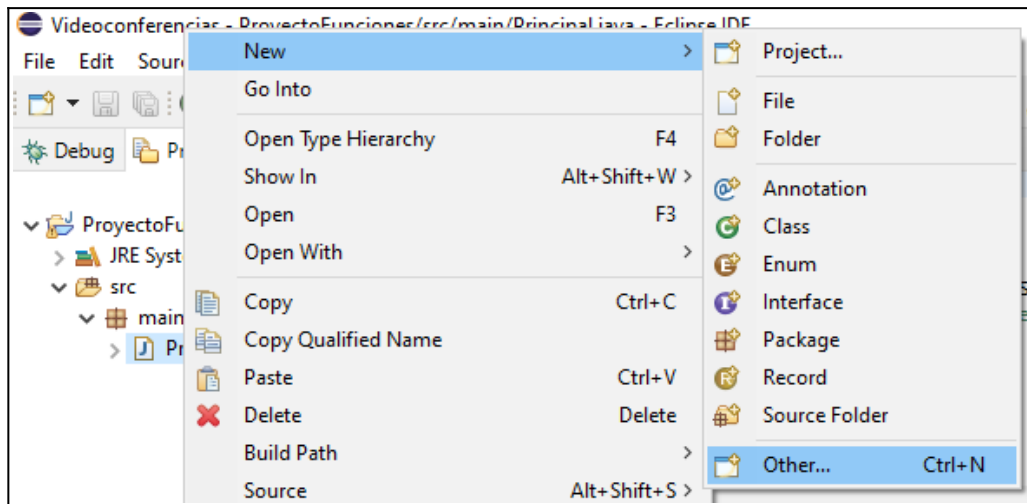


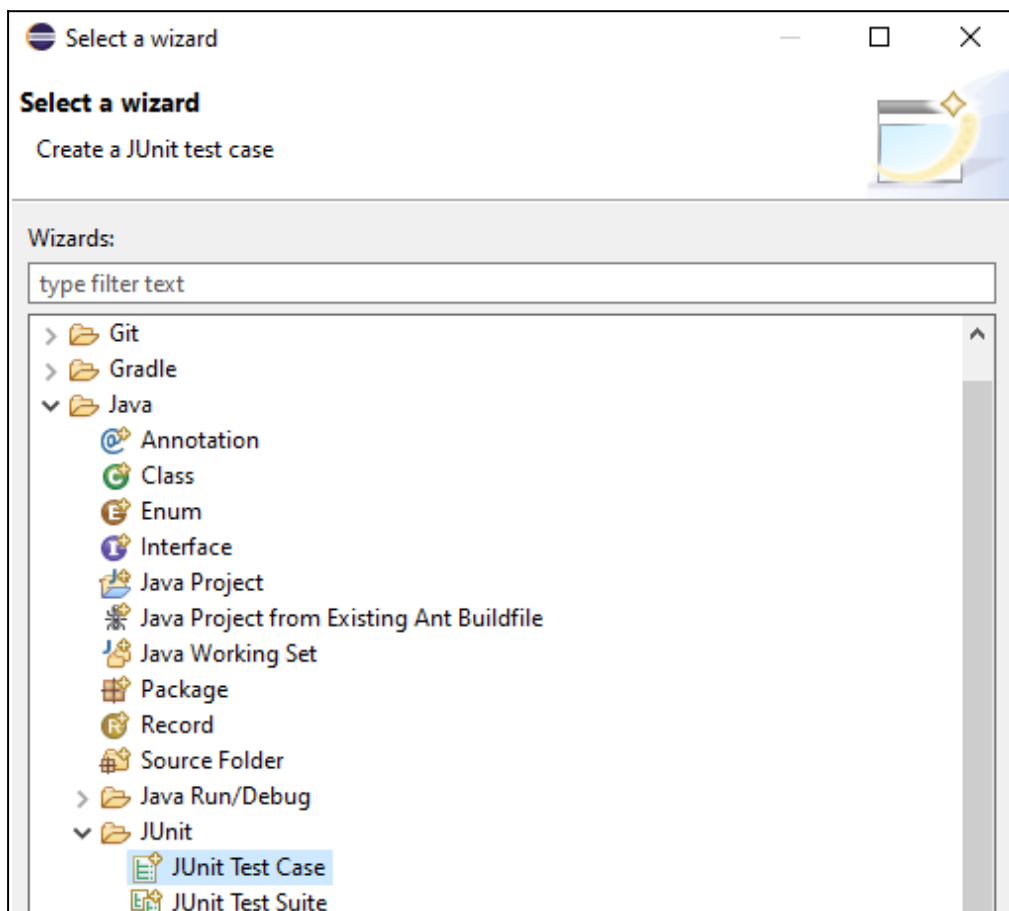
Test Junit 5

Utilizaremos este framework para desarrollar las pruebas unitarias. En primer lugar, en el Eclipse, pulsaremos en el paquete de la clase sobre la que queremos crear el test con el botón derecho y seguiremos los siguientes pasos:

Paso 1: New/Other



Paso 2: Java/JUnit/JUnit Test Case



Paso 3: Rellenar el campo de nombre y pulsar Fin

New JUnit Test Case

JUnit Test Case
Select the name of the new JUnit test case. Specify the class under test to select methods to be tested on the next page.

☐ New JUnit 3 test ☐ New JUnit 4 test ☒ New JUnit Jupiter test

Source folder:

Package:

Name:

Superclass:

Which method stubs would you like to create?

☐ @BeforeAll setUpBeforeClass() ☐ @AfterAll tearDownAfterClass()
☐ @BeforeEach setUp() ☐ @AfterEach tearDown()
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

Class under test:

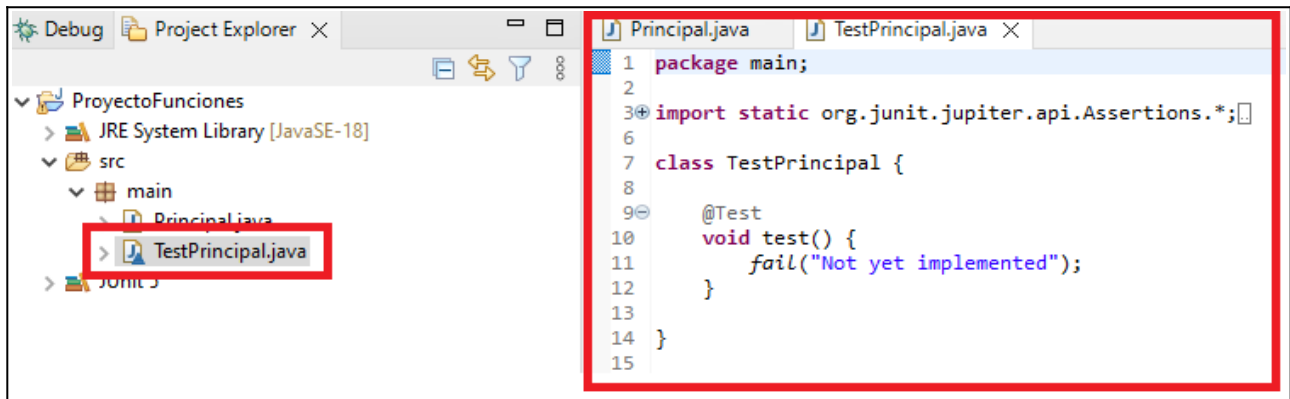
Paso 4: Tan solo pulsamos en OK para añadir la librería de JUnit al proyecto

New JUnit Test Case

JUnit 5 is not on the build path. Do you want to add it?

☐ Not now
☐ Open the build path property page
☒ Perform the following action:

Paso 5: Se nos habrá creado una nueva clase con el nombre que le hemos dado y su contenido mínimo más básico.



