Lab TDD 1: Setting up a TDD project

Objectives

In this lab, you will set up a basic TDD project using Junit 4 to write some code for a simple calculator project.

The purpose of this lab is to get you familiar with the mechanics of setting up a project and the TDD workflow.

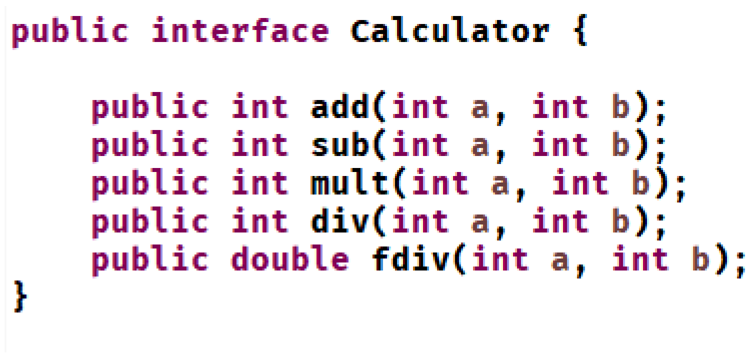
Part One – Project Setup

The specification for the project is to produce a simple calculator that implements a calculator interface with the four standard arithmetic operations and a method that calculates a floating point quotient.

All of the operations take two integers as inputs and return an integer, except for the floating point division which returns a double.

Step 1: Set up the project

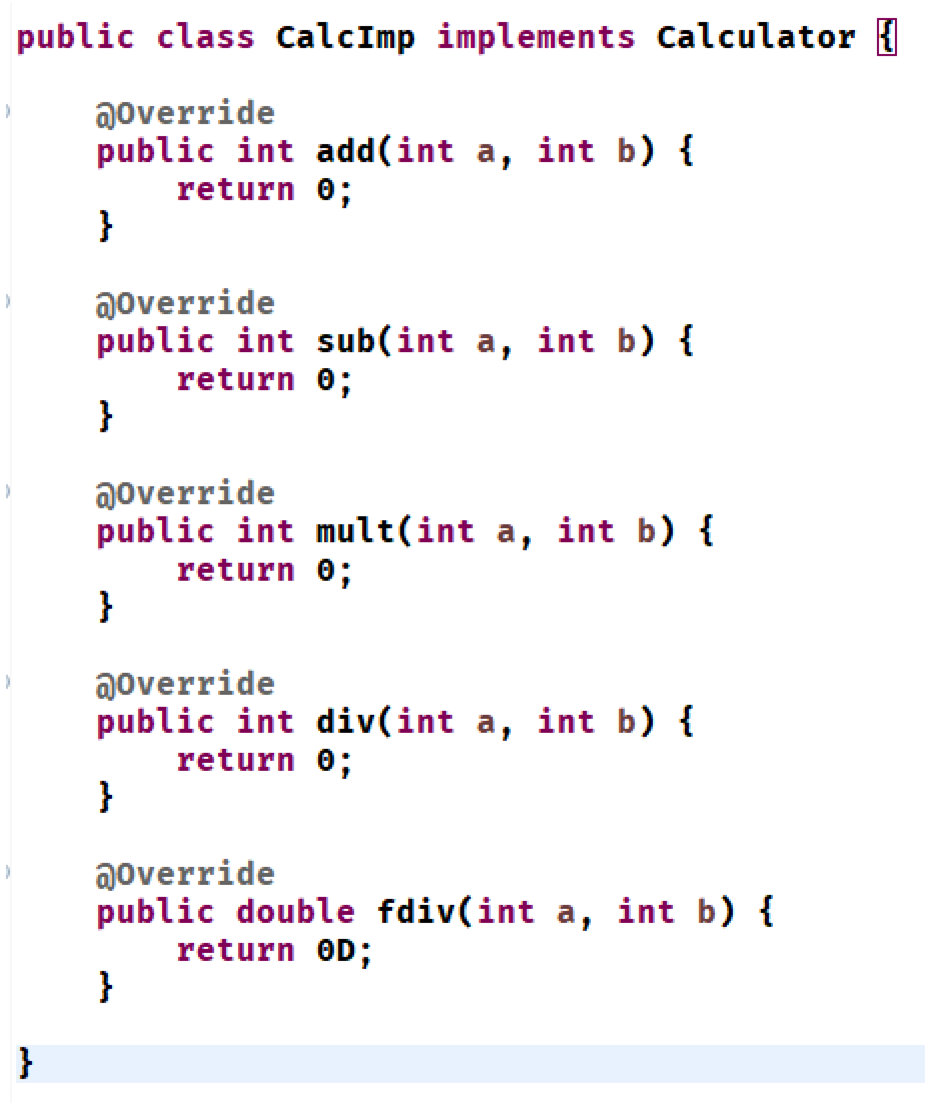
1. Create a standard Java project with a package named "calc"
2. Create an interface called Calculator



```
public interface Calculator {

    public int add(int a, int b);
    public int sub(int a, int b);
    public int mult(int a, int b);
    public int div(int a, int b);
    public double fdiv(int a, int b);
}
```

