

Introducción a las pruebas automáticas unitarias con JUnit

- -
-
- -
 - -
 -
 -
 -
 - -
 -
- -
 -
 -
 -
 -
 -
- -
 -
- - -
 - -
 -
 - -
 - -
 - -
 - -
 - -

Assertions

ValidadorContrasena

GestorInventario

@Nested

@Nested

@Disabled

@Suite

- o
- o
-
- o
- o
- o
-
-
- o
-
- o
- o
- o
-
- o
- o
- o

esPalindromo

calcularFactorial



1. Introducción

1.1. Historia y versiones de JUnit

- `TestCase`
- `@Test` `@Before` `@After`
-

2. Instalación de JUnit

1.

2.

`pom.xml`

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.10.0</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.10.0</version>
  <scope>test</scope>
</dependency>
```

3.

`build.gradle`

```
testImplementation 'org.junit.jupiter:junit-jupiter-api:5.10.0'
testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.10.0'
```

3. Ejemplo de pruebas unitarias con JUnit

Calculadora

Calculadora

suma resta

```
//Clase Calculadora
public class Calculadora {

    public int suma(int a, int b) {
        return a + b;
    }

    public int resta(int a, int b) {
        return a - b;
    }
}
```

CalculadoraTest

testSuma testResta
Calculadora

suma

resta

CalculadoraTest

```
//Ejemplo de uso de JUnit
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

public class CalculadoraTest {

    @Test
    public void testSuma() {
        Calculadora calc = new Calculadora();
        assertEquals(5, calc.suma(2, 3));
    }

    @Test
    public void testResta() {
        Calculadora calc = new Calculadora();
        assertEquals(1, calc.resta(3, 2));
    }
}
```

assertEquals

3.1. Estructura general del proyecto

src/main/java

src/test/java

```
mi-proyecto/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   ├── com/
│   │   │   │   ├── ejemplo/
│   │   │   │   │   └── Calculadora.java
│   │   └── test/
│   │       ├── java/
│   │       │   ├── com/
│   │       │   │   ├── ejemplo/
│   │       │   │   │   └── CalculadoraTest.java
│   └── pom.xml
```

3.2. Estructura de una Prueba en JUnit

3.2.1. Anotaciones Básicas

1. @Test

-

-

@Test

-

```
@Test
public void testSuma() {
    Calculadora calculadora = new Calculadora();
    assertEquals(5, calculadora.suma(2, 3));
}
```

2. @BeforeEach @AfterEach

- @BeforeEach

@Test

- @AfterEach

-

```

private Calculadora calculadora;

@BeforeEach
public void setUp() {
    calculadora = new Calculadora();
}

@AfterEach
public void tearDown() {
    calculadora = null;
}

```

3. @BeforeAll @AfterAll

- @BeforeAll

- @AfterAll

- static

-

```

@BeforeAll
public static void init() {
    System.out.println("Iniciando recursos para
todas las pruebas");
}

@AfterAll
public static void cleanup() {
    System.out.println("Liberando recursos después de
todas las pruebas");
}

```

3.2.2. Métodos de Prueba

@Test

-

-

-

- testSuma

- testDivisionPorCero

-

- camelCase

- test_suma

-

-

-

-


```

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

public class CalculadoraTest {

    private Calculadora calculadora;

    @BeforeAll
    public static void init() {
        System.out.println("Inicializando recursos para todas
las pruebas");
    }

    @AfterAll
    public static void cleanup() {
        System.out.println("Liberando recursos después de
todas las pruebas");
    }

    @BeforeEach
    public void setUp() {
        calculadora = new Calculadora();
    }

    @AfterEach
    public void tearDown() {
        calculadora = null;
    }

    @Test
    public void testSuma() {
        assertEquals(5, calculadora.suma(2, 3), "2 + 3 debería
ser 5");
    }

