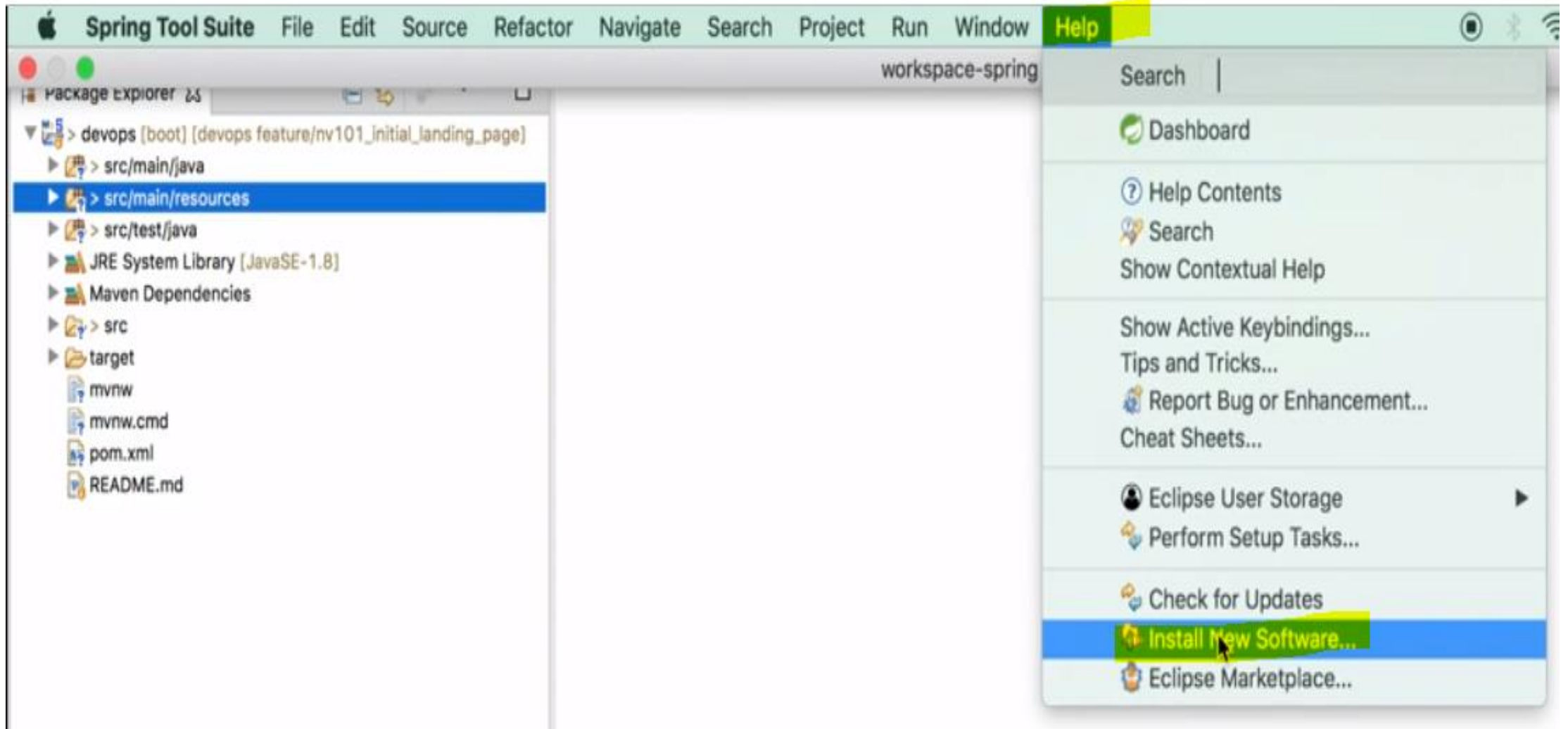


# Spring Boot Basic Server Code

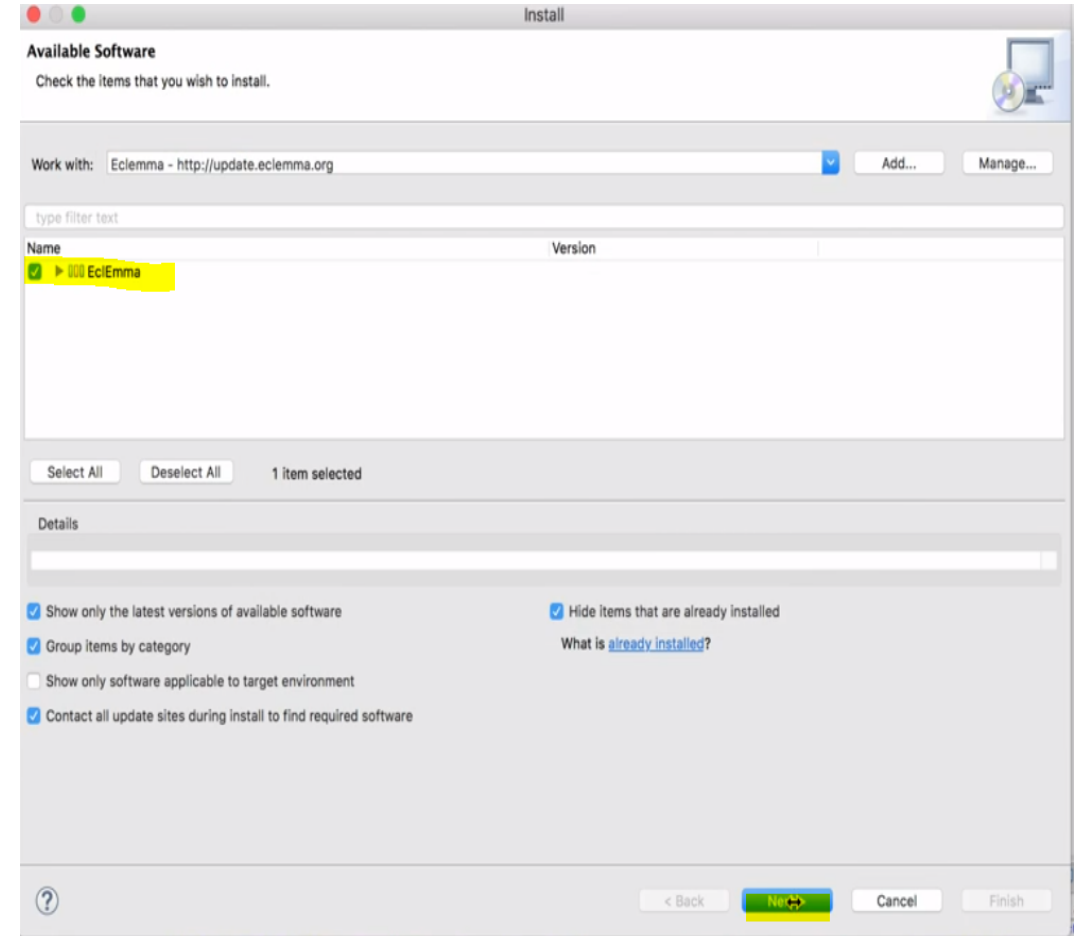
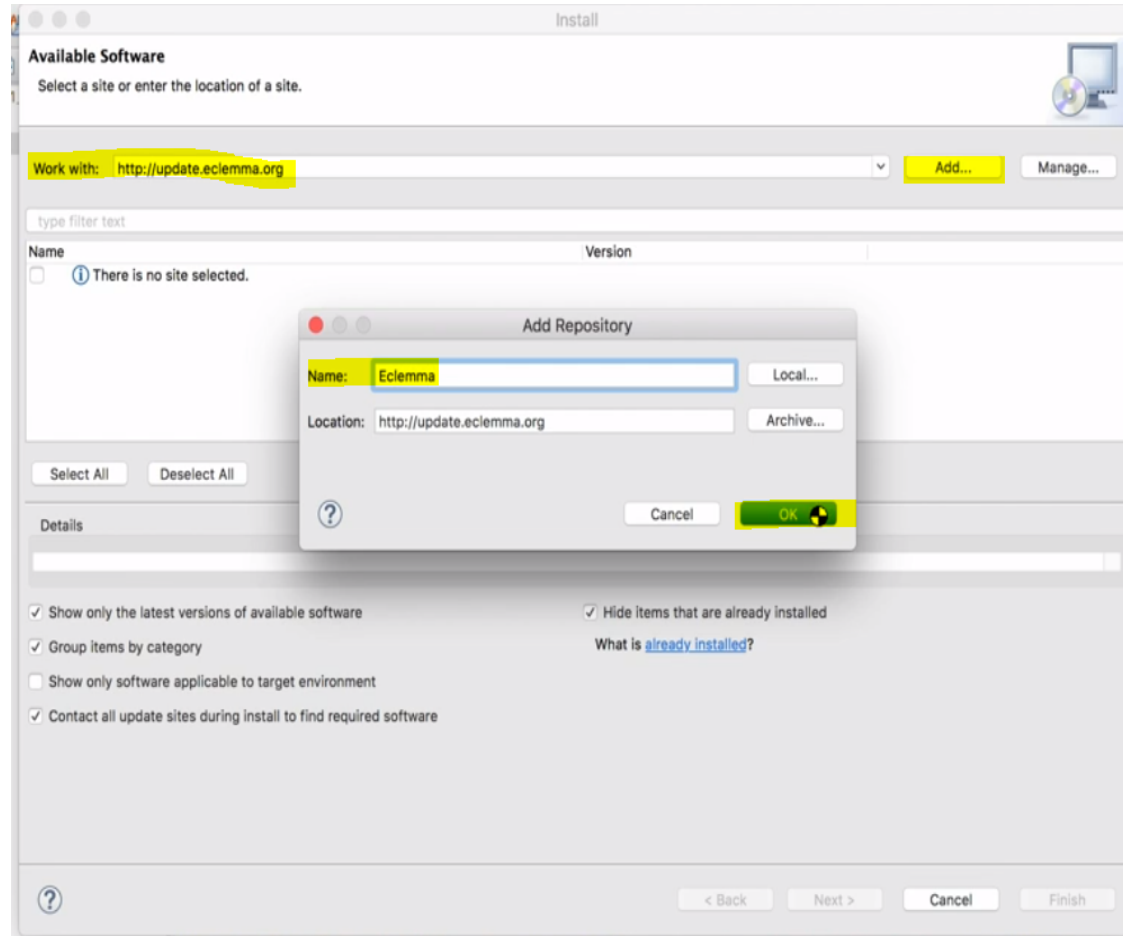
By

