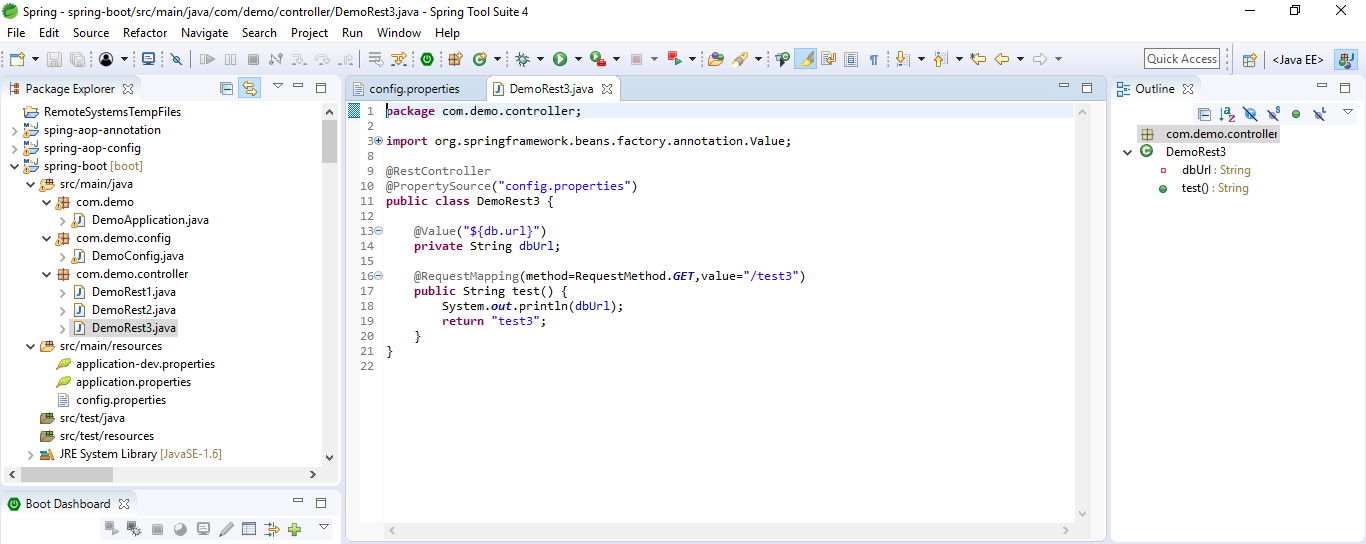




@PropertySource

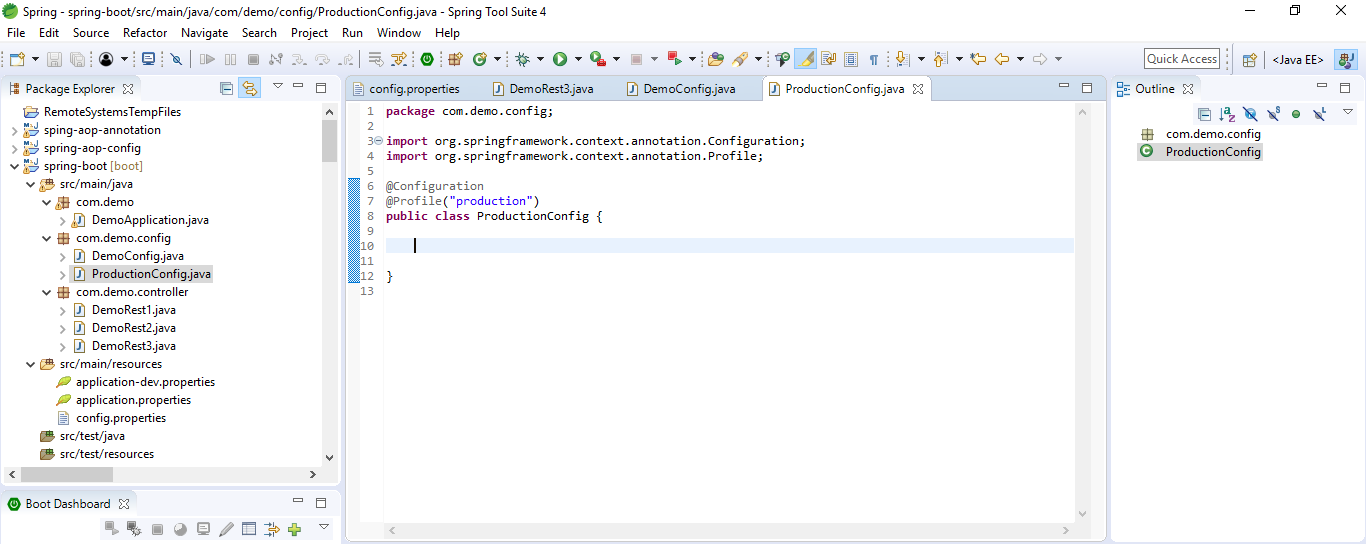
To read from custom property file





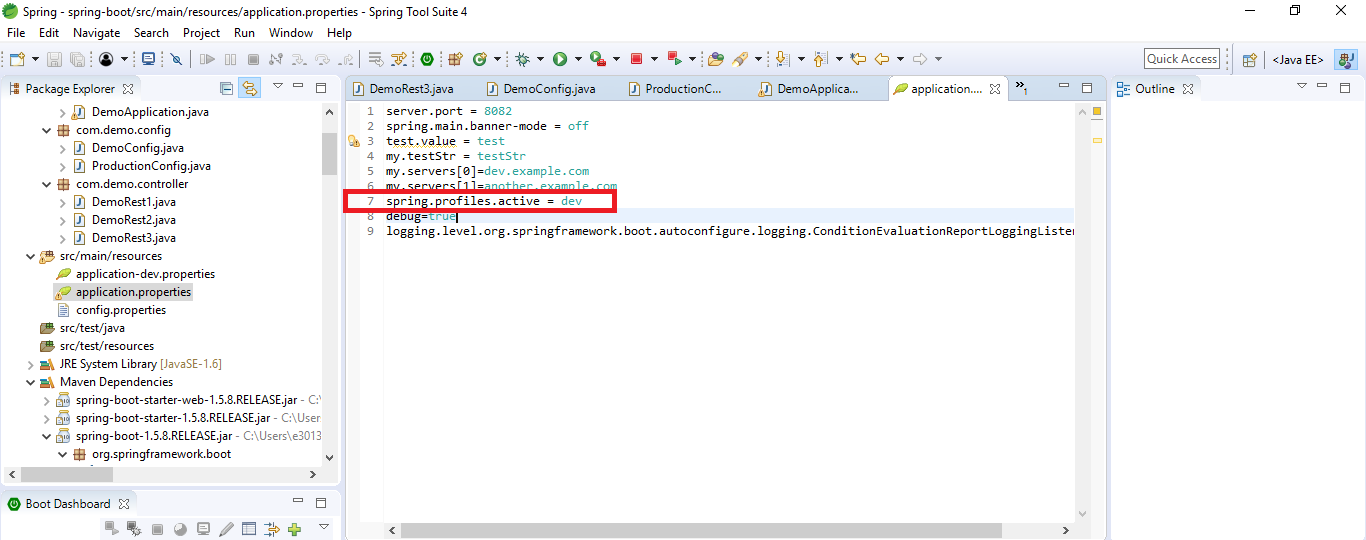
@Profile

Spring Profiles provide a way to segregate parts of your application configuration and make it be available only in certain environments. Any @Component or @Configuration can be marked with @Profile to limit when it is loaded, as shown in the following example:



Dynamic profile

Set in property file



Or dynamic

Java -jar –spring.profiles.active

Debugging