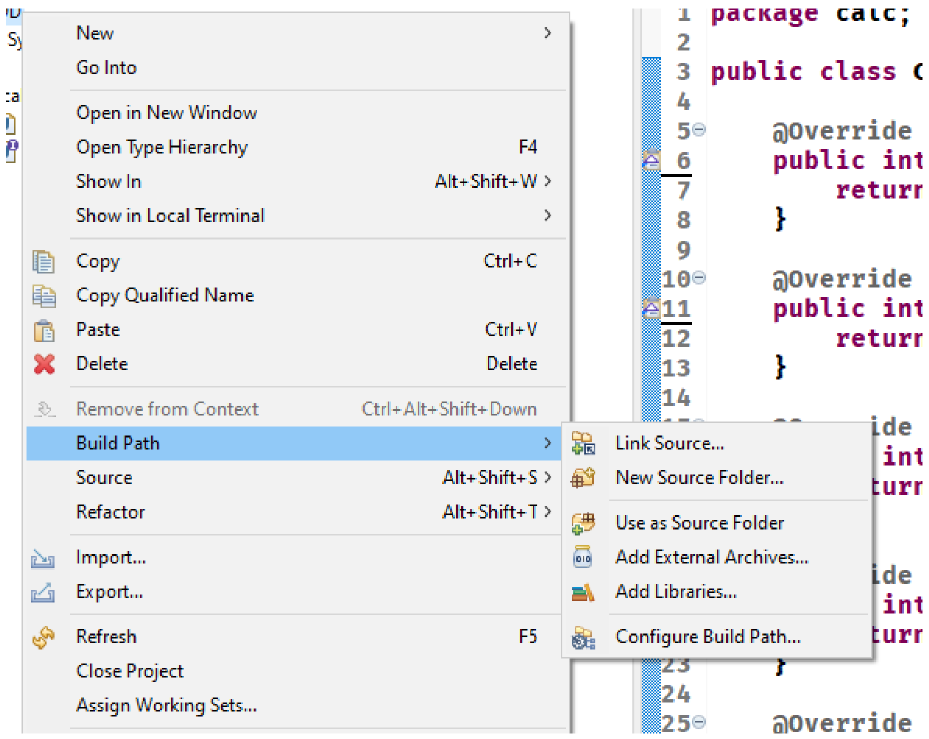
3. Create an implementation class called CalcImp that implements the Calculator interface.

4. The class methods should all have empty bodies, as shown on the next page.


```java
public class CalcImp implements Calculator {

    @Override
    public int add(int a, int b) {
        return 0;
    }

    @Override
    public int sub(int a, int b) {
        return 0;
    }

    @Override
    public int mult(int a, int b) {
        return 0;
    }

    @Override
    public int div(int a, int b) {
        return 0;
    }

    @Override
    public double fdiv(int a, int b) {
        return 0D;
    }

}
```

