

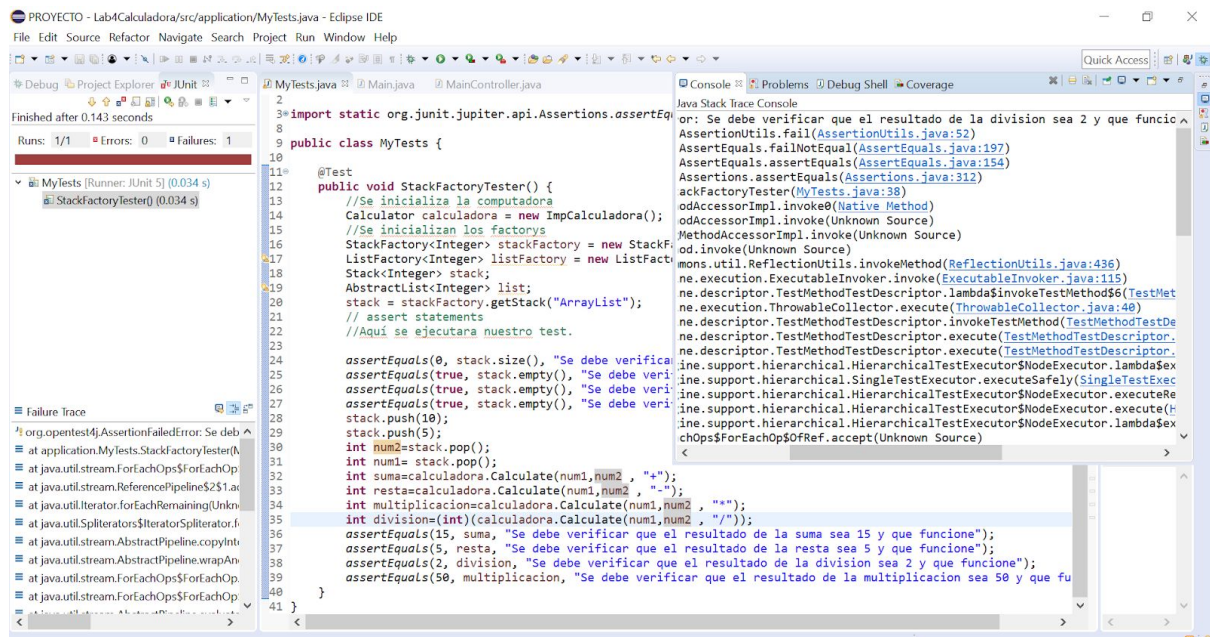
Lab 4

Para instanciar la clase calculadora se utilizó el patrón de diseño Singleton y esto fue adecuado para el programa ya que garantiza que el programa solo existe una instancia de la clase factory y segura de que no van a haber dos tipos de calculadoras. Por lo que sí fue útil este patrón de diseño.

