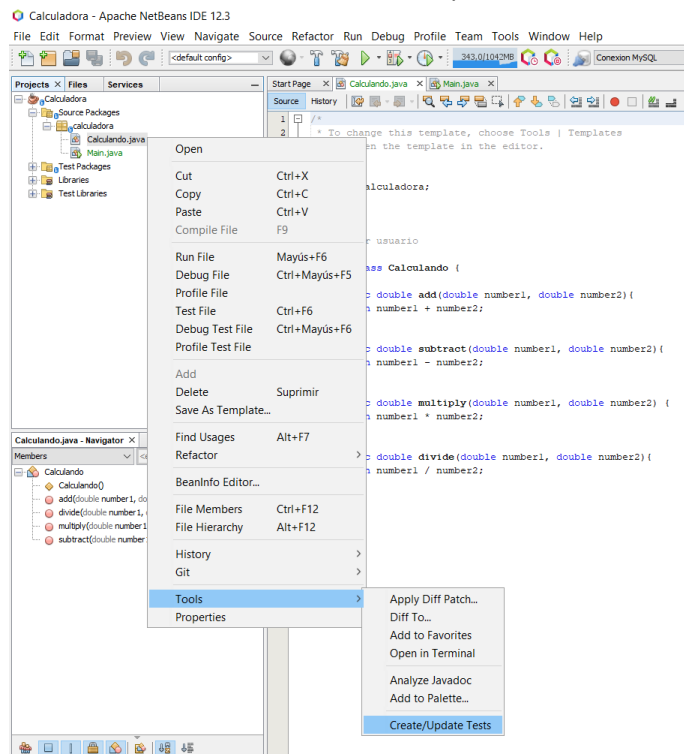
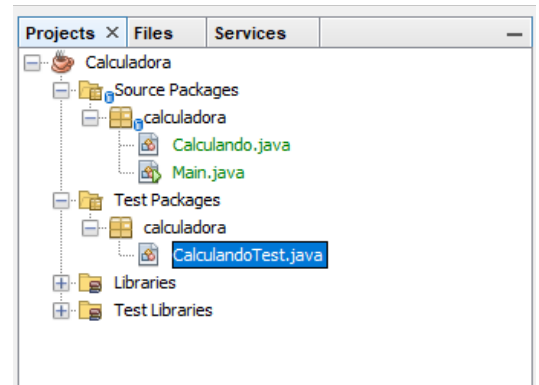
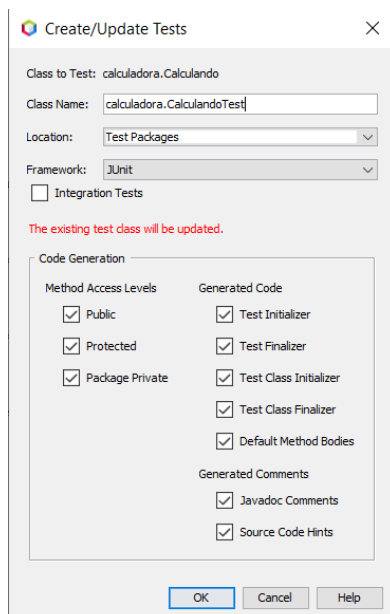


## 1. PRUEBAS UNITARIAS DE LA CLASE CALCULANDO:

Cargamos el fichero, que viene en la tarea, dentro de NetBeans pinchamos con el botón derecho encima de la clase calculando y seleccionamos “herramientas/crea-update test”:



Clickamos todas las opciones y damos ok, lo que va a generar la nueva clase dentro del paquete de test:



---

### CÓDIGO GENERADO PARA LA CLASE CALCULANDOTEST.JAVA

---

```
public class CalculandoTest extends TestCase {

    public CalculandoTest(String testName) {
        super(testName);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
    }

    @Override
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    public void testAdd() {
        System.out.println("add");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.add(number1, number2);
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

    public void testSubtract() {
        System.out.println("subtract");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.subtract(number1, number2);
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

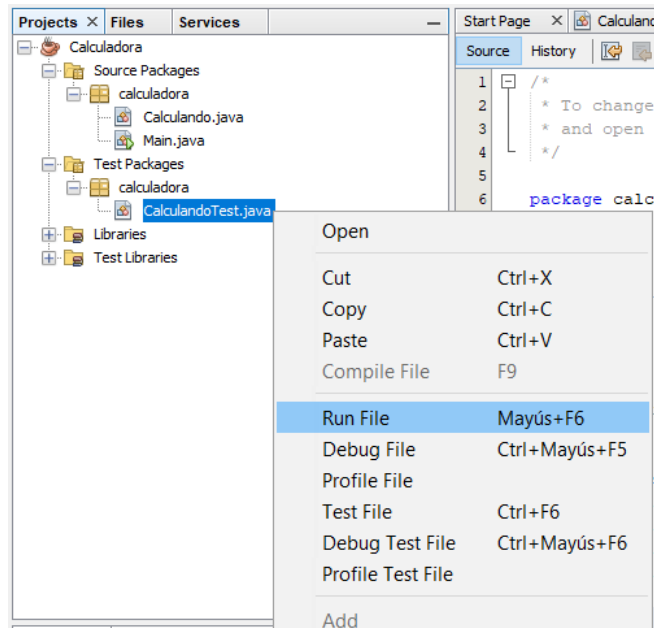
    public void testMultiply() {
        System.out.println("multiply");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.multiply(number1, number2);
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

    public void testDivide() {
        System.out.println("divide");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.divide(number1, number2);
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

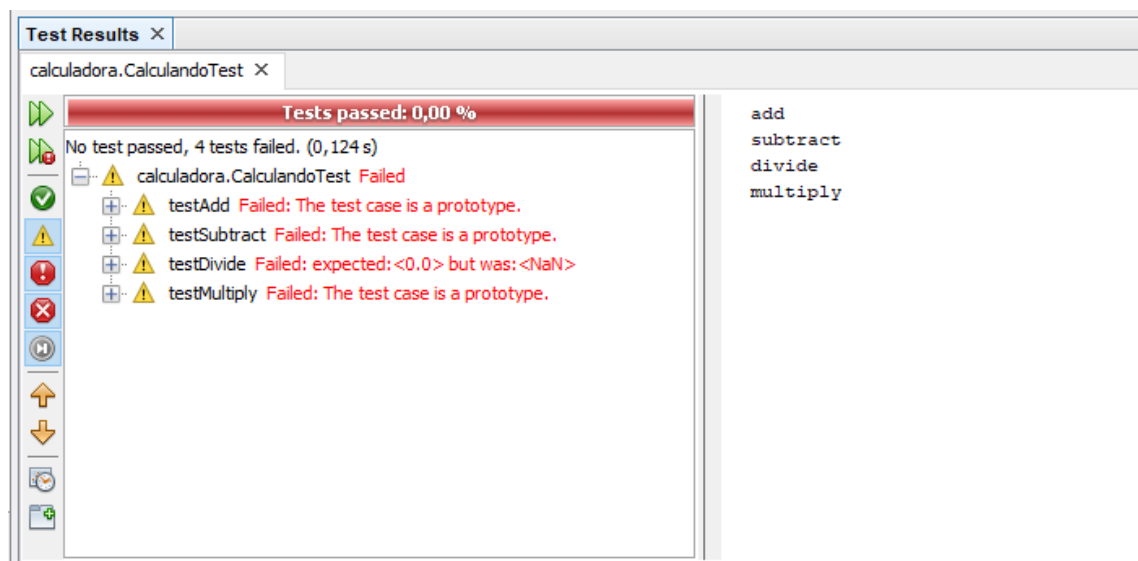
}
```

## 2. EJECUTAMOS LA CLASE DE PRUEBAS

Ponemos el puntero encima de la clase, pulsamos botón derecho y le decimos que queremos ejecutar:



El resultado de la ejecución nos dará fallo en todos los módulos ya que no tienen valores para ser evaluados:



## 3. ELIMINAMOS LAS 2 ÚLTIMAS LÍNEAS DE CADA MÉTODO

Vamos a eliminar las líneas dentro de todos los módulos que contienen lo siguiente:

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

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

---

### CÓDIGO RESULTANTE DESPUÉS DE ELIMINAR LAS 2 ÚLTIMAS LÍNEAS

---

```
public class CalculandoTest extends TestCase {

    public CalculandoTest(String testName) {
        super(testName);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
    }

    @Override
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    public void testAdd() {
        System.out.println("add");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.add(number1, number2);
        assertEquals(expResult, result, 0.0);
    }

    public void testSubtract() {
        System.out.println("subtract");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.subtract(number1, number2);
        assertEquals(expResult, result, 0.0);
    }

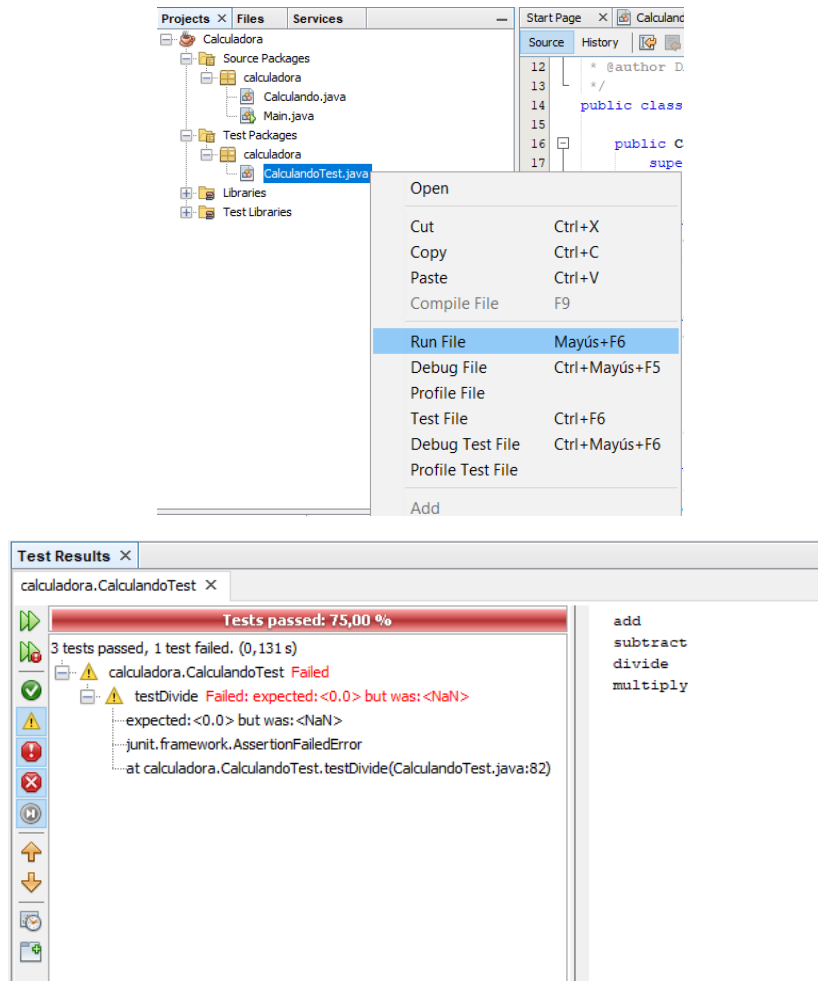
    public void testMultiply() {
        System.out.println("multiply");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.multiply(number1, number2);
        assertEquals(expResult, result, 0.0);
    }

    public void testDivide() {
        System.out.println("divide");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.divide(number1, number2);
        assertEquals(expResult, result, 0.0);
    }
}
```

---

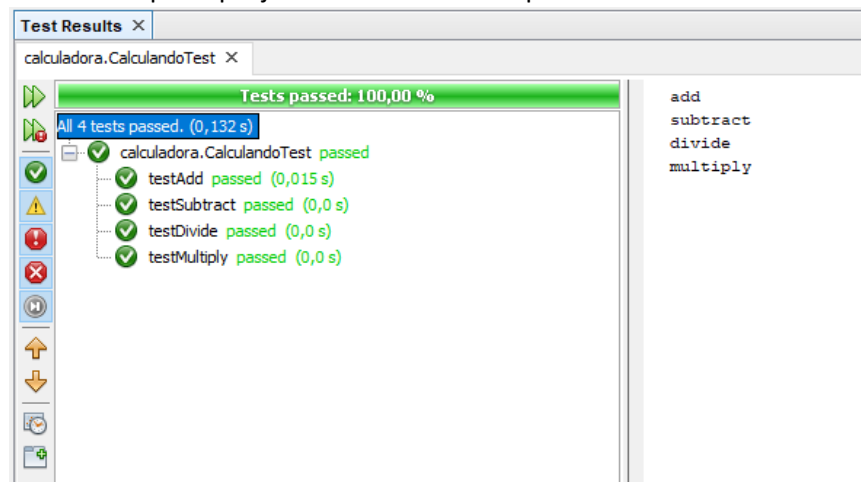
#### 4. CÓDIGO CORREGIDO LIBRE DE ERRORES

