

# Tarea 8

DESARROLLO DE INTERFACES

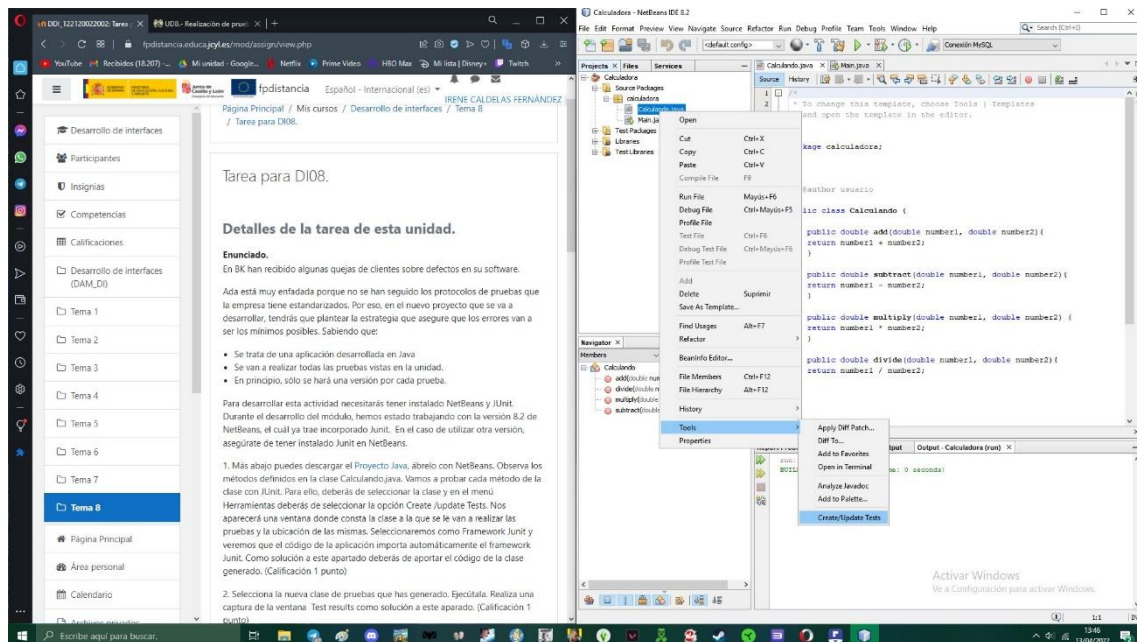
IRENE CALDELAS FERNÁNDEZ

## Contenido

Apartado 1.....	1
Apartado 2.....	3
Apartado 3.....	4
Apartado 4.....	5
Apartado 5.....	6
Apartado 6.....	9

## Apartado 1

Más abajo puedes descargar el [Proyecto Java](#), ábrelo con NetBeans. Observa los métodos definidos en la clase Calculando.java. Vamos a probar cada método de la clase con JUnit. Para ello, deberás de seleccionar la clase y en el menú Herramientas deberás de seleccionar la opción Create /update Tests. Nos aparecerá una ventana donde consta la clase a la que se le van a realizar las pruebas y la ubicación de las mismas. Seleccionaremos como Framework Junit y veremos que el código de la aplicación importa automáticamente el framework Junit. Como solución a este apartado deberás de aportar el código de la clase generado.



## Tarea 7 DDI – Irene Caldelas Fernández

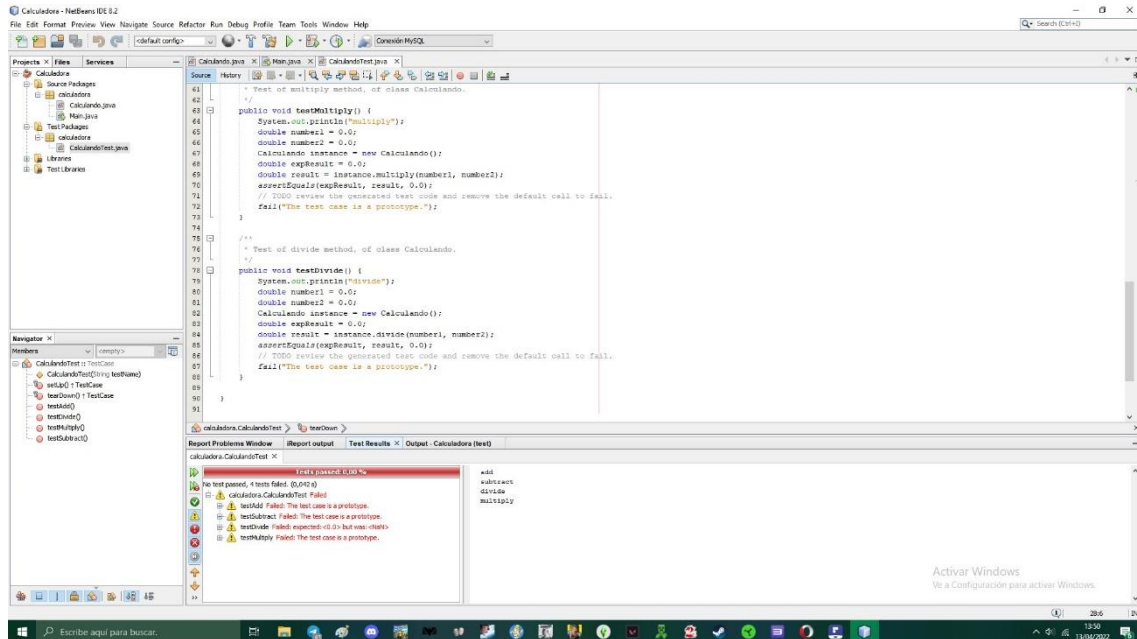
```
Calculando.java x Main.java x CalculandoTest.java x
Source History
1  /**
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package calculadora;
7
8   import junit.framework.TestCase;
9
10  /**
11   *
12   * @author Irene Caldelas Fernández
13   */
14  public class CalculandoTest extends TestCase {
15
16      public CalculandoTest(String testName) {
17          super(testName);
18      }
19
20      @Override
21      protected void setUp() throws Exception {
22          super.setUp();
23      }
24
25      @Override
26      protected void tearDown() throws Exception {
27          super.tearDown();
28      }
29
30      /**
31       * Test of add method, of class Calculando.
32       */
33  }
```

```
25  @Override
26  protected void tearDown() throws Exception {
27      super.tearDown();
28  }
29
30  /**
31   * Test of add method, of class Calculando.
32   */
33  public void testAdd() {
34      System.out.println("add");
35      double number1 = 0.0;
36      double number2 = 0.0;
37      Calculando instance = new Calculando();
38      double expectedResult = 0.0;
39      double result = instance.add(number1, number2);
40      assertEquals(expResult, result, 0.0);
41      // TODO review the generated test code and remove the default call to fail.
42      fail("The test case is a prototype.");
43  }
44
45  /**
46   * Test of subtract method, of class Calculando.
47   */
48  public void testSubtract() {
49      System.out.println("subtract");
50      double number1 = 0.0;
51      double number2 = 0.0;
52      Calculando instance = new Calculando();
53      double expectedResult = 0.0;
54      double result = instance.subtract(number1, number2);
55      assertEquals(expResult, result, 0.0);
56      // TODO review the generated test code and remove the default call to fail.
57      fail("The test case is a prototype.");
58  }
59
60  /**
61   * Test of multiply method, of class Calculando.
62   */
63  }
```

```
Calculando.java x Main.java x CalculandoTest.java x
Source History
61  /**
62   * Test of multiply method, of class Calculando.
63   */
64  public void testMultiply() {
65      System.out.println("multiply");
66      double number1 = 0.0;
67      double number2 = 0.0;
68      Calculando instance = new Calculando();
69      double expectedResult = 0.0;
70      double result = instance.multiply(number1, number2);
71      assertEquals(expResult, result, 0.0);
72      // TODO review the generated test code and remove the default call to fail.
73      fail("The test case is a prototype.");
74  }
75
76  /**
77   * Test of divide method, of class Calculando.
78   */
79  public void testDivide() {
80      System.out.println("divide");
81      double number1 = 0.0;
82      double number2 = 0.0;
83      Calculando instance = new Calculando();
84      double expectedResult = 0.0;
85      double result = instance.divide(number1, number2);
86      assertEquals(expResult, result, 0.0);
87      // TODO review the generated test code and remove the default call to fail.
88      fail("The test case is a prototype.");
89  }
90
91  }
```

## Apartado 2

Selecciona la nueva clase de pruebas que has generado. Ejecútala. Realiza una captura de la ventana Test results como solución a este apartado.



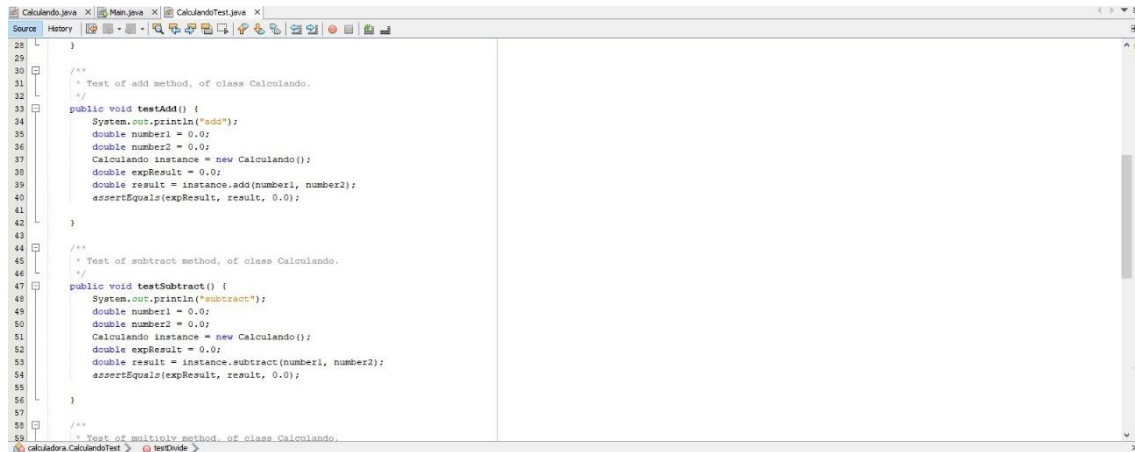
## Apartado 3

Accede al código de la clase de pruebas y elimina las líneas:

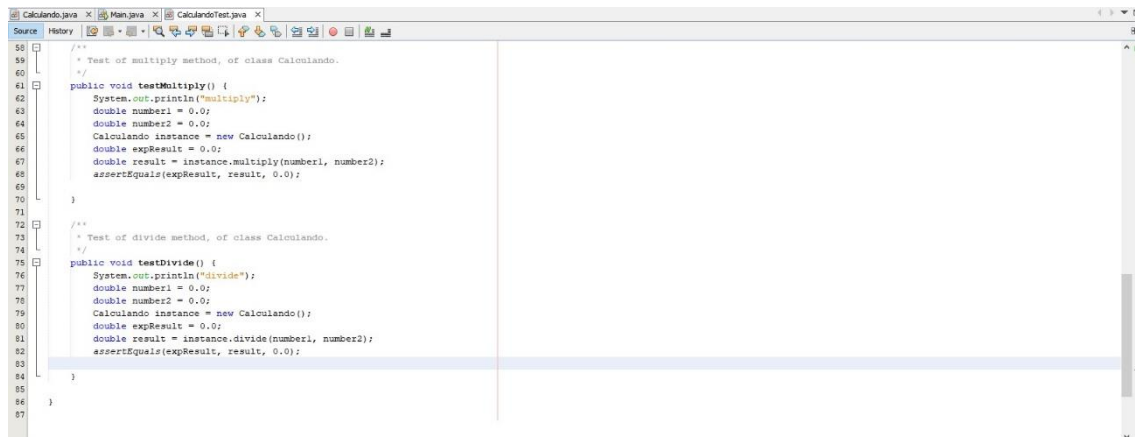
```
// TODO review the generated test code and remove the default call to fail.
```

```
fail("The test case is a prototype.");
```

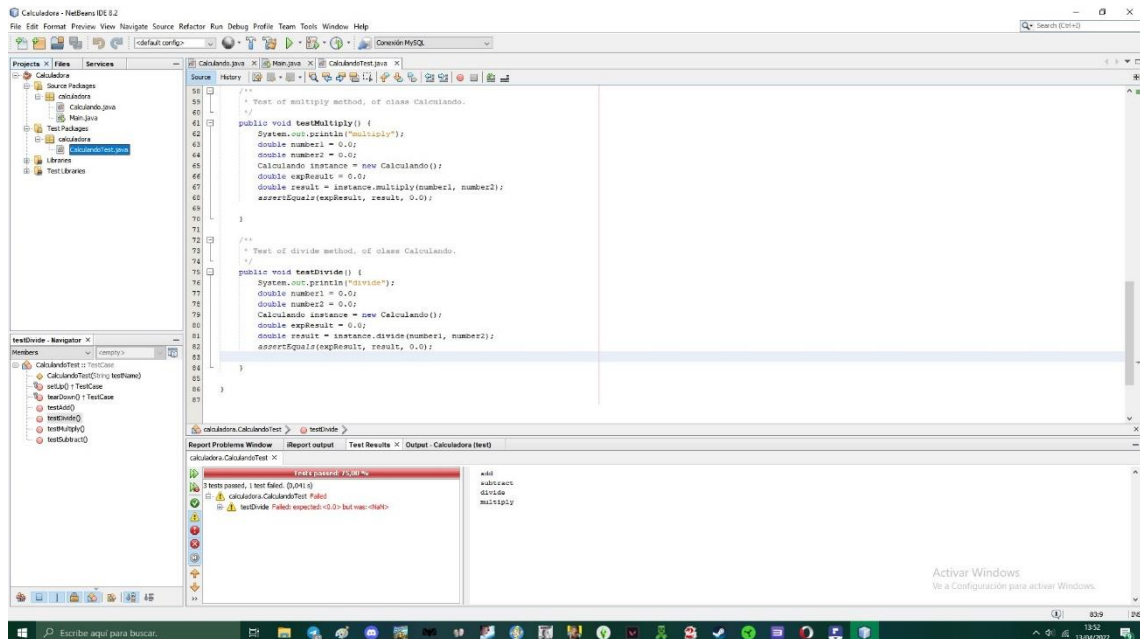
que aparece al final de cada método. Como solución a este apartado deberás de entregar el código de la clase generado.



```
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87
```

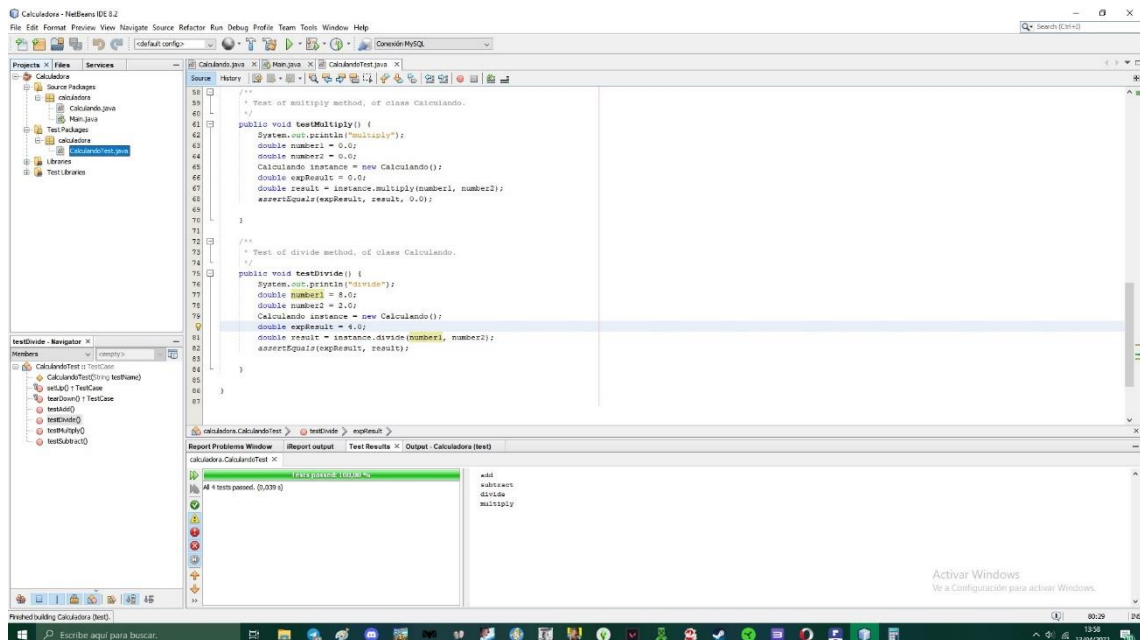


```
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87
```



## Apartado 4

Selecciona la clase de prueba y ejecútala de nuevo. Debes de corregir todos los errores asignándole valores a las variables. Al final, debes de conseguir que la ejecución de la prueba sea satisfactoria. Como solución a este apartado deberás de aportar el código de la clase de prueba una vez que ha sido modificado para conseguir que las pruebas fueran satisfactorias.



## Apartado 5

Implementa la planificación de las pruebas de integración, sistema y regresión.

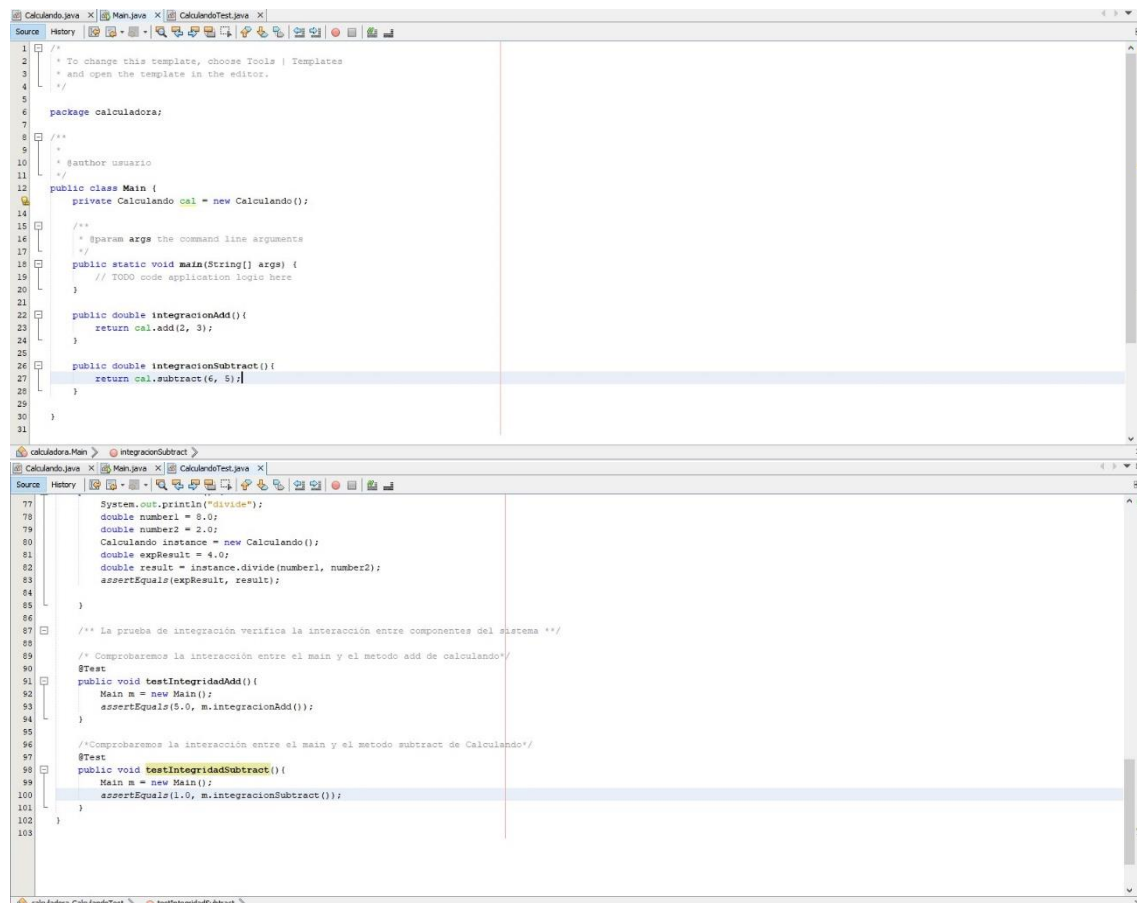
Las pruebas de integración son aquellas que se realizan una vez probado el funcionamiento de las partes o módulos que componen la aplicación por separado, es decir, una vez concluidas las pruebas unitarias.

Consiste en verificar que el software, en conjunto, cumple su misión y se realiza sobre la unión de todos los módulos a la vez, para probar que la interrelación entre ellos no da lugar a ningún error o defecto.

Lo que haremos será comprobar la relación que hay entre la clase main y la clase Calculando, llamando desde esta primera a los métodos de calculando.

En el caso de la suma esperaremos un 5 de resultado y en el caso de la resta un 1. En los tests vamos a llamar a la clase main y desde ella llamaremos a los métodos add y subtract, donde con assertEquals comprobaremos si el valor esperado son los mismos que el que nos devuelven los métodos. En el caso de la multiplicación y de la división haremos exactamente lo mismo esperando en el caso de la multiplicación un resultado de 9 y en el caso de la división un resultado de 3.

Como podemos comprobar a la hora de ejecutar los tests estos se realizan sin problema ninguno y sin ningún error.

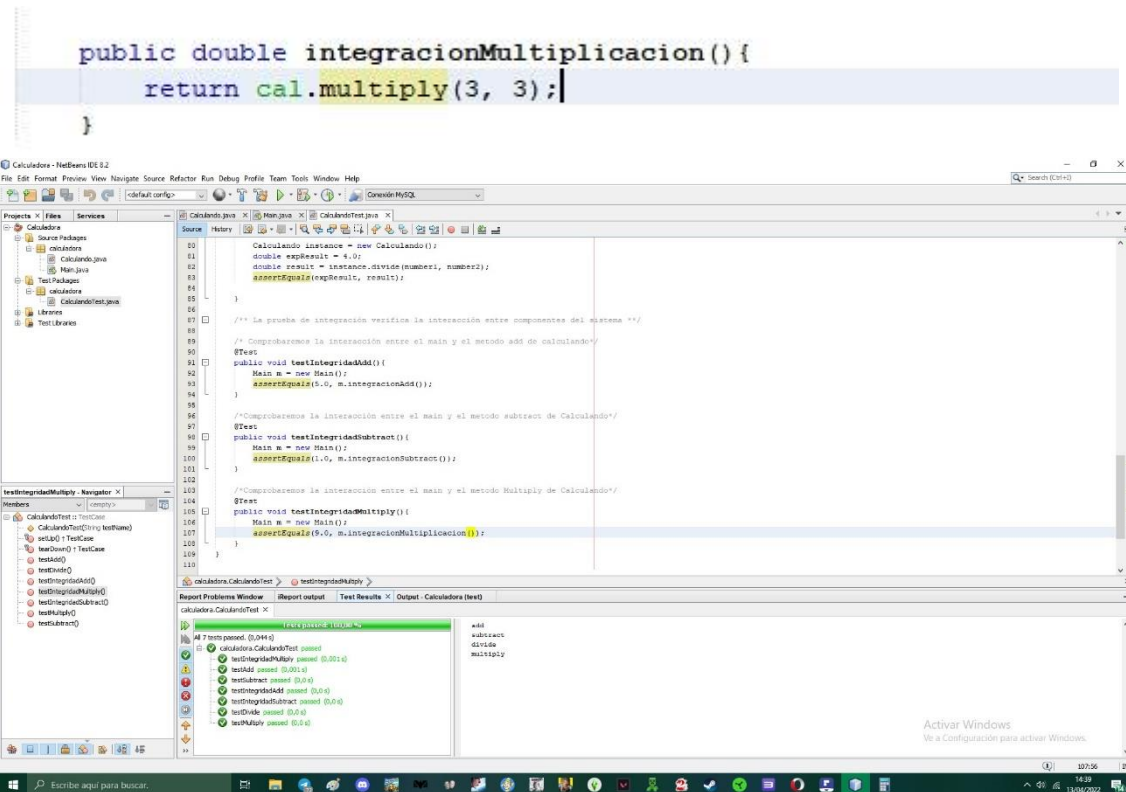


```

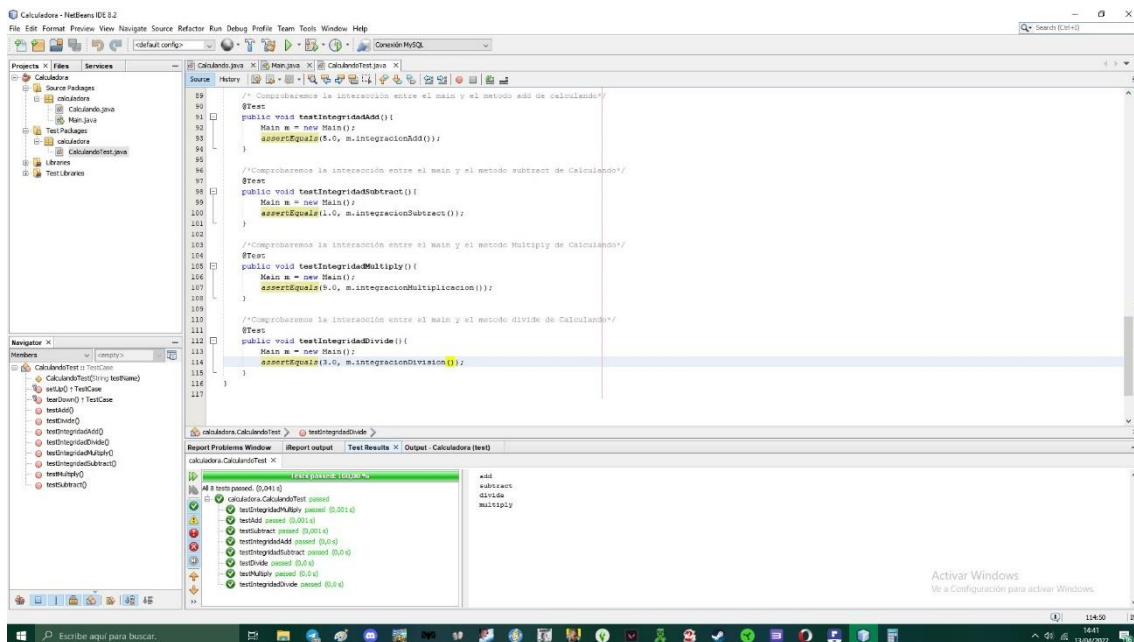
1  /**
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5
6   package calculadora;
7
8   /**
9    *
10   * @author usuario
11   */
12   public class Main {
13       private Calculando cal = new Calculando();
14
15       /**
16        * @param args the command line arguments
17        */
18       public static void main(String[] args) {
19           // TODO code application logic here
20       }
21
22       public double integracionAdd(){
23           return cal.add(2, 3);
24       }
25
26       public double integracionSubtract(){
27           return cal.subtract(6, 5);
28       }
29   }
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103

