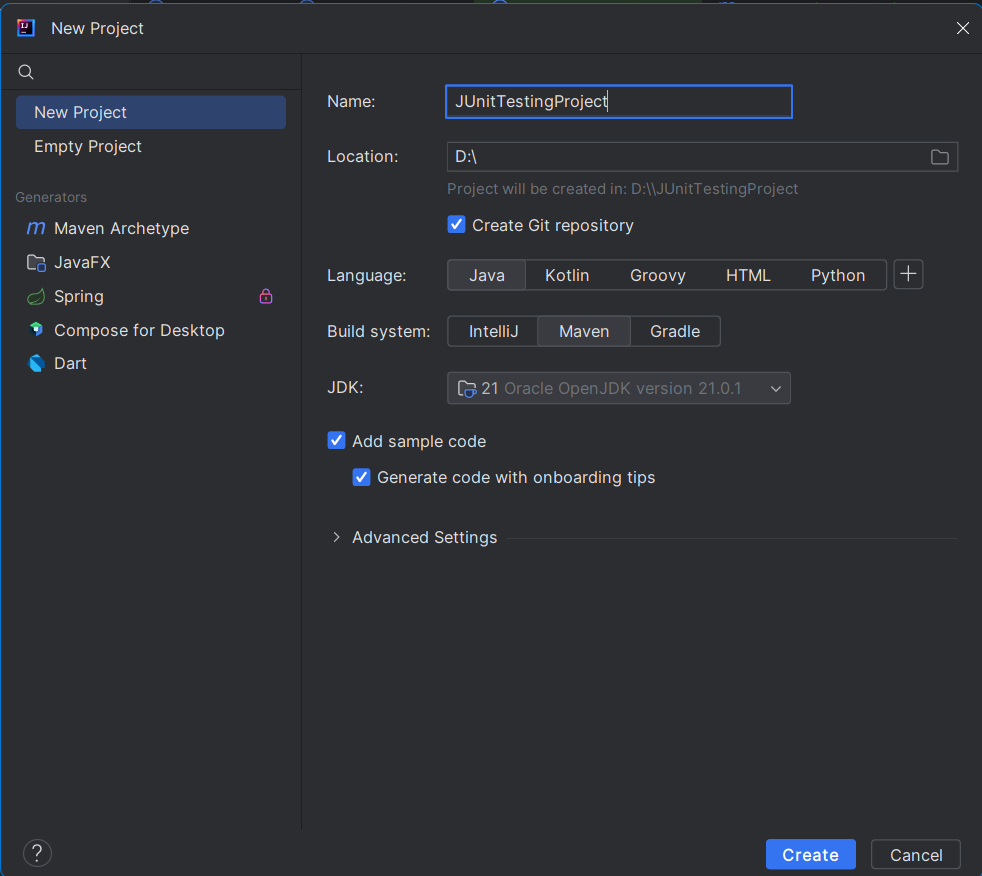
**Junit Basic Testing Exercises**

**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: junit junit 4.13.2 test

<dependency> <groupId>junit</groupId> <artifactId>junit</artifactId> <version>4.13.2</version> <scope>test</scope> </dependency>

pom.xml:

<?xml version="1.0" encoding="UTF-8"?>  
<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>org.example</groupId>  
 <artifactId>JUnitTest</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>21</maven.compiler.source>  
 <maven.compiler.target>21</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
</project>

Calculator.java

public class Calculator {  
  
 public double add(double a, double b) {  
 return a + b;  
 }  
  
  
 public double subtract(double a, double b) {  
 return a - b;  
 }  
  
  
 public double multiply(double a, double b) {  
 return a \* b;  
 }  
   
 public double divide(double a, double b) {  
 if (b == 0) {  
 throw new ArithmeticException("Cannot divide by zero.");  
 }  
 return a / b;  
 }  
}

CalculatorTest.java

import org.junit.Assert;

import org.junit.Test;

import static org.junit.Assert.\*;

public class ServiceTest {

Calculator calc = new Calculator();  
  
@Test  
public void testAddition() {  
 *assertEquals*(15.0, calc.add(10, 5), 0.0001);  
 *assertEquals*(-2.0, calc.add(-1, -1), 0.0001);  
}  
  
@Test  
public void testSubtraction() {  
 *assertEquals*(5.0, calc.subtract(10, 5), 0.0001);  
 *assertEquals*(0.0, calc.subtract(5, 5), 0.0001);  
}  
  
@Test  
public void testMultiplication() {  
 *assertEquals*(50.0, calc.multiply(10, 5), 0.0001);  
 *assertEquals*(0.0, calc.multiply(10, 0), 0.0001);  
}  
  
@Test  
public void testDivision() {  
 *assertEquals*(2.0, calc.divide(10, 5), 0.0001);  
 *assertEquals*(0.5, calc.divide(1, 2), 0.0001);  
}  
  
@Test  
public void testDivisionByZero() {  
 Exception exception = *assertThrows*(ArithmeticException.class, () -> {  
 calc.divide(10, 0);  
 });  
 *assertEquals*("Cannot divide by zero.", exception.getMessage());  
}}

**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.

**BasicAssertionTest.java**

public class BasicAssertionTest {

@Test

public void testVariousAssertions() {

assertEquals("Sum should be 7", 7, 4 + 3);

assertTrue("6 is greater than 2", 6 > 2);

assertFalse("5 is not less than 3", 5 < 3);

assertNull("Object should be null", null);

assertNotNull("Object should not be null", new Object());

}

}

**NumberChecker.java**

package org.example;

public class NumberChecker {

public String getOddOrEven(int number) {

return number % 2 == 0 ? "even" : "odd";

}

public boolean isOdd(int number) {

return number % 2 != 0;

}

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**OddEvenTest.java**

package org.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class OddEvenTest {

OddEven o = new OddEven();

@Test

public void oddevenEvenTest() {

assertEquals("even", o.oddeven(2));

}

@Test

public void oddevenOddTest() {

assertEquals("odd", o.oddeven(7));

}

@Test

public void oddOddTest() {

assertTrue(o.odd(7));

}

@Test

public void oddEvenTest() {

assertFalse(o.odd(8));

}

@Test

public void evenEvenTest() {

assertTrue(o.even(8));

}

@Test

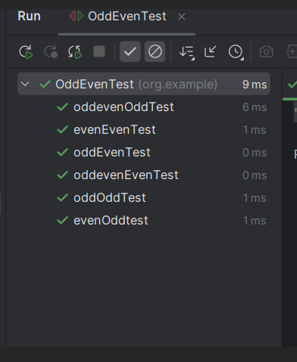
public void evenOddTest() {

assertFalse(o.even(7));

}

}

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.

**Calculator.java**

package org.example;  
  
public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
  
 public int subtract(int a, int b) {  
 return a - b;  
 }  
}

**CalculatorTest.java**

package org.example;  
import org.junit.After;  
import org.junit.Before;  
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 Complete");  
 }  
 @After  
 public void tearDown() {  
 System.*out*.println("Teardown Complete");  
 }  
 @Test  
 public void testAdd() {  
 int result = calculator.add(10, 5);  
 *assertEquals*(15, result);  
 }  
 @Test  
 public void testSubtract() {  
 int result = calculator.subtract(10, 3);  
 *assertEquals*(7, result);  
 }  
}

Output:

