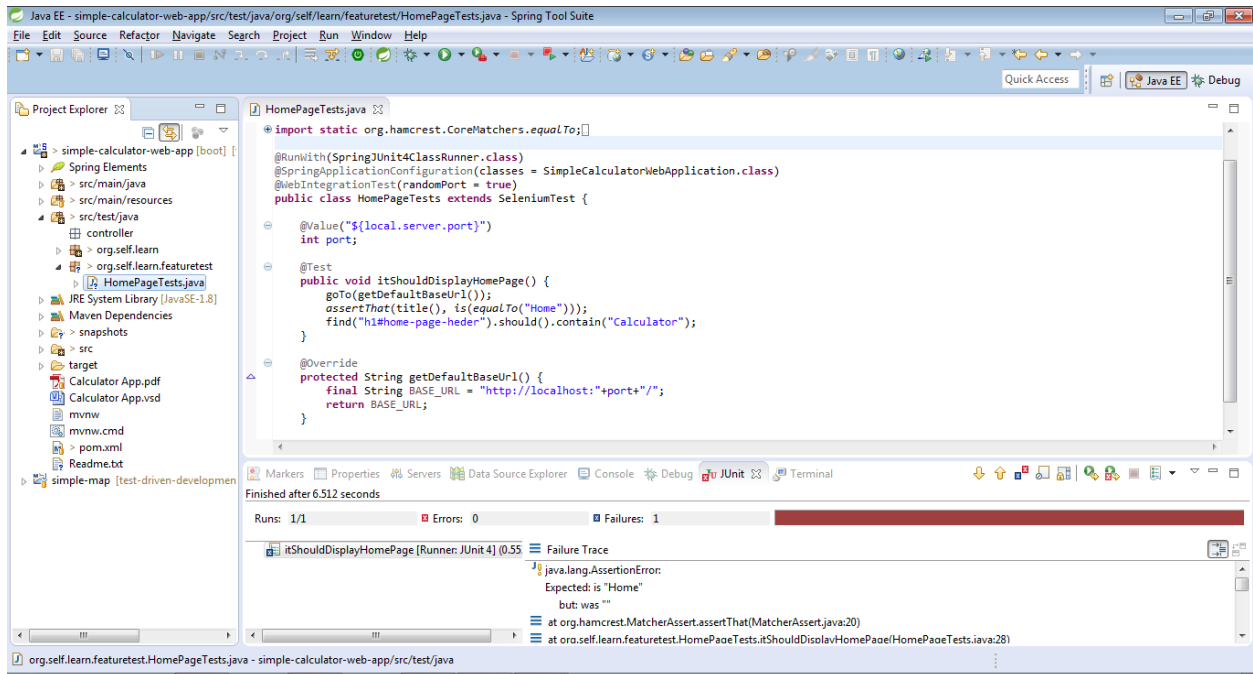
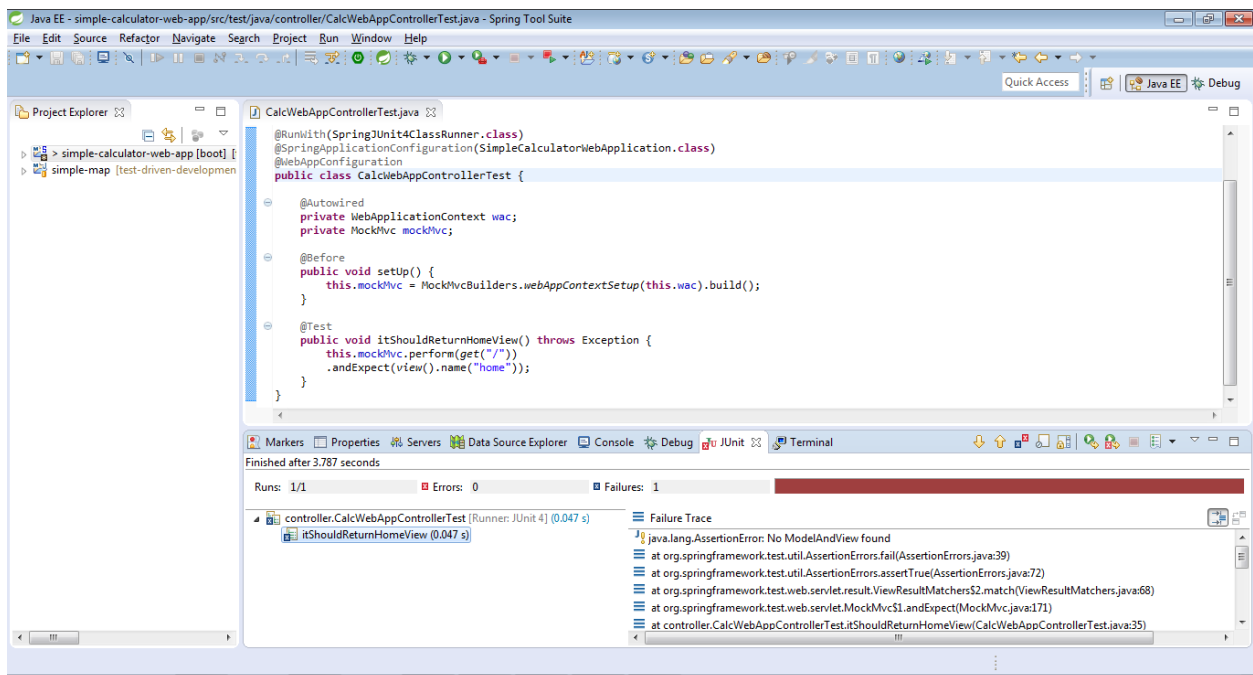


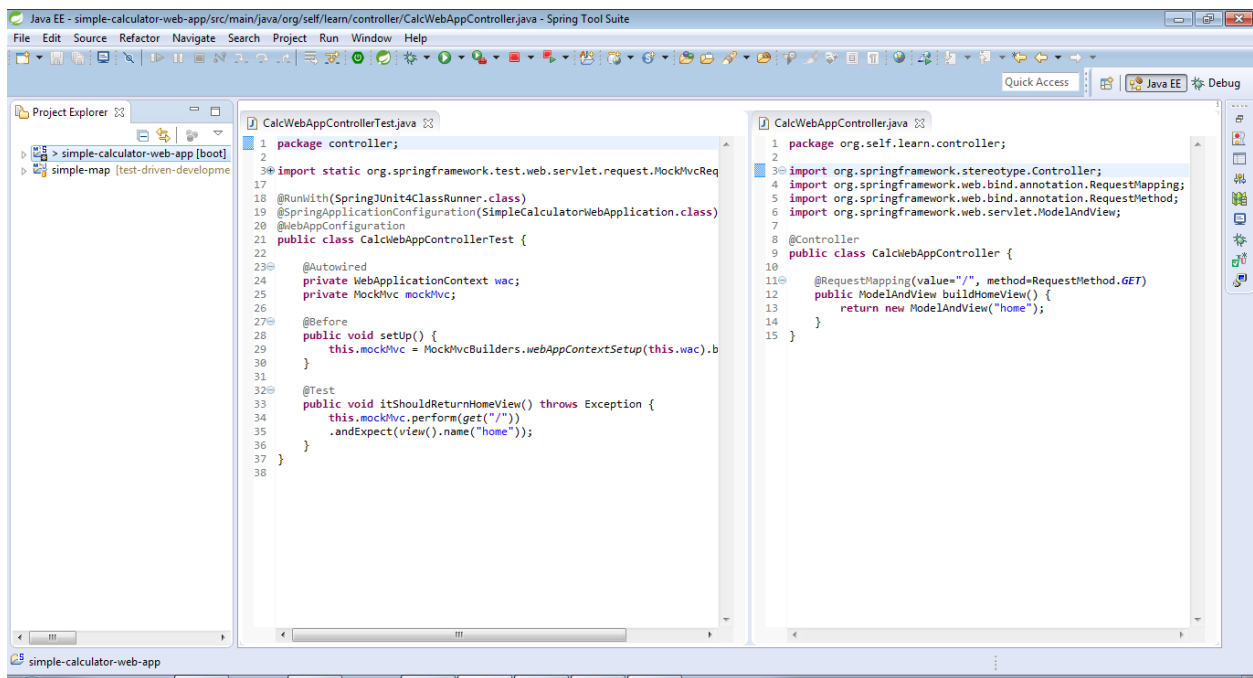
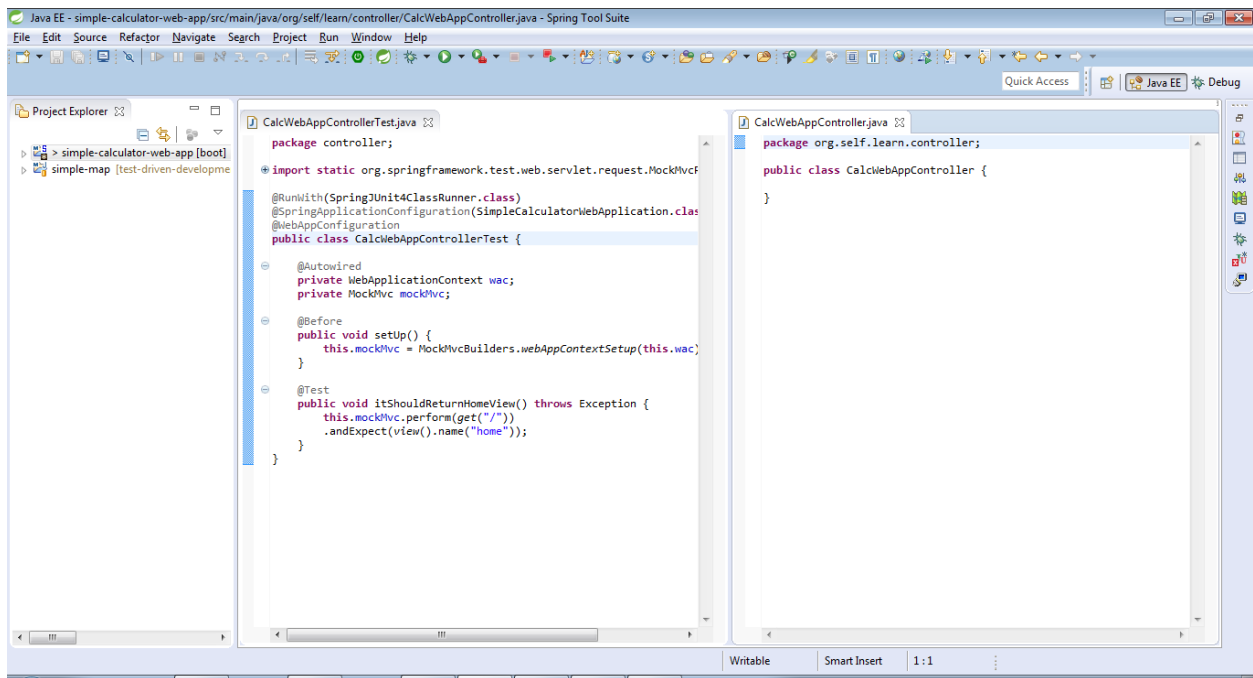
Let's start with a feature test... As expected, it fails!



