**HƯỚNG DẪN LẬP TRÌNH JUNIT 5**

**By traltb@fe.edu.vn**

Bước 1: Cài đặt jdk, chọn phiên bản jdk 17.

Download ở link sau: <https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

Nên chọn jdk 17 vì:

**JDK 17** là phiên bản LTS (Long-Term Support) phổ biến, tương thích tốt với:

* junit-jupiter (JUnit 5.x)
* maven-surefire-plugin ≥ 2.22.2+
* jacoco-maven-plugin ≥ 0.8.5+

 Sau khi tải và cài đặt JDK 17, hãy đảm bảo rằng biến môi trường JAVA\_HOME được thiết lập đúng và PATH đã bao gồm đường dẫn đến thư mục bin của JDK.

 Kiểm tra phiên bản Java đã cài đặt bằng cách mở terminal hoặc command prompt và chạy lệnh:

java -version

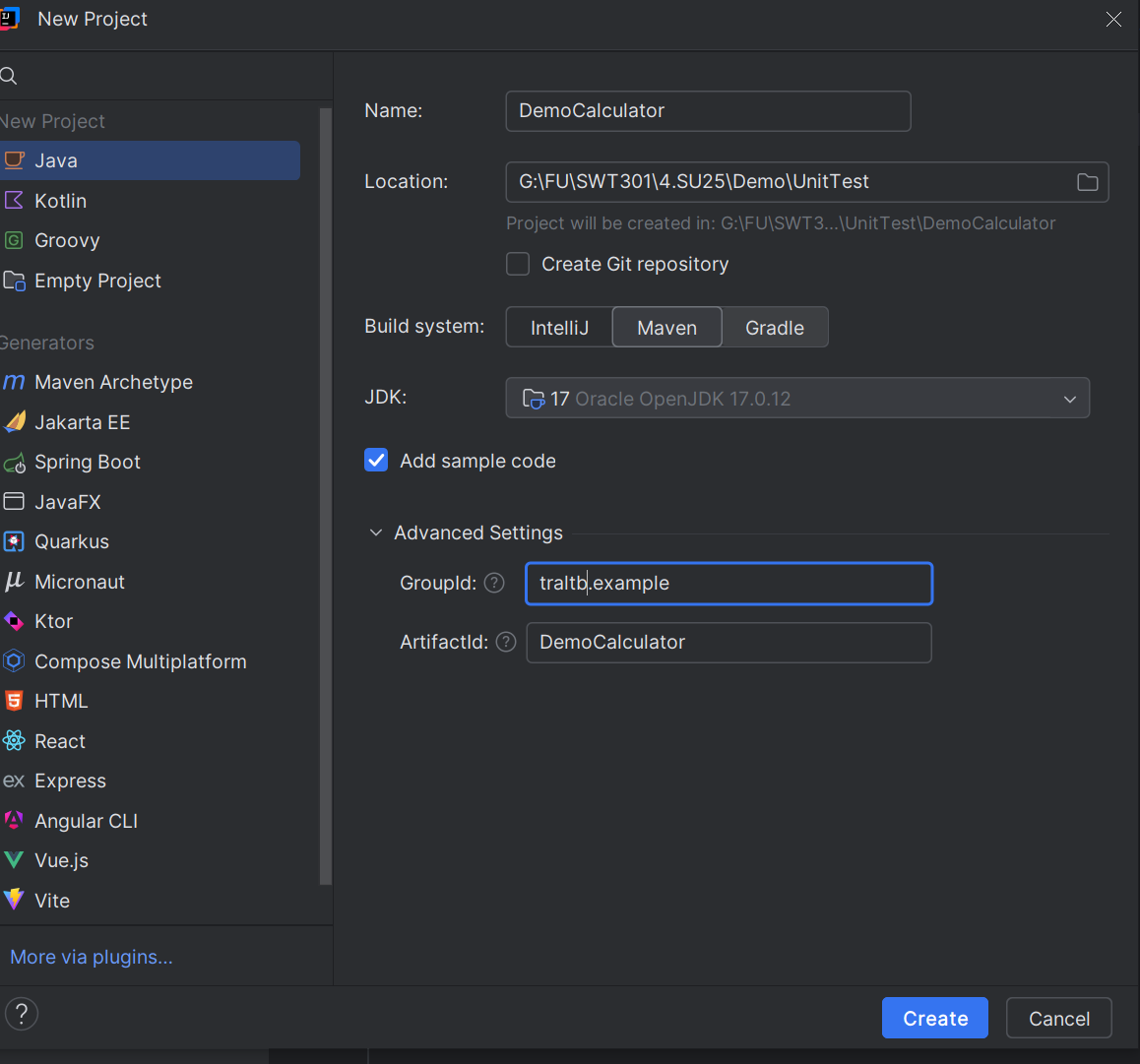
Kết quả sẽ hiển thị phiên bản JDK hiện tại.