Si ejecutamos de nuevo el test seguirá dando errores:



*Pasa el test correctamente el 75% de los módulos, es decir 3 de los 4 módulos funcionan bien y falla el módulo de la división porque divide entre 0*

Modificamos el código para que todos los módulos pasen el test correctamente, lo que vamos a hacer es asignar valores a las variables para que junit al realizar el test pueda evaluar los resultados:



---

### CÓDIGO MODIFICADO PARA QUE TODOS LOS MÓDULOS PASEN LOS TEST

---

```
public class CalculandoTest extends TestCase {

    public CalculandoTest(String testName) {
        super(testName);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
    }

    @Override
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    public void testAdd() {
        System.out.println("add");
        double number1 = 4.0;
        double number2 = 3.0;
        Calculando instance = new Calculando();
        double expectedResult = 7.0;
        double result = instance.add(number1, number2);
        assertEquals(expResult, result, 7.0);
    }

    public void testSubtract() {
        System.out.println("subtract");
        double number1 = 4.0;
        double number2 = 3.0;
        Calculando instance = new Calculando();
        double expectedResult = 1.0;
        double result = instance.subtract(number1, number2);
        assertEquals(expResult, result, 1.0);
    }

    public void testMultiply() {
        System.out.println("multiply");
        double number1 = 4.0;
        double number2 = 3.0;
        Calculando instance = new Calculando();
        double expectedResult = 12.0;
        double result = instance.multiply(number1, number2);
        assertEquals(expResult, result, 12.0);
    }

    public void testDivide() {
        System.out.println("divide");
        double number1 = 9.0;
        double number2 = 3.0;
        Calculando instance = new Calculando();
        double expectedResult = 3.0;
        double result = instance.divide(number1, number2);
        assertEquals(expResult, result, 3.0);
    }
}
```

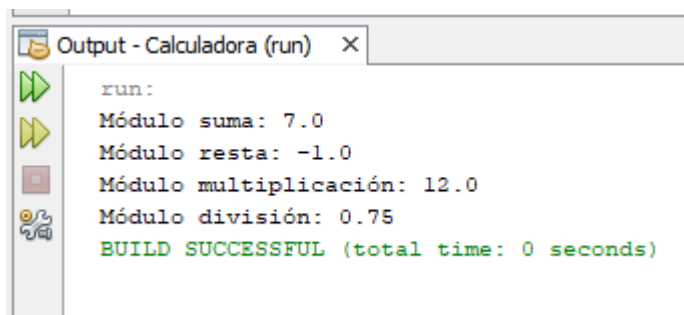
---

## 5. PLANIFICACIÓN DE PRUEBAS DE INTEGRACIÓN, SISTEMA Y REGRESIÓN

### • Pruebas de integración

Las pruebas de integración suponen comprobar que todos los módulos creados funcionen además de manera unitaria de manera conjunta entre unos y otros. Para ello vamos a instanciar la clase calculando desde la clase main, para a continuación llamar a cada módulo, cuyo resultado será mostrado por pantalla:

```
public class Main {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Calculando operacion = new Calculando();
        System.out.println("Módulo suma: " + operacion.add(3, 4));
        System.out.println("Módulo resta: " + operacion.subtract(3, 4));
        System.out.println("Módulo multiplicación: " + operacion.multiply(3, 4));
        System.out.println("Módulo división: " + operacion.divide(3, 4));
    }
}
```



```
Output - Calculadora (run) X
run:
Módulo suma: 7.0
Módulo resta: -1.0
Módulo multiplicación: 12.0
Módulo división: 0.75
BUILD SUCCESSFUL (total time: 0 seconds)
```

*Los resultados mostrados son correctos lo que indica que todos funcionan*

### • Pruebas de sistema

En esta fase de pruebas se comprueban los requisitos no funcionales:

- Seguridad: Para que la aplicación fuera segura tendríamos que contemplar en el código todas las excepciones que se pudieran producir durante la ejecución. Por ejemplo, tenemos que asegurarnos mediante un bloque try-catch que la entrada sea del tipo double y no rebase el máximo, también tenemos que tener en cuenta que los resultados no provoquen un rebasamiento de pila y que la división no sea por 0.
- Velocidad: Esta aplicación funciona muy rápido ya que tampoco tiene muchos módulos ni muchas operaciones que tenga que procesar por lo que no es relevante en las pruebas.
- Exactitud: Cuando modificamos el test de pruebas de la clase, cada módulo tiene una variable que contiene el resultado de las operaciones del mismo:

```
double expResult = 7.0;
double result = instance.add(number1, number2);
assertEquals(expResult, result, 7.0);
```