    @Test
    public void testResta() {
        assertEquals(1, calculadora.resta(3, 2), "3 - 2
debería ser 1");
    }
}

```

3.2.3. Aserciones (Assertions)

Assertions

1. `assertEquals(expected, actual)`

`expected`

`actual`

```
@Test
public void testSuma() {
    assertEquals(5, calculadora.suma(2, 3));
}
```

2. `assertTrue(condition)`

`assertFalse(condition)`

```
@Test
public void testEsPositivo() {
    assertTrue(calculadora.esPositivo(10));
    assertFalse(calculadora.esPositivo(-5));
}
```

3. `assertNull(object)`

`assertNotNull(object)`

`null`

```
@Test
public void testObjetoNulo() {
    assertNull(calculadora.obtenerResultado());
    assertNotNull(calculadora);
}
```

4. `assertThrows(Exception.class, executable)`

```
@Test
public void testDivisionPorCero() {
    assertThrows(ArithmeticException.class, () ->
        calculadora.dividir(10, 0));
}
```

5. `assertAll`

-

```
@Test
public void testVariasOperaciones() {
    assertAll(
        () -> assertEquals(5, calculadora.suma(2, 3)),
        () -> assertEquals(1, calculadora.resta(3, 2)),
        () -> assertTrue(calculadora.esPositivo(10))
    );
}
```

Assertions

3.2.4. Buenas Prácticas

1.

-

2.

-

```
assertEquals(5, calculadora.suma(2, 3), "La suma de 2 y  
3 debería ser 5");
```

3.

4.

3.3. Aplicación de casos de prueba con JUnit

3.3.1. Pruebas de Caja Blanca con JUnit

1.

-

if-else switch

-

```
public class Calculadora {
    public int dividir(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("División por
cero");
        }
        return a / b;
    }
}

@Test
public void testDividir_CasoNormal() {
    Calculadora calculadora = new Calculadora();
    assertEquals(2, calculadora.dividir(10, 5));
}

@Test
public void testDividir_DivisionPorCero() {
    Calculadora calculadora = new Calculadora();
    assertThrows(ArithmeticException.class, () ->
calculadora.dividir(10, 0));
}
```

2.

-

-

```

public class Verificador {
    public String verificarEdad(int edad) {
        if (edad < 0) {
            return "Edad inválida";
        } else if (edad < 18) {
            return "Menor de edad";
        } else {
            return "Mayor de edad";
        }
    }
}

@Test
public void testVerificarEdad_MenorDeEdad() {
    Verificador verificador = new Verificador();
    assertEquals("Menor de edad",
verificador.verificarEdad(15));
}

@Test
public void testVerificarEdad_MayorDeEdad() {
    Verificador verificador = new Verificador();
    assertEquals("Mayor de edad",
verificador.verificarEdad(20));
}

@Test
public void testVerificarEdad_EdadInvalida() {
    Verificador verificador = new Verificador();
    assertEquals("Edad inválida",
verificador.verificarEdad(-5));
}

```

3.

-

-

```

public class Contador {
    public int contarLetrasA(String texto) {
        int count = 0;
        for (char c : texto.toCharArray()) {
            if (c == 'a' || c == 'A') {
                count++;
            }
        }
        return count;
    }
}

@Test
public void testContarLetrasA_CasoVacio() {
    Contador contador = new Contador();
    assertEquals(0, contador.contarLetrasA(""));
}

@Test
public void testContarLetrasA_UnaLetraA() {
    Contador contador = new Contador();
    assertEquals(1, contador.contarLetrasA("Hola"));
}

@Test
public void testContarLetrasA_VariasLetrasA() {
    Contador contador = new Contador();
    assertEquals(3,
contador.contarLetrasA("Anaranjado"));
}

```

3.3.2. Pruebas de Caja Negra con JUnit

1.

◦

◦

```

public class Validador {
    public boolean esNumeroValido(int numero) {
        return numero >= 1 && numero <= 100;
    }
}

@Test
public void testEsNumeroValido_CasoValido() {
    Validador validador = new Validador();
    assertTrue(validador.esNumeroValido(50));
}

