

Entornos de Desarrollo Unidad 3 Ejercicios Prácticos 1 y 2

Ejercicio 1

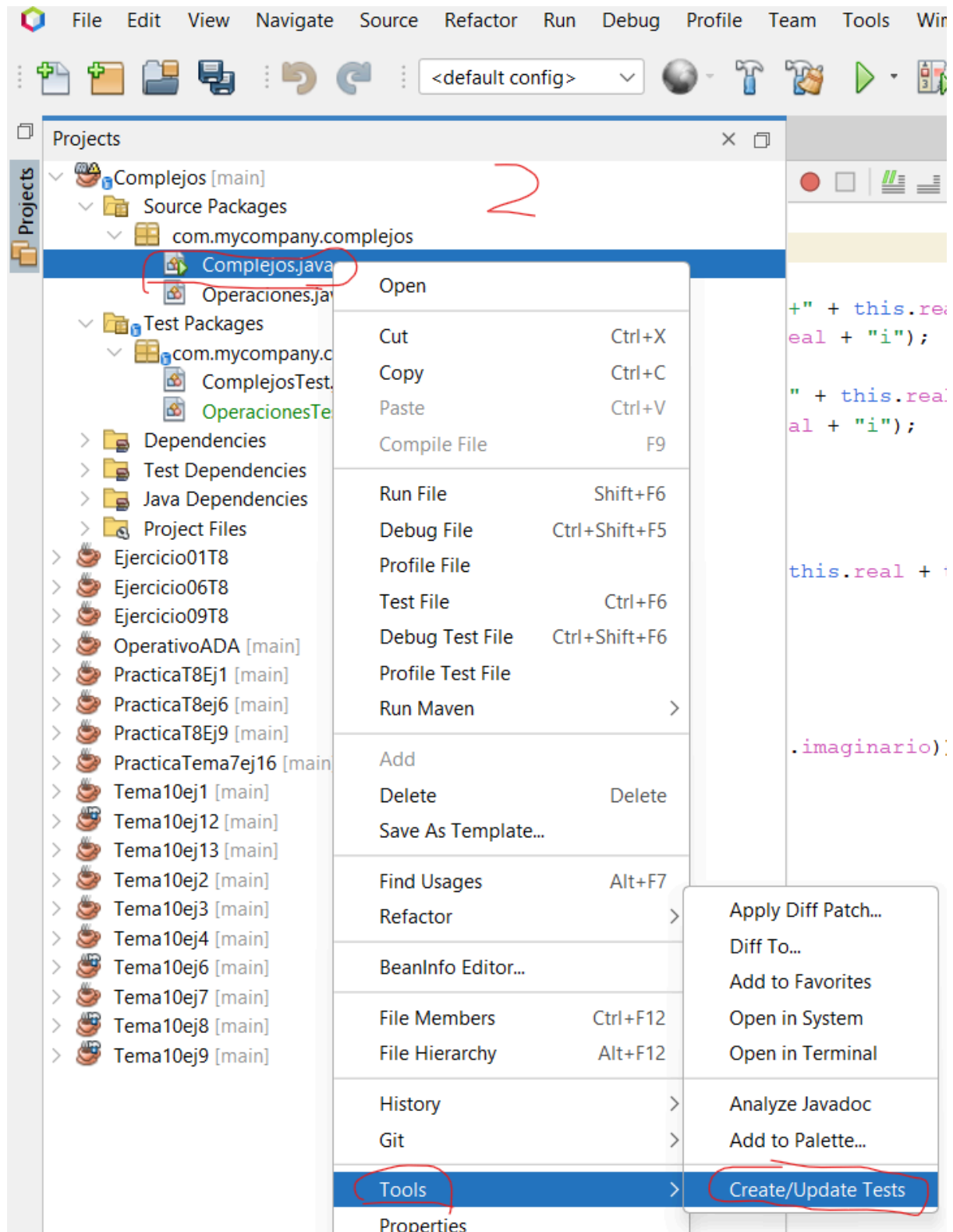
1) Lo primero que mostraremos es el código fuente de la clase Complejos:

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  */
4  package com.mycompany.complejos;
5
6  import java.lang.Math;
7
8  /**
9   *
10   * @author Abel
11   */
12  public class Complejos {
13
14      private float real;
15      private float imaginario;
16
17      public Complejos(float a, float b) {
18          real = a;
19          imaginario = b;
20      }
21
22      public float partereal() {
23          return this.real;
24      }
25
26      public float parteimaginaria() {
27          return this.imaginario;
28      }
29
30      public boolean imaginariopuro() {
31          return (this.imaginario == 0);
32      }
33
34      public boolean realpuro() {
35          return (this.real == 0);
36      }
37  }
```

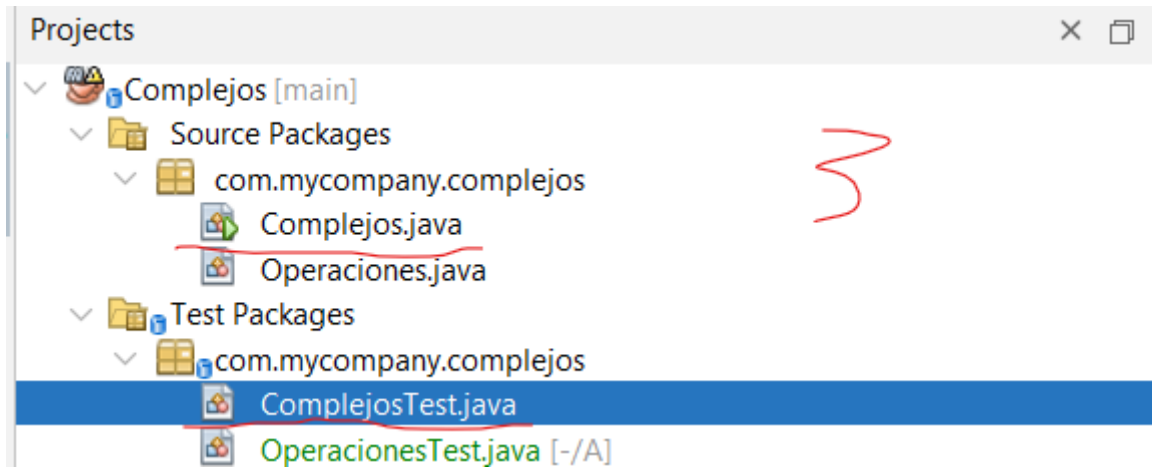


