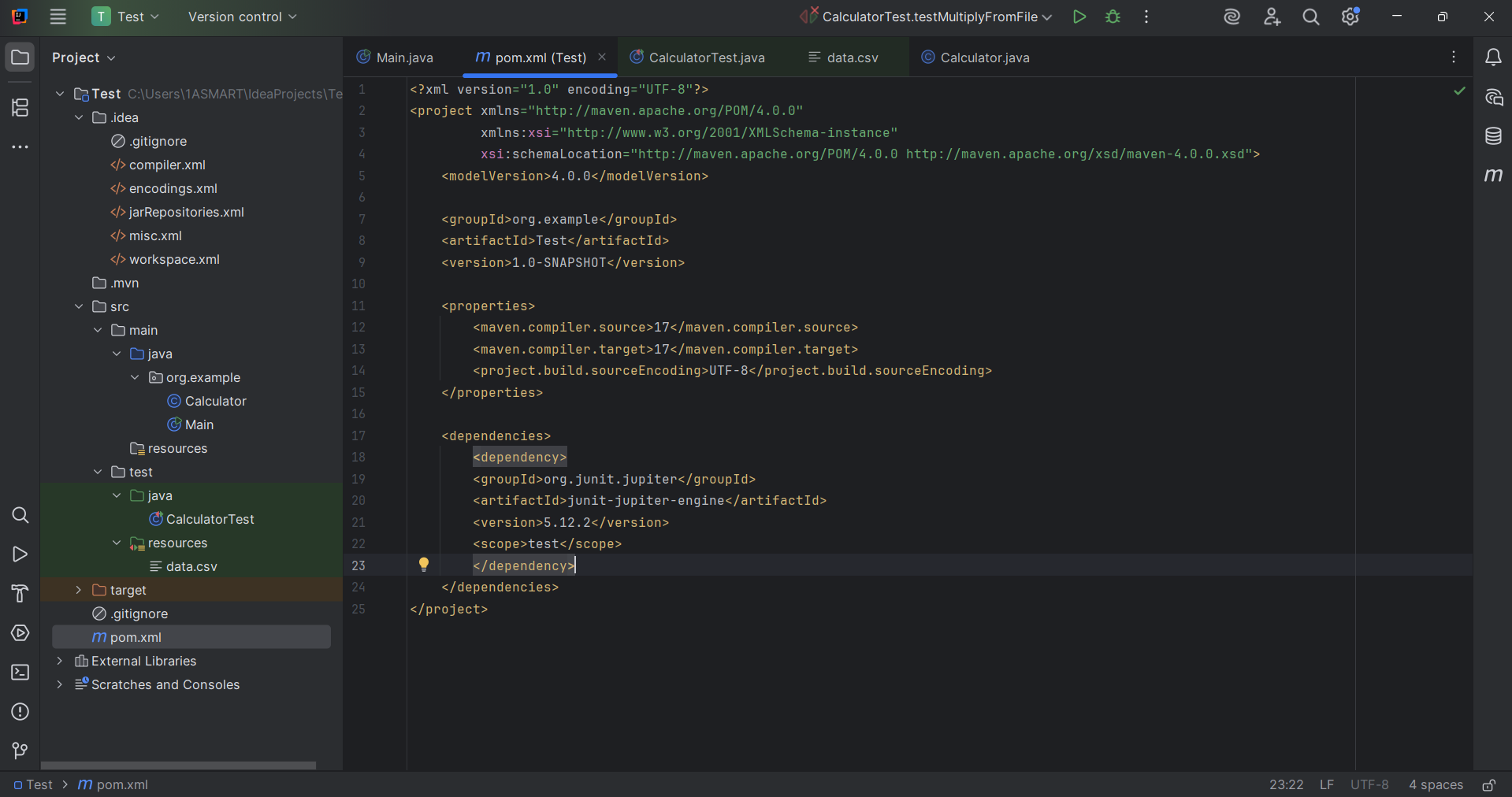