@Test
public void testEsNumeroValido_CasoInvalido() {
    Validador validador = new Validador();
    assertFalse(validador.esNumeroValido(0));
    assertFalse(validador.esNumeroValido(101));
}

```

2.

-
-

```

@Test
public void testEsNumeroValido_ValoresLimite() {
    Validador validador = new Validador();
    assertTrue(validador.esNumeroValido(1)); // Límite inferior
    assertTrue(validador.esNumeroValido(100)); // Límite superior
    assertFalse(validador.esNumeroValido(0)); // Fuera del límite inferior
    assertFalse(validador.esNumeroValido(101)); // Fuera del límite superior
}

```

3.

-
-

```

@Test
public void testCombinaciones() {
    Validador validador = new Validador();
    assertAll(
        () -> assertTrue(validador.esNumeroValido(50)),
        () -> assertFalse(validador.esNumeroValido(-1)),
        () -> assertFalse(validador.esNumeroValido(101))
    );
}

```

4. Como usar JUnit en IntelliJ IDEA

4.1. Configurar JUnit en el Proyecto

1.

o

a.

pom.xml

b.

```

<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.10.0</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-
engine</artifactId>
    <version>5.10.0</version>
    <scope>test</scope>
  </dependency>
</dependencies>

```

c.

-

- a. `build.gradle`

- b.

```
dependencies {  
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.10.0'  
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.10.0'  
}
```

- c.

- 2.

-

```
src/  
├─ main/  
│   └─ java/           # Código fuente principal  
└─ test/  
    └─ java/           # Código de pruebas
```

`src/test/java`

4.2. Crear una Clase de Prueba

- 1. `src/test/java`

-

-

-

-

2.

Calculadora

suma

resta

src/test/java

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

public class CalculadoraTest {

    @Test
    public void testSuma() {
        Calculadora calculadora = new Calculadora();
        assertEquals(5, calculadora.suma(2, 3));
    }

    @Test
    public void testResta() {
        Calculadora calculadora = new Calculadora();
        assertEquals(1, calculadora.resta(3, 2));
    }
}
```

4.3. Ejecutar las Pruebas

1.

◦

testResta

testSuma

◦

2.

◦

CalculadoraTest

◦

3.

◦

◦

4.4. Depurar Pruebas

1.

-

2.

-
-

3.

-

4.5. Ver la Cobertura de Código

1.

-
-

2.

-
-

5. Ejemplos de aplicación de JUnit

5.1. Ejemplo 1: Clase `ValidadorContrasena`

- 1.
- 2.
- 3.
- 4.
- 5.

! @ #


```

public class ValidadorContrasena {

    public boolean validar(String contrasena) {
        // Verificar longitud mínima
        if (contrasena.length() < 8) {
            return false;
        }

        // Verificar al menos una letra mayúscula
        boolean tieneMayuscula = false;
        // Verificar al menos una letra minúscula
        boolean tieneMinuscula = false;
        // Verificar al menos un dígito
        boolean tieneDigito = false;
        // Verificar al menos un carácter especial
        boolean tieneEspecial = false;

        for (char c : contrasena.toCharArray()) {
            if (Character.isUpperCase(c)) {
                tieneMayuscula = true;
            } else if (Character.isLowerCase(c)) {
                tieneMinuscula = true;
            } else if (Character.isDigit(c)) {
                tieneDigito = true;
            } else if (esCaracterEspecial(c)) {
                tieneEspecial = true;
            }
        }

        // La contraseña es válida si cumple todos los requisitos
        return tieneMayuscula && tieneMinuscula && tieneDigito &&
tieneEspecial;
    }

    private boolean esCaracterEspecial(char c) {
        // Definir caracteres especiales permitidos
        String caracteresEspeciales = "!@#$%^&*()-_+=[]{};:'\".,.<>/?
`~";
        return caracteresEspeciales.contains(String.valueOf(c));
    }
}