```
public String Escribircomplejo() {  
    if (this.real >= 0) {  
        System.out.println(this.imaginario + "+" + this.real + "i");  
        return (this.imaginario + "+" + this.real + "i");  
    } else {  
        System.out.println(this.imaginario + "-" + this.real + "i");  
        return (this.imaginario + "-" + this.real + "i");  
    }  
}  
  
public float modulo() {  
    float mod = (float) Math.sqrt(this.real * this.real + this.imaginario * this.imaginario);  
    return mod;  
}  
  
public float fase() {  
    float arco;  
    arco = (float) Math.atan(this.real / (this.imaginario));  
    return arco;  
}  
  
public void conjugado() {  
    this.real = (-1) * this.real;  
}  
  
public static void main(String[] args) {  
    // TODO code application logic here  
    Complejos z1 = new Complejos(-3, 4);  
    z1.Escribircomplejo();  
}
```

- 2)** Después crearemos la clase Test. Esto se hace haciendo click derecho en la clase Complejos, luego en Tools y Create/Update Tests.



3) Vemos la clase Test creada:



4) A continuación se mostrará todo el código fuente de la clase test:

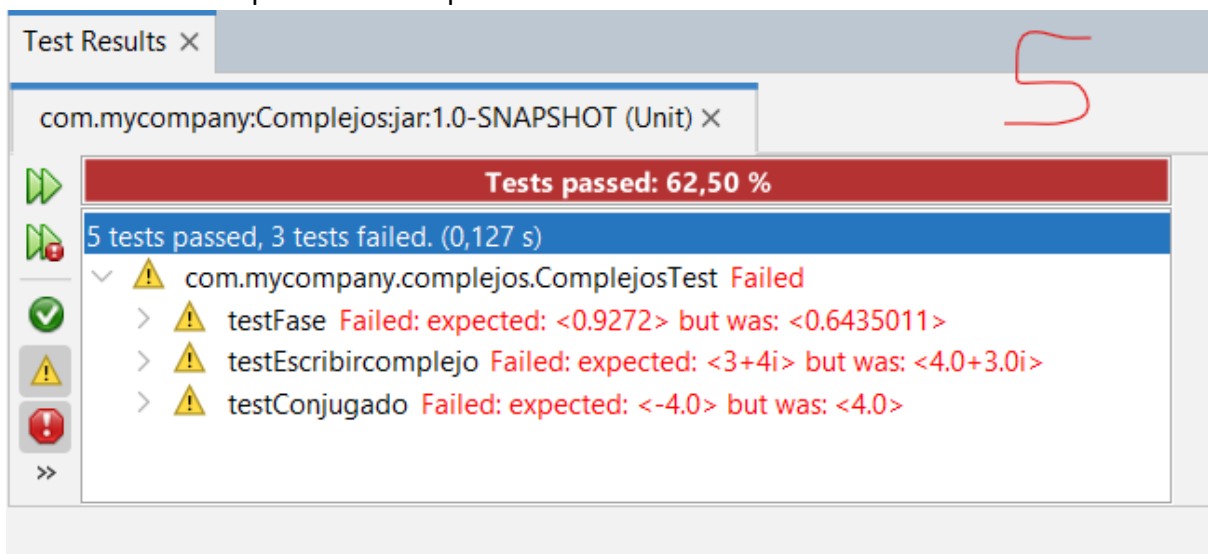
```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/UnitTests/JUnit5TestClass.java to edit this template
4  */
5  package com.mycompany.complejos;
6
7  import org.junit.jupiter.api.Test;
8  import static org.junit.jupiter.api.Assertions.*;
9
10 /**
11  *
12  * @author alumno
13  */
14 public class ComplejosTest {
15
16     /**
17      * Test of partereal method, of class Complejos.
18      */
19     @Test
20     void testPartereal() {
21         //fail("Not yet implemented");
22
23         // Crear un objeto complejo
24         Complejos C1 = new Complejos(3, 4);
25
26         //Valor esperado y valor obtenido
27         float esperado = 3.0F;
28         float obtenido = C1.partereal();
29
30         assertEquals(esperado, obtenido, 0);
31     }
32
33
34     /**
35      * Test of parteimaginaria method, of class Complejos.
36      */
37 }
```

