P Karthiyayini

Superset ID:

**Exercise 1: Setting Up JUnit**

**Scenario:**

**You need to set up JUnit in your Java project to start writing unit tests.**

**Steps:**

**1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).**

**2. Add JUnit dependency to your project. If you are using Maven, add the following to your**

**pom.xml:**

**<dependency>**

**<groupId>junit</groupId>**

**<artifactId>junit</artifactId>**

**<version>4.13.2</version>**

**<scope>test</scope>**

**</dependency>**

**3. Create a new test class in your project**

Code:

<!-- Maven pom.xml dependency -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

<!-- Sample Calculator class (src/main/java/Calculator.java) -->

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 double divide(int a, int b) {

if (b == 0) {

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

}

return (double) a / b;

}

}

<!-- Test class (src/test/java/CalculatorTest.java) -->

import org.junit.Test;

import org.junit.Before;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

}

@Test

public void testAdd() {

int result = calculator.add(5, 3);

assertEquals(8, result);

}

@Test

public void testSubtract() {

int result = calculator.subtract(10, 4);

assertEquals(6, result);

}

@Test

public void testMultiply() {

int result = calculator.multiply(6, 7);

assertEquals(42, result);

}

@Test

public void testDivide() {

double result = calculator.divide(15, 3);

assertEquals(5.0, result, 0.001);

}

@Test(expected = ArithmeticException.class)

public void testDivideByZero() {

calculator.divide(10, 0);

}

}

**Exercise 2: Writing Basic JUnit Tests**

**Scenario:**

**You need to write basic JUnit tests for a simple Java class.**

**Steps:**

**1. Create a new Java class with some methods to test.**

**2. Write JUnit tests for these methods.**

Code:

// Step 1: Java class to test (src/main/java/StringUtils.java)

public class StringUtils {

public boolean isEmpty(String str) {

return str == null || str.length() == 0;

}

public String reverse(String str) {

if (str == null) {

return null;

}

return new StringBuilder(str).reverse().toString();

}

public boolean isPalindrome(String str) {

if (str == null) {

return false;

}

String cleaned = str.toLowerCase().replaceAll("[^a-zA-Z0-9]", "");

return cleaned.equals(new StringBuilder(cleaned).reverse().toString());

}

public int countVowels(String str) {

if (str == null) {

return 0;

}

int count = 0;

String vowels = "aeiouAEIOU";

for (char c : str.toCharArray()) {

if (vowels.indexOf(c) != -1) {

count++;

}

}

return count;

}

public String capitalize(String str) {

if (str == null || str.isEmpty()) {

return str;

}

return str.substring(0, 1).toUpperCase() + str.substring(1).toLowerCase();

}

}

// Step 2: JUnit tests (src/test/java/StringUtilsTest.java)

import org.junit.Test;

import org.junit.Before;

import org.junit.After;

import static org.junit.Assert.\*;

public class StringUtilsTest {

private StringUtils stringUtils;

@Before

public void setUp() {

stringUtils = new StringUtils();

}

@After

public void tearDown() {

stringUtils = null;

}

// Tests for isEmpty method

@Test

public void testIsEmpty\_NullString() {

assertTrue(stringUtils.isEmpty(null));

}

@Test

public void testIsEmpty\_EmptyString() {

assertTrue(stringUtils.isEmpty(""));

}

@Test

public void testIsEmpty\_NonEmptyString() {

assertFalse(stringUtils.isEmpty("hello"));

}

@Test

public void testIsEmpty\_WhitespaceString() {

assertFalse(stringUtils.isEmpty(" "));

}

// Tests for reverse method

@Test

public void testReverse\_NormalString() {

assertEquals("olleh", stringUtils.reverse("hello"));

}

@Test

public void testReverse\_SingleCharacter() {

assertEquals("a", stringUtils.reverse("a"));

}

@Test

public void testReverse\_EmptyString() {

assertEquals("", stringUtils.reverse(""));

}

@Test

public void testReverse\_NullString() {

assertNull(stringUtils.reverse(null));

}

@Test

public void testReverse\_PalindromeString() {

assertEquals("aba", stringUtils.reverse("aba"));