```

## Tarea 7 DDI – Irene Caldelas Fernández



```
public double integracionDivision(){
    return cal.divide(9, 3);
}
```





En el caso de las pruebas de sistema consisten en que verifican el comportamiento del sistema en s conjunto siendo una de las más usadas la recuperación de errores y el tratamiento de excepciones.

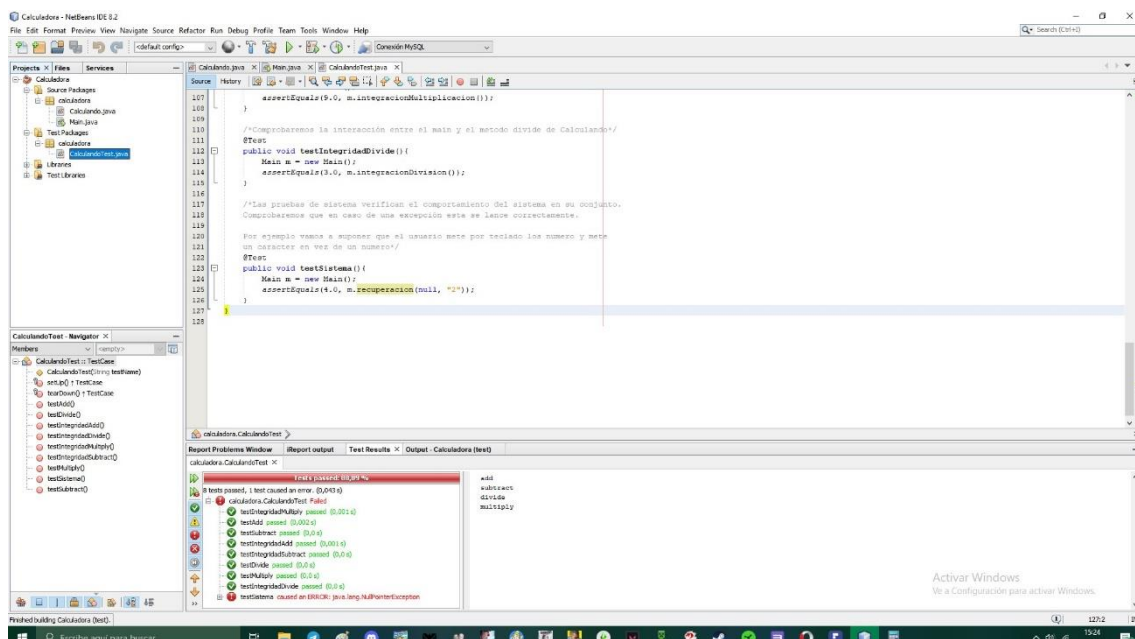
Para este caso vamos a suponer que nos pasan un número que sea nulo. Para poder ponernos en este caso deberemos crear un nuevo método que reciba como parámetros dos Strings aunque también podríamos haber decidido que se pasaran un String y un double. Es importante que parseemos los Strings a double para que el método de sumar (que en este caso es el que he elegido, pero sería exactamente igual con el resto) nos coja sin problemas los parámetros.