```
37
38 ☐ @Test
39 void testParteimaginaria() {
40     //fail("Not yet implemented");
41
42     Complejos C1 = new Complejos(3, 4);
43
44     //Valor esperado y valor obtenido
45     float esperado = 4.0F;
46     float obtenido = C1.parteimaginaria();
47
48     assertEquals(esperado, obtenido, 0);
49 }
50
51 ☐ /**
52  * Test of imaginariopuro method, of class Complejos.
53  */
54 @Test
55 ☐ void testImaginariopuro() {
56     //fail("Not yet implemented");
57
58     Complejos C1 = new Complejos(3, 4);
59
60     //Valor esperado y valor obtenido
61     boolean esperado = false;
62     boolean obtenido = C1.imaginariopuro();
63
64     assertEquals(esperado, obtenido);
65
66 }
67
68 ☐ /**
69  * Test of realpuro method, of class Complejos.
70  */
```

```
71  @Test
72  void testRealpuro() {
73      //fail("Not yet implemented");
74
75      Complejos C1 = new Complejos(3, 4);
76
77      //Valor esperado y valor obtenido
78      boolean esperado = false;
79      boolean obtenido = C1.realpuro();
80
81      assertEquals(esperado, obtenido);
82
83  }
84
85  @Test
86  void testEscribircomplejo() {
87      //fail("Not yet implemented");
88
89      Complejos C1 = new Complejos(3, 4);
90
91      //Valor esperado y valor obtenido
92      String esperado = "3+4i";
93      String obtenido = C1.Escribircomplejo();
94
95      assertEquals(esperado, obtenido);
96
97  }
98
99  @Test
100 void testModulo() {
101     //fail("Not yet implemented");
102
103     Complejos C1 = new Complejos(3, 4);
104
105     //Valor esperado y valor obtenido
106     float esperado = 5.0F;
107     float obtenido = C1.modulo();
```

```
108
109     assertEquals(esperado, obtenido, 0);
110
111 }
112
113 @Test
114 void testFase() {
115     //fail("Not yet implemented");
116
117     Complejos C1 = new Complejos(3, 4);
118
119     //Valor esperado y valor obtenido
120     float esperado = 0.9272F;
121     float obtenido = C1.fase();
122
123     assertEquals(esperado, obtenido, 0);
124
125 }
126
127 @Test
128 void testConjugado() {
129     //fail("Not yet implemented");
130
131     Complejos C1 = new Complejos(3, 4);
132
133     C1.conjugado();
134
135     float esperado = -4;
136     float obtenido = C1.parteimaginaria();
137
138     assertEquals(esperado, obtenido, 0);
139
140 }
141
142 }
```

- 5) Si hacemos click derecho y después “Test file” comenzaremos con las pruebas. Nos aparecerá este mensaje diciéndonos que 3 de los 8 tests han fallado. Podemos ver cuáles son: testFase, testEscribirComplejo y testConjugado. Se nos indica el resultado esperado en comparación con el resultado obtenido.



The screenshot shows the 'Test Results' window in an IDE. The title bar says 'Test Results x'. Below it, the test runner is identified as 'com.mycompany:Complejos:jar:1.0-SNAPSHOT (Unit) x'. A red progress bar indicates 'Tests passed: 62,50 %'. Below the progress bar, a blue bar states '5 tests passed, 3 tests failed. (0,127 s)'. A list of test results follows, with three tests marked as failed (indicated by a yellow warning icon):

- com.mycompany.complejos.ComplejosTest Failed
 - testFase Failed: expected: <0.9272> but was: <0.6435011>
 - testEscribirComplejo Failed: expected: <3+4i> but was: <4.0+3.0i>
 - testConjugado Failed: expected: <-4.0> but was: <4.0>

A large red handwritten number '5' is visible in the top right corner of the screenshot.

- 6) Podemos ver un objeto tipo Complejos creado, que recibe dos parámetros: el resultado esperado y el obtenido. Después utiliza assertEquals para comprobar si ambas variables son iguales en la prueba unitaria. La variable “obtenido” llama al siguiente método:

```
@Test
void testFase() {
    //fail("Not yet implemented");

    Complejos C1 = new Complejos(3, 4);

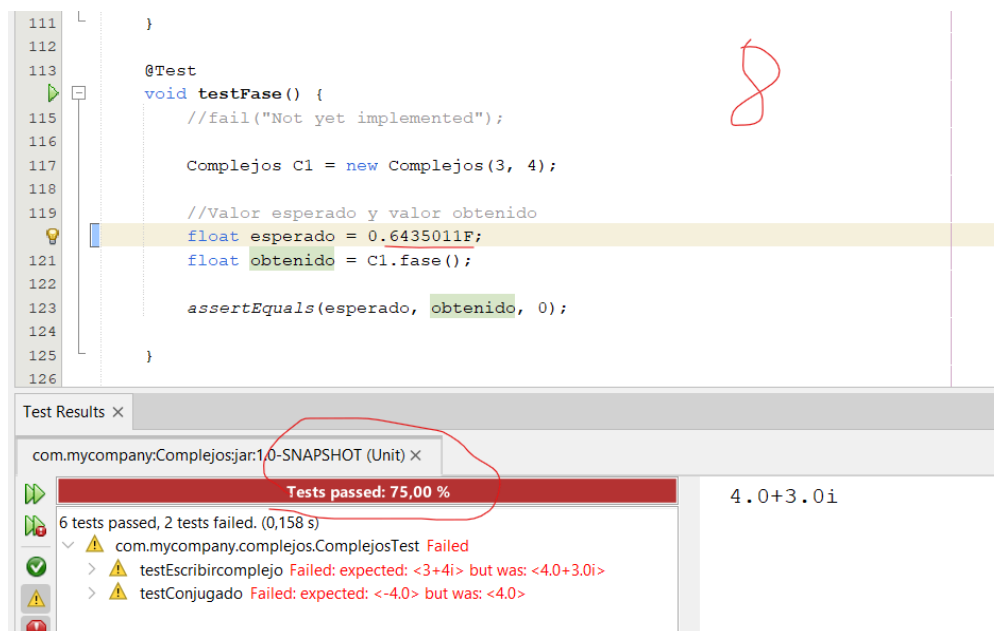
    //Valor esperado y valor obtenido
    float esperado = 0.9272F;
    float obtenido = C1.fase();

    assertEquals(esperado, obtenido, 0);
}
```

- 7) El método fase sirve para calcular la arcotangente inversa entre dos variables: “real” e “imaginario”:

```
public float fase() {
    float arco;
    arco = (float) Math.atan(this.real / (this.imaginario));
    return arco;
}
```

- 8) Después de corregir el resultado esperado, vemos que el porcentaje de tests acertados ahora es de 75%. Debemos tener en cuenta que en variables de tipo double o float, los decimales pueden ocasionar problemas a la hora de realizar los tests.



```
111 }
112
113 @Test
114 void testFase() {
115     //fail("Not yet implemented");
116
117     Complejos C1 = new Complejos(3, 4);
118
119     //Valor esperado y valor obtenido
120     float esperado = 0.6435011F;
121     float obtenido = C1.fase();
122
123     assertEquals(esperado, obtenido, 0);
124 }
125
126
```

Test Results x

com.mycompany:Complejos.jar:1.0-SNAPSHOT (Unit) x

Tests passed: 75,00 %

4.0+3.0i

6 tests passed, 2 tests failed. (0,158 s)

- com.mycompany.complejos.ComplejosTest Failed
 - testEscribirComplejo Failed: expected: <3+4i> but was: <4.0+3.0i>
 - testConjugado Failed: expected: <-4.0> but was: <4.0>

Ahora vamos con los dos tests restantes.

- 9) El test que prueba el método **Escribircomplejo** de la clase **Complejos** que devuelve un String de caracteres en función de si un valor es mayor que el otro o no. Si real es mayor que imaginario, obtendríamos, en este caso: **"4.0+3.0i"**. El resultado esperado aquí tiene dos errores. El primero que los números son float, por tanto el resultado esperado debería tener el **".0"** en cuenta (como hemos mencionado antes), y por otro lado que el valor imaginario va antes que el real:

```

84
85  @Test
86  void testEscribircomplejo() {
87      //fail("Not yet implemented");
88
89      Complejos C1 = new Complejos(3, 4);
90
91      //Valor esperado y valor obtenido
92      String esperado = "3+4i";
93      String obtenido = C1.Escribircomplejo();
94
95      assertEquals(esperado, obtenido);
96
97  }

```

37

```

38  public String Escribircomplejo() {
39      if (this.real >= 0) {
40          System.out.println(this.imaginario + "+" + this.real + "i");
41          return (this.imaginario + "+" + this.real + "i");
42      } else {
43          System.out.println(this.imaginario + "" + this.real + "i");
44          return (this.imaginario + "" + this.real + "i");
45      }
46  }

```

- 11) Una vez ajustamos el resultado, el porcentaje de tests aprobados pasa a 87,50%. Queda uno más todavía.

```

85  @Test
86  void testEscribircomplejo() {
87      //fail("Not yet implemented");
88
89      Complejos C1 = new Complejos(3, 4);
90
91      //Valor esperado y valor obtenido
92      String esperado = "4.0+3.0i";
93      String obtenido = C1.Escribircomplejo();
94
95      assertEquals(esperado, obtenido);
96
97  }

```

Test Results x

com.mycompany:Complejos:jar:1.0-SNAPSHOT (Unit) x

Tests passed: 87,50 %

7 tests passed, 1 test failed. (0,158 s)


com.mycompany.complejos.ComplejosTest Failed

> testConjugado Failed: expected: <-4.0> but was: <4.0>


4.0+3.0i

12) El método conjugado devuelve el valor “real” multiplicado por -1.

13) En el resultado esperado tenemos -4.0, pero hemos obtenido 4.0. Esto se debe bien a que hemos escrito mal el código en conjugado, o a que hemos escrito mal los resultados esperados.

```
58  
59  public void conjugado() {  
60     this.real = (-1) * this.real;  
61 }  
62
```


72


```
126  
127 @Test  
128  void testConjugado() {  
129     // fail("Not yet implemented");  
130  
131     Complejos C1 = new Complejos(3, 4);  
132  
133     C1.conjugado();  
134  
135     float esperado = -4;  
136     float obtenido = C1.parteimaginaria();  
137  
138     assertEquals(esperado, obtenido, 0);  
139  
140 }  
141  
142 }  
143
```


73



Test Results ×

com.mycompany:Complejos:jar:1.0-SNAPSHOT (Unit) ×

 Tests passed: 87,50 %

 7 tests passed, 1 test failed. (0,158 s)

 com.mycompany.complejos.ComplejosTest Failed

 >  testConjugado Failed: expected: <-4.0> but was: <4.0>

4.0+3.0i

Hay dos posibles soluciones:

14) Podemos cambiar el código de conjugado para que cambie el valor “imaginario” en lugar del “real”.