}

// Tests for isPalindrome method

@Test

public void testIsPalindrome\_SimplePalindrome() {

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

}

@Test

public void testIsPalindrome\_NotPalindrome() {

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

}

@Test

public void testIsPalindrome\_PalindromeWithSpaces() {

assertTrue(stringUtils.isPalindrome("A man a plan a canal Panama"));

}

@Test

public void testIsPalindrome\_EmptyString() {

assertTrue(stringUtils.isPalindrome(""));

}

@Test

public void testIsPalindrome\_NullString() {

assertFalse(stringUtils.isPalindrome(null));

}

@Test

public void testIsPalindrome\_MixedCase() {

assertTrue(stringUtils.isPalindrome("RaceCar"));

}

// Tests for countVowels method

@Test

public void testCountVowels\_AllVowels() {

assertEquals(5, stringUtils.countVowels("aeiou"));

}

@Test

public void testCountVowels\_NoVowels() {

assertEquals(0, stringUtils.countVowels("bcdfg"));

}

@Test

public void testCountVowels\_MixedString() {

assertEquals(2, stringUtils.countVowels("hello"));

}

@Test

public void testCountVowels\_EmptyString() {

assertEquals(0, stringUtils.countVowels(""));

}

@Test

public void testCountVowels\_NullString() {

assertEquals(0, stringUtils.countVowels(null));

}

@Test

public void testCountVowels\_UppercaseVowels() {

assertEquals(3, stringUtils.countVowels("HELLO"));

}

// Tests for capitalize method

@Test

public void testCapitalize\_LowercaseString() {

assertEquals("Hello", stringUtils.capitalize("hello"));

}

@Test

public void testCapitalize\_UppercaseString() {

assertEquals("Hello", stringUtils.capitalize("HELLO"));

}

@Test

public void testCapitalize\_MixedCaseString() {

assertEquals("Hello", stringUtils.capitalize("hELLO"));

}

@Test

public void testCapitalize\_SingleCharacter() {

assertEquals("A", stringUtils.capitalize("a"));

}

@Test

public void testCapitalize\_EmptyString() {

assertEquals("", stringUtils.capitalize(""));

}

@Test

public void testCapitalize\_NullString() {

assertNull(stringUtils.capitalize(null));

}

// Additional edge case tests

@Test

public void testMultipleMethods\_Integration() {

String input = "hello";

assertFalse(stringUtils.isEmpty(input));

assertEquals("olleh", stringUtils.reverse(input));

assertFalse(stringUtils.isPalindrome(input));

assertEquals(2, stringUtils.countVowels(input));

assertEquals("Hello", stringUtils.capitalize(input));

}

}

**Exercise 3: Assertions in JUnit**

**Scenario:**

**You need to use different assertions in JUnit to validate your test results.**

**Steps:**

**1. Write tests using various JUnit assertions.**

**Solution Code:**

**public class AssertionsTest {**

**@Test**

**public void testAssertions() {**

**// Assert equals**

**assertEquals(5, 2 + 3);**

**// Assert true**

**assertTrue(5 > 3);**

**// Assert false**

**assertFalse(5 < 3);**

**// Assert null**

**assertNull(null);**

**// Assert not null**

**assertNotNull(new Object());**

**}**

**}**

Code:

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testBasicAssertions() {

// Assert equals

assertEquals(5, 2 + 3);

assertEquals("hello", "he" + "llo");

assertEquals(3.14, Math.PI, 0.01); // Delta for floating point

// Assert true

assertTrue(5 > 3);

assertTrue("hello".startsWith("he"));

// Assert false

assertFalse(5 < 3);

assertFalse("hello".endsWith("lo"));

// Assert null

assertNull(null);

String nullString = null;

assertNull(nullString);

// Assert not null

assertNotNull(new Object());

assertNotNull("hello");

}

@Test

public void testStringAssertions() {

String expected = "Hello World";

String actual = "Hello" + " " + "World";

assertEquals(expected, actual);

assertEquals(11, actual.length());

assertTrue(actual.contains("World"));

assertFalse(actual.isEmpty());

assertNotNull(actual);

}

@Test

public void testArrayAssertions() {

int[] expected = {1, 2, 3, 4, 5};

int[] actual = {1, 2, 3, 4, 5};

assertArrayEquals(expected, actual);

assertEquals(5, actual.length);

assertTrue(actual[0] == 1);

assertFalse(actual.length == 0);

