

## Descripción de la Práctica:

La práctica tiene como objetivo introducir a los estudiantes en el uso de JUnit, un marco de pruebas para Java, a través de un ejemplo educativo centrado en la clase `rectangulo1``. En esta actividad, se enseña cómo escribir pruebas unitarias utilizando anotaciones básicas de JUnit para evaluar el correcto funcionamiento de los métodos `area()` y `perimetro()` de la clase `rectangulo1``.

## Requisitos de Instalación:

Para realizar esta práctica, es necesario contar con el siguiente software instalado:

1. IntelliJ IDEA: (se omite el paso si tienes instalado el IntelliJ IDEA)
  - Descarga e instala IntelliJ IDEA desde el sitio oficial: [JetBrains IntelliJ IDEA](https://www.jetbrains.com/idea/download/).
2. Configuración del Proyecto:
  - Abre IntelliJ IDEA y crea un nuevo proyecto Java.
  - Añade la clase `rectangulo1`` al proyecto.
3. Dependencias:
  - Asegúrate de tener configuradas las dependencias de JUnit en tu proyecto. Puedes hacer esto agregando las bibliotecas de JUnit al classpath del proyecto.
4. Estructura del Proyecto:
  - Organiza tu proyecto de manera que tengas una carpeta para las clases y otra para las pruebas (por ejemplo, `src`` y `test``).
5. Copiar el Código de Prueba:
  - Copia y pega el código del test proporcionado en la práctica ( por el profesor) en un nuevo archivo de prueba dentro de la carpeta de pruebas. Asegúrate de que el nombre del archivo termine con `Test`` para que JUnit lo reconozca automáticamente.



## rectangulo1.java

```
package ejemplo;

public class rectangulo1 {

    private int ancho;

    private int alto;

    public rectangulo1(int ancho, int alto) {
        this.ancho = ancho;
        this.alto = alto;
    }

    public int area() {
        return ancho * alto;
    }

    public int perimetro() {
        return 2 * ancho + 2 * alto;
    }

}
```

creamos el primer Test 1:

## RectanguloTest.java

```
package ejemplo;

import org.junit.jupiter.api.Test;

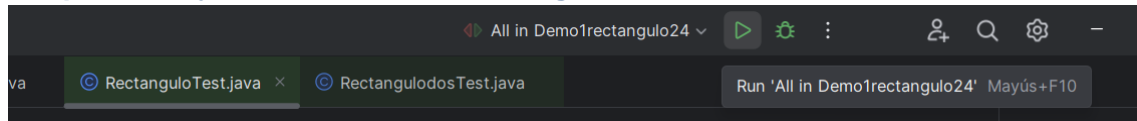
import static org.junit.jupiter.api.Assertions.*;

public class RectanguloTest {

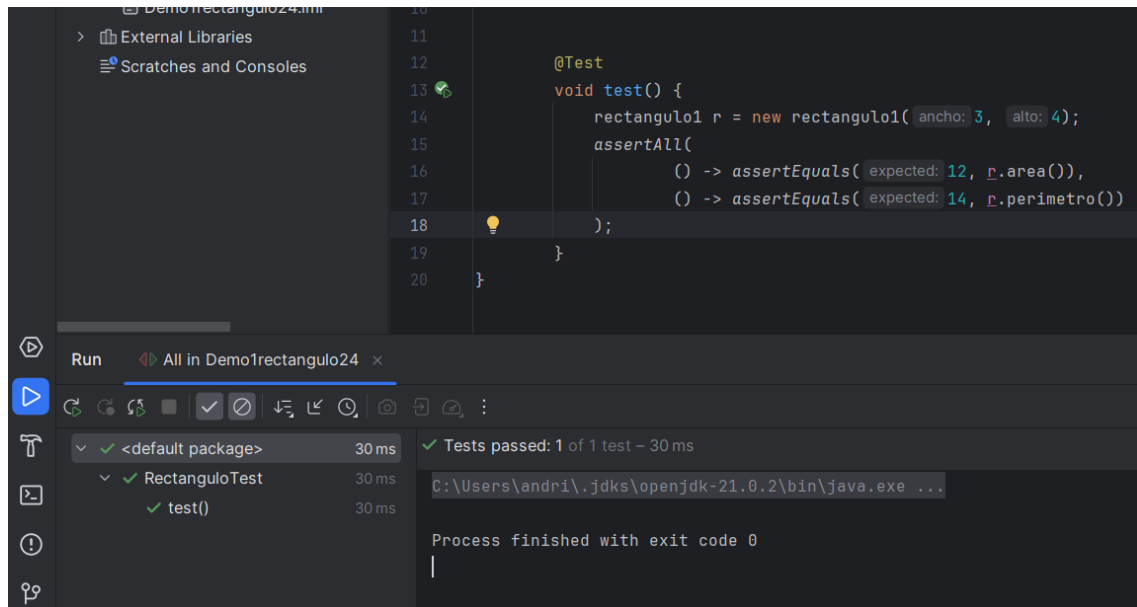
    @Test
    void test() {
        rectangulo1 r = new rectangulo1(3, 4);
        assertEquals(12, r.area());
        assertEquals(14, r.perimetro());
    }

}
```

