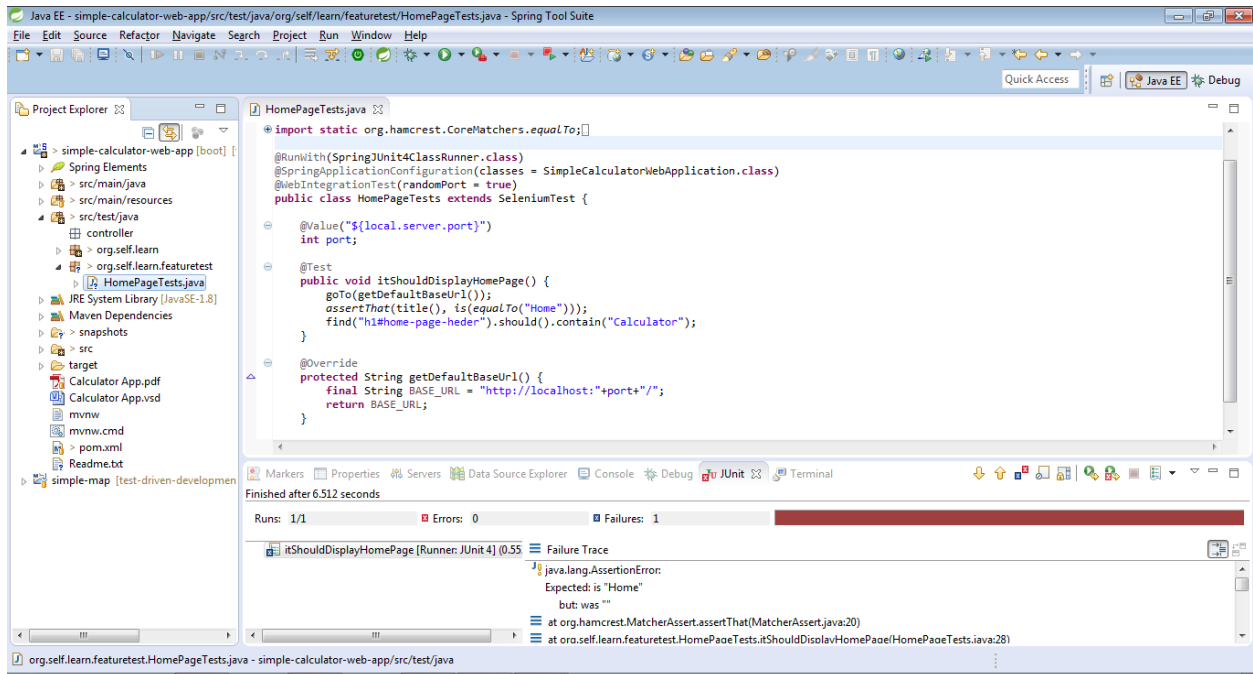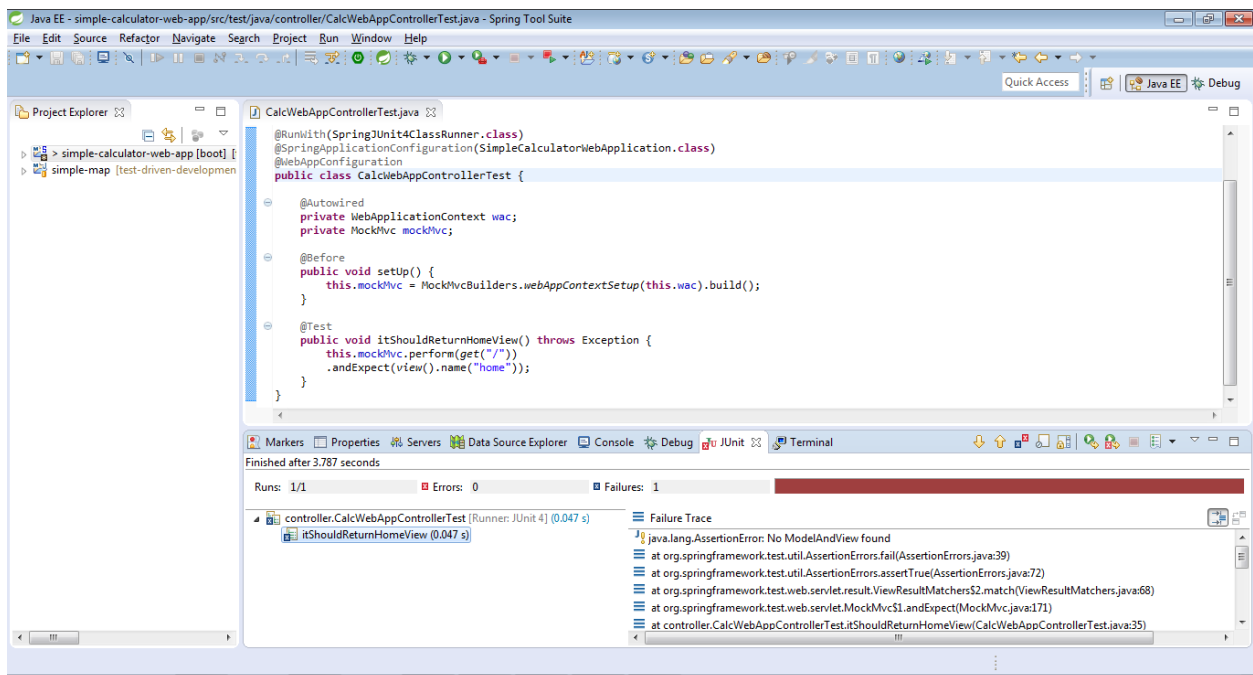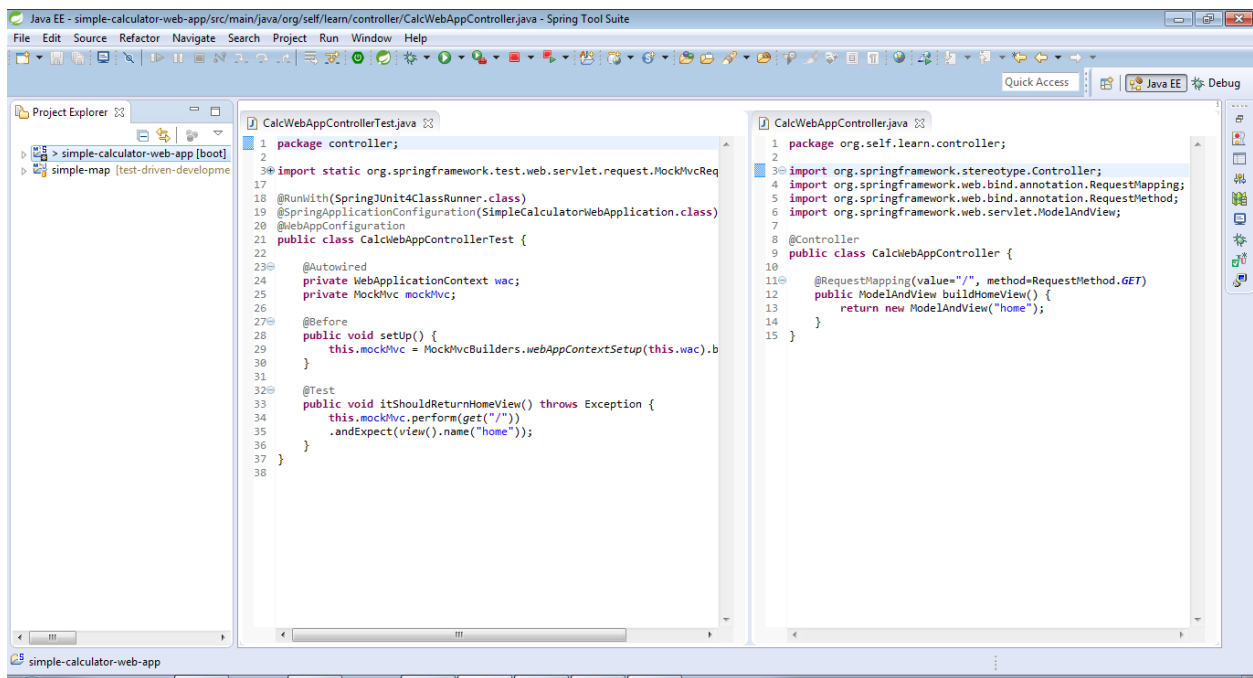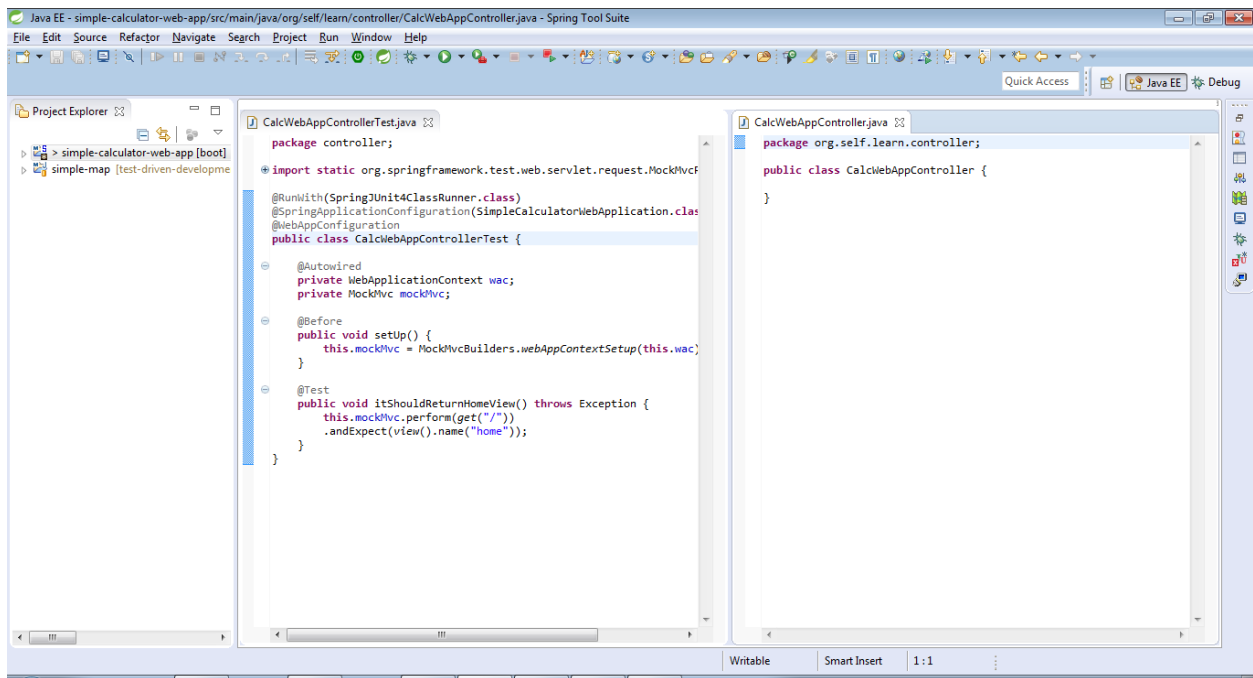Let's start with a feature test… As expected, it fails!

Let's hop on to our controller test which fails as well, as we expect it to.