*Este es el resultado de la operación del módulo de suma (5+2=7)*

- **Fiabilidad:** Para esta prueba hemos sobrecargando de operaciones el método main para ver que la aplicación responde de manera correcta y da los resultados esperados:

```
for (int i = 0; i < 100; i++) {
    System.out.println("Módulo suma: " + operacion.add(3, 4));
    System.out.println("Módulo suma: " + operacion.add(3, 4));
    System.out.println("Módulo suma: " + operacion.add(3, 4));
    System.out.println("Módulo resta: " + operacion.subtract(3, 4));
    System.out.println("Módulo resta: " + operacion.subtract(3, 4));
    System.out.println("Módulo resta: " + operacion.subtract(3, 4));
    System.out.println("Módulo multiplicación: " + operacion.multiply(3, 4));
    System.out.println("Módulo multiplicación: " + operacion.multiply(3, 4));
    System.out.println("Módulo multiplicación: " + operacion.multiply(3, 4));
    System.out.println("Módulo división: " + operacion.divide(3, 4));
    System.out.println("Módulo división: " + operacion.divide(3, 4));
    System.out.println("Módulo división: " + operacion.divide(3, 4));
}
```

```
Output - Calculadora (run) X
Módulo multiplicación: 12.0
Módulo multiplicación: 12.0
Módulo multiplicación: 12.0
Módulo división: 0.75
Módulo división: 0.75
Módulo división: 0.75
Módulo suma: 7.0
Módulo suma: 7.0
Módulo suma: 7.0
Módulo resta: -1.0
Módulo resta: -1.0
Módulo resta: -1.0
Módulo multiplicación: 12.0
Módulo multiplicación: 12.0
Módulo multiplicación: 12.0
Módulo división: 0.75
Módulo división: 0.75
Módulo división: 0.75
Módulo suma: 7.0
Módulo suma: 7.0
Módulo suma: 7.0
Módulo resta: -1.0
Módulo resta: -1.0
Módulo resta: -1.0
Módulo multiplicación: 12.0
Módulo multiplicación: 12.0
Módulo multiplicación: 12.0
Módulo división: 0.75
Módulo división: 0.75
Módulo división: 0.75
Módulo suma: 7.0
Módulo suma: 7.0
Módulo suma: 7.0
Módulo resta: -1.0
Módulo resta: -1.0
Módulo resta: -1.0
Módulo multiplicación: 12.0
Módulo multiplicación: 12.0
Módulo multiplicación: 12.0
Módulo división: 0.75
Módulo división: 0.75
Módulo división: 0.75
Módulo suma: 7.0
Módulo suma: 7.0
Módulo suma: 7.0
Módulo resta: -1.0
Módulo resta: -1.0
Módulo resta: -1.0
Módulo multiplicación: 12.0
Módulo multiplicación: 12.0
Módulo multiplicación: 12.0
Módulo división: 0.75
Módulo división: 0.75
Módulo división: 0.75
BUILD SUCCESSFUL (total time: 0 seconds)
```

*En la salida vemos que los resultados son los esperados*



- **Pruebas de regresión**

Las pruebas de regresión suponen volver a pasar test unitarios y de integración cada vez que se realicen cambios en el código. A lo largo de esta tarea, antes de llegar a este punto se han hecho modificaciones en los distintos módulos y se han pasado a continuación pruebas de código y test lo que supone que ya hemos realizado varias pruebas de regresión.