Bước 2: Cài đặt Intellij (nên cài IntelliJ IDEA 2022+)

Bước 3: Tạo mới ứng dụng junit

Vào New Project:

* Name: đặt tên Project: DemoCalculator
* Location: chọn đường dẫn lưu dự án
* Build system: chọn Maven
* JDK: chọn jdk 17
* Phần GroupID: ghi name của mình



Bước 4: Cấu hình file pom

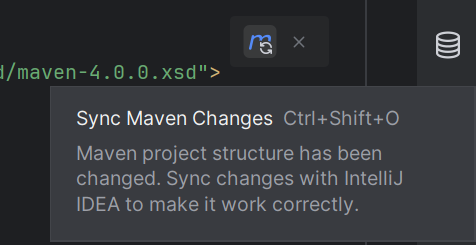
Xem ở link dưới đây, để chọn phiên bản phù hợp

<https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine>

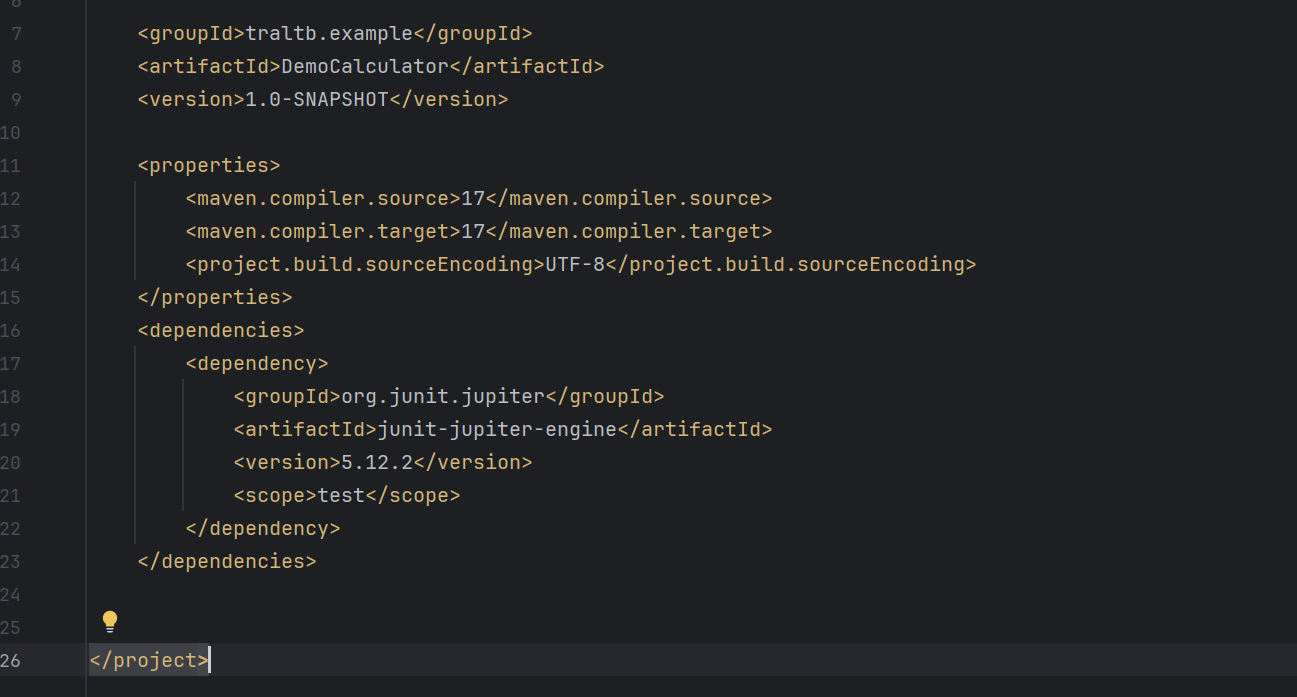
Thêm thẻ dependencies sau vào file pom, tùy thuộc vào phiên bản junit Jupiter engine, cô chọn phiên bản 5.12.2