Let's put some implementation together for the controller…

Let's run the test again… It still fails since we have not created a view ("home") yet…

Let's re-run the Home Page Feature Test too, which fails as expected…

Let's put our home view together. It's a simple HTML doing exactly what the test is expecting it to do. Let's re-run our tests now…
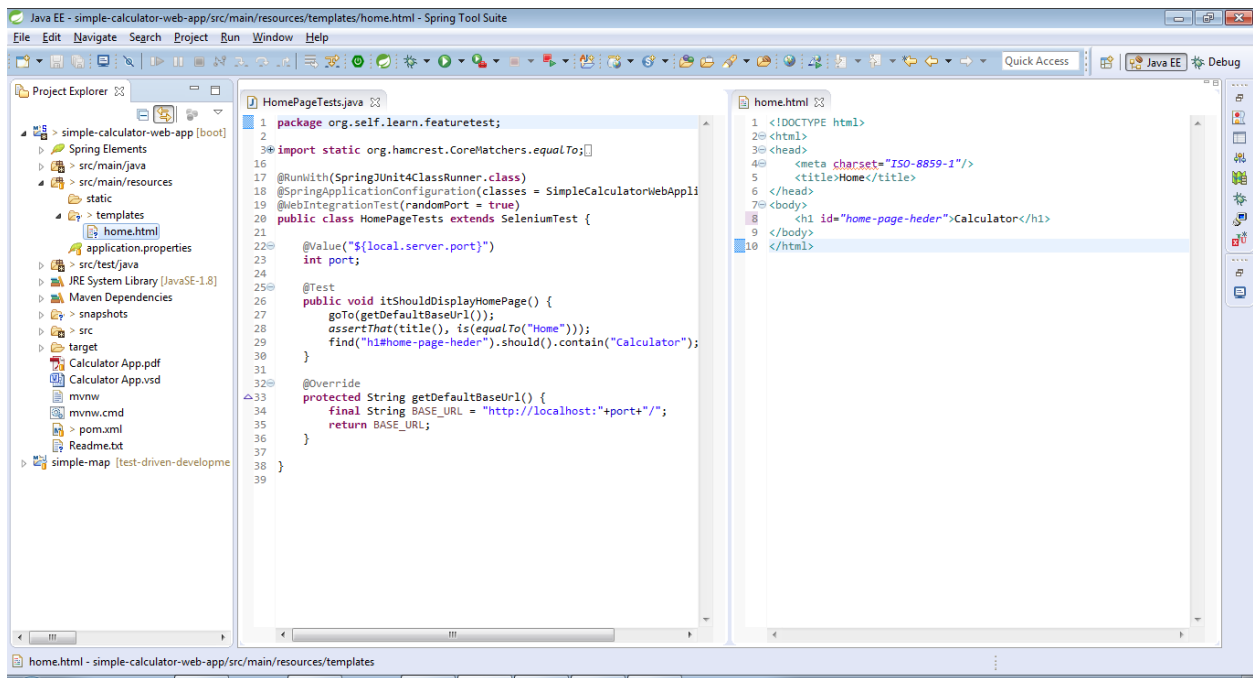
The tests did complete successfully this time.
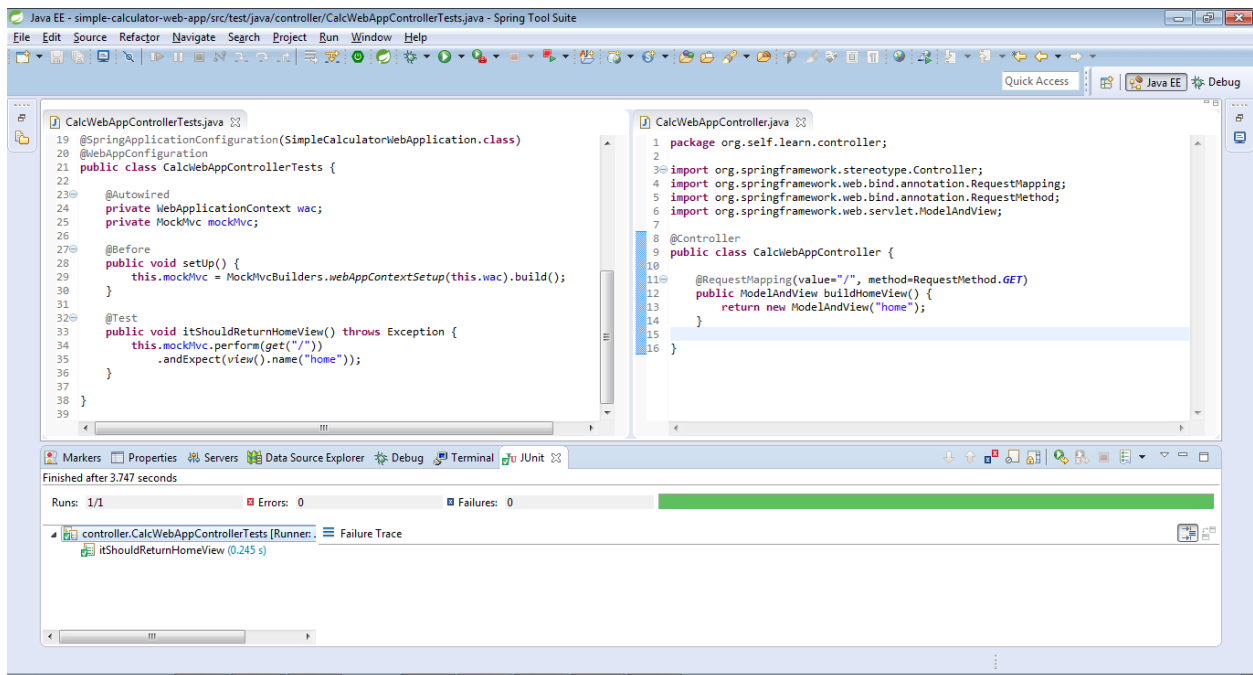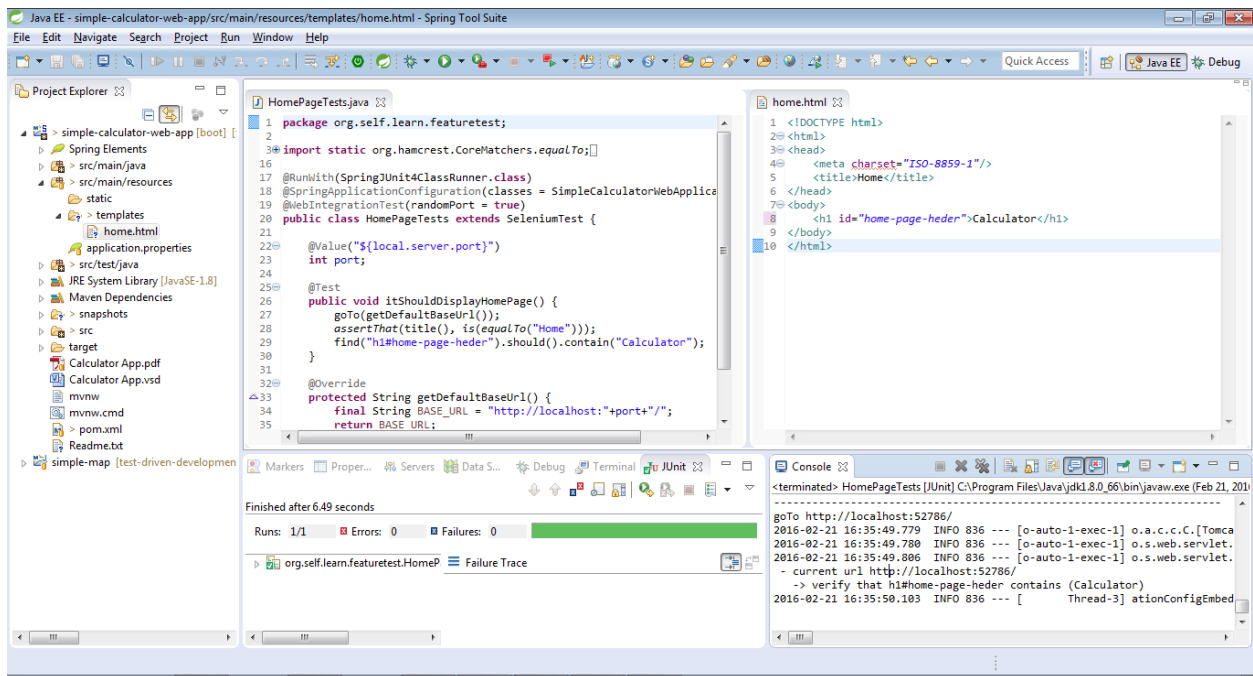
We will now write a few other tests and corresponding implementations (one after another of course) as per wireframe. I am providing screenshot with the final set of tests and implementation for reference…
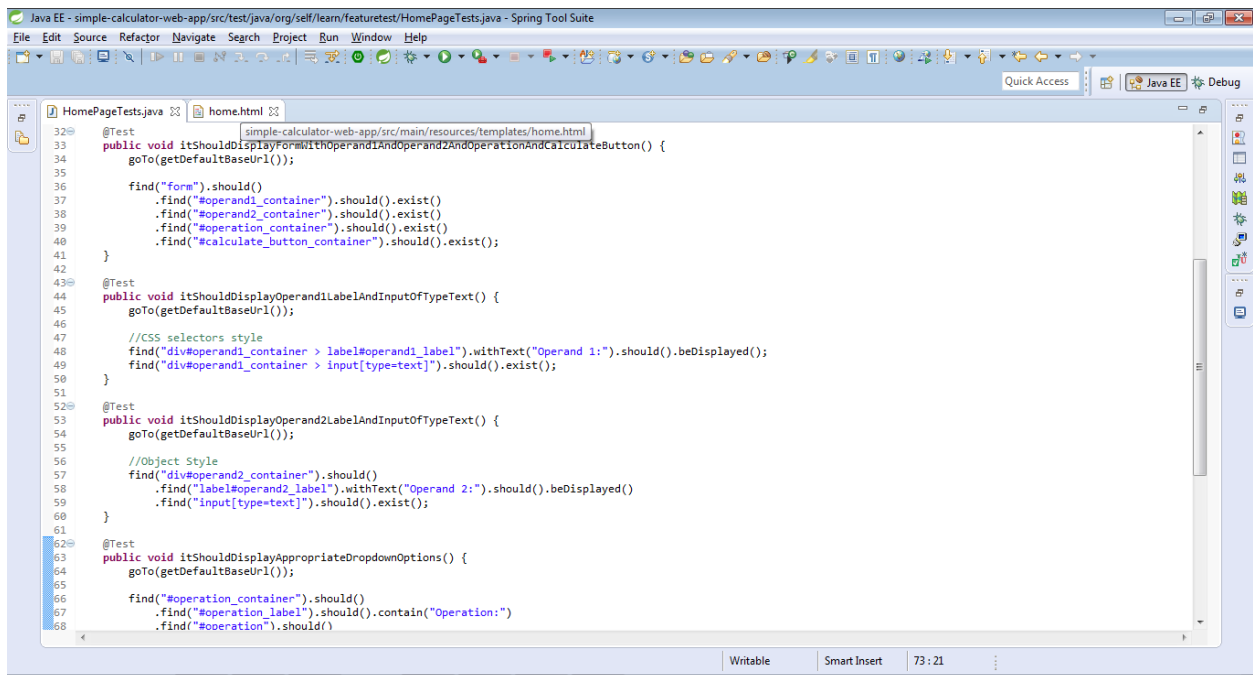
```java
55
56              //Object Style
57              find("div#operand2_container").should()
58                  .find("label#operand2_label").withText("Operand 2:").should().beDisplayed()
59                  .find("input[type=text]").should().exist();
60          }
61
62⊖         @Test
63          public void itShouldDisplayAppropriateDropdownOptions() {
64              goTo(getDefaultBaseUrl());
65
66              find("#operation_container").should()
67                  .find("#operation_label").should().contain("Operation:")
68                  .find("#operation").should()
69                      .find("option[value='']").withText("Specify operation").should().exist()
70                      .find("option[value=addition]").withText("Addition (+)").should().exist()
71                      .find("option[value=subtraction]").withText("Subtraction (-)").should().exist()
72                      .find("option[value=multiplication]").withText("Multiplication (*)").should().exist()
73                      .find("option[value=division]").withText("Division (/)").should().exist();
74          }
75
76⊖         @Test
77          public void itShouldDisplayCalculateButtonInputOfTypeSubmit() {
78              goTo(getDefaultBaseUrl());
79
80              find("#calculate_button_container").should()
81                  .find("input[type=submit][value=Calculate]").should().exist();
82          }
83
84⊖         @Override
△85         protected String getDefaultBaseUrl() {
86              final String BASE_URL = "http://localhost:"+port+"/";
87              return BASE_URL;
88          }
89
90      }
91
```
