Lab 1. First Unit Test

# Marco Antonio Torres Escárcega A01568181 16 de febrero 2021

# Setup

1. Make sure you have Java 13 (JDK) installed in your computer

<https://www.oracle.com/technetwork/java/javase/downloads/jdk13-downloads-5672538.html>

1. Download Spring Tools 4 for Eclipse from <https://spring.io/tools>
2. Install the Spring Tools Suite 4
3. Change the VM path in the SpringToolsSuite4.ini file to the path of your Java installation

<https://wiki.eclipse.org/Eclipse.ini>

1. Start Spring Tool Suite 4
2. Go to **File -> New -> Spring Starter Project** to create a new Spring Project
3. Type the following information in the wizard:

**Service URL:** <https://start.spring.io>

**Name:** firstUnitTest

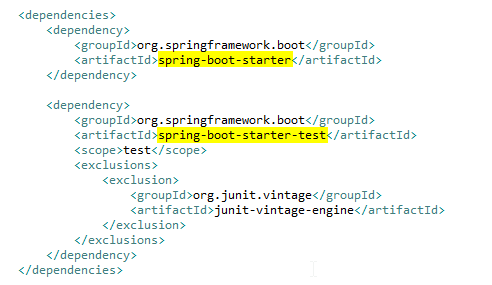
**Group:** mx.tec.lab

**Artifact**: firstUnitTest

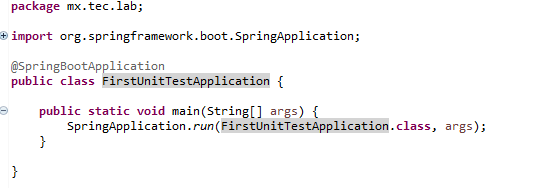
**Description:** First Unit Test Lab

**Package:** mx.tec.lab

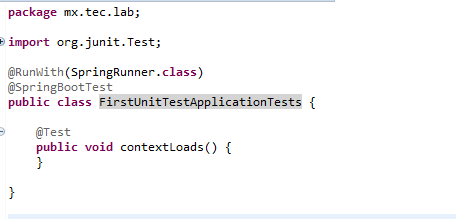
1. Select Spring Boot Version 2.2.4 and click Finish
2. Open pom.xml file and verify you have the following dependencies



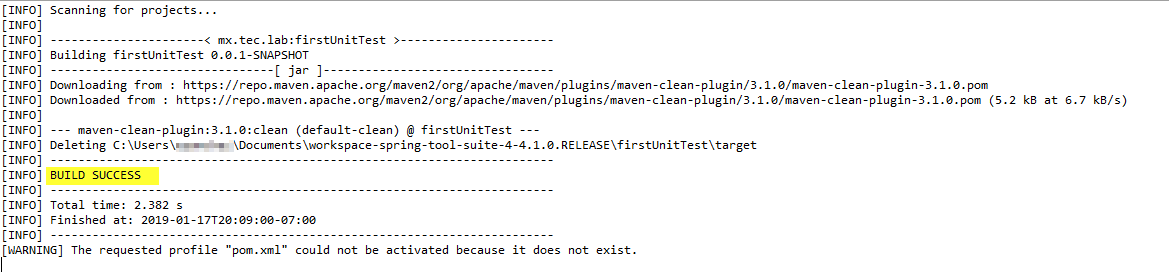
1. Verify that you have the **FirstUnitTestApplication** class in src/main/java (Package Explorer)



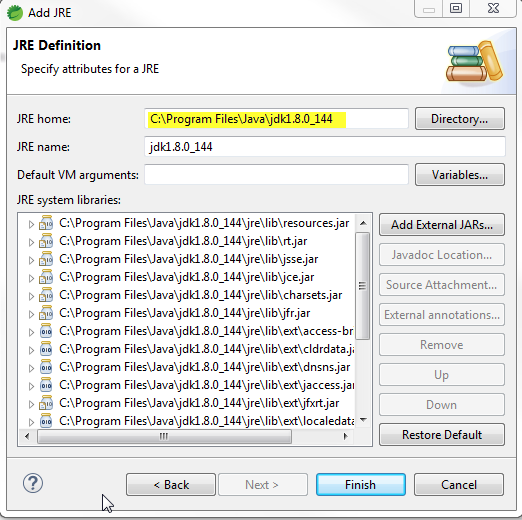
1. Verify that you have the **FirstUnitTestApplicationTests** class in src/test/java (Package Explorer)