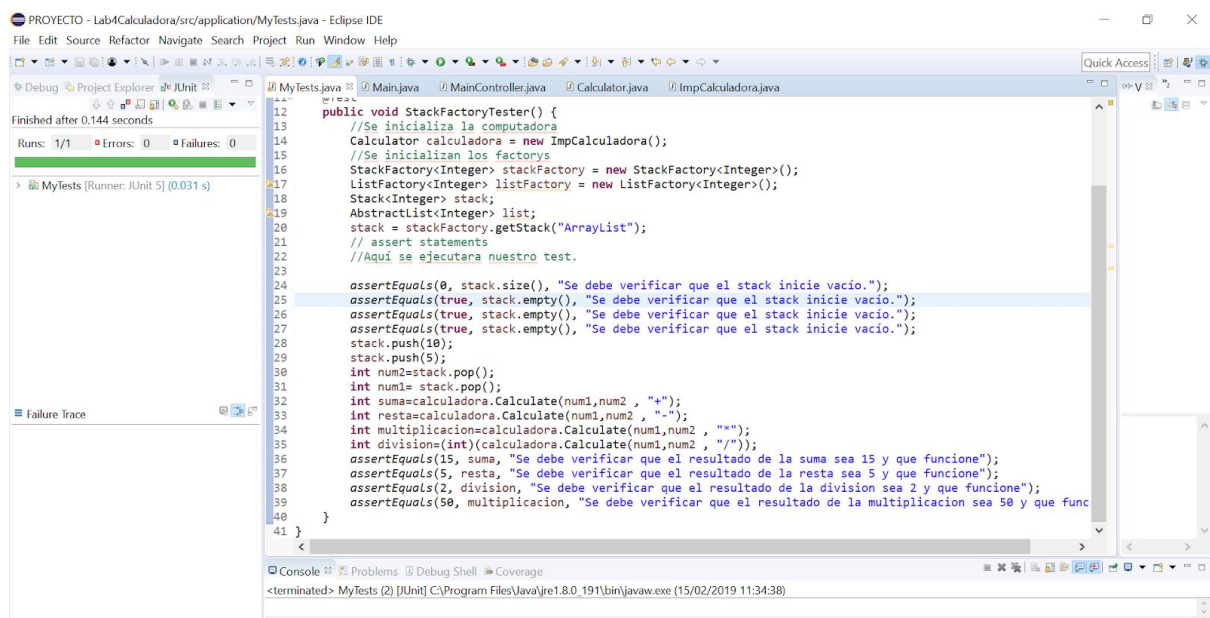
JUNITS

Pruebas StackFactory

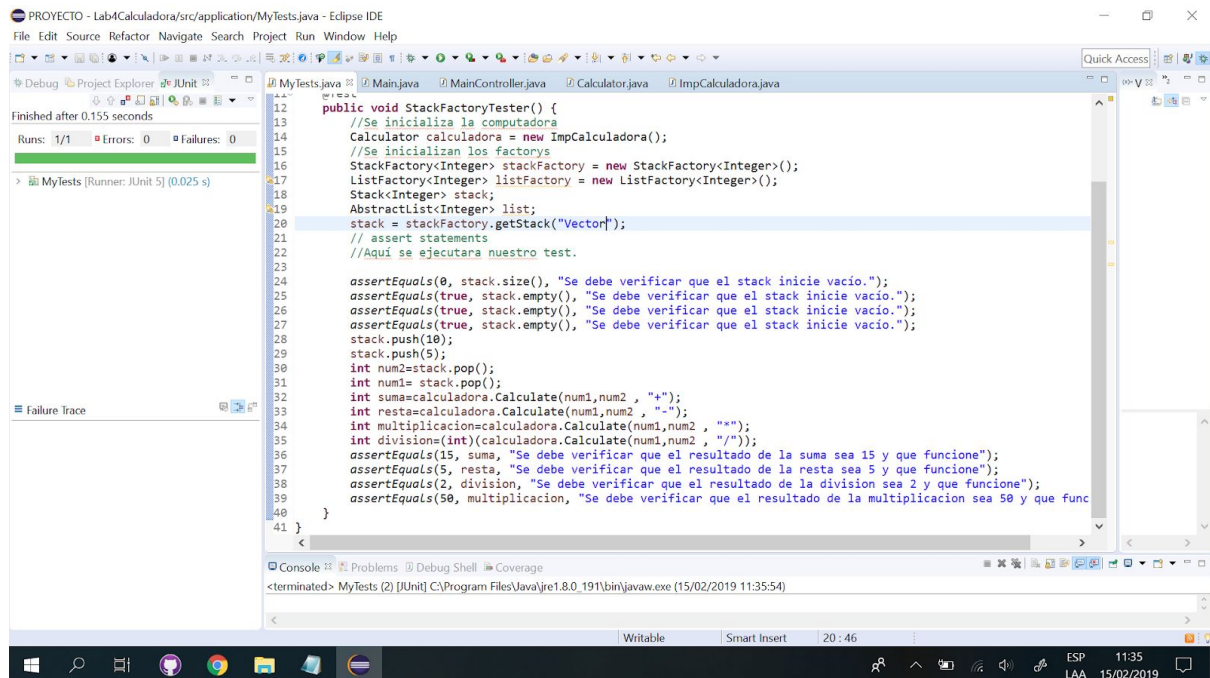
Prueba Stack Array List



Teníamos un error en la calculadora para la division, ya que los pops recibidos estaban cambiados.