Keshav Kummari

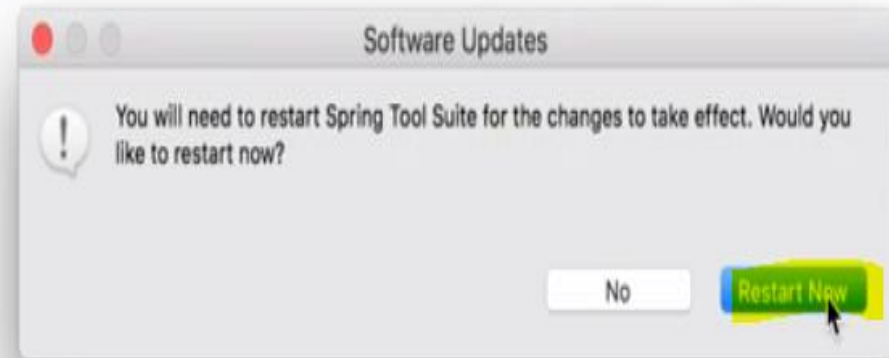
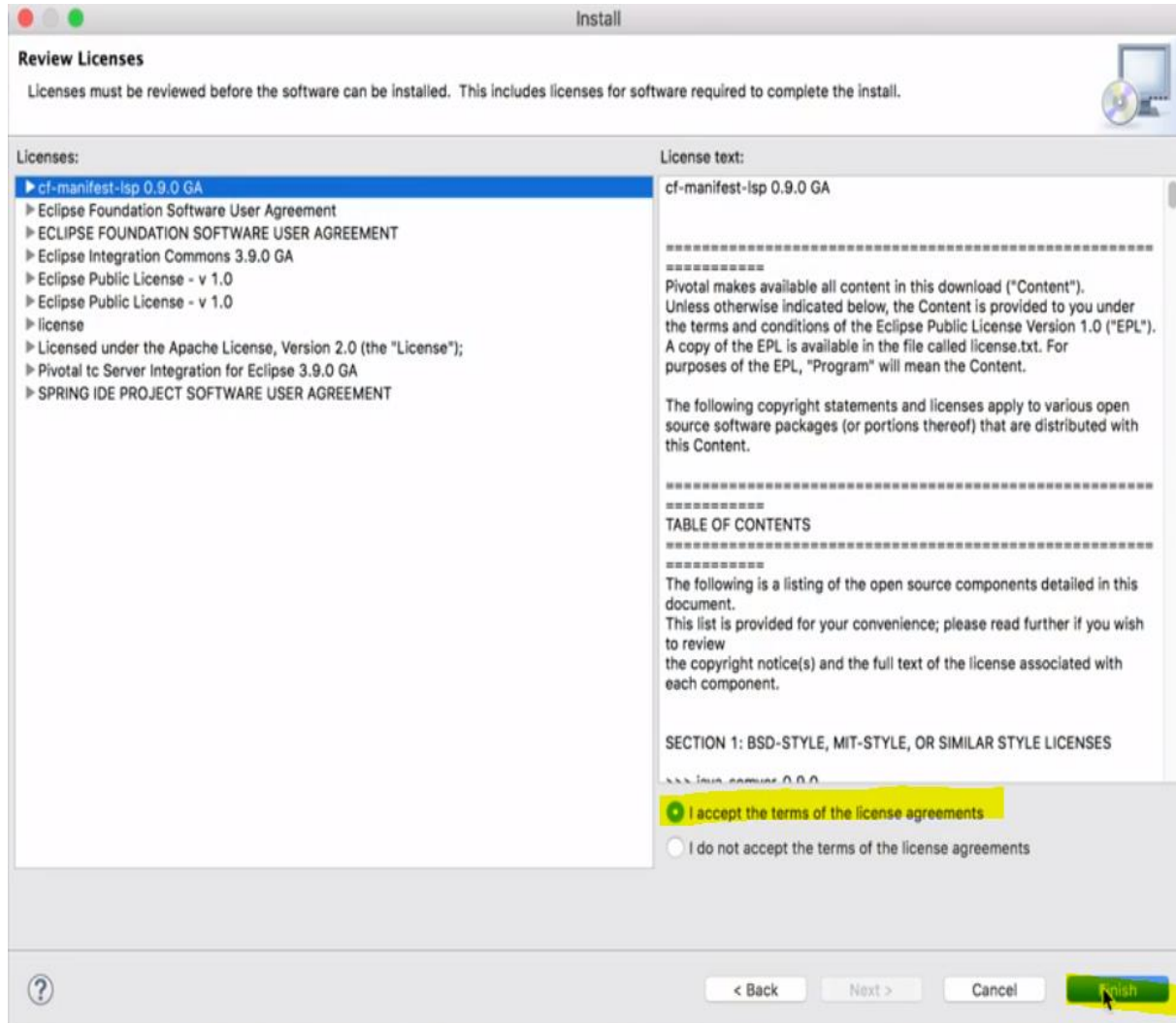
# Test Driven Development Tool - eclemma



# Click on Add & add Name & click on OK



# Click on I accept & Restart the STS

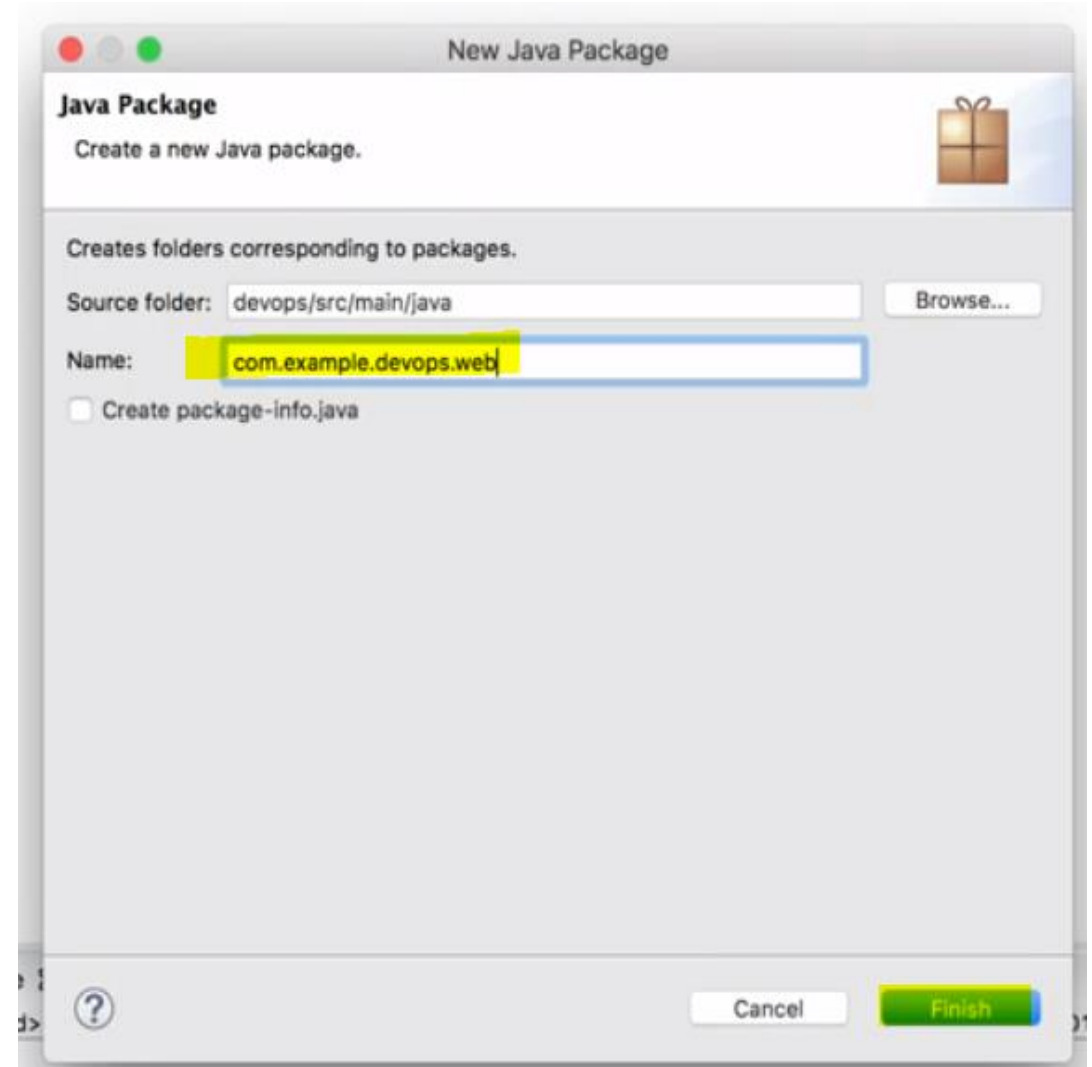
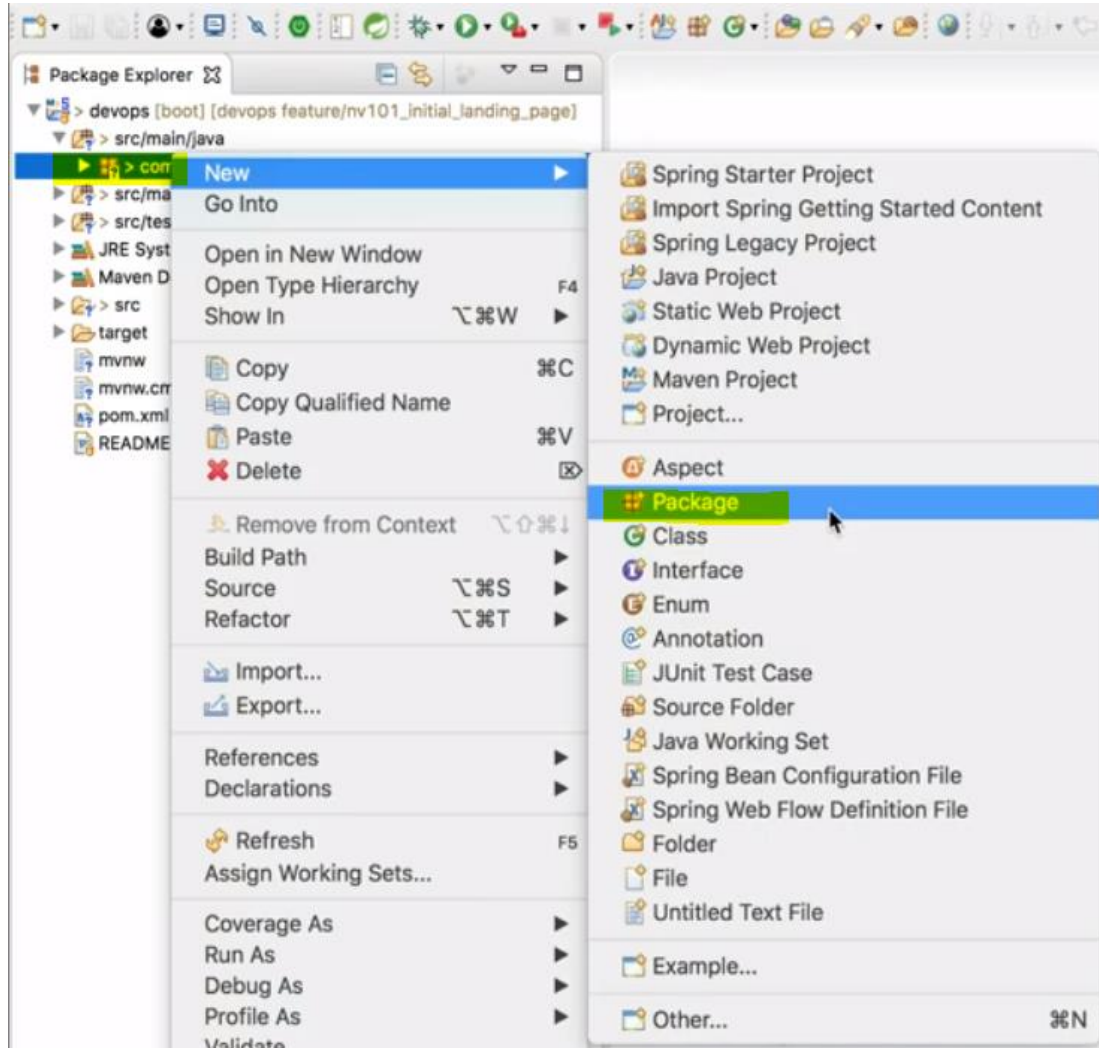


# Execute Junit Test & check Coverage Report

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays the project structure. The 'src/test' package is selected, and a context menu is open. The 'Coverage As' option is highlighted, which has opened a sub-menu where '2 JUnit Test' is selected. On the right, the Coverage view is active, displaying a table of coverage data for the 'devops' project.

