



IES VALLE DEL JERTE PLASENCIA

TERCERA PRÁCTICA UNIDAD 3 1ºDAM

PRUEBAS UNITARIAS: JUNIT

Abel López Ruiz

Editado por



30 de diciembre de 2024

ÍNDICE GENERAL

1. Introducción	2
1.1. Pruebas Unitarias	2
1.2. Herramientas para pruebas unitarias	3
2. JUnit en Netbeans	6
2.1. Primer contacto	6
2.2. Primeras pruebas	11
3. Tarea propuesta JUnit	17
3.1. Primer ejercicio	17
3.2. Segundo ejercicio	26
4. Herramientas de construcción en Java	34
5. Entrega	35
6. Bibliografía	36

1. Introducción

1.1. Pruebas Unitarias

Como vimos en apartados anteriores, primeramente se realiza un análisis del proyecto que nos trae el cliente, se hacen esquemas, reuniones, y una síntesis de la estructura de lo que se va a hacer. Luego llega el diseño donde entra en juego el papel de los desarrolladores o programadores, que son los encargados de crear los primeros códigos fuentes. Pero claro, estos códigos fuentes hay que probarlos, y hacerles Test para comprobar que funciona como se espera que funcione o que trabaje como deseemos.

Después de la fase de diseño, o lo que es la fase de implementar o pasar lo requerido a código fuente, una vez hecho, no se debe presuponer que hace lo que queremos debemos hacer pruebas, para comprobar que lo que obtenemos con el código es lo que esperamos. Para ello hay muchas librerías de pruebas que nos ayudarán a realizarlas. Se pueden hacer pruebas manuales sacando por pantalla los resultados y comprobando, pero son muy tediosas, para ello existen ya muchas herramientas para ello, y con ellas realizaremos los tests fácilmente.

Respecto a las pruebas hay de muchos tipos, existen muchas clasificaciones de ellas. Una de las clasificaciones son las pruebas de caja negra, en la que no nos preocupamos por el funcionamiento del interior, se hacen las pruebas a la caja sin tener en cuenta el interior. Además de estas las pruebas de caja blanca, que si hacemos pruebas de lo que hay en el interior de la caja, es decir, los programas internos.

Además de estas pruebas se pueden clasificar en pruebas unitarias que se trata de hacer pruebas a una determinada función, clase o método, es decir, solo a una parte del código que nos interese. En este tipo de prueba centraremos esta práctica. Luego las pruebas de integración que se trata de las pruebas del código ya completo, una vez integrado todo en el código completo, estas pruebas se hacen después de realizadas las unitarias.

También se pueden realizar pruebas de seguridad para comprobar cuánto de seguro o protegido está nuestra aplicación. Otras pruebas son las de volumen o carga, para asegurarnos cuanta capacidad de volumen o cuanto es capaz de soportar nuestra aplicación en términos de memoria, o accesos. Otras pruebas muy concretas son las de regresión, cuando se cambia algo en nuestro código hay que volver a hacer las pruebas para comprobar si con este cambio sigue funcionando bien.

También se pueden clasificar las pruebas en manuales y automáticas.

1.2. Herramientas para pruebas unitarias

Hay muchas herramientas para hacer pruebas de forma automática, Mockito, bug-zille, y JUnit son las más conocidas, nosotros en esta práctica vamos a trabajar con la herramienta JUnit.



Figura 1.1: Icono JUnit.

JUnit como se muestra en la imagen va ya por la versión 5, aunque al principio presentó algunos fallos o bug en eclipse y Netbeans, también podéis trabajar con las versiones 4.x aunque cabe destacar que sí que hay diferencias significativas, de todas formas para nuestra práctica podéis trabajar con las versiones 4.x perfectamente.

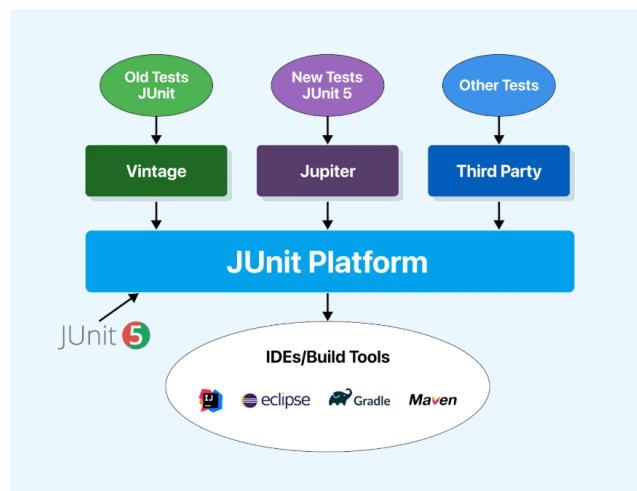


Figura 1.2: Arquitectura JUnit.

Como se observa en la imagen anterior que es la arquitectura, la diferencias de JUnit 4 con respecto a la 5 son muchas, una de ellas es la gran cantidad de anotaciones que incluye, por ejemplo las anotaciones `@After` y `@Before` son sustituidas por `@BeforeEach`, y `@AfterEach`, y muchas otras anotaciones nuevas que incluye.

Otra diferencia es los tests de excepciones en JUnit 5 que es diferentes a JUnit 4, además de otros tipos de asertos o aserciones que incluyen nuevos. Las aserciones serán los métodos que vamos a utilizar en JUnit para hacer las pruebas.

Features	JUnit 4	JUnit 5
Architecture	Single jar file containing all components.	Composed of three subcomponents: JUnit Platform, JUnit Jupiter, and JUnit Vintage.
Required JDK Version	Java 5 or higher	Java 8 or higher.
Assertions	<i>org.junit.Assert</i> with assert methods.	<i>org.junit.jupiter.Assertions</i> with enhanced assert methods, including <i>assertThrows()</i> and <i>assertAll()</i> .
Assumptions	<i>org.junit.Assume</i> with various methods	<i>org.junit.jupiter.api.Assumptions</i> with a reduced set of methods.
Tagging and Filtering	<i>@category</i> annotation	<i>@tag</i> annotation.
Test Suites	<i>@RunWith</i> and <i>@Suite</i> annotation	<i>@Suite</i> , <i>@SelectPackages</i> , and <i>@SelectClasses</i> annotations.
Non-public Test Methods	It must be public.	It can be package-protected, with no requirement for a public no-args constructor.
3rd Party Integration	Lacks dedicated support for third-party plugins.	The JUnit platform project facilitates third-party integration, defining the TestEngine API for testing frameworks.

Figura 1.3: Diferencias JUnit4 y JUnit5.

En la tabla anterior se puede ver un resumen de las diferencias de JUnit4 y JUnit5. Una anotación novedosa que permite JUnit5 es `@RepeatedTest` que repite los test o realiza bucles de pruebas.

La importancia de los tests unitarios es la detección de bugs, mejorar la calidad del código y quitar cosas repetidas o redundantes y además podemos generar documentación sobre las pruebas realizadas.

Respecto a las aserciones o asertos, JUnit5 incluye todas las de JUnit4 y además incluye otras nuevas.

- **assertEquals():** Ya viene en JUnit4, es el más utilizado y el que más utilizaremos, comprueba si son iguales, el valor esperado con el obtenido. Además se puede agregar un tercer parámetro que es un mensaje.
- **assertArrayEquals():** Comprueba que dos arrays son iguales, y además como el anterior admite un tercer parámetro que es un mensaje (String).
- **assertNull():** Comprueba que un objeto sea null, o apunte a null. Puede admitir un segundo parámetro que es un mensaje.
- **assertNotNull() :** Lo mismo que el anterior pero comprueba que el objeto que se mete por parámetro no es null.

