



JavaTM

JUnit5

Formadora: Calletana López Baleta

JUnit

JUnit se trata de un **Framework Open Source** para la **automatización de las pruebas (tanto unitarias, como de integración) en los proyectos Software**. El framework provee al usuario de herramientas, clases y métodos que le facilitan la tarea de realizar pruebas en su sistema y así asegurar su consistencia y funcionalidad.

JUnit 5 requiere Java 8 (o superior) en tiempo de ejecución.

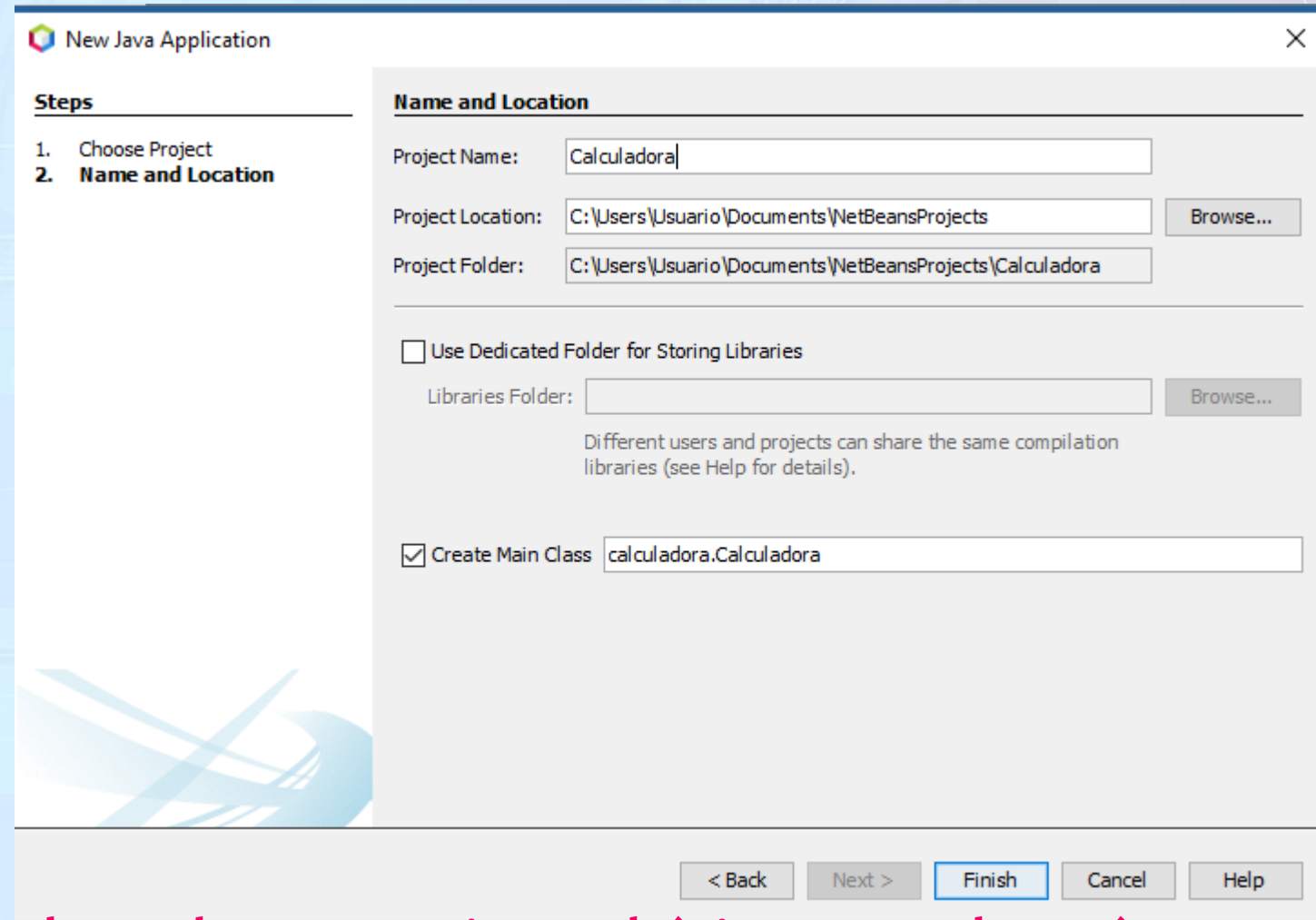
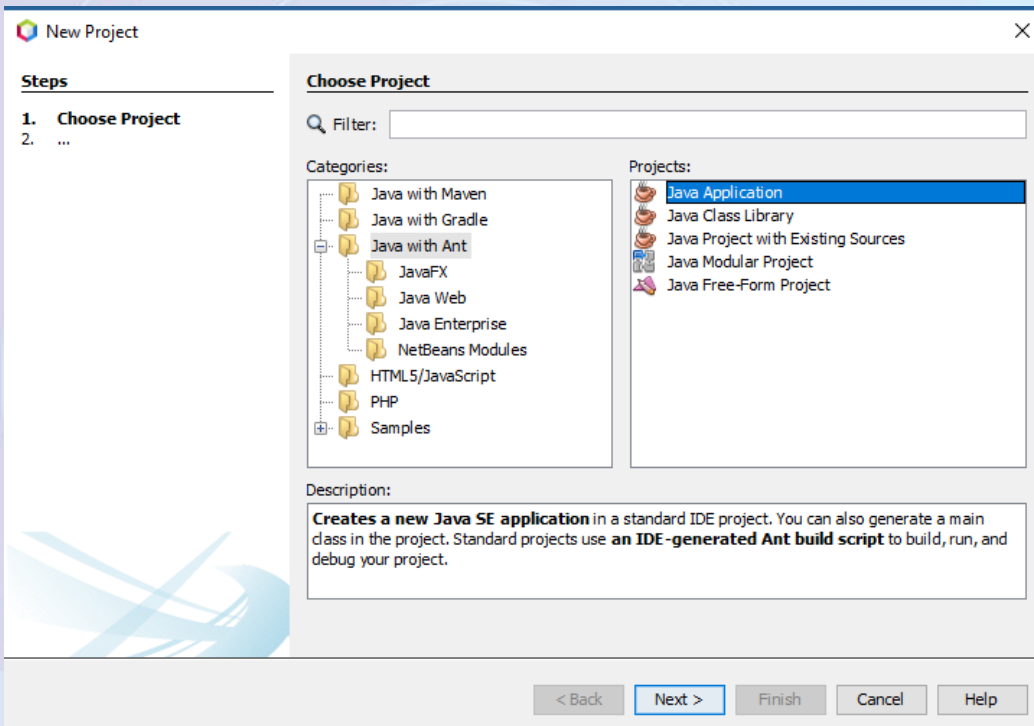
JUnit 5 = *Plataforma JUnit + JUnit Júpiter + JUnit Vintage*