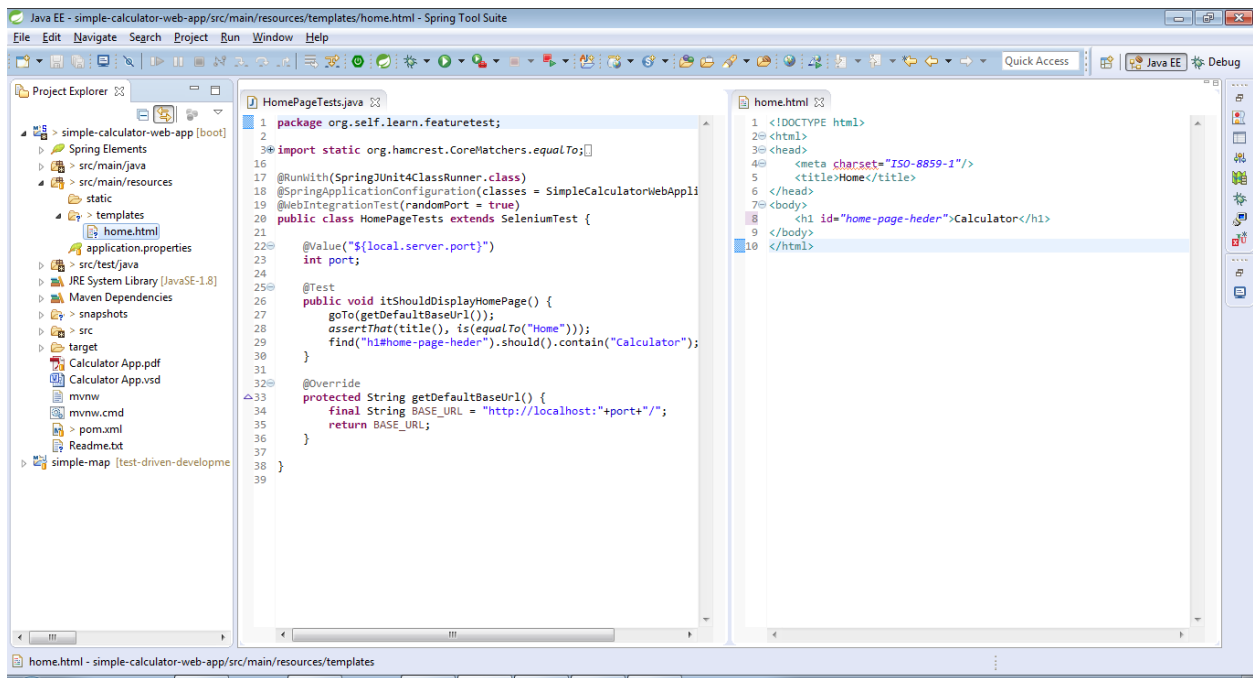
Let's hop on to our controller test which fails as well, as we expect it to.



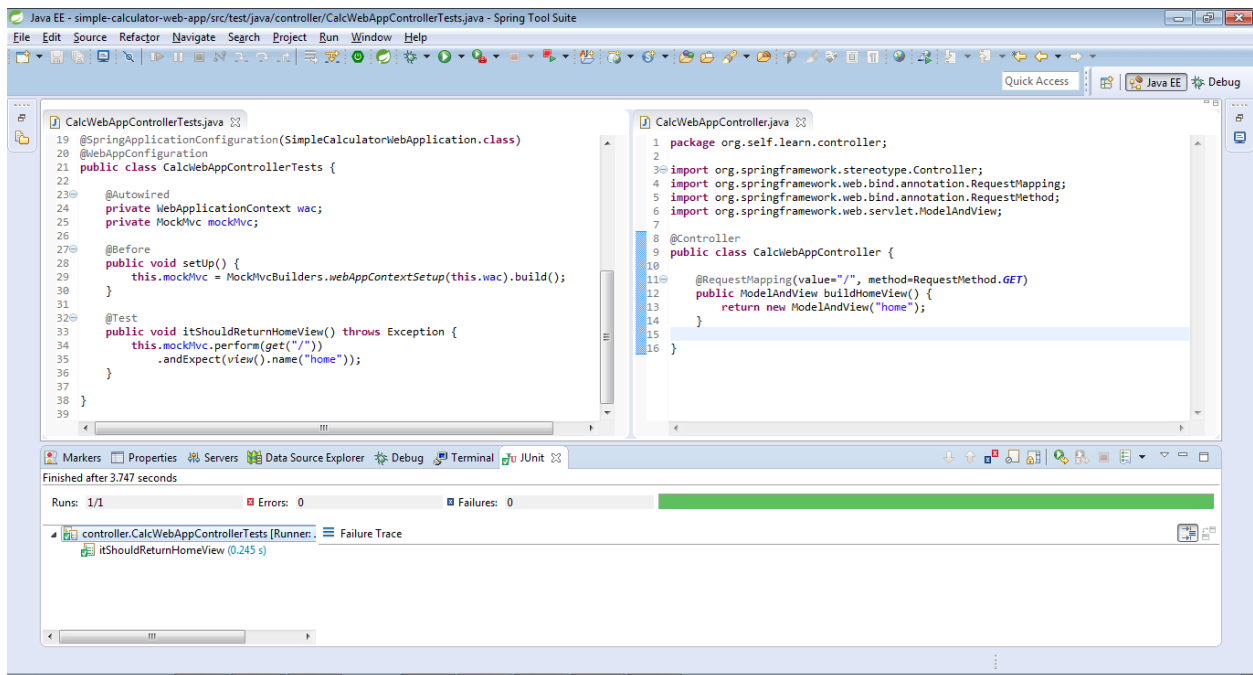
Let's put some implementation together for the controller...

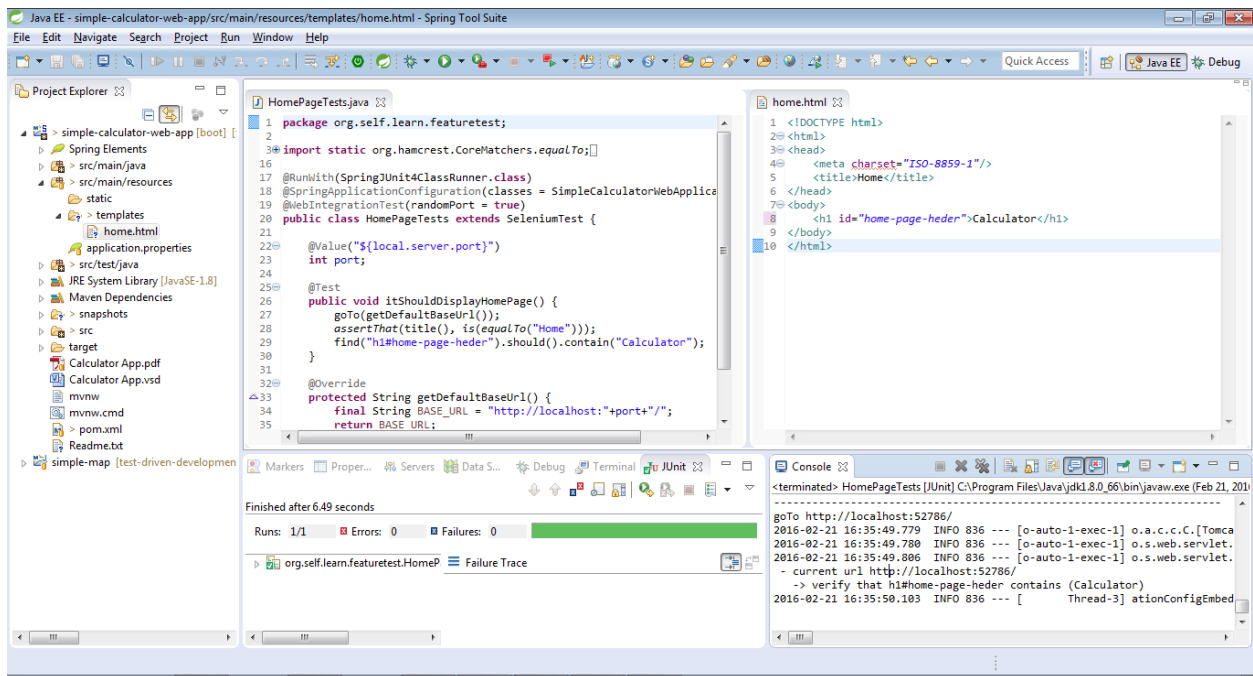


Let's run the test again... It still fails since we have not created a view ("home") yet...

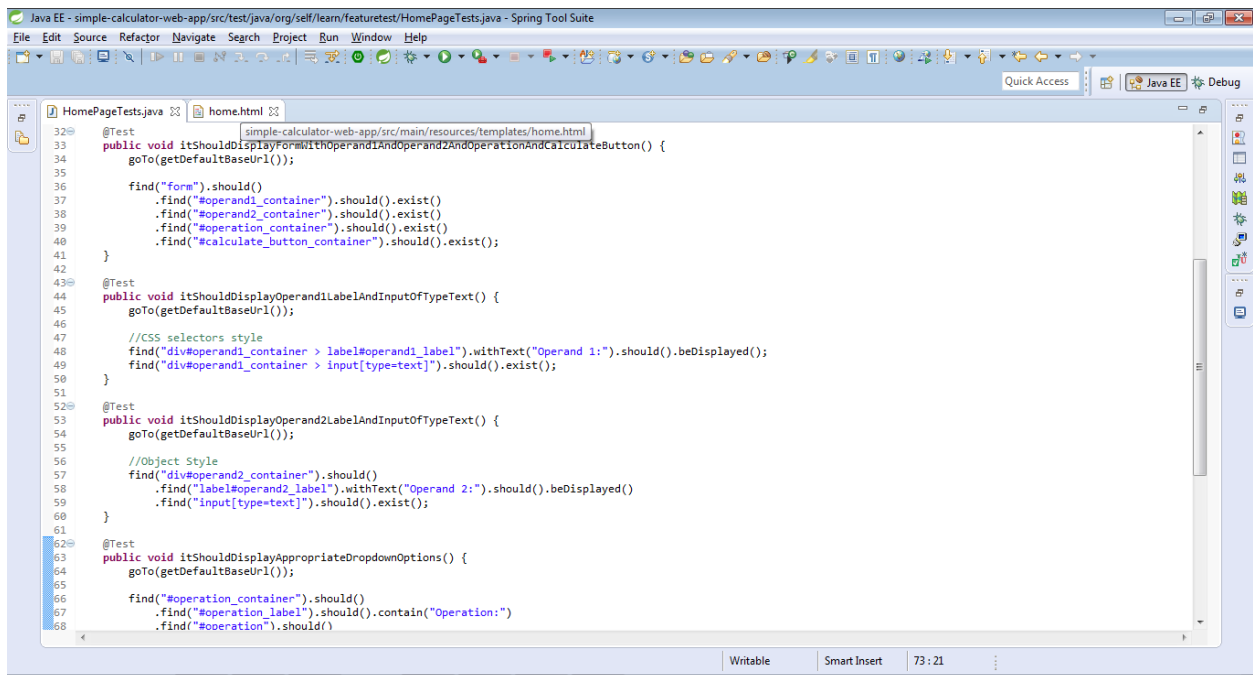


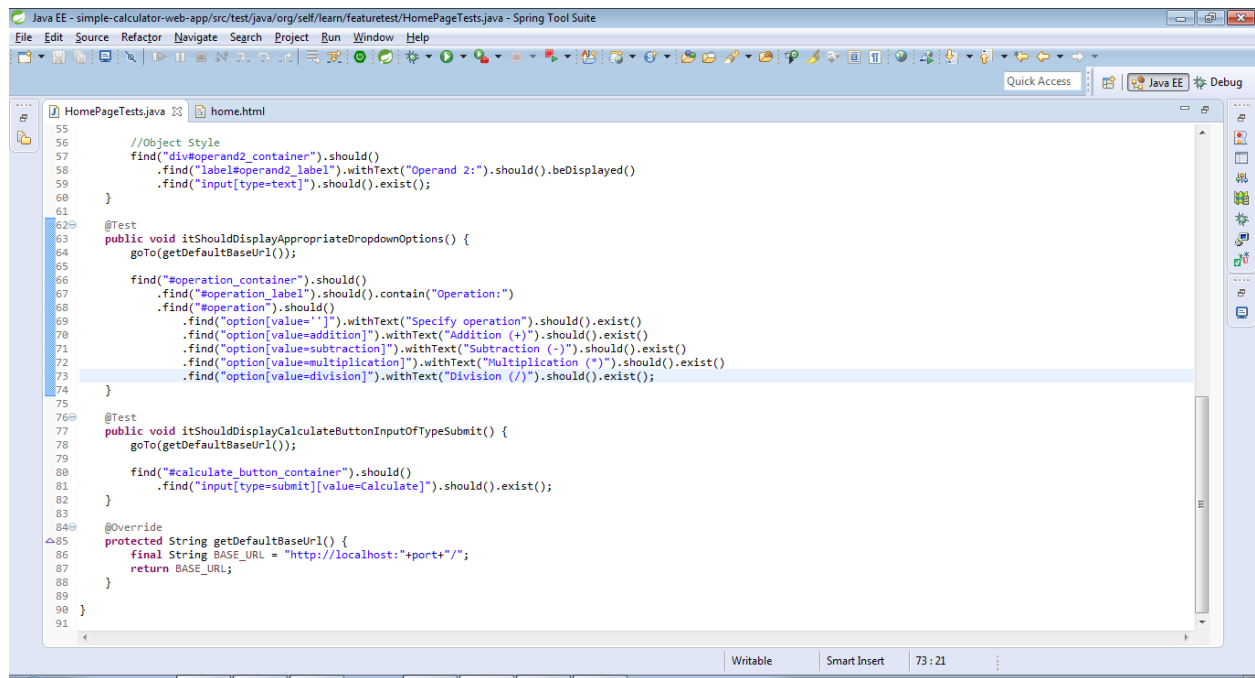
The tests did complete successfully this time.