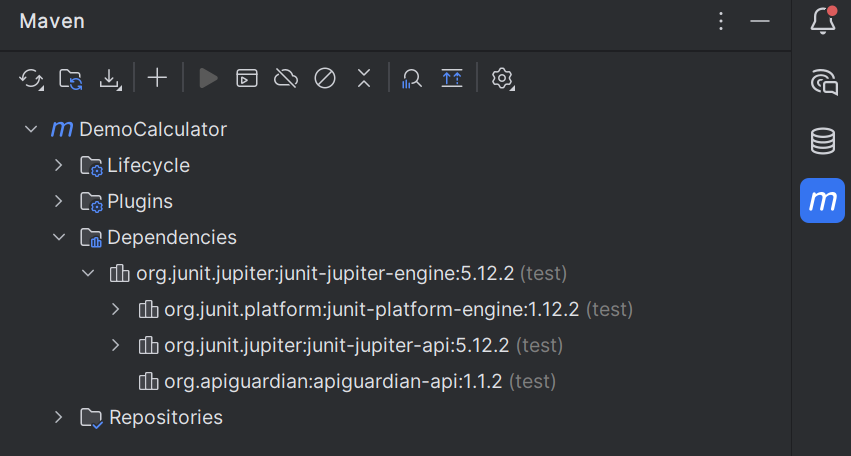
Sau đó, reload maven bằng cách nhấn vào biểu tượng Sync maven changes.



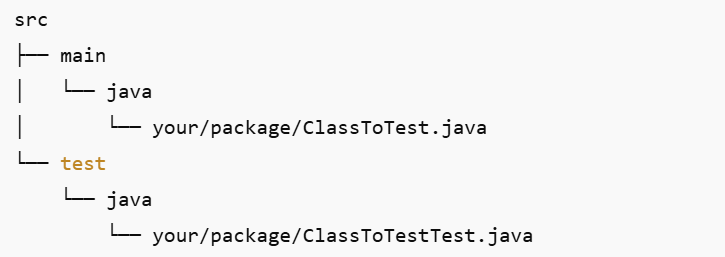
Khi đó, hết lỗi xảy ra:



Đã tải các dependencies phù hợp ở tab maven

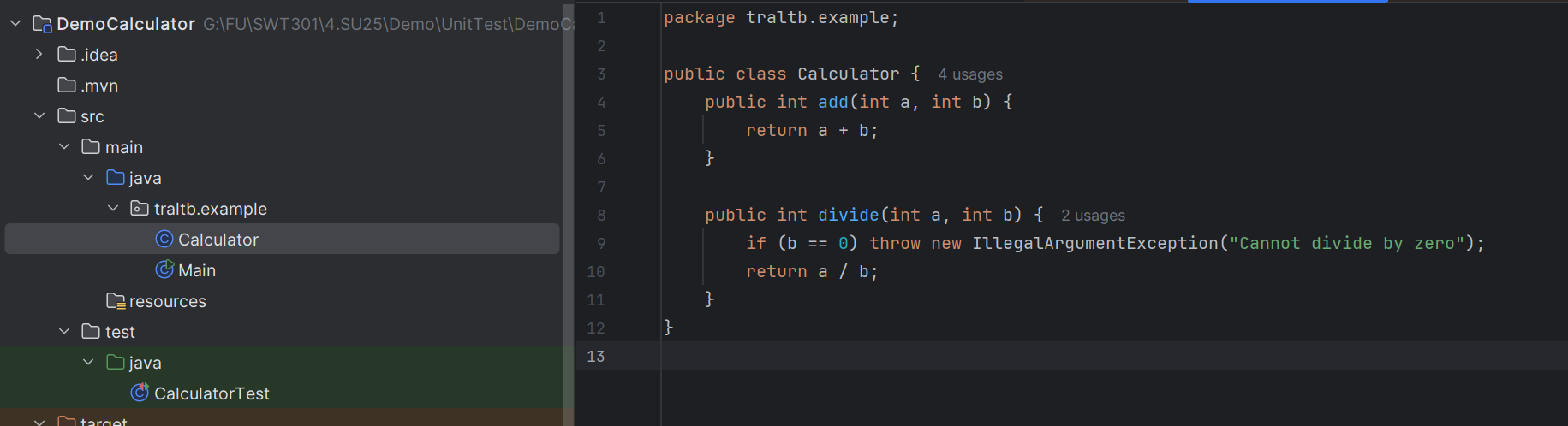


Cấu trúc thư mục chuẩn Maven:



Bước 5: Tạo lớp cần kiểm thử

Tạo lớp Calculator trong main/java/package



package traltb.example;  
  
public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
  
 public int divide(int a, int b) {  
 if (b == 0) throw new IllegalArgumentException("Cannot divide by zero");  
 return a / b;  
 }  
}

Bước 6: Tạo lớp kiểm thử với junit

Tạo lớp CalculatorTest trong test/java/

import org.junit.jupiter.api.Test;  
import traltb.example.Calculator;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.junit.jupiter.api.Assertions.*assertThrows*;  
  
public class CalculatorTest {  
 Calculator calculator = new Calculator();  
 @Test  
 void testAddition() {  
 *assertEquals*(5, calculator.add(2, 3), "Addition should return 5");  
 }  
  
 @Test  
 void testDivide() {  
 *assertEquals*(2, calculator.divide(6, 3));  
 }  
  
 @Test  
 void testDivideByZero() {  
 Exception exception = *assertThrows*(IllegalArgumentException.class, () -> {  
 calculator.divide(10, 0);  
 });  
  
 *assertEquals*("Cannot divide by zero", exception.getMessage());  
 }  
}

Một số điểm lưu ý khi lập trình Junit:

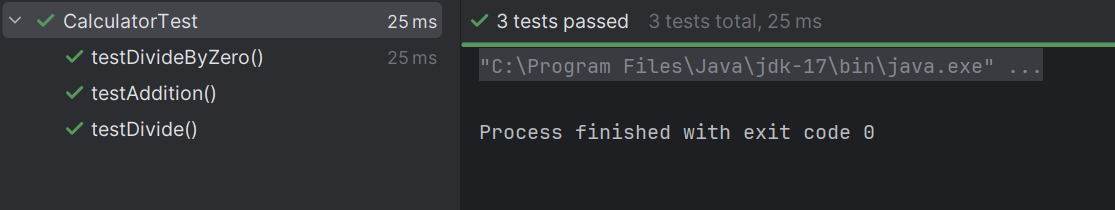
| **Chủ đề** | **Lưu ý** |
| --- | --- |
| **@Test** | Đánh dấu phương thức kiểm thử |
| **assertEquals, assertThrows,...** | Các hàm assertion so sánh kết quả mong đợi và thực tế |
| **Isolation (cô lập)** | Mỗi phương thức test nên kiểm thử một logic duy nhất |
| **Test Naming** | Tên hàm kiểm thử nên mô tả rõ mục đích (shouldReturnX\_whenConditionY) |
| **Setup chung** | Dùng @BeforeEach để khởi tạo nếu có nhiều test dùng chung |
| **Test Coverage** | Đảm bảo kiểm thử tất cả các trường hợp (normal, edge case, error case) |
| **Không test logic trong test** | Tránh viết lại logic cần kiểm thử trong test |