## Después de ejecutar el test desde triangulo verde "Run ...."



## obtendremos este resultado



## Explicación del test

Este test en JUnit es una prueba para verificar el correcto funcionamiento de la clase `rectangulo1`.

A continuación, se explicará lo que hace el test:

### 1. Anotaciones de JUnit:

- `@Test`: Indica que el método siguiente es una prueba.

### 2. Método de Prueba (`test`):

- Se crea una instancia de la clase `rectangulo1` con un ancho de 3 y un alto de 4:  
`rectangulo1 r = new rectangulo1(3, 4);`

- `assertAll`: Permite agrupar varias aserciones juntas. Todas las aserciones dentro de `assertAll` se evaluarán, y si alguna de ellas falla, se informará como un solo error.

- `assertEquals(12, r.area())`: Asegura que el método `area()` de la instancia `r` devuelva el valor esperado, que en este caso es 12. La fórmula del área para un rectángulo es ancho \* alto.

- `assertEquals(14, r.perimetro())`: Asegura que el método `perimetro()` de la instancia `r` devuelva el valor esperado, que en este caso es 14. La fórmula del perímetro para un rectángulo es 2 \* (ancho + alto).

Básicamente, este test verifica que la clase `rectangulo1` produce los resultados esperados para el área y el perímetro cuando se crea con un ancho de 3 y un alto de 4. Si ambos valores coinciden con las expectativas (12 para el área y 14 para el perímetro), la prueba se considerará exitosa. Si alguno de los valores no coincide, la prueba fallará y se informará cuál fue la discrepancia.

Ahora, realizaremos una modificación en el código de nuestro test para propósitos educativos, de manera que la prueba intencionalmente falle. Esto puede ser útil para observar cómo se manejan los errores en las pruebas.

Podemos hacer algo como lo siguiente:

```
```java
package ejemplo;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