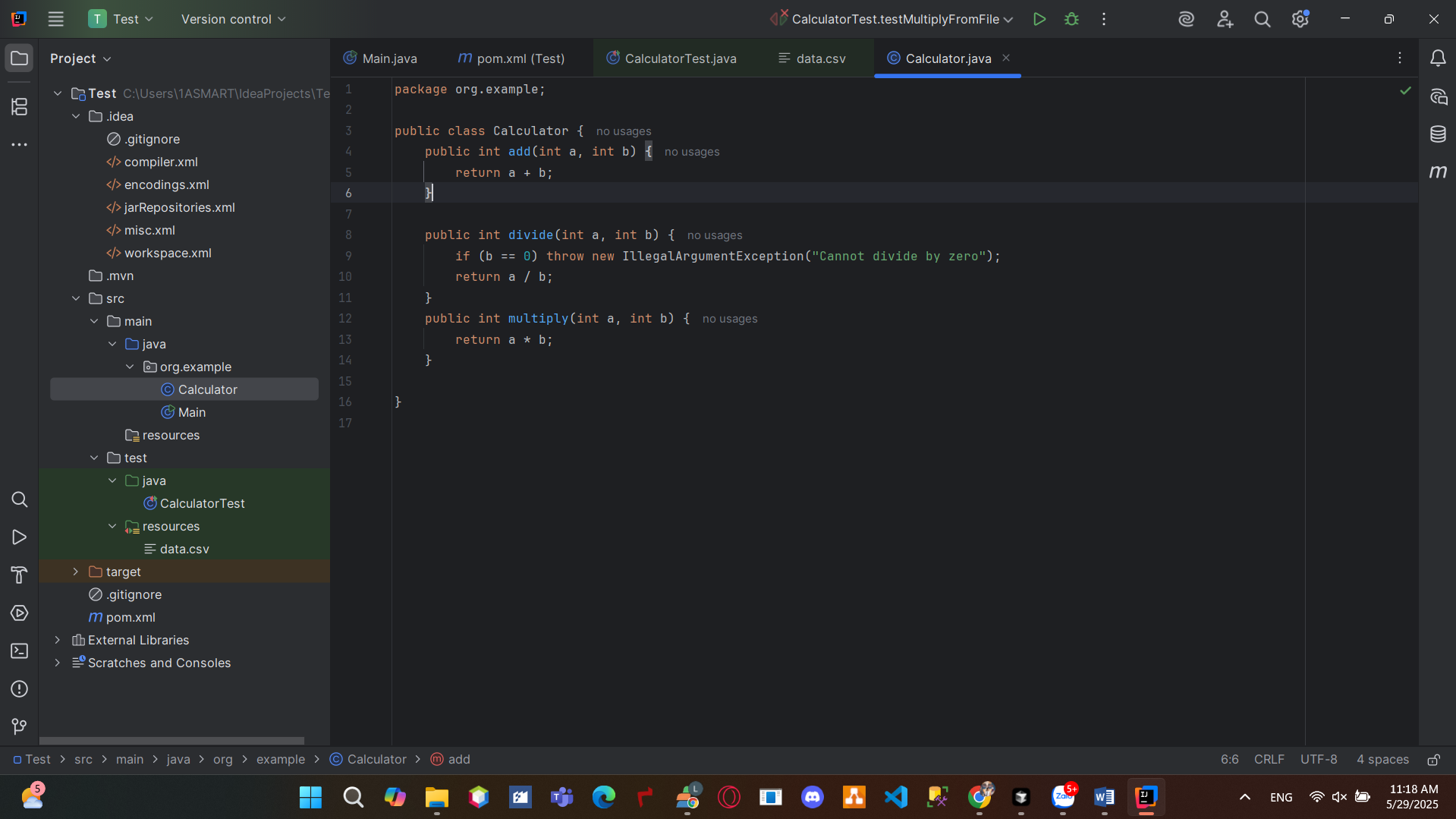
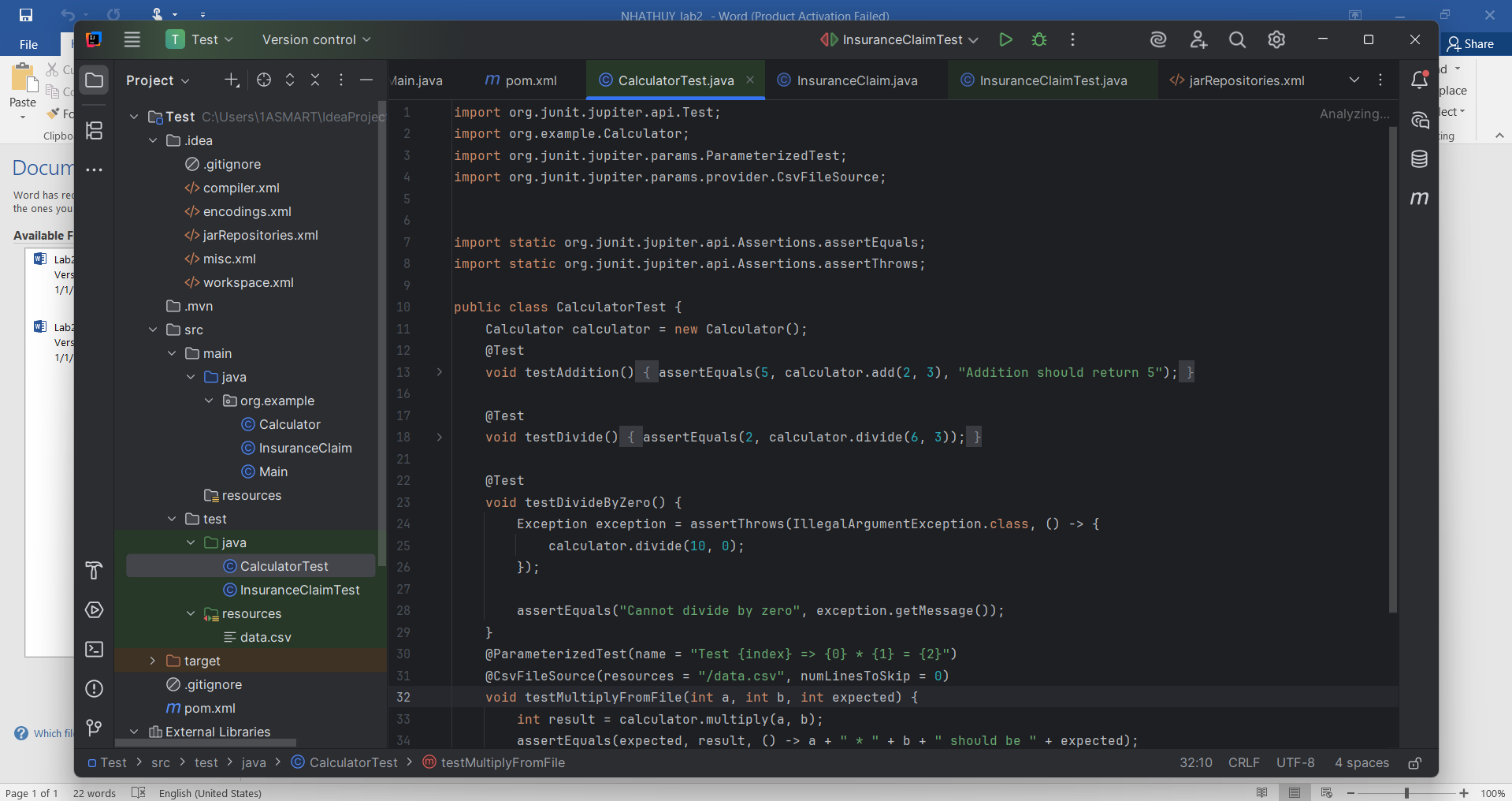
Lê Trần Nhật Huy – DE180335  
1. Cấu hình file pom.xml