```
57     }
58
59     public void conjugado() {
60         this.imaginario = (-1) * this.imaginario;
61     }
62
63     public static void main(String[] args) {
64         // TODO code application logic here
65         Complejos z1 = new Complejos(-3, 4);
66         z1.Escribircomplejo();
67     }
68 }
69
```

74

Test Results ×

com.mycompany:Complejos.jar:1.0-SNAPSHOT (Unit) ×

Tests passed: 100,00 %

4.0+3.0i

All 8 tests passed. (0,123 s)

15) Otra solución es llamar a `partereal` en lugar de a `parteimaginaria`, en cuyo caso deberíamos esperar -3 como resultado.

```
127     @Test
128     void testConjugado() {
129         // fail("Not yet implemented");
130
131         Complejos C1 = new Complejos(3, 4);
132
133         C1.conjugado();
134
135         float esperado = -3;
136         float obtenido = C1.partereal();
137
138         assertEquals(esperado, obtenido, 0);
139     }
140
141 }
142
143
```

75

Test Results ×

com.mycompany:Complejos.jar:1.0-SNAPSHOT (Unit) ×

Tests passed: 100,00 %

4.0+3.0i

All 8 tests passed. (0,152 s)

16) También podemos simplemente cambiar el resultado esperado a 4.

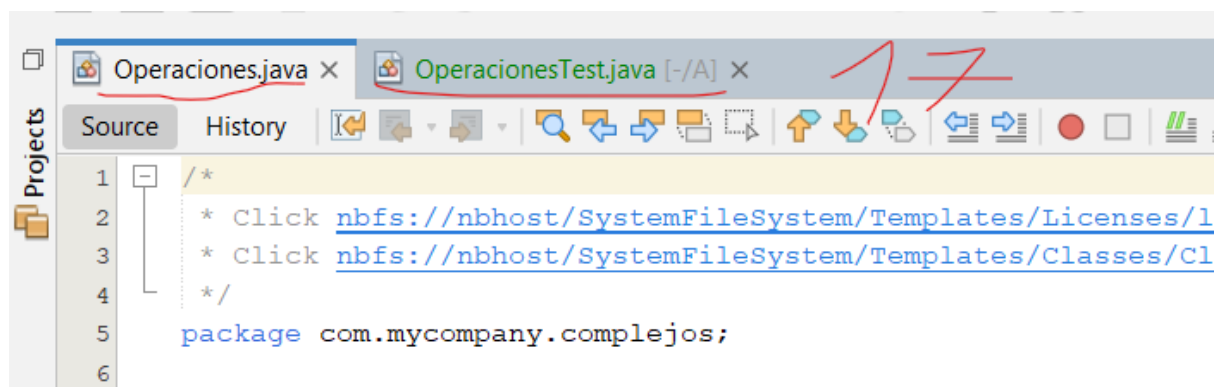
```
126
127     @Test
128     void testConjugado() {
129         //      fail("Not yet implemented");
130
131         Complejos C1 = new Complejos(3, 4);
132
133         C1.conjugado();
134
135         float esperado = 4;
136         float obtenido = C1.parteimaginaria();
137
138         assertEquals(esperado, obtenido, 0);
139
140     }
141
142 }
```

76

En cualquiera de los casos, la solución dependerá de nuestras necesidades.

Ejercicio 2

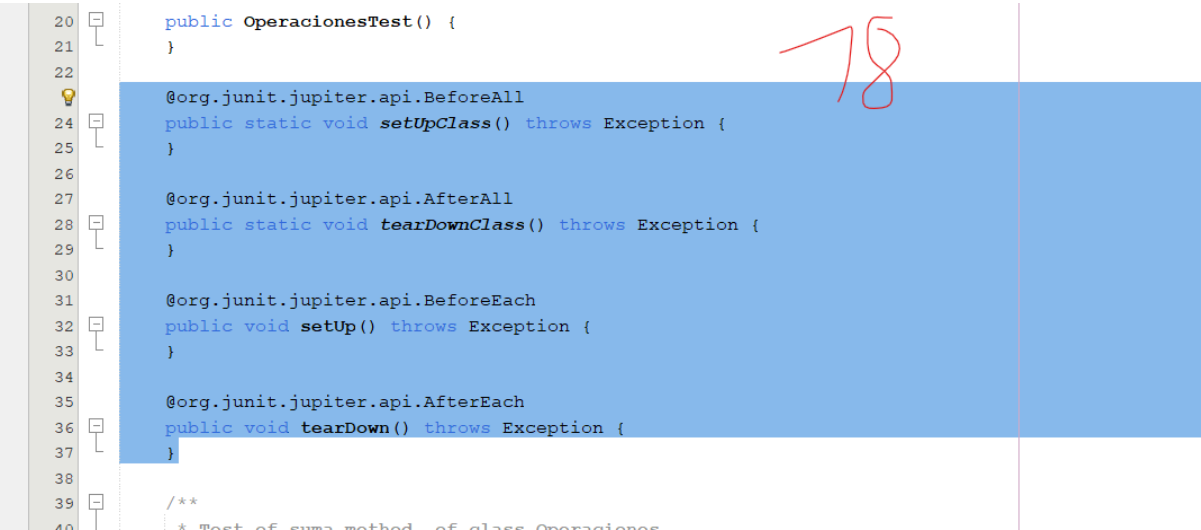
17) Volvemos a crear una clase Test para la clase Operaciones, cuyo código hemos copiado y pegado tal como se nos ha dado en el documento.