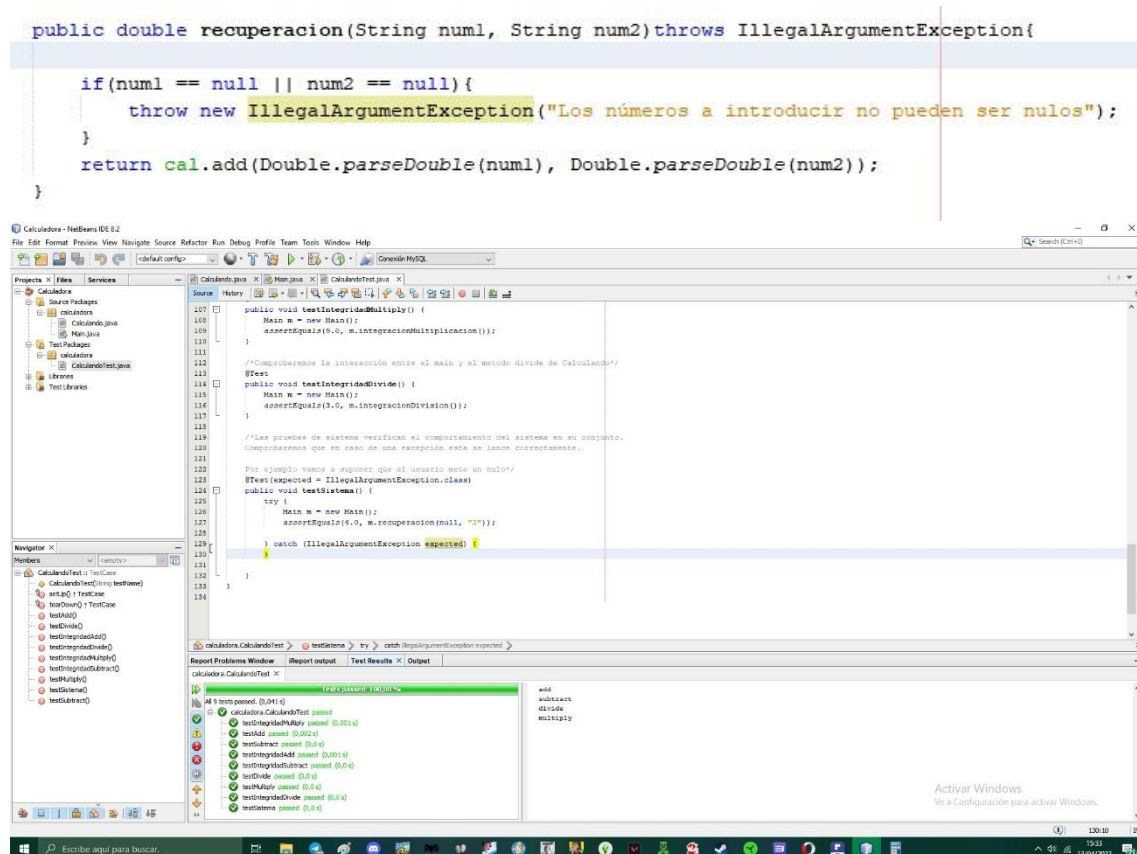
Vamos a los tests y nos ponemos en el caso de que nos metan un null sin controlarlo. Como vemos salta un error y vemos el test en rojo. Para que el test pase sin problemas tenemos que controlar este error. Volvemos al método que hemos utilizado con los parámetros de tipo String y le vamos a decir que lance una excepción si el argumento no es válido. Además como queremos controlar el tema de nulos pondremos al condición de que si alguno de los números que se pasan es nulo lance la excepción. También podríamos haber elegido un NullPointerException.

Ahora en los tests tenemos que poner que esperamos que se reciba esta excepción y encerrar la llamada al método en un bloque try-catch. Como vemos ahora el test pasa sin problemas.

En caso de que nos pasan solo un número sería lo mismo pero con la condición de .isEmpty y en caso de hacer las transformaciones de String a double y querer controlarlo tendríamos ver que sea posible parsearlo a un número y de no ser así lanzar una excepción. Por ejemplo "cuatro" + "dos" lo podría parsear pero "cuatro" + "hola" no.

```
public double recuperacion(String num1, String num2){
    return cal.add(Double.parseDouble(num1), Double.parseDouble(num2));
}
```





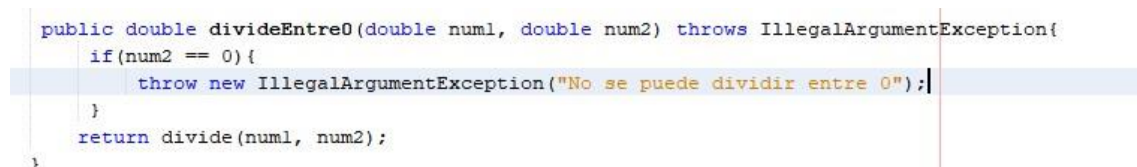
En el caso de la regresión consiste en que si añadimos algún módulo nuevo a la aplicación no debería de error en los tests anteriores. Este último test fue añadido a posteriori de los hechos en integración sobre el método main. Como vemos no ha habido ningún problema con los tests anteriores.

## Apartado 6

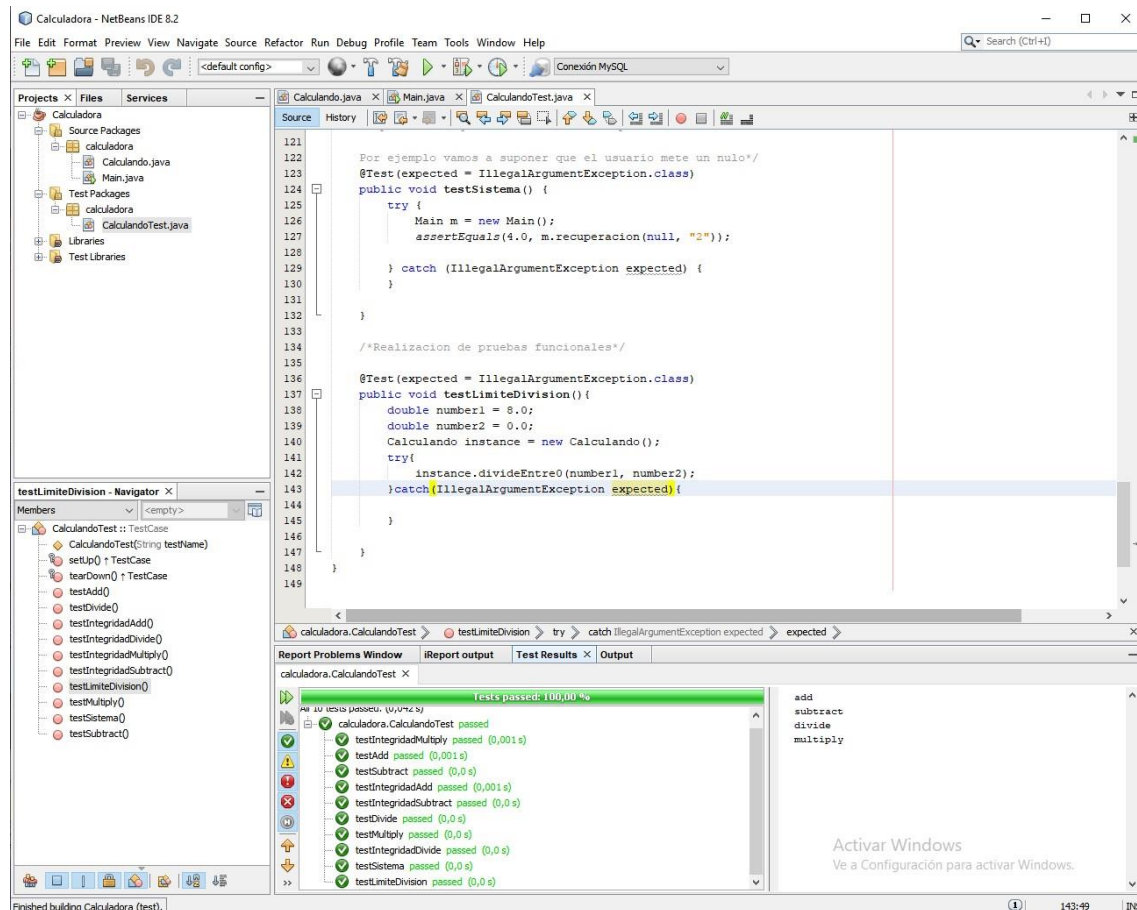
Planifica las restantes pruebas, estableciendo qué parámetros se van a analizar.

