Unit test:

JUnit es un marco de pruebas unitarias para Java que proporciona diversas funciones para escribir y ejecutar pruebas. Aquí tienes un listado de las principales funciones y una explicación breve con un ejemplo simple para cada una:

---

### 1. \*\*@Test\*\*

Indica que un método es un caso de prueba.

Ejemplo:

```java

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

import org.junit.jupiter.api.Test;

public class MyTests {

@Test

void testAddition() {

assertEquals(2, 1 + 1);

}

}

```

---

### 2. \*\*@BeforeEach\*\*

Se ejecuta antes de cada método de prueba. Se usa para preparar el entorno.

Ejemplo:

```java

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

public class MyTests {

private int number;

@BeforeEach

void setup() {

number = 10;

}

@Test

void testNumber() {

assertEquals(10, number);

}

}

```

---

### 3. \*\*@AfterEach\*\*

Se ejecuta después de cada método de prueba. Se usa para limpiar recursos.

Ejemplo:

```java

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.Test;

public class MyTests {

@AfterEach

void tearDown() {

System.out.println("Test terminado");

}

@Test

void testExample() {

System.out.println("Ejecutando test");

}

}

```

---

### 4. \*\*@BeforeAll\*\*

Se ejecuta una vez antes de todos los métodos de prueba. Debe ser estático.

Ejemplo:

```java

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

public class MyTests {

@BeforeAll

static void setupAll() {

System.out.println("Preparando tests...");

}

@Test

void testExample() {

System.out.println("Ejecutando prueba");

}

}

```

---

### 5. \*\*@AfterAll\*\*

Se ejecuta una vez después de todos los métodos de prueba. También debe ser estático.

Ejemplo:

```java

import org.junit.jupiter.api.AfterAll;

import org.junit.jupiter.api.Test;

public class MyTests {

@AfterAll

static void tearDownAll() {

System.out.println("Todos los tests completados");

}

@Test

void testExample() {

System.out.println("Ejecutando prueba");

}

}

```

---

### 6. \*\*@Disabled\*\*

Desactiva temporalmente un método de prueba.

Ejemplo:

```java

import org.junit.jupiter.api.Disabled;

import org.junit.jupiter.api.Test;

public class MyTests {

@Test

void testEnabled() {

System.out.println("Esta prueba se ejecuta");

}

@Disabled("Esta prueba está desactivada")

@Test

void testDisabled() {

System.out.println("No se ejecutará");

}

}

```

---

### 7. \*\*Assertions (assertEquals, assertTrue, assertFalse, etc.)\*\*

Funciones estáticas usadas para verificar condiciones.

Ejemplo:

```java

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

import org.junit.jupiter.api.Test;

public class MyTests {

@Test

void testAssertions() {

assertEquals(5, 2 + 3); // Comprueba que son iguales

assertTrue(3 > 2); // Comprueba que la condición es verdadera

assertFalse(2 > 3); // Comprueba que la condición es falsa

}

}

```

---

### 8. \*\*@ParameterizedTest\*\*

Permite ejecutar pruebas con múltiples entradas.

Ejemplo:

```java

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

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

public class MyTests {

@ParameterizedTest

@ValueSource(ints = {1, 2, 3, 4, 5})

void testWithParameters(int number) {

assertTrue(number > 0);

}

}

```

---

### 9. \*\*@DisplayName\*\*

Permite dar un nombre legible a un caso de prueba.

Ejemplo:

```java

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

public class MyTests {

@Test

@DisplayName("Prueba de suma básica")

void testAddition() {

System.out.println("Ejecutando prueba de suma");

}

}

```

---

### 10. \*\*@Timeout\*\*

Especifica un tiempo máximo para que una prueba se ejecute.

