**JUNIT EXERCISES FOR JAVA PROJECT**

# Introduction

JUnit is a popular testing framework for Java. It enables developers to write and run repeatable tests. This document demonstrates the basic setup and usage of JUnit including various assertion techniques.

# Objectives

- Understand how to set up JUnit in a Java project.

- Learn to write unit tests using JUnit.

- Explore different assertion methods to validate logic.

# Implementation

## Exercise 1: Setting Up JUnit

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

1. Steps:

* Create a new Java project in your IDE (Eclipse).
* 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>

* Create a new test class in your project.

**PasswordValidator.java:**

public class PasswordValidator {

public boolean isValid(String password) {

if (password == null || password.length() < 8) return false;

boolean hasDigit = false;

boolean hasSpecial = false;

for (char ch : password.toCharArray()) {

if (Character.isDigit(ch)) hasDigit = true;

if (!Character.isLetterOrDigit(ch)) hasSpecial = true;

}

return hasDigit && hasSpecial;

}

public String evaluate(String password) {

if (isValid(password)) {

return "Strong password: " + password;

} else {

return "Weak password: " + password;

}

}

}

**PasswordValidatorTest.java:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class PasswordValidatorTest {

PasswordValidator validator = new PasswordValidator();

@Test

public void testStrongPassword1() {

String pwd = "hello@123";

System.out.println(validator.evaluate(pwd));

assertTrue(validator.isValid(pwd));

}

@Test

public void testShortPassword() {

String pwd = "hi@12";

System.out.println(validator.evaluate(pwd));

assertFalse(validator.isValid(pwd));

}

@Test

public void testNoSpecialChar() {

String pwd = "hello1234";

System.out.println(validator.evaluate(pwd));

assertFalse(validator.isValid(pwd));

}

@Test

public void testNoDigit() {

String pwd = "hello@you";

System.out.println(validator.evaluate(pwd));

assertFalse(validator.isValid(pwd));

}

@Test

public void testOnlyLetters() {

String pwd = "abcdefgh";

System.out.println(validator.evaluate(pwd));

assertFalse(validator.isValid(pwd));

}

@Test

public void testOnlyDigits() {

String pwd = "12345678";

System.out.println(validator.evaluate(pwd));

assertFalse(validator.isValid(pwd));

}

@Test

public void testOnlySpecialChars() {

String pwd = "@#$%^&\*!";

System.out.println(validator.evaluate(pwd));

assertFalse(validator.isValid(pwd));

}

@Test

public void testNullPassword() {

String pwd = null;

System.out.println("Testing null password");

assertFalse(validator.isValid(pwd));

}

@Test

public void testEmptyPassword() {

String pwd = "";

System.out.println(validator.evaluate(pwd));

assertFalse(validator.isValid(pwd));

}

@Test

public void testStrongPassword2() {

String pwd = "P@ssw0rd!";

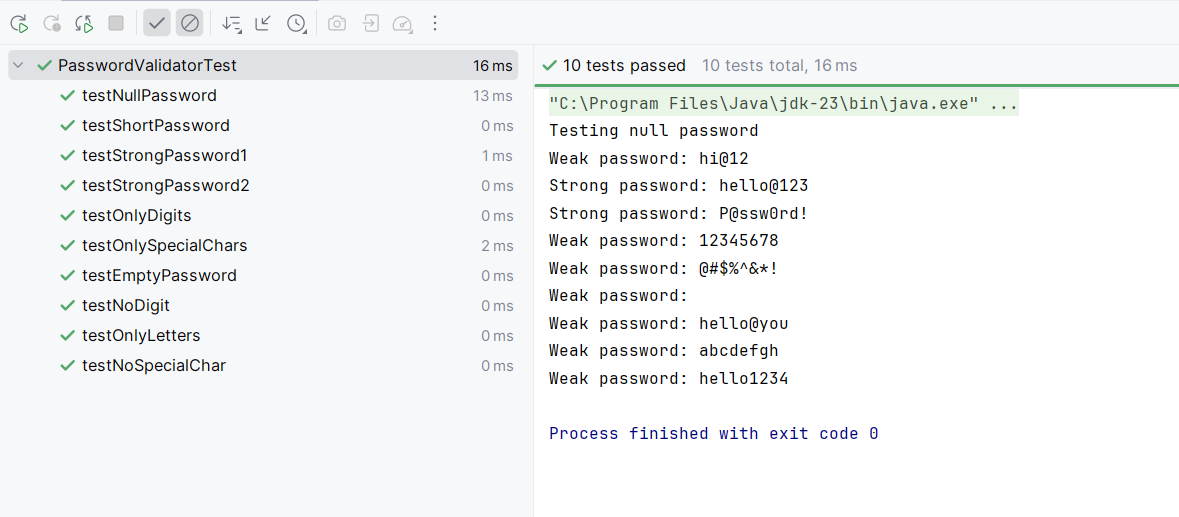
System.out.println(validator.evaluate(pwd));

assertTrue(validator.isValid(pwd));

}

}

**OUTPUT**

****

## EXERCISE 3: ASSERTIONS IN JUNIT

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

1. Steps:

* Write tests using various JUnit assertions.