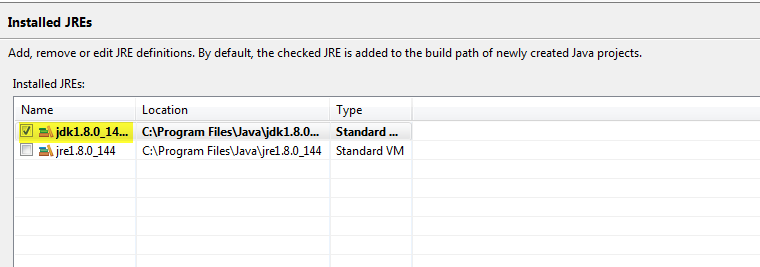
1. Right click on the **firstUnitTest** project in the Package Explorer
2. Click on **Run As -> Maven clean**
3. Verify that clean was successful in the Console



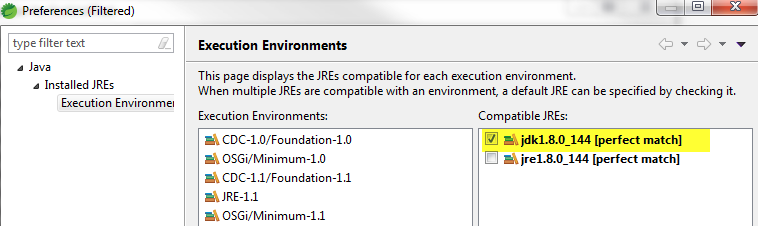
1. Click on **Run As -> Maven install**
   1. If you get this error: **No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK? (Use the JDK versioned installed instead of 1.8.0\_144)**
   2. Go to **Window -> Preferences -> Java -> Installed JREs**
   3. Click on **Add… -> Directory…** and select the folder where the JDK is installed and click Finish



* 1. Select the new JDK and click on **Apply and Close**



* 1. Right click on firstUnitTest project and click **Properties -> Java Build Path -> Libraries**
  2. Select **JRE System Library** and Click on **Edit…**
  3. Make sure **Execution environment** radio button is selected and **JavaSE-1.8(jre…)** is selected in the dropdown
  4. Click on environments, select **JavaSE-1.8**, and check **jdk1.8.0…** option. Finally click on **Apply and Close**



* 1. Click **Finish** on **Edit Library** overlay ad click on **Apply and Close** in **Properties for firstUnitTest** overlay
  2. Click on **Run As -> Maven install** again