Ejemplo:

```java

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.Timeout;

import java.util.concurrent.TimeUnit;

public class MyTests {

@Test

@Timeout(value = 1, unit = TimeUnit.SECONDS)

void testTimeout() throws InterruptedException {

Thread.sleep(500); // No excede el límite, prueba pasa

}

}

```

---

### 11. \*\*@Nested\*\*

Organiza pruebas relacionadas dentro de clases internas.

Ejemplo:

```java

import org.junit.jupiter.api.Nested;

import org.junit.jupiter.api.Test;

public class MyTests {

@Nested

class InnerTests {

@Test

void testInner() {

System.out.println("Prueba dentro de una clase anidada");

}

}

}

```

---

### 12. \*\*@RepeatedTest\*\*

Ejecuta un método de prueba varias veces.

Ejemplo:

```java

import org.junit.jupiter.api.RepeatedTest;

public class MyTests {

@RepeatedTest(3)

void testRepeated() {

System.out.println("Ejecutando prueba repetida");

}

}

```

---

Con estas funciones, JUnit te permite escribir pruebas flexibles y bien estructuradas para tus aplicaciones Java.

JUnit proporciona varias \*\*assertions\*\* (afirmaciones) que permiten verificar el comportamiento esperado del código en las pruebas unitarias. Aquí tienes una lista de los principales tipos de assertions en \*\*JUnit 5\*\*, con una breve descripción y ejemplos:

---

### 1. \*\*assertEquals\*\*

Verifica que dos valores sean iguales.

- \*\*Uso:\*\* `assertEquals(expected, actual)`

- También puedes especificar un mensaje opcional.

Ejemplo:

```java

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

assertEquals(5, 2 + 3); // Pasa porque 2 + 3 es 5

assertEquals("Hola", "Hola", "Los textos deben ser iguales");

```

---

### 2. \*\*assertNotEquals\*\*

Verifica que dos valores \*\*no\*\* sean iguales.

Ejemplo:

```java

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

assertNotEquals(5, 2 + 2); // Pasa porque 2 + 2 no es igual a 5

```

---

### 3. \*\*assertTrue\*\*

Verifica que una condición sea \*\*verdadera\*\*.

Ejemplo:

```java

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

assertTrue(3 > 2); // Pasa porque 3 es mayor que 2

```

---

### 4. \*\*assertFalse\*\*

Verifica que una condición sea \*\*falsa\*\*.

Ejemplo:

```java

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

assertFalse(2 > 3); // Pasa porque 2 no es mayor que 3

```

---

### 5. \*\*assertNull\*\*

Verifica que un objeto sea \*\*nulo\*\*.

Ejemplo:

```java

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

String text = null;

assertNull(text); // Pasa porque el texto es nulo

```

---

### 6. \*\*assertNotNull\*\*

Verifica que un objeto \*\*no\*\* sea nulo.

Ejemplo:

```java

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

String text = "JUnit";

assertNotNull(text); // Pasa porque el texto no es nulo

```

---

### 7. \*\*assertSame\*\*

Verifica que dos referencias apunten al \*\*mismo objeto\*\* (identidad de objeto).

Ejemplo:

```java

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

String text = "JUnit";

assertSame(text, text); // Pasa porque ambas referencias son iguales

```

---

### 8. \*\*assertNotSame\*\*

Verifica que dos referencias no apunten al mismo objeto.

Ejemplo:

```java

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

String text1 = "Hola";

String text2 = new String("Hola");

assertNotSame(text1, text2); // Pasa porque son diferentes instancias

```

---

### 9. \*\*assertArrayEquals\*\*

Verifica que dos arrays sean iguales.

Ejemplo:

```java

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

int[] expected = {1, 2, 3};

int[] actual = {1, 2, 3};

assertArrayEquals(expected, actual); // Pasa porque ambos arrays son iguales

```

---

### 10. \*\*assertThrows\*\*

Verifica que se lance una excepción específica durante la ejecución de un bloque de código.