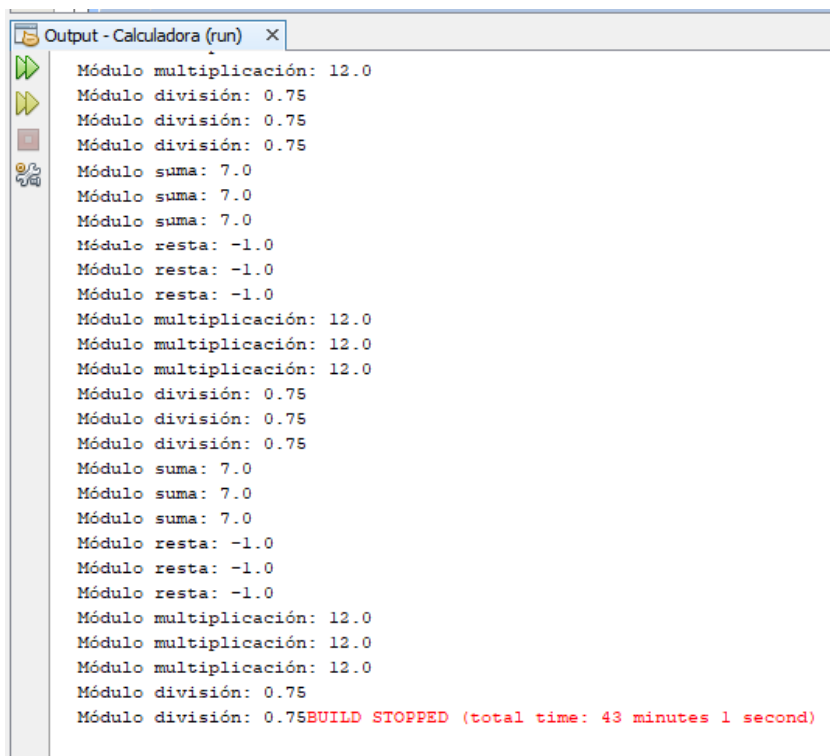
## 6. PLANIFICACIÓN DEL RESTO DE PRUEBAS

- **Pruebas de capacidad y rendimiento**

Ponemos a prueba la aplicación solicitando que realice el máximo número de operaciones que nos permita el bucle for mediante el valor límite del tipo entero (2147483647):

```
for (int i = 0; i < 2147483647; i++) {
    System.out.println("Módulo suma: " + operacion.add(3, 4));
    System.out.println("Módulo suma: " + operacion.add(3, 4));
    System.out.println("Módulo suma: " + operacion.add(3, 4));
    System.out.println("Módulo resta: " + operacion.subtract(3, 4));
    System.out.println("Módulo resta: " + operacion.subtract(3, 4));
    System.out.println("Módulo resta: " + operacion.subtract(3, 4));
    System.out.println("Módulo multiplicación: " + operacion.multiply(3, 4));
    System.out.println("Módulo multiplicación: " + operacion.multiply(3, 4));
    System.out.println("Módulo multiplicación: " + operacion.multiply(3, 4));
    System.out.println("Módulo división: " + operacion.divide(3, 4));
    System.out.println("Módulo división: " + operacion.divide(3, 4));
    System.out.println("Módulo división: " + operacion.divide(3, 4));
}
```

Ejecutamos para ver que la aplicación es capaz de realizar la sobrecarga de operaciones y comprobamos el tiempo en que las realiza:



*He parado la ejecución a los 43 minutos, pero la aplicación podría haber continuado y en ese tiempo no ha fallado lo que ya nos sirve como muestra de que la capacidad de la aplicación es grande*