- **assertSame()**: Comprueba que dos objetos son iguales y admite un tercer parámetro como mensaje.
- **assertNotSame()**: Lo mismo que el anterior pero comprueba que dos objetos son diferentes.
- **assertTrue()**: Comprueba que la condición es verdadera, además admite segundo parámetro como mensaje de tipo String.
- **assertFalse()**: Lo contrario que la anterior, comprueba que la condición es falsa.
- **assertThat()**: Es muy versátil e incluye objetos Matcher, pero no la veremos.
- **fail()** : Falla intencionadamente, es un aserto de prueba.

Los anteriores eran asertos de JUnit4, pero los siguientes que no detallaremos son más complejos están incorporados en JUnit5 pero dejamos como tarea al alumno que investigue sobre ellos: `assertIterableEquals()`, `assertLinesMatch()`, `assertThrows()` (para excepciones), `assertTimeout()` (para comprobar si tarda más o menos tiempo el método).

Dejamos como tarea al alumno que es la herramienta Selenium, para que se usa y que relación tiene con JUnit. También que investiguen qué es un test suite y que anotación se necesita

Otra herramienta interesante que se le encarga al alumno investigar es cucumber y qué tiene que ver con las bases de datos.

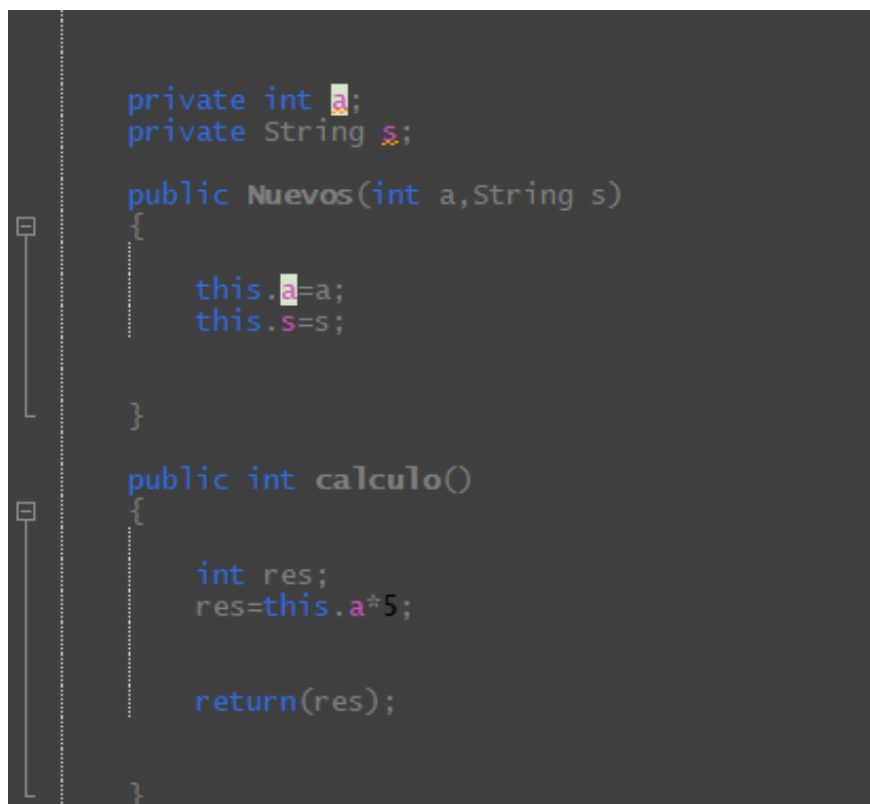
También se pide investigar la herramienta TestNG y para qué se utiliza.

2. JUnit en Netbeans

2.1. Primer contacto

Vamos a trabajar en JUnit con el IDE Apache Netbeans pero se le deja al alumno como tarea, como se trabajaría con JUnit en Eclipse. Primeramente se va a crear un proyecto cualquiera en Netbeans, aunque JUnit funciona mejor con proyectos tipo Gradler o Maven, pero no importa para las pruebas que vamos a hacer se puede crear cualquiera de ellos.

Se va a crear una clase sencilla con un constructor, un método y dos atributos como se muestra en la imagen:



```
private int a;
private String s;

public Nuevos(int a,String s)
{
    this.a=a;
    this.s=s;
}

public int calculo()
{
    int res;
    res=this.a*5;

    return(res);
}
```

Figura 2.1: Proyecto nuevo.

Luego le vamos a dar al archivo de la clase con click derecho y en tools, vamos a la opción de Test. Lo mostramos a continuación:

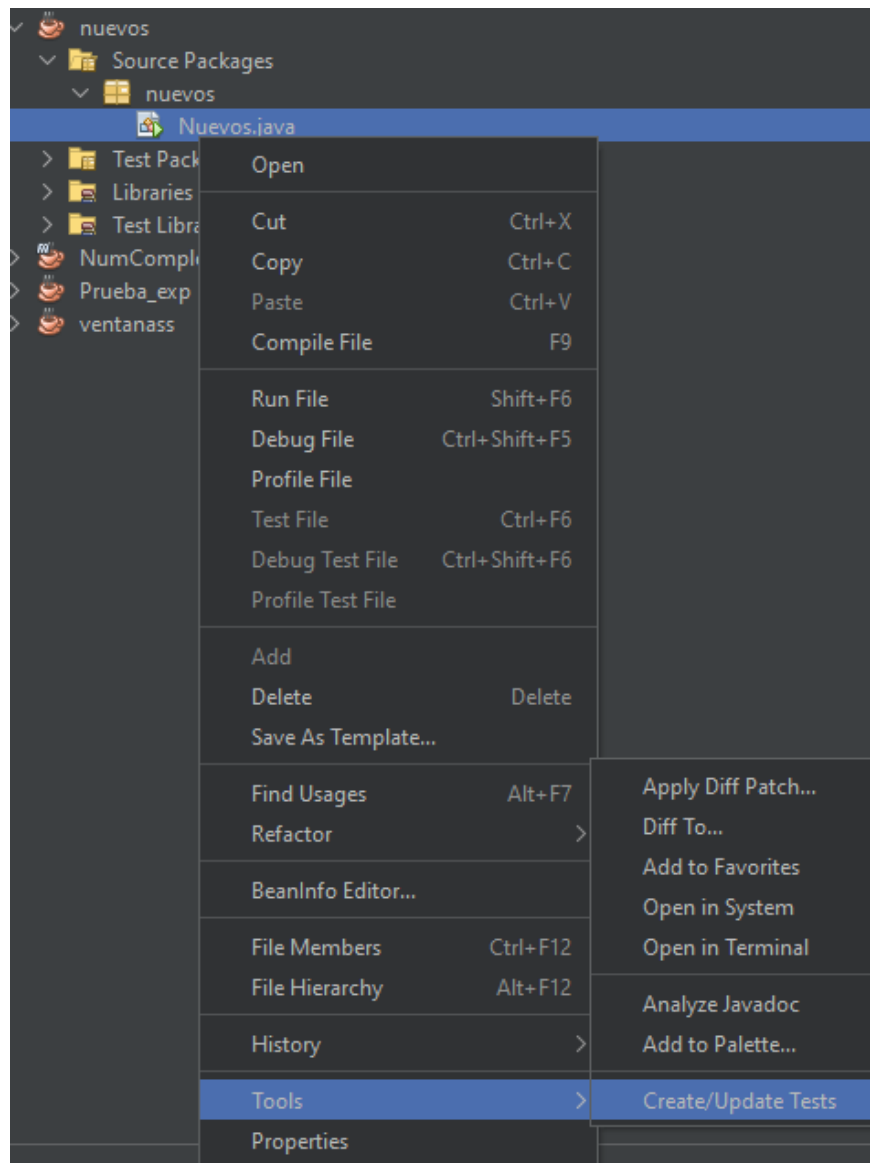


Figura 2.2: Pruebas.

