**Student Name:** KAMALESHWARAN A

**Registration No:** 22CSR087

**Course/Batch:** KONGU ENGINEERING COLLEGE (B.E COMPUTER SCIENCE AND ENGINEERING)

**EXERCISE 1: SETTING UP JUnit**

**Introduction:**

This Java program validates passwords based on security rules using a PasswordValidator class. Unit tests are written using JUnit to verify various password strength cases.

**Objective:**

* To implement password validation based on standard rules (length, digits, special characters).
* To evaluate password strength and provide feedback.
* To write and execute unit tests for strong and weak password cases.

**Implementation Breakdown:**

**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!";

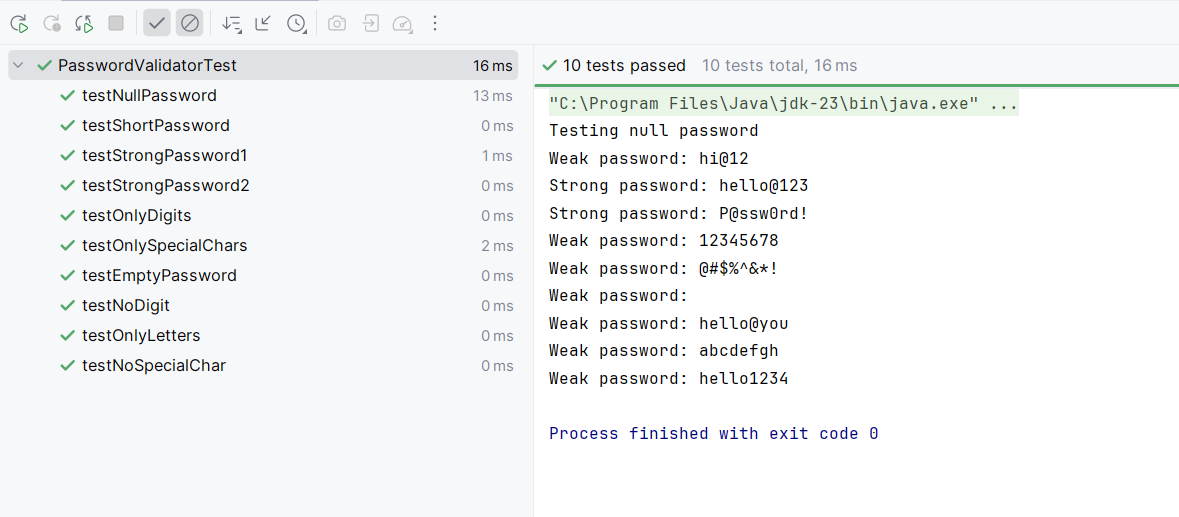
System.out.println(validator.evaluate(pwd));

assertTrue(validator.isValid(pwd));

}

}

**Output:**

****

**Conclusion:**

The implementation successfully validates password strength using simple logic. The test suite covers edge cases and ensures reliable behaviour for real-world password scenarios.

**EXERCISE 3: ASSERTIONS IN JUnit**

**Introduction:**

This Java application converts temperatures from Celsius to Fahrenheit and checks whether the temperature falls within a human-survivable range. Unit testing is done using JUnit to ensure accuracy of conversion and validation logic.

**Objective:**

* To implement a method that accurately converts Celsius to Fahrenheit.
* To verify if the given temperature is within a reasonable and safe range.
* To write test cases using JUnit for different real-world and edge-case temperatures.

**Implementation Breakdown:**

**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);

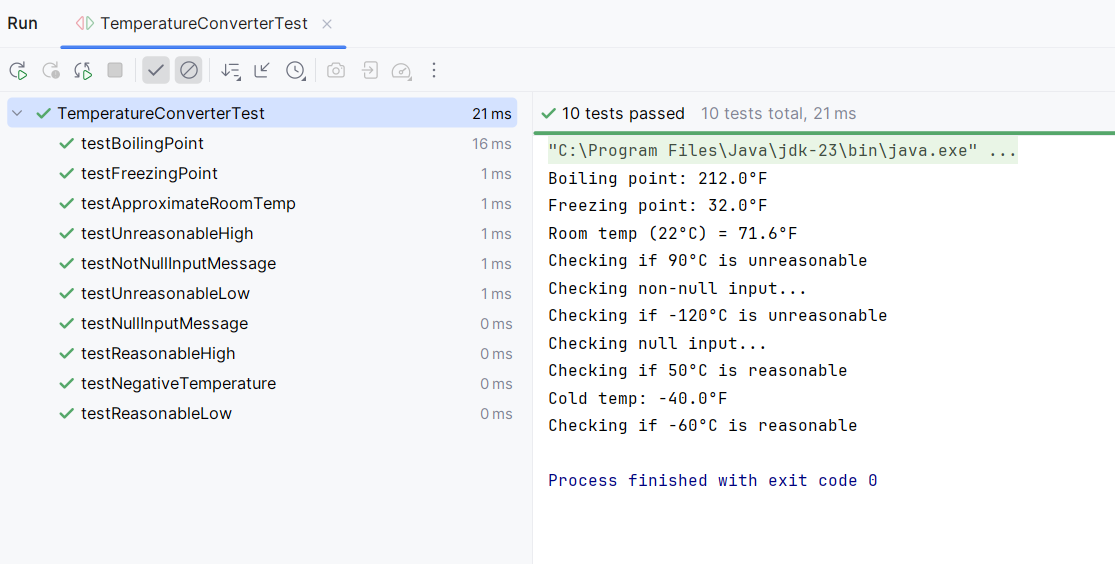
System.out.println("Room temp (22°C) = " + result + "°F");

assertEquals(71.6, result, 0.1);

}

}

**Output:**

****

**Conclusion:**

The program provides reliable and accurate temperature conversion along with reasonability checks. With extensive unit testing, it ensures correctness under various input conditions, including edge and null cases.

