**Student Name:** Jayaanth S R

**SuperSet Id :** 6420871

**EXERCISE 1: Setting Up JUnit**

**Introduction:**

In this java program we are going to built a simple password checker using Java. The goal is to ensure that user passwords follow basic security rules—like having at least 8 characters, at least one digit, and at least one special character. We used JUnit to write test cases that cover both strong and weak password examples.

**Objective:**

* Checks if a password is long enough and secure.
* Gives feedback on whether the password is strong or weak.
* Tests different edge cases using Junit

**Implementation:**

**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) {

return isValid(password) ? "Strong password: " + password

: "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() {

assertFalse(validator.isValid("hi@12"));

}

@Test public void testNoSpecialChar() {

assertFalse(validator.isValid("hello1234"));

}

@Test public void testNoDigit() {

assertFalse(validator.isValid("hello@you"));

}

@Test public void testOnlyLetters() {

assertFalse(validator.isValid("abcdefgh"));

}

@Test public void testOnlyDigits() {

assertFalse(validator.isValid("12345678"));

}

@Test public void testOnlySpecialChars() {

assertFalse(validator.isValid("@#$%^&\*!"));

}

@Test public void testNullPassword() {

assertFalse(validator.isValid(null));

}

@Test public void testEmptyPassword() {

assertFalse(validator.isValid(""));

}

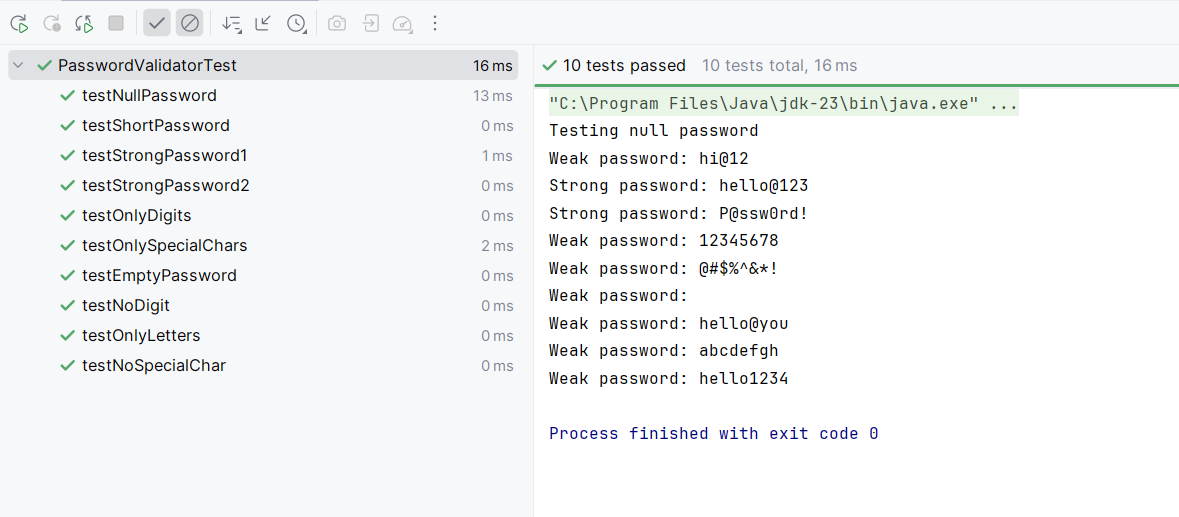
@Test public void testStrongPassword2() {

assertTrue(validator.isValid("P@ssw0rd!"));

}

}

**Output:**

****

**Conclusion:**

The Passwordvalidator works well across different inputs and returns expected results. Our JUnit test suite checks all the important use cases including short, weak, and null passwords.

**EXERCISE 3: ASSERTIONS IN JUnit**

**Introduction:**

This Java utility converts Celsius to Fahrenheit and checks if the given temperature is within the safe range for humans. Using JUnit, we test the conversion logic and the safety check

.**Objective:**

* Covers both typical and edge-case temperatures.
* Converts temperatures from °C to °F.
* Validates if the temperature is within survivable limits.

**Implementation :**

**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() {

assertEquals(32.0, converter.toFahrenheit(0), 0.01);

}

@Test public void testBoilingPoint() {

assertEquals(212.0, converter.toFahrenheit(100), 0.01);

}

@Test public void testNegativeTemperature() {

assertEquals(-40.0, converter.toFahrenheit(-40), 0.01);

}

@Test public void testReasonableLow() {

assertTrue(converter.isReasonableTemperature(-60));

}

@Test public void testReasonableHigh() {

assertTrue(converter.isReasonableTemperature(50));

}

@Test public void testUnreasonableLow() {

assertFalse(converter.isReasonableTemperature(-120));

}

@Test public void testUnreasonableHigh() {

assertFalse(converter.isReasonableTemperature(90));

}

@Test public void testNullInputMessage() {

String input = null;

assertNull(input);

}

@Test public void testNotNullInputMessage() {

String input = "25";

assertNotNull(input);