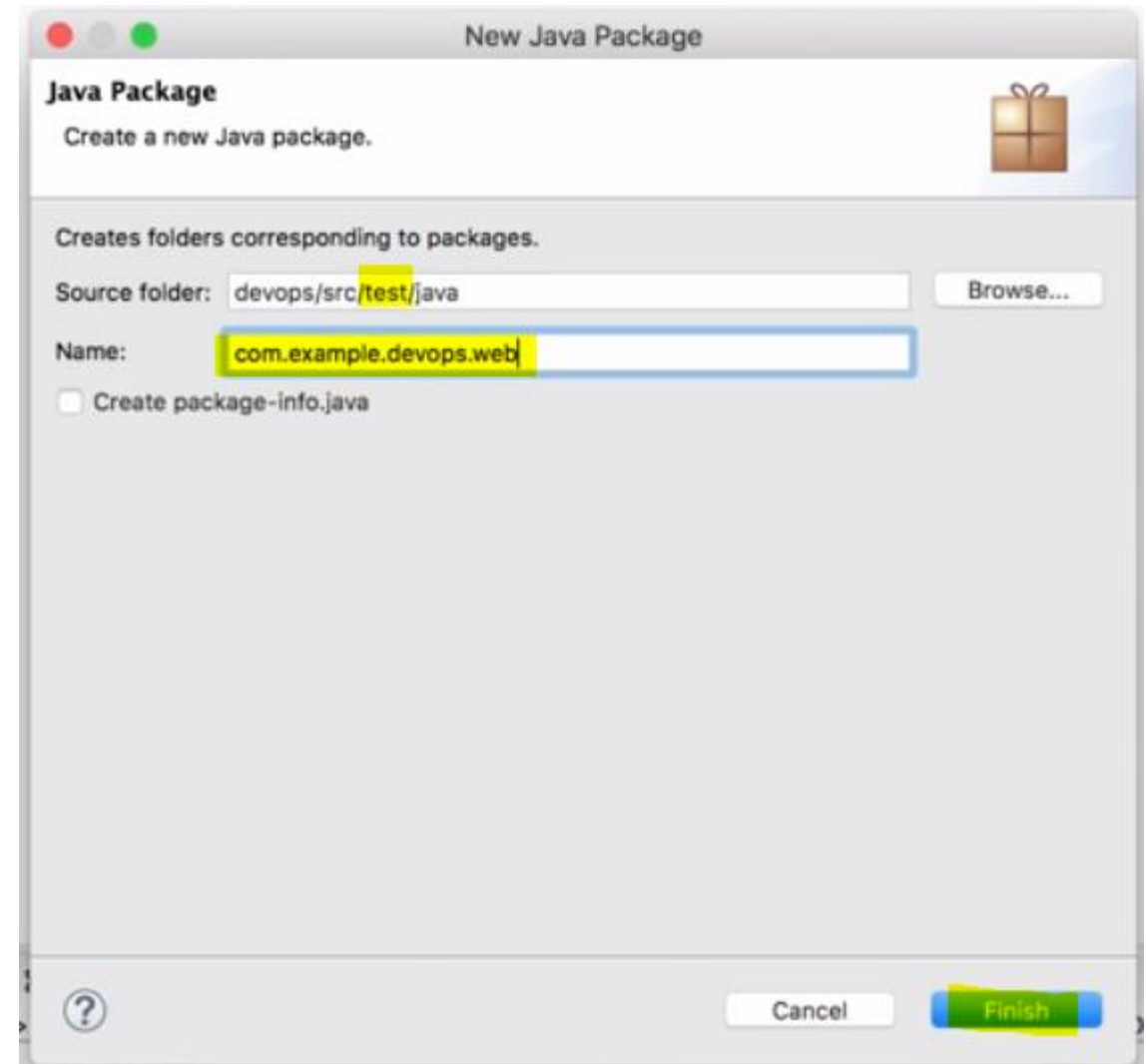
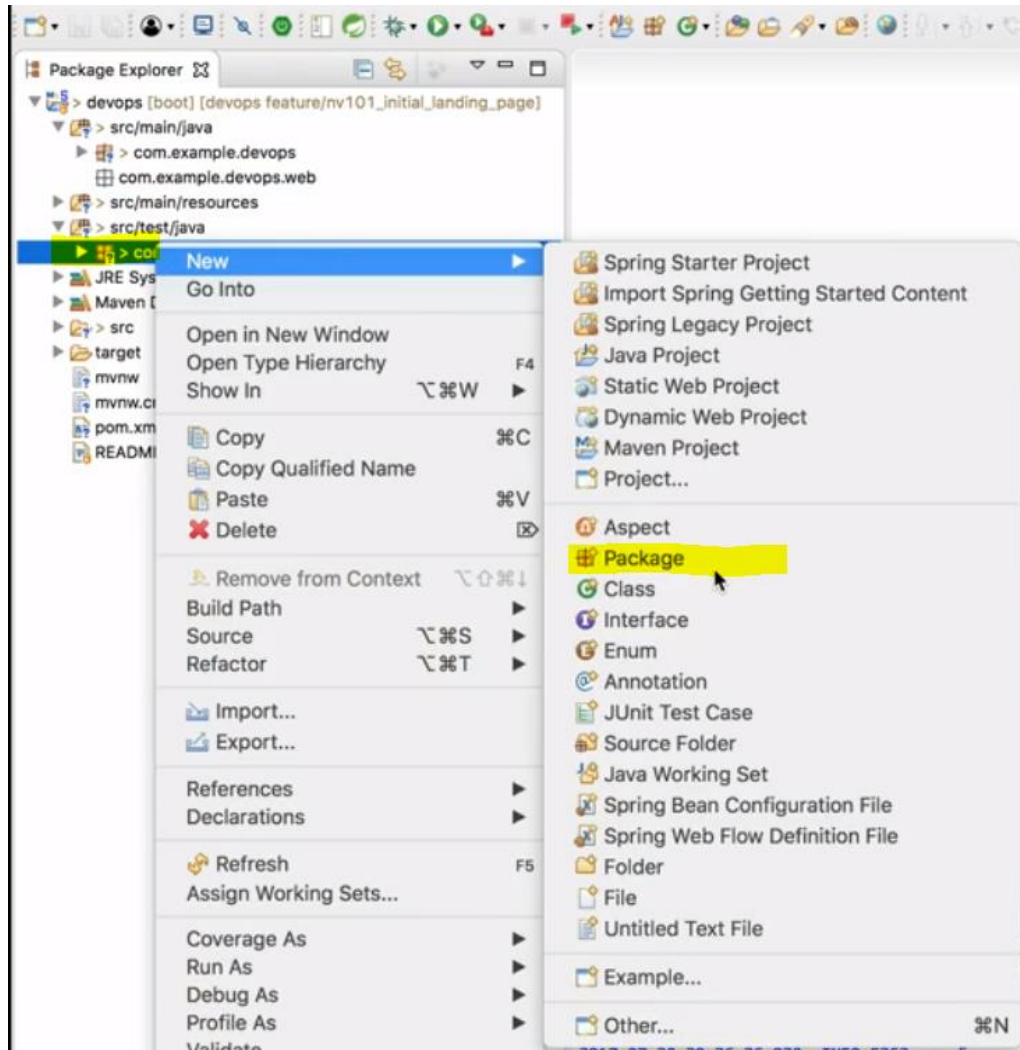
Element	Coverage	Covered Instructions	Missed Instructions
devops	29.2 %	7	
src/main/java	15.0 %	3	
com.example.devops	15.0 %	3	
ServletInitializer.java	0.0 %	0	
DevopsApplication.java	37.5 %	3	
src/test/java	100.0 %	4	

# Package – Welcome Controller For “main”

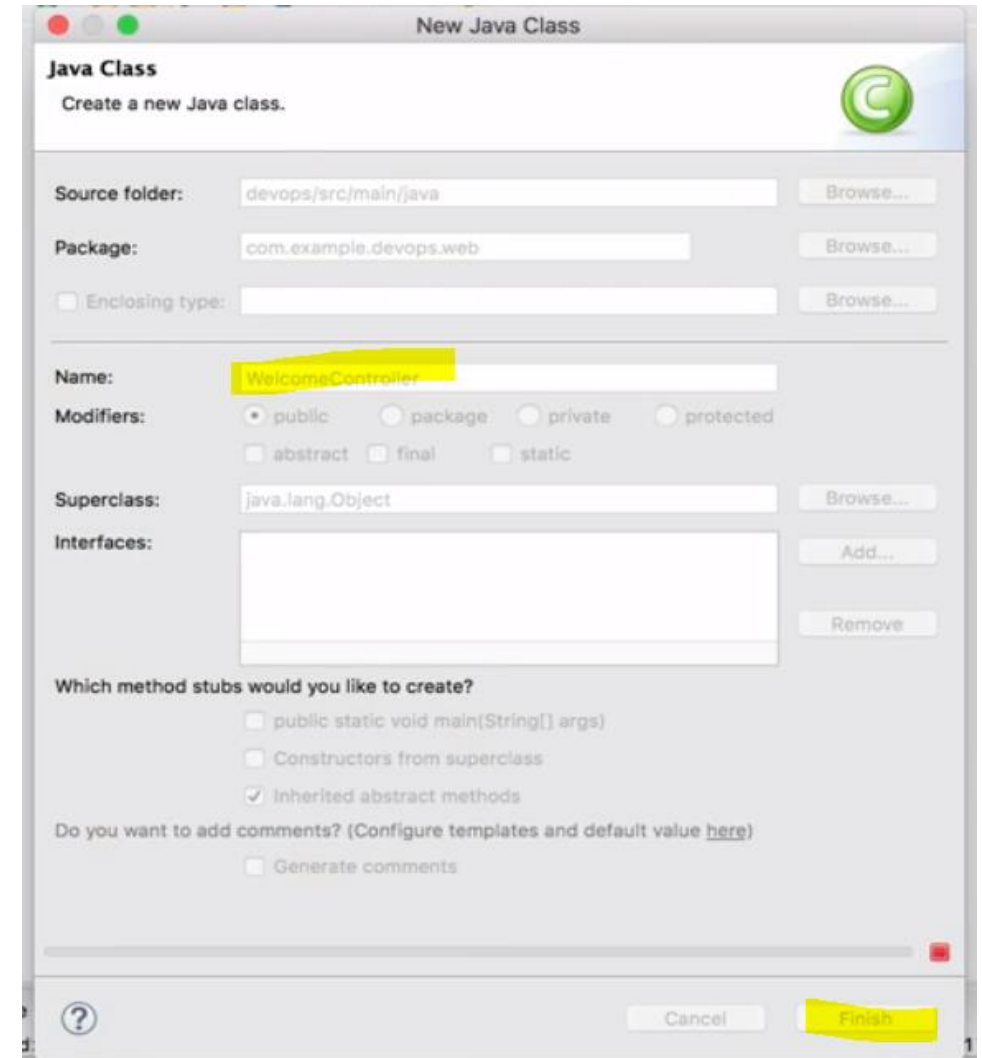
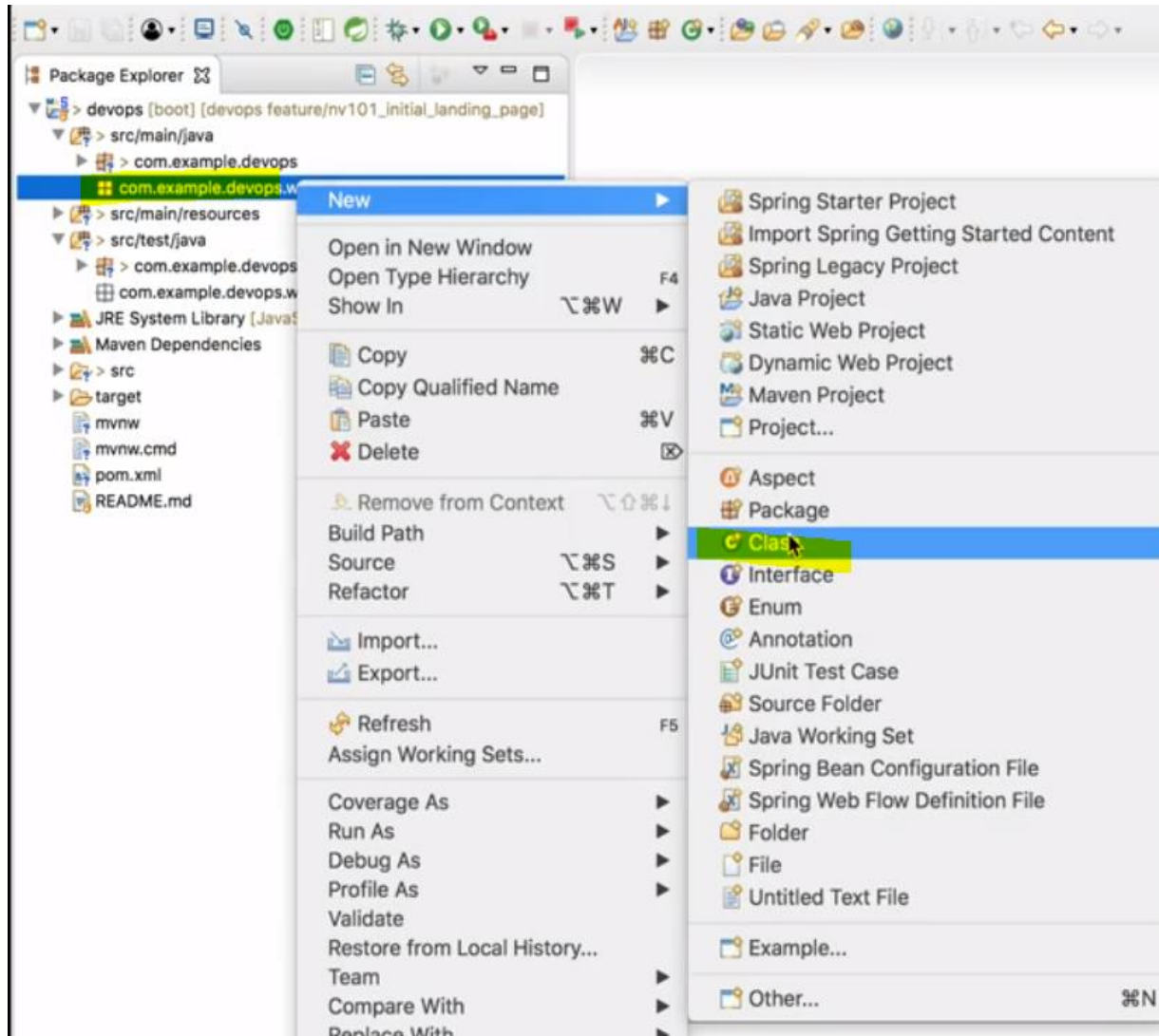