Una vez pulsado nos aparecerá un menú donde podemos cambiar la versión de JUNIT y podemos marcar algunas opciones o no.

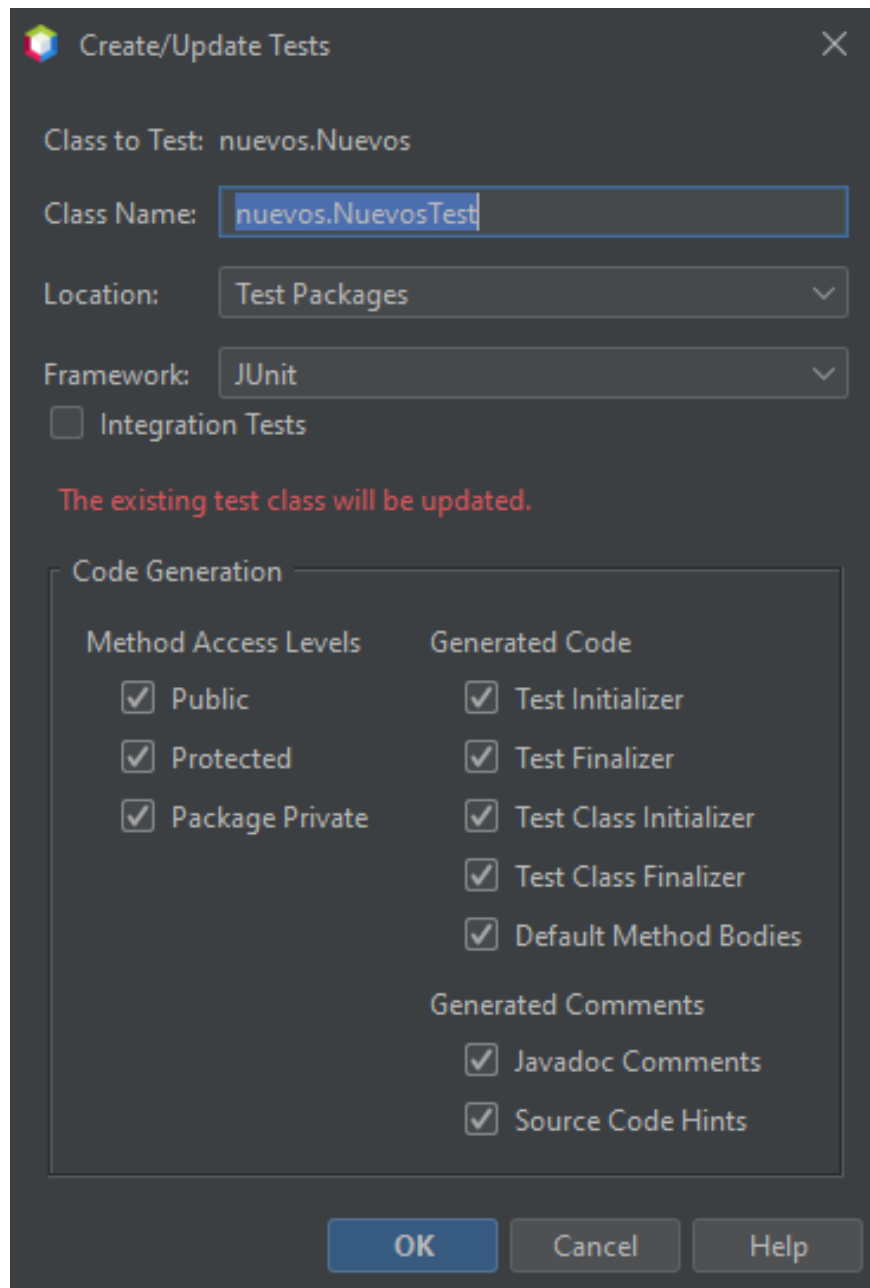


Figura 2.3: Pruebas menú.

Una vez pulsado en la ventana de proyecto dentro de la carpeta Test, aparece un nuevo paquete con una nueva clase que se llama `nuevosTest.java`. es decir el nombre de proyecto con la palabra Test al final, lo mostramos en la imagen siguiente:

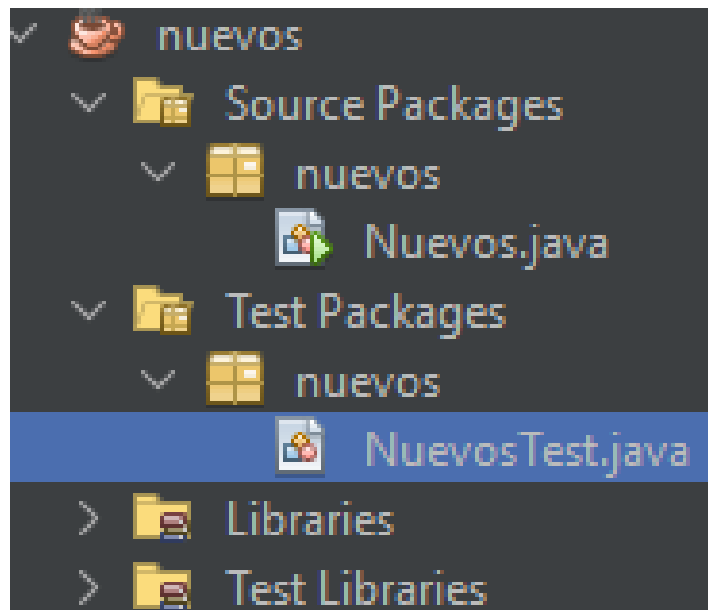


Figura 2.4: Carpeta pruebas.

Pero es más,, si nos fijamos en la carpeta de Test Libraries podemos ver que se han agregado librerías en esa carpeta de JUnit.

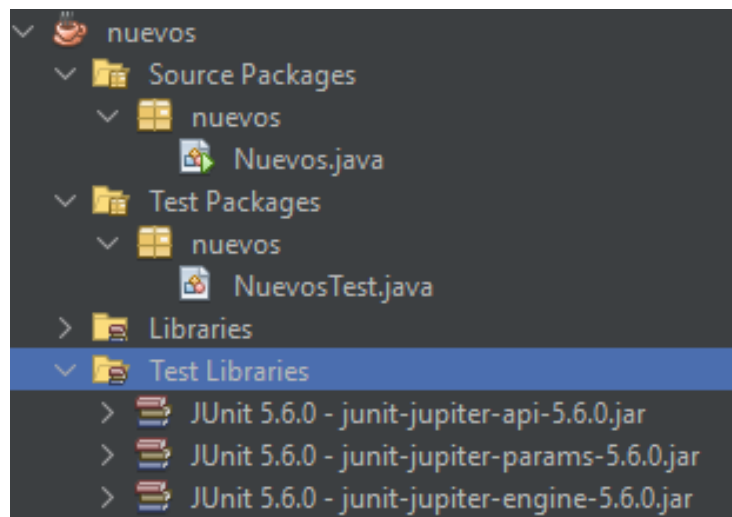


Figura 2.5: Librerías JUNIT.

En el código me aparece lo siguiente que vamos a explicar a continuación en las siguientes páginas.