Tìm hiểu kĩ hơn về junit 5 ở link sau: <https://junit.org/junit5/docs/current/user-guide/#overview-what-is-junit-5>

Bước 7: Chạy test trong IntelliJ

* Chuột phải vào class test > **Run ‘CalculatorTest’**
* Hoặc dùng icon ▶️ bên cạnh tên hàm
* Kết quả test hiện trong tab “Run” kèm màu (✅ Pass, ❌ Fail)

Kết quả chạy:



**Ví dụ 2: Sử dụng** @ParameterizedTest với @CsvSource

Bước 1: Thêm dependency sau:

<dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-params</artifactId>  
 <version>5.12.2</version>  
 <scope>test</scope>  
</dependency>

Load lại Maven dependencies

Link chọn phiên bản params phù hợp: <https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-params>

Bước 2: Thêm phương thức kiểm tra nhân 2 số multiply vào lớp Calculator

public int multiply(int a, int b) {  
 return a \* b;  
}

Bước 3: Thêm phương thức kiểm thử phương thức multiply ở trên dùng ParameterizedTest vào lớp

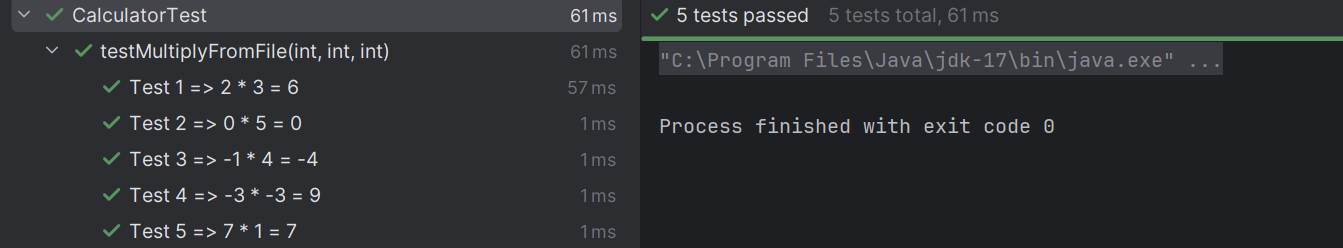
@ParameterizedTest(name = "Test {index} => {0} \* {1} = {2}")  
@CsvFileSource(resources = "/data.csv", numLinesToSkip = 0)  
void testMultiplyFromFile(int a, int b, int expected) {  
 int result = calculator.multiply(a, b);  
 *assertEquals*(expected, result, () -> a + " \* " + b + " should be " + expected);  
}

Bước 4: Tạo file data.csv: Click phải vào thư mục test -> New->Directory, chọn resources. Gõ tên file data.csv.

File data.csv chứa dữ liệu sau:

a,b,expected  
2,3,6  
0,5,0  
-1,4,-4  
-3,-3,9  
7,1,7

Bước 5: Chạy test, click phải run testMultiplyFromFile



**Ví dụ 3**: Sử dụng các annotation @BeforeAll và @AfterAll

Cập nhật lại code sau:

public class CalculatorTest {  
 static Calculator *calculator*;  
  
 @BeforeAll  
 static void initAll() {  
 *calculator* = new Calculator();  
 }  
  
 @AfterAll  
 static void cleanupAll() {  
 *calculator* = null;  
 }  
  
 //Calculator calculator = new Calculator();  
 @DisplayName("Kiểm tra phép cộng với hai số dương")  
 @Test  
 void testAddition() {  
 *assertEquals*(5, *calculator*.add(2, 3), "Addition should return 5");  
 }

}

@BeforeAll và @AfterAll để Khởi tạo tài nguyên dùng chung cho tất cả test

@DisplayName để Thêm mô tả dễ đọc cho từng test case.

@BeforeEach và @AfterEach để Ghi log trước và sau mỗi test để kiểm tra lifecycle.