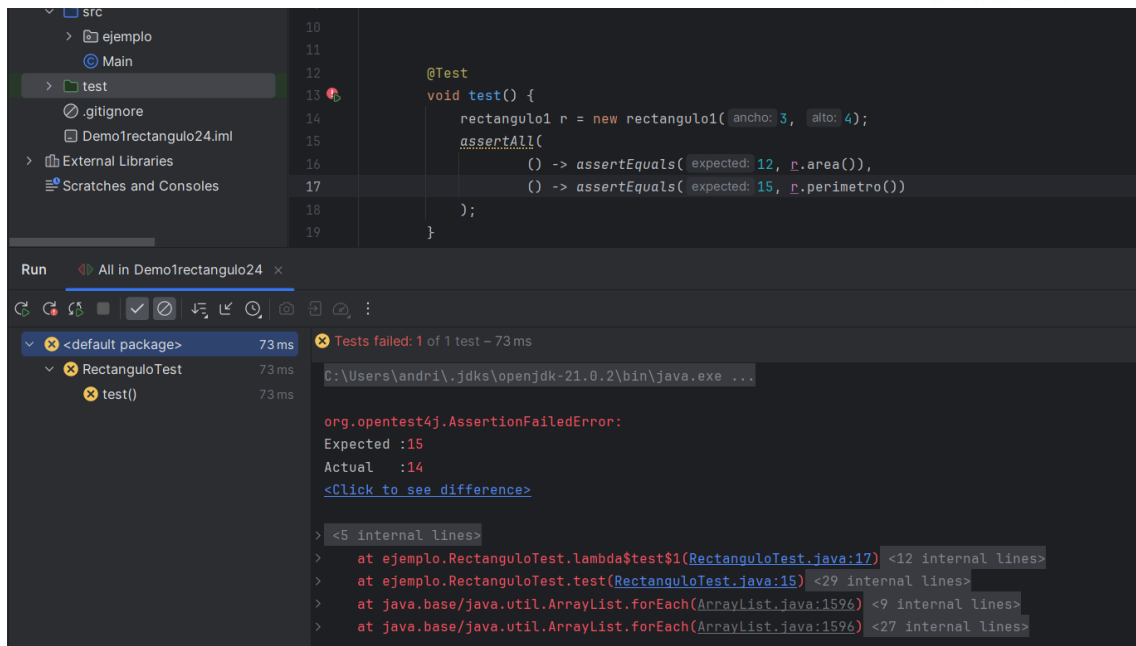
public class RectanguloTest {

    @Test
    void test() {
        rectangulo1 r = new rectangulo1(3, 4);

        // Modificamos el valor esperado para el área a propósito (debería ser 12)
        int valorEsperadoArea = 15;

        assertAll(
            () -> assertEquals(valorEsperadoArea, r.area()), // Fallback a 15 para que falle
            () -> assertEquals(14, r.perimetro())
        );
    }
}
```
```

Este es el resultado del test fallido:



```

10
11
12 @Test
13 void test() {
14     Rectangulo1 r = new Rectangulo1( ancho: 3, alto: 4);
15     assertEquals( expected: 12, r.area());
16     () -> assertEquals( expected: 15, r.perimetro());
17     () -> assertEquals( expected: 15, r.perimetro());
18 }
19
Run All in Demo1rectangulo24 x
Tests failed: 1 of 1 test - 73 ms
org.opentest4j.AssertionFailedError:
Expected :15
Actual :14
<Click to see difference>
> <5 internal lines>
> at ejemplo.RectanguloTest.lambda$test$1(RectanguloTest.java:17) <12 internal lines>
> at ejemplo.RectanguloTest.test(RectanguloTest.java:15) <29 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <9 internal lines>
> at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <27 internal lines>
  
```

En este caso, hemos cambiado el valor esperado para el área a 15 (en lugar de 12, que debería ser el valor correcto). Al hacer esto, estás introduciendo una discrepancia intencional entre el valor esperado y el valor real, y la prueba fallará. Este tipo de escenario puede ser útil para demostrar cómo las pruebas pueden detectar cambios no deseados en el código.

Recuerda revertir esta modificación después de que se haya entendido el concepto, para que la prueba vuelva a pasar.

**Test2:** Creamos otro test y le ponemos el nombre como indica a continuación, para probar otras anotaciones.

### RectangulodosTest.java

```
package ejemplo;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.TestInstance;

// import static org.junit.jupiter.api.Assertions.*; // Comentado
// temporalmente, puede ser útil más adelante

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
class RectangulodosTest {

    @BeforeAll
    static void setUpBeforeAll() {
        // Código de inicialización que se ejecutará una vez antes de
        // todas las pruebas
        System.out.println("Inicialización antes de todas las
pruebas");
    }

}
```