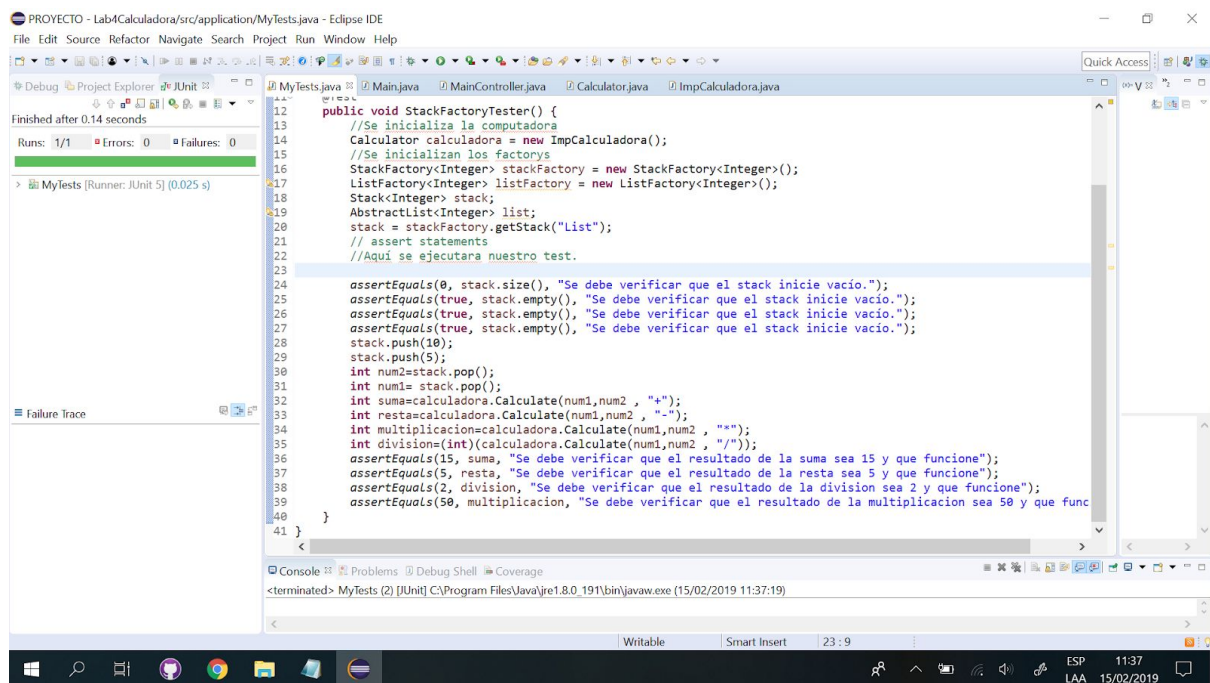
Prueba Vector



The screenshot shows the Eclipse IDE with a Java project named 'PROYECTO - Lab4Calculadora'. The main editor displays the file 'MyTests.java'. The code defines a test method 'StackFactoryTester()' which initializes a calculator and a stack. It then performs several assertions to verify the stack's behavior, including checking its size, emptiness, and the results of push, pop, and calculate operations. The console at the bottom shows the test execution output, indicating that the test passed successfully.

```
12 public void StackFactoryTester() {
13     //Se inicializa la computadora
14     Calculator calculadora = new ImpCalculadora();
15     //Se inicializan los factories
16     StackFactory<Integer> stackFactory = new StackFactory<Integer>();
17     ListFactory<Integer> listFactory = new ListFactory<Integer>();
18     Stack<Integer> stack;
19     AbstractList<Integer> list;
20     stack = stackFactory.getStack("Vector");
21     // assert statements
22     //Aquí se ejecutara nuestro test.
23
24     assertEquals(0, stack.size(), "Se debe verificar que el stack inicie vacío.");
25     assertEquals(true, stack.empty(), "Se debe verificar que el stack inicie vacío.");
26     assertEquals(true, stack.empty(), "Se debe verificar que el stack inicie vacío.");
27     assertEquals(true, stack.empty(), "Se debe verificar que el stack inicie vacío.");
28     stack.push(10);
29     stack.push(5);
30     int num2=stack.pop();
31     int num1= stack.pop();
32     int suma=calculadora.Calculate(num1,num2 , "+");
33     int resta=calculadora.Calculate(num1,num2 , "-");
34     int multiplicacion=calculadora.Calculate(num1,num2 , "*");
35     int divisione(int)(calculadora.Calculate(num1,num2 , "/"));
36     assertEquals(15, suma, "Se debe verificar que el resultado de la suma sea 15 y que funcione");
37     assertEquals(5, resta, "Se debe verificar que el resultado de la resta sea 5 y que funcione");
38     assertEquals(2, division, "Se debe verificar que el resultado de la division sea 2 y que funcione");
39     assertEquals(50, multiplicacion, "Se debe verificar que el resultado de la multiplicacion sea 50 y que funcione");
40 }
41 }
```