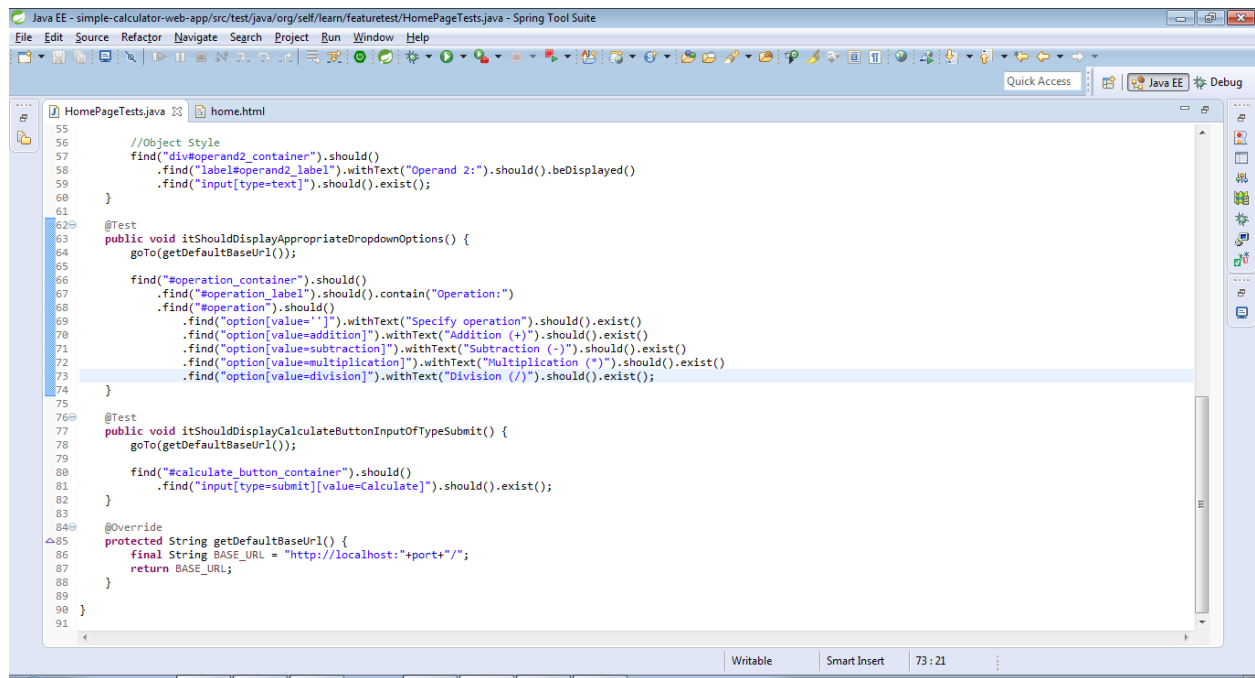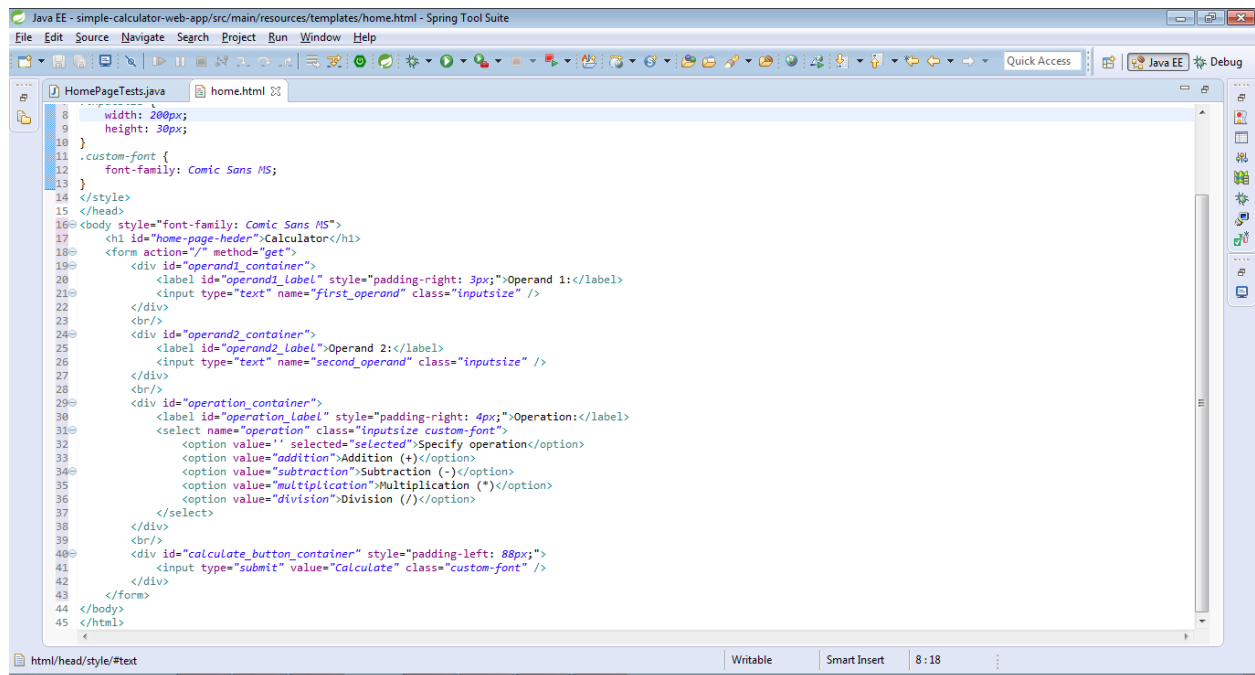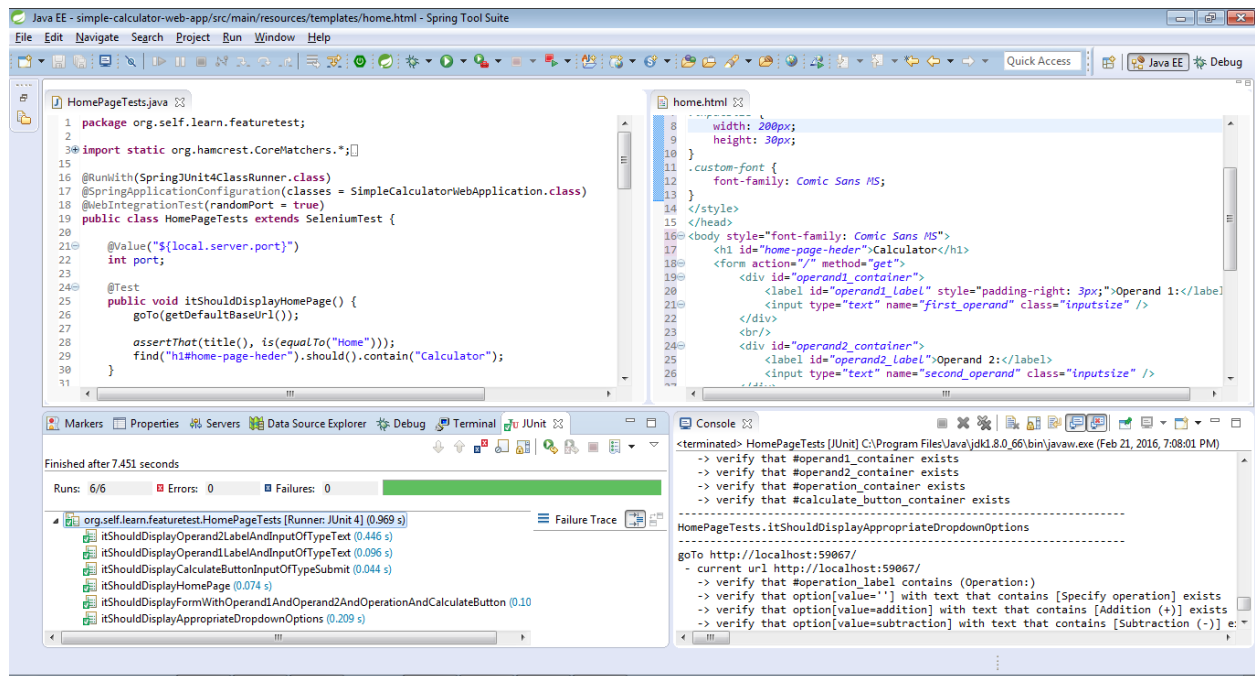
Note that we are only concerned about displaying the home page up to this point


```html
 8          width: 200px;
 9          height: 30px;
10      }
11      .custom-font {
12          font-family: Comic Sans MS;
13      }
14      </style>
15      </head>
16⊖     <body style="font-family: Comic Sans MS">
17          <h1 id="home-page-heder">Calculator</h1>
18⊖         <form action="/" method="get">
19⊖             <div id="operand1_container">
20                  <label id="operand1_label" style="padding-right: 3px;">Operand 1:</label>
21⊖                 <input type="text" name="first_operand" class="inputsize" />
22              </div>
23              <br/>
24⊖             <div id="operand2_container">
25                  <label id="operand2_label">Operand 2:</label>
26                  <input type="text" name="second_operand" class="inputsize" />
27              </div>
28              <br/>
29⊖             <div id="operation_container">
30                  <label id="operation_label" style="padding-right: 4px;">Operation:</label>
31⊖                 <select name="operation" class="inputsize custom-font">
32                      <option value='' selected="selected">Specify operation</option>
33                      <option value="addition">Addition (+)</option>
34⊖                     <option value="subtraction">Subtraction (-)</option>
35                      <option value="multiplication">Multiplication (*)</option>
36                      <option value="division">Division (/)</option>
37                  </select>
38              </div>
39              <br/>
40⊖             <div id="calculate_button_container" style="padding-left: 88px;">
41                  <input type="submit" value="Calculate" class="custom-font" />
42              </div>
43          </form>
44      </body>
45      </html>
```
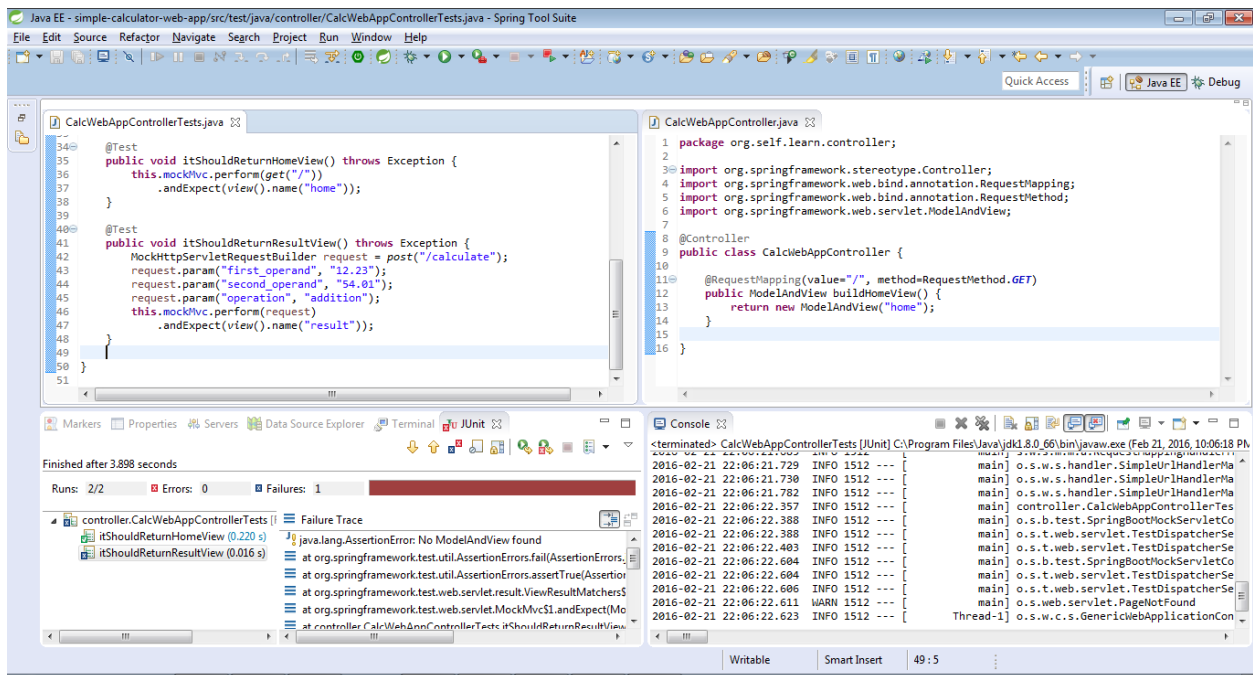
I also wrote a test case to verify transition away from home page upon clicking Calculate button
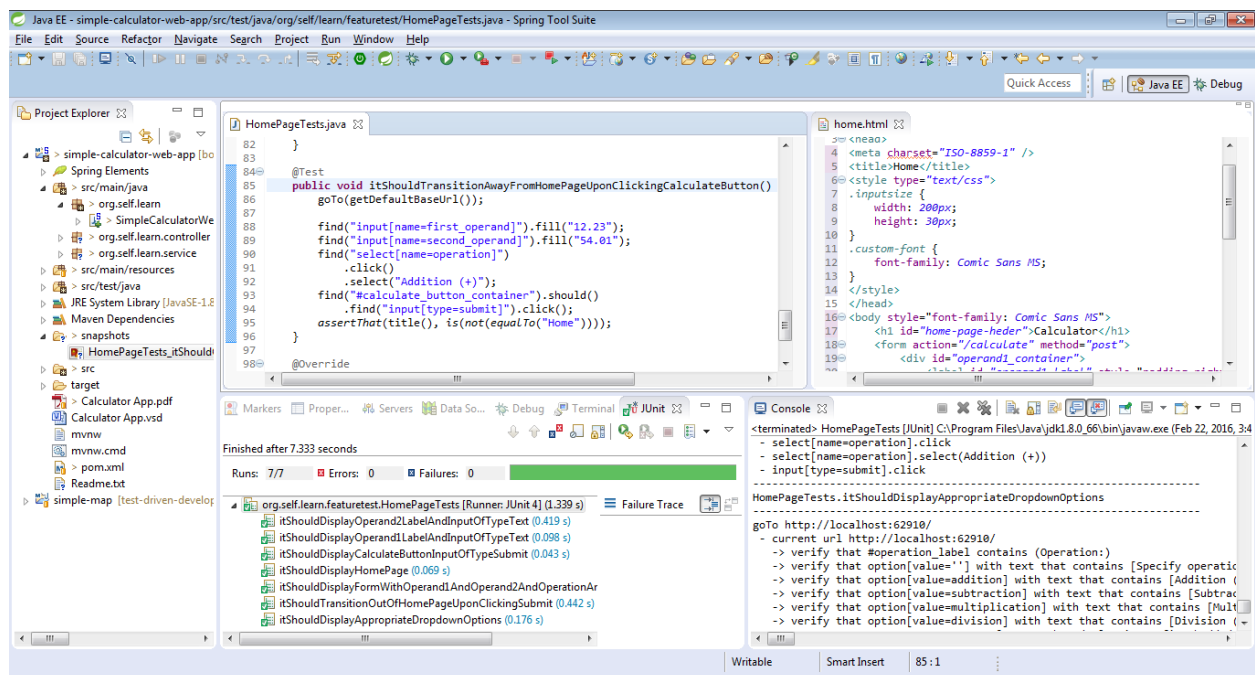
Now let's write a few test cases for the result page. When run, we expect the tests to fail and so they do…
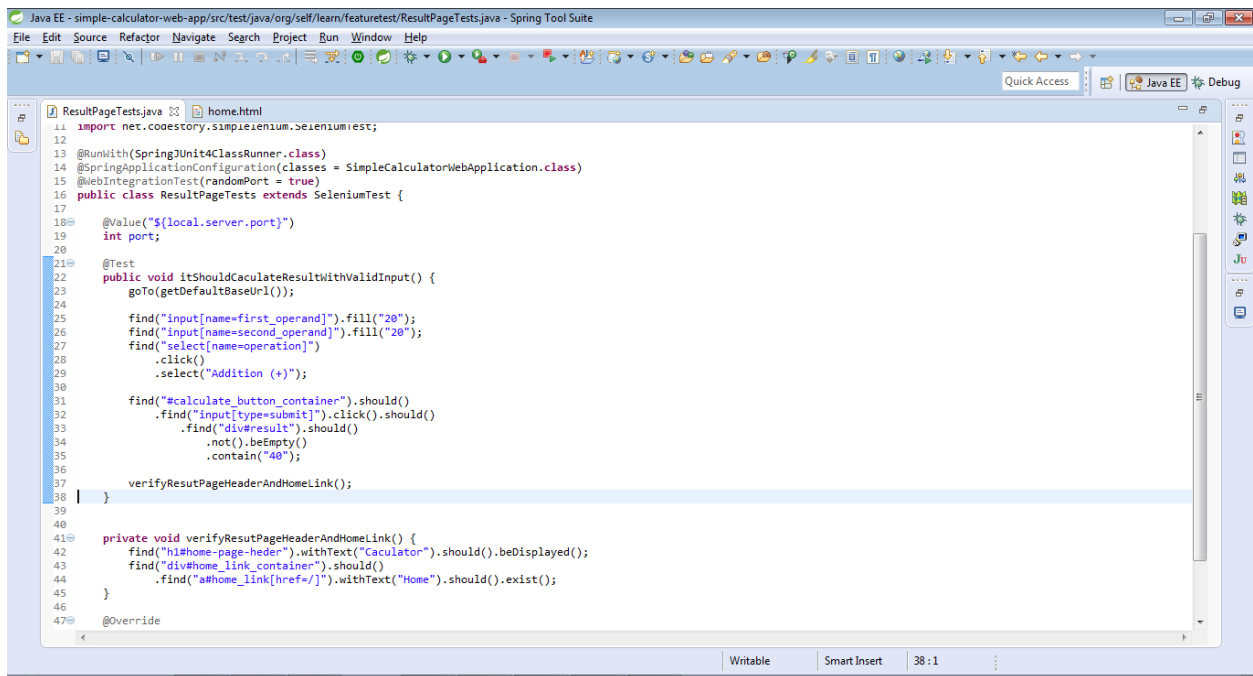
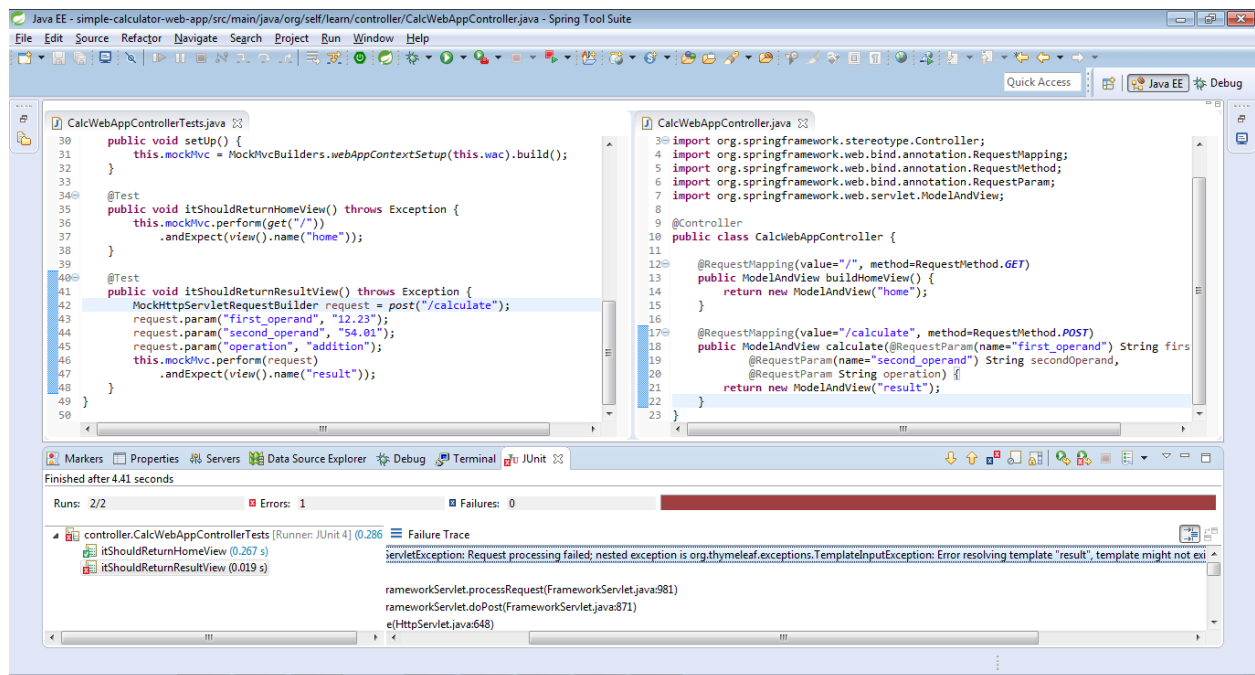We will require adding a controller end point and corresponding view to take care of this…