Página
Oficial

<https://junit.org/junit5/>

Implementar JUnit 5 en Netbeans

1. Crear un proyecto con Java Ant



Ejemplo: Crear una calculadora que haga las operaciones básicas con dos números, cada una de las operaciones en una clase y métodos individuales.

2. Crear las clases Suma y Resta con los atributos numero1, numero 2, y todos sus constructores y métodos.

```
12 public class Suma {
13     private int numero1;
14     private int numero2;
15
16     public Suma() {
17     }
18
19     public Suma(int numero1, int numero2) {
20         this.numero1 = numero1;
21         this.numero2 = numero2;
22     }
23
24     public int getNumero1() {
25         return numero1;
26     }
27
28     public void setNumero1(int numero1) {
29         this.numero1 = numero1;
30     }
31
32     public int getNumero2() {
33         return numero2;
34     }
35
36     public void setNumero2(int numero2) {
37         this.numero2 = numero2;
38     }
39
40     @Override
41     public String toString() {
42         return "Suma{" + "numero1=" + numero1 + ", numero2=" + numero2 + '}';
43     }
44
45
46 }
```

2. Crear las clases Suma y Resta con los atributos numero1, numero 2, y todos sus constructores y métodos.

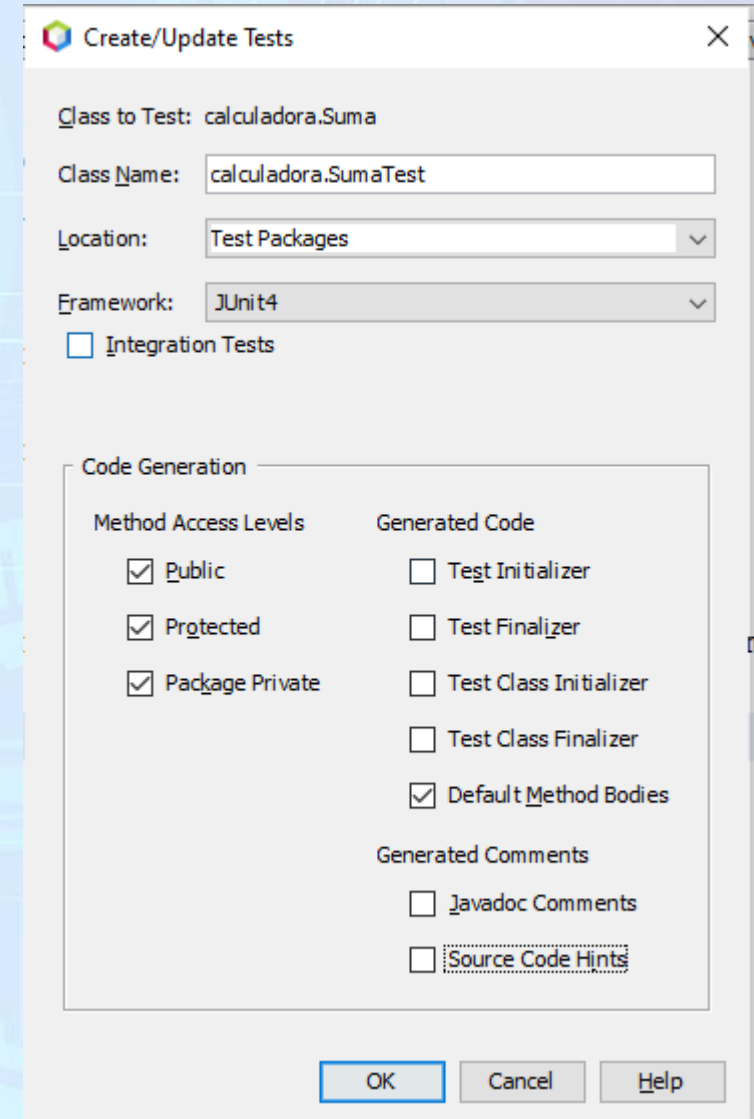
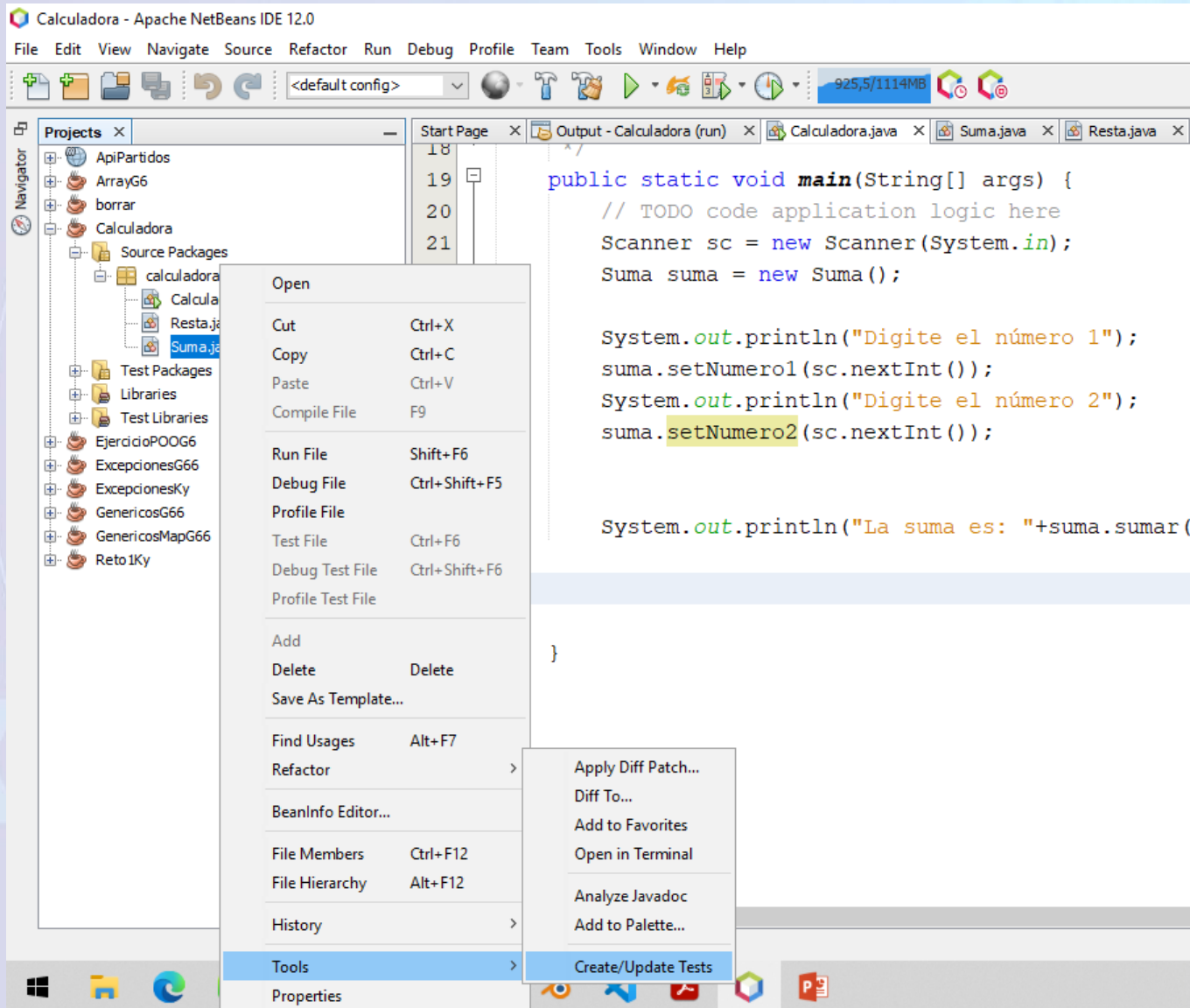
```
12 public class Resta {  
13     private int numero1;  
14     private int numero2;  
15  
16     public Resta() {  
17     }  
18  