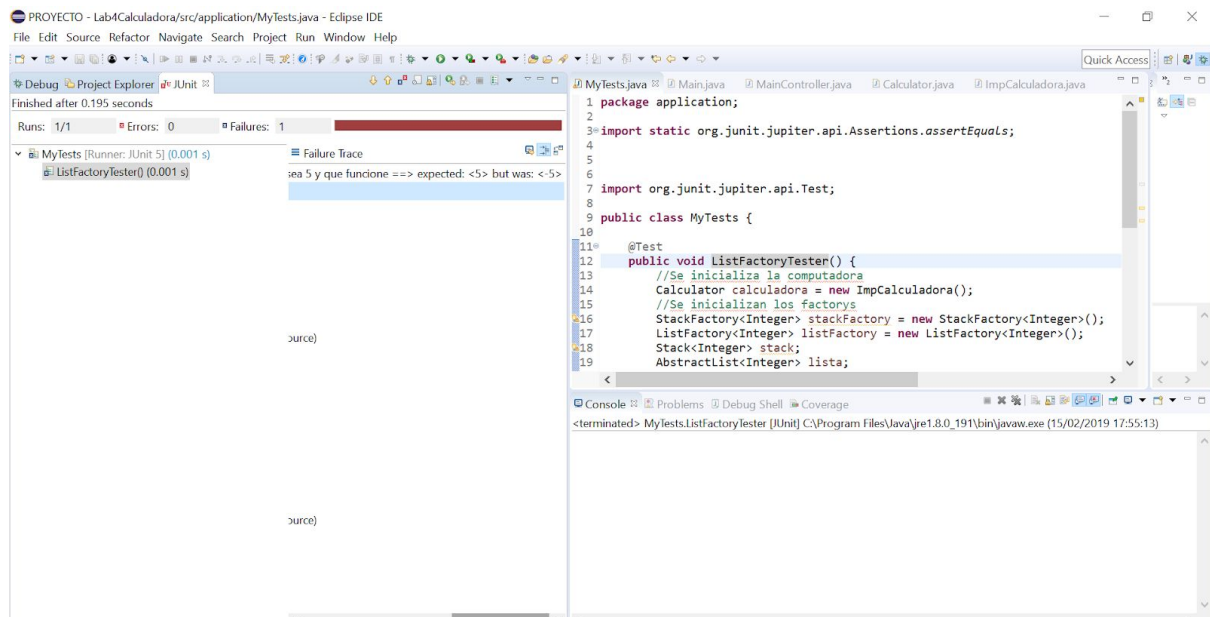
Prueba List



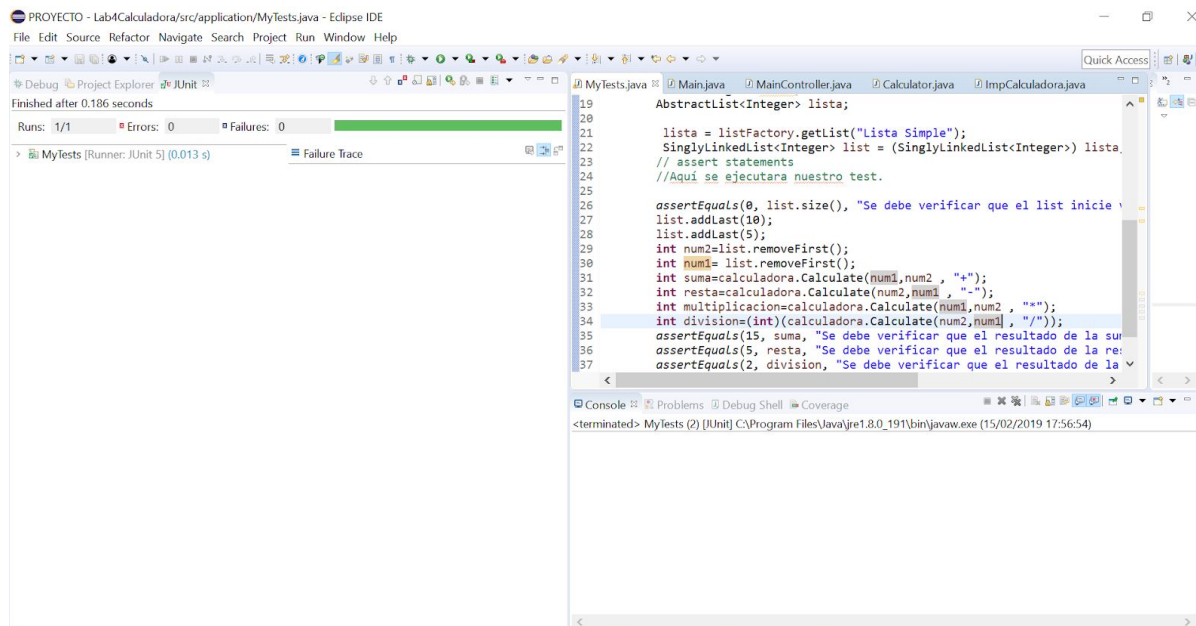
The screenshot shows the Eclipse IDE with the same Java project. The main editor displays the file 'MyTests.java'. The code defines a test method 'StackFactoryTester()' which initializes a calculator and a stack. It then performs several assertions to verify the stack's behavior, including checking its size, emptiness, and the results of push, pop, and calculate operations. The console at the bottom shows the test execution output, indicating that the test passed successfully.

```
12 public void StackFactoryTester() {
13     //Se inicializa la computadora
14     Calculator calculadora = new ImpCalculadora();
15     //Se inicializan los factories
16     StackFactory<Integer> stackFactory = new StackFactory<Integer>();
17     ListFactory<Integer> listFactory = new ListFactory<Integer>();
18     Stack<Integer> stack;
19     AbstractList<Integer> list;
20     stack = stackFactory.getStack("List");
21     // assert statements
22     //Aquí se ejecutara nuestro test.
23
24     assertEquals(0, stack.size(), "Se debe verificar que el stack inicie vacío.");
25     assertEquals(true, stack.empty(), "Se debe verificar que el stack inicie vacío.");
26     assertEquals(true, stack.empty(), "Se debe verificar que el stack inicie vacío.");
27     assertEquals(true, stack.empty(), "Se debe verificar que el stack inicie vacío.");
28     stack.push(10);
29     stack.push(5);
30     int num2=stack.pop();
31     int num1= stack.pop();
32     int suma=calculadora.Calculate(num1,num2 , "+");
33     int resta=calculadora.Calculate(num1,num2 , "-");
34     int multiplicacion=calculadora.Calculate(num1,num2 , "*");
35     int divisione(int)(calculadora.Calculate(num1,num2 , "/"));
36     assertEquals(15, suma, "Se debe verificar que el resultado de la suma sea 15 y que funcione");
37     assertEquals(5, resta, "Se debe verificar que el resultado de la resta sea 5 y que funcione");
38     assertEquals(2, division, "Se debe verificar que el resultado de la division sea 2 y que funcione");
39     assertEquals(50, multiplicacion, "Se debe verificar que el resultado de la multiplicacion sea 50 y que funcione");
40 }
41 }
```