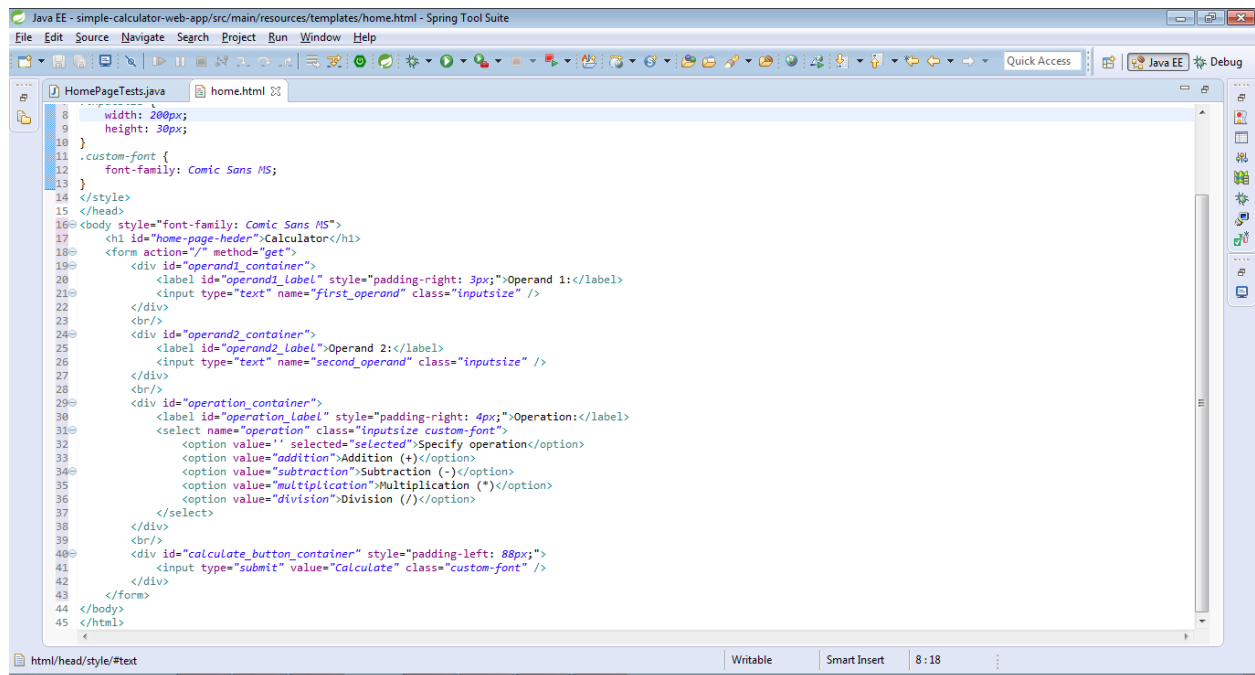
We will now write a few other tests and corresponding implementations (one after another of course) as per wireframe. I am providing screenshot with the final set of tests and implementation for reference...



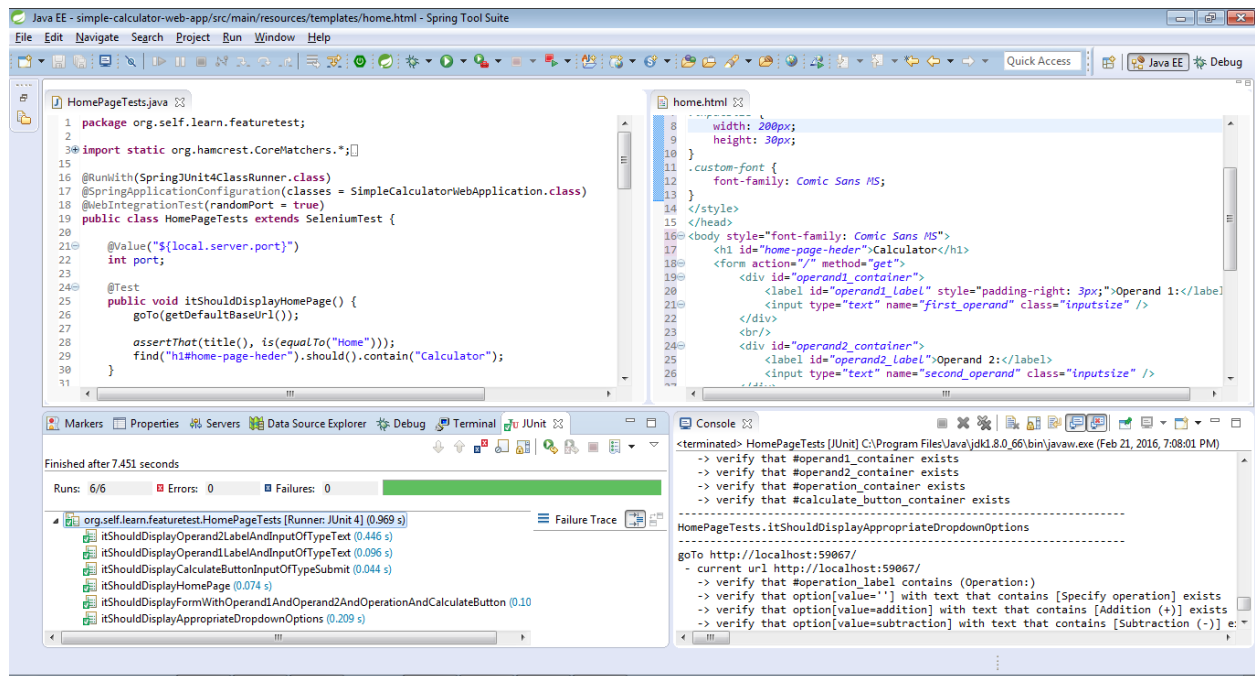


```
55 //Object Style
56 find("#operand2_container").should()
57   .find("#operand2_label").withText("Operand 2:").should().beDisplayed()
58   .find("input[type=text]").should().exist();
59 }
60
61
62 @Test
63 public void itShouldDisplayAppropriateDropdownOptions() {
64     goTo(getDefaultBaseUrl());
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
```

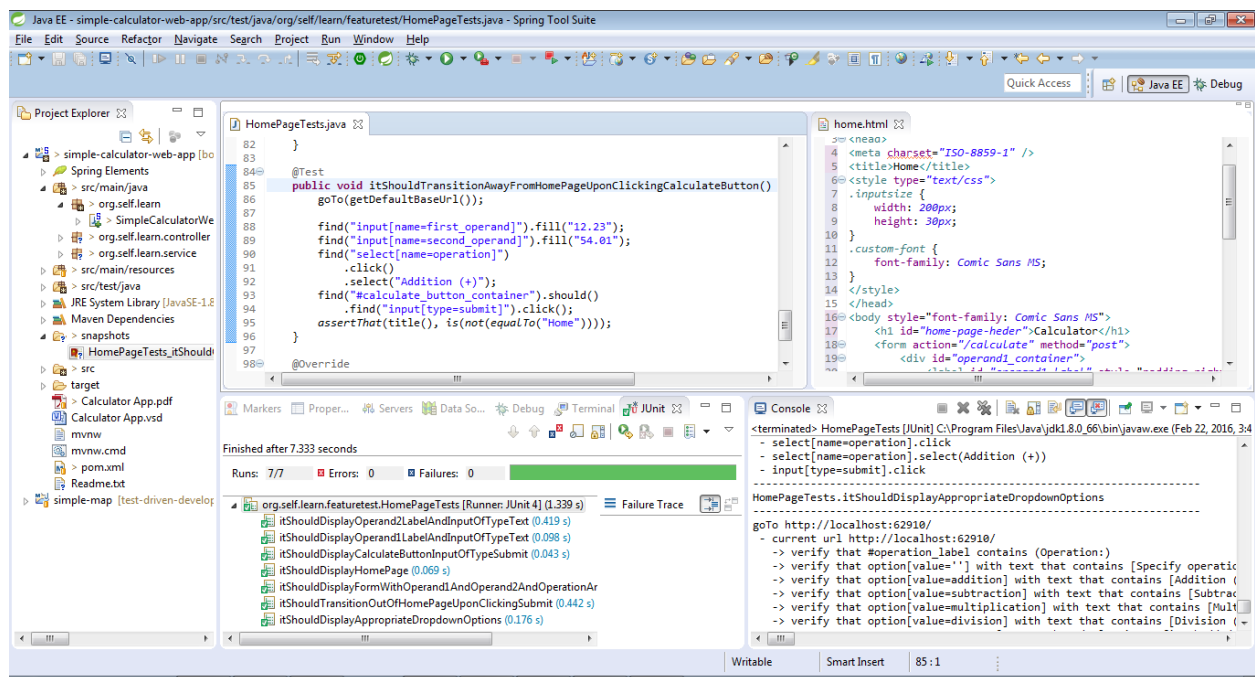
Note that we are only concerned about displaying the home page up to this point