Here is the test case I have put together… which fails as expected. Let's implement the endpoint "/calculate"
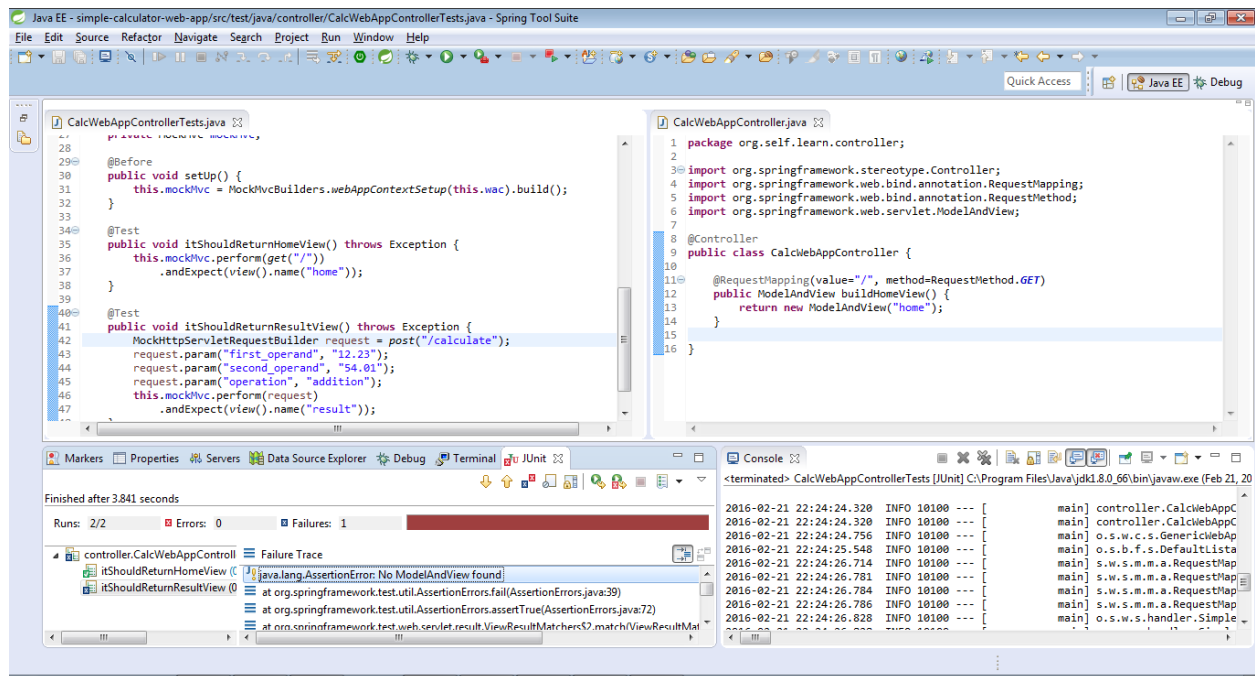
Although the endpoint exists now, the test still fails because the view "result" does not exist yet. Let's go create that.