19     public Resta(int numero1, int numero2) {  
20         this.numero1 = numero1;  
21         this.numero2 = numero2;  
22     }  
23  
24     public int getNumero1() {  
25         return numero1;  
26     }  
27  
28     public void setNumero1(int numero1) {  
29         this.numero1 = numero1;  
30     }  
31  
32     public int getNumero2() {  
33         return numero2;  
34     }  
35  
36     public void setNumero2(int numero2) {  
37         this.numero2 = numero2;  
38     }  
39  
40     @Override  
41     public String toString() {  
42         return "Resta{" + "numero1=" + numero1 + ", numero2=" + numero2 + '}';  
43     }  
44  
45 }  
46
```

3. En la clase principal hacer el llamado y enviar atributos o parámetros

```
19 public static void main(String[] args) {  
20     // TODO code application logic here  
21     Scanner sc = new Scanner(System.in);  
22     Suma suma = new Suma();  
23  
24     System.out.println("Digite el número 1");  
25     suma.setNumero1(sc.nextInt());  
26     System.out.println("Digite el número 2");  
27     suma.setNumero2(sc.nextInt());  
28  
29  
30     System.out.println("La suma es: "+suma.sumar(suma.getNumero1(), suma.getNumero2()));  
31  
32  
33  
34 }  
35  
36 }
```

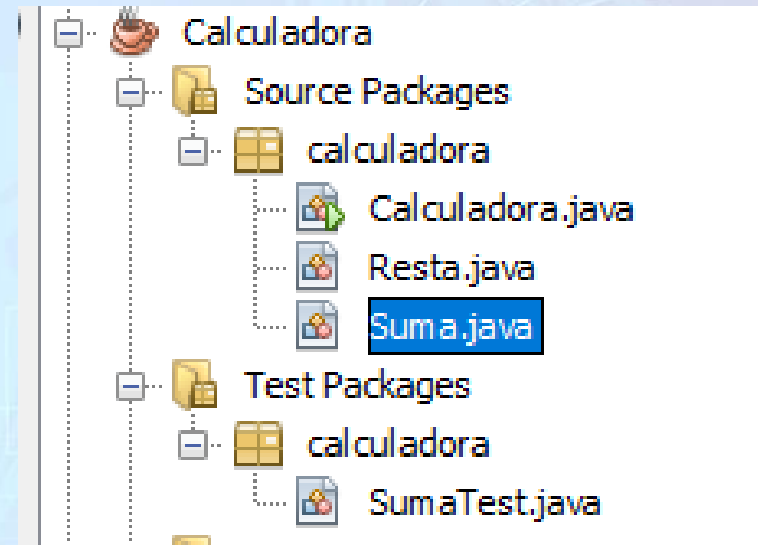

4. Hacer la prueba Unitaria con Junit

Sólo elegir los métodos principales y JUnit 4, ya que en mi caso Netbeans no es compatible para JUnit 5