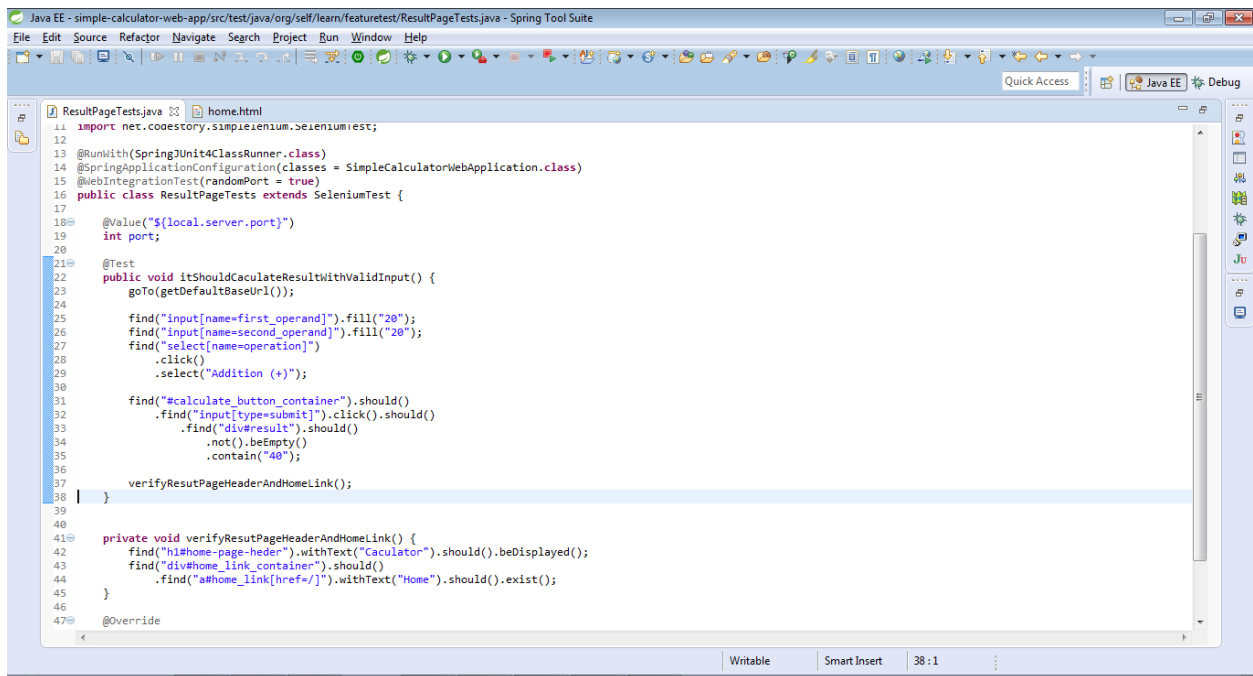
```
8 width: 200px;
9 height: 30px;
10 }
11 .custom-font {
12     font-family: Comic Sans MS;
13 }
14 </style>
15 </head>
16 <body style="font-family: Comic Sans MS">
17     <h1 id="home-page-header">Calculator</h1>
18     <form action="/" method="get">
19         <div id="operand1_container">
20             <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" style="padding-right: 3px;">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>
```



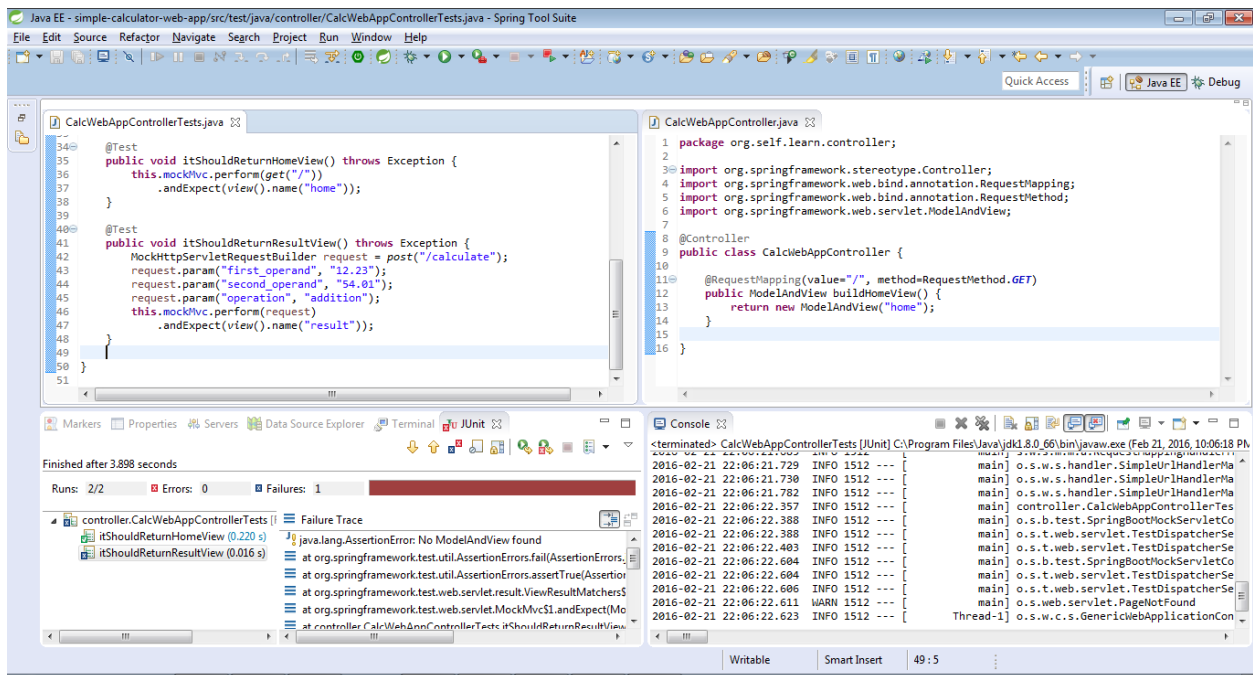
I also wrote a test case to verify transition away from home page upon clicking Calculate button



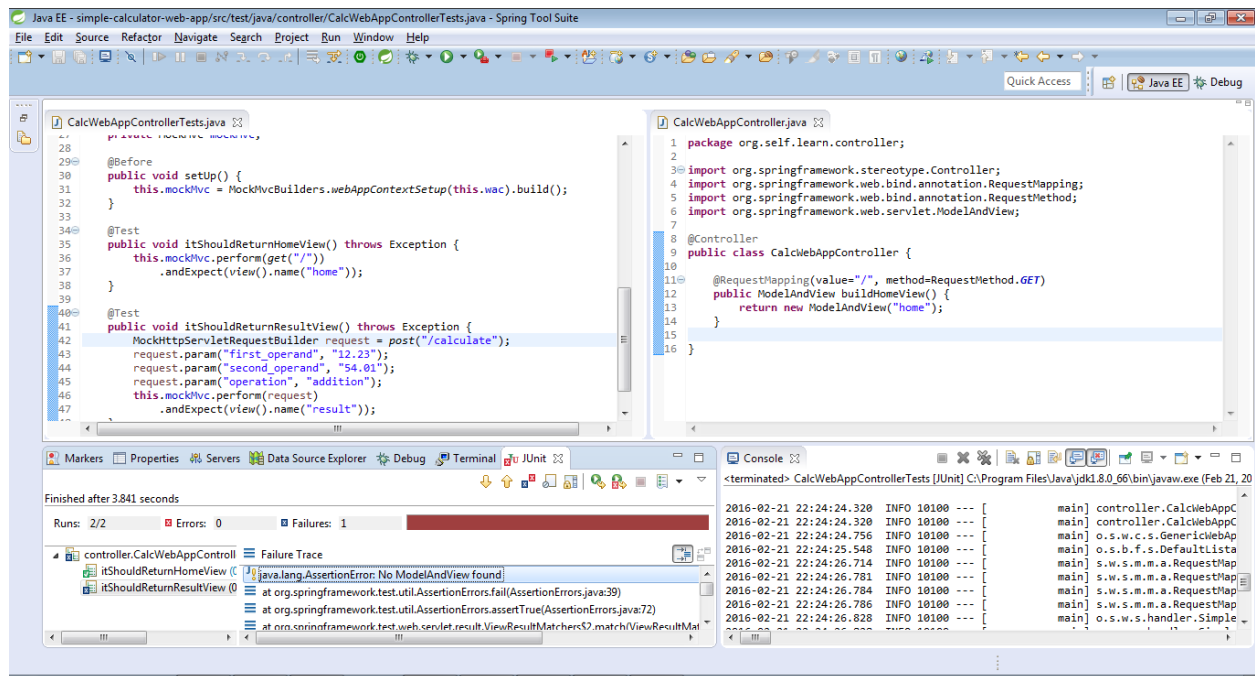
Now let's write a few test cases for the result page. When run, we expect the tests to fail and so they do...



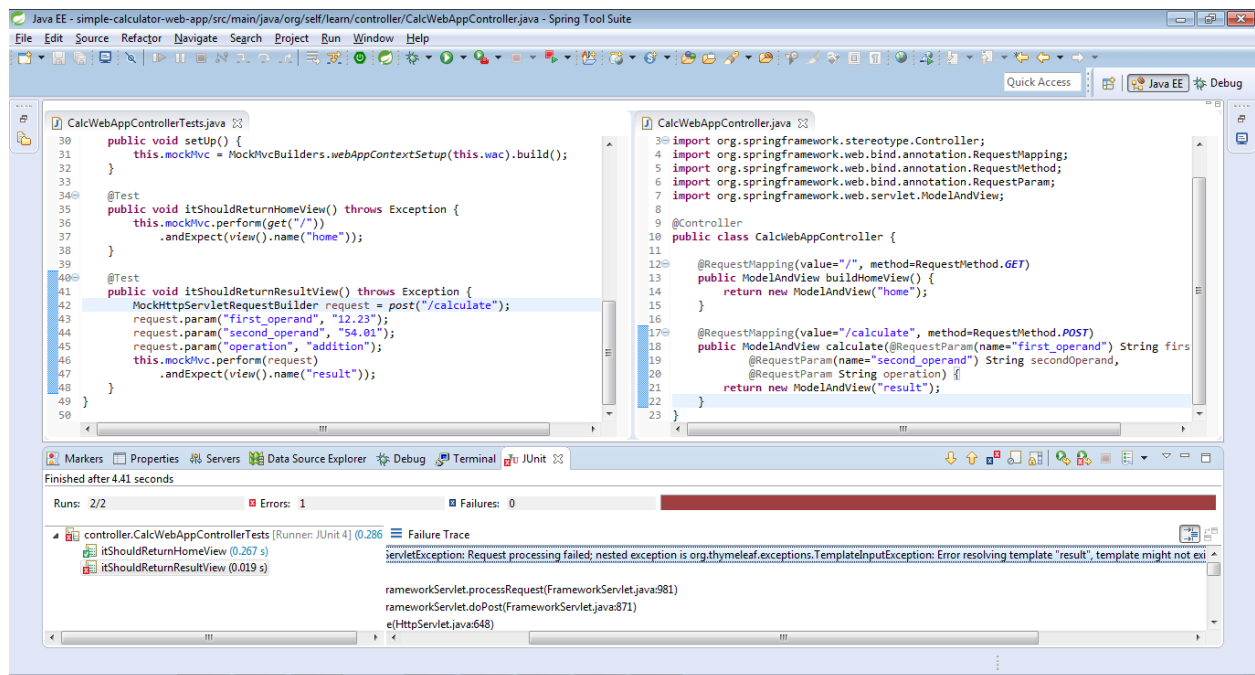
We will require adding a controller end point and corresponding view to take care of this...