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

/**
 *
 * @author Abel
 */
public class NuevosTest {

    public NuevosTest() {

    }

    @BeforeAll
    public static void setUpClass() {

    }

    @AfterAll
    public static void tearDownClass() {

    }

    @BeforeEach
    public void setUp() {

    }

    @AfterEach
    public void tearDown() {

    }

    /**
     * Test of main method, of class Nuevos.
     */
    @Test
    public void testMain() {
        System.out.println(x: "main");
        String[] args = null;
        Nuevos.main(args);
        // TODO review the generated test code and remove
        fail(message: "The test case is a prototype.");
    }
}
```

Figura 2.6: Código JUnit.

El primer método es el constructor de la clase NuevosTest, no lo utilizaremos, luego los dos métodos setUpClass() y tearDownClass() con anotaciones @BeforeAll y @AfterAll se van a ejecutar antes de **todas** las pruebas y la otra después de realizada las pruebas, es decir el código que pongamos ahí se ejecuta antes y después pero de todas las pruebas. Es decir si se realizan 3 pruebas al final de las 3 pruebas se ejecuta tearDownClass().

Luego los métodos setUp() y tearDown(), uno se ejecuta al principio de cada prueba o test y el segundo al final que sirve para liberar los recursos asignados en el método setUp. Por ejemplo si creo dos objetos en setUp, luego en tearDown los libero igualando los objetos a null. Llevan las anotaciones @BeforeEach, y @AfterEach, que en JUnit4 era @Before y @After.

El método fails es un aserto de prueba que ya se comentó algo y siempre dará fallo, es decir, que el test sale mal, lo tenemos que quitar cuando vayamos a hacer las pruebas. Podemos hacer pruebas de cada método poniendo siempre test y seguido nombre del método de mi clase a la que le hago la prueba con la primera letra en mayúscula como el caso de testMain, veamos un ejemplo con el método calculo:

```
@Test
public int testCalculo()
{

    return 0;
}
```

Figura 2.7: Código JUnit Testcase.

En las páginas siguientes vamos a hacer una primera prueba para que se aprenda a realizar las primeras pruebas con JUnit.

2.2. Primeras pruebas

Para realizar las primeras pruebas pinchamos click derecho en el archivo .java creado de Test y pinchamos como viene en la imagen, en test file.

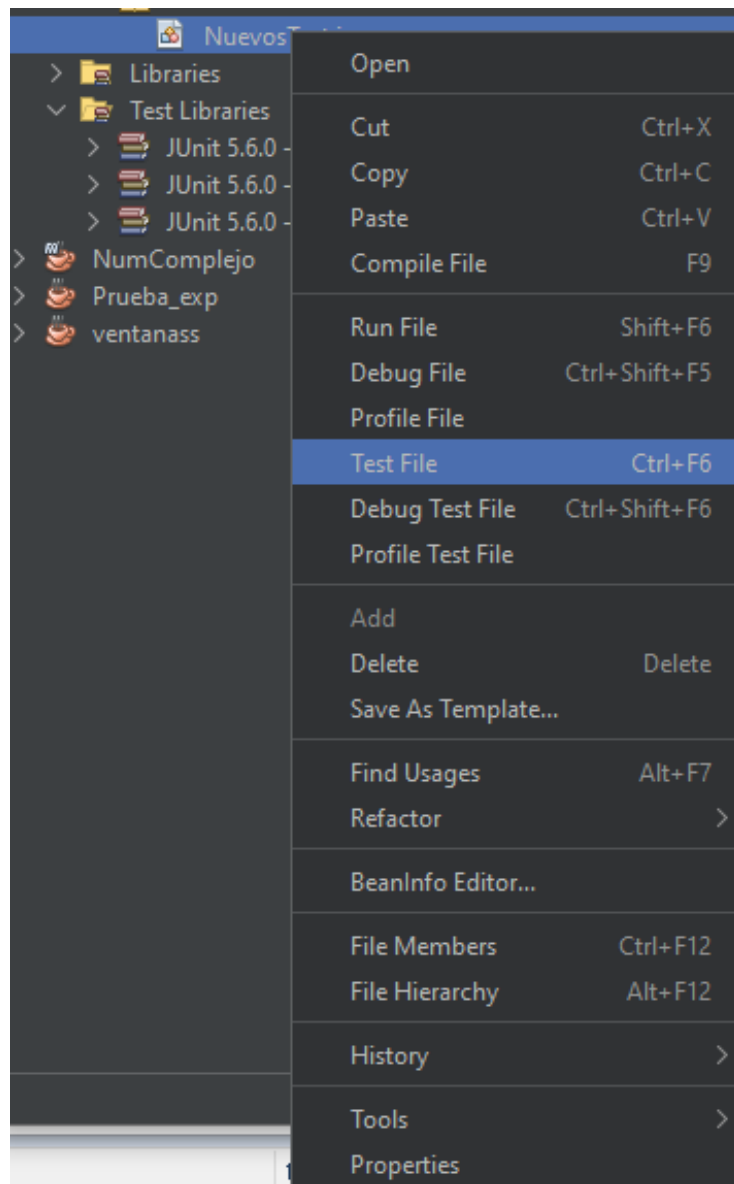


Figura 2.8: JUnit Test File.

Una vez le damos, el test no resultará bien porque no quité fails y nos aparecerá en rojo la barra cuando el test no da correcto. Lo vemos en la siguiente imagen. (Recomendamos hacer el proyecto en Maven que es más compatible con JUnit).

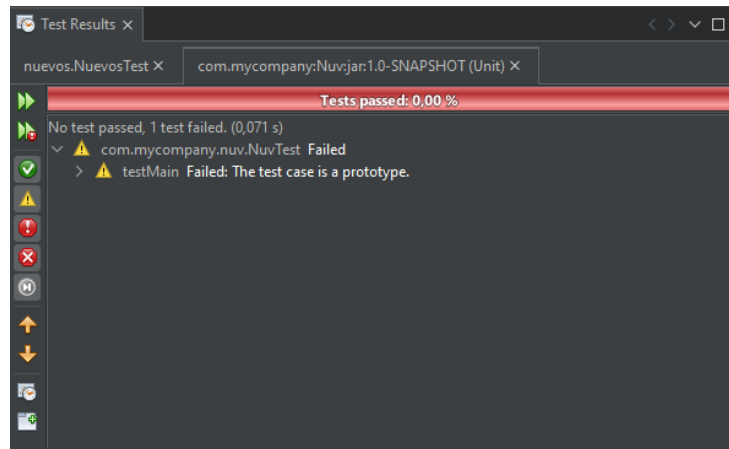


Figura 2.9: JUnit Test File.

Pero si quitamos fails nos aparecerá en verde:

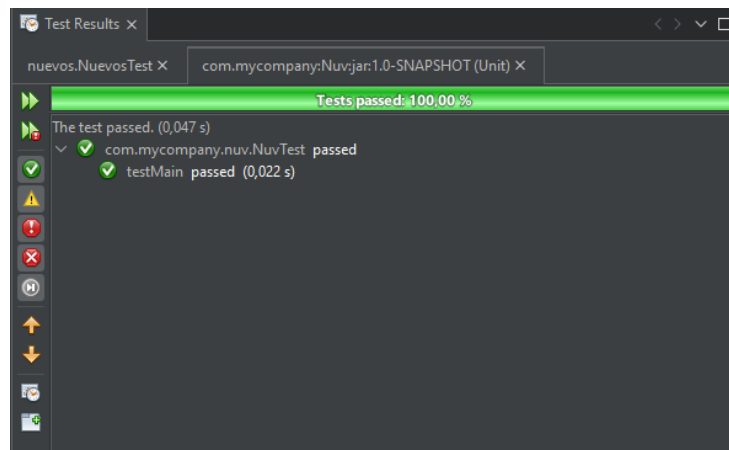


Figura 2.10: JUnit Test File.

Vamos a hacer una pequeña prueba con el aserto equals del método cálculo. Para ello mostramos el siguiente código:

```
1      import org.junit.jupiter.api.AfterEach;
2      import org.junit.jupiter.api.AfterAll;
3      import org.junit.jupiter.api.BeforeEach;
4      import org.junit.jupiter.api.BeforeAll;
5      import org.junit.jupiter.api.Test;
6      import static org.junit.jupiter.api.Assertions.*;
7
8      /**
9       *
10      * @author Abel
```