# Package – Welcome Controller for “test”

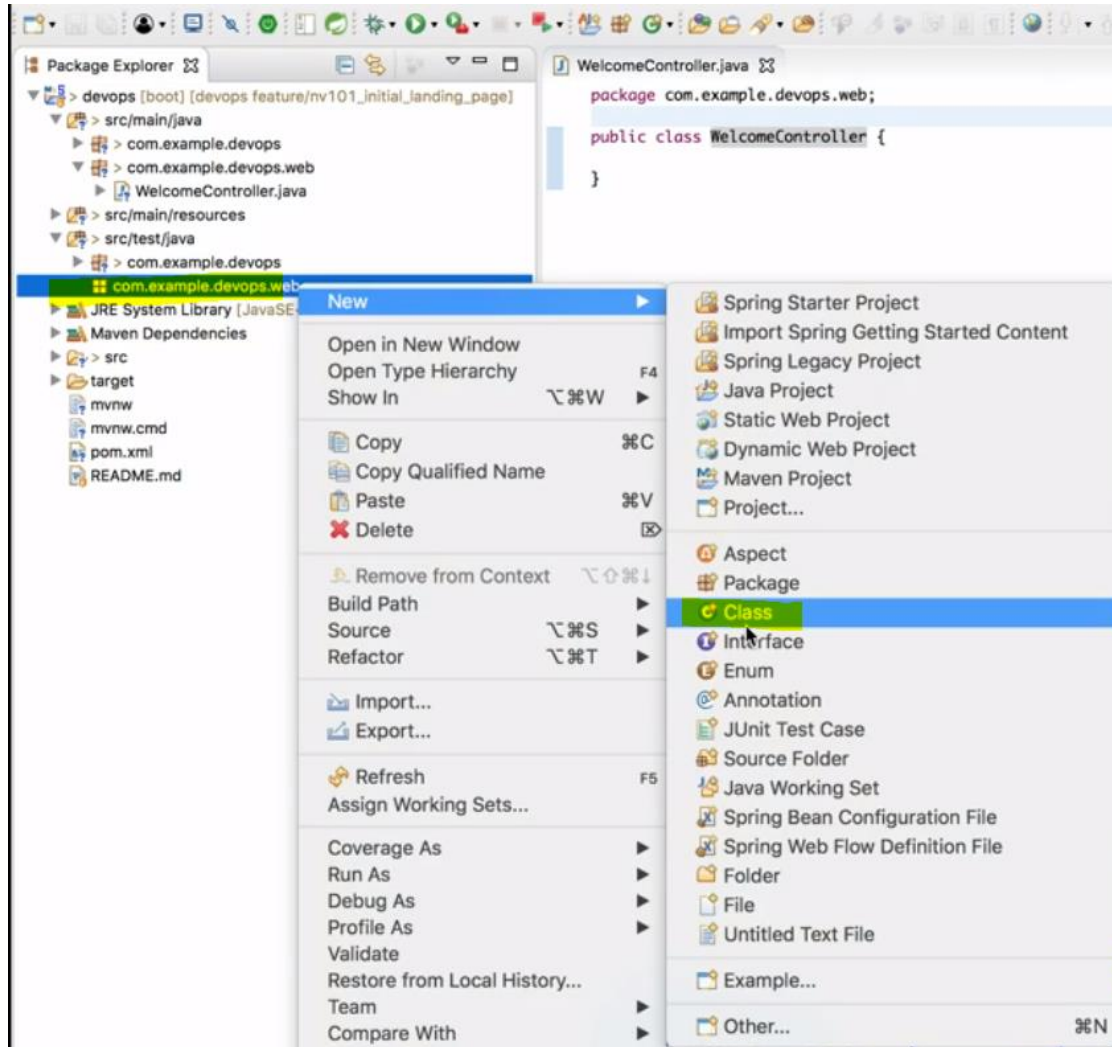


# Create a Welcome Controller Class - main

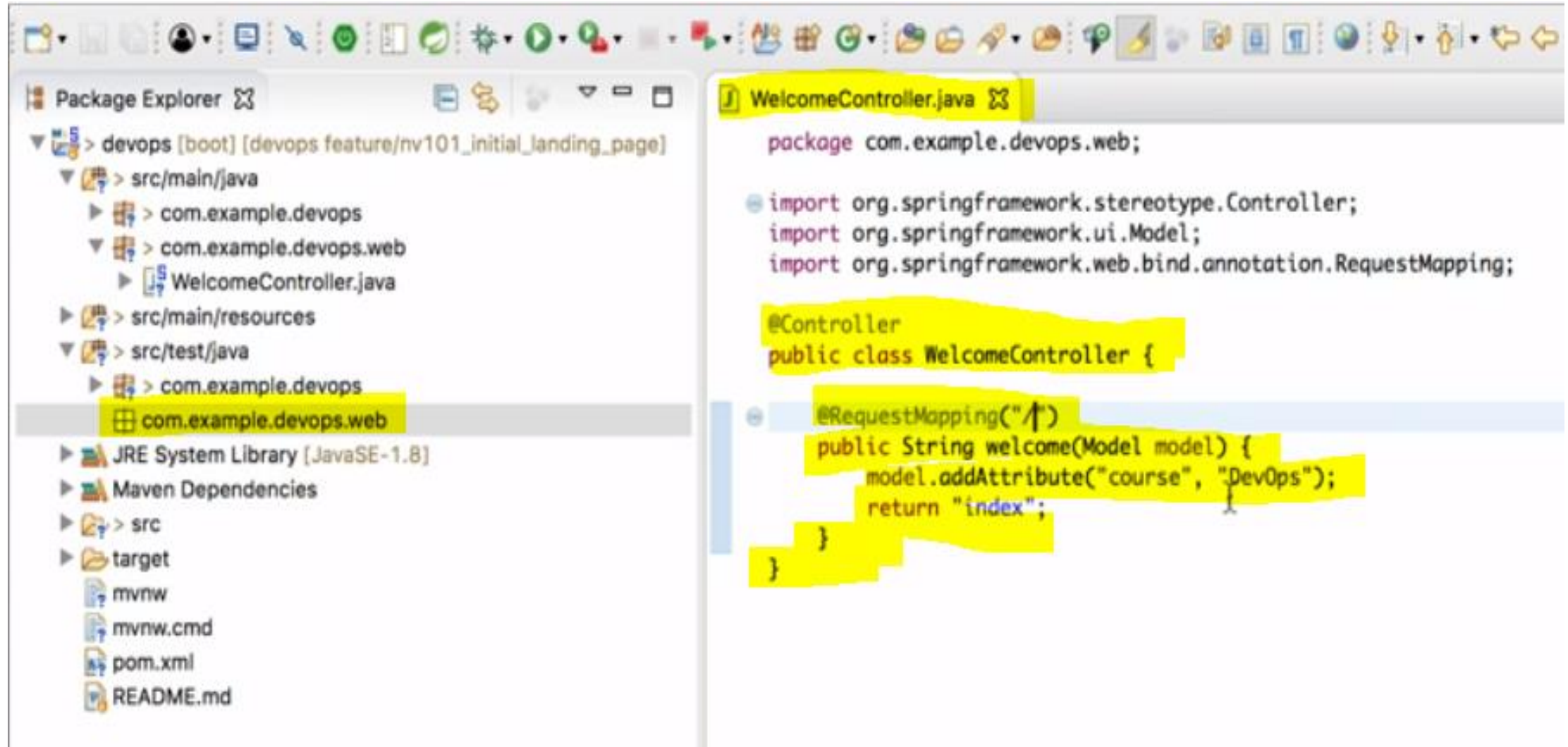




# Create a Welcome Controller Class - test



# Do changes in WelcomeController.java



The screenshot shows an IDE interface with two main panels. The left panel is the Package Explorer, showing a project structure with the following hierarchy:

- devops [boot] [devops feature/nv101\_initial\_landing\_page]
  - src/main/java
    - com.example.devops
      - com.example.devops.web
        - WelcomeController.java
  - src/main/resources
  - src/test/java
    - com.example.devops
- com.example.devops.web (selected)
- JRE System Library [JavaSE-1.8]
- Maven Dependencies
- src
- target
  - mvnw
  - mvnw.cmd
  - pom.xml
  - README.md

The right panel shows the code for WelcomeController.java. The code is as follows:

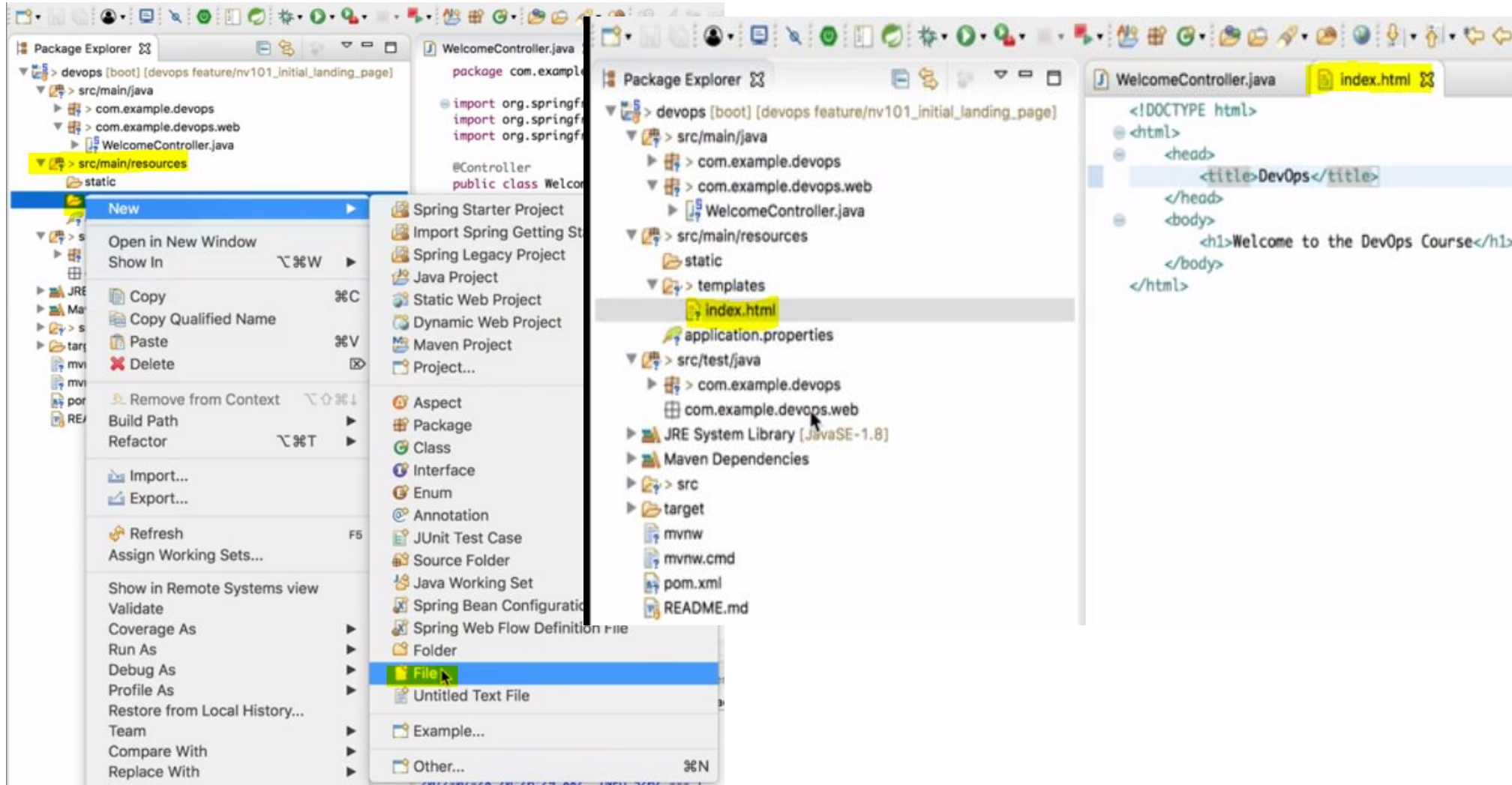
```
package com.example.devops.web;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

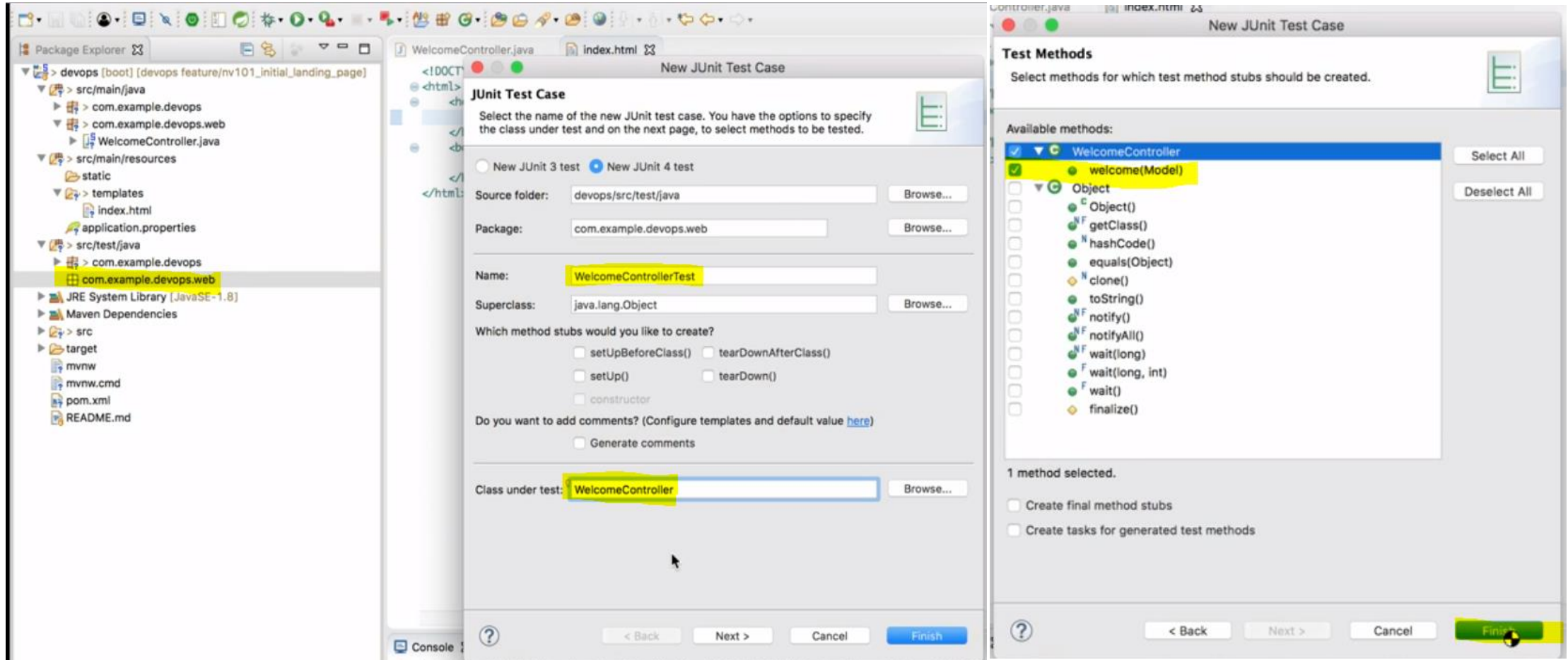
@Controller
public class WelcomeController {

    @RequestMapping("/")
    public String welcome(Model model) {
        model.addAttribute("course", "DevOps");
        return "index";
    }
}
```

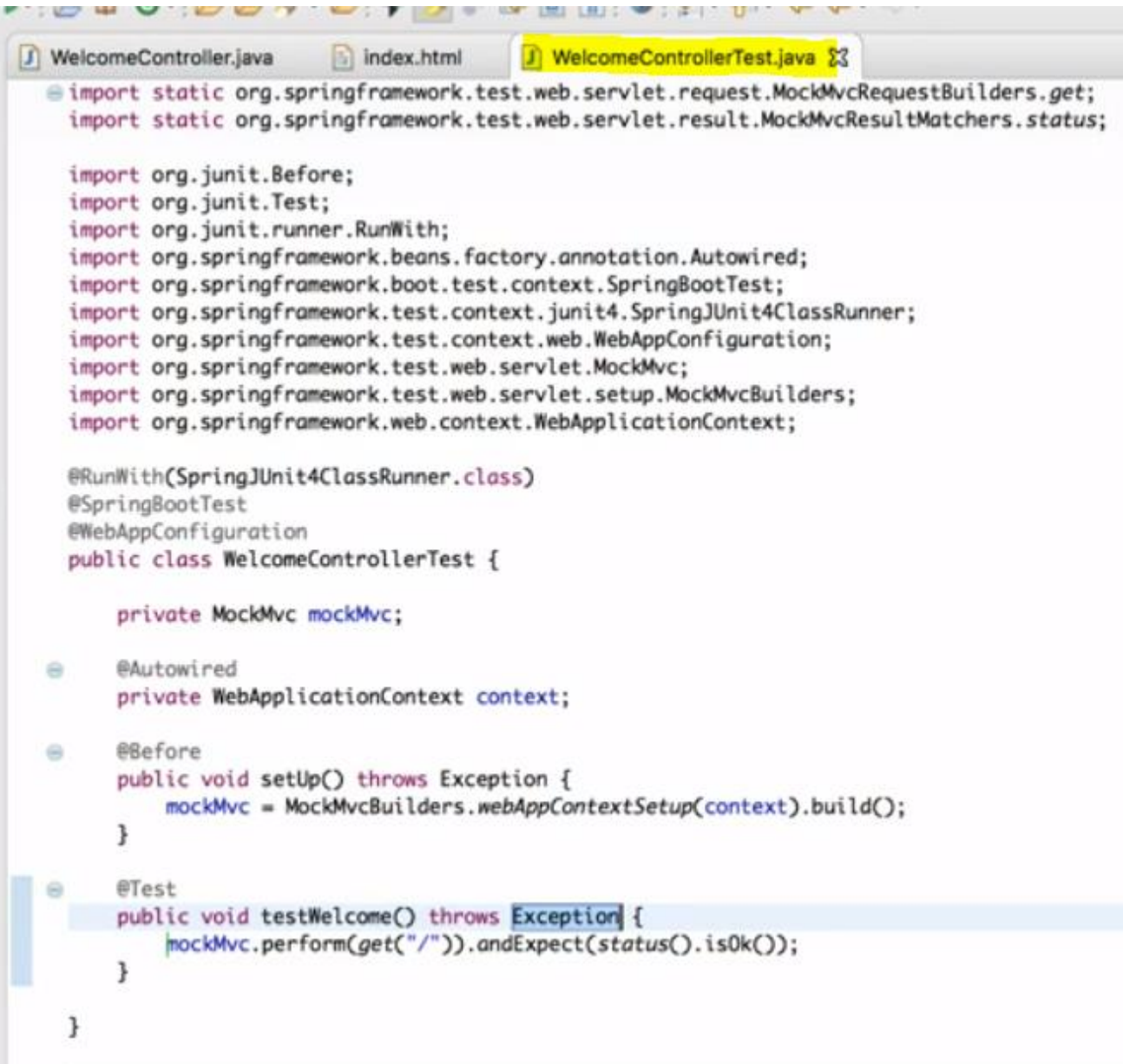
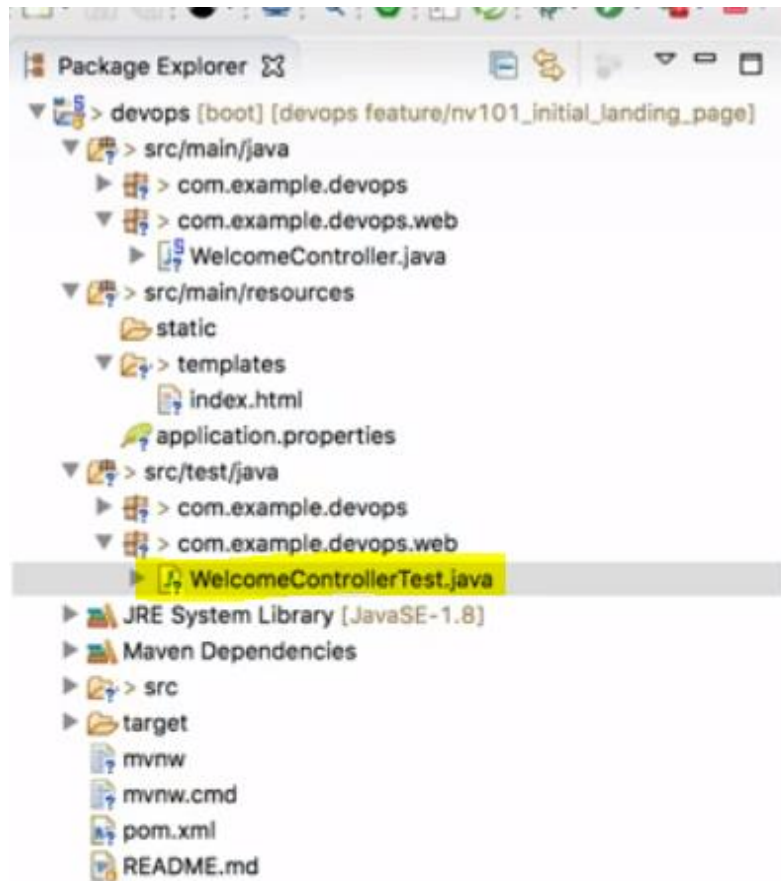
# Go to >> templates & create index.html file