I just wrote an empty HTML and re-ran the test. The test did complete successfully this time…
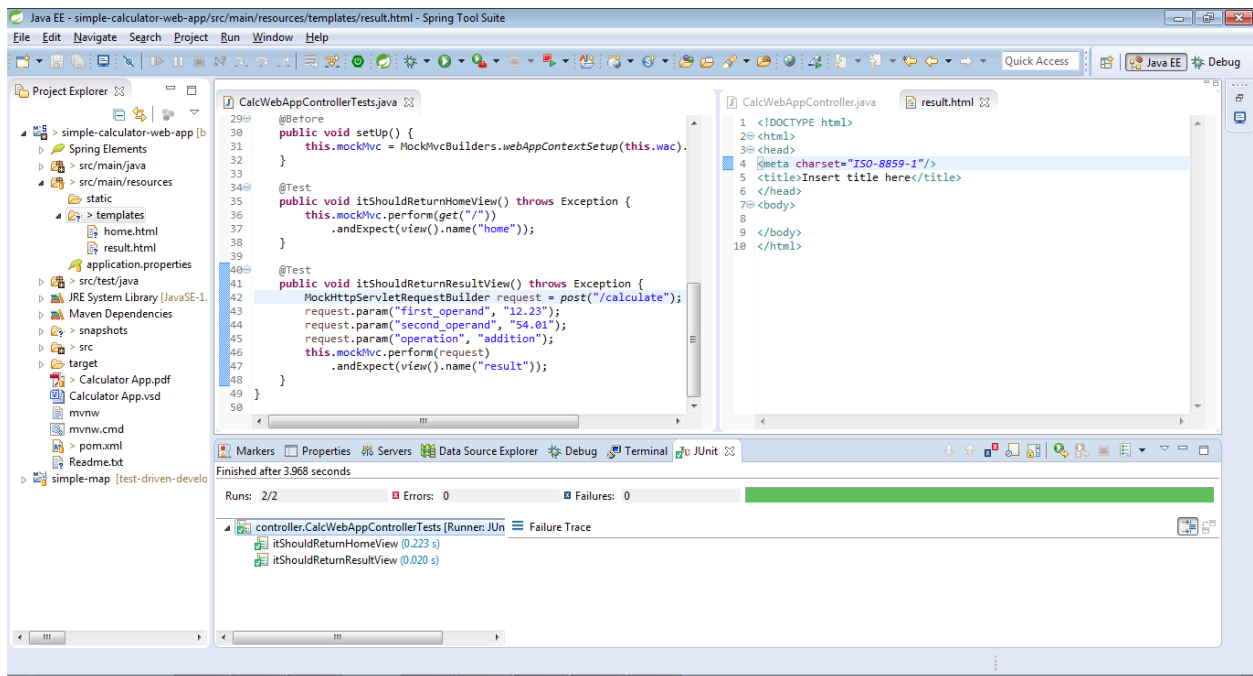
Before we go and add stuff to the result view, let's try to create the model containing the actual result. Our "result" view will have nothing to display otherwise.

We will like to use a service class for the actual computation. But we do not have to implement it yet...

Note that I intend to create a mock of the service class and inject the same into controller...
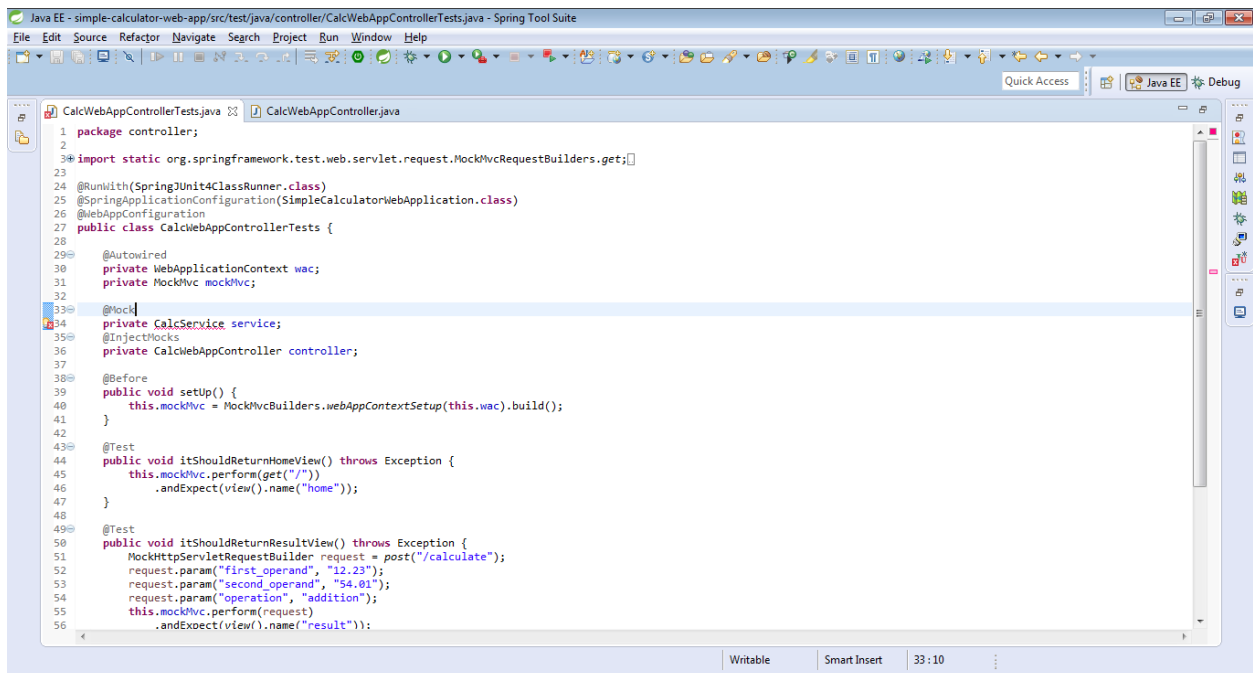
Let's create the service class skeleton to avoid compilation error
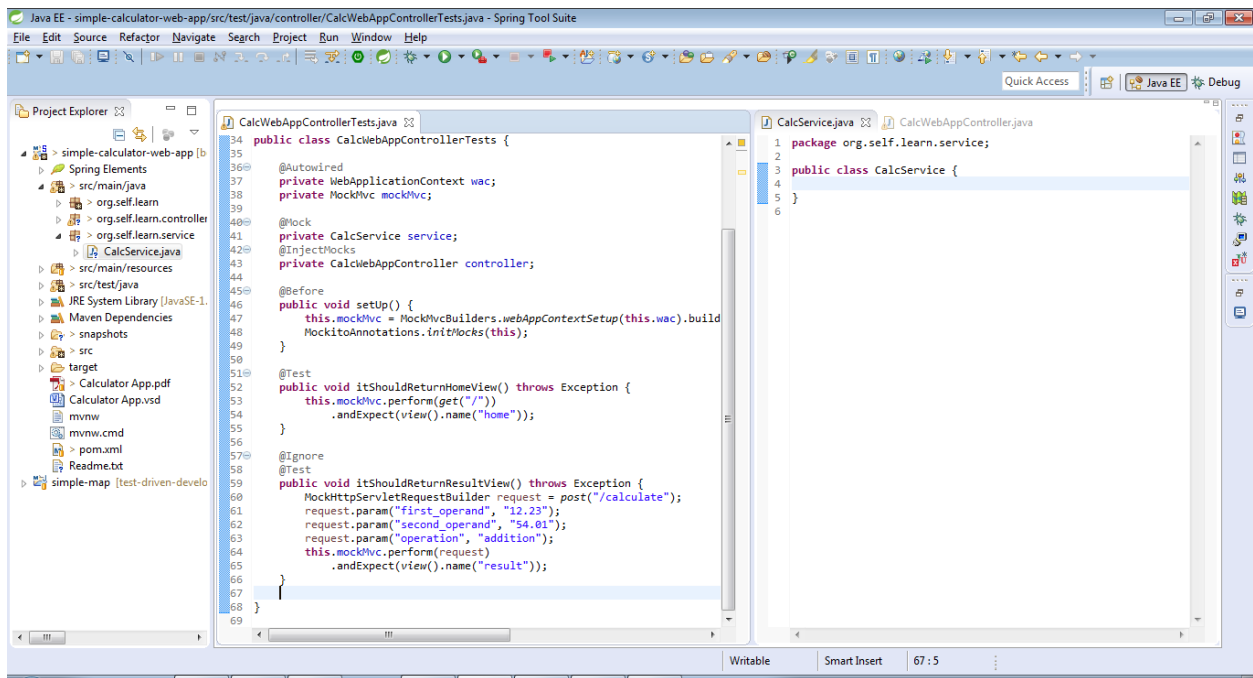
Next, I want to write tests for the controller by stubbing out service method using Mockito…
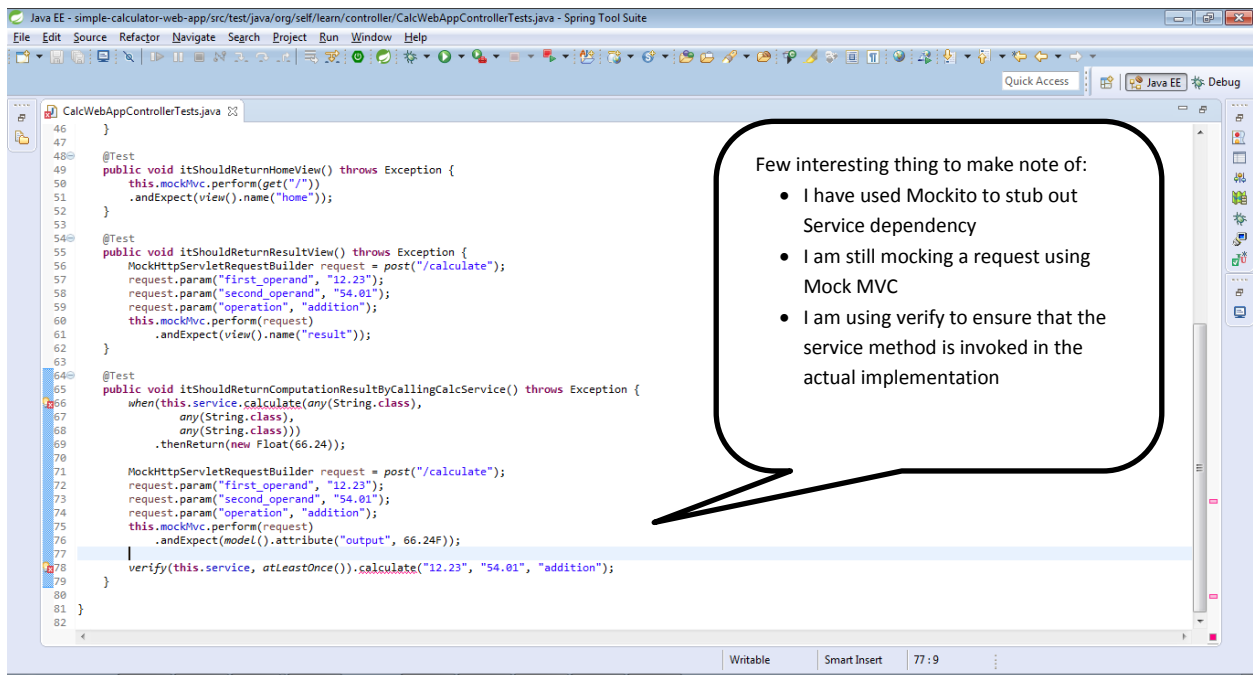
I created STS to auto-generate method in the service to resolve compilation errors.
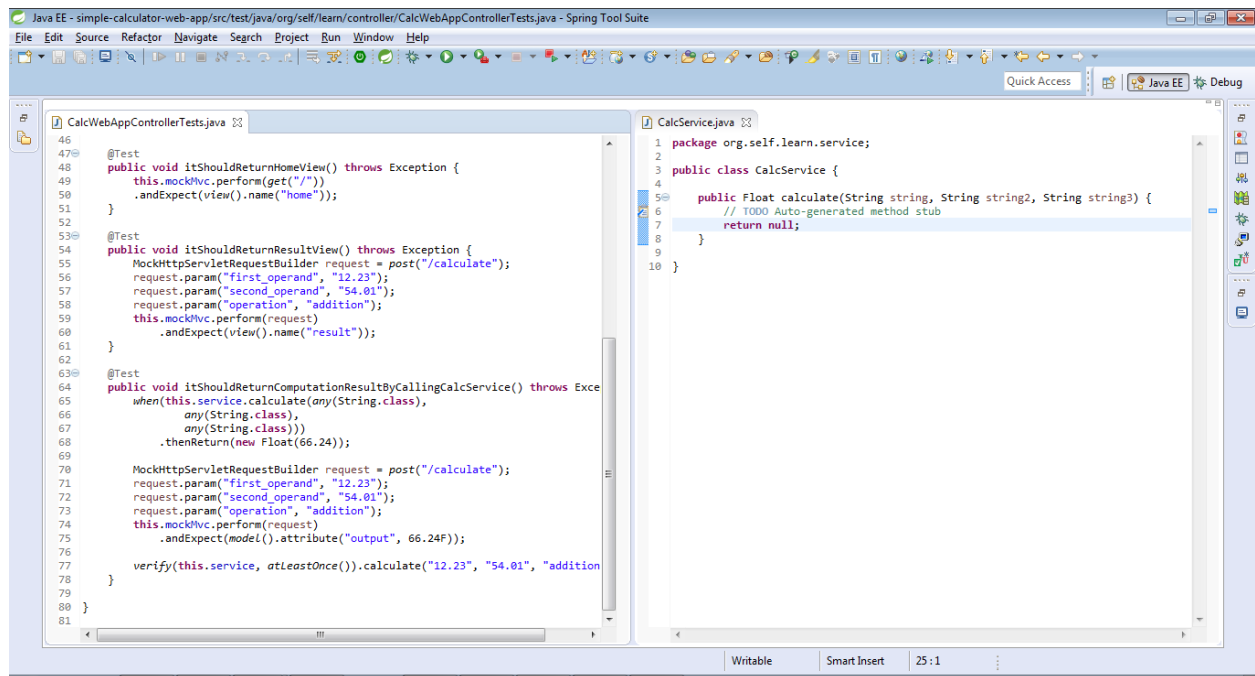
The compilation errors are gone, but we are not done quite yet! Let me explain why…