```
11      */
12      public class NuvTest {
13
14          Nuv n;
15
16          public NuvTest() {
17          }
18
19          @BeforeAll
20          public static void setUpClass() {
21
22
23          }
24
25          @AfterAll
26          public static void tearDownClass() {
27          }
28
29          @BeforeEach
30          public void setUp() {
31
32              Nuv n= new Nuv(3, "Hola");
33
34          }
35
36          @AfterEach
37          public void tearDown() {
38
39              Nuv n=null;
40
41          }
42
43          /**
44           * Test of main method, of class Nuv.
45           */
46          @Test
47          public void testMain() {
48              System.out.println("main");
49              String[] args = null;
50              Nuv.main(args);
51
52              assertEquals(20, n.calculo());
53
54              // TODO review the generated test code and
55              // remove the default call to fail.
56              fail("The test case is a prototype.");
57          }
58      }
```

En `SetUp` creamos un objeto para usarlo en la prueba y en `tearDown` lo eliminamos, estos métodos no tienen porque llamarse `SetUp`, ni `tearDown`, les puedo cambiar el nombre, siempre que tenga la anotación `@AfterEach` y `@BeforeEach`. En esta prueba yo esperaba el valor 20 pero recibo 15 porque es lo que me da el método `calculo`, y se obtiene lo siguiente:

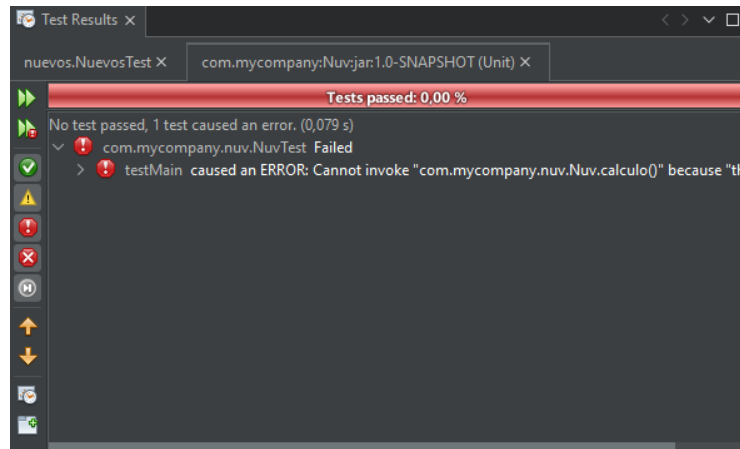


Figura 2.11: JUnit Test File.

Por lo si espero el valor 20 me he equivocado en el método `calculo` que debo arreglar el método para obtener 20, sumarle 5 por ejemplo.

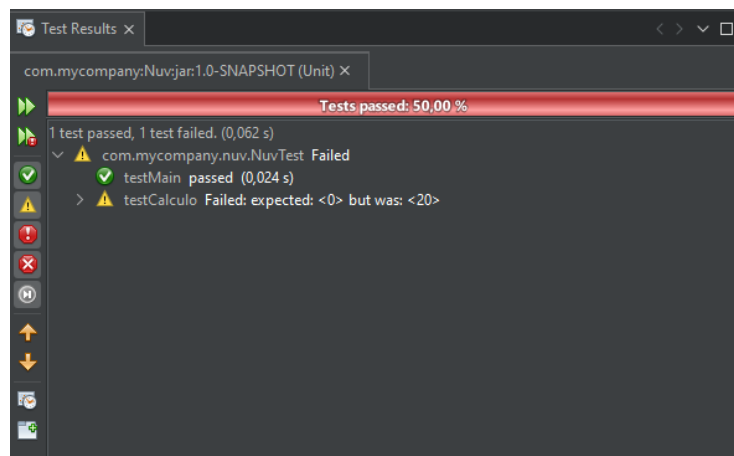


Figura 2.12: JUnit Test File.

Lo que hace JUnit, cuando hacemos el método `testMetodo`, buscar el Metodo en la clase que estamos testeando.

Se pide al alumno que investigue como se utiliza o se agrupa asertos con `assertAll()`, es decir, se agrupan varios tests.

Ten en cuenta que no tiene sentido crear objetos de la clase que estamos testeando en los métodos After y Before, en los cuatros. Se pueden crear objetos de cualquier otro método.

Desplegando todas las pruebas realizadas que son dos en nuestro caso se puede ver cada prueba:

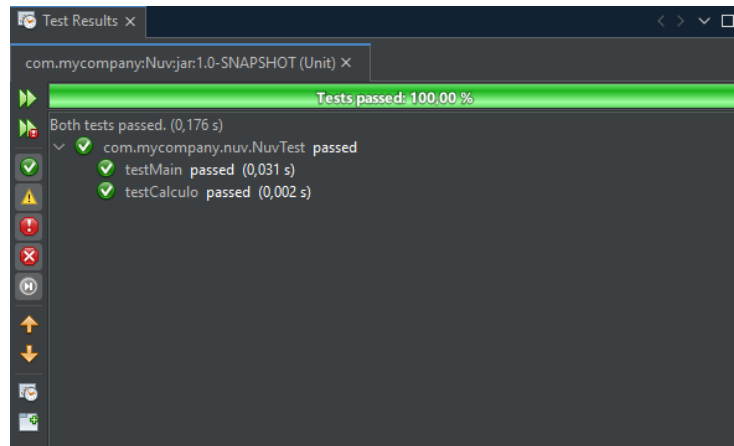


Figura 2.13: JUnit Test File.

Se deja al alumno hacer un pequeño resumen de cómo se usaría JUnit en Eclipse, poner alguna captura en el informe de la tarea.

3. Tarea propuesta JUnit

3.1. Primer ejercicio

Como ejercicio que se va a proponer al alumno, se expone el siguiente código o clase de los números complejos (tener en cuenta que junit 4 se distribuía en jar, pero en junit 5 no está en jar y funciona solo con proyectos Maven donde podemos ver las dependencias en el archivo pom.xml, los tipos de proyectos Ant solo funciona con JUnit4, con lo cual si queréis trabajar con ant tenéis que trabajar con JUnit4, debéis comentarlo en la práctica, al no haber .jar no podemos añadirlo en los proyectos Ant y no nos funcionará con JUnit5, con lo cual usar Proyectos Maven con proyectos de JUnit 5):

```
61     package complejos;
62
63     import java.lang.Math;
64
65     /**
66     *
67     * @author Abel
68     */
69     public class Complejos {
70
71
72         private float real;
73         private float imaginario;
74
75
76         public Complejos(float a, float b)
77         {
78
79             real= a;
80             imaginario=b;
81
82         }
83
84         public float partereal()
85         {
86
87             return this.real;
88
89         }
90
91         public float parteimaginaria()
92         {
```

```
93         return this.imaginario;
94     }
95
96     public boolean imaginariopuro()
97     {
98         return (this.imaginario==0);
99     }
100
101     public boolean realpuro()
102     {
103         return (this.real==0);
104     }
105
106     public void Escribircomlejo()
107     {
108         if (this.real>=0)
109             System.out.println(this.imaginario + "+" + this
110                                 .real + "i");
111         else
112             System.out.println(this.imaginario + "-" + this.
113                                 real + "i");
114     }
115
116     public float modulo()
117     {
118         float mod=(float) Math.sqrt(this.real*this.real
119                                     +this.imaginario*this.imaginario);
120
121         return mod;
122     }
123
124     public float fase()
125     {
126         float arco;
```

```
140
141         arco=(float) Math.atan(this.real/(this.
            imaginario));
142
143         return arco;
144
145     }
146
147     public void conjugado()
148     {
149
150         this.real=(-1)*this.real;
151
152     }
153
154     public static void main(String[] args) {
155         // TODO code application logic here
156
157
158         Complejos z1=new Complejos(-3,4);
159
160         z1.Escribircomplejo();
161
162     }
163
164 }
```

Hacer las pruebas que se proponen a continuación y según las pruebas intentar corregir algunos errores, explique todos los errores a qué pueden deberse. Cuando se le da a crear Test y vamos a la carpeta de test se tiene el siguiente código, quitaremos los fails de todos o lo comentaremos:

```
166     package complejos;
167
168     import org.junit.jupiter.api.AfterEach;
169     import org.junit.jupiter.api.AfterAll;
170     import org.junit.jupiter.api.BeforeEach;
171     import org.junit.jupiter.api.BeforeAll;
172     import org.junit.jupiter.api.Test;
173     import static org.junit.jupiter.api.Assertions.*;
174
175     /**
176     *
177     * @author Abel
178     */
179     public class ComplejosTest {
180
181         public ComplejosTest() {
182
183 }
```

Tercera Práctica Unidad 3. Curso 2024/2025 20

```
230     */
231     @Test
232     public void testImaginario puro() {
233         System.out.println("imaginario puro");
234         Complejos instance = null;
235         boolean expResult = false;
236         boolean result = instance.imaginario puro();
237         assertEquals(expResult, result);
238         // TODO review the generated test code and
239         // remove the default call to fail.
240         fail("The test case is a prototype.");
241     }
242
243     /**
244     * Test of realpuro method, of class Complejos.
245     */
246     @Test
247     public void testRealpuro() {
248         System.out.println("realpuro");
249         Complejos instance = null;
250         boolean expResult = false;
251         boolean result = instance.realpuro();
252         assertEquals(expResult, result);
253         // TODO review the generated test code and
254         // remove the default call to fail.
255         fail("The test case is a prototype.");
256     }
257
258     /**
259     * Test of Escribircomplejo method, of class
260     * Complejos.
261     */
262     @Test
263     public void testEscribircomplejo() {
264         System.out.println("Escribircomplejo");
265         Complejos instance = null;
266         instance.Escribircomplejo();
267         // TODO review the generated test code and
268         // remove the default call to fail.
269         fail("The test case is a prototype.");
270     }
271
272     /**
273     * Test of modulo method, of class Complejos.
274     */
275     @Test
276     public void testModulo() {
277         System.out.println("modulo");
278         Complejos instance = null;
279         float expResult = 0.0F;
```

```
276         float result = instance.modulo();
277         assertEquals(expResult, result, 0);
278         // TODO review the generated test code and
279         //      remove the default call to fail.
280         fail("The test case is a prototype.");
281     }
282     /**
283     * Test of fase method, of class Complejos.
284     */
285     @Test
286     public void testFase() {
287         System.out.println("fase");
288         Complejos instance = null;
289         float expResult = 0.0F;
290         float result = instance.fase();
291         assertEquals(expResult, result, 0);
292         // TODO review the generated test code and
293         //      remove the default call to fail.
294         fail("The test case is a prototype.");
295     }
296     /**
297     * Test of conjugado method, of class Complejos.
298     */
299     @Test
300     public void testConjugado() {
301         System.out.println("conjugado");
302         Complejos instance = null;
303         instance.conjugado();
304         // TODO review the generated test code and
305         //      remove the default call to fail.
306         fail("The test case is a prototype.");
307     }
308     /**
309     * Test of main method, of class Complejos.
310     */
311     @Test
312     public void testMain() {
313         System.out.println("main");
314         String[] args = null;
315         Complejos.main(args);
316         // TODO review the generated test code and
317         //      remove the default call to fail.
318         fail("The test case is a prototype.");
319     }
320 }
```

Vamos a realizar los siguientes asertos quedando el código como sigue:

```
320     package com.mycompany.complejoss;
321
322     import org.junit.jupiter.api.AfterEach;
323     import org.junit.jupiter.api.AfterAll;
324     import org.junit.jupiter.api.BeforeEach;
325     import org.junit.jupiter.api.BeforeAll;
326     import org.junit.jupiter.api.Test;
327     import static org.junit.jupiter.api.Assertions.*;
328
329     /**
330     *
331     * @author Abel
332     */
333     public class ComplejossTest {
334
335         Complejoss numcomplex= new Complejoss(3,4);
336
337         public ComplejossTest() {
338         }
339
340         @BeforeAll
341         public static void setUpClass() {
342         }
343
344         @AfterAll
345         public static void tearDownClass() {
346         }
347
348         @BeforeEach
349         public void setUp() {
350         }
351
352         @AfterEach
353         public void tearDown() {
354         }
355
356         /**
357         * Test of partereal method, of class Complejoss.
358         */
359         @Test
360         public void testPartereal() {
361             System.out.println("partereal");
362             Complejoss instance = null;
363             float expResult = 0.0F;
364             float result = numcomplex.partereal();
365             assertEquals(3, result, 0);
366             // TODO review the generated test code and
367             //      remove the default call to fail.
368             fail("The test case is a prototype.");
369         }
370     }
```



```
369
370
371     /**
372      * Test of parteimaginaria method, of class
373      * Complejoss.
374      */
375     @Test
376     public void testParteimaginaria() {
377         System.out.println("parteimaginaria");
378         Complejoss instance = null;
379         float expResult = 0.0F;
380         float result = numcomplex.parteimaginaria();
381         assertEquals(4, result, 0);
382         // TODO review the generated test code and
383         // remove the default call to fail.
384         // fail("The test case is a prototype.");
385     }
386
387     /**
388      * Test of imaginariopuro method, of class
389      * Complejoss.
390      */
391     @Test
392     public void testImaginariopuro() {
393         System.out.println("imaginariopuro");
394         Complejoss instance = null;
395         boolean expResult = false;
396         boolean result = numcomplex.imaginariopuro();
397         assertEquals(false, result);
398         // TODO review the generated test code and
399         // remove the default call to fail.
400         // fail("The test case is a prototype.");
401     }
402
403     /**
404      * Test of realpuro method, of class Complejoss.
405      */
406     @Test
407     public void testRealpuro() {
408         System.out.println("realpuro");
409         Complejoss instance = null;
410         boolean expResult = false;
411         boolean result = numcomplex.realpuro();
412         assertEquals(false, result);
413         // TODO review the generated test code and
414         // remove the default call to fail.
415         // fail("The test case is a prototype.");
416     }
417
418     /**
419      * Test of Escribircomlejo method, of class
```

```
414         Complejoss.  
415     */  
416     @Test  
417     public void testEscribircomlejo() {  
418         System.out.println("Escribircomlejo");  
419         Complejoss instance = null;  
420         numcomplex.Escribircomlejo();  
421         //assertEquals("3+4i", numcomplex.  
422             Escribircomlejo());  
423         // TODO review the generated test code and  
424             remove the default call to fail.  
425         // fail("The test case is a prototype.");  
426     }  
427  
428     /**  
429     * Test of modulo method, of class Complejoss.  
430     */  
431     @Test  
432     public void testModulo() {  
433         System.out.println("modulo");  
434         Complejoss instance = null;  
435         float expResult = 0.0F;  
436         float result = numcomplex.modulo();  
437         assertEquals(5, result, 0);  
438         // TODO review the generated test code and  
439             remove the default call to fail.  
440         // fail("The test case is a prototype.");  
441     }  
442  
443     /**  
444     * Test of fase method, of class Complejoss.  
445     */  
446     @Test  
447     public void testFase() {  
448         System.out.println("fase");  
449         Complejoss instance = null;  
450         float expResult = 0.0F;  
451         float result = numcomplex.fase();  
452         assertEquals(0.92729, result, 0.001);  
453         // TODO review the generated test code and  
454             remove the default call to fail.  
455         // fail("The test case is a prototype.");  
456     }  
457  
458     /**  
459     * Test of conjugado method, of class Complejoss.  
460     */  
461     @Test  
462     public void testConjugado() {  
463         System.out.println("conjugado");
```