después de realizar el RUN del test tenemos este resultado

```
Project ▾
  DemoRectangulo24 C:\Users\lan...
    .idea
    .out
    src
      ejemplo
        Main
        test
      .gitignore
      DemoRectangulo24.iml
  External Libraries
  Scratches and Consoles

Main.java  Rectangulo.java  RectanguloTest.java  RectangulodosTest.java x
1 package ejemplo;
2
3 import org.junit.jupiter.api.BeforeAll;
4 import org.junit.jupiter.api.TestInstance;
5
6 // import static org.junit.jupiter.api.Assertions.*; // Comentado temporalmente, puede ser útil más adelante
7
8
9 @TestInstance(TestInstance.Lifecycle.PER_CLASS)
10 class RectangulodosTest {
11
12     @BeforeAll
13     static void setUpBeforeAll() {
14         // Código de inicialización que se ejecutará una vez antes de todas las pruebas
15         System.out.println("Inicialización antes de todas las pruebas");
16     }
17
18     // Resto de las pruebas y métodos aquí...
19
20 }
21

Run ▾ All in DemoRectangulo24 x
  <default package> 30 ms
  RectanguloTest 30 ms
  test() 30 ms
  Tests passed: 1 of 1 test - 30 ms
  C:\Users\landri\jdk\openjdk-21.0.2\bin\java.exe ...
  Process finished with exit code 0
```

Explicación del código de tu clase de prueba `RectangulodosTest`. Este archivo de prueba está escrito utilizando el framework de pruebas JUnit 5. A continuación, hay una explicación detallada del código:

package ejemplo;

```
import org.junit.jupiter.api.BeforeAll;  
import org.junit.jupiter.api.TestInstance;
```

```
// import static org.junit.jupiter.api.Assertions.*; // Comentado temporalmente, puede  
ser útil más adelante
```

```
@TestInstance(TestInstance.Lifecycle.PER_CLASS)  
class RectangulodosTest {
```

```
    @BeforeAll
```

```
    static void setUpBeforeAll() {  
        // Código de inicialización que se ejecutará una vez antes de todas las pruebas  
        System.out.println("Inicialización antes de todas las pruebas");  
    }
```

```
    // Resto de las pruebas y métodos aquí...
```

```
}
```

1. `@TestInstance(TestInstance.Lifecycle.PER_CLASS)`: Esta anotación configura el ciclo de vida de las instancias de prueba. Con `PER_CLASS`, se crea una instancia de la clase de prueba para todas las pruebas dentro de la clase. Esto significa que `@BeforeAll` y `@AfterAll` se ejecutarán en el mismo objeto de prueba.

2. `@BeforeAll`: Este método estático se ejecutará una vez antes de todas las pruebas en la clase. En este caso, se utiliza para imprimir un mensaje indicando que se está realizando una inicialización antes de todas las pruebas. Puedes agregar aquí cualquier código de inicialización que necesites para todas tus pruebas.

3. El comentario `// import static org.junit.jupiter.api.Assertions.*;`: Este comentario indica que la importación estática de `Assertions` está *comentada temporalmente* y no se está utilizando actualmente en el código.



### Test número 3.

Creamos un test le ponemos nombre **RectangulodostresTest.java**

```
package ejemplo;

import org.junit.jupiter.api.*;

import static org.junit.jupiter.api.Assertions.*;

class RectangulotresTest {

    @BeforeAll
    static void setUpBeforeAll() {
        // Código de inicialización que se ejecutará una vez antes de
        todas las pruebas
        System.out.println("Inicialización antes de todas las
pruebas");
    }

    @BeforeEach
    void setUpBeforeEach() {
        // Código de inicialización que se ejecutará antes de cada
prueba
        System.out.println("Inicialización antes de cada prueba");
    }

    @Test
    void testArea() {
        rectangulo1 r = new rectangulo1(3, 4);
        assertEquals(12, r.area());
    }

    @Test
    void testPerimetro() {
        rectangulo1 r = new rectangulo1(3, 4);
        assertEquals(14, r.perimetro());
    }

    @AfterEach
    void tearDownAfterEach() {
        // Código de limpieza que se ejecutará después de cada prueba
        System.out.println("Limpieza después de cada prueba");
    }

    @AfterAll
    static void tearDownAfterAll() {
        // Código de limpieza que se ejecutará una vez después de
todas las pruebas
        System.out.println("Limpieza después de todas las pruebas");
    }
}
```

A continuación, se expone una explicación de las anotaciones utilizadas en el test conjunto



Vamos a analizar el código del test línea por línea:

```
package ejemplo;

import org.junit.jupiter.api.*;

import static org.junit.jupiter.api.Assertions.*;

class RectangulotresTest {

    @BeforeAll
    static void setUpBeforeAll() {
        // Código de inicialización que se ejecutará una vez antes de todas las pruebas
        System.out.println("Inicialización antes de todas las pruebas");
    }
}
```

- **@BeforeAll**: Anotación que marca un método para ser ejecutado una vez antes de todas las pruebas en la clase de prueba. En este caso, el método `setUpBeforeAll` se ejecutará antes de cualquier prueba y se utiliza para la inicialización que es común a todas las pruebas.

```
@BeforeEach
void setUpBeforeEach() {
    // Código de inicialización que se ejecutará antes de cada prueba
    System.out.println("Inicialización antes de cada prueba");
}
```

- **@BeforeEach**: Anotación que marca un método para ser ejecutado antes de cada prueba en la clase de prueba. El método `setUpBeforeEach` se utiliza para la inicialización que debe realizarse antes de cada prueba individual.

```
@Test
void testArea() {
    rectangulo1 r = new rectangulo1(3, 4);
    assertEquals(12, r.area());
}
```

- **@Test**: Anotación que marca un método como un método de prueba. En este caso, el método `testArea` crea una instancia de la clase `rectangulo1` con dimensiones 3x4

y luego verifica si el resultado del método `area()` es igual a 12 utilizando `assertEquals`.

#### @Test

```
void testPerimetro() {  
    rectangulo1 r = new rectangulo1(3, 4);  
    assertEquals(14, r.perimetro());  
}
```

- Otro método de prueba similar al anterior, pero esta vez verifica el método `perimetro()` de la clase `rectangulo1` para asegurarse de que el resultado sea igual a 14.

#### @AfterEach

```
void tearDownAfterEach() {  
    // Código de limpieza que se ejecutará después de cada prueba  
    System.out.println("Limpieza después de cada prueba");  
}
```

- `@AfterEach`: Anotación que marca un método para ser ejecutado después de cada prueba en la clase de prueba. En este caso, el método `tearDownAfterEach` se utiliza para la limpieza que debe realizarse después de cada prueba individual.

#### @AfterAll

```
static void tearDownAfterAll() {  
    // Código de limpieza que se ejecutará una vez después de todas las pruebas  
    System.out.println("Limpieza después de todas las pruebas");  
}
```

- `@AfterAll`: Anotación que marca un método para ser ejecutado una vez después de todas las pruebas en la clase de prueba. El método `tearDownAfterAll` se utiliza para la limpieza que es común a todas las pruebas y se ejecutará al final de todas las pruebas.

Estas anotaciones son parte del framework de pruebas JUnit 5 y se utilizan para organizar y ejecutar las pruebas de manera estructurada. Los métodos marcados con estas anotaciones se ejecutarán en el orden indicado por sus nombres.

Vamos a repasar el orden que se debería seguir en las pruebas unitarias, a continuación, véase la siguiente tabla:

| Anotación                | Método                           | Descripción   |
|--------------------------|----------------------------------|---|
| <code>@BeforeAll</code>  | <code>setUpBeforeAll()</code>    | Inicialización antes de todas las pruebas                                       |
| <code>@BeforeEach</code> | <code>setUpBeforeEach()</code>   | Inicialización antes de cada prueba   |
| <code>@Test</code>       | <code>testArea()</code>          | Prueba del método <code>area()</code> de la clase <code>rectangulo1</code>      |
| <code>@AfterEach</code>  | <code>tearDownAfterEach()</code> | Limpieza después de cada prueba   |
| <code>@Test</code>       | <code>testPerimetro()</code>     | Prueba del método <code>perimetro()</code> de la clase <code>rectangulo1</code> |
| <code>@AfterEach</code>  | <code>tearDownAfterEach()</code> | Limpieza después de cada prueba   |
| <code>@AfterAll</code>   | <code>tearDownAfterAll()</code>  | Limpieza después de todas las pruebas   |

### ¡Recuerda que !

`@BeforeAll` y `@AfterAll` se ejecutan una vez para todas las pruebas,

mientras que

`@BeforeEach` y `@AfterEach` se ejecutan antes y después de cada prueba, respectivamente.

Los métodos de prueba marcados con `@Test` se ejecutan individualmente en el orden en que aparecen en la clase.