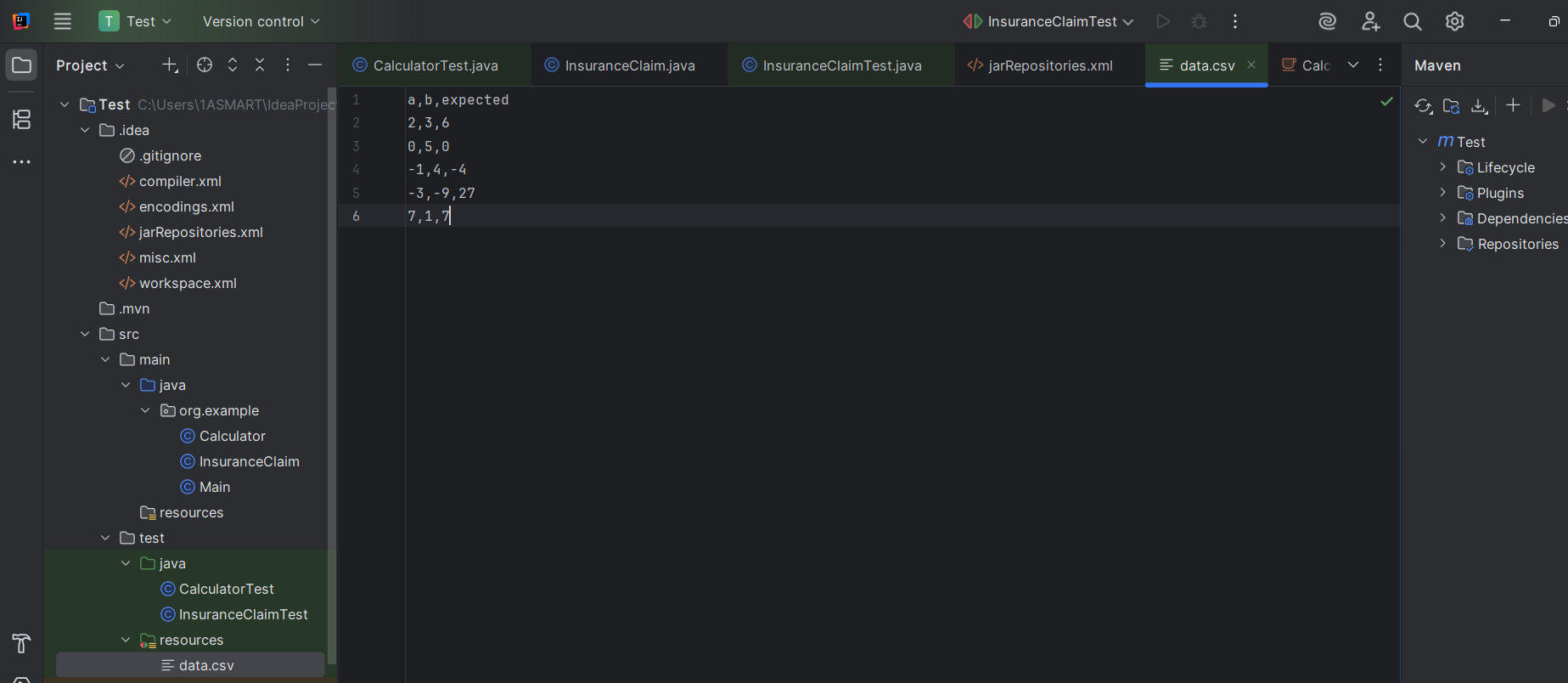
2. Tạo lớp Calculator trong main/java/package