- **Pruebas funcionales**

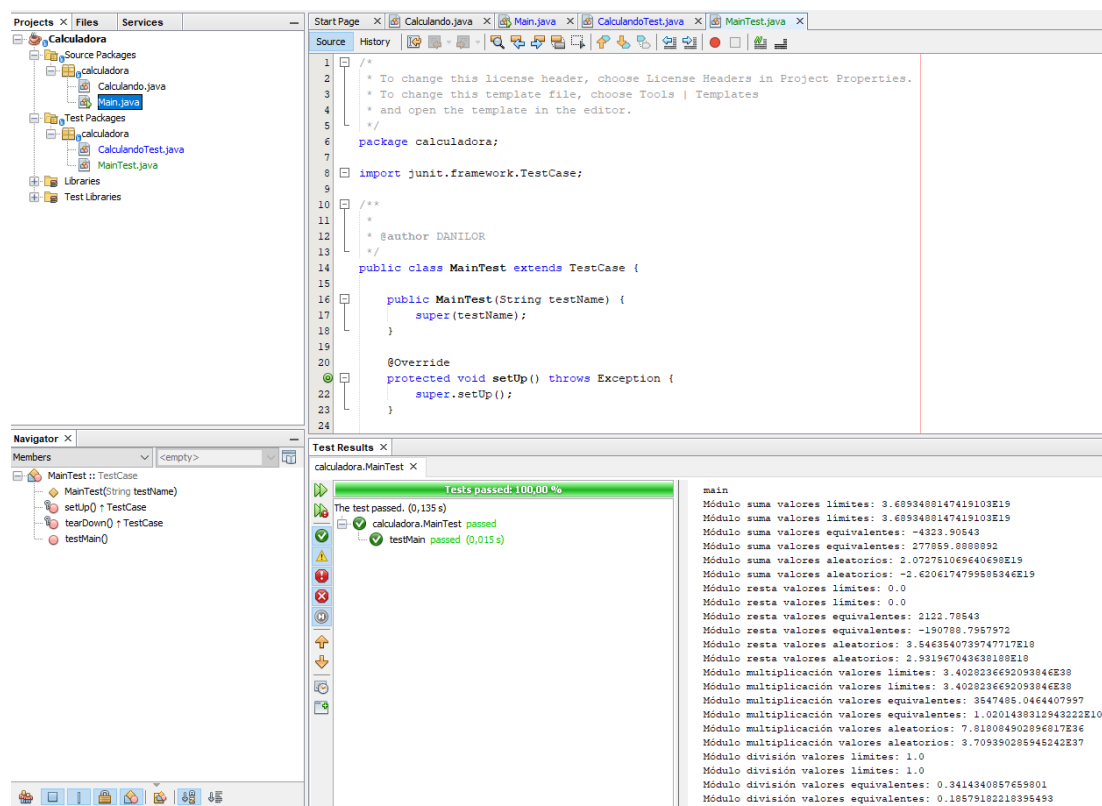
Estas pruebas validan si el resultado cumple con las especificaciones requeridas. La forma de hacerlas es la llamada caja negra por las que se meten en la entrada distintos valores límites, equivalentes y aleatorios y se comprueba que en la salida los resultados sean los esperados.

Para ello modificamos el código en el que vamos a incluir 2 valores límite por módulo, 2 valores equivalentes y 2 valores aleatorios, para posteriormente mostrarlos por pantalla:

### CÓDIGO MODIFICADO PARA REALIZAR LAS PRUEBAS FUNCIONALES

```
public static void main(String[] args) {
    Calculando operación = new Calculando();
    //MÓDULO DE SUMA
    //Valores límites
    System.out.println("Módulo suma valores límites: " + operación.add(Math.pow(2, 64), Math.pow(-2, 64)));
    System.out.println("Módulo suma valores límites: " + operación.add(Math.pow(-2, 64), Math.pow(2, 64)));
    //Valores equivalentes
    System.out.println("Módulo suma valores equivalentes: " + operación.add(-1100.56, -3223.34543));
    System.out.println("Módulo suma valores equivalentes: " + operación.add(43535.546546, 234324.3423432));
    //Valores aleatorios
    System.out.println("Módulo suma valores aleatorios: " + operación.add(Math.random()*Math.pow(2, 64),
Math.random()*Math.pow(2, 64)));
    System.out.println("Módulo suma valores aleatorios: " + operación.add((Math.random()*Math.pow(2, 64))*-1,
(Math.random()*Math.pow(2, 64))*-1));
    //MÓDULO DE RESTA
    //Valores límites
    System.out.println("Módulo resta valores límites: " + operación.subtract(Math.pow(2, 64), Math.pow(-2, 64)));
    System.out.println("Módulo resta valores límites: " + operación.subtract(Math.pow(-2, 64), Math.pow(2, 64)));
    //Valores equivalentes
    System.out.println("Módulo resta valores equivalentes: " + operación.subtract(-1100.56, -3223.34543));
    System.out.println("Módulo resta valores equivalentes: " + operación.subtract(43535.546546, 234324.3423432));
    //Valores aleatorios
    System.out.println("Módulo resta valores aleatorios: " + operación.subtract(Math.random()*Math.pow(2, 64),
Math.random()*Math.pow(2, 64)));
    System.out.println("Módulo resta valores aleatorios: " + operación.subtract((Math.random()*Math.pow(2, 64))*-1,
(Math.random()*Math.pow(2, 64))*-1));
    //MÓDULO DE MULTIPLICACIÓN
    //Valores límites
    System.out.println("Módulo multiplicación valores límites: " + operación.multiply(Math.pow(2, 64), Math.pow(-2,
64)));
    System.out.println("Módulo multiplicación valores límites: " + operación.multiply(Math.pow(-2, 64), Math.pow(2,
64)));
    //Valores equivalentes
    System.out.println("Módulo multiplicación valores equivalentes: " + operación.multiply(-1100.56, -3223.34543));
    System.out.println("Módulo multiplicación valores equivalentes: " + operación.multiply(43535.546546,
234324.3423432));
    //Valores aleatorios
    System.out.println("Módulo multiplicación valores aleatorios: " + operación.multiply(Math.random()*Math.pow(2, 64),
Math.random()*Math.pow(2, 64)));
    System.out.println("Módulo multiplicación valores aleatorios: " + operación.multiply((Math.random()*Math.pow(2,
64))*-1, (Math.random()*Math.pow(2, 64))*-1));
    //MÓDULO DE DIVISIÓN
    //Valores límites
    System.out.println("Módulo división valores límites: " + operación.divide(Math.pow(2, 64), Math.pow(-2, 64)));
    System.out.println("Módulo división valores límites: " + operación.divide(Math.pow(-2, 64), Math.pow(2, 64)));
    //Valores equivalentes
    System.out.println("Módulo división valores equivalentes: " + operación.divide(-1100.56, -3223.34543));
    System.out.println("Módulo división valores equivalentes: " + operación.divide(43535.546546, 234324.3423432));
    //Valores aleatorios
    System.out.println("Módulo división valores aleatorios: " + operación.divide(Math.random()*Math.pow(2, 64),
Math.random()*Math.pow(2, 64)));
    System.out.println("Módulo división valores aleatorios: " + operación.divide((Math.random()*Math.pow(2, 64))*-1,
(Math.random()*Math.pow(2, 64))*-1));
}
```