Una de las pruebas más utilizadas son las funcionales. La mayoría de los tests que nos encontremos irán orientados a esta parte. Con el tema de una calculadora no tenemos que acotar mucho los resultados pero en otros casos habría que controlar también la entrada de números negativos o de una cierta longitud.

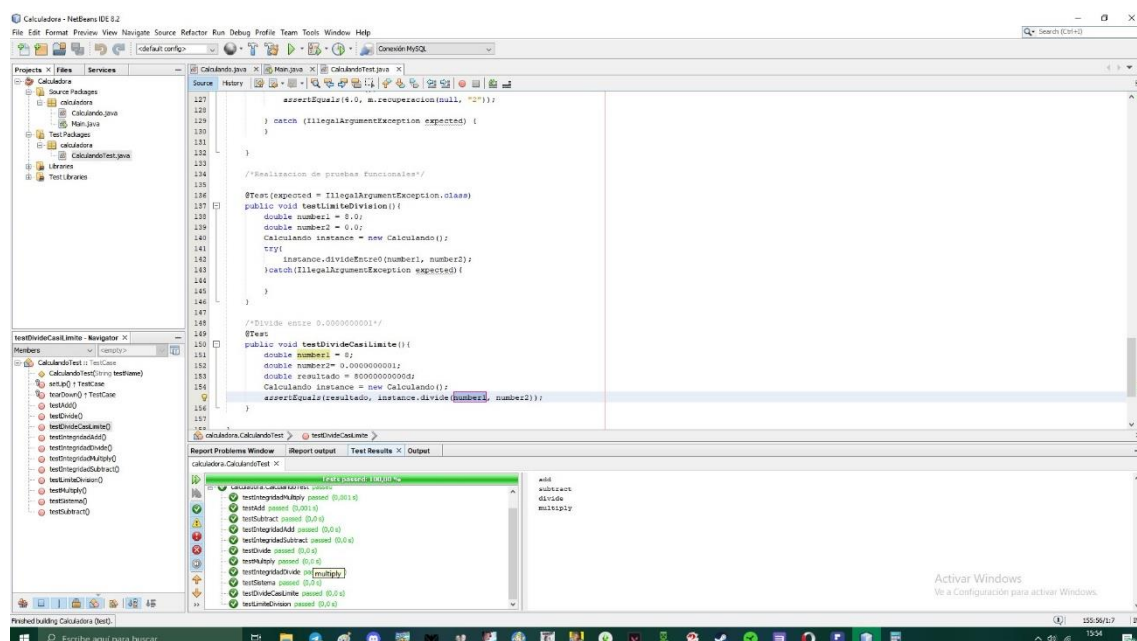
Para estos he realizado sobre todo con valores límites como el caso de la división entre cero donde la calculadora no debería dejarnos hacer nada y debiera saltar una excepción. Para ello creé en la clase Calculando un nuevo método llamado divideEntre0 el cual lanzará una excepción si el segundo número que le pasamos -suponiendo que es el divisor- sea un 0. Si se da el caso lanzará un `IllegalArgumentException`. En los tests deberemos esperar la excepción y recogerla.