We are standing our mock MVC set up on the Webapplication context. It does not care about the controller instance (with mock service injected) that we created in the test class.
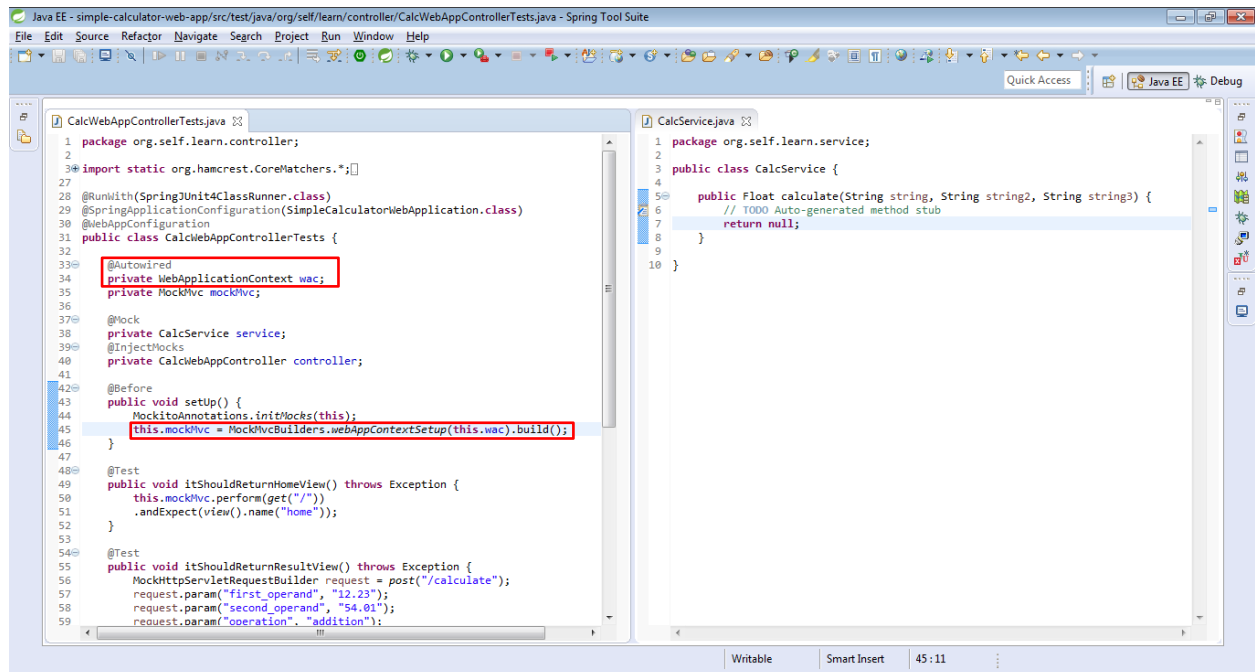
Let's make a quick modification to make the mock MVC object consider the controller object we created (which uses the mocked service instance and not the real one). And the following should take care of it.
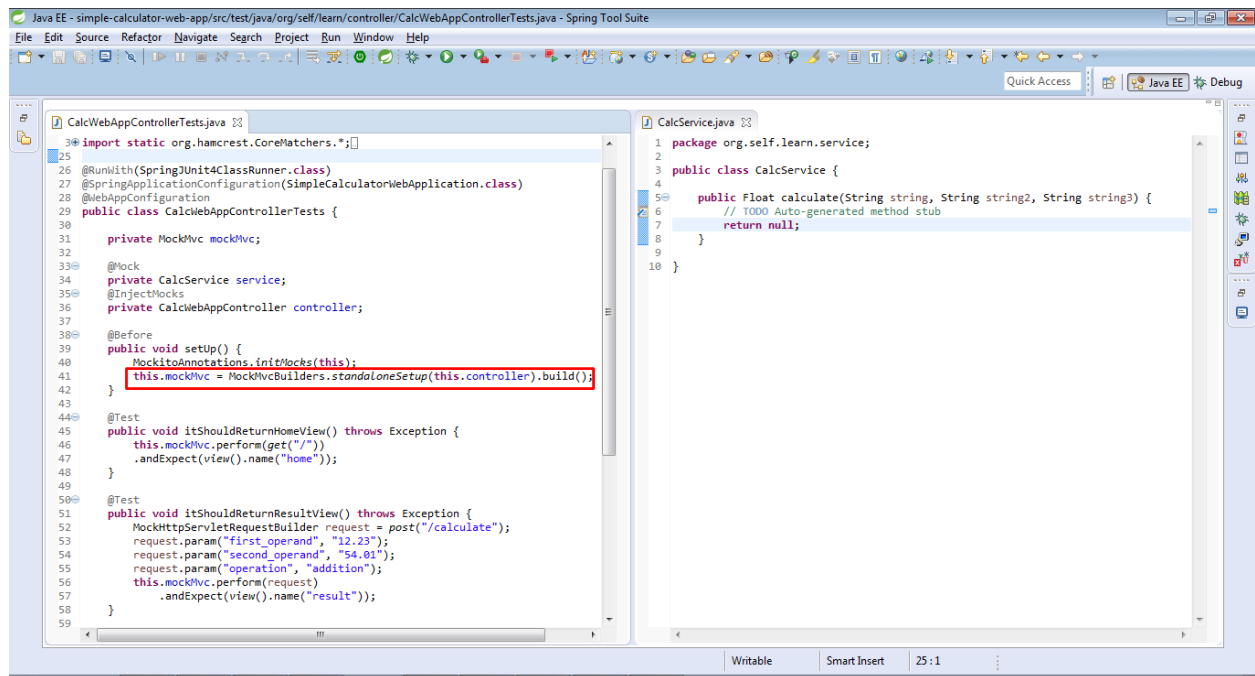
The test fails because we are neither returning a model nor are we calling the service.
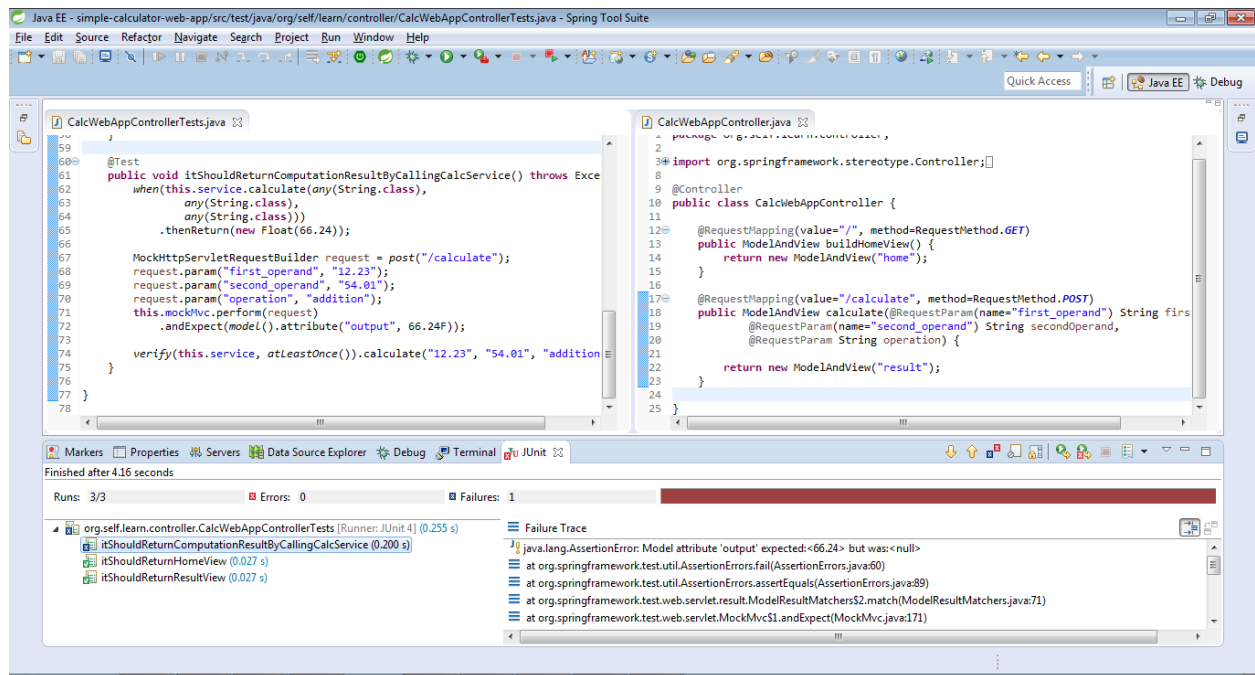
Next I tried to return a hard-coded response without making the actual service call. Although the Hamcrest assertion passes, the Mockito verification fails with the following error message:

```
Wanted but not invoked:
service.calculate(
    "12.23",
    "54.01",
    "addition"
```

```
);
-> at
controller.CalcWebAppControllerTests.itShouldReturnComputationResultByCallingCalcService(CalcWebAppControl
lerTests.java:70)
Actually, there were zero interactions with this mock.
```
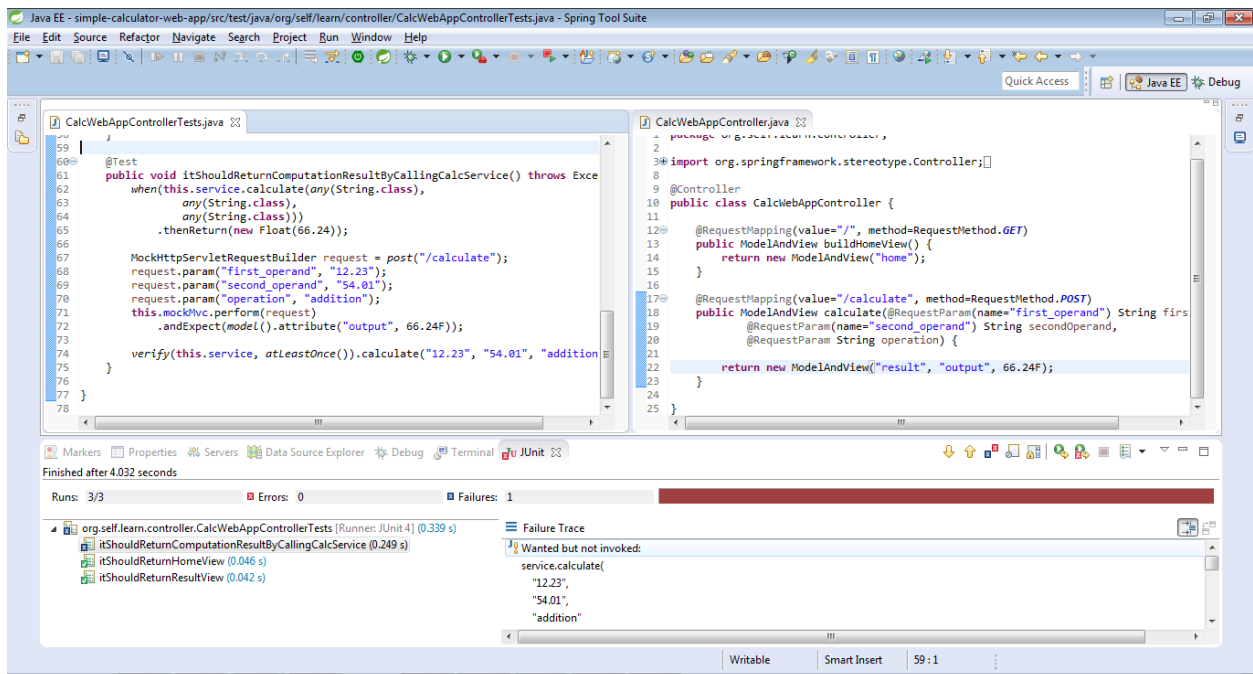

