**Testing for Expected Exceptions**

test whether the code throws an exception when invalid input is provided.

**Test Class: SimpleMath.java**

public class SimpleMath {

public int divide(int a, int b) {

if (b == 0) {

throw new ArithmeticException("Cannot divide by zero");

}

return a / b;

}

}

**Test Class: SimpleMathTest.java**

import org.junit.jupiter.api.Test;

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

public class SimpleMathTest {

@Test

public void testDivideByZero() {

SimpleMath math = new SimpleMath();

ArithmeticException thrown = assertThrows(ArithmeticException.class, () -> {

math.divide(5, 0);

});

assertEquals("Cannot divide by zero", thrown.getMessage());

}

}

**Test Setup and Teardown with @BeforeEach and @AfterEach**

**Test Class: SimpleMathTest.java**

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.Test;

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

public class SimpleMathTest {

private SimpleMath math;

@BeforeEach

public void setup() {

math = new SimpleMath(); // Setup before each test

}

@AfterEach

public void teardown() {

math = null; // Cleanup after each test

}

@Test

public void testAdd() {

int result = math.add(3, 2);

assertEquals(5, result);

}

@Test

public void testDivide() {

int result = math.divide(6, 2);

assertEquals(3, result);

}

}

**Parameterized Tests**

JUnit 5 provides parameterized tests, where the same test logic can be applied to multiple sets of data.

**Test Class: SimpleMathTest.java**

import org.junit.jupiter.api.ParameterizedTest;

import org.junit.jupiter.api.Arguments;

import org.junit.jupiter.api.MethodSource;

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

import java.util.stream.Stream;

public class SimpleMathTest {

@ParameterizedTest

@MethodSource("provideNumbersForAddition")

public void testAdd(int a, int b, int expected) {

SimpleMath math = new SimpleMath();

int result = math.add(a, b);

assertEquals(expected, result);

}

private static Stream<Arguments> provideNumbersForAddition() {

return Stream.of(

Arguments.of(1, 1, 2),

Arguments.of(2, 3, 5),

Arguments.of(3, 4, 7),

Arguments.of(-1, 1, 0)

);

}

}

* @ParameterizedTest marks the method as a parameterized test.
* @MethodSource("methodName") specifies a method that provides input arguments to the test.
* The Arguments.of() method is used to create the input data for each test execution.

**Using Assertions to Compare Arrays**

This example shows how to assert equality for arrays.

**Test Class: SimpleMath.java**

public class SimpleMath {

public int[] reverseArray(int[] arr) {

int[] reversed = new int[arr.length];

for (int i = 0; i < arr.length; i++) {

reversed[i] = arr[arr.length - 1 - i];

}

return reversed;

}

}

**Test Class: SimpleMathTest.java**

import org.junit.jupiter.api.Test;

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

public class SimpleMathTest {

@Test

public void testReverseArray() {

SimpleMath math = new SimpleMath();

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

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

assertArrayEquals(expected, math.reverseArray(input), "Arrays should be equal after reversing");

}

}

**Test with Timeout**

In this example, we set a timeout for the test. If the test takes longer than the specified time, it will fail.

**Test Class: SimpleMathTest.java**

import org.junit.jupiter.api.Test;

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

public class SimpleMathTest {

@Test

public void testTimeout() {

assertTimeoutPreemptively(java.time.Duration.ofMillis(100), () -> {

// Simulate some work that might take too long

Thread.sleep(50);

});

}

}

.