}

@Test

public void testFloatingPointAssertions() {

double result = 10.0 / 3.0;

assertEquals(3.333, result, 0.001);

assertTrue(result > 3.3);

assertFalse(result > 3.4);

assertNotNull(result);

}

@Test

public void testObjectAssertions() {

String str1 = new String("hello");

String str2 = new String("hello");

String str3 = str1;

assertEquals(str1, str2); // Content equality

assertNotSame(str1, str2); // Different objects

assertSame(str1, str3); // Same object reference

assertNotNull(str1);

}

@Test

public void testBooleanAssertions() {

boolean isValid = true;

boolean isEmpty = false;

assertTrue(isValid);

assertFalse(isEmpty);

assertEquals(true, isValid);

assertEquals(false, isEmpty);

assertNotNull(isValid);

}

@Test

public void testCollectionAssertions() {

java.util.List<String> list = java.util.Arrays.asList("a", "b", "c");

assertNotNull(list);

assertEquals(3, list.size());

assertTrue(list.contains("a"));

assertFalse(list.contains("d"));

assertEquals("a", list.get(0));

}

@Test

public void testCustomMessageAssertions() {

int age = 25;

assertTrue("Age should be positive", age > 0);

assertEquals("Age should be 25", 25, age);

assertFalse("Age should not be negative", age < 0);

assertNotNull("Age should not be null", Integer.valueOf(age));

}

@Test

public void testFailAssertion() {

boolean shouldNotReachHere = false;

if (shouldNotReachHere) {

fail("This should never be reached");

}

// This will pass

assertTrue(true);

}

@Test(expected = IllegalArgumentException.class)

public void testExceptionAssertion() {

throw new IllegalArgumentException("Expected exception");

}

@Test

public void testMultipleAssertions() {

String text = "JUnit Testing";

assertNotNull("Text should not be null", text);

assertTrue("Text should not be empty", !text.isEmpty());

assertEquals("Text length should be 13", 13, text.length());

assertTrue("Text should contain 'JUnit'", text.contains("JUnit"));

assertFalse("Text should not contain 'Python'", text.contains("Python"));

assertEquals("Text should start with 'JUnit'", "JUnit", text.substring(0, 5));

}

}

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

**Teardown Methods in JUnit**

**Scenario:**

**You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup**

**and teardown methods.**

**Steps:**

**1. Write tests using the AAA pattern.**

**2. Use @Before and @After annotations for setup and teardown methods.**

Code:

import org.junit.\*;

import static org.junit.Assert.\*;

import java.util.\*;

// Class to test

class BankAccount {

private String accountNumber;

private double balance;

private List<String> transactions;

public BankAccount(String accountNumber) {

this.accountNumber = accountNumber;

this.balance = 0.0;

this.transactions = new ArrayList<>();

}

public void deposit(double amount) {

if (amount <= 0) {

throw new IllegalArgumentException("Amount must be positive");

}

balance += amount;

transactions.add("Deposit: $" + amount);

}

public void withdraw(double amount) {

if (amount <= 0) {

throw new IllegalArgumentException("Amount must be positive");

}

if (amount > balance) {

throw new IllegalStateException("Insufficient funds");

}

balance -= amount;

transactions.add("Withdrawal: $" + amount);

}

public double getBalance() {

return balance;

}

public String getAccountNumber() {

return accountNumber;

}

public List<String> getTransactions() {

return new ArrayList<>(transactions);

}

public void transfer(BankAccount toAccount, double amount) {

withdraw(amount);

toAccount.deposit(amount);

transactions.add("Transfer to " + toAccount.getAccountNumber() + ": $" + amount);

}

}

// Test class using AAA pattern with setup and teardown

public class BankAccountTest {

// Test fixtures

private BankAccount account;

private BankAccount secondAccount;

private final String ACCOUNT\_NUMBER = "123456789";

private final String SECOND\_ACCOUNT\_NUMBER = "987654321";

@BeforeClass

public static void setUpClass() {

System.out.println("Setting up test class - runs once before all tests");

}

@AfterClass

public static void tearDownClass() {

System.out.println("Tearing down test class - runs once after all tests");

}

@Before

public void setUp() {

System.out.println("Setting up test - runs before each test");