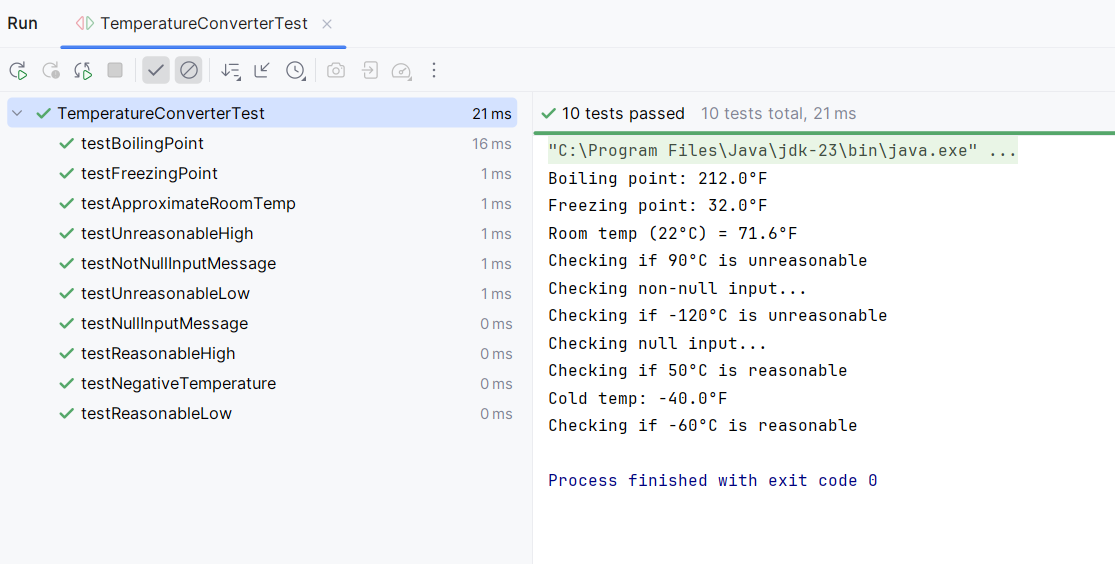
}

@Test public void testApproximateRoomTemp() {

assertEquals(71.6, converter.toFahrenheit(22), 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 utility checks if an email is valid and whether it belongs to a corporate domain like @cognizant.com. JUnit tests are written using **setup and teardown methods** to ensure proper resource handling and testing flow.

**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);

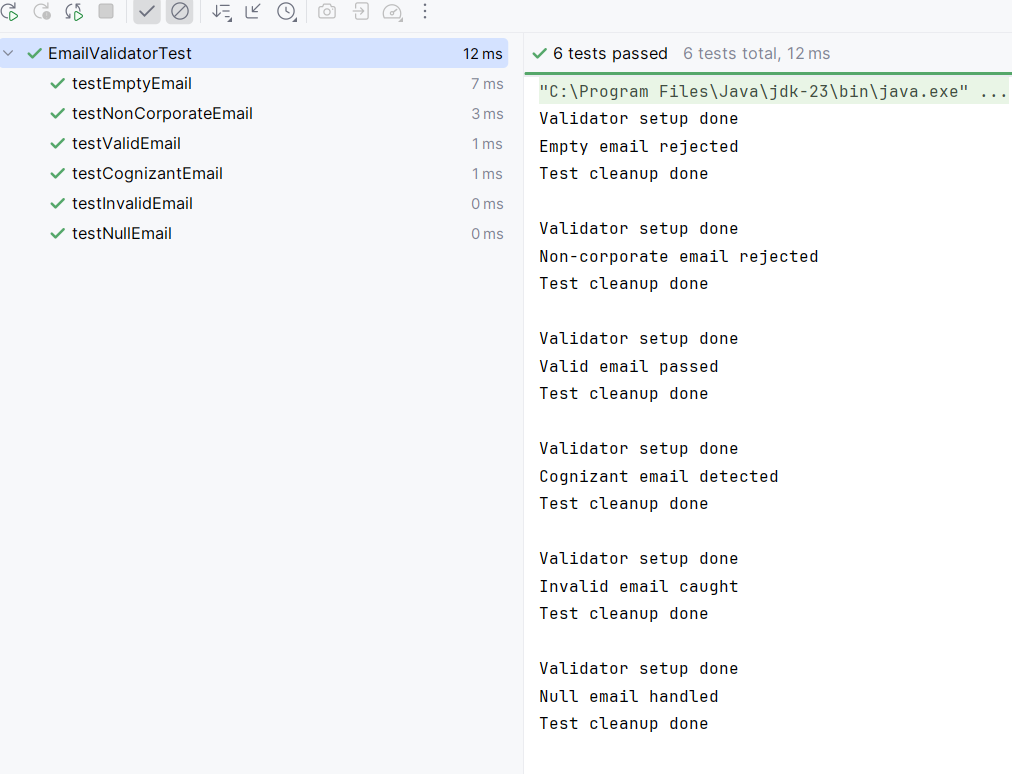
assertFalse(result);

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.