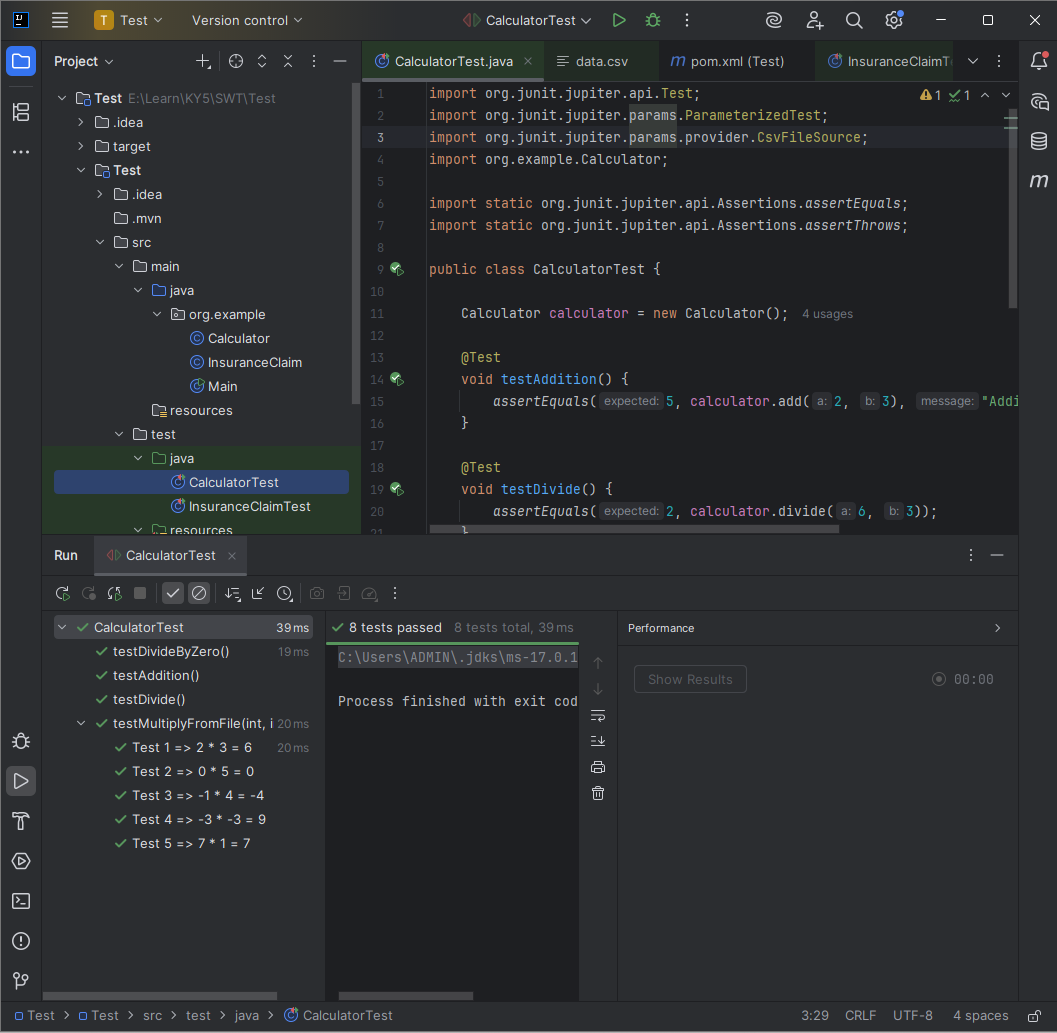
3. Tạo lớp CalculatorTest trong test/java/