```

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

public class ValidadorContrasenaTest {

    private ValidadorContrasena validador = new
ValidadorContrasena();

    // Pruebas de Caja Negra
    @Test
    public void testcontrasenaValida() {
        assertTrue(validador.validar("Passw0rd!"), "La contrasena
debería ser válida");
    }

    @Test
    public void testcontrasenaCorta() {
        assertFalse(validador.validar("Pwd1!"), "La contrasena es
demasiado corta");
    }

    @Test
    public void testcontrasenaSinMayuscula() {
        assertFalse(validador.validar("password1!"), "La contrasena
no tiene mayúscula");
    }

    @Test
    public void testcontrasenaSinMinuscula() {
        assertFalse(validador.validar("PASSWORD1!"), "La contrasena
no tiene minúscula");
    }

    @Test
    public void testcontrasenaSinDigito() {
        assertFalse(validador.validar("Password!"), "La contrasena no
tiene dígito");
    }

    @Test
    public void testcontrasenaSinCaracterEspecial() {
        assertFalse(validador.validar("Password1"), "La contrasena no
tiene carácter especial");
    }

    // Pruebas de Caja Blanca
    @Test
```

```
public void testCoberturaDeCaminos() {
    // Prueba para cubrir el caso donde falta una mayúscula
    assertFalse(validador.validar("password1!"), "Falta
mayúscula");

    // Prueba para cubrir el caso donde falta una minúscula
    assertFalse(validador.validar("PASSWORD1!"), "Falta
minúscula");

    // Prueba para cubrir el caso donde falta un dígito
    assertFalse(validador.validar("Password!"), "Falta dígito");

    // Prueba para cubrir el caso donde falta un carácter
especial
    assertFalse(validador.validar("Password1"), "Falta carácter
especial");

    // Prueba para cubrir el caso donde todo es correcto
    assertTrue(validador.validar("Passw0rd!"), "Todo es
correcto");
}

@Test
public void testCaracterEspecialInvalido() {
    // Prueba para cubrir el caso donde se usa un carácter no
permitido
    assertFalse(validador.validar("Password1|"), "Carácter
especial no permitido");
}
}
```

5.2. Ejemplo 2: Clase `GestorInventario`

```
import java.util.HashMap;
import java.util.Map;

public class GestorInventario {

    private Map<String, Integer> inventario;

    public GestorInventario() {
        inventario = new HashMap<>();
    }

    /**
     * Añade un producto al inventario.
     *
     * @param nombre El nombre del producto.
     * @param cantidad La cantidad a añadir.
     */
    public void anadirProducto(String nombre, int cantidad) {
        if (cantidad <= 0) {
            throw new IllegalArgumentException("La cantidad debe ser mayor que cero.");
        }
        inventario.put(nombre, inventario.getDefault(nombre, 0) + cantidad);
    }

    /**
     * Elimina un producto del inventario.
     *
     * @param nombre El nombre del producto.
     * @param cantidad La cantidad a eliminar.
     */
    public void eliminarProducto(String nombre, int cantidad) {
        if (!inventario.containsKey(nombre)) {
            throw new IllegalArgumentException("El producto no existe en el inventario.");
        }
        if (cantidad <= 0) {
            throw new IllegalArgumentException("La cantidad debe ser mayor que cero.");
        }
        int stockActual = inventario.get(nombre);
        if (cantidad > stockActual) {
            throw new IllegalArgumentException("No hay suficiente stock para eliminar.");
        }
        inventario.put(nombre, stockActual - cantidad);
    }
}
```

```
}

/**
 * Busca un producto en el inventario.
 *
 * @param nombre El nombre del producto.
 * @return La cantidad en stock, o null si el producto no existe.
 */
public Integer buscarProducto(String nombre) {
    return inventario.get(nombre);
}