Ejemplo:

```java

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

assertThrows(ArithmeticException.class, () -> {

int result = 10 / 0; // Genera una excepción

});

```

---

### 11. \*\*assertDoesNotThrow\*\*

Verifica que un bloque de código \*\*no lance ninguna excepción\*\*.

Ejemplo:

```java

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

assertDoesNotThrow(() -> {

int result = 10 / 2; // No genera excepción

});

```

---

### 12. \*\*assertTimeout\*\*

Verifica que un bloque de código se ejecute dentro de un límite de tiempo.

Ejemplo:

```java

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

import java.time.Duration;

assertTimeout(Duration.ofSeconds(1), () -> {

Thread.sleep(500); // Pasa porque toma menos de 1 segundo

});

```

---

### 13. \*\*assertTimeoutPreemptively\*\*

Verifica que un bloque de código se ejecute dentro de un límite de tiempo y aborta si no lo cumple (interrumpiendo el hilo).

Ejemplo:

```java

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

import java.time.Duration;

assertTimeoutPreemptively(Duration.ofMillis(500), () -> {

Thread.sleep(300); // Pasa porque toma menos de 500 ms

});

```

---

### 14. \*\*fail\*\*

Fuerza el fallo de una prueba si se ejecuta. Útil para escenarios que no deberían alcanzarse.

Ejemplo:

```java

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

void someMethod() {

fail("Esto no debería ejecutarse");

}

```

---

Estas \*\*assertions\*\* te permiten validar múltiples aspectos de tu código, asegurándote de que se comporta como se espera en diferentes escenarios.

El uso adecuado de los métodos de JUnit depende del contexto de tu prueba y del objetivo que tengas. A continuación, te detallo \*\*cuándo deberías usar o evitar\*\* los principales métodos de JUnit:

---

### \*\*1. `assertEquals` y `assertNotEquals`\*\*

- \*\*Cuándo usarlo:\*\*

- Para comparar valores simples o resultados calculados.

- Ejemplo: Comparar resultados de métodos (`getName()`, operaciones matemáticas, etc.).

- \*\*Cuándo evitarlo:\*\*

- Si estás comparando objetos complejos que no implementan adecuadamente `equals()`. En ese caso, verifica los atributos manualmente.

---

### \*\*2. `assertTrue` y `assertFalse`\*\*

- \*\*Cuándo usarlo:\*\*

- Para validar condiciones booleanas específicas, como el resultado de un método o una comparación lógica.

- Ejemplo: `assertTrue(user.isActive());`

- \*\*Cuándo evitarlo:\*\*

- Si tienes un valor esperado específico que puedes verificar con `assertEquals`. Por ejemplo, en lugar de:

```java

assertTrue(result == 5);

```

Usa:

```java

assertEquals(5, result);

```

Esto hace que la prueba sea más descriptiva y fácil de entender.

---

### \*\*3. `assertNull` y `assertNotNull`\*\*

- \*\*Cuándo usarlo:\*\*

- Cuando esperas que un objeto esté inicializado (`assertNotNull`) o que sea nulo (`assertNull`).

- Ejemplo:

- Validar que un objeto recién creado no sea nulo.

- Comprobar que un valor no se ha inicializado aún.

- \*\*Cuándo evitarlo:\*\*

- Si necesitas verificar más detalles del objeto (como sus atributos). En ese caso, combina con otros asserts.

---

### \*\*4. `assertSame` y `assertNotSame`\*\*

- \*\*Cuándo usarlo:\*\*

- Para verificar que dos referencias apuntan al \*\*mismo objeto\*\*.

- Ejemplo: Comparar objetos con un patrón Singleton o referencias explícitas.

- \*\*Cuándo evitarlo:\*\*

- Si solo necesitas verificar que dos objetos son equivalentes. Usa `assertEquals` en su lugar.

---

### \*\*5. `assertArrayEquals`\*\*

- \*\*Cuándo usarlo:\*\*

- Para comparar arrays completos, asegurándote de que tienen los mismos elementos en el mismo orden.