4. Hacer la prueba Unitaria con Junit

```
6 package calculadora;
7
8 import org.junit.Test;
9 import static org.junit.Assert.*;
10
11 /**
12  *
13  * @author Usuario
14  */
15 public class SumaTest {
16
17     public SumaTest() {
18     }
19
20     @Test
21     public void testSumar() {
22         System.out.println("sumar");
23         int numero1 = 0;
24         int numero2 = 0;
25         Suma instance = new Suma();
26         int expResult = 0;
27         int result = instance.sumar(numero1, numero2);
28         assertEquals(expResult, result);
29         fail("The test case is a prototype.");
30     }
31 }
```



Netbeans automáticamente crea el test de la clase Suma, entonces podemos cambiar los valores a los números 1 y 2, y hacemos el testeo haciendo run sobre la clase que creó SumaTest.java

4. Hacer la prueba Unitaria con Junit

```
8  import org.junit.Test;
9  import static org.junit.Assert.*;
10
11  /**
12   *
13   * @author Usuario
14   */
15  public class SumaTest {
16
17      public SumaTest() {
18      }
19
20      @Test
21      public void testSumar() {
22          System.out.println("sumar");
23          int numero1 = 1;
24          int numero2 = 2;
25          Suma instance = new Suma();
26          int expectedResult = 3;
27          int result = instance.sumar(numero1, numero2);
28          assertEquals(expectedResult, result);
29      }
30  }
31
32
33
34 }
```

Modificamos los valores de numero1 y numero2, y el resultado esperado... y lo testeamos, sólo a sumar, borramos los demás porque no están haciendo operaciones...

4. Hacer la prueba Unitaria con Junit

The screenshot shows the Apache NetBeans IDE 12.0 interface. The main window displays the source code of the `SumaTest` class, which is a JUnit test for the `Suma` class. The code is as follows:

```
7
8 import org.junit.Test;
9 import static org.junit.Assert.*;
10
11 /**
12  *
13  * @author Usuario
14  */
15 public class SumaTest {
16
17     public SumaTest() {
18     }
19
20     @Test
21     public void testSumar() {
22         System.out.println("sumar");
```

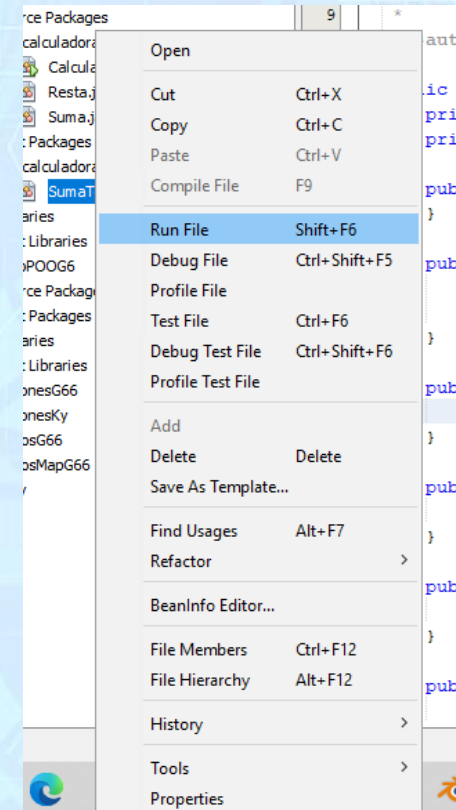
The Test Results window at the bottom shows that the test passed successfully. The output of the test is "sumar".

Debemos mostrar la ventana para que nos muestre que pasamos el test

4. Modificamos la clase suma, y testeamos de nuevo para que nos muestre el error

Si modificamos algo en la clase suma, por ejemplo cambiamos el + por menos (-) y volvemos a testear, nos va a presentar errores

```
Start Page X Output - Calculadora (test) X Calculadora.java X Suma.java X
5  L  */
6  package calculadora;
7
8  /**
9   *
10  * @author Usuario
11  */
12  public class Suma {
13      private int numerol;
14      private int numero2;
15
16      public Suma() {
17      }
18
19      public Suma(int numerol, int numero2) {
20          this.numerol = numerol;
21          this.numero2 = numero2;
22      }
23
24      public int sumar(int numerol, int numero2){
25          return numerol-numero2;
26      }
27
```



4. Modificamos la clase suma, y testeamos de nuevo para que nos muestre el error

The screenshot shows the Apache NetBeans IDE interface. The top menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, and Help. The toolbar contains various icons for file operations, running, and debugging. The left sidebar shows the 'Projects' view with a tree structure: 'Calculadora' (Source Packages: 'calculadora' containing 'Calculadora.java', 'Resta.java', 'Suma.java'; Test Packages: 'calculadora' containing 'SumaTest.java'). The main editor window displays the 'Suma.java' file with the following code:

```
5  /*
6  package calculadora;
7
8  /**
9   *
10  * @author Usuario
11  */
12  public class Suma {
13      private int numero1;
14      private int numero2;
15
16      public Suma() {
17      }
18
19      public Suma(int numero1, int numero2) {
20          this.numero1 = numero1;
21          this.numero2 = numero2;
22      }
23  }
```