**TemperatureConverter.java:**

public class TemperatureConverter {

public double toFahrenheit(double celsius) {

return (celsius \* 9 / 5) + 32;

}

public boolean isReasonableTemperature(double celsius) {

return celsius >= -90 && celsius <= 60;

}

}

**TemperatureConverterTest.java:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class TemperatureConverterTest {

TemperatureConverter converter = new TemperatureConverter();

@Test

public void testFreezingPoint() {

double result = converter.toFahrenheit(0);

System.out.println("Freezing point: " + result + "°F");

assertEquals(32.0, result, 0.01);

}

@Test

public void testBoilingPoint() {

double result = converter.toFahrenheit(100);

System.out.println("Boiling point: " + result + "°F");

assertEquals(212.0, result, 0.01);

}

@Test

public void testNegativeTemperature() {

double result = converter.toFahrenheit(-40);

System.out.println("Cold temp: " + result + "°F");

assertEquals(-40.0, result, 0.01);

}

@Test

public void testReasonableLow() {

double celsius = -60;

System.out.println("Checking if -60°C is reasonable");

assertTrue(converter.isReasonableTemperature(celsius));

}

@Test

public void testReasonableHigh() {

double celsius = 50;

System.out.println("Checking if 50°C is reasonable");

assertTrue(converter.isReasonableTemperature(celsius));

}

@Test

public void testUnreasonableLow() {

double celsius = -120;

System.out.println("Checking if -120°C is unreasonable");

assertFalse(converter.isReasonableTemperature(celsius));

}

@Test

public void testUnreasonableHigh() {

double celsius = 90;

System.out.println("Checking if 90°C is unreasonable");

assertFalse(converter.isReasonableTemperature(celsius));

}

@Test

public void testNullInputMessage() {

String input = null;

System.out.println("Checking null input...");

assertNull(input);

}

@Test

public void testNotNullInputMessage() {

String input = "25";

System.out.println("Checking non-null input...");

assertNotNull(input);

}

@Test

public void testApproximateRoomTemp() {

double result = converter.toFahrenheit(22);

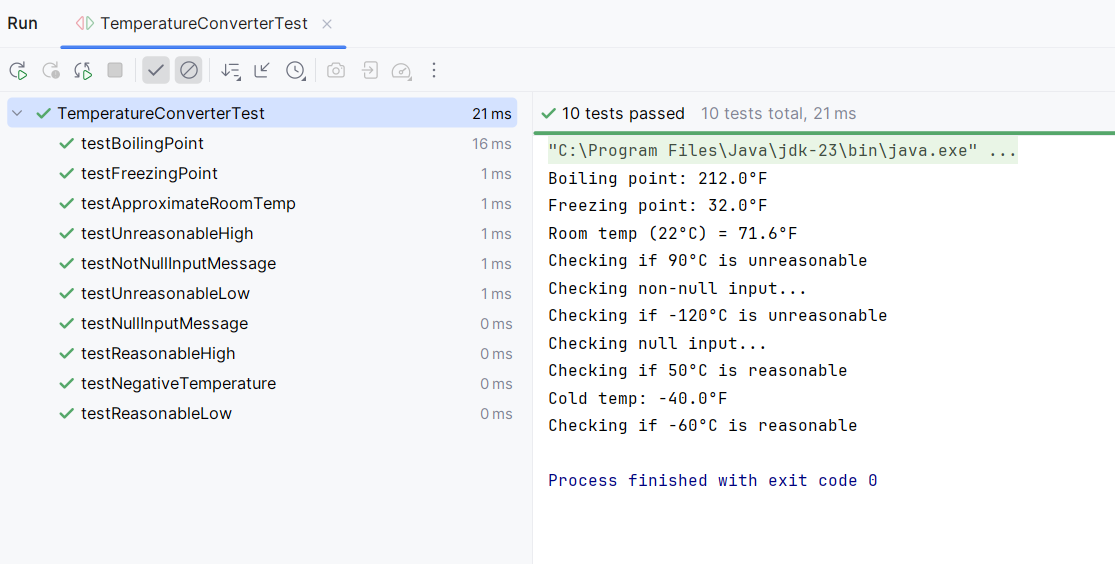
System.out.println("Room temp (22°C) = " + result + "°F");

assertEquals(71.6, result, 0.1);

