**JUnit Testing Exercises**

**Exercise 1: Setting Up JUnit**

## Step 1: Creating a New Java Project

Use your preferred IDE (IntelliJ IDEA, Eclipse).

Name your project JUnitExample.

## Step 2: Add JUnit Dependency using Maven

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope></dependency>

Full pom.xml:

xml

CopyEdit

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>JUnitExample</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit Dependency -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies></project>

## Step 3: Write Java Code to be Tested

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public int divide(int a, int b) {

if (b == 0) throw new ArithmeticException("Cannot divide by zero");

return a / b;

}

}

## Step 4: Write JUnit Test Class

import org.junit.Test;import static org.junit.Assert.\*;

public class CalculatorTest {

Calculator calc = new Calculator();

@Test

public void testAdd() {

assertEquals(10, calc.add(6, 4));

}

@Test

public void testSubtract() {

assertEquals(2, calc.subtract(5, 3));

}

@Test

public void testMultiply() {

assertEquals(20, calc.multiply(4, 5));

}

@Test

public void testDivide() {

assertEquals(3, calc.divide(9, 3));

}

@Test(expected = ArithmeticException.class)

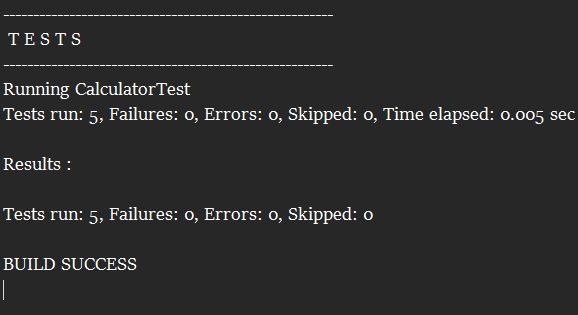
public void testDivideByZero() {

calc.divide(10, 0);

}

}

**OUTPUT:**



**Exercise 2: Writing Basic JUnit Test**

## Step 1: Creating a Simple Java Class

public class StringUtils {

// Returns true if the string is a palindrome

public boolean isPalindrome(String input) {

if (input == null) return false;

String reversed = new StringBuilder(input).reverse().toString();

return input.equals(reversed);

}

// Returns true if the string is empty or null

public boolean isEmpty(String input) {

return input == null || input.trim().isEmpty();

}

// Converts a string to uppercase

public String toUpperCase(String input) {

return input == null ? null : input.toUpperCase();

}

}

## Step 2: Write JUnit Test Class

import org.junit.Test;import static org.junit.Assert.\*;

public class StringUtilsTest {

StringUtils utils = new StringUtils();

@Test

public void testIsPalindrome() {

assertTrue(utils.isPalindrome("madam"));

assertTrue(utils.isPalindrome("racecar"));

assertFalse(utils.isPalindrome("hello"));

assertFalse(utils.isPalindrome(null));

}

@Test

public void testIsEmpty() {

assertTrue(utils.isEmpty(""));

assertTrue(utils.isEmpty(" "));

assertTrue(utils.isEmpty(null));

assertFalse(utils.isEmpty("text"));

}

@Test

public void testToUpperCase() {

assertEquals("HELLO", utils.toUpperCase("hello"));

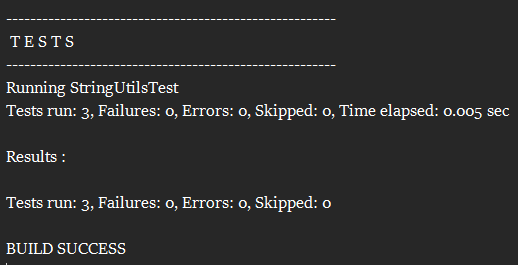
assertEquals("JAVA", utils.toUpperCase("Java"));

assertNull(utils.toUpperCase(null));

}

}

**OUTPUT:**



**Exercise 3: Assertions in JUnit**

import org.junit.Test;

import static org.junit.Assert.\*;

public class MyAssertionsTest {

@Test

public void testCustomAssertions() {

// assertEquals: check if two values are equal

int expectedSum = 15;

int actualSum = 10 + 5;

assertEquals(expectedSum, actualSum);

// assertTrue: check if a condition is true

String name = "Bindhu";

assertTrue(name.startsWith("B"));

// assertFalse: check if a condition is false

int age = 18;

assertFalse(age < 10);

// assertNull: check if value is null

String empty = null;

assertNull(empty);

// assertNotNull: check if object is not null

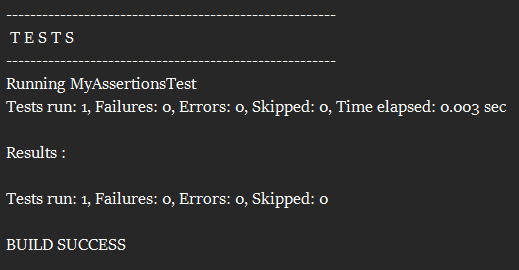
String message = "Hello JUnit";

assertNotNull(message);

}

}

**OUTPUT:**



**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit**

**Writing tests using the AAA pattern**

public class BankAccount {

private int balance;

public BankAccount(int initialBalance) {

this.balance = initialBalance;

}

public void deposit(int amount) {

if (amount > 0)

balance += amount;

}

public void withdraw(int amount) {

if (amount > 0 && amount <= balance)

balance -= amount;

}

public int getBalance() {

return balance;

}

}

**Using @Before and @After annotations for setup and teardown methods**

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class BankAccountTest {

private BankAccount account;

// Setup method – runs before each test

@Before

public void setUp() {

account = new BankAccount(100); // Arrange

System.out.println("Before: New account created with balance 100");