1. Verify that install was successful in the Console



# Exercise

1. In Class FirstUnitTestApplicationTests add a LOG variable

**private** **static** **final** Log ***LOG*** = LogFactory.*getLog*(FirstUnitTestApplicationTests.**class**);

1. Add 2 testing methods

@Test

**public** **void** simpleTest() {

***LOG***.info("simple test");

}

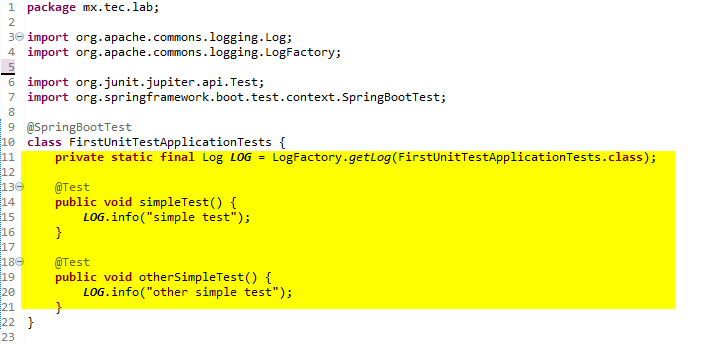
@Test

**public** **void** otherSimpleTest() {

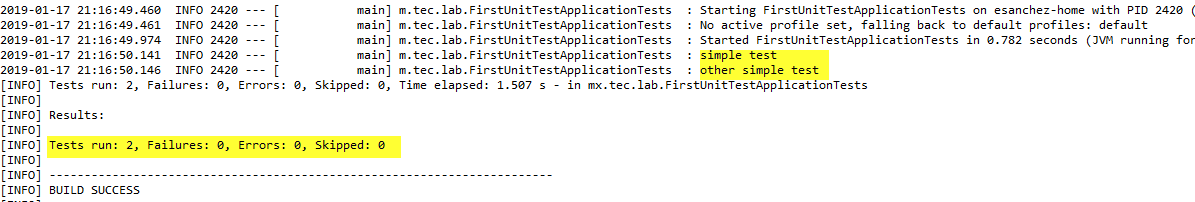
***LOG***.info("other simple test");

}

1. The code should look like this:



1. Right click on the project and execute **Run As -> Maven Test**
2. Verify that tests are executed correctly



1. Add setup, init, teardown and done methods

@BeforeAll

**public** **static** **void** setup() {

***LOG***.info("@BeforeAll - executes once before all test methods in this class");

}

@BeforeEach

**public** **void** init() {

***LOG***.info("@BeforeEach - executes before each test method in this class");

}

@AfterEach

**public** **void** tearDown() {

***LOG***.info("@AfterEach - executed after each test method.");

}

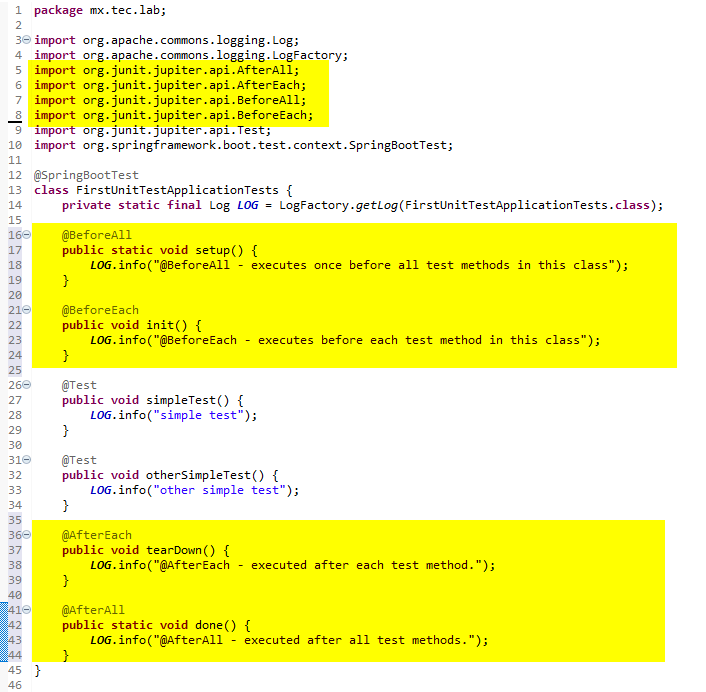
@AfterAll

**public** **static** **void** done() {

***LOG***.info("@AfterAll - executed after all test methods.");

}

1. The code should look like this:



1. Right click on the project and execute **Run As -> Maven Test**
2. Verify that tests are executed correctly in the console
3. Add the notImplementedTest method

@Test

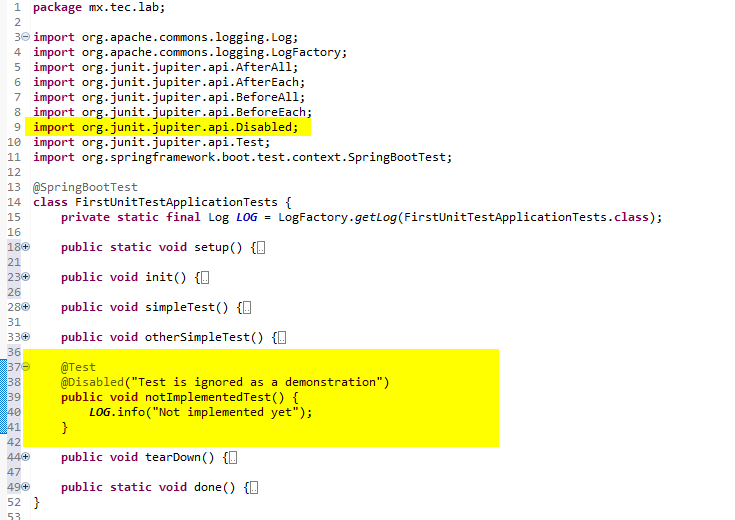
@Disabled("Test is ignored as a demonstration")