}

}

**Output:**

****

## EXERCISE 4: AAA PATTERN, SETUP AND TEARDOWN

Scenario: You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

1. Steps:

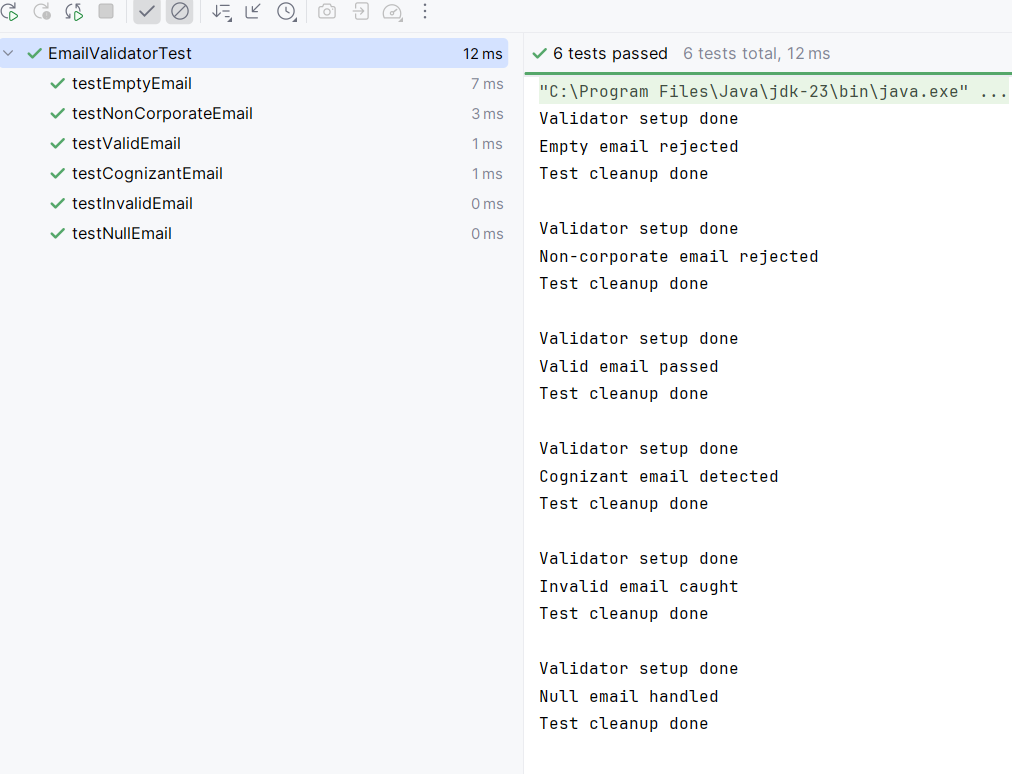
* Write tests using the AAA pattern.
* Use @Before and @After annotations for setup and teardown methods.

Solution Code:

import static org.junit.Assert.\*;  
import org.junit.Before;  
import org.junit.After;  
import org.junit.Test;  
  
public class CalculatorTest {  
  
 private Calculator calculator;  
  
 @Before  
 public void setUp() {  
 // Arrange: Initialize resources before each test  
 calculator = new Calculator();  
 }  
  
 @After  
 public void tearDown() {  
 // Clean up after each test  
 calculator = null;  
 }  
  
 @Test  
 public void testAddition() {  
 // Act: Call the method to test  
 int result = calculator.add(2, 3);  
  
 // Assert: Verify the result  
 assertEquals(5, result);  
 }  
  
 @Test  
 public void testSubtraction() {  
 int result = calculator.subtract(5, 3);  
 assertEquals(2, result);  
 }  
}  
  
class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
  
 public int subtract(int a, int b) {  
 return a - b;  
 }  
}

# Output

Upon running the test class, JUnit will validate the defined assertions. If all conditions hold true, the test will pass. Otherwise, it will provide detailed error messages for the failed assertions.

****

# Conclusion

Setting up JUnit and writing assertion-based tests enhances the reliability of Java applications. These unit tests help in early detection of bugs and encourage clean, maintainable code.