```
459         Complejoss instance = null;
460         numcomplex.conjugado();
461         // TODO review the generated test code and
462         //      remove the default call to fail.
463         //      fail("The test case is a prototype.");
464     }
465
466     /**
467     * Test of main method, of class Complejoss.
468     */
469     @Test
470     public void testMain() {
471         System.out.println("main");
472         String[] args = null;
473         Complejoss.main(args);
474         // TODO review the generated test code and
475         //      remove the default call to fail.
476         //      fail("The test case is a prototype.");
477     }
478 }
```

Las pruebas aparentemente están bien, solo falla un método que no da lo que se espera, pero los demás sí, ¿cómo puedo arreglar este método?. ¿Con esta prueba es suficiente o hay que hacer más?, pruebe creando objetos complejos como $0+4i$ y $3+0i$ y haga una prueba. En clase se dirán los valores esperados para que lo probéis.

3.2. Segundo ejercicio

En este segundo ejercicio se va a utilizar la clase anterior, se compone de cuatro métodos de operaciones sobre números complejos. El código fuente es el siguiente:

```
479 package com.mycompany.complejoss;
480
481     /**
482     *
483     * @author Abel
484     */
485     public class operac {
486
487     public Complejoss suma(Complejoss a, Complejoss b)
488     {
489
490         float r;
491         float im;
492
493         r=a.partereal()+b.partereal();
494         im=a.parteimaginaria()+b.parteimaginaria();
495     }
```

```
496         Complejoss c=new Complejoss(r,im);
497
498
499         return c;
500
501     }
502     public Complejoss resta(Complejoss a, Complejoss b)
503     {
504
505         float r;
506         float im;
507
508         r=a.partereal()-b.partereal();
509         im=a.parteimaginaria()-b.parteimaginaria();
510
511         Complejoss c=new Complejoss(r,im);
512
513
514         return c;
515
516     }
517     public Complejoss multiplicar(Complejoss a, Complejoss b)
518     {
519
520         float r;
521         float im;
522
523         r=a.partereal()*b.partereal()-a.parteimaginaria()*b.
           parteimaginaria();
524         im=a.partereal()*b.parteimaginaria()-b.partereal()*a.
           parteimaginaria();
525
526         Complejoss c=new Complejoss(r,im);
527
528         return c;
529
530     }
531     public Complejoss dividir(Complejoss a, Complejoss b)
532     {
533
534         float r;
535         float im;
536         float d=b.partereal()*b.partereal()+b.parteimaginaria()
           *b.parteimaginaria();
537
538         r=a.partereal()*b.partereal()+a.parteimaginaria()*b.
           parteimaginaria();
539         im=a.partereal()*b.parteimaginaria()-b.partereal()*a.
           parteimaginaria();
540
```

```
541         r=r/d;
542         im=im/d;
543
544         Complejoss c=new Complejoss(r,im);
545
546
547         return c;
548
549     }
550
551
552 }
```

Una vez pinchamos creado la clase test se obtiene el siguiente código que modificaremos más adelante para realizar las pruebas:

```
553 import org.junit.jupiter.api.AfterEach;
554 import org.junit.jupiter.api.AfterAll;
555 import org.junit.jupiter.api.BeforeEach;
556 import org.junit.jupiter.api.BeforeAll;
557 import org.junit.jupiter.api.Test;
558 import static org.junit.jupiter.api.Assertions.*;
559
560 /**
561  *
562  * @author Abel
563  */
564 public class operacTest {
565
566     public operacTest() {
567     }
568
569     @BeforeAll
570     public static void setUpClass() {
571     }
572
573     @AfterAll
574     public static void tearDownClass() {
575     }
576
577     @BeforeEach
578     public void setUp() {
579     }
580
581     @AfterEach
582     public void tearDown() {
583     }
584
585     /**
586     * Test of suma method, of class operac.
```

```
587     */
588     @Test
589     public void testSuma() {
590         System.out.println("suma");
591         Complejoss a = null;
592         Complejoss b = null;
593         operac instance = new operac();
594         Complejoss expResult = null;
595         Complejoss result = instance.suma(a, b);
596         assertEquals(expResult, result);
597         // TODO review the generated test code and remove the
598         //      default call to fail.
599         fail("The test case is a prototype.");
600     }
601
602     /**
603     * Test of resta method, of class operac.
604     */
605     @Test
606     public void testResta() {
607         System.out.println("resta");
608         Complejoss a = null;
609         Complejoss b = null;
610         operac instance = new operac();
611         Complejoss expResult = null;
612         Complejoss result = instance.resta(a, b);
613         assertEquals(expResult, result);
614         // TODO review the generated test code and remove the
615         //      default call to fail.
616         fail("The test case is a prototype.");
617     }
618
619     /**
620     * Test of multiplicar method, of class operac.
621     */
622     @Test
623     public void testMultiplicar() {
624         System.out.println("multiplicar");
625         Complejoss a = null;
626         Complejoss b = null;
627         operac instance = new operac();
628         Complejoss expResult = null;
629         Complejoss result = instance.multiplicar(a, b);
630         assertEquals(expResult, result);
631         // TODO review the generated test code and remove the
632         //      default call to fail.
633         fail("The test case is a prototype.");
634     }
635
636     /**
```

```
634     * Test of dividir method, of class operac.
635     */
636     @Test
637     public void testDividir() {
638         System.out.println("dividir");
639         Complejoss a = null;
640         Complejoss b = null;
641         operac instance = new operac();
642         Complejoss expResult = null;
643         Complejoss result = instance.dividir(a, b);
644         assertEquals(expResult, result);
645         // TODO review the generated test code and remove the
646         // default call to fail.
647         fail("The test case is a prototype.");
648     }
649 }
```

Modificando el código anterior para realizar las pruebas tenemos que queda como (corregir errores lo que sea posible del código original):

```
653 package com.mycompany.complejoss;
654
655 import org.junit.jupiter.api.AfterEach;
656 import org.junit.jupiter.api.AfterAll;
657 import org.junit.jupiter.api.BeforeEach;
658 import org.junit.jupiter.api.BeforeAll;
659 import org.junit.jupiter.api.Test;
660 import static org.junit.jupiter.api.Assertions.*;
661
662 /**
663  *
664  * @author Abel
665  */
666 public class operacTest {
667
668     public operacTest() {
669     }
670
671     @BeforeAll
672     public static void setUpClass() {
673     }
674
675     @AfterAll
676     public static void tearDownClass() {
677     }
678
679     @BeforeEach
680     public void setUp() {
681     }
```

```
682
683 @AfterEach
684 public void tearDown() {
685 }
686
687 /**
688  * Test of suma method, of class operac.
689  */
690 @Test
691 public void testSuma() {
692     System.out.println("suma");
693     Complejoss a = new Complejoss (3,4);
694     Complejoss b = new Complejoss(-2,5);
695     operac instance = new operac();
696     Complejoss expectedResult = new Complejoss(1,9);
697     Complejoss result = instance.suma(a, b);
698     assertEquals(expResult.partereal(), result.partereal())
699         ;
700     assertEquals(expResult.parteimaginaria(), result.
701         parteimaginaria());
702     // TODO review the generated test code and remove the
703     // default call to fail.
704     // fail("The test case is a prototype.");
705 }
706
707 /**
708  * Test of resta method, of class operac.
709  */
710 @Test
711 public void testResta() {
712     System.out.println("resta");
713     Complejoss a = new Complejoss (3,4);
714     Complejoss b = new Complejoss(-2,5);
715     operac instance = new operac();
716     Complejoss expectedResult = new Complejoss(5,-1);
717     Complejoss result = instance.resta(a, b);
718     assertEquals(expResult.partereal(), result.partereal())
719         ;
720     assertEquals(expResult.parteimaginaria(), result.
721         parteimaginaria());
722     // TODO review the generated test code and remove the
723     // default call to fail.
724     // fail("The test case is a prototype.");
725 }
726
727 /**
728  * Test of multiplicar method, of class operac.
729  */
730 @Test
731 public void testMultiplicar() {
```