**BÀI TẬP ÁP DỤNG**

**Dựa vào bài Lab1\_Static testing**

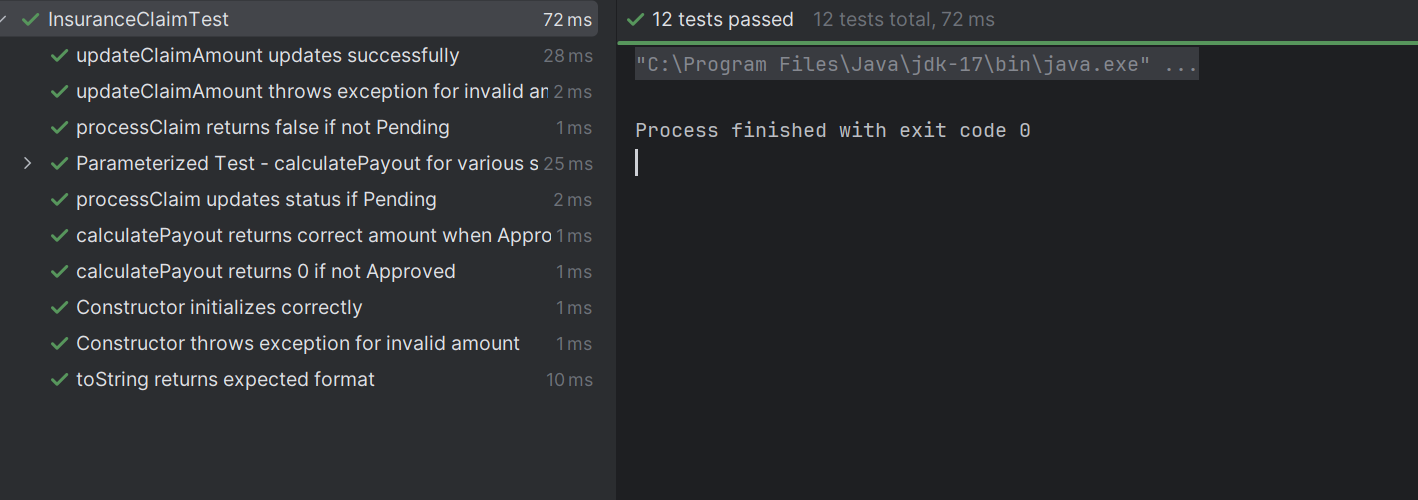
**Bước 1: Thêm lớp InsuranceTest.java**

Copy code đã chỉnh sửa ở phần đáp án vào file này.

**Bước 2: InsuranceClaimTest.java**

import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.DisplayName;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.params.ParameterizedTest;  
import org.junit.jupiter.params.provider.CsvSource;  
import traltb.example.InsuranceClaim;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class InsuranceClaimTest {  
  
 private InsuranceClaim claim;  
  
 @BeforeEach  
 void setUp() {  
 claim = new InsuranceClaim("C001", 1000.0);  
 }  
  
 @Test  
 @DisplayName("Constructor initializes correctly")  
 void testConstructorInitializesValues() {  
 *assertEquals*("C001", claim.getClaimId());  
 *assertEquals*(1000.0, claim.getAmount());  
 *assertEquals*("Pending", claim.getClaimStatus());  
 }  
  
 @Test  
 @DisplayName("Constructor throws exception for invalid amount")  
 void testConstructorInvalidAmount() {  
 *assertThrows*(IllegalArgumentException.class, () -> new InsuranceClaim("C002", -500));  
 }  
  
 @Test  
 @DisplayName("processClaim updates status if Pending")  
 void testProcessClaimWhenPending() {  
 boolean result = claim.processClaim("Approved");  
 *assertTrue*(result);  
 *assertEquals*("Approved", claim.getClaimStatus());  
 }  
  
 @Test  
 @DisplayName("processClaim returns false if not Pending")  
 void testProcessClaimWhenNotPending() {  
 claim.processClaim("Approved");  
 boolean result = claim.processClaim("Rejected");  
 *assertFalse*(result);  
 *assertEquals*("Approved", claim.getClaimStatus());  
 }  
  
 @Test  
 @DisplayName("calculatePayout returns correct amount when Approved")  
 void testCalculatePayoutApproved() {  
 claim.processClaim("Approved");  
 *assertEquals*(850.0, claim.calculatePayout(), 0.001);  
 }  
  
 @Test  
 @DisplayName("calculatePayout returns 0 if not Approved")  
 void testCalculatePayoutNotApproved() {  
 *assertEquals*(0, claim.calculatePayout());  
 }  
  
 @Test  
 @DisplayName("updateClaimAmount updates successfully")  
 void testUpdateClaimAmount() {  
 claim.updateClaimAmount(2000.0);  
 *assertEquals*(2000.0, claim.getAmount());  
 }  
  
 @Test  
 @DisplayName("updateClaimAmount throws exception for invalid amount")  
 void testUpdateClaimAmountInvalid() {  
 *assertThrows*(IllegalArgumentException.class, () -> claim.updateClaimAmount(0));  
 }  
  
 @ParameterizedTest  
 @CsvSource({  
 "Approved,850.0",  
 "Rejected,0",  
 "Pending,0"  
 })  
 @DisplayName("Parameterized Test - calculatePayout for various statuses")  
 void testCalculatePayoutVariousStatuses(String status, double expectedPayout) {  
 claim.processClaim(status);  
 *assertEquals*(expectedPayout, claim.calculatePayout(), 0.001);  
 }  
  
 @Test  
 @DisplayName("toString returns expected format")  
 void testToStringFormat() {  
 String output = claim.toString();  
 *assertTrue*(output.contains("InsuranceClaim"));  
 *assertTrue*(output.contains("claimId='C001'"));  
 *assertTrue*(output.contains("amount=1000.0"));  
 *assertTrue*(output.contains("claimStatus='Pending'"));  
 }  
}

**Bước 3: Chạy kiểm thử**

****

**Yêu cầu:** Viết thêm test cho null inputs nếu logic xử lý null được thêm vào

**Bước 1: Thêm vào l**ớp InsuranceClaim phải có null check logic

Thêm đoạn code để xử lý trường id null:

public InsuranceClaim(String id, double claimAmount) {  
 if (id == null || id.isEmpty()) {  
 throw new IllegalArgumentException("Claim ID cannot be null or empty");  
 }

**Bước 2:**

@Test  
@DisplayName("Constructor throws exception for null claim ID")  
void testConstructorNullClaimId() {  
 *assertThrows*(IllegalArgumentException.class, () -> new InsuranceClaim(null, 1000.0));  
}  
  
@Test  
@DisplayName("processClaim throws exception for null input")  
void testProcessClaimNullInput() {  
 *assertThrows*(IllegalArgumentException.class, () -> claim.processClaim(null));  
}

**Bài tập nộp Lab 2:** Viết unit test cho chức năng Đăng ký tài khoản.

Tạo 1 lớp AccountService có chứa ít nhất 2 phương thức:

boolean registerAccount(username, password, email)

boolean isValidEmail(email)

….

Yêu cầu: Password phải lớn hơn 6 kí tự, email hợp lệ.

**Với dữ liệu đọc từ file test-data.csv như sau:**

username,password,email,expected

john123,pass123,john@example.com,true

,pass123,john@example.com,false

alice,short,alice@mail.com,false

bob123,password,bobmail.com,false

carol,password,carol@domain.com,true

**Yêu cầu:**

1. **Viết các phương thức test để kiểm tra việc đăng ký thành công hay không?**
2. **Cập nhật kết quả vào file UnitTest**

**Bài làm LAB2:**

* File **AccountService.java**

A screenshot of a computer

AI-generated content may be incorrect.

* File **test-data.csv**

A screen shot of a computer

AI-generated content may be incorrect.

* File **AccountServeTest.java**

A screenshot of a computer

AI-generated content may be incorrect.

* File **pom.xml**

A screen shot of a computer program

AI-generated content may be incorrect.

* **Kết quả test**

**A screenshot of a computer

AI-generated content may be incorrect.**