**Day1 Labs: JUnit assertTrue()**

# Project setup (Maven)

Add JUnit 5 to your pom.xml (test scope):

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version> <!-- use latest if you want -->

<scope>test</scope>

</dependency>

**Exercise 1 — Email validator**

Create a small validator in src/main/java:

// EmailValidator.java

import java.util.regex.Pattern;

public class EmailValidator {

private static final Pattern EMAIL = Pattern.compile("^[A-Za-z0-9+\_.-]+@[A-Za-z0-9.-]+$");

public static boolean isValid(String email) {

if (email == null) return false;

return EMAIL.matcher(email).matches();

}

}

**Test it with JUnit (src/test/java):**

// EmailValidatorTest.java

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

class EmailValidatorTest {

@Test

void validEmailShouldReturnTrue() {

assertTrue(EmailValidator.isValid("user@example.com"));

}

@Test

void multipleValidEmails\_usingParameterized() {

org.junit.jupiter.params.ParameterizedTest.class.getName(); // (for clarity)

}

}

**Exercise 2 — Business rule: adult user**

Imagine UserService.isAdult(user) is used before allowing age-restricted actions:

// User.java

public class User {

private final String name;

private final int age;

public User(String name, int age) { this.name = name; this.age = age; }

public int getAge() { return age; }

}

// UserService.java

public class UserService {

public static boolean isAdult(User user) {

return user != null && user.getAge() >= 18;

}

}

Test:

// UserServiceTest.java

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

class UserServiceTest {

@Test

void userAge25\_isAdult() {

User u = new User("Ravi", 25);

assertTrue(UserService.isAdult(u), "25-year-old should be considered an adult");

}

@Test

void userAge18\_boundary\_isAdult() {

User u = new User("Kiran", 18);

assertTrue(UserService.isAdult(u)); // no custom message

}

}

**Show a failing assertion (learning moment)**

If you assert the wrong expectation, assertTrue will fail and print the message:

@Test

void invalidEmail\_shouldFail\_demo() {

// This will fail and show the custom message

assertTrue(EmailValidator.isValid("bad\_at\_example.com"),

"Expected valid email but input was malformed");

}

Running this test produces a failure showing the message — useful for debugging.

**Run & verify**

Run test use your IDE. A passing run shows tests executed with 0 failures. When an assertTrue fails, JUnit prints which test failed and the failure message you provided.