/**
 * Verifica el stock disponible de un producto.
 *
 * @param nombre El nombre del producto.
 * @return La cantidad en stock.
 * @throws IllegalArgumentException Si el producto no existe.
 */
public int verificarStock(String nombre) {
    if (!inventario.containsKey(nombre)) {
        throw new IllegalArgumentException("El producto no existe en el inventario.");
    }
    return inventario.get(nombre);
}
}
```

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class GestorInventarioTest {

    private GestorInventario gestor;

    @BeforeEach
    public void setUp() {
        gestor = new GestorInventario();
    }

    // Pruebas de Caja Negra
    @Test
    public void testanadirProducto() {
        gestor.anadirProducto("Manzana", 10);
        assertEquals(10, gestor.verificarStock("Manzana"), "El stock de Manzana debería ser 10");
    }

    @Test
    public void testanadirProductoExistente() {
        gestor.anadirProducto("Manzana", 10);
        gestor.anadirProducto("Manzana", 5);
        assertEquals(15, gestor.verificarStock("Manzana"), "El stock de Manzana debería ser 15");
    }

    @Test
    public void testEliminarProducto() {
        gestor.anadirProducto("Manzana", 10);
        gestor.eliminarProducto("Manzana", 4);
        assertEquals(6, gestor.verificarStock("Manzana"), "El stock de Manzana debería ser 6");
    }

    @Test
    public void testBuscarProductoExistente() {
        gestor.anadirProducto("Manzana", 10);
        assertEquals(10, gestor.buscarProducto("Manzana"), "El stock de Manzana debería ser 10");
    }

    @Test
    public void testBuscarProductoInexistente() {
        assertNull(gestor.buscarProducto("Pera"), "El producto no
```

```

    debería existir en el inventario");
    }

    @Test
    public void testVerificarStockProductoInexistente() {
        assertThrows(IllegalArgumentException.class, () ->
gestor.verificarStock("Pera"), "Debería lanzar una excepción si el
producto no existe");
    }

    // Pruebas de Caja Blanca
    @Test
    public void testanadirProductoCantidadInvalida() {
        assertThrows(IllegalArgumentException.class, () ->
gestor.anadirProducto("Manzana", 0), "Debería lanzar una excepción si
la cantidad es menor o igual a cero");
    }

    @Test
    public void testEliminarProductoCantidadInvalida() {
        gestor.anadirProducto("Manzana", 10);
        assertThrows(IllegalArgumentException.class, () ->
gestor.eliminarProducto("Manzana", 0), "Debería lanzar una excepción
si la cantidad es menor o igual a cero");
    }

    @Test
    public void testEliminarProductoStockInsuficiente() {
        gestor.anadirProducto("Manzana", 10);
        assertThrows(IllegalArgumentException.class, () ->
gestor.eliminarProducto("Manzana", 15), "Debería lanzar una excepción
si no hay suficiente stock");
    }

    @Test
    public void testEliminarProductoInexistente() {
        assertThrows(IllegalArgumentException.class, () ->
gestor.eliminarProducto("Pera", 5), "Debería lanzar una excepción si
el producto no existe");
    }
}

```


- -
- 2.

-

eliminarProducto

anadirProducto

-

6. Pruebas Avanzadas

6.1. Pruebas Parametrizadas

6.1.1. Cómo usar pruebas parametrizadas en JUnit 5

1. @ParameterizedTest

-

-

@Test

- 2.

-

- @ValueSource

- @CsvSource

- @MethodSource

- @EnumSource

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-params</artifactId>
  <version>5.11.3</version>
  <scope>test</scope>
</dependency>
```

```
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;
import static org.junit.jupiter.api.Assertions.assertTrue;

public class PruebasParametrizadasTest {

    @ParameterizedTest
    @ValueSource(ints = {2, 4, 6, 8, 10})
    public void testEsPar(int numero) {
        assertTrue(numero % 2 == 0, "El número " + numero + " debería ser par");
    }
}
```

@ValueSource

@CsvSource

```
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class SumaParametrizadaTest {

