JUnit Testing Exercises

# 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>

1. Create a new test class in your project.

**Program:**

**Calculator.java:**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

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

}

}

**.replit:**

run = "javac -cp '.:junit-4.13.2.jar:hamcrest-core-1.3.jar' Calculator.java CalculatorTest.java && java -cp '.:junit-4.13.2.jar:hamcrest-core-1.3.jar' org.junit.runner.JUnitCore CalculatorTest"

**replit.nix:**

{ pkgs }: {

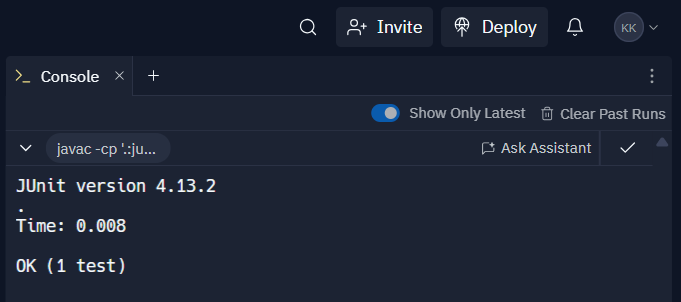
deps = [

pkgs.openjdk

];

}

**Output:**



# 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());

}

}

**Program:**

**AssertionsTest.java:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

assertNull(null);

assertNotNull(new Object());

}

}

**.replit:**

run = "javac -cp '.:junit-4.13.2.jar:hamcrest-core-1.3.jar' AssertionsTest.java && java -cp '.:junit-4.13.2.jar:hamcrest-core-1.3.jar' org.junit.runner.JUnitCore AssertionsTest"

**replit.nix:**

{ pkgs }: {

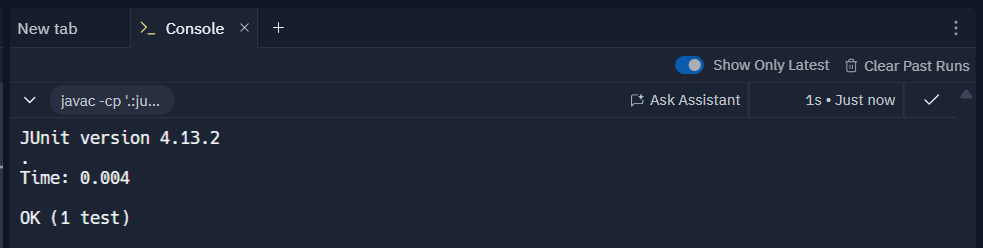
deps = [

pkgs.openjdk

];

}

**Output:**



# 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.

**Program:**

**Calculator.java:**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

**CalculatorTest.java:**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

System.out.println("Setup: Calculator created");

}

@After

public void tearDown() {

calculator = null;

System.out.println("Teardown: Calculator cleared");

}

@Test

public void testAddition() {

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

assertEquals(5, result);

}

@Test

public void testMultiplication() {

int result = calculator.multiply(4, 2);

assertEquals(8, result);

}

}

**.replit:**

run = "javac -cp '.:junit-4.13.2.jar:hamcrest-core-1.3.jar' Calculator.java CalculatorTest.java && java -cp '.:junit-4.13.2.jar:hamcrest-core-1.3.jar' org.junit.runner.JUnitCore CalculatorTest"

**replit.nix:**

{ pkgs }: {

deps = [

pkgs.openjdk

];

}

**Output:**