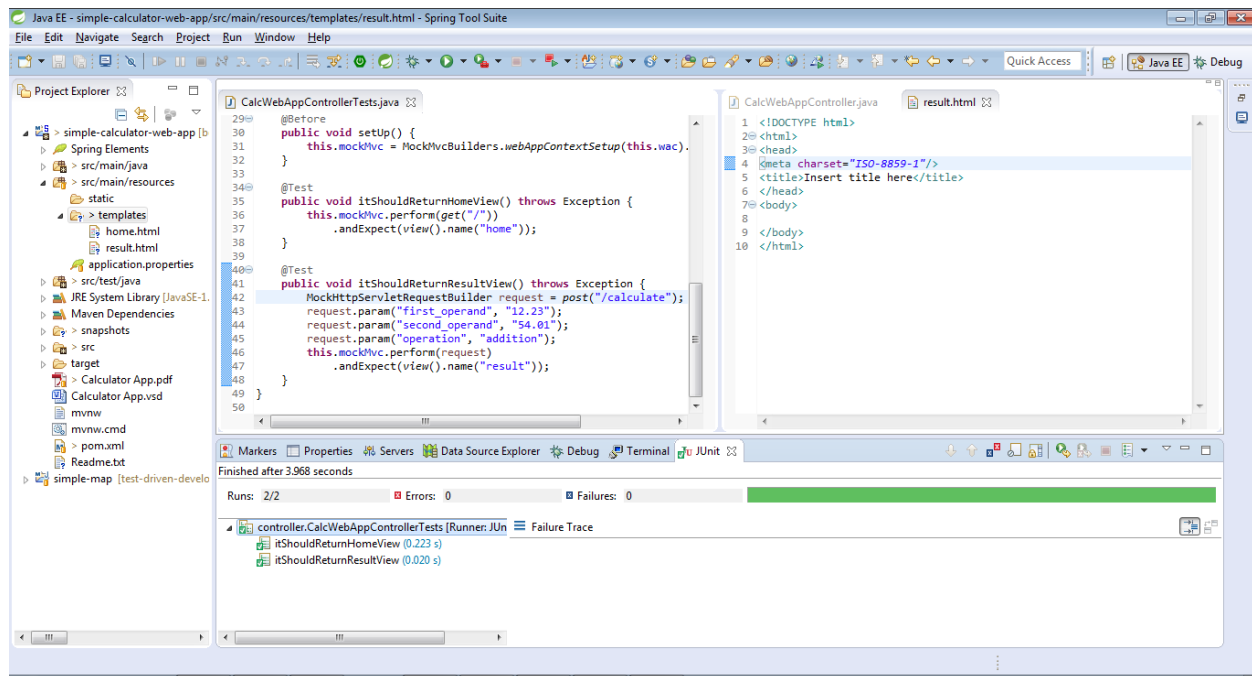
Here is the test case I have put together... which fails as expected. Let's implement the endpoint `/calculate`



Although the endpoint exists now, the test still fails because the view “result” does not exist yet. Let’s go create that.



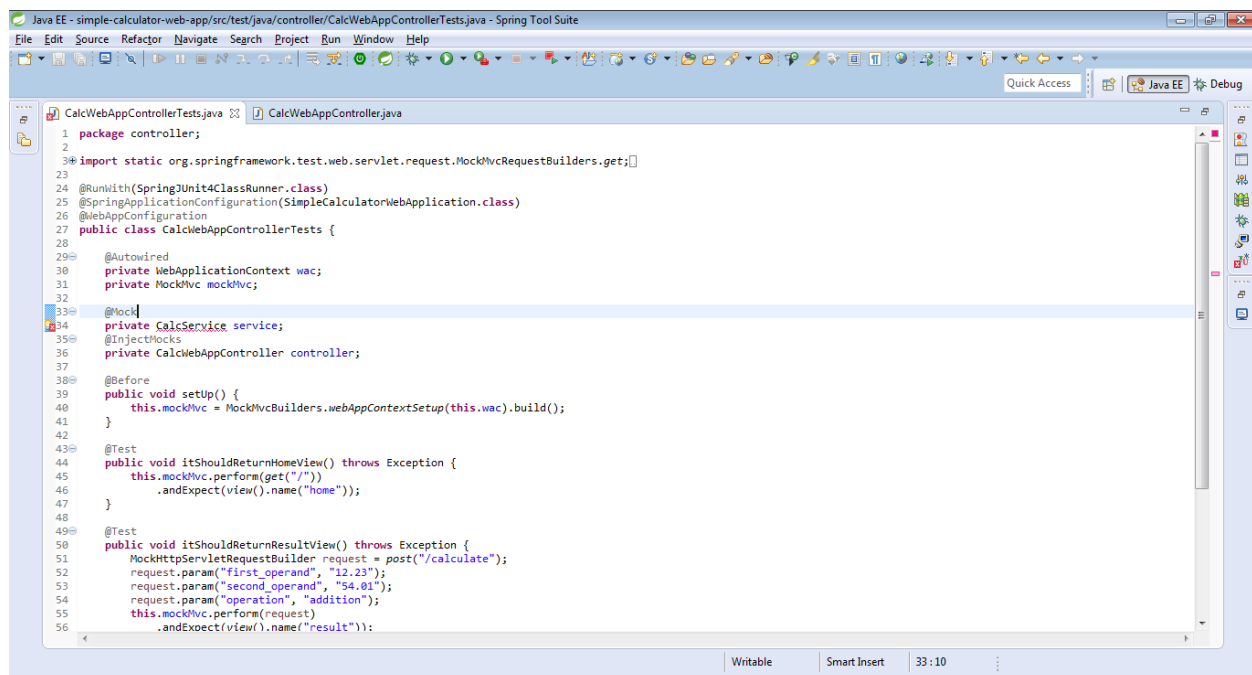
I just wrote an empty HTML and re-ran the test. The test did complete successfully this time...



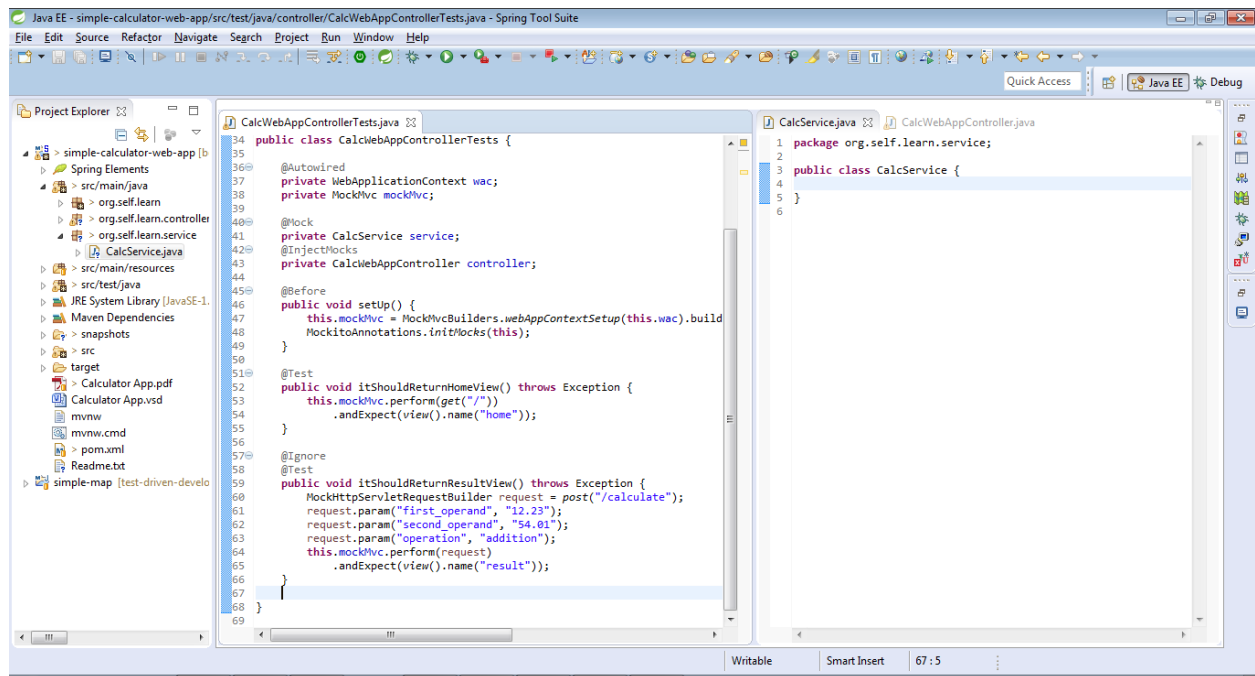
Before I go and implement the HTML, let's try to create the model containing the actual result. Our "result" view will have nothing to display otherwise.

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

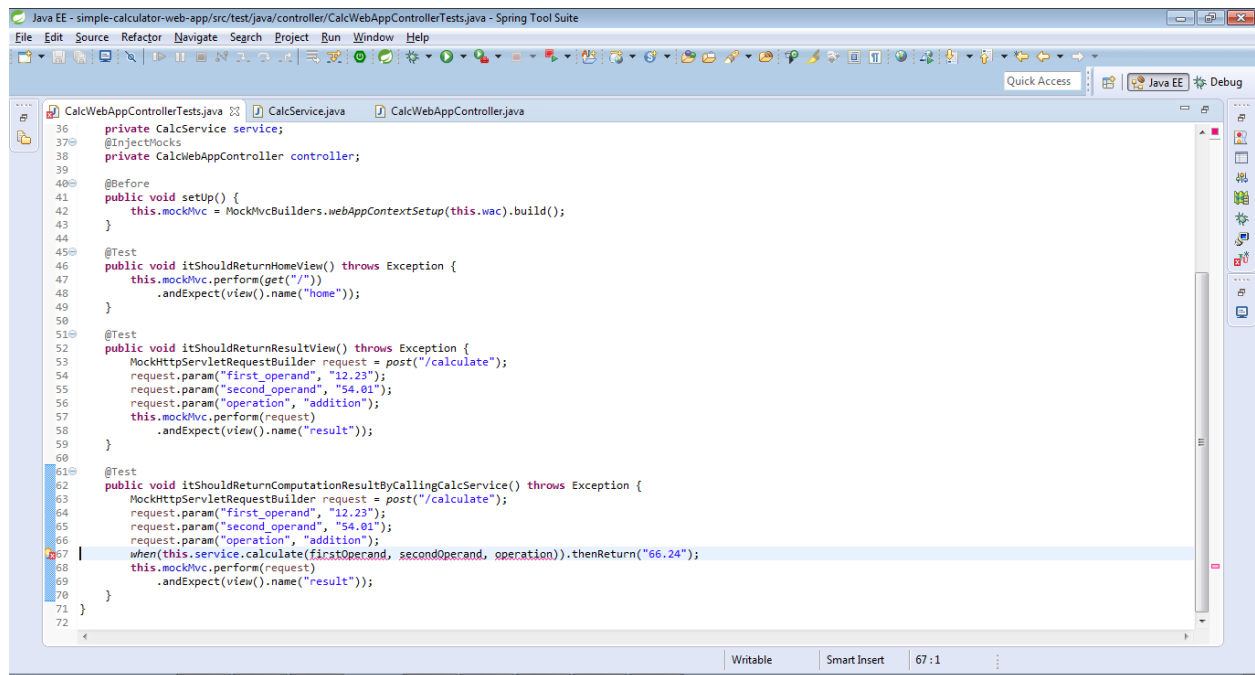
Note that I intend to create a mock of the service class and inject the same into controller...