    @ParameterizedTest
    @CsvSource({
        "1, 2, 3",
        "5, 5, 10",
        "0, 0, 0",
        "-1, 1, 0"
    })
    public void testSuma(int a, int b, int resultadoEsperado) {
        Calculadora calculadora = new Calculadora();
        assertEquals(resultadoEsperado, calculadora.suma(a, b), "La
suma de " + a + " y " + b + " debería ser " + resultadoEsperado);
    }
}
```

@CsvSource

@CsvSource

@CsvSource

6.2. Pruebas de Excepciones

6.2.1. Cómo probar excepciones en JUnit 5

1. **assertThrows**
 -
 - `assertThrows(ExcepcionEsperada.class, () -> { ... })`

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

public class PruebasExcepcionesTest {

    @Test
    public void testDivisionPorCero() {
        Calculadora calculadora = new Calculadora();
        assertThrows(ArithmeticException.class, () ->
calculadora.dividir(10, 0), "Debería lanzar ArithmeticException al
dividir por cero");
    }
}
```

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

public class PruebasExcepcionesConMensajeTest {

    @Test
    public void testMensajeExcepcion() {
        Calculadora calculadora = new Calculadora();
        Exception excepcion = assertThrows(ArithmeticException.class,
() -> calculadora.dividir(10, 0));
        assertEquals("División por cero no permitida",
excepcion.getMessage(), "El mensaje de la excepción no coincide");
    }
}
```

6.3. Combinación de Pruebas Parametrizadas y de Excepciones

validar

```

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;
import static org.junit.jupiter.api.Assertions.assertThrows;

public class ValidadorPositivoTest {

    private ValidadorPositivo validador = new ValidadorPositivo();

    @ParameterizedTest
    @ValueSource(ints = {0, -1, -100}) // Números no positivos
    public void testValidarNumerosNoPositivos(int numero) {
        // Verificamos que se lanza una excepción para números no
        positivos
        assertThrows(IllegalArgumentException.class, () ->
validador.validar(numero), "Debería lanzar IllegalArgumentException
para " + numero);
    }

    @ParameterizedTest
    @ValueSource(ints = {1, 10, 100}) // Números positivos
    public void testValidarNumerosPositivos(int numero) {
        // Verificamos que no se lanza una excepción para números
        positivos
        assertDoesNotThrow(() -> validador.validar(numero), "No
debería lanzar excepción para " + numero);
    }
}

```

1.

- @ParameterizedTest @ValueSource
- testValidarNumerosNoPositivos
- testValidarNumerosPositivos

2.

- assertThrows
- assertDoesNotThrow

6.4. Pruebas Anidadas (@Nested)

6.4.1. Cómo usar `@Nested` en JUnit 5

1. `@Nested`

-
-

2.

-
-

`@AfterEach`

`@BeforeEach`

```

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

public class CalculadoraTest {

    private Calculadora calculadora;

    @BeforeEach
    public void setUp() {
        calculadora = new Calculadora();
    }

    @Nested
    class PruebasDeSuma {

        @Test
        public void testSumaPositivos() {
            assertEquals(5, calculadora.suma(2, 3), "2 + 3 debería
ser 5");
        }

        @Test
        public void testSumaNegativos() {
            assertEquals(-5, calculadora.suma(-2, -3), "-2 + (-3)
debería ser -5");
        }
    }

    @Nested
    class PruebasDeResta {

        @Test
        public void testRestaPositivos() {
            assertEquals(1, calculadora.resta(3, 2), "3 - 2 debería
ser 1");
        }

        @Test
        public void testRestaNegativos() {
            assertEquals(-1, calculadora.resta(-2, -1), "-2 - (-1)
debería ser -1");
        }
    }
}

```

-
-

PruebasDeSuma

PruebasDeResta

6.5. Deshabilitar Pruebas (@Disabled)

@Disabled

1. @Disabled
 -
 -
2.
 -
 -
 -


```

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

public class CalculadoraTest {

    private Calculadora calculadora;

    @BeforeEach
    public void setUp() {
        calculadora = new Calculadora();
    }