```
726     System.out.println("multiplicar");
727     Complejoss a = new Complejoss (3,4);
728     Complejoss b = new Complejoss(-2,5);
729     operac instance = new operac();
730     Complejoss expResult = new Complejoss(-26,7);
731     Complejoss result = instance.multiplicar(a, b);
732     assertEquals(expResult.partereal(), result.partereal())
733     ;
734     assertEquals(expResult.parteimaginaria(), result.
735         parteimaginaria());
736     // TODO review the generated test code and remove the
737     // default call to fail.
738     // fail("The test case is a prototype.");
739 }
740
741 /**
742  * Test of dividir method, of class operac.
743  */
744 @Test
745 public void testDividir() {
746     System.out.println("dividir");
747     Complejoss a = new Complejoss (3,4);
748     Complejoss b = new Complejoss(-2,5);
749     operac instance = new operac();
750     Complejoss expResult = new Complejoss((float)0.4827, (
751         float)- 0.793);
752     Complejoss result = instance.dividir(a, b);
753     assertEquals(expResult.partereal(), result.partereal(),
754         0.001);
755     assertEquals(expResult.parteimaginaria(), result.
756         parteimaginaria(), 0.001);
757     // TODO review the generated test code and remove the
758     // default call to fail.
759     // fail("The test case is a prototype.");
760 }
761 }
```

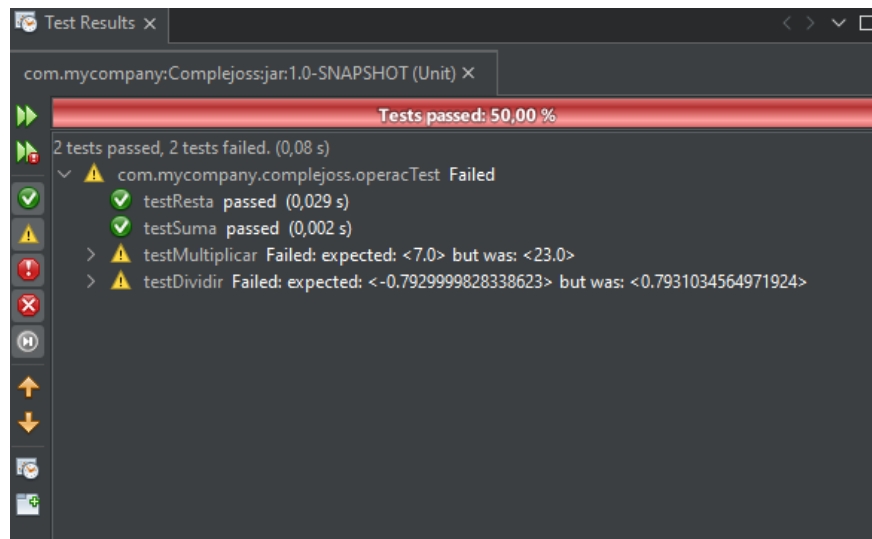


Figura 3.1: JUnit Test File.

4. Herramientas de construcción en Java

En este capítulo se va a realizar un breve resumen de las herramientas de Apache que tiene integradas en Netbeans para construir proyectos de forma automático, se verán los tres más usados **Apache Ant**, **Apache Maven**, **Gradle**:

- **Apache Ant**: Apache Ant ("Another Neat Tool") es una librería de Java usada para construcción automática de procesos para aplicaciones Java. También puede ser usado para construir proyectos que no sean de Java. Formó parte del código de Apache Tomcat. Sustituyó a Make de Unix pero es muy similar. Es muy fácil de usar para usuarios que están empezando. Ant construye archivos que son escritos en xml, es conocido como el build.xml. Este archivo se puede ver en la pestaña file del proyecto.

En el archivo build define cuatro propiedades o targets: clean, compile, jar y run. Se puede compilar el código corriendo ant compile. Se deja al alumno como tarea investigar para qué es el archivo manifest. El archivo xml ahora se encuentra mayormente en la carpeta nbproject. Lo único bueno que tiene Ant es la flexibilidad y portabilidad, no impone convenciones o cosas ya predefinidas.

- **Apache Maven**: Maven continúa usando al igual que Ant archivos XML para la construcción de proyectos pero mucho más manejable que Ant. Aquí sí hay convenciones mientras que Ant da flexibilidad y requiere que todo sea escrito, Maven confía en las convenciones y comandos ya predefinidos. La configuración de Maven va en un archivo XML. La estructura del proyecto es estandarizado como opuesto a Ant, no hay que poner cada una de las fases manualmente en el proceso de construcción del proyecto. Maven está hecho mayormente por plugins. El archivo XML se llama pom.xml (Project Object Model). El significado de la palabra Maven viene de una palabra judía que significa acumulador de conocimiento. En definitiva, se puede decir que Maven sacrifica portabilidad y flexibilidad en aras a la rigidez de las convenciones y comandos ya predefinidos.
- **Gradle**: Es construido sobre los conceptos de Ant y Maven. Pero una de las diferencias de Gradle es que no usa archivos XML. Aquí Gradle usa un lenguaje propio DSL basado en Groovy o Kotlin. Esto conduce a unos archivos de configuración más pequeños. Se construye en un archivo llamado build.gradle en Groovy o build.gradle.kts en Kotlin. Kotlin suele ofrecer mucho mejor soporte al IDE que el Groovy. Gradle utiliza pasos de construcción llamadas "tasks", lo contrario a lo que usa ANT "target" o "phases" en Maven.

Se propone como tarea que el alumno profundice un poco más sobre cada uno de las herramientas anteriores y haga un breve resumen exponiendo tres principales características de cada uno.

5. Entrega

La entrega de la práctica se enviará por la plataforma moodle con el formato siguiente: nombrealumno_apellido1_apellido2.pdf, deberá incluir todas las capturas comentadas brevemente lo que se ha hecho en ellas con una o dos líneas. El plazo queda fijado un día antes de las evaluaciones.

Además se tendrá en cuenta la presentación del documento de la práctica, y aportaciones extras que se hagan.

6. Bibliografía

- [1] JUnit 5 para pruebas <https://junit.org/junit5/> Página oficial de JUnit
- [2] Manual de Junit, Diciembre 2014 https://www.ejie.euskadi.eus/contenidos/informacion/ejie2016_contratacion_tecnica/eu_def/adjuntos/JUnit.%20Manual%20de%20Usuario%20v2.0.pdf
- [3] Prácticas pruebas unitarias, algunos ejercicios <https://www.cc.uah.es/drg/docencia/Pruebas/Practicas.pdf>
- [4] Pruebas unitarias con JUnit 2016, UPM (universidad politécnica de Madrid) <https://www.dit.upm.es/~pepe/doc/adsw/base/junit.pdf>
- [5] JUnit DAM/DAW <https://www.slideshare.net/slideshow/pruebas-unitarias-unitarias-en-java-con-junit/267719843>
- [6] Ejercicio con JUnit 4. Youtube url: <https://www.youtube.com/watch?v=v-e3OgEP6uQ>
- [7] Curso Youtube Junit5: <https://www.youtube.com/watch?v=4SD8q891ZQc>
- [8] Guía JUnit en Eclipse Universidad de Málaga (UMA) http://www.lcc.uma.es/~vicente/docencia/poo/teoria/poo_guia_eclipse_junit.pdf
- [9] Tutorial de JUnit <https://www.guru99.com/es/junit-tutorial.html>