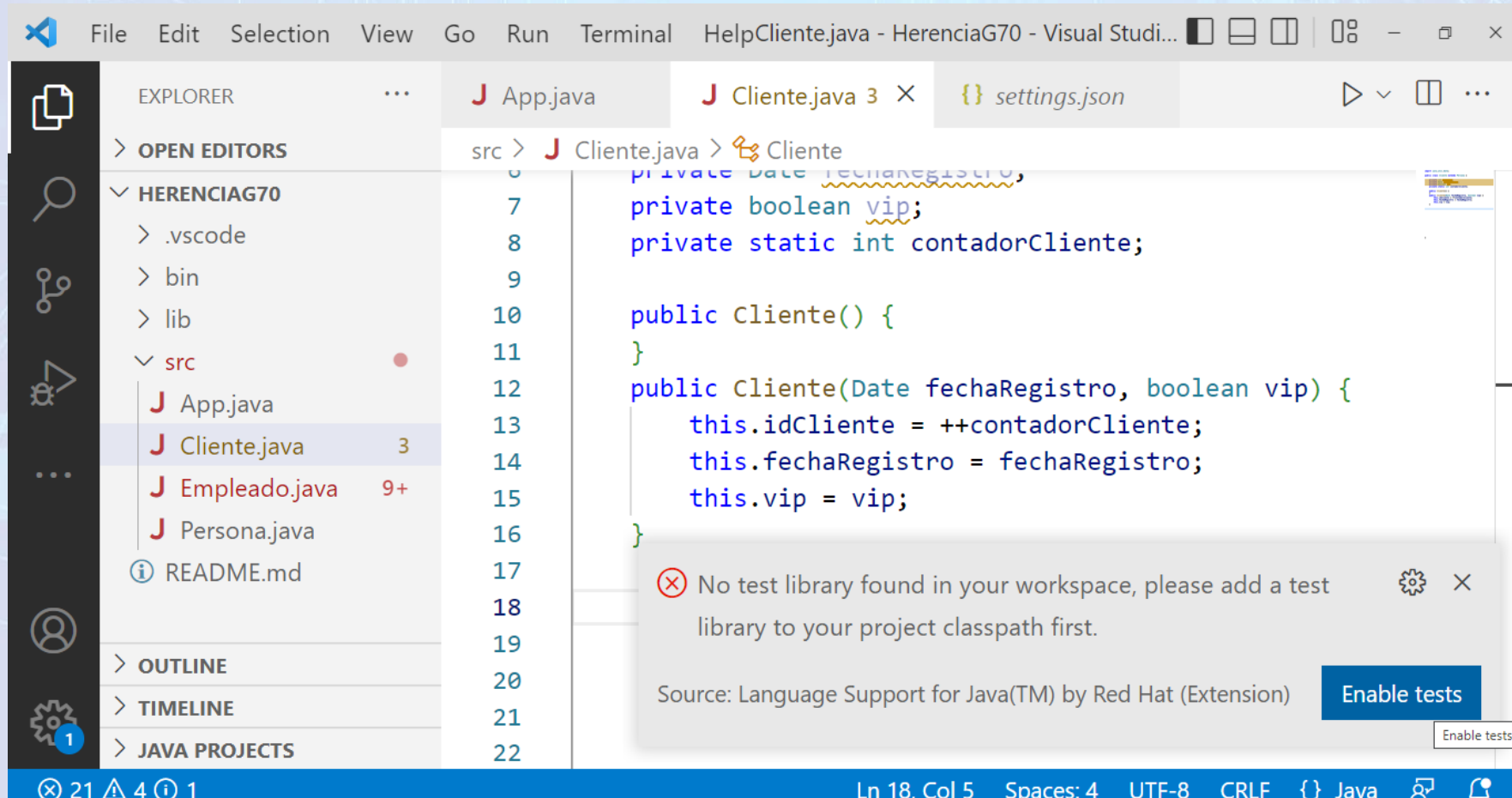
The bottom panel shows the 'Test Results' view for 'calculadora.SumTest'. It indicates 'Tests passed: 0,00 %' and 'No test passed, 1 test failed. (0,219 s)'. The failed test is 'testSumar', with the message 'Failed: expected: <3> but was: <-1>'. The stack trace shows the error occurred in 'SumaTest.java:28'.

On the right side of the bottom panel, the text 'sumar' is displayed.

The background of the slide features a light blue and purple gradient. Overlaid on this are faint, stylized white line drawings of mechanical gears and electronic circuit components, such as resistors and capacitors, arranged in a complex, interconnected pattern.

**Repetimos los pasos para testear la
Clase Resta, Producto y División**

En VSC, donde insertamos el código, nos aparece Generate Test e instalamos la extensión



1. En la clase generar el test

The screenshot shows the Visual Studio Code interface with the 'Cliente.java' file open. A context menu is displayed over the file, listing various actions. The 'Generate Tests...' option is highlighted in blue. To the right, a separate window shows the 'Generate Tests...' menu with its sub-options. At the bottom right, a notification message states: 'No test library found in your workspace, please add a test library to your project classpath first.' with an 'Enable tests' button.

Generate Tests...

- Organize imports Shift+Alt+O
- Generate Getters and Setters...
- Generate Getters...
- Generate Setters...
- Generate Constructors...
- Generate hashCode() and equals()...
- Generate toString()...
- Override/Implement Methods...
- Generate Delegate Methods...
- Change modifiers to final where possible

Context Menu Actions:

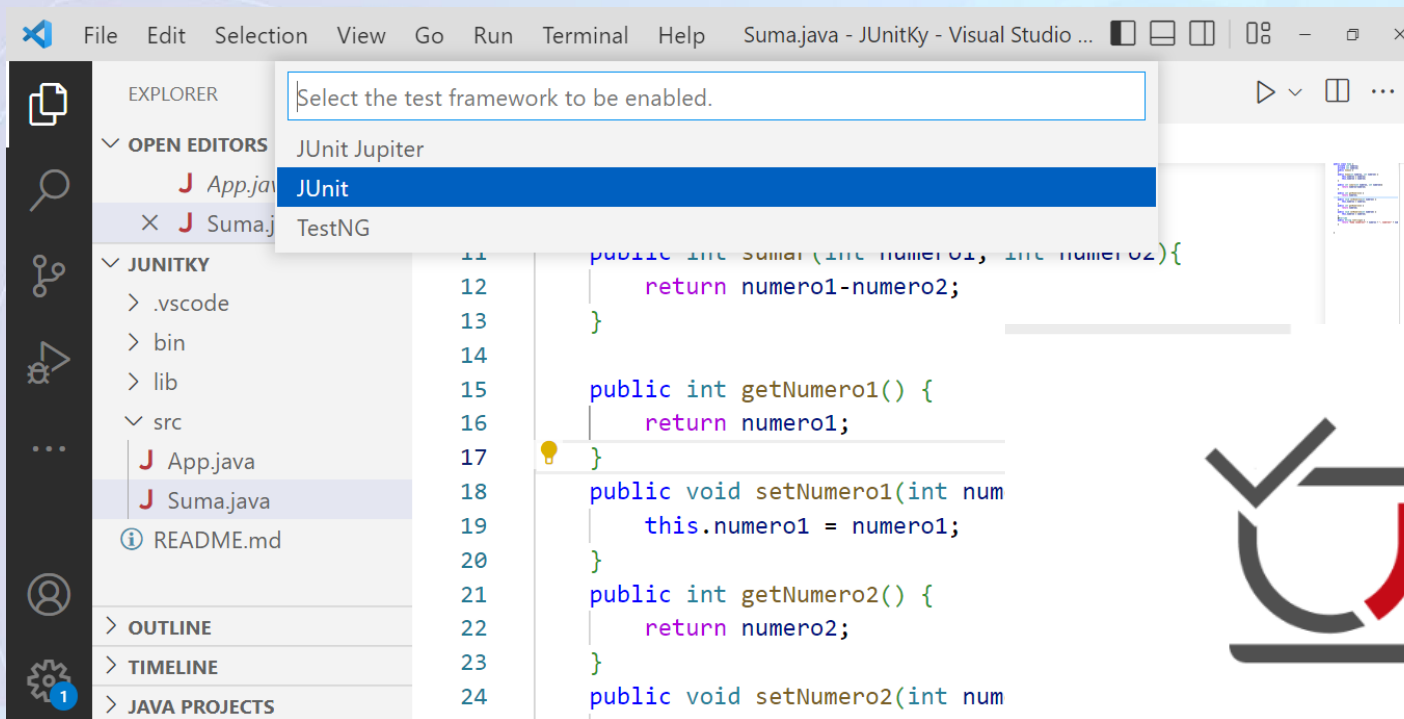
- Find All References Shift+Alt+F12
- Find All Implementations
- Show Call Hierarchy Shift+Alt+H
- Show Type Hierarchy
- Rename Symbol F2
- Change All Occurrences Ctrl+F2
- Format Document Shift+Alt+F
- Format Document With...
- Refactor... Ctrl+Shift+R
- Source Action...**
- Cut Ctrl+X
- Copy Ctrl+C
- Paste Ctrl+V
- Run Java

Notification:

⊗ No test library found in your workspace, please add a test library to your project classpath first.

Source: Language Support for Java(TM) by Red Hat (Extension) [Enable tests](#)

1. En la clase generar el test



Test Runner for Java v0

Microsoft | 14.109.834 | ★★

Run and debug JUnit or TestNG test cases

Disable

Uninstall

Switch to Pre-Releases

This extension is enabled globally.