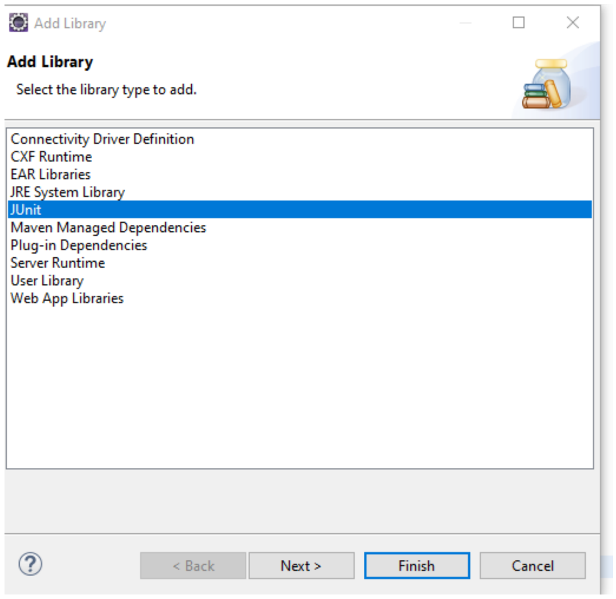
Step 2: Add Junit

1. Add the Junit 4 library to the build path by selecting build path and selecting "Add libraries"



```
package calc;

public class C

    @Override
    public int
        return
    }

    @Override
    public int
        return
    }
```

2. Select JUnit

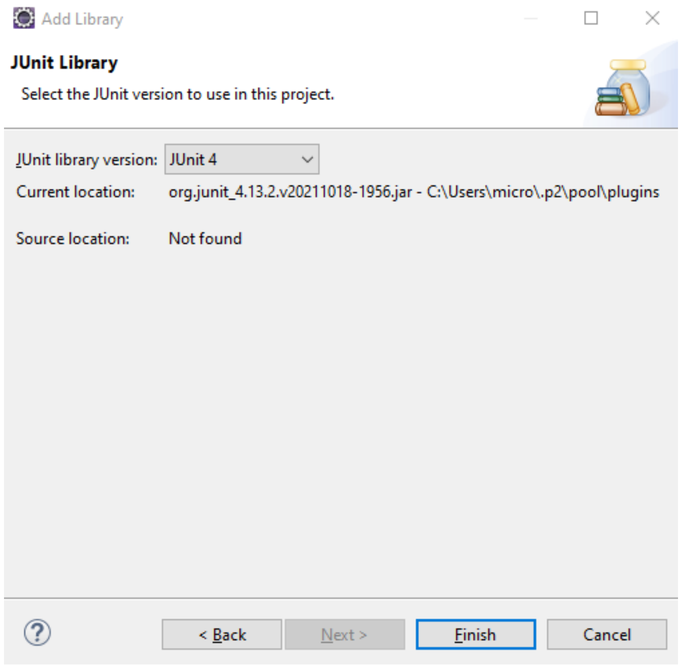


Add Library

Select the library type to add.

Connectivity Driver Definition
CXF Runtime
EAR Libraries
JRE System Library
JUnit
Maven Managed Dependencies
Plug-in Dependencies
Server Runtime
User Library
Web App Libraries

3. Select Next and JUnit 4 then select finish



Add Library

JUnit Library

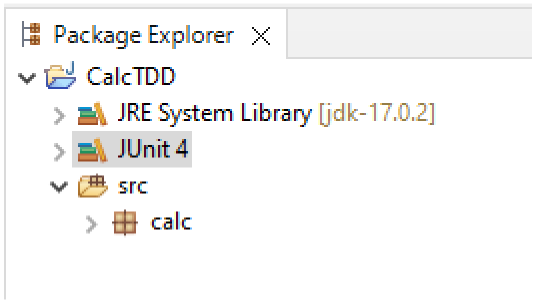
Select the JUnit version to use in this project.

JUnit library version: JUnit 4

Current location: org.junit_4.13.2.v20211018-1956.jar - C:\Users\micro\.p2\pool\plugins