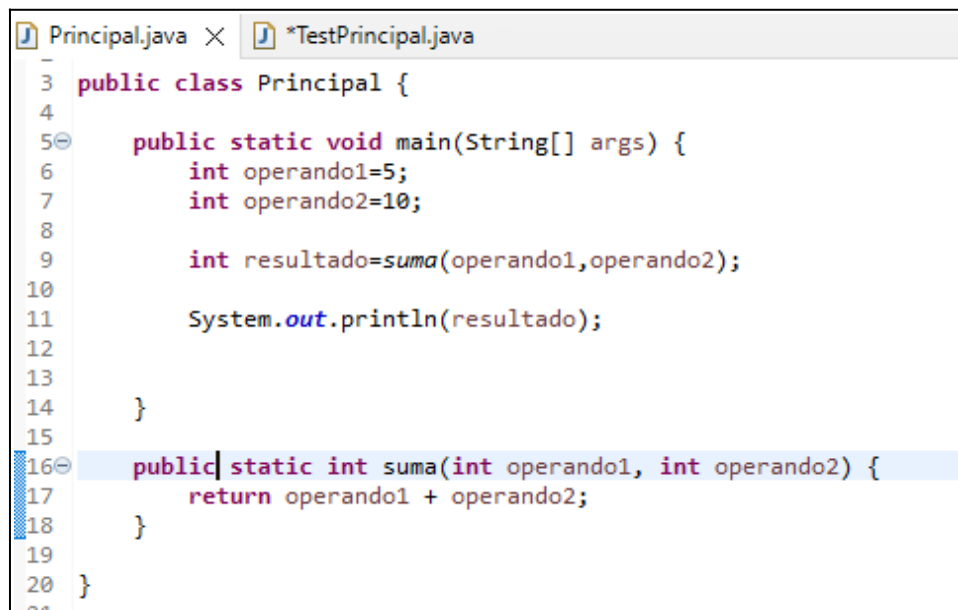
Ahora procederemos a editar dicha clase.

Conviene mencionar que hay distintas etiquetas para poner al principio de las funciones que creemos, pero de momento solo trabajaremos con la etiqueta `@Test`. Esta etiqueta indica que la función que la sigue es una función de test unitario.

Dentro de dicha función podemos llamar a las siguientes funciones, o asserts, que son condiciones de aceptación de los test que desarrollemos. Los que vamos a usar de momento son:

- **assertTrue/assertFalse(condición a testear):** Comprueba que una condición es cierta o falsa. Si usamos el `assertTrue` con una condición que se hace cierta, se pasará el test de forma correcta. Si, por el contrario, utilizamos el `assertFalse` con una condición que se hace cierta, el test resultará incorrecto, y viceversa.
- **assertEquals y assertNotEquals(valor esperado, valor obtenido):** Se pasa el test si el resultado es el mismo en el caso de `assertEquals`, y se pasa el de `assertNotEquals` si el resultado no es el mismo.
- **assertArrayEquals(array esperado, array obtenido):** En caso de que sean iguales ambos arrays el test será superado.

Paso 6: Diseñar el Test. Ejemplo: Partiendo del siguiente programa, que unicamente tiene una función suma, que realiza dicha operación sobre dos operandos

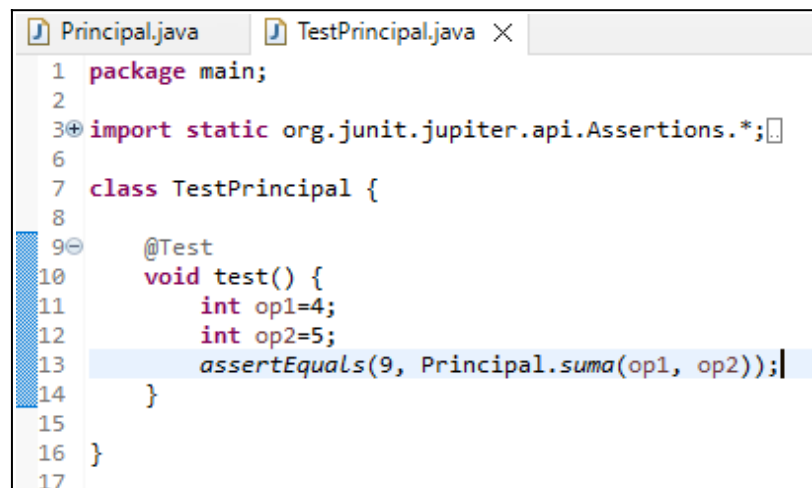


```
Principal.java x *TestPrincipal.java
3 public class Principal {
4
5     public static void main(String[] args) {
6         int operando1=5;
7         int operando2=10;
8
9         int resultado=suma(operando1,operando2);
10
11         System.out.println(resultado);
12
13     }
14
15
16     public static int suma(int operando1, int operando2) {
17         return operando1 + operando2;
18     }
19
20 }
```

Digamos que queremos realizar un test para dicha función suma y comprobar que funciona correctamente para distintas entradas. En este caso podríamos utilizar un assertEquals, ya que queremos comparar un valor correcto con el que nos devuelve el test.

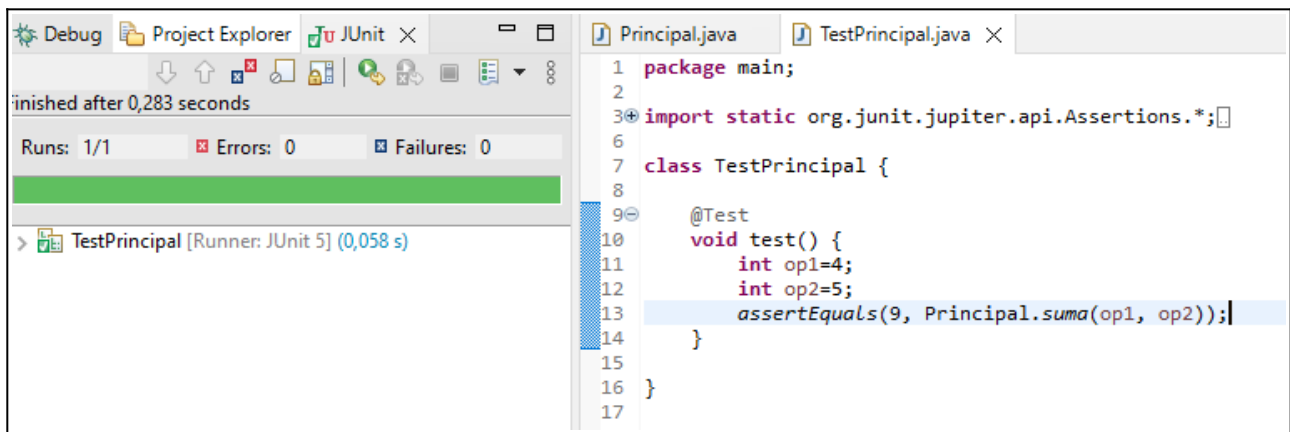
Véase que para llamar a la función suma de nuestra clase Principal hemos de escribir Principal.suma

Esto sirve, como veréis más adelante en Programación, para llamar a una función presente en otra clase.

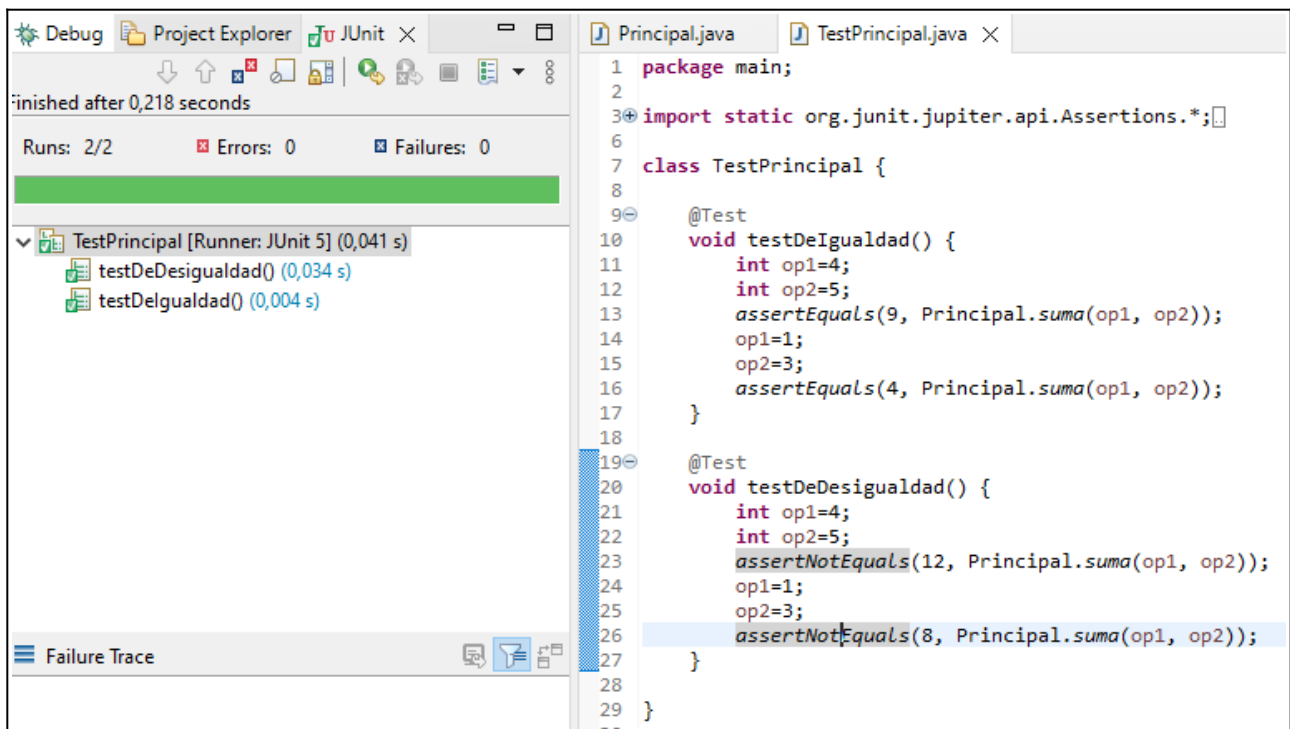


```
Principal.java TestPrincipal.java x
1 package main;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5
6
7 class TestPrincipal {
8
9     @Test
10     void test() {
11         int op1=4;
12         int op2=5;
13         assertEquals(9, Principal.suma(op1, op2));
14     }
15
16 }
17
```