    @Test
    public void testSuma() {
        assertEquals(5, calculadora.suma(2, 3), "2 + 3 debería ser 5");
    }

    @Test
    @Disabled("Esta prueba está en desarrollo")
    public void testResta() {
        assertEquals(1, calculadora.resta(3, 2), "3 - 2 debería ser 1");
    }

    @Test
    @Disabled("Esta prueba falla y necesita ser revisada")
    public void testMultiplicacion() {
        assertEquals(6, calculadora.multiplicar(2, 3), "2 * 3 debería ser 6");
    }
}

```

- `testResta`
- `testMultiplicacion`
-

6.6. Suite de pruebas (@Suite)

@Suite

@Suite

6.6.1. Pasos para crear un suite de tests en JUnit 5

1.

pom.xml

```
<dependency>
  <groupId>org.junit.platform</groupId>
  <artifactId>junit-platform-suite</artifactId>
  <version>1.9.1</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.9.1</version>
  <scope>test</scope>
</dependency>
```

```
testImplementation      'org.junit.platform:junit-platform-
suite:1.9.1'
testImplementation      'org.junit.jupiter:junit-jupiter-
engine:5.9.1'
```

2.

```
package com.bibliotecas.app;

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

public class MainAppTest {
    @Test
    public void testApp() {
        assertTrue(true);
    }
}
```

```
package com.bibliotecas.model;

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

public class LibroTest {
    @Test
    public void testLibro() {
        assertTrue(true);
    }
}
```

3.

@Suite

```
package com.bibliotecas;

import org.junit.platform.suite.api.SelectClasses;
import org.junit.platform.suite.api.Suite;

@Suite
@SelectClasses({
    com.bibliotecas.app.MainAppTest.class,
    com.bibliotecas.model.LibroTest.class
})
public class TestSuite {
    // Esta clase no necesita código, solo actúa como
    contenedor del suite.
}
```

- @Suite

- `@SelectClasses`

4.

- `TestSuite`
- - `mvn test`
 - `gradle test`

6.7. Otras opciones para agrupar tests

`@SelectClasses`

- `@SelectPackages`

```
@Suite
@SelectPackages("com.bibliotecas")
public class TestSuite {
}
```

- `@IncludeTags` `@ExcludeTags`

```
@Suite
@IncludeTags("slow")
public class SlowTestsSuite {
}
```

6.8. Resumen

- `@Suite`
- `@SelectClasses`
-

7. MetricsReloaded

7.1. ¿Qué es MetricsReloaded?

7.2. Características principales

-
-
-
-

7.3. Instalación de MetricsReloaded

.zip

7.3.1. Instalación en modo online

- 1.
- 2.
- 3.
- 4.
- 5.

7.3.2. Instalación en modo offline

1. `.zip`
- 2.
- 3.
4. `.zip`
- 5.

7.4. Uso de MetricsReloaded

- 1.
- 2.
- 3.

Analyze

- 4.

8. Ejercicios Prácticos

8.1. Método `esPalindromo`

```
public class PalindromoUtil {  
  
    public boolean esPalindromo(String cadena) {  
        if (cadena == null) {  
            return false;  
        }  
        String limpia = cadena.replaceAll("\\s+", "").toLowerCase();  
        return limpia.equals(new  
StringBuilder(limpia).reverse().toString());  
    }  
}
```



8.2. Método `calcularFactorial`

```
public class MatematicaUtil {  
  
    public int calcularFactorial(int n) {  
        if (n < 0) {  
            throw new IllegalArgumentException("El número no puede  
ser negativo.");  
        }  
        return (n == 0) ? 1 : n * calcularFactorial(n - 1);  
    }  
}
```



9. Recursos Adicionales

9.1. Documentación Oficial de JUnit

- -

9.2. Libros y Tutoriales Recomendados

1.
 -
 -
2.
 -
 -

9.3. Comunidades y Foros para Resolver Dudas

1.
 -
 -
2.
 -
 -