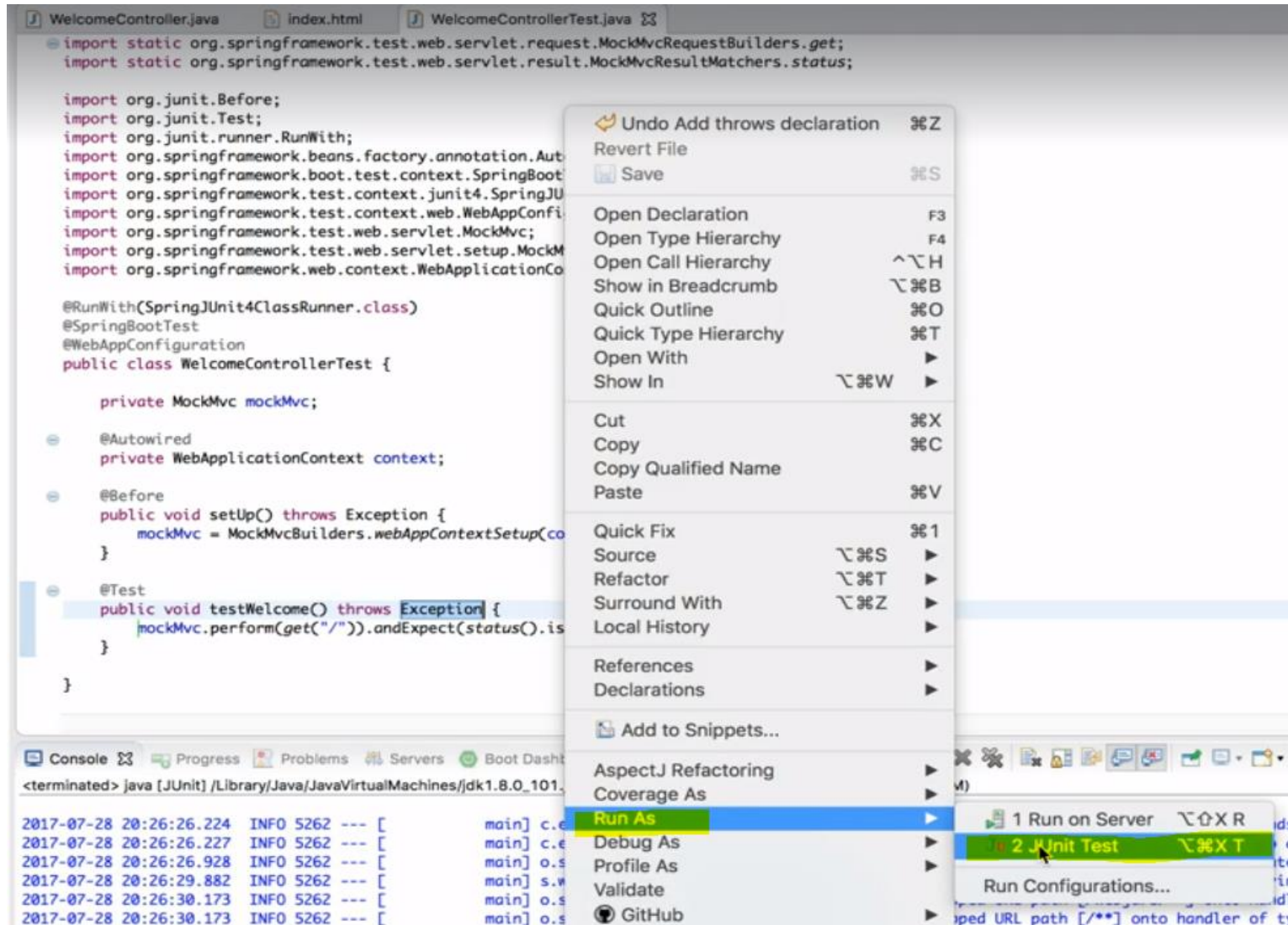
# Go to >> WelcomeControllerTest - Test





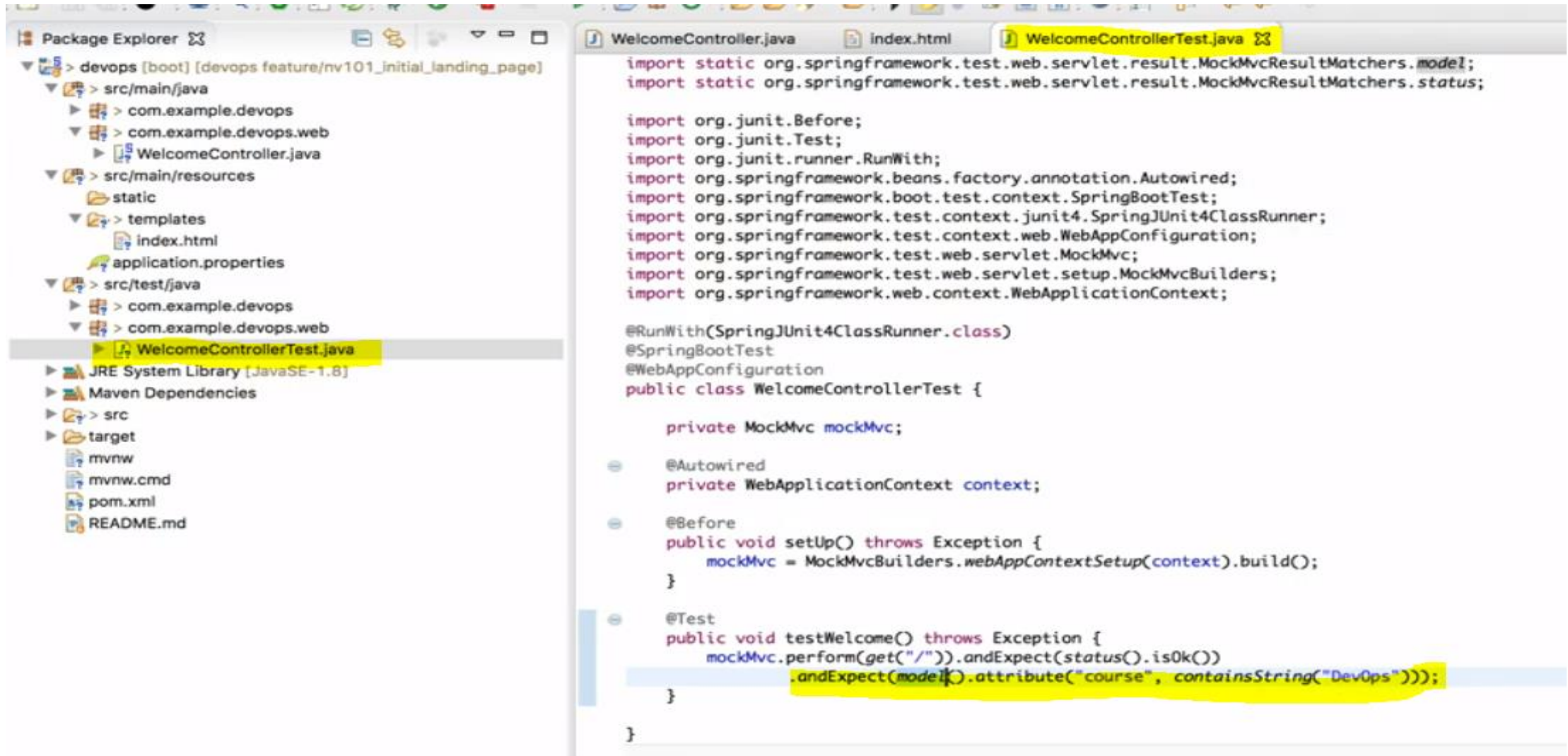


# Right click on & Run as Junit Test





# Added a row



The screenshot shows an IDE with two panels. The left panel displays the Package Explorer, and the right panel shows the source code of `WelcomeControllerTest.java`.

**Package Explorer (Left Panel):**

- devops [boot] [devops feature/nv101\_initial\_landing\_page]
  - src/main/java
    - com.example.devops
      - com.example.devops.web
        - WelcomeController.java
  - src/main/resources
    - static
    - templates
      - index.html
    - application.properties
  - src/test/java
    - com.example.devops
      - com.example.devops.web
        - WelcomeControllerTest.java
  - JRE System Library [JavaSE-1.8]
  - Maven Dependencies
  - src
  - target
    - mvnw
    - mvnw.cmd
    - pom.xml
    - README.md

**WelcomeControllerTest.java (Right Panel):**

```
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.model;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
@WebAppConfiguration
public class WelcomeControllerTest {

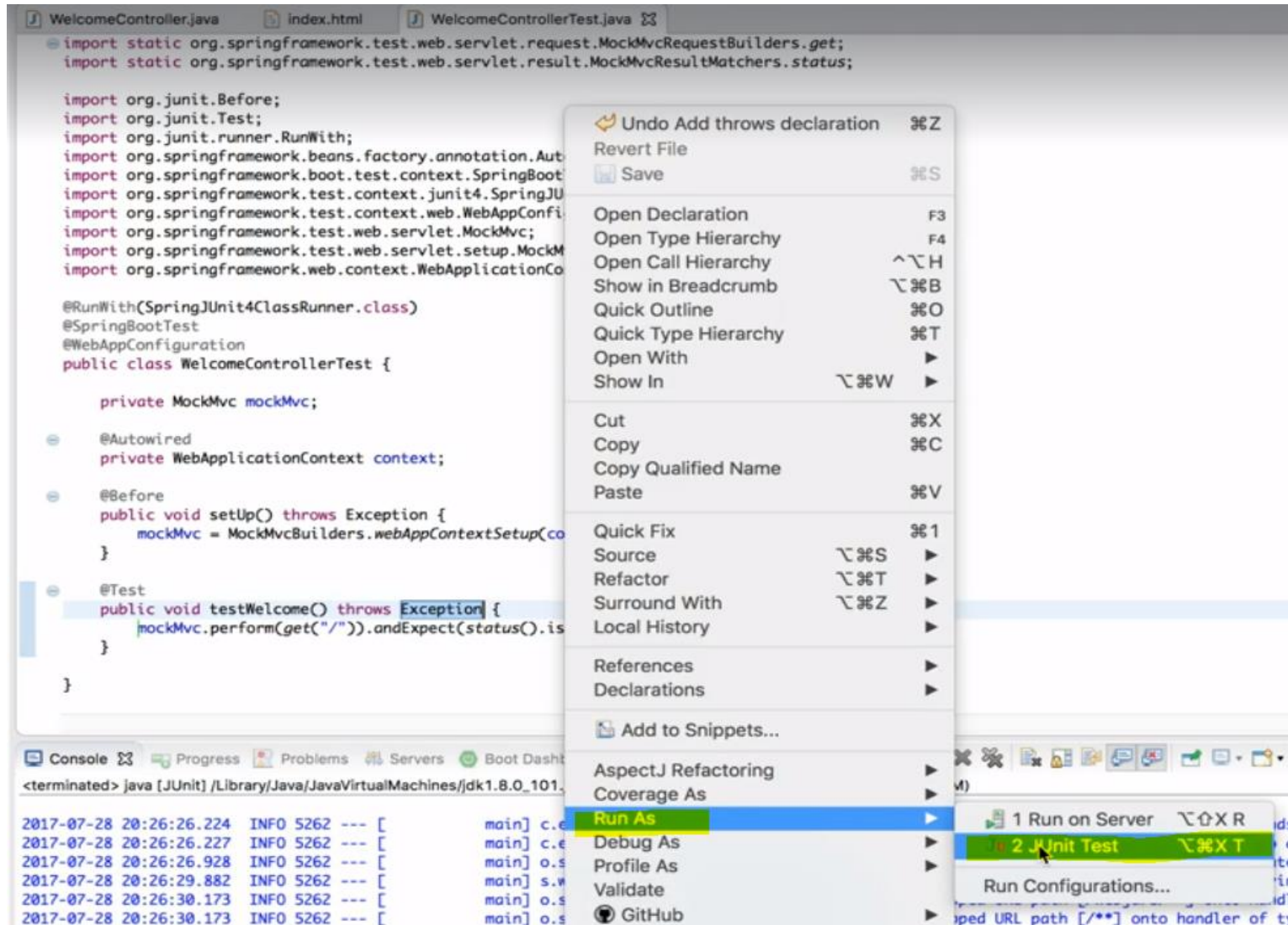
    private MockMvc mockMvc;

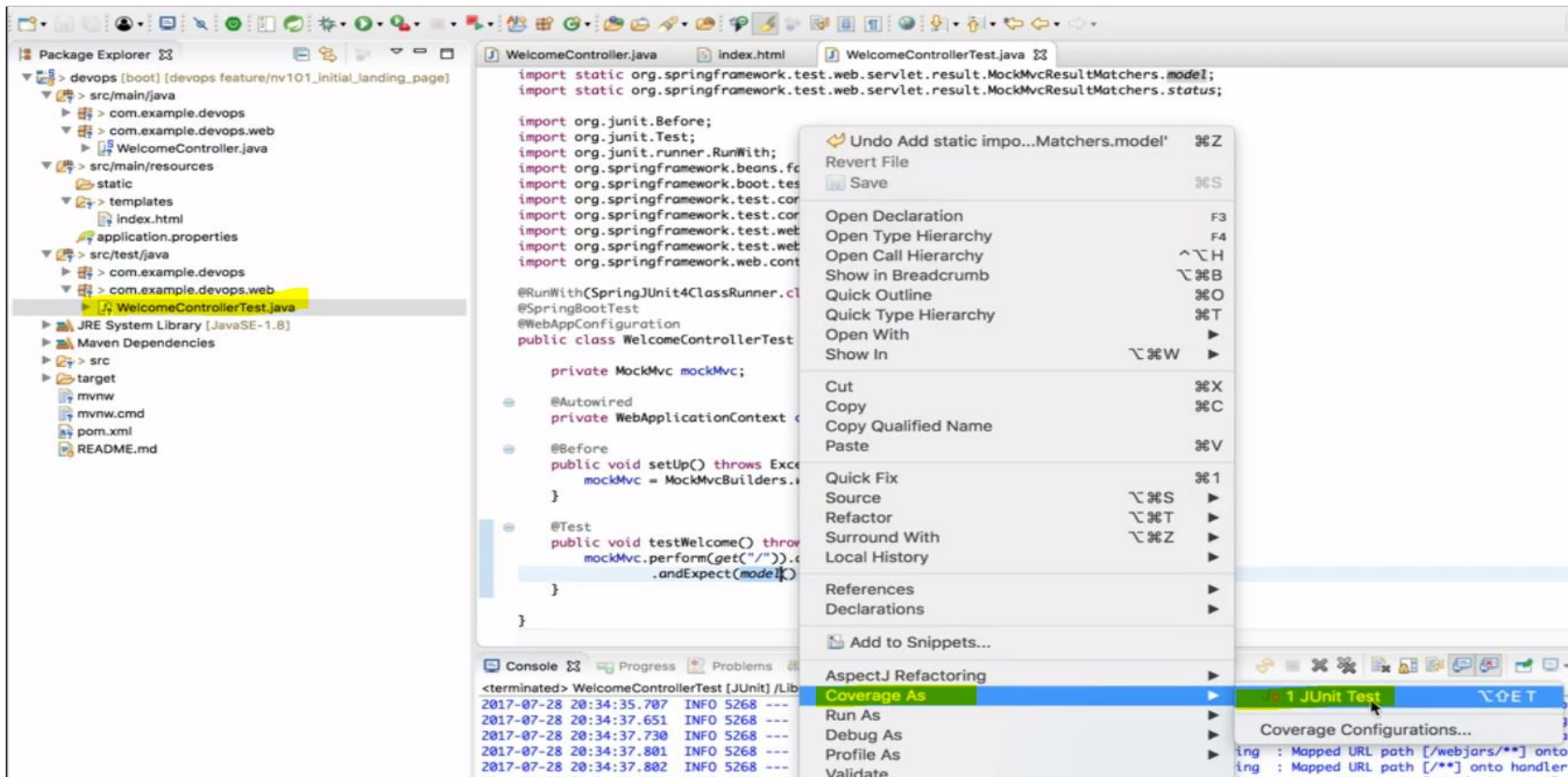
    @Autowired
    private WebApplicationContext context;

    @Before
    public void setUp() throws Exception {
        mockMvc = MockMvcBuilders.webAppContextSetup(context).build();
    }

    @Test
    public void testWelcome() throws Exception {
        mockMvc.perform(get("/").andExpect(status().isOk())
            .andExpect(model().attribute("course", containsString("DevOps"))));
    }
}
```