Para ejecutar el test lo hacemos como ejecutaríamos cualquier proyecto. En este caso como los dos operandos suman 9 y el valor esperado es también 9, el test pasará satisfactoriamente:

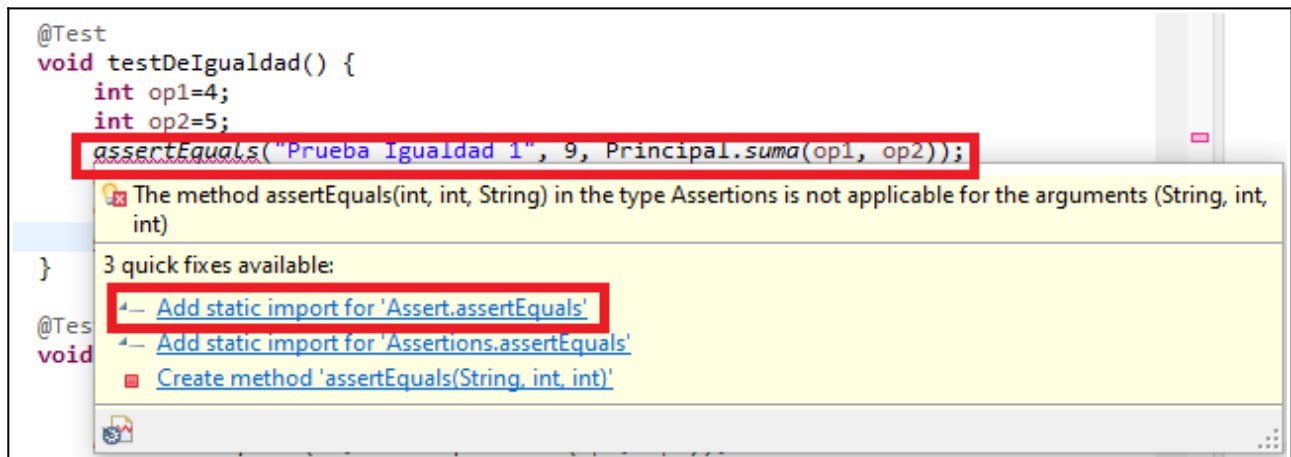


Podemos también concatenar más pruebas en la misma función test, o crear distintas funciones e ir probando. Por ejemplo, a continuación se hacen dos funciones para pruebas unitarias de esta función, unas en las que se comprueba que efectivamente la suma se realiza correctamente para valores que deberían ser iguales, y otra función en la que se comprueba que la suma no da cierto valor para otros valores. Es un simple ejemplo, no hay que buscarle en este caso mucha lógica, pero sí entender el funcionamiento y las posibilidades



También se puede poner un parámetro adicional al principio de nuestros assert con un mensaje que se nos mostrará en caso de no pasar la prueba con éxito. Por ejemplo, para assertEquals(valor Esperado, valor Obtenido) → assertEquals("Cadena de texto a mostrar", valor Esperado, valor Obtenido)

Aquí habría ahora que importar además el primer paquete que se nos ofrece al superponer el ratón sobre el assertEquals:



Ahora ejecutando la prueba con la clase ya modificada e introduciendo un error, podríamos ver un mensaje claro sobre el test o tests que han fallado