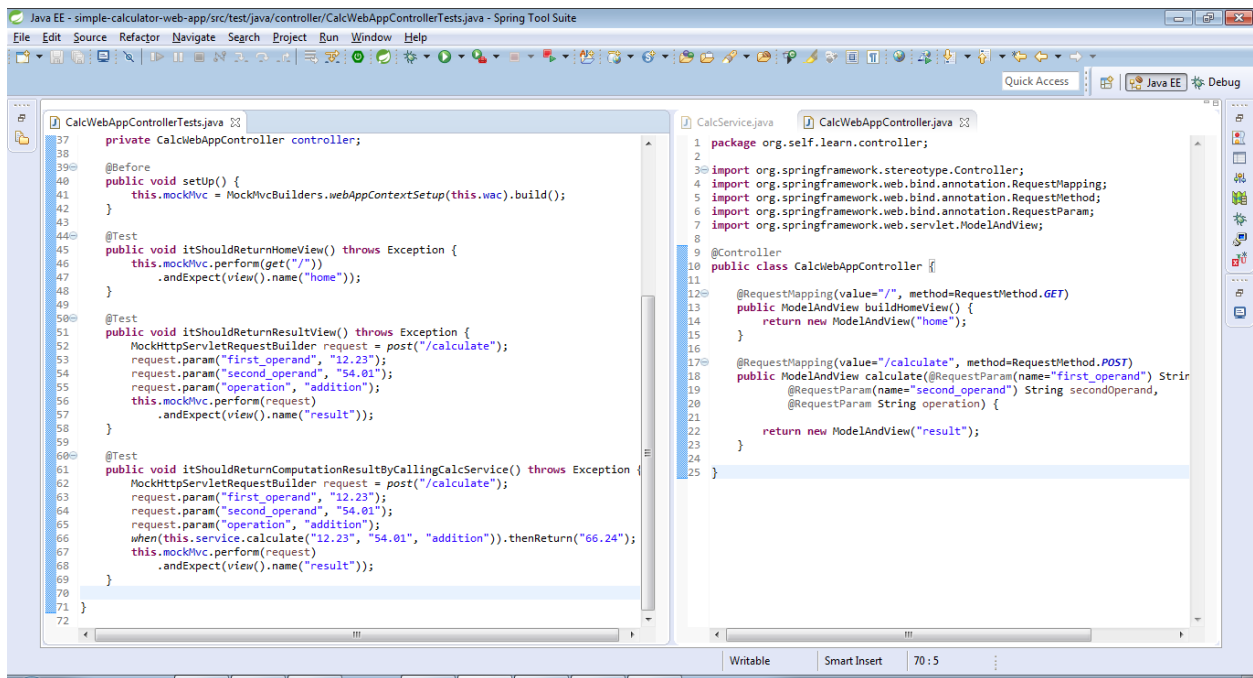
Let's create the service class skeleton to avoid compilation error



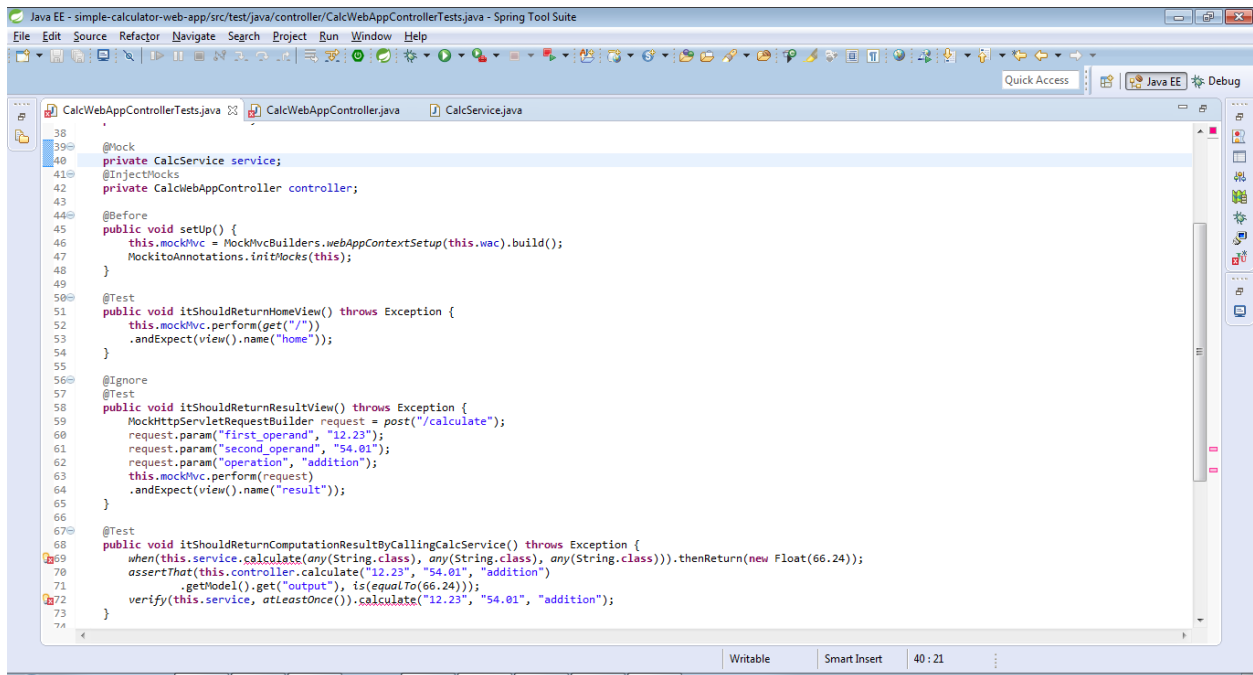
Next, I want to write unit test cases for the controller by stubbing out service method using Mockito...



I created a blank method in the service to address compilation issue.

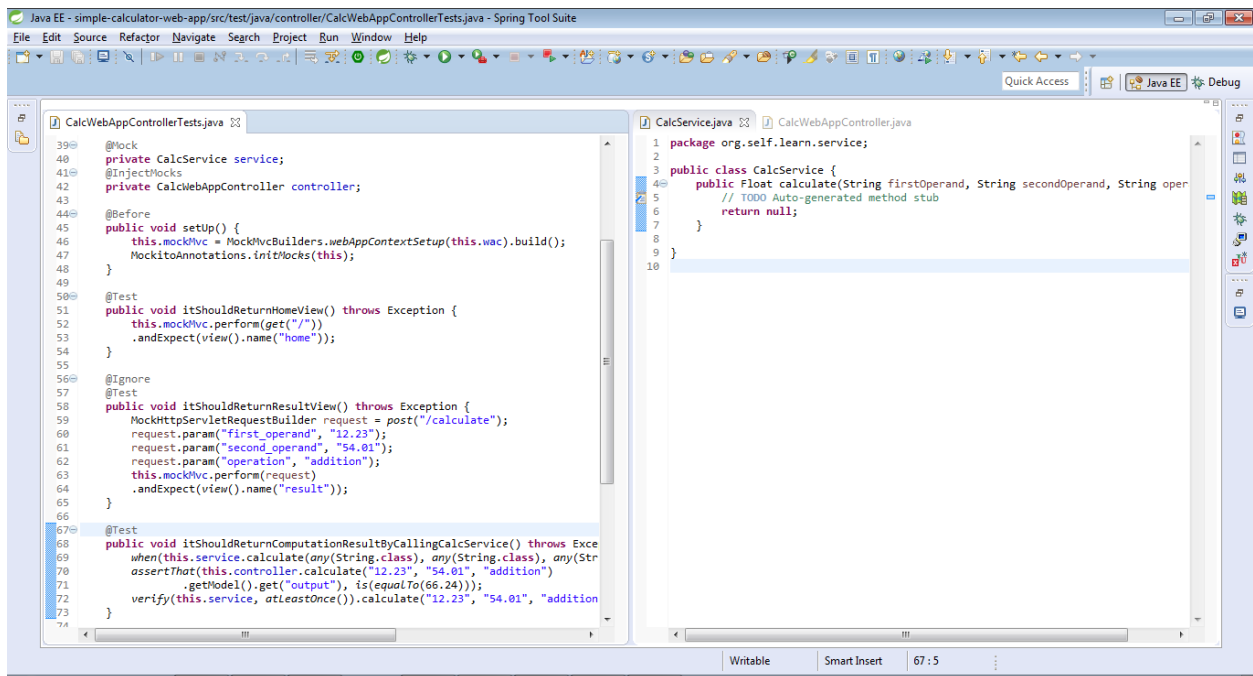


While we will create the calculate() method on the service class momentarily, let's make note of the test.

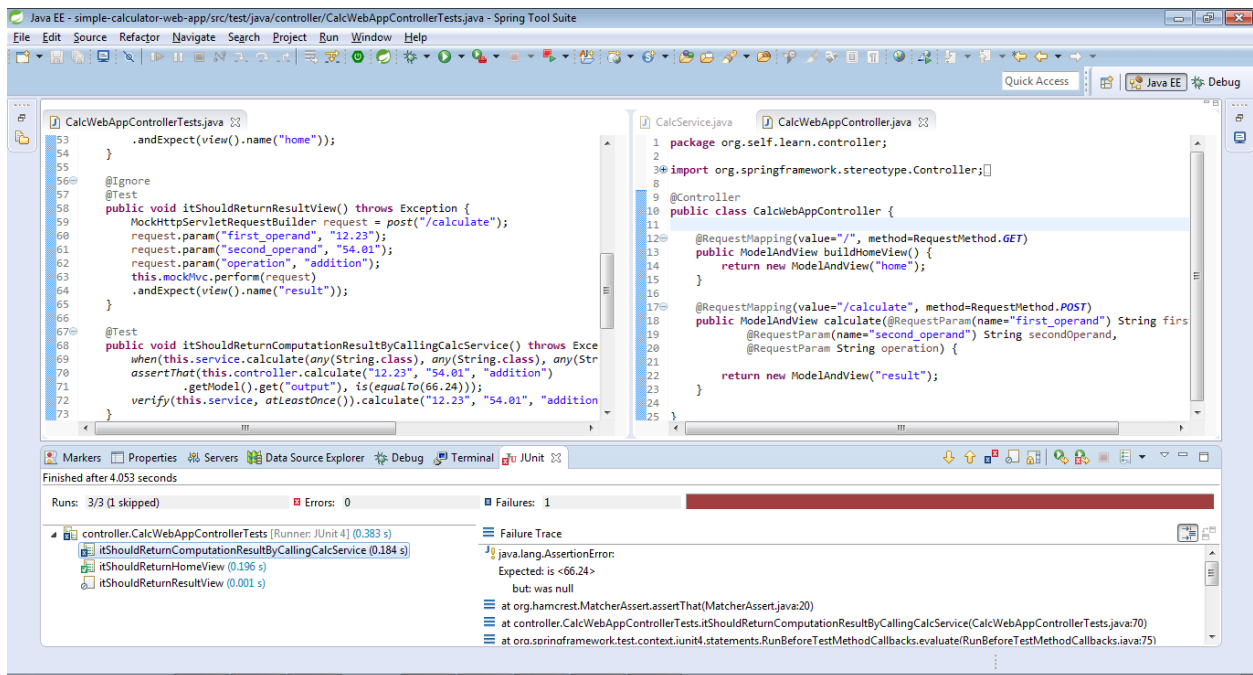


Compilation issues resolved after adding the auto generated method stub to the service class

N.B: I have ignored controller tests in the class for the time being



The test fails when ran for known reason...



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:

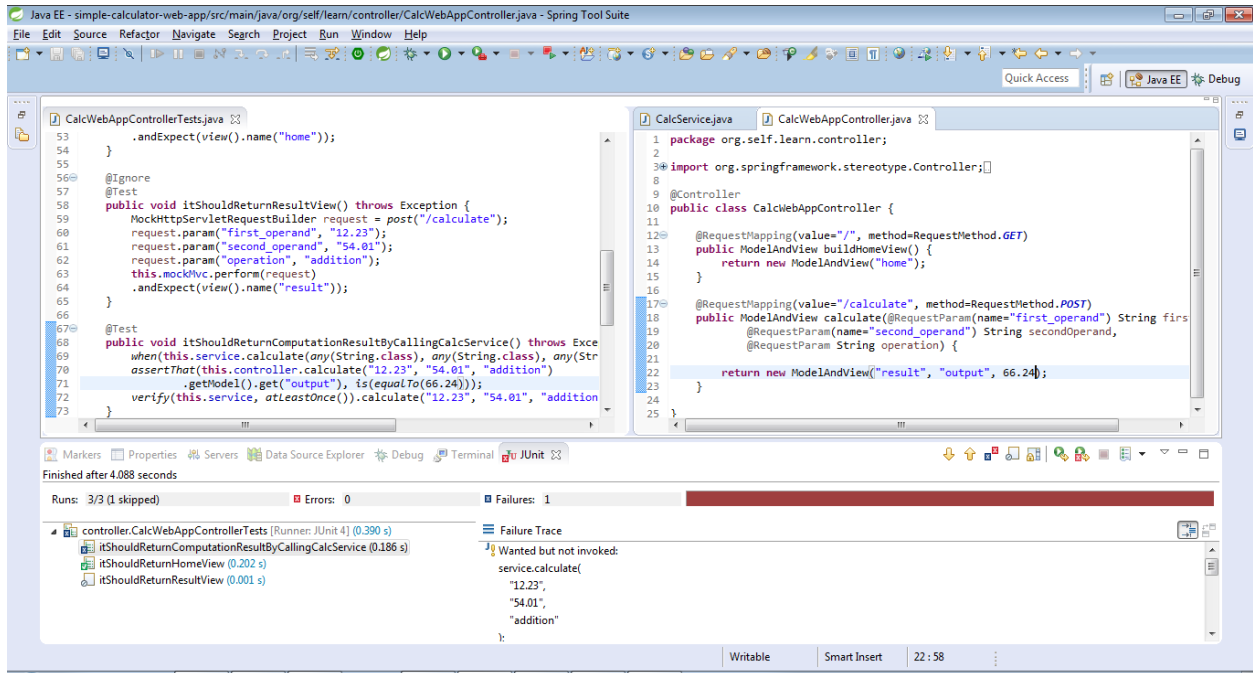
Wanted but not invoked:

```

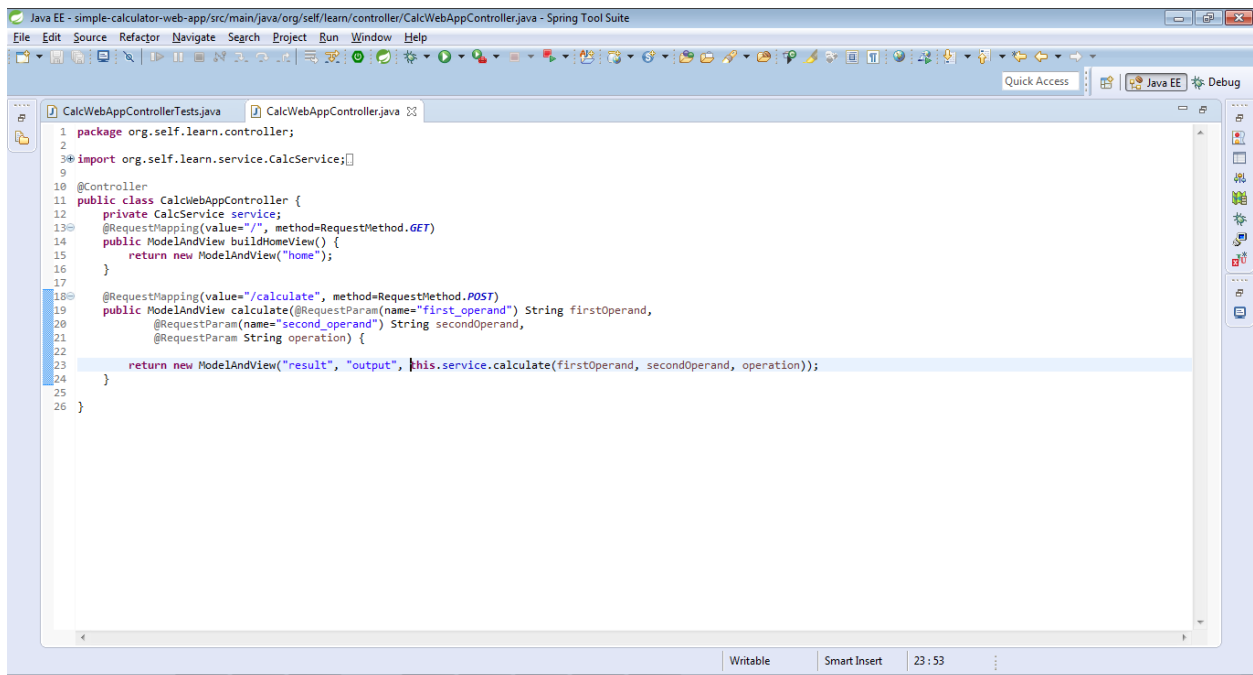
service.calculate(
    "12.23",
    "54.01",
    "addition"
)

```

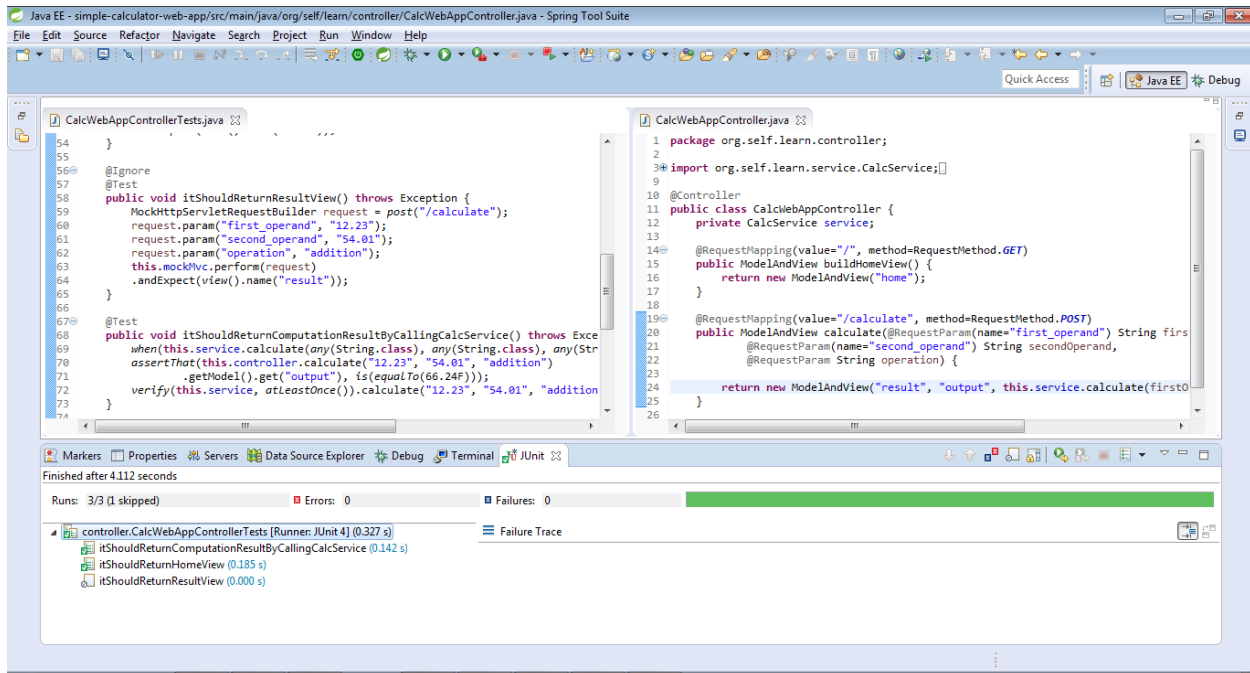
```
);  
-> at  
controller.CalcWebAppControllerTests.itShouldReturnComputationResultByCallingCalcService(CalcWebAppControllerTests.java:70)  
Actually, there were zero interactions with this mock.
```



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.

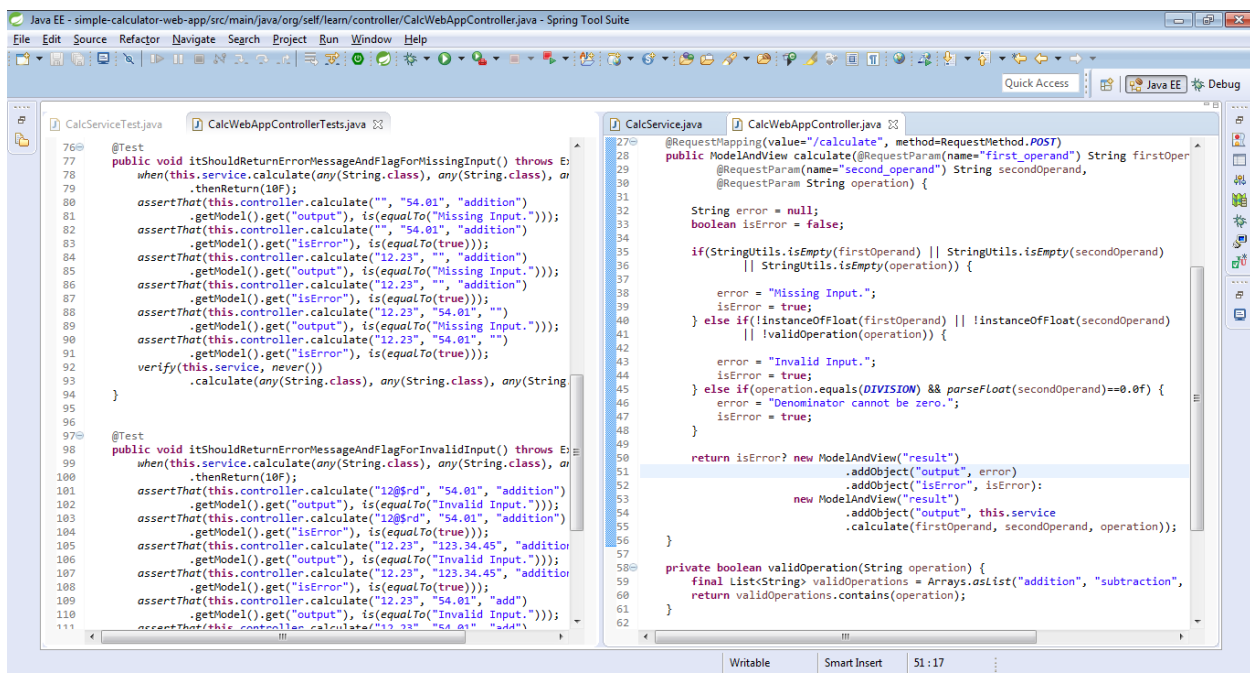


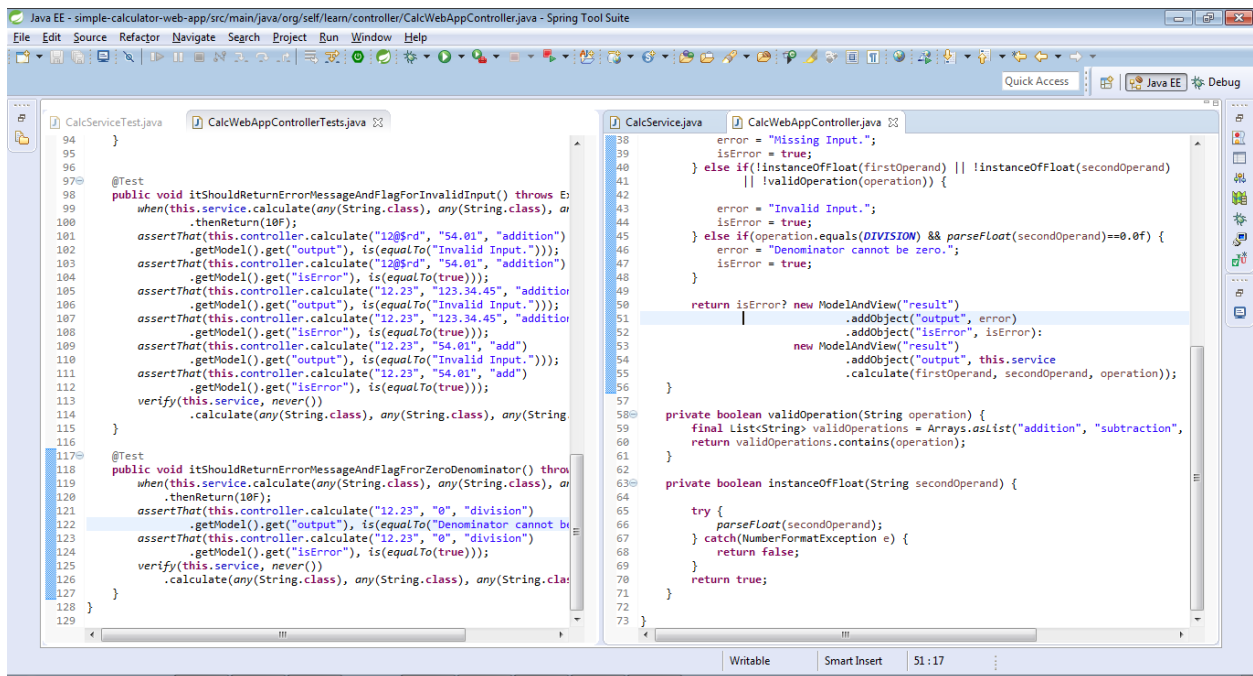
Let's re-run the test after ignoring the Controller test for `/calculate` (which won't run just yet because the actual service instance is neither ready nor wired)