Details

Feature Contributions

Changelog

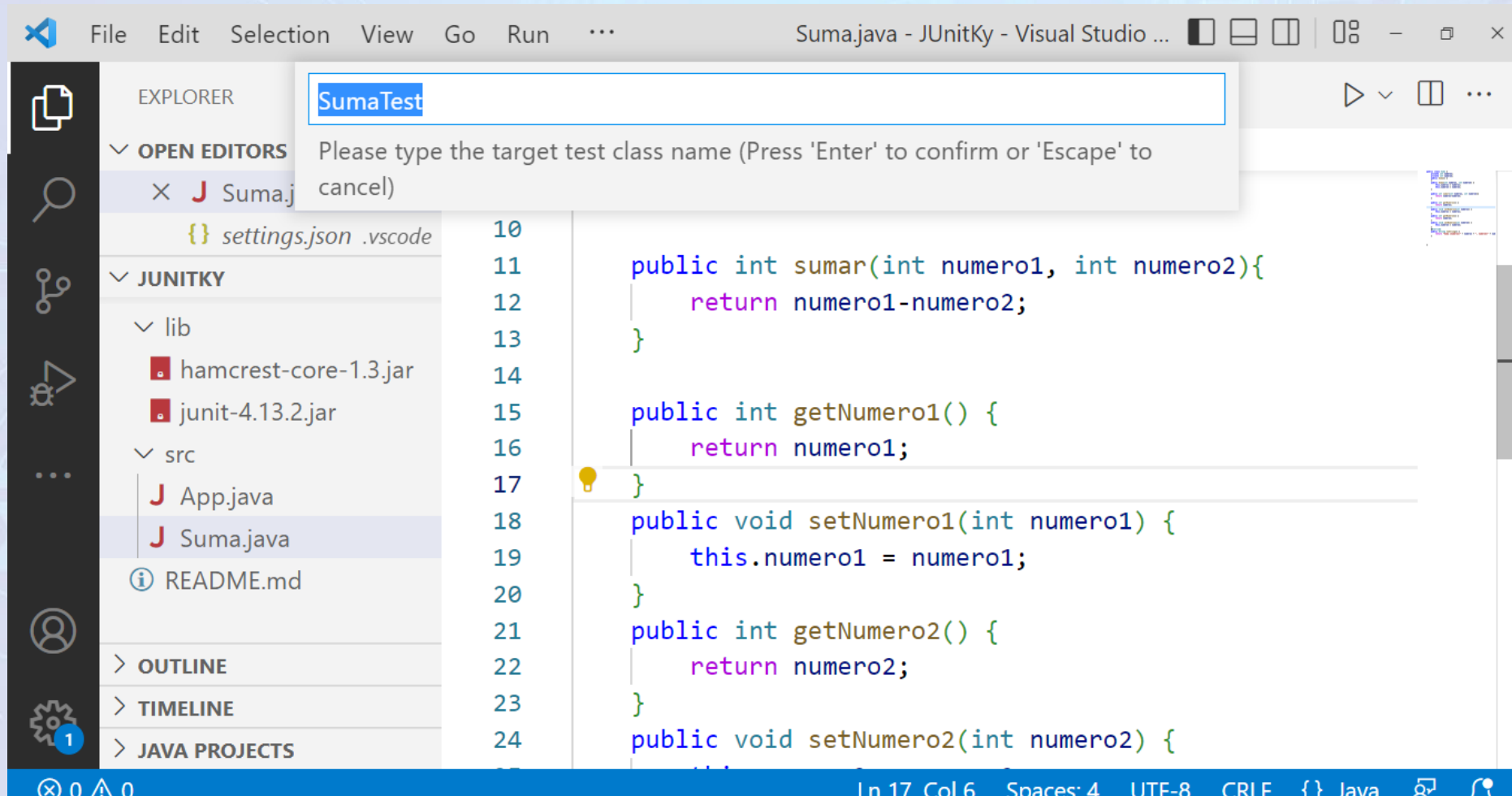
Dependencies

Test Runner for
Java

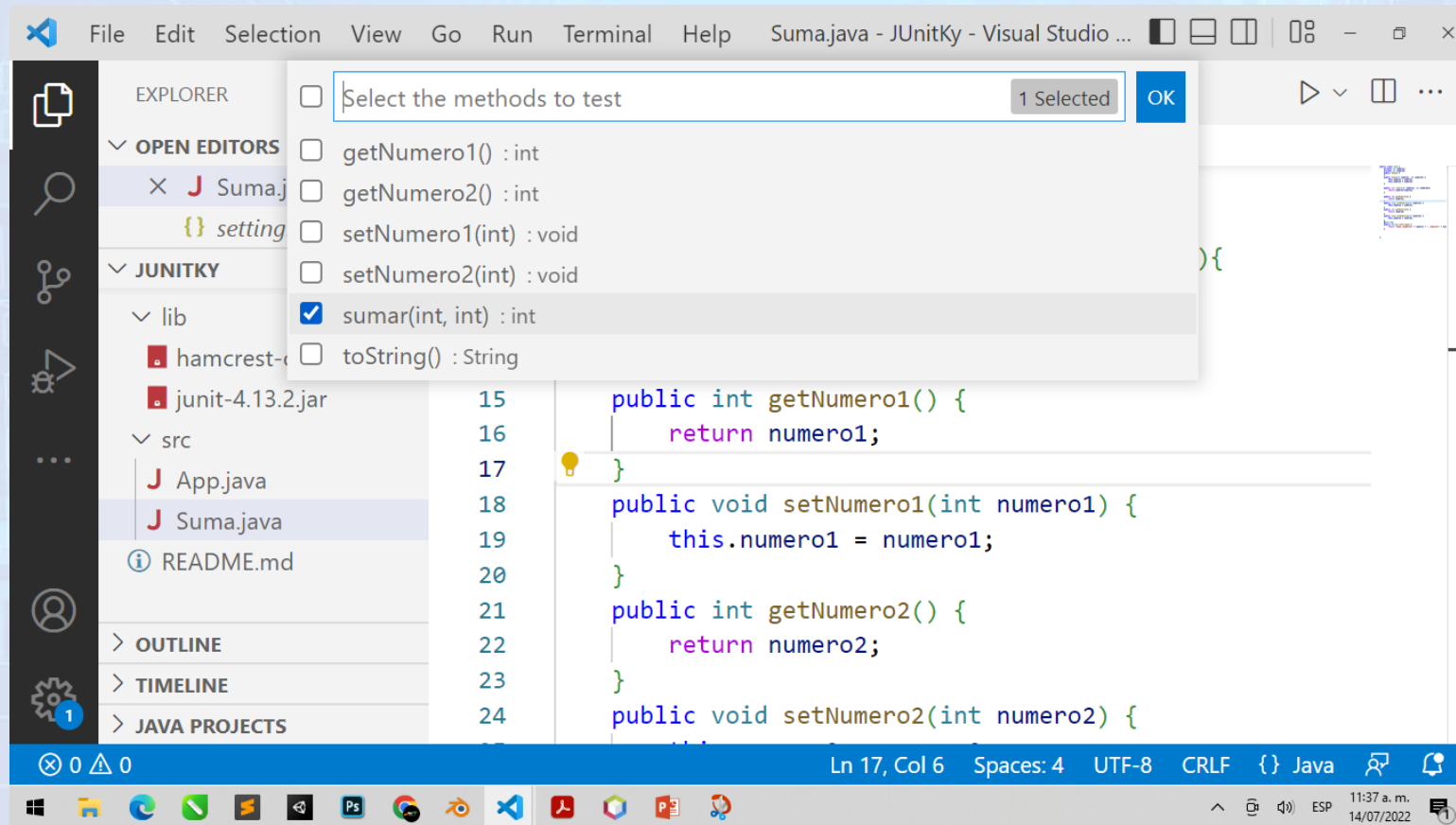
Categories

Test libraries have been downloaded into 'lib/'