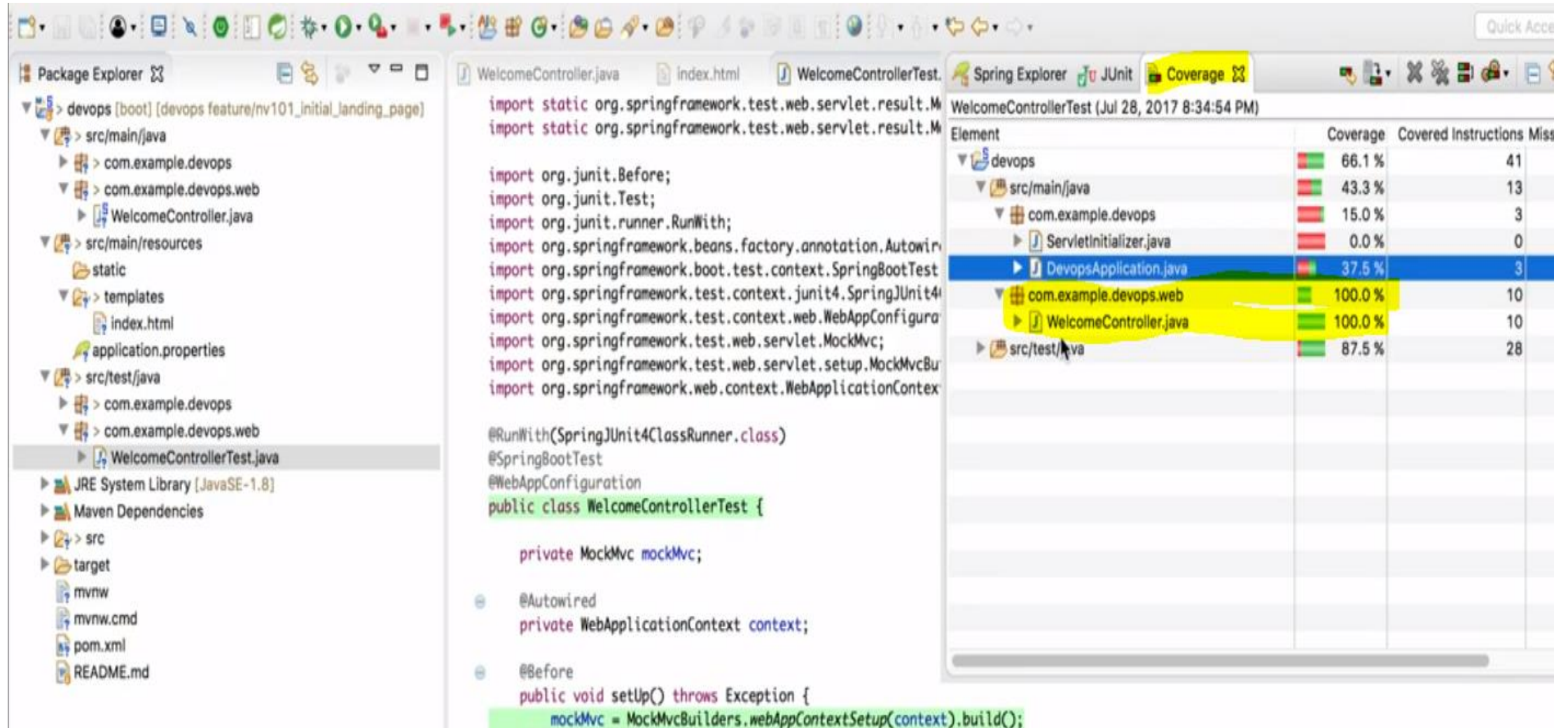
# Right click on & Run as Junit Test







# Execute the Coverage & Check the Controller %

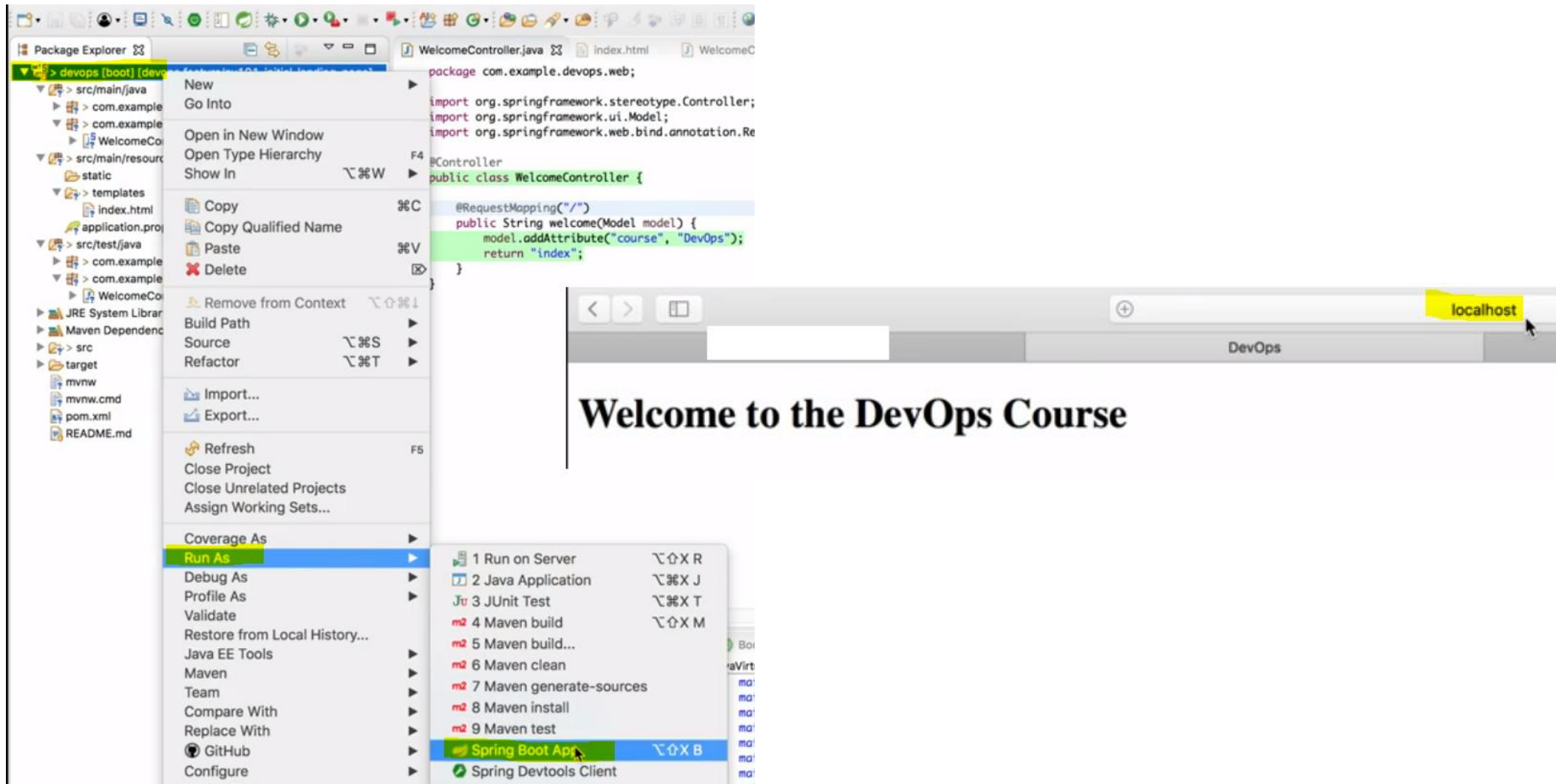


The screenshot displays an IDE interface with three main panels:

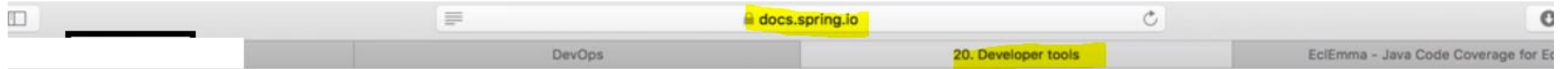
- Package Explorer (Left):** Shows the project structure. The path `src/test/java/com/example/devops/web/WelcomeControllerTest.java` is selected.
- Editor (Center):** Displays the `WelcomeControllerTest.java` file. The code includes imports for Spring Framework test utilities, JUnit, and Spring Boot test annotations. The test class `WelcomeControllerTest` is annotated with `@RunWith(SpringJUnit4ClassRunner.class)`, `@SpringBootTest`, and `@WebAppConfiguration`. It contains a `setUp()` method that initializes a `MockMvc` instance using `MockMvcBuilders.webAppContextSetup(context).build()`.
- Coverage View (Right):** Shows the coverage results for the test run. The table below summarizes the data:

Element	Coverage	Covered Instructions	Miss
devops	66.1 %	41	
src/main/java	43.3 %	13	
com.example.devops	15.0 %	3	
ServletInitializer.java	0.0 %	0	
DevopsApplication.java	37.5 %	3	
com.example.devops.web	100.0 %	10	
WelcomeController.java	100.0 %	10	
src/test/java	87.5 %	28	