## Tarea 7 DDI – Irene Caldelas Fernández

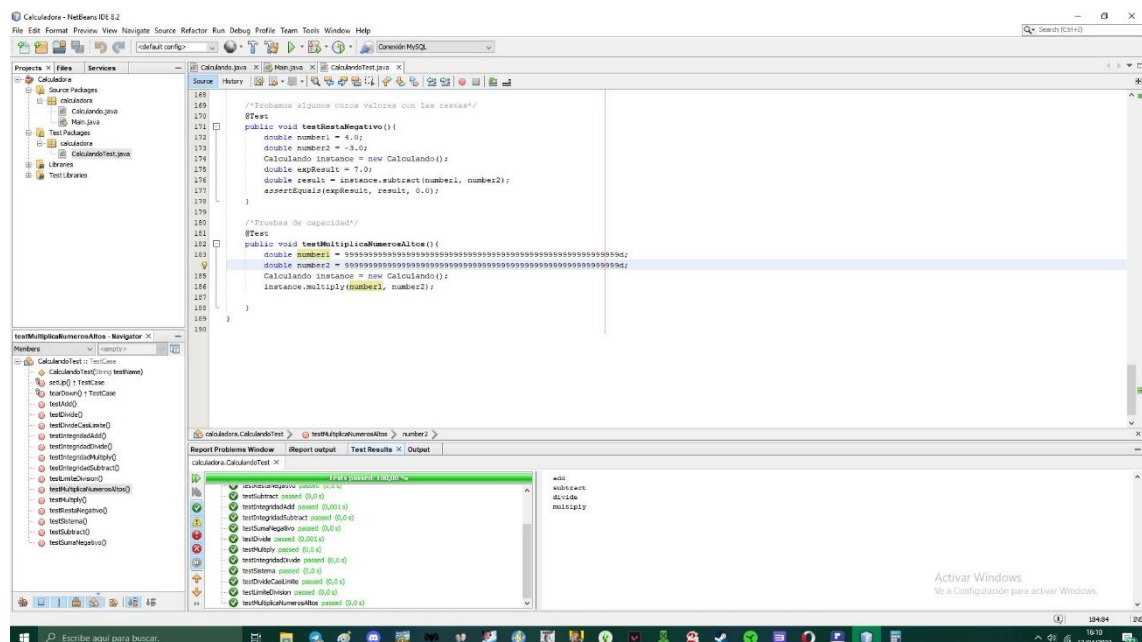
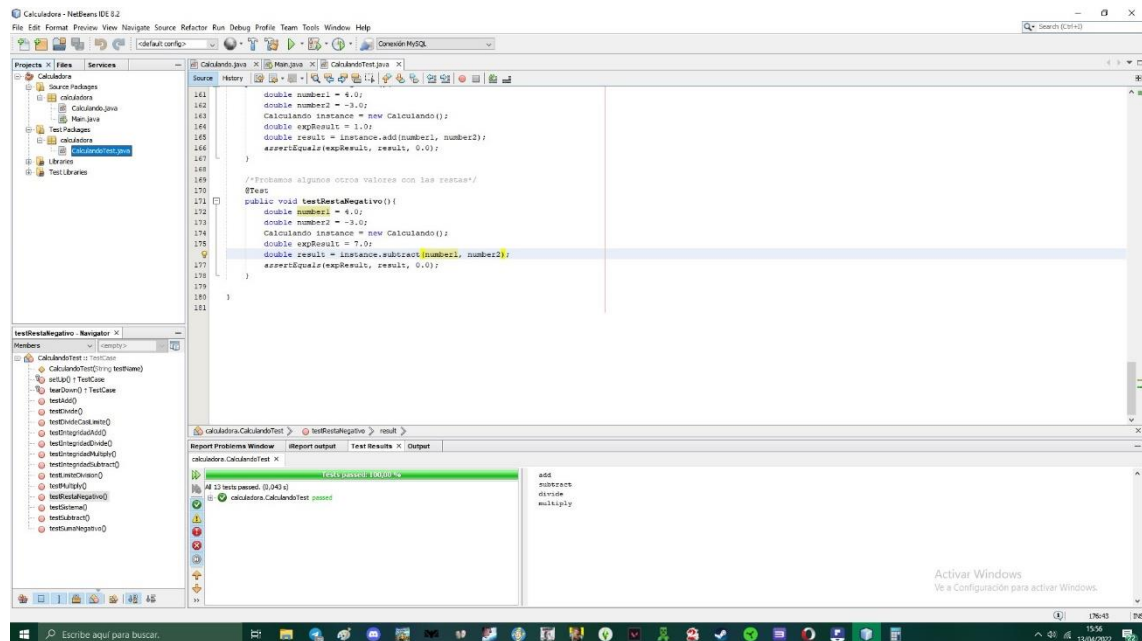


Ahora vamos a hacer una división entre un número muy pequeño que no es 0 por lo que debería entrar dentro del rango de los números permitidos. Para saber que resultado debería esperar me he ayudado de una calculadora externa.



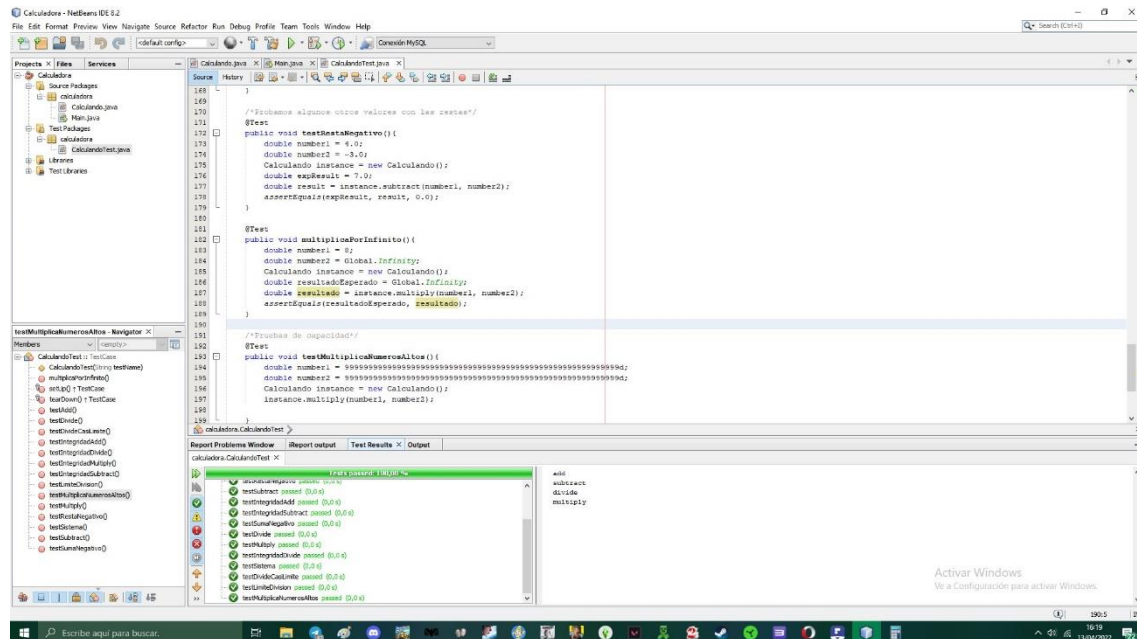
## Tarea 7 DDI – Irene Caldelas Fernández

También he realizado pruebas de sumar un numero positivo con un número negativo y restar a un numero positivo un número negativo para ver si realizaba bien este tipo de operaciones.



Otro tipo de pruebas que nos encontramos son las de capacidad donde le he proporcionado a una multiplicación unos valores anormalmente grandes. Aún a pesar del tamaño de estas entradas lo ha soportado sin problemas.

## Tarea 7 DDI – Irene Caldelas Fernández



También podríamos encontrarnos con pruebas de rendimiento que determinarían el tiempo de respuesta. Podríamos sacar esta prueba analizando la anterior y viendo a ver como es el tiempo de respuesta ante unas condiciones extremas.

Otra de las pruebas que podemos encontrar son las del uso de recursos donde veríamos si el programa logra un uso eficiente de recursos.