I was just trying to find a way to cheat the test here to show how powerful Mockito is!

Next let's try to actually invoke the service inside the controller and return what it returns.
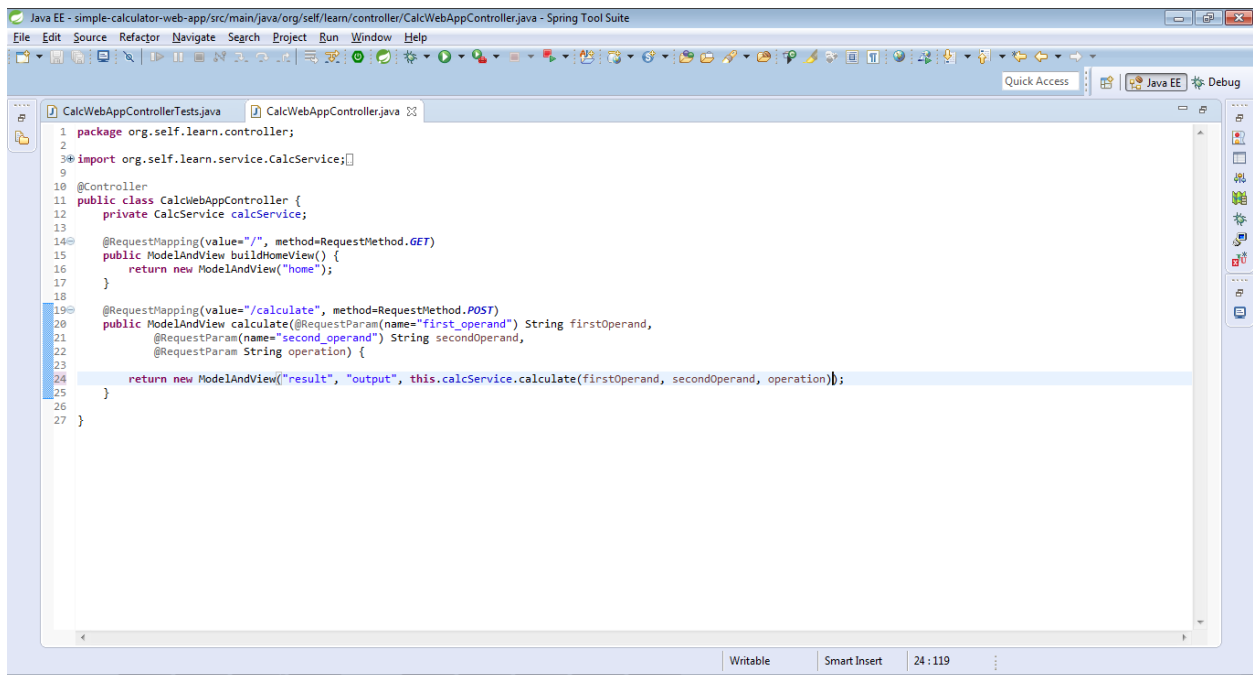
The following change should address the error above. The test result confirms that…
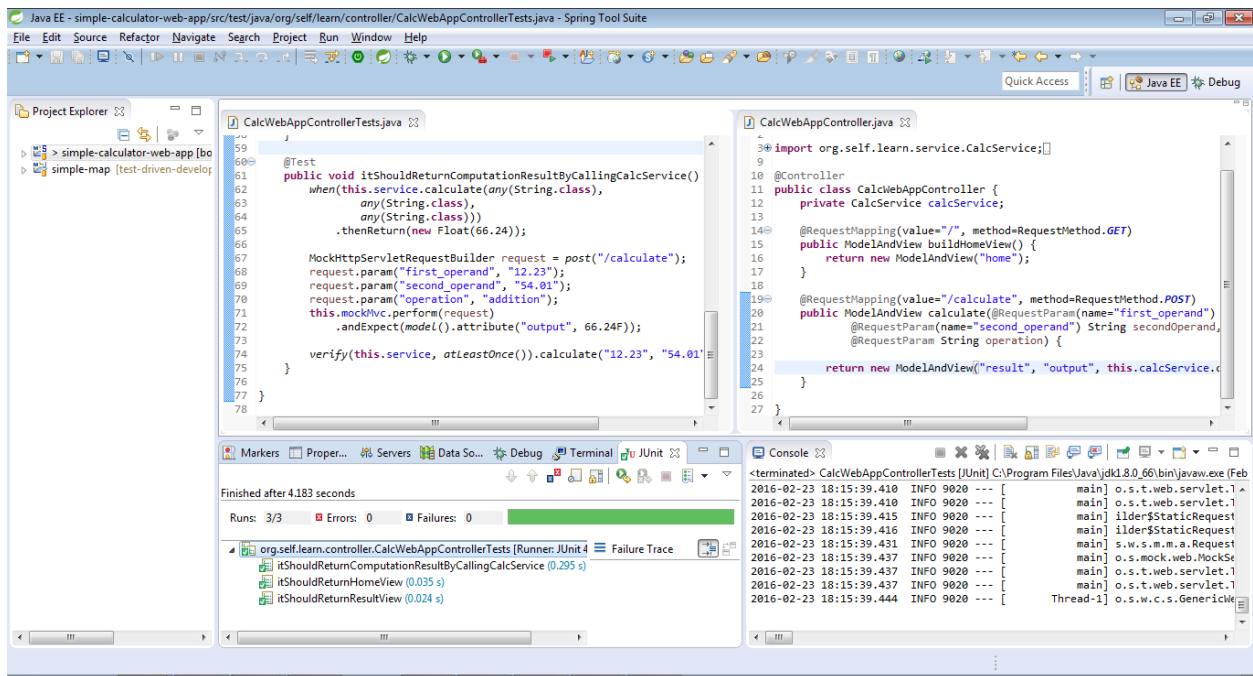
Note that we are able to test our controller regardless service implementation availability…

I have written a few more unit test cases for the controller and provided the final screenshot for reference.