Pruebas ListFactory SingleLinkedListTest

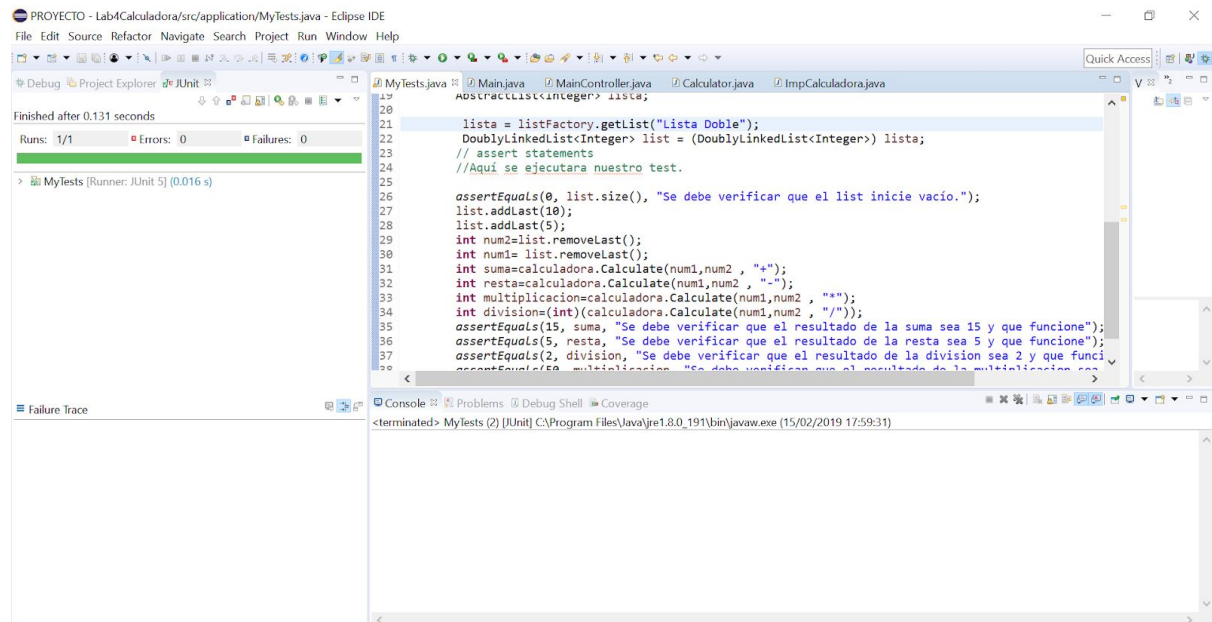


Para este hubo error ya que saca el primero de la lista y no el ultimo, como el metodo pop por lo que los resultados de resta y division estaban cambiados.

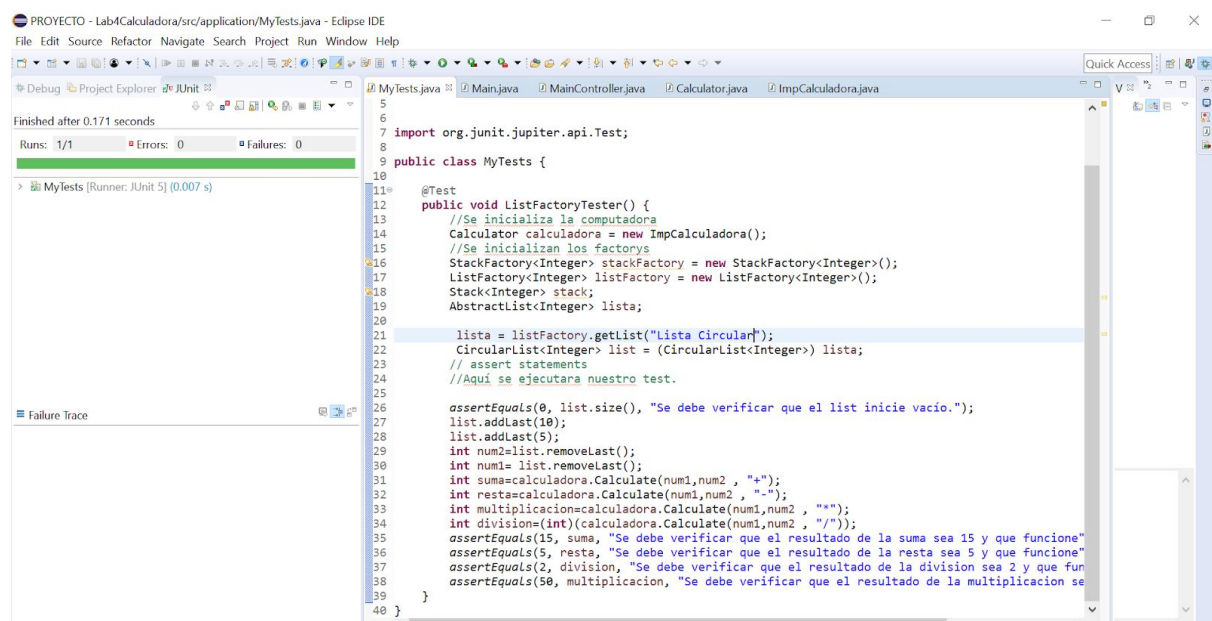


Luego ya cambiamos y lo arreglamos.

DoublyLinkedList



CircularList



En todas las pruebas se utiliza la calculadora.