2. Instalamos la extensión y luego volvemos a repetir los pasos para crear el test de Suma



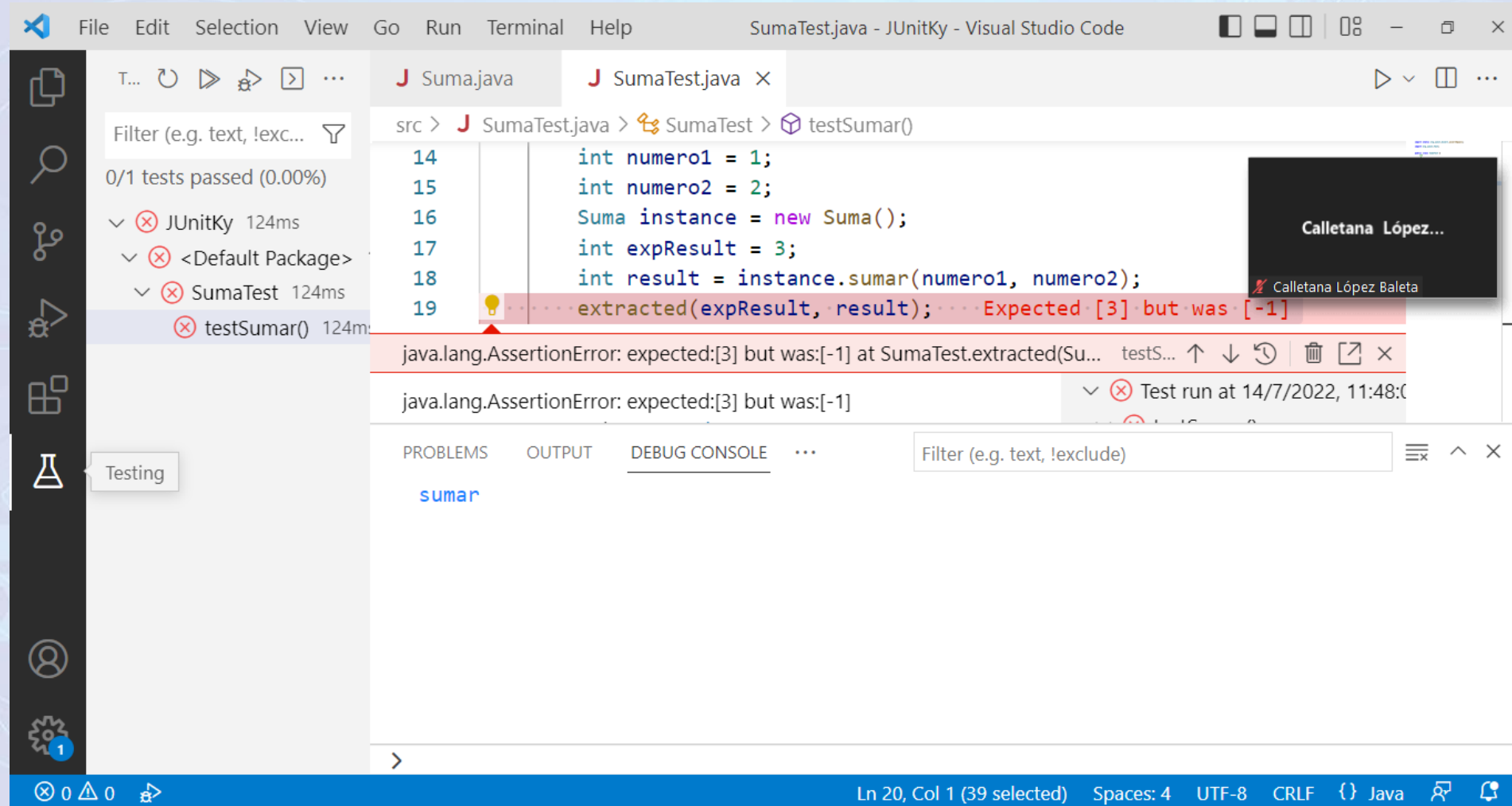
3. Seleccionamos a quien va a realizar el test, en éste caso al método sumar



4. Copiamos el código porque éste no lo genera automáticamente

```
1  import static org.junit.Assert.assertEquals;
2
3  import org.junit.Test;
4
5  public class SumaTest {
6      /**
7       *
8       */
9      @Test
10     public void testSumar() {
11
12         System.out.println("sumar");
13         int numero1 = 1;
14         int numero2 = 2;
15         Suma instance = new Suma();
16         int expectedResult = 3;
17         int result = instance.sumar(numero1, numero2);
18         assertEquals(expectedResult, result);
19     }
20 }
21 }
```

4. En texting debajo de extensiones buscamos el del test que creamos y le damos play



4. Y si tenemos todo bien ya nos va a aparecer en Chulitos

The screenshot shows the Visual Studio Code interface with the following components:

- Menu Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Tab Bar:** SumaTest.java - JUnitKy - Visual Studio Code.
- Editor:** Contains the code for `SumaTest.java`. The code is as follows:

```
src > J SumaTest.java > SumaTest > testSumar()
14     int numero1 = 1;
15     int numero2 = 2;
16     Suma instance = new Suma();
17     int expResult = 3;
18     int result = instance.sumar(numero1, numero2);
19     extracted(expResult, result);
20
21 }
22
```
- Test Explorer (Left Panel):** Shows the test results. It indicates "1/1 tests passed (100%)". The test hierarchy is: JUnitKy (10ms) > <Default Package> > SumaTest (10ms) > testSumar() (10ms). All tests are marked with a green checkmark.
- Output Console (Bottom):** Displays the message: "Please start a debug session to evaluate expressions".
- Search Bar (Bottom Right):** Contains the text "sumar".
- Launch Java Tests - Suma (Bottom Right):** A button to launch the tests.
- Footer:** A black box with the text "Calletana López..." and "Calletana López Baleta".



Gracias....