Una vez tenemos modificado el código de la clase main modificado, pinchamos encima vamos a herramientas y creamos un test, lo que nos va a generar una nueva clase “mainTest” que vamos a ejecutar mostrando los siguientes resultados:



El resultado de las pruebas es satisfactorio con los distintos tipos de valores que le hemos dado

### • Pruebas de seguridad

Esta aplicación no está configurada para que tenga seguridad, los métodos de los módulos no están encapsulados y son públicos por lo que cualquier persona puede tener acceso a todas las partes del programa con el peligro que conlleva al poder realizar cualquier modificación tanto a propósito como por error. Para evitar esto podemos modificar los métodos y hacerlos privados:

```
public class Calculando {

    protected double add(double number1, double number2){
        return number1 + number2;
    }

    protected double subtract(double number1, double number2){
        return number1 - number2;
    }

    protected double multiply(double number1, double number2) {
        return number1 * number2;
    }

    protected double divide(double number1, double number2){
        return number1 / number2;
    }
}
```

Solamente las clases contenidas en el paquete tendrán acceso a dichos métodos

Otra forma de hacer más segura la aplicación será crear un fichero .jar, firmarlo y distribuirlo a los usuarios junto con el certificado para que solamente los usuarios que tengan autorización hagan uso de la aplicación, dando a la misma más seguridad ya que el atacante tendría que descifrar la clave para poder usar la aplicación.

- **De uso de recursos**

Trata de la eficiencia en el uso de los recursos de hardware del dispositivo o dispositivos concretos donde se va a ejecutar la aplicación. Hoy en día la mayoría de estos dispositivos tienen altas capacidades de memoria RAM y espacio en disco siendo estos los recursos más necesarios para la aplicación, por lo que no es un requisito imprescindible, pero a medida que la aplicación se hace más grande conviene optimizarla para que el uso de recursos no se dispare y sobre todo para que el tiempo de procesamiento de las operaciones no se vea incrementado, suponiendo así una pérdida de eficiencia. Para ello tenemos que tener una aplicación muy modular, en la que vamos a reutilizar partes e incluso variables para no tener partes de código duplicadas. También es conveniente hacer buen uso de la API que nos proporciona JAVA que ya contiene clases eficientes.

## 7. DOCUMENTACIÓN DE RESULTADOS

La documentación de las pruebas realizadas que se nos solicita en este apartado es la que he ido haciendo en cada uno de los puntos de la planificación anteriores. Cada cosa que se ha propuesto en la planificación de las pruebas ha sido probada y documentada con el uso de pantallazos también. Considero que este punto ya está realizado de una manera más ordenada en cada uno de los casos.