**public** **void** notImplementedTest() {

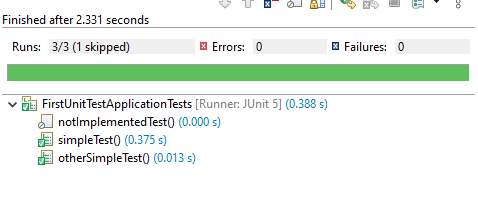
***LOG***.info("Not implemented yet");

}

1. The code should look like this:



1. Right click on the project and execute **Run As -> Maven Test**
2. Verify that tests are executed correctly in the console
3. Right click on the project and execute **Run As -> Junit Test**
4. Verify that tests are executed correctly in the console



# Challenge

1. What is the output of each method in the console?

The output of each method are the following:

•setup() : @BeforeAll - executes once before all test methods in this class

•init(): @BeforeEach - executes before each test method in this class

•simpleTest(): simple test

•otherSimpleTest(): other simple test

•notImplementedTest() \*: Not implemented yet

•tearDown(): @AfterEach - executed after each test method.

•done(): @AfterAll - executed after all test methods.

\*Note: The notImplementedTest(), while it has a text that could be printed in the console, on this lab practice there would be none, as the test is disabled using @Disabled.

The following lines represent an example of a Junit Test execution, as well as each method or test in order of appearance:

[main] INFO mx.tec.lab.FirstUnitTestApplicationTests - @BeforeAll - executes once before all test methods in this class

INFO 15480 --- [ main] m.tec.lab.FirstUnitTestApplicationTests : @BeforeEach - executes before each test method in this class

INFO 15480 --- [ main] m.tec.lab.FirstUnitTestApplicationTests : simple test

INFO 15480 --- [ main] m.tec.lab.FirstUnitTestApplicationTests : @AfterEach - executed after each test method.

INFO 15480 --- [ main] m.tec.lab.FirstUnitTestApplicationTests : @BeforeEach - executes before each test method in this class

INFO 15480 --- [ main] m.tec.lab.FirstUnitTestApplicationTests : other simple test

INFO 15480 --- [ main] m.tec.lab.FirstUnitTestApplicationTests : @AfterEach - executed after each test method.

INFO 15480 --- [ main] m.tec.lab.FirstUnitTestApplicationTests : @AfterAll - executed after all test methods.

1. What is the difference between the setup and init methods?

• The setup method is executed before any of the tests or methods begin their execution. It happens during the booting of the main context methods (these are marked as either [main] DEBUG or [main] INFO), which means this method happens only once. On the other hand, the init method only occurs right before the tests defined by the user (like simpleTest() or otherSimpleTest()), when the tests start executing. This method is executed once per test, though not when executing tearDown() or done() methods. Therefore, one simple example goes like INIT -> SIMPLETEST -> … -> INIT -> OTHERTEST -> … .

1. Give an example of when you can use init and tearDown methods

• Perhaps the init could be used to initialize a variable to a certain value right at the beginning before testing, and teardown could be used either to destroy it or to persist the resulting variable to a database or something.

1. What is @Disabled used for?

• @Disabled sets a test method as disabled, which means it will bypass the marked method. Perhaps it could be used to first define the tests that are wanted, then making sure these work in conjunction by means of adding one at a time. That, or just wanting not to execute the specific test without deleting it completely for whatever reason.

1. What is the difference and pros/cons of using Maven Test and Junit Test options?

• JUnit allows to see overall each run of the tests (whether disabled or not), and any encountered errors or failures on a separate section that appears along the console. The JUnit’s additional view displays each test’s completion time, the sum of runtimes for the project, and time elapsed. However, while both tests use the console, a JUnit test seems to omit certain details on it that a Maven Test can show. These include warnings about a missing “pom.xml” file or skipped tests, total finish time and date of the test, as well as some building information when initializing. In a certain way, it might be up to the user what test to use depending on what he/she wants to see.