}

// Teardown method – runs after each test

@After

public void tearDown() {

account = null;

System.out.println("After: Account reset to null\n");

}

@Test

public void testDeposit() {

// Arrange: already done in setUp()

// Act

account.deposit(50);

// Assert

assertEquals(150, account.getBalance());

}

@Test

public void testWithdraw() {

// Arrange: already done in setUp()

// Act

account.withdraw(40);

// Assert

assertEquals(60, account.getBalance());

}

@Test

public void testWithdrawMoreThanBalance() {

// Act

account.withdraw(200);

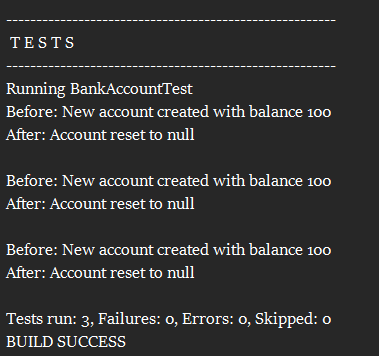
// Assert

assertEquals(100, account.getBalance()); // should not change

}

}

**OUTPUT:**



Advanced JUnit Testing Exercises

**Exercise 1: Parameterized Tests**

**Step 1: EvenChecker.java**

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**Step 2: EvenCheckerTest.java with JUnit 5**

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class EvenCheckerTest {

EvenChecker evenChecker = new EvenChecker();

@ParameterizedTest

@ValueSource(ints = {2, 4, 6, 8, 10, 0, -2})

void testIsEven\_WithEvenNumbers(int number) {

assertTrue(evenChecker.isEven(number), number + " should be even");

}

@ParameterizedTest

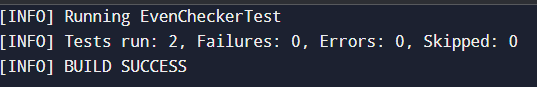
@ValueSource(ints = {1, 3, 5, 7, -1, -3})

void testIsEven\_WithOddNumbers(int number) {

assertFalse(evenChecker.isEven(number), number + " should be odd");

}

}

**OUTPUT:**

**Exercise 2: Test Suites and Categories**

### Step 1: Create Test Classes (Example: EvenCheckerTest and MathUtilsTest)

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class EvenCheckerTest {

EvenChecker checker = new EvenChecker();

@ParameterizedTest

@ValueSource(ints = {2, 4, 6})

void testEvenNumbers(int number) {

assertTrue(checker.isEven(number));

}

@ParameterizedTest

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

void testOddNumbers(int number) {

assertFalse(checker.isEven(number));

}

}

import org.junit.jupiter.api.Test;

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

public class MathUtilsTest {

@Test

void testAddition() {

assertEquals(5, 2 + 3);

}

@Test

void testSubtraction() {

assertEquals(1, 3 - 2);

}

}

### Step 2: Create AllTests.java Suite Class

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

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

@Suite

@SelectClasses({

EvenCheckerTest.class,

MathUtilsTest.class

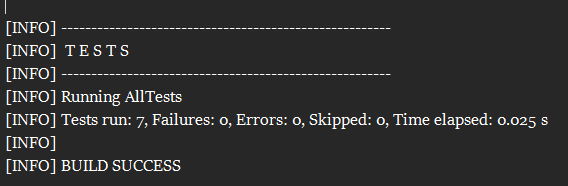
})

public class AllTests {

// This class remains empty. It is used only as a holder for the above annotations.

}

**OUTPUT:**

****

**Exercise 3: Test Execution Order**

## **Step 1: Create a test class OrderedTests**

import org.junit.jupiter.api.Test;

public class OrderedTests {

@Test

void testA() {

System.out.println("Running testA");

}

@Test

void testB() {

System.out.println("Running testB");

}

@Test

void testC() {

System.out.println("Running testC");

}

}

## Step 2: Use @TestMethodOrder and @Order

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

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)

public class OrderedTests {

@Test

@Order(3)

void testC() {

System.out.println("Test 3: C");

}

@Test

@Order(1)

void testA() {

System.out.println("Test 1: A");

}

@Test

@Order(2)

void testB() {

System.out.println("Test 2: B");

}

}

**OUTPUT:**



**Exercise 4: Exception Testing**

### Step 1: Create the ExceptionThrower class

public class ExceptionThrower {

public void throwException(String input) {

if (input == null) {

throw new IllegalArgumentException("Input cannot be null");

}

Other logic (if needed)

System.out.println("Input is: " + input);

}

}

### Step 2: Create the test class ExceptionThrowerTest

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

ExceptionThrower thrower = new ExceptionThrower();

@Test

void testThrowsExceptionWhenInputIsNull() {

Exception exception = assertThrows(IllegalArgumentException.class, () -> {

thrower.throwException(null);

});

assertEquals("Input cannot be null", exception.getMessage());

}

@Test

void testDoesNotThrowExceptionWhenInputIsValid() {

assertDoesNotThrow(() -> {

thrower.throwException("Hello");

});

}

}

**OUTPUT:**

### 

**Exercise 5: Timeout and Performance Testing**

## **Step 1: Create the PerformanceTester class**

public class PerformanceTester {

public void performTask() {

try {

// Simulate a task that takes 300 milliseconds

Thread.sleep(300);

} catch (InterruptedException e) {

Thread.currentThread().interrupt(); // Best practice

}

}

}

**Step 2: Create the test class PerformanceTesterTest**

import org.junit.jupiter.api.Test;

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

import java.time.Duration;

public class PerformanceTesterTest {

PerformanceTester tester = new PerformanceTester();

@Test

void testPerformTaskCompletesWithin500ms() {

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

tester.performTask();

});

}

}

**OUTPUT:**