// Arrange - Create test fixtures

account = new BankAccount(ACCOUNT\_NUMBER);

secondAccount = new BankAccount(SECOND\_ACCOUNT\_NUMBER);

}

@After

public void tearDown() {

System.out.println("Tearing down test - runs after each test");

// Clean up resources

account = null;

secondAccount = null;

}

@Test

public void testCreateAccount() {

// Arrange - already done in setUp()

// Act - account is created in setUp()

// Assert

assertEquals(ACCOUNT\_NUMBER, account.getAccountNumber());

assertEquals(0.0, account.getBalance(), 0.001);

assertTrue(account.getTransactions().isEmpty());

}

@Test

public void testDeposit() {

// Arrange

double depositAmount = 100.0;

double expectedBalance = 100.0;

// Act

account.deposit(depositAmount);

// Assert

assertEquals(expectedBalance, account.getBalance(), 0.001);

assertEquals(1, account.getTransactions().size());

assertTrue(account.getTransactions().get(0).contains("Deposit: $100.0"));

}

@Test

public void testMultipleDeposits() {

// Arrange

double firstDeposit = 50.0;

double secondDeposit = 75.0;

double expectedBalance = 125.0;

// Act

account.deposit(firstDeposit);

account.deposit(secondDeposit);

// Assert

assertEquals(expectedBalance, account.getBalance(), 0.001);

assertEquals(2, account.getTransactions().size());

}

@Test

public void testWithdraw() {

// Arrange

double initialDeposit = 200.0;

double withdrawAmount = 50.0;

double expectedBalance = 150.0;

account.deposit(initialDeposit);

// Act

account.withdraw(withdrawAmount);

// Assert

assertEquals(expectedBalance, account.getBalance(), 0.001);

assertEquals(2, account.getTransactions().size());

assertTrue(account.getTransactions().get(1).contains("Withdrawal: $50.0"));

}

@Test(expected = IllegalArgumentException.class)

public void testDepositNegativeAmount() {

// Arrange

double negativeAmount = -50.0;

// Act

account.deposit(negativeAmount);

// Assert - exception expected

}

@Test(expected = IllegalStateException.class)

public void testWithdrawInsufficientFunds() {

// Arrange

double withdrawAmount = 100.0;

// Account starts with 0 balance

// Act

account.withdraw(withdrawAmount);

// Assert - exception expected

}

@Test

public void testTransfer() {

// Arrange

double initialAmount = 300.0;

double transferAmount = 100.0;

double expectedSenderBalance = 200.0;

double expectedReceiverBalance = 100.0;

account.deposit(initialAmount);

// Act

account.transfer(secondAccount, transferAmount);

// Assert

assertEquals(expectedSenderBalance, account.getBalance(), 0.001);

assertEquals(expectedReceiverBalance, secondAccount.getBalance(), 0.001);

assertEquals(3, account.getTransactions().size()); // deposit, withdraw, transfer log

assertEquals(1, secondAccount.getTransactions().size()); // deposit from transfer

}

@Test

public void testComplexScenario() {

// Arrange

double deposit1 = 500.0;

double deposit2 = 200.0;

double withdrawal1 = 150.0;

double transferAmount = 100.0;

double expectedFinalBalance = 450.0; // 500 + 200 - 150 - 100

// Act

account.deposit(deposit1);

account.deposit(deposit2);

account.withdraw(withdrawal1);

account.transfer(secondAccount, transferAmount);

// Assert

assertEquals(expectedFinalBalance, account.getBalance(), 0.001);

assertEquals(100.0, secondAccount.getBalance(), 0.001);

assertEquals(5, account.getTransactions().size());

assertFalse(account.getTransactions().isEmpty());

}

@Test

public void testGetTransactionsReturnsDefensiveCopy() {

// Arrange

account.deposit(100.0);

// Act

List<String> transactions = account.getTransactions();

transactions.clear(); // Try to modify returned list

// Assert

assertEquals(1, account.getTransactions().size()); // Original should be unchanged

}

@Test

public void testAccountNumberImmutable() {

// Arrange

String originalAccountNumber = account.getAccountNumber();

// Act

account.deposit(100.0);

account.withdraw(50.0);

// Assert

assertEquals(originalAccountNumber, account.getAccountNumber());

}

}