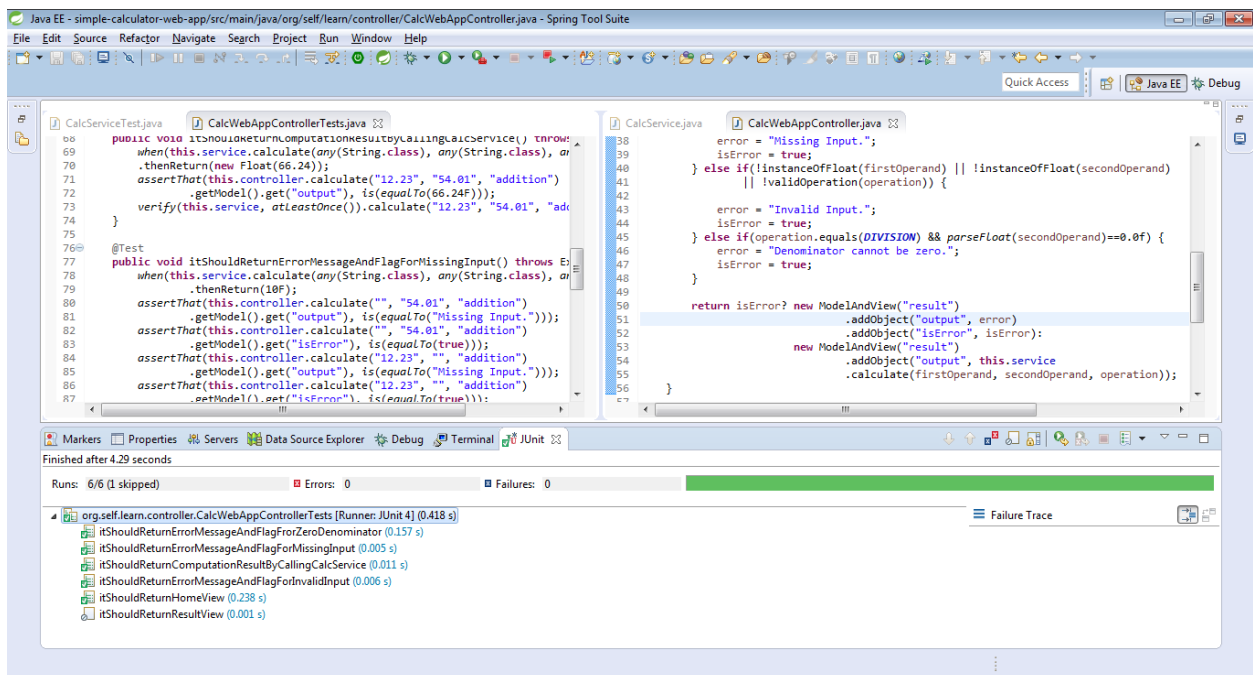
Note that we are able to unit test controller regardless service implementation availability...

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





The test results are positive for all tests but the controller test for “/calculate”...



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.

The screenshot shows an IDE with two open files: `CalcServiceTest.java` and `CalcService.java`. The `CalcServiceTest.java` file contains unit tests for the `CalcService` class, including tests for addition, subtraction, multiplication, and division. The `CalcService.java` file contains the implementation of the `CalcService` class, which provides a `calculate` method that takes two operands and an operation string, and returns the result as a `Float`.

```
CalcServiceTest.java
3 import static org.hamcrest.CoreMatchers.*;
11 @RunWith(BlockJUnit4ClassRunner.class)
12 public class CalcServiceTest {
13     private CalcService service;
14
15     @Before
16     public void setUp() {
17         this.service = new CalcService();
18     }
19
20     @Test
21     public void itShouldAddTheNumbers() throws Exception {
22         assertThat(this.service.calculate("12.34", "54.01", "addition"),
23             is(equalTo(66.35F)));
24     }
25
26     @Test
27     public void itShouldSubtractTheNumbers() throws Exception {
28         assertThat(this.service.calculate("40", "20", "subtraction"),
29             is(equalTo(20F)));
30     }
31
32     @Test
33     public void itShouldMultiplyTheNumbers() throws Exception {
34         assertThat(this.service.calculate("2", "3.5", "multiplication"),
35             is(equalTo(7F)));
36     }
37
38     @Test
39     public void itShouldDivideTheNumbers() throws Exception {
40         assertThat(this.service.calculate("20", "10", "division"),
41             is(equalTo(2F)));
42     }
43 }

CalcService.java
1 package org.self.learn.service;
2
3 import static java.lang.Float.parseFloat;
4
5 public class CalcService {
6     private static final String ADDITION = "addition";
7     private static final String SUBTRACTION = "subtraction";
8     private static final String MULTIPLICATION = "multiplication";
9     private static final String DIVISION = "division";
10
11     public Float calculate(String firstOperand, String secondOperand, String operation) {
12         Float result = null;
13
14         switch(operation) {
15             case ADDITION:
16                 result = parseFloat(firstOperand) + parseFloat(secondOperand);
17                 break;
18             case SUBTRACTION:
19                 result = parseFloat(firstOperand) - parseFloat(secondOperand);
20                 break;
21             case MULTIPLICATION:
22                 result = parseFloat(firstOperand) * parseFloat(secondOperand);
23                 break;
24             case DIVISION:
25                 result = parseFloat(firstOperand) / parseFloat(secondOperand);
26                 break;
27             default:
28                 break;
29         }
30         return result;
31     }
32 }
33
34 }
```

The screenshot shows the same IDE with the unit tests for `CalcService` being executed. The `CalcServiceTest.java` file is open, and the `CalcService.java` file is also open. The IDE shows the test results in the bottom panel, indicating that all tests passed successfully.

```
CalcServiceTest.java
3 import static org.hamcrest.CoreMatchers.*;
11 @RunWith(BlockJUnit4ClassRunner.class)
12 public class CalcServiceTest {
13     private CalcService service;
14
15     @Before
16     public void setUp() {
17         this.service = new CalcService();
18     }
19
20     @Test
21     public void itShouldAddTheNumbers() throws Exception {
22         assertThat(this.service.calculate("12.34", "54.01", "addition"),
23             is(equalTo(66.35F)));
24     }
25
26     @Test
27     public void itShouldSubtractTheNumbers() throws Exception {
28         assertThat(this.service.calculate("40", "20", "subtraction"),
29             is(equalTo(20F)));
30     }
31
32     @Test
33     public void itShouldMultiplyTheNumbers() throws Exception {
34         assertThat(this.service.calculate("2", "3.5", "multiplication"),
35             is(equalTo(7F)));
36     }
37
38     @Test
39     public void itShouldDivideTheNumbers() throws Exception {
40         assertThat(this.service.calculate("20", "10", "division"),
41             is(equalTo(2F)));
42     }
43 }

CalcService.java
1 package org.self.learn.service;
2
3 import static java.lang.Float.parseFloat;
4
5 public class CalcService {
6     private static final String ADDITION = "addition";
7     private static final String SUBTRACTION = "subtraction";
8     private static final String MULTIPLICATION = "multiplication";
9     private static final String DIVISION = "division";
10
11     public Float calculate(String firstOperand, String secondOperand, String operation) {
12         Float result = null;
13
14         switch(operation) {
15             case ADDITION:
16                 result = parseFloat(firstOperand) + parseFloat(secondOperand);
17                 break;
18             case SUBTRACTION:
19                 result = parseFloat(firstOperand) - parseFloat(secondOperand);
20                 break;
21             case MULTIPLICATION:
22                 result = parseFloat(firstOperand) * parseFloat(secondOperand);
23                 break;
24             case DIVISION:
25                 result = parseFloat(firstOperand) / parseFloat(secondOperand);
26                 break;
27             default:
28                 break;
29         }
30         return result;
31     }
32 }
33
34 }
```

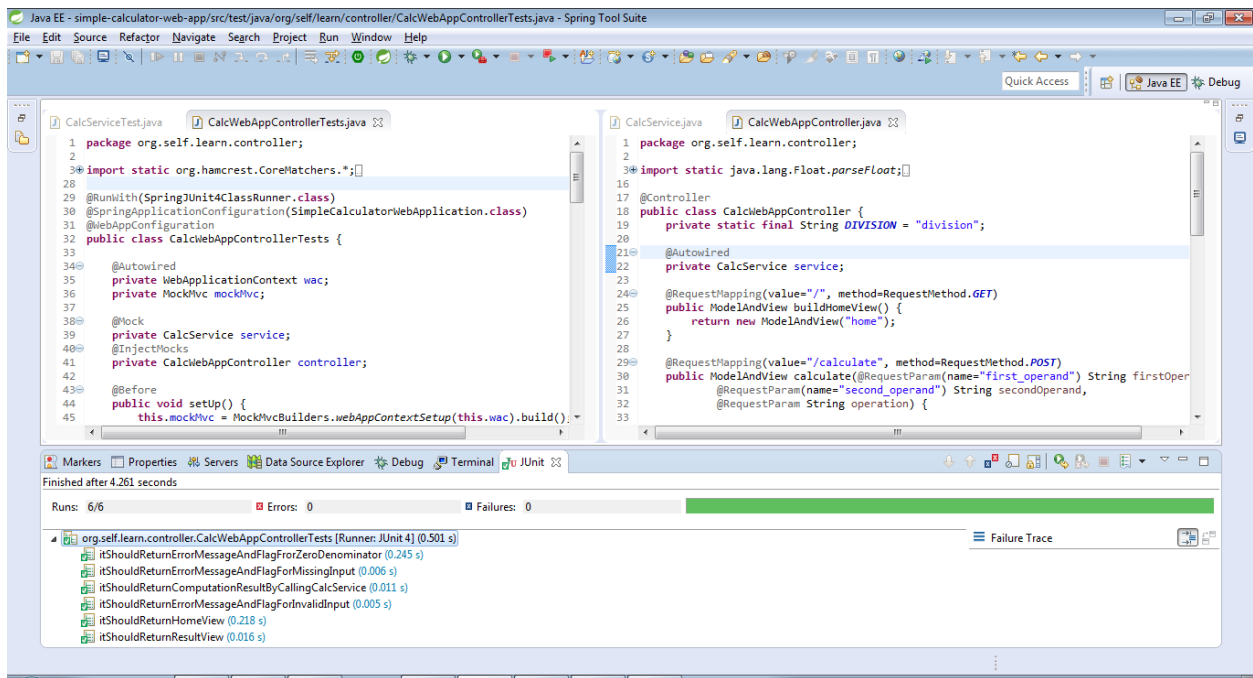
Finished after 0.027 seconds

Runs: 4/4 Errors: 0 Failures: 0

org.self.learn.service.CalcServiceTest [Runner: JUnit 4] (0.003 s)

- itShouldAddTheNumbers (0.002 s)
- itShouldSubtractTheNumbers (0.000 s)
- itShouldDivideTheNumbers (0.000 s)
- itShouldMultiplyTheNumbers (0.001 s)

Let's wire up the service to the controller now and run the controller test for the `/calculate` end point



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