```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/1
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/C1
4  */
5  package com.mycompany.complejos;
6
```

1-7

18) Debemos tener presente eliminar primero estas clases antes que nada.



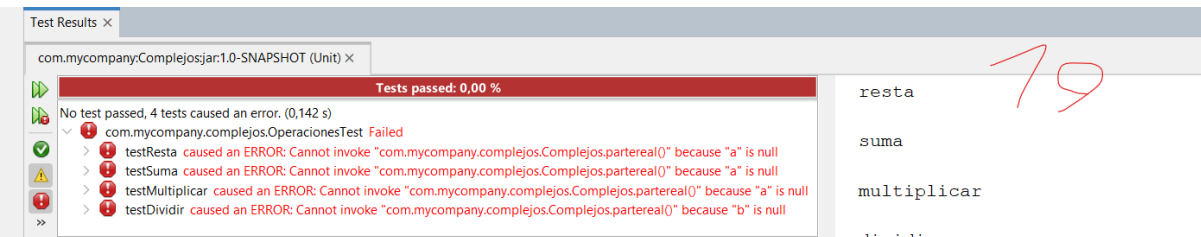
```

20 public OperacionesTest() {
21 }
22
23 @org.junit.jupiter.api.BeforeAll
24 public static void setUpClass() throws Exception {
25 }
26
27 @org.junit.jupiter.api.AfterAll
28 public static void tearDownClass() throws Exception {
29 }
30
31 @org.junit.jupiter.api.BeforeEach
32 public void setUp() throws Exception {
33 }
34
35 @org.junit.jupiter.api.AfterEach
36 public void tearDown() throws Exception {
37 }
38
39 /**
40  * Rest of sum method of class Operaciones

```

Handwritten annotations: 18 (pointing to setUpClass), 19 (pointing to setUp), 20 (pointing to new Complejos(3,4)).

19) En este caso, todos los tests son fallidos. Este problema es más serio, ya que se nos da un error de **NullPointerException**. Esto se debe a que estamos llamando a direcciones vacías de memoria:



Test Results window showing 0 tests passed and 4 tests failed. The failed tests are:

- testResta caused an ERROR: Cannot invoke "com.mycompany.complejos.Complejos.parereal()" because "a" is null
- testSuma caused an ERROR: Cannot invoke "com.mycompany.complejos.Complejos.parereal()" because "a" is null
- testMultiplicar caused an ERROR: Cannot invoke "com.mycompany.complejos.Complejos.parereal()" because "a" is null
- testDividir caused an ERROR: Cannot invoke "com.mycompany.complejos.Complejos.parereal()" because "b" is null

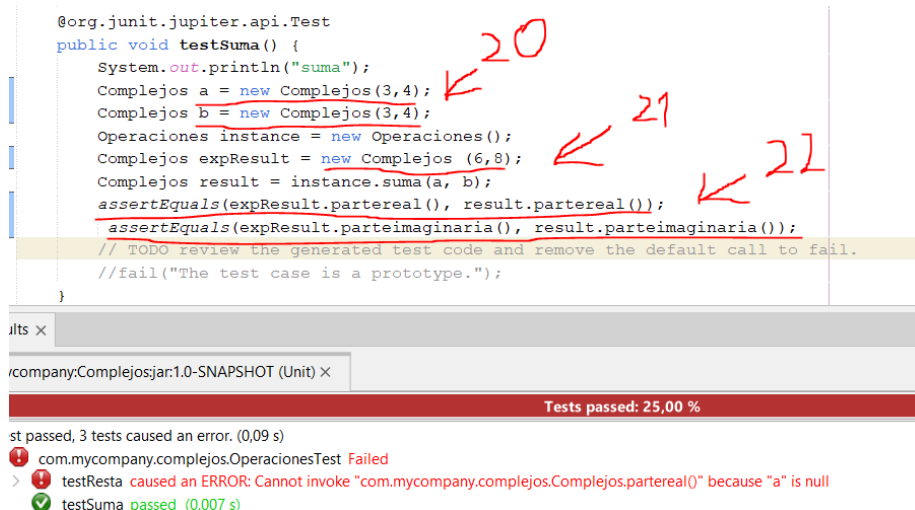
Handwritten annotation: 19 (pointing to the error messages).

20) Para solucionar el problema, lo primero es crear los objetos cuyos valores vamos a sumar entre sí (a con a y b con b).

21) Creamos el objeto con el que los vamos a comparar.

22) Usamos assertEquals para comparar que la suma de a entre ambos objetos es correcta, y otra vez para hacer lo mismo con b.

Vemos que la clase testSuma esta vez es correcta.



```

@org.junit.jupiter.api.Test
public void testSuma() {
    System.out.println("suma");
    Complejos a = new Complejos(3,4);
    Complejos b = new Complejos(3,4);
    Operaciones instance = new Operaciones();
    Complejos expResult = new Complejos (6,8);
    Complejos result = instance.suma(a, b);
    assertEquals(expResult.parereal(), result.parereal());
    assertEquals(expResult.parteimaginaria(), result.parteimaginaria());
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

```

Handwritten annotations: 20 (pointing to new Complejos(3,4)), 21 (pointing to new Operaciones()), 22 (pointing to assertEquals calls).

Test Results window showing 3 tests passed and 1 test failed (testResta).

23) Hacemos las correcciones pertinentes con el método **testResta**:

```

41  /**
42   * Test of resta method, of class Operaciones.
43   */
44  @org.junit.jupiter.api.Test
45  public void testResta() {
46      System.out.println("resta");
47      Complejos a = new Complejos(3,4);
48      Complejos b = new Complejos(3,4);
49      Operaciones instance = new Operaciones();
50      Complejos expectedResult = new Complejos(0,0);
51      Complejos result = instance.resta(a, b);
52      assertEquals(expResult.partereal(), result.partereal());
53      assertEquals(expResult.parteimaginaria(), result.parteimaginaria());
54      // TODO review the generated test code and remove the default call to fail.
55      //fail("The test case is a prototype.");
56  }

```

Test Results X

com.mycompany:Complejos:jar:1.0-SNAPSHOT (Unit) X

Tests passed: 50,00 %

2 tests passed, 2 tests caused an error. (0,088 s)

- com.mycompany.complejos.OperacionesTest Failed
 - testResta passed (0,043 s)
 - testSuma passed (0,001 s)
 - testMultiplicar caused an ERROR: Cannot invoke "com.mycompany.complejos.Complejos.partereal()" because "a" is null
 - testDividir caused an ERROR: Cannot invoke "com.mycompany.complejos.Complejos.partereal()" because "b" is null

24) En el caso del método **testMultiplicar**, existe otro problema. Vemos que los valores a de ambos objetos no se multiplican entre sí correctamente, pues el resultado de la máquina no se ajusta al esperado, que debería ser 25, 24:

```

58  /**
59   * Test of multiplicar method, of class Operaciones.
60   */
61  @org.junit.jupiter.api.Test
62  public void testMultiplicar() {
63      System.out.println("multiplicar");
64      Complejos a = new Complejos(3, 4);
65      Complejos b = new Complejos(3, 4);
66      Operaciones instance = new Operaciones();
67      Complejos expectedResult = new Complejos(25, 24);
68      Complejos result = instance.multiplicar(a, b);
69      assertEquals(expResult.partereal(), result.partereal());
70      assertEquals(expResult.parteimaginaria(), result.parteimaginaria());
71      // TODO review the generated test code and remove the default call to fail.
72      //fail("The test case is a prototype.");
73  }

```

Test Results X

com.mycompany:Complejos:jar:1.0-SNAPSHOT (Unit) X

Tests passed: 50,00 %

2 tests passed, 1 test failed, 1 test caused an error. (0,088 s)

- com.mycompany.complejos.OperacionesTest Failed
 - testResta passed (0,039 s)
 - testSuma passed (0,001 s)
 - testMultiplicar Failed: expected: <25,0> but was: <-7,0>
 - testDividir caused an ERROR: Cannot invoke "com.mycompany.complejos.Complejos.partereal()" because "b" is null

25) La fórmula es incorrecta, solo basta con sustituir los - por +:

```

36 public Complejos multiplicar(Complejos a, Complejos b) {
37     float r;
38     float im;
39
40     r = a.partereal() * b.partereal() - a.parteimaginaria() * b.parteimaginaria();
41     im = a.partereal() * b.parteimaginaria() - b.partereal() * a.parteimaginaria();
42
43     Complejos c = new Complejos(r, im);
44
45     return c;
46 }

```

25

26) Ahora, el resultado se ajusta a lo esperado:

```

59 /**
60  * @org.junit.jupiter.api.Test
61  * public void testMultiplicar() {
62  *     System.out.println("multiplicar");
63  *     Complejos a = new Complejos(3, 4);
64  *     Complejos b = new Complejos(3, 4);
65  *     Operaciones instance = new Operaciones();
66  *     Complejos expectedResult = new Complejos(25, 24);
67  *     Complejos result = instance.multiplicar(a, b);
68  *     assertEquals(expectedResult.partereal(), result.partereal());
69  *     assertEquals(expectedResult.parteimaginaria(), result.parteimaginaria());
70  *     // TODO review the generated test code and remove the default call to fail.
71  *     //fail("The test case is a prototype.");
72  * }
73  */

```

26

Test Results X

com.mycompany:Complejosjar:1.0-SNAPSHOT (Unit) X

Tests passed: 75.00 %

3 tests passed, 1 test caused an error. (0.093 s)

- com.mycompany.complejos.OperacionesTest Failed
 - testResta passed (0.041 s)
 - testSuma passed (0.002 s)
 - testMultiplicar passed (0.001 s)

27) En el método testDividir tenemos doble fallo:

```

74 /**
75  * Test of dividir method, of class Operaciones.
76  */
77 @org.junit.jupiter.api.Test
78 public void testDividir() {
79     System.out.println("dividir");
80     Complejos a = new Complejos(3, 4);
81     Complejos b = new Complejos(3, 4);
82     Operaciones instance = new Operaciones();
83     Complejos expectedResult = new Complejos(0, 1);
84     Complejos result = instance.dividir(a, b);
85     assertEquals(expectedResult.partereal(), result.partereal());
86     assertEquals(expectedResult.parteimaginaria(), result.parteimaginaria());
87
88     // TODO review the generated test code and remove the default call to fail.
89     //fail("The test case is a prototype.");
90 }

```

27

Test Results X

com.mycompany:Complejosjar:1.0-SNAPSHOT (Unit) X

Tests passed: 75.00 %

3 tests passed, 1 test failed. (0.091 s)

- com.mycompany.complejos.OperacionesTest Failed
 - testResta passed (0.041 s)
 - testSuma passed (0.002 s)
 - testMultiplicar passed (0.001 s)
 - testDividir Failed: expected: <0.0> but was: <1.0>

28) El fallo es, por un lado, que los términos de la parte imaginaria están invertidos, y por otro le falta controlar la excepción de la división entre 0, aunque esto último en sí mismo no es el motivo del fallo..