Source location: Not found

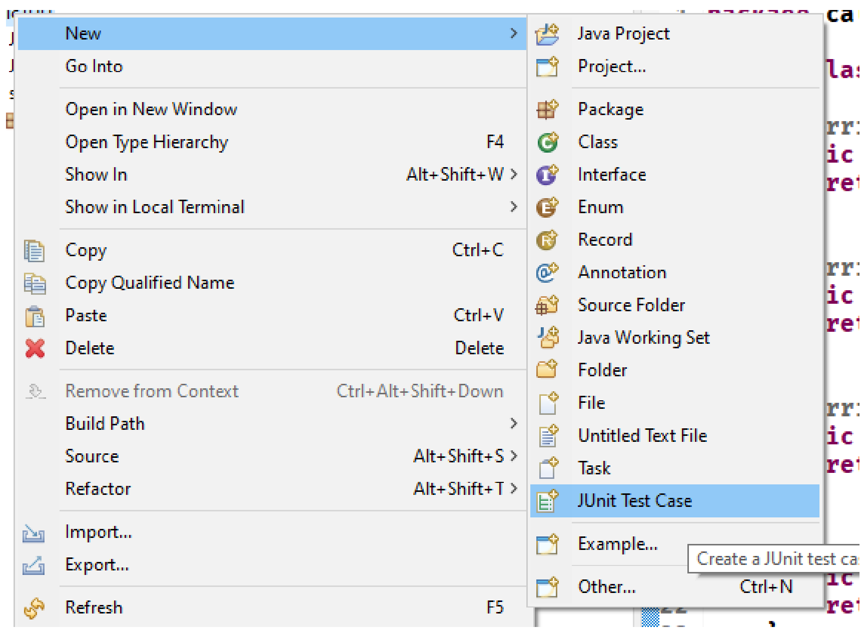
4. You should see the library now as part of the project



```
Package Explorer
CalcTDD
  JRE System Library [jdk-17.0.2]
  JUnit 4
  src
    calc
```

Step 4: Create a test class

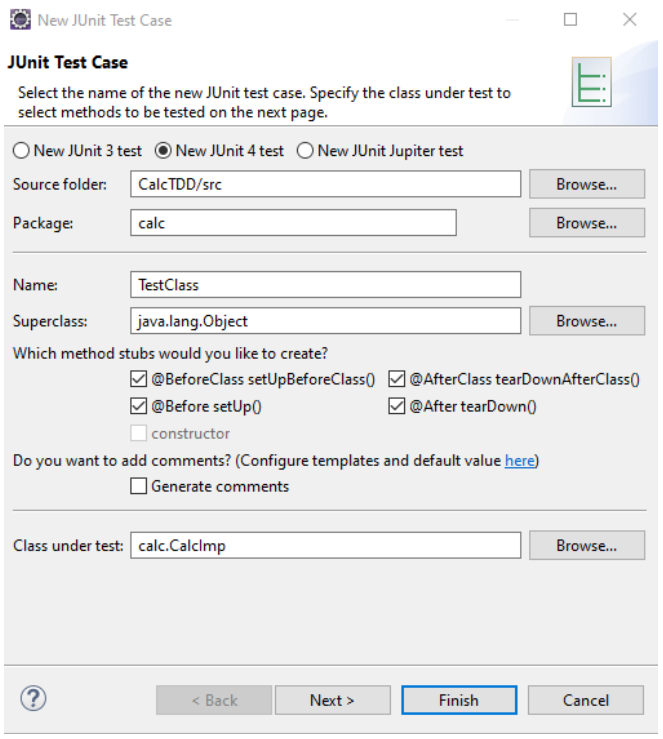
1. Select new JUnit test case from the new menu.



2. JUnit will create a new test class to hold the test methods. In this case, you can name the test class whatever you want. In the example the imaginative name “TestClass” is used.

3. Chose all the fixture stubs to be autogenerated

4. Select the CalcImp as the class under test



JUnit Test Case

Select the name of the new JUnit test case. Specify the class under test to select methods to be tested on the next page.

( ) New JUnit 3 test [✓ New JUnit 4 test] ( ) New JUnit Jupiter test

Source folder: CalcTDD/src

Package: calc

Name: TestClass

Superclass: java.lang.Object

Which method stubs would you like to create?