**EXERCISE 4: ARRANGE-ACT-ASSERT(AAA) PATTERN, TEST FIXTURES, SETUP AND TEARDOWN METHODS IN JUnit**

**Introduction:**

This Java program implements an EmailValidator class to validate general email formats and detect corporate email addresses specific to a domain like @cognizant.com. JUnit test cases are used to ensure correctness under various scenarios.

**Objective:**

* To validate whether a given string is a properly formatted email.
* To identify if the email belongs to a specific corporate domain.
* To ensure robustness of the validation logic using JUnit testing for different types of inputs.

**Implementation:**

**EmailValidator.java:**

public class EmailValidator {

public boolean isValid(String email) {

if (email == null || email.isEmpty()) return false;

return email.matches("^[A-Za-z0-9+\_.-]+@[A-Za-z0-9.-]+$");

}

public boolean isCorporateEmail(String email) {

return isValid(email) && email.endsWith("@cognizant.com");

}

}

**EmailValidatorTest.java:**

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class EmailValidatorTest {

private EmailValidator validator;

@Before

public void setUp() {

validator = new EmailValidator();

System.out.println("Validator setup done");

}

@After

public void tearDown() {

System.out.println("Test cleanup done\n");

}

@Test

public void testValidEmail() {

String email = "user@example.com";

boolean result = validator.isValid(email);

assertTrue(result);

System.out.println("Valid email passed");

}

@Test

public void testInvalidEmail() {

String email = "invalid-email";

boolean result = validator.isValid(email);

assertFalse(result);

System.out.println("Invalid email caught");

}

@Test

public void testEmptyEmail() {

String email = "";

boolean result = validator.isValid(email);

assertFalse(result);

System.out.println("Empty email rejected");

}

@Test

public void testNullEmail() {

String email = null;

boolean result = validator.isValid(email);

assertFalse(result);

System.out.println("Null email handled");

}

@Test

public void testCognizantEmail() {

String email = "employee@cognizant.com";

boolean result = validator.isCorporateEmail(email);

assertTrue(result);

System.out.println("Cognizant email detected");

}

@Test

public void testNonCorporateEmail() {

String email = "john.doe@gmail.com";

boolean result = validator.isCorporateEmail(email);

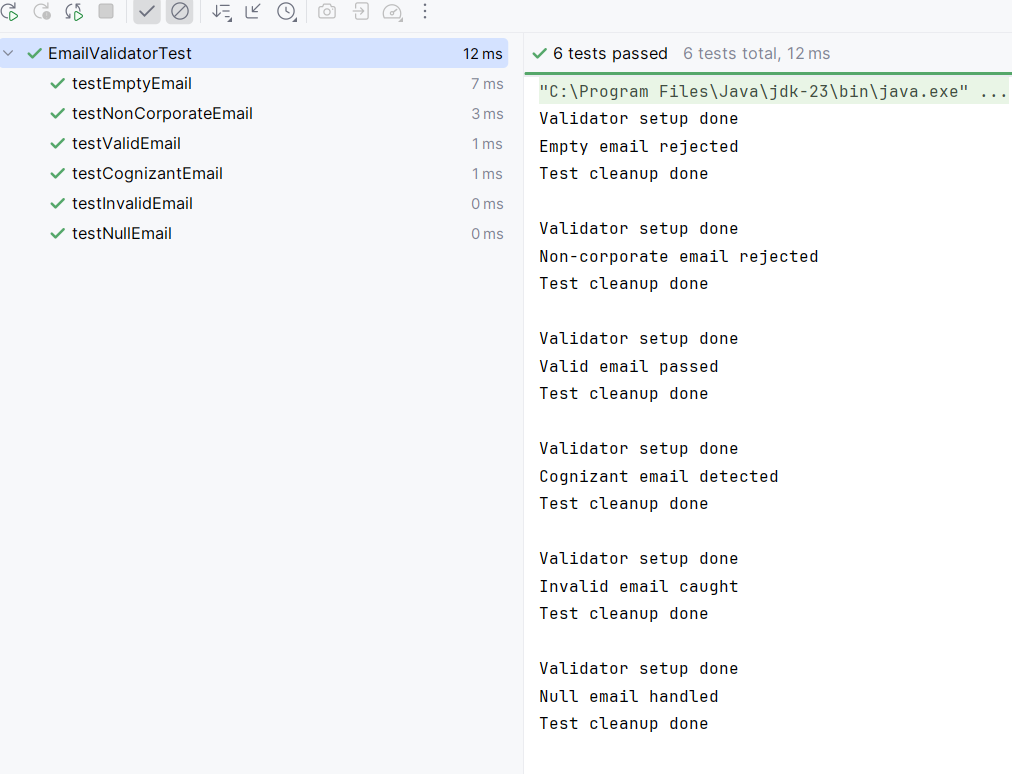
assertFalse(result);

System.out.println("Non-corporate email rejected");

}

}

**Output:**

****

**Conclusion:**

The EmailValidator program effectively validates both general and corporate-specific emails. With well-structured JUnit tests, it handles valid, invalid, null, and empty inputs gracefully, ensuring reliability for real-world applications.