- Ejemplo: Comparar resultados de métodos que devuelven arrays.

- \*\*Cuándo evitarlo:\*\*

- Si necesitas comparar estructuras más complejas (por ejemplo, listas o mapas). En esos casos, usa comparaciones manuales o métodos específicos.

---

### \*\*6. `assertThrows`\*\*

- \*\*Cuándo usarlo:\*\*

- Para verificar que un método lanza una excepción esperada.

- Ejemplo:

- Validar que se lanza una excepción al dividir por cero.

- Probar que un método falla con entradas inválidas.

- \*\*Cuándo evitarlo:\*\*

- Si no esperas excepciones en el flujo normal del código. En ese caso, usa `assertDoesNotThrow`.

---

### \*\*7. `assertDoesNotThrow`\*\*

- \*\*Cuándo usarlo:\*\*

- Para validar que un método no lanza ninguna excepción inesperada.

- Ejemplo: Métodos que no deberían fallar bajo circunstancias normales.

- \*\*Cuándo evitarlo:\*\*

- Si no esperas excepciones y la validación de la salida del método es suficiente.

---

### \*\*8. `assertTimeout` y `assertTimeoutPreemptively`\*\*

- \*\*Cuándo usarlo:\*\*

- Para validar que un método cumple con restricciones de tiempo.

- Ejemplo: Métodos que procesan datos grandes o esperan respuestas externas.

- \*\*Cuándo evitarlo:\*\*

- Para métodos con tiempos de ejecución variables o dependientes del hardware, ya que podrían dar falsos negativos.

---

### \*\*9. `@Test`, `@BeforeEach`, `@AfterEach`, `@BeforeAll`, `@AfterAll`\*\*

- \*\*Cuándo usarlo:\*\*

- \*\*@Test:\*\* Siempre, para marcar métodos de prueba.

- \*\*@BeforeEach:\*\* Cuando necesitas inicializar datos antes de cada prueba.

- \*\*@AfterEach:\*\* Para liberar recursos después de cada prueba.

- \*\*@BeforeAll:\*\* Cuando necesitas configurar algo \*\*una vez\*\* antes de todas las pruebas (como una base de datos o conexión).

- \*\*@AfterAll:\*\* Para liberar recursos compartidos después de todas las pruebas (como cerrar una base de datos).

- \*\*Cuándo evitarlo:\*\*

- Si la configuración es demasiado específica para una prueba. En ese caso, inicializa directamente dentro del método de prueba.

---

### \*\*10. `fail`\*\*

- \*\*Cuándo usarlo:\*\*

- Para forzar el fallo de una prueba cuando no se debería alcanzar un bloque de código.

- Ejemplo:

```java

try {

someMethod();

fail("Debería haber lanzado una excepción");

} catch (Exception e) {

// Validar la excepción

}

```

- \*\*Cuándo evitarlo:\*\*

- Si no tienes escenarios donde necesites validar que un bloque no debería ejecutarse.

---

### \*\*11. `@Disabled`\*\*

- \*\*Cuándo usarlo:\*\*

- Cuando deseas deshabilitar una prueba temporalmente (por ejemplo, en desarrollo o debugging).

- \*\*Cuándo evitarlo:\*\*

- Si simplemente necesitas eliminar o comentar una prueba innecesaria.

---

### \*\*12. `@ParameterizedTest`\*\*

- \*\*Cuándo usarlo:\*\*

- Para probar un método con múltiples valores o combinaciones de entrada.

- Ejemplo: Probar un método con diferentes números o cadenas.

- \*\*Cuándo evitarlo:\*\*

- Si solo tienes un valor de entrada o el número de combinaciones es pequeño.

---

En resumen, elige el método adecuado según lo que deseas verificar (valores, excepciones, condiciones o tiempos) y prioriza la claridad de las pruebas para que sean fáciles de entender y mantener.