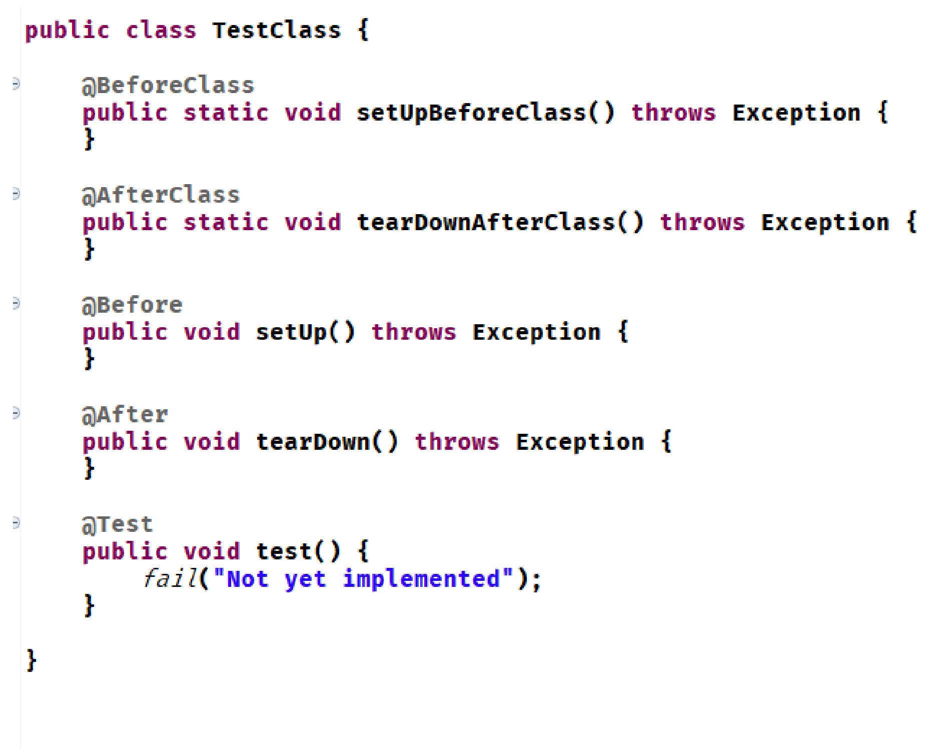
- [✓] @BeforeClass setUpBeforeClass()
- [✓] @AfterClass tearDownAfterClass()
- [✓] @Before setUp()
- [✓] @After tearDown()
- [ ] constructor

Do you want to add comments? (Configure templates and default value here)

- [ ] Generate comments

Class under test: calc.CalcImp

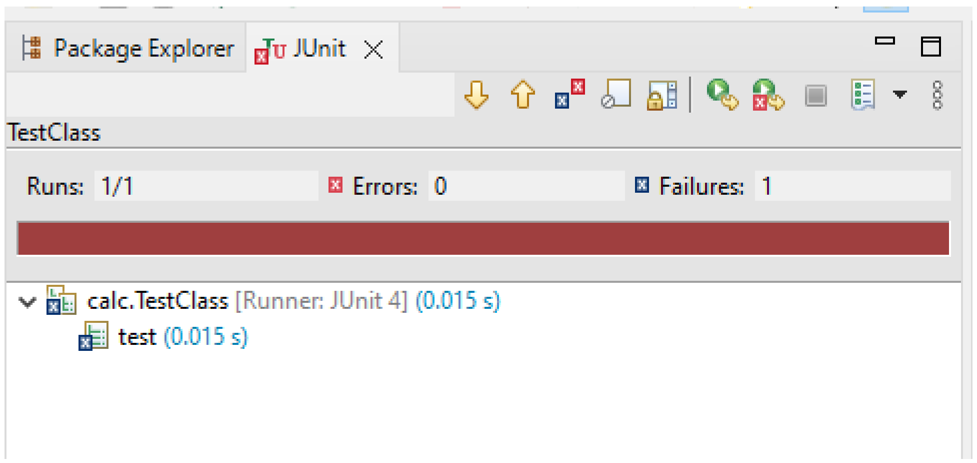
5. This produces the class on the next page.



```
public class TestClass {

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void test() {
        fail("Not yet implemented");
    }

}
```