Note: it is not customary to mix up controller tests and unit tests in the same class. Usually, you would want them to be in separate classes. I have added them in the same class for ease of reference.
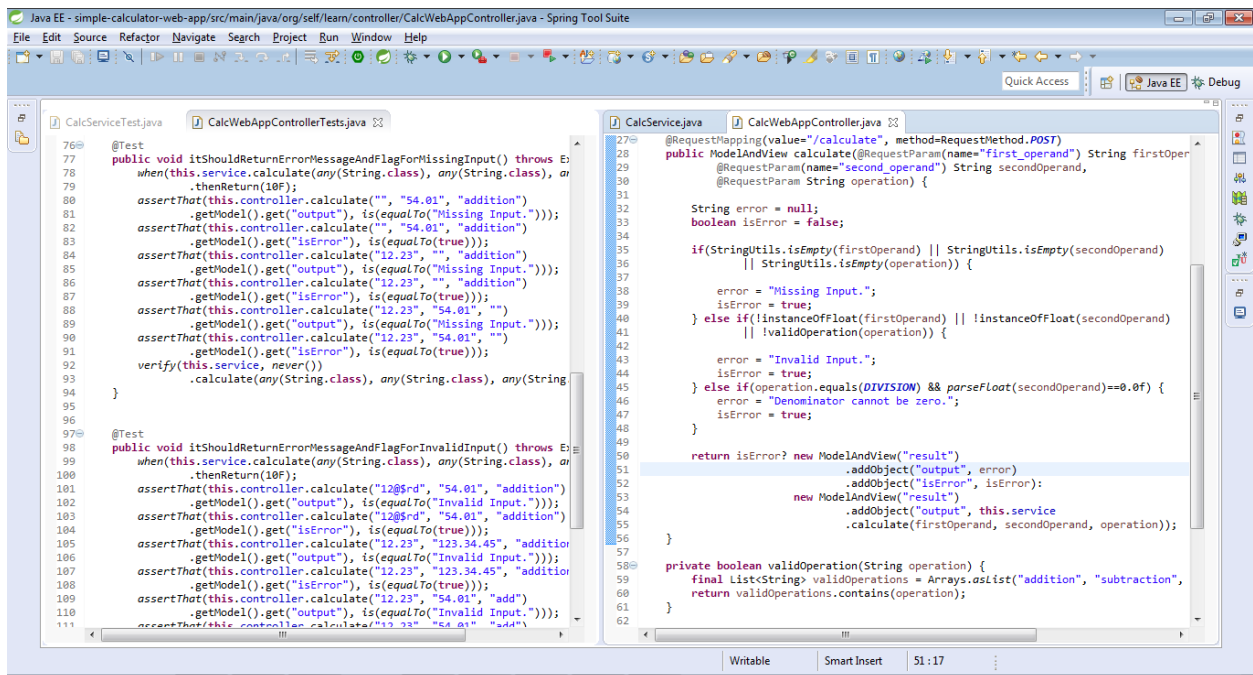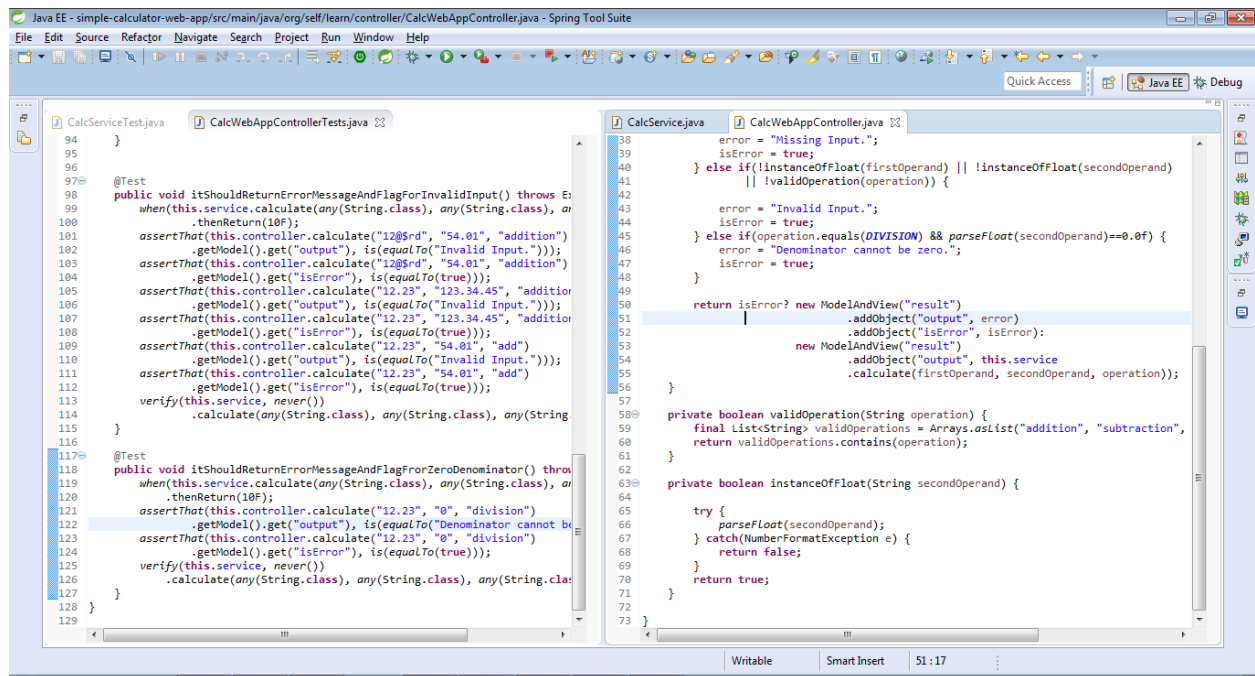
The test results are positive for all tests confirming completeness of our implementation.
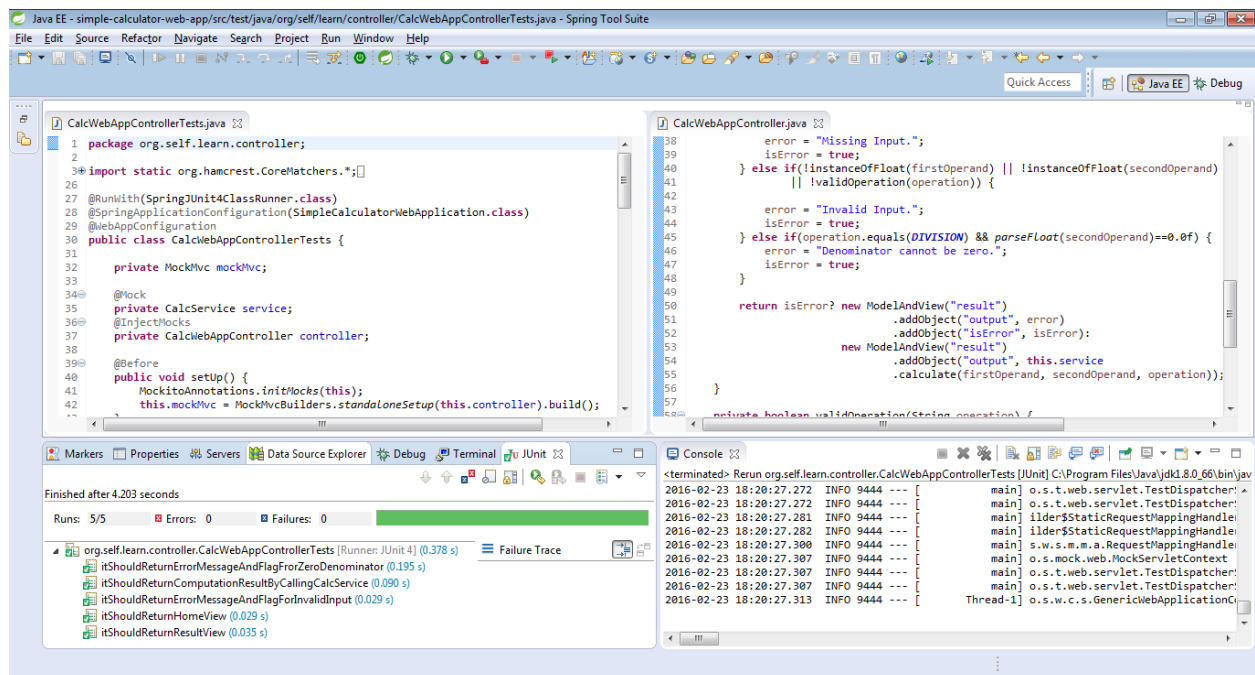
Now, let's get our service working… I will skip over to the final set of tests and implementation for the service as it is a much like another basic Java program.

Our Result page feature test will still fail… Let's fix the "result" view to get this working

Screenshot 1 — ResultPageTests.java:

```java
16  public class ResultPageTests extends SeleniumTest {
17
18      @Value("${local.server.port}")
19      int port;
20
21      @Test
22      public void itShouldCaculateResultWithValidInput() {
23          goTo(getDefaultBaseUrl());
24          enterInputAndClickCalculate("20", "20", "Addition (+)");
25          verifyResutPageHeaderAndHomeLink();
26          verifyDisplayedResult("40");
27      }
28
29      @Test
30      public void itShouldDisplayMissingInputError() {
31          goTo(getDefaultBaseUrl());
32          enterInputAndClickCalculate("20", "", "Addition (+)");
33          verifyResutPageHeaderAndHomeLink();
34          verifyDisplayedResult("Missing Input");
35      }
36
37      @Test
38      public void itShouldDisplayInvalidInputError() {
39          goTo(getDefaultBaseUrl());
40          enterInputAndClickCalculate("12.23.45", "12", "Addition (+)"
41          verifyResutPageHeaderAndHomeLink();
42          verifyDisplayedResult("Invalid Input");
43      }
44
45      @Test
46      public void itShouldDisplayZeroDenominatorError() {
47          goTo(getDefaultBaseUrl());
48          enterInputAndClickCalculate("67.23", "0", "Division (/)");
49          verifyResutPageHeaderAndHomeLink();
50          verifyDisplayedResult("Denominator cannot be zero.");
51      }
```

Screenshot 1 — result.html:

```html
1  <!DOCTYPE html>
2  <html xmlns="http://www.w3.org/1999/xhtml"
3      xmlns:th="http://www.thymeleaf.org">
4  <head>
5  <meta charset="ISO-8859-1" />
6  <title>Result</title>
7  </head>
8  <body style="font-family: Comic Sans MS">
9      <h1 id="result-page-heder">Calculator</h1>
10     <div id="result">
11         <p th:text="'Result: '+${output}" th:if="${isError==null}">
12         <p th:text="'Error: '+${output}" th:if="${isError}" style="
13     </div>
14     <div id="home_link_container">
15         <a id="home_link" href="/">Home</a>
16     </div>
17 </body>
18 </html>
```


Screenshot 2 — ResultPageTests.java:

```java
47          goTo(getDefaultBaseUrl());
48          enterInputAndClickCalculate("67.23", "0", "Division (/)");
49          verifyResutPageHeaderAndHomeLink();
50          verifyDisplayedResult("Denominator cannot be zero.");
51      }
52
53      private void verifyResutPageHeaderAndHomeLink() {
54          find("h1#result-page-heder").withText("Calculator").should()
55          find("div#home_link_container").should()
56              .find("a#home_link").withText("Home").should().exist();
57      }
58
59      private void enterInputAndClickCalculate(String firstOperand, St
60          find("input[name=first_operand]").fill(firstOperand);
61          find("input[name=second_operand]").fill(secondOperand);
62          find("select[name=operation]")
63              .click()
64              .select(operation);
65
66          find("#calculate_button_container").should()
67              .find("input[type=submit]").click();
68      }
69
70      private void verifyDisplayedResult(String check) {
71          find("div#result >p").should()
72              .not().beEmpty()
73              .contain(check);
74      }
75
76      @Override
77      protected String getDefaultBaseUrl() {
78          final String BASE_URL = "http://localhost:"+port+"/";
79          return BASE_URL;
80      }
81
82  }
```

Screenshot 2 — result.html:

```html
1  <!DOCTYPE html>
2  <html xmlns="http://www.w3.org/1999/xhtml"
3      xmlns:th="http://www.thymeleaf.org">
4  <head>
5  <meta charset="ISO-8859-1" />
6  <title>Result</title>
7  </head>
8  <body style="font-family: Comic Sans MS">
9      <h1 id="result-page-heder">Calculator</h1>
10     <div id="result">
11         <p th:text="'Result: '+${output}" th:if="${isError==null}">
12         <p th:text="'Error: '+${output}" th:if="${isError}" style="
13     </div>
14     <div id="home_link_container">
15         <a id="home_link" href="/">Home</a>
16     </div>
17 </body>
18 </html>
```

And this is the final thing… Let's wire up the service to the controller now re-run our feature tests (feature tests are integration tests and use the actual service instance)

File   Edit   Source   Refactor   Navigate   Search   Project   Run   Window   Help

Quick Access   Java EE   Debug

Markers   Properties   Servers   Data Source Explorer   Debug   Terminal   JUnit

Finished after 11.173 seconds

Runs: 22/22   Errors: 0   Failures: 0

Failure Trace

▲ org.self.learn.service.CalcServiceTest [Runner: JUnit 4] (6.061 s)
   itShouldAddTheNumbers (0.000 s)
   itShouldSubtractTheNumbers (0.000 s)
   itShouldDivideTheNumbers (0.000 s)
   itShouldMultiplyTheNumbers (6.060 s)
▲ org.self.learn.featuretest.HomePageTests [Runner: JUnit 4] (2.089 s)
   itShouldDisplayOperand2LabelAndInputOfTypeText (0.616 s)
   itShouldDisplayOperand1LabelAndInputOfTypeText (0.101 s)
   itShouldDisplayCalculateButtonInputOfTypeSubmit (0.053 s)
   itShouldDisplayHomePage (0.078 s)
   itShouldDisplayFormWithOperand1AndOperand2AndOperationAndCalculateButton
   itShouldDisplayAppropriateDropdownOptions (0.314 s)
   itShouldTransitionAwayFromHomePageUponClickingCalculateButton (0.814 s)
▲ org.self.learn.featuretest.ResultPageTests [Runner: JUnit 4] (2.612 s)
   itShouldDisplayMissingInputError (0.416 s)
   itShouldCaculateResultWithValidInput (0.425 s)
   itShouldDisplayInvalidInputError (0.413 s)
   itShouldDisplayZeroDenominatorError (1.357 s)
▲ org.self.learn.SimpleCalculatorWebApplicationTests [Runner: JUnit 4] (0.000 s)
   contextLoads (0.000 s)
▲ org.self.learn.controller.CalcWebAppControllerTests [Runner: JUnit 4] (0.315 s)
   itShouldReturnErrorMessageAndFlagFrorZeroDenominator (0.238 s)
   itShouldReturnErrorMessageAndFlagForMissingInput (0.004 s)
   itShouldReturnComputationResultByCallingCalcService (0.014 s)
   itShouldReturnErrorMessageAndFlagForInvalidInput (0.007 s)
   itShouldReturnHomeView (0.040 s)
   itShouldReturnResultView (0.012 s)