```

46  }
47
48  public Complejos dividir(Complejos a, Complejos b) {
49      float r;
50      float im;
51      float d = b.partereal() * b.partereal() + b.parteimaginaria() * b.parteimaginaria();
52
53      r = a.partereal() * b.partereal() + a.parteimaginaria() * b.parteimaginaria();
54      im = a.partereal() * b.parteimaginaria() - b.partereal() * a.parteimaginaria();
55      r = r / d;
56      im = im / d;
57
58      Complejos c = new Complejos(r, im);
59
60      return c;
61  }
62
63  }
64

```

28

29) Inviertiendo el orden de los términos deberíamos tener la corrección hecha:

```

public Complejos dividir(Complejos a, Complejos b) {
    float r;
    float im;
    float d = b.partereal() * b.partereal() + b.parteimaginaria() * b.parteimaginaria();

    r = a.partereal() * b.partereal() + a.parteimaginaria() * b.parteimaginaria();
    im = a.parteimaginaria() * b.partereal() - a.partereal() * b.parteimaginaria(); // Corrección aquí
    r = r / d;
    im = im / d;

    Complejos c = new Complejos(r, im);

    return c;
}

```

29

30) (Hemos cambiado los valores a 10 y 20 para que se vea más claro) Antes de la corrección, el resultado nos aparecía como número negativo:

```

77  @org.junit.jupiter.api.Test
78  public void testDividir() {
79      System.out.println("dividir");
80      Complejos a = new Complejos(10, 20);
81      Complejos b = new Complejos(20, 10);
82      Operaciones instance = new Operaciones();
83      Complejos expectedResult = new Complejos(0.8f, 0.6f);
84      Complejos result = instance.dividir(a, b);
85      assertEquals(expResult.partereal(), result.partereal());
86      assertEquals(expResult.parteimaginaria(), result.parteimaginaria());
87
88      // TODO review the generated test code and remove the default call to fail.
89      //fail("The test case is a prototype.");
90  }
91

```

30

Test Results ×

com.mycompany:Complejos.jar:1.0-SNAPSHOT (Unit) ×

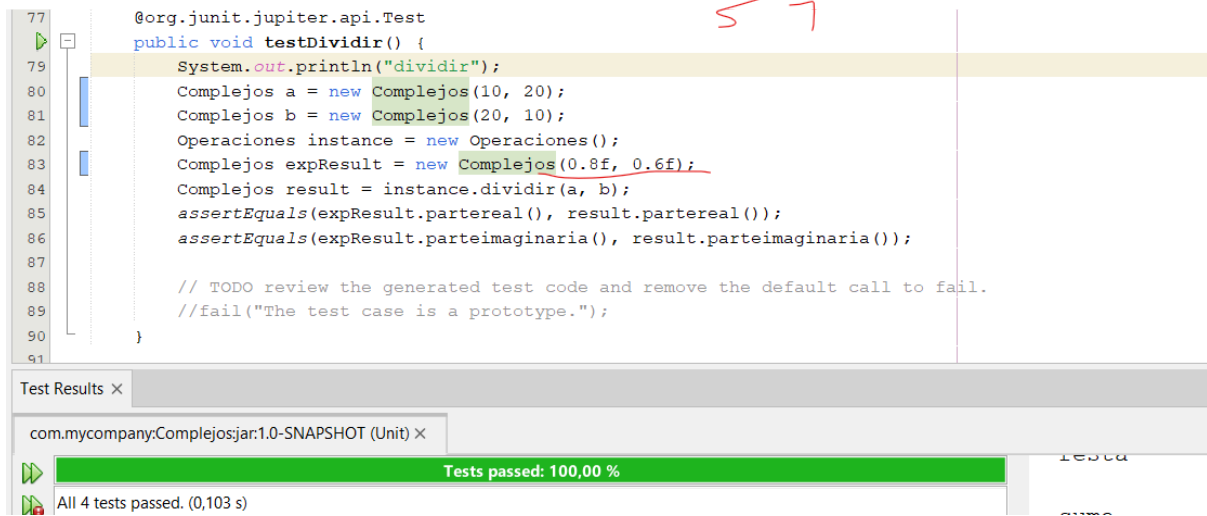
Tests passed: 75,00 %

3 tests passed, 1 test failed. (0,157 s)

com.mycompany.complejos.OperacionesTest Failed

> testDividir Failed: expected: <0.6> but was: <-0.6>

31) Después de la corrección, el resultado es correcto y podemos volver a enamorarnos:



The screenshot shows an IDE with a Java test method `testDividir()` and its execution results. The code is as follows:

```
77  @org.junit.jupiter.api.Test
78  public void testDividir() {
79      System.out.println("dividir");
80      Complejos a = new Complejos(10, 20);
81      Complejos b = new Complejos(20, 10);
82      Operaciones instance = new Operaciones();
83      Complejos expResult = new Complejos(0.8f, 0.6f);
84      Complejos result = instance.dividir(a, b);
85      assertEquals(expResult.partereal(), result.partereal());
86      assertEquals(expResult.parteimaginaria(), result.parteimaginaria());
87
88      // TODO review the generated test code and remove the default call to fail.
89      //fail("The test case is a prototype.");
90  }
91
```

Handwritten red annotations include a checkmark and the number '7' above the code, and a red underline under the `Complejos(0.8f, 0.6f)` constructor call on line 83.

The Test Results window shows the following information:

- com.mycompany:Complejos.jar:1.0-SNAPSHOT (Unit) x
- Tests passed: 100,00 %
- All 4 tests passed. (0,103 s)