Part 5: Run the tests

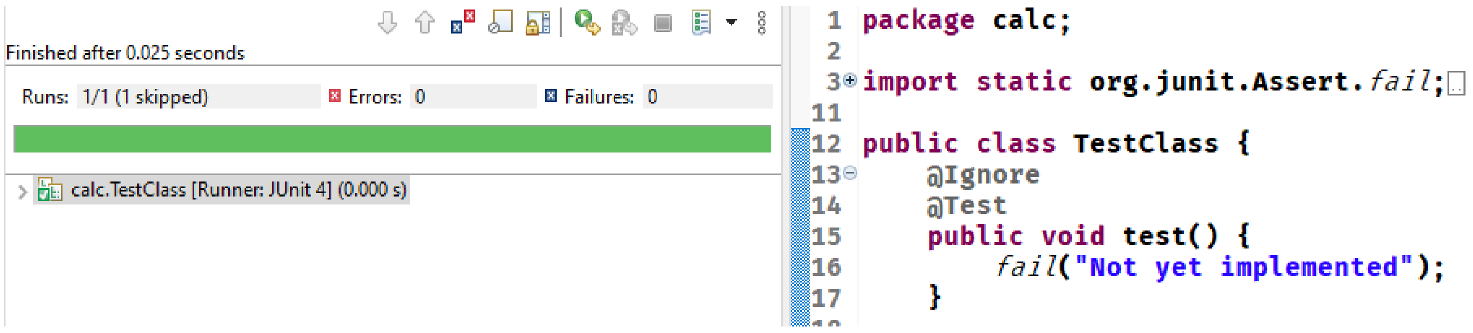
1. In the test class, right mouse click and select the "run as JUnit test case" option.
2. The test output should show the one test method we have has failed. This is because the default "fail()" method throws a test fail exception.



```
TestClass

Runs: 1/1        Errors: 0        Failures: 1

calc.TestClass [Runner: JUnit 4] (0.015 s)
    test (0.015 s)
```

Part 6: Ignore the test

1. To get JUnit to skip the test, add the @Ignore annotation and rerun the tests
2. Notice that all the tests passed but it also notes that one test is skipped



Finished after 0.025 seconds

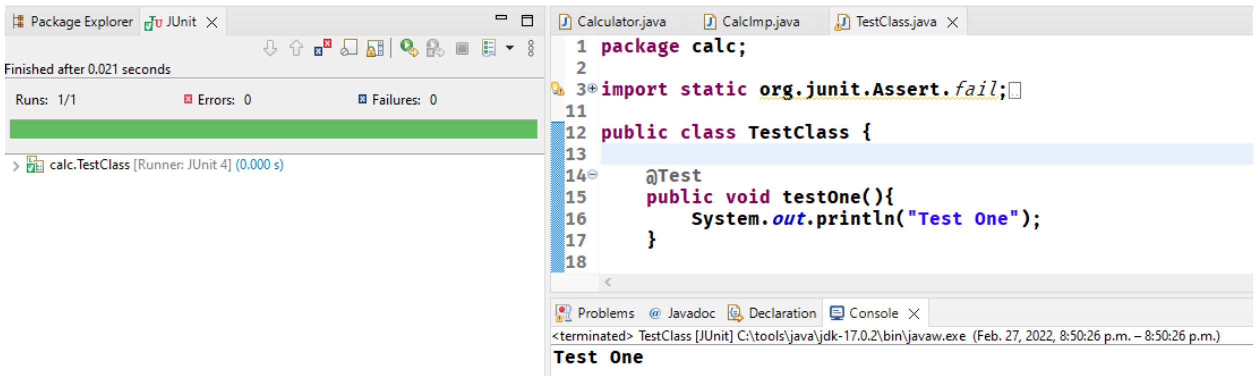
Runs: 1/1 (1 skipped) Errors: 0 Failures: 0

calc.TestClass [Runner: JUnit 4] (0.000 s)

```java
package calc;

import static org.junit.Assert.fail;

public class TestClass {
    @Ignore
    @Test
    public void test() {
        fail("Not yet implemented");
    }
```

Part 7: Make the test past vacuously

1. Remove the @Ignore annotation and replace the fail() method with an output method that prints a message to the console
2. Also change the name of the test method to testOne()



Finished after 0.021 seconds

Runs: 1/1 Errors: 0 Failures: 0

calc.TestClass [Runner: JUnit 4] (0.000 s)

```java
package calc;

import static org.junit.Assert.fail;

public class TestClass {

    @Test
    public void testOne(){
        System.out.println("Test One");
    }
```

```
<terminated> TestClass [JUnit] C:\tools\java\jdk-17.0.2\bin\javaw.exe (Feb. 27, 2022, 8:50:26 p.m. – 8:50:26 p.m.)
Test One
```

Save your lab because the next lab picks up from here.