# Run the App & Check the Results on Browser



Go to >> docs.spring.io & copy maven code & add under POM.xml file



## 20. Developer tools

### Part III. Using Spring Boot

## 20. Developer tools

Spring Boot includes an additional set of tools that can make the application development experience a little more pleasant. The `spring-boot-devtools` module can be included in any project to provide additional development-time features. To include devtools support, simply add the module dependency to your build:

### Maven.

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

### Gradle.

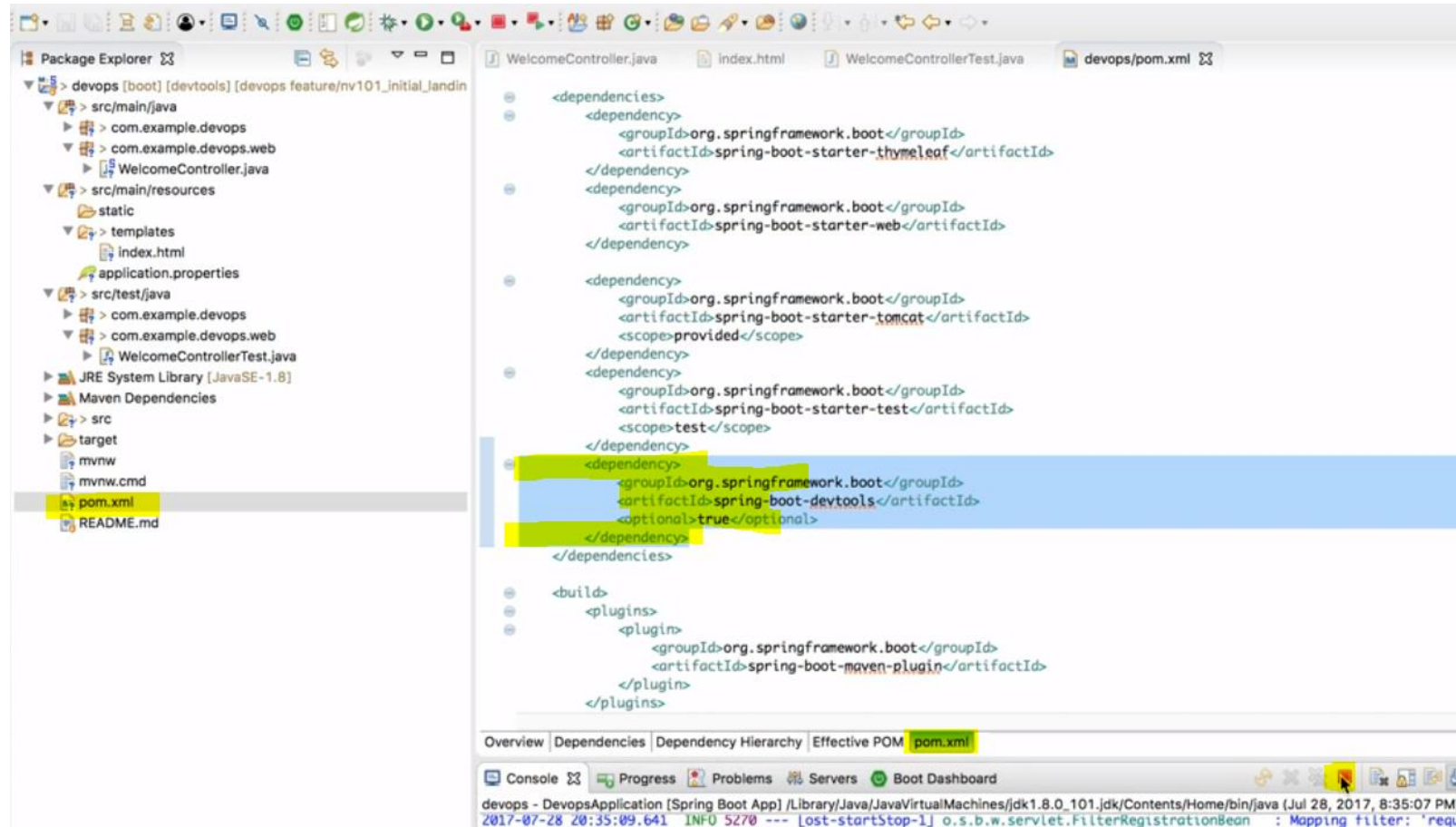
```
dependencies {
    compile("org.springframework.boot:spring-boot-devtools")
}
```



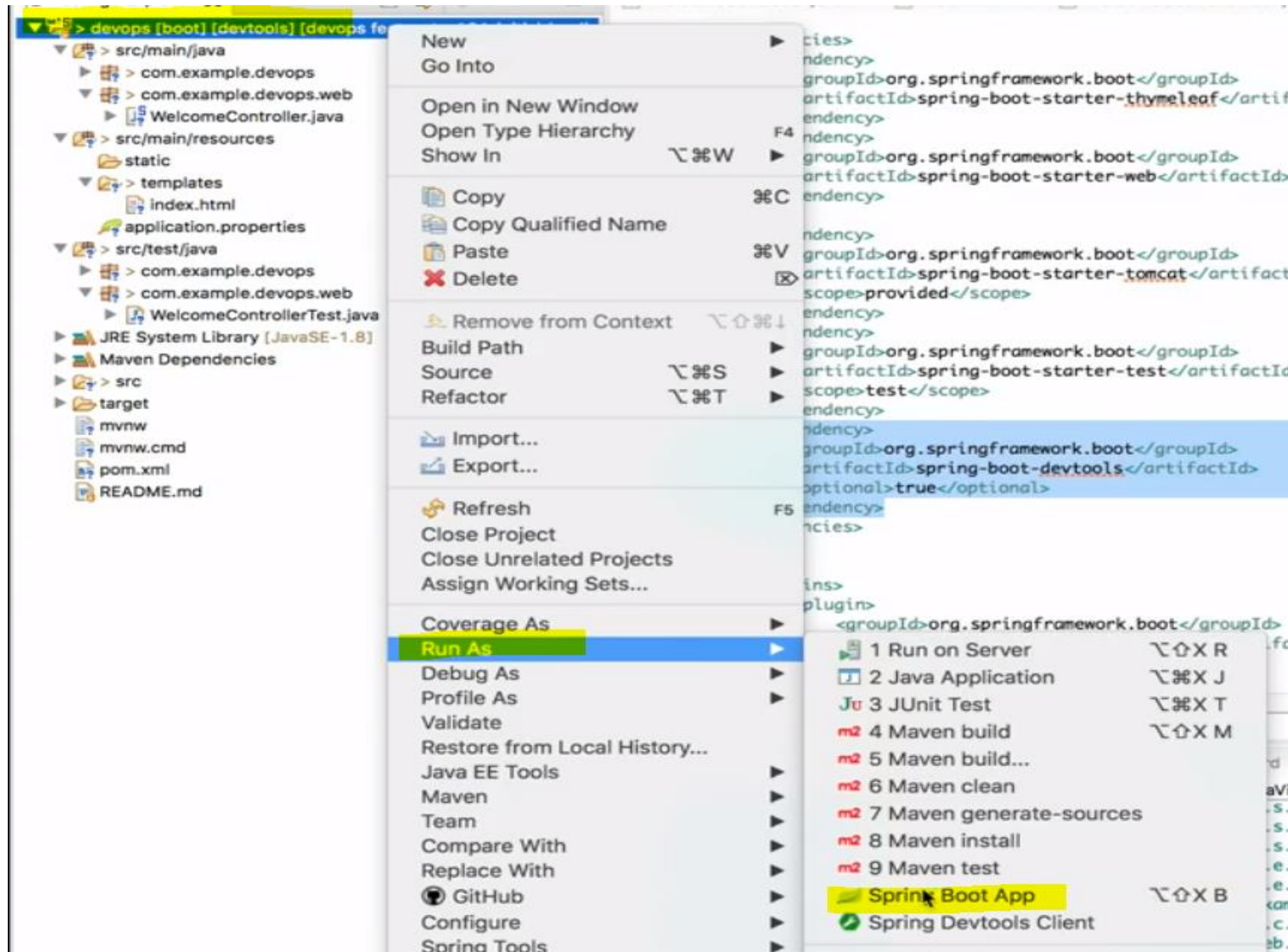
<https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-devtools.html>

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

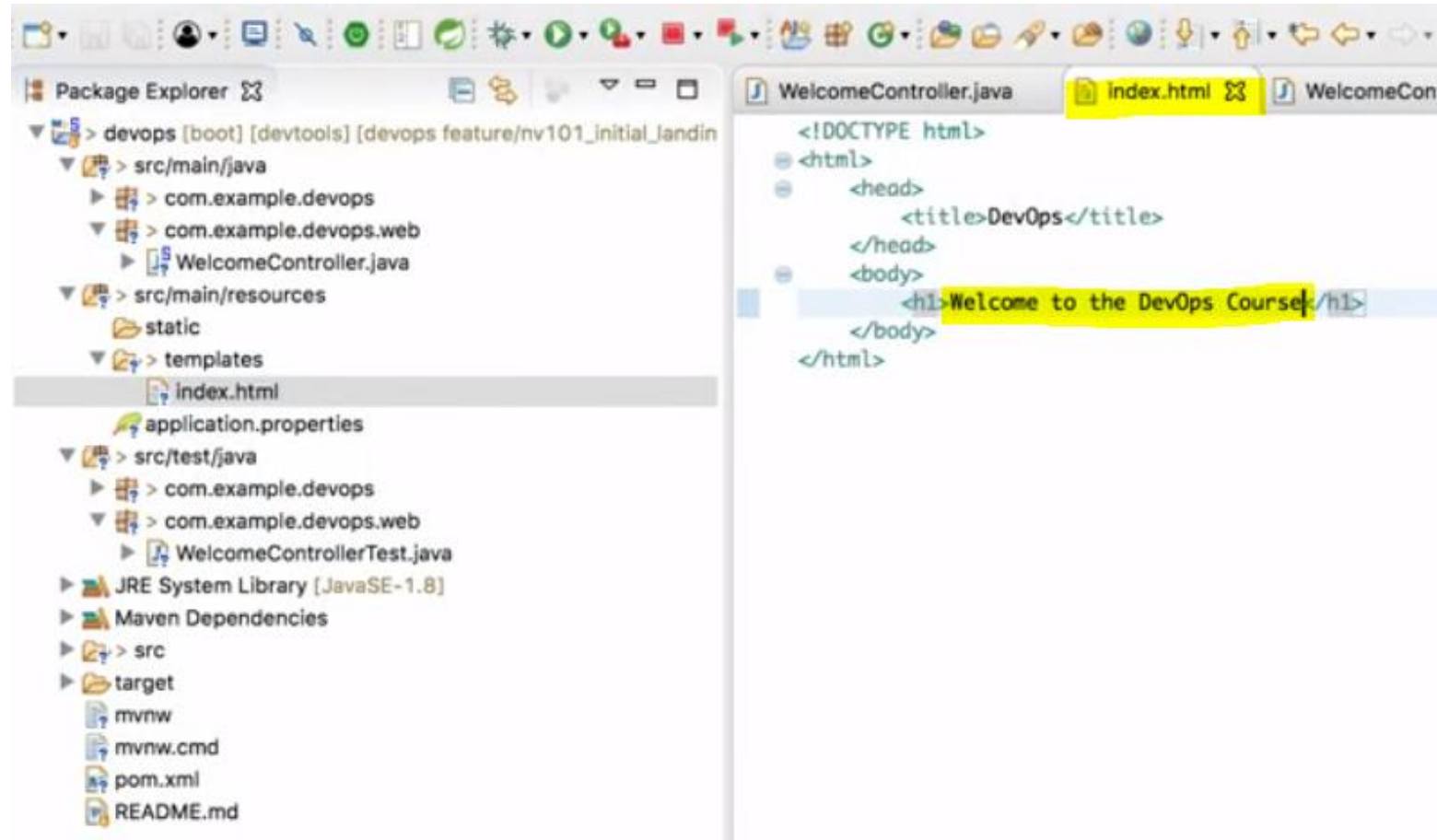
# Go to POM.xml file & Add Maven Dependencies element & Stop development server



# Run the Development Server



Now, we can modify the index.html file and no need to stop & start development server



Go to the Browser & Check

**Welcome to the DevOps Course**