4.Data.csv



5.Kết quả:



VD2: InsuranceClaim

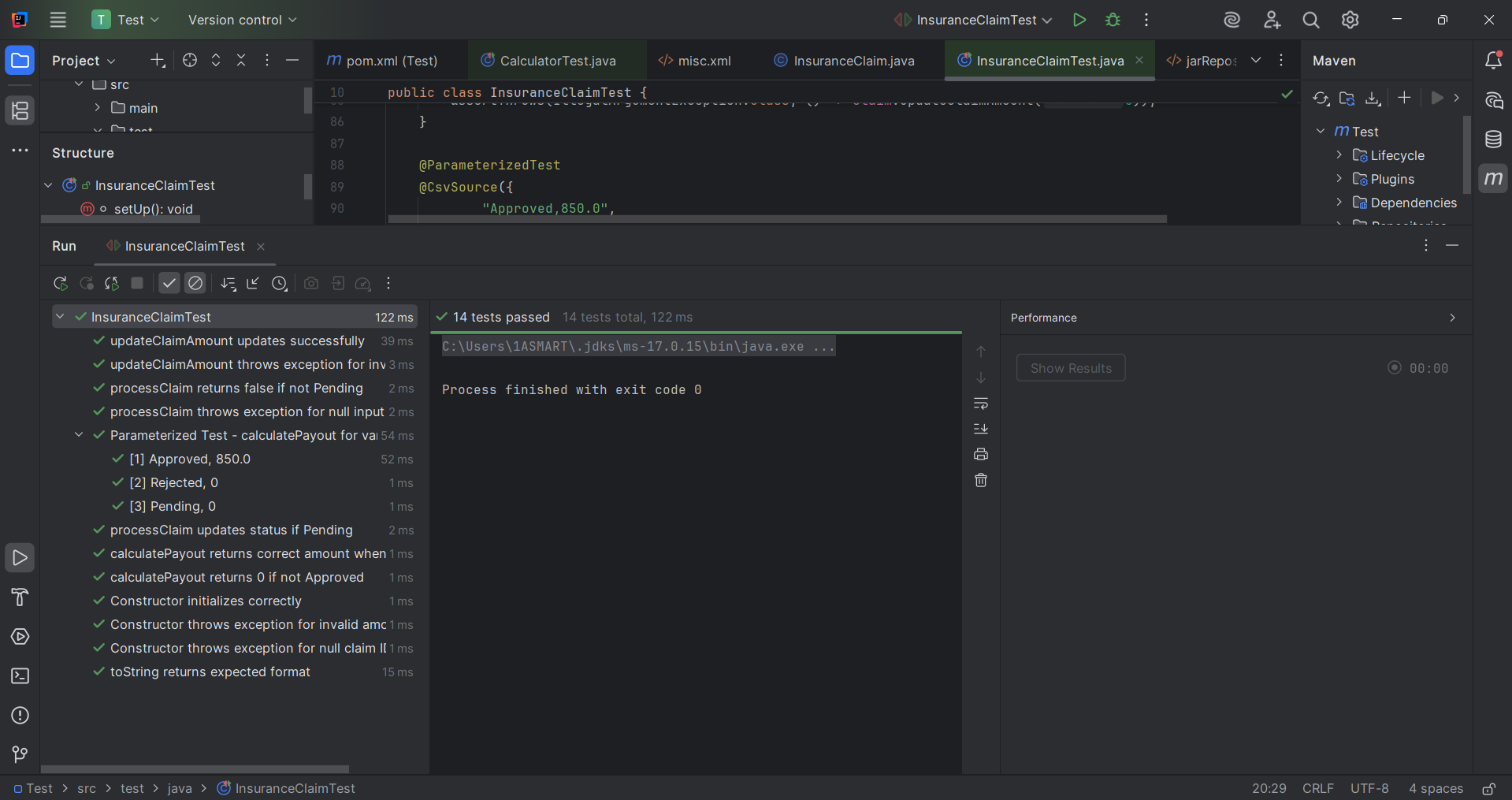
1. Class InsuranceClaim

package org.example;  
  
public class InsuranceClaim {  
  
 private String claimId;  
 private double amount;  
 private String claimStatus;  
  
 public InsuranceClaim(String claimId, double amount) {  
 if (claimId == null || claimId.isEmpty()) {  
 throw new IllegalArgumentException("Claim ID cannot be null or empty");  
 }  
 if (amount <= 0) {  
 throw new IllegalArgumentException("Amount must be positive");  
 }  
 this.claimId = claimId;  
 this.amount = amount;  
 this.claimStatus = "Pending";  
 }  
  
 public String getClaimId() {  
 return claimId;  
 }  
  
 public double getAmount() {  
 return amount;  
 }  
  
 public String getClaimStatus() {  
 return claimStatus;  
 }  
  
 public boolean processClaim(String newStatus) {  
 if (newStatus == null) {  
 throw new IllegalArgumentException("Claim status cannot be null");  
 }  
 if (!"Pending".equalsIgnoreCase(this.claimStatus)) {  
 return false;  
 }  
 this.claimStatus = newStatus;  
 return true;  
 }  
  
 public double calculatePayout() {  
 if ("Approved".equalsIgnoreCase(this.claimStatus)) {  
 return amount \* 0.85;  
 }  
 return 0;  
 }  
  
 public void updateClaimAmount(double newAmount) {  
 if (newAmount <= 0) {  
 throw new IllegalArgumentException("Amount must be positive");  
 }  
 this.amount = newAmount;  
 }  
  
 @Override  
 public String toString() {  
 return "InsuranceClaim{" +  
 "claimId='" + claimId + '\'' +  
 ", amount=" + amount +  
 ", claimStatus='" + claimStatus + '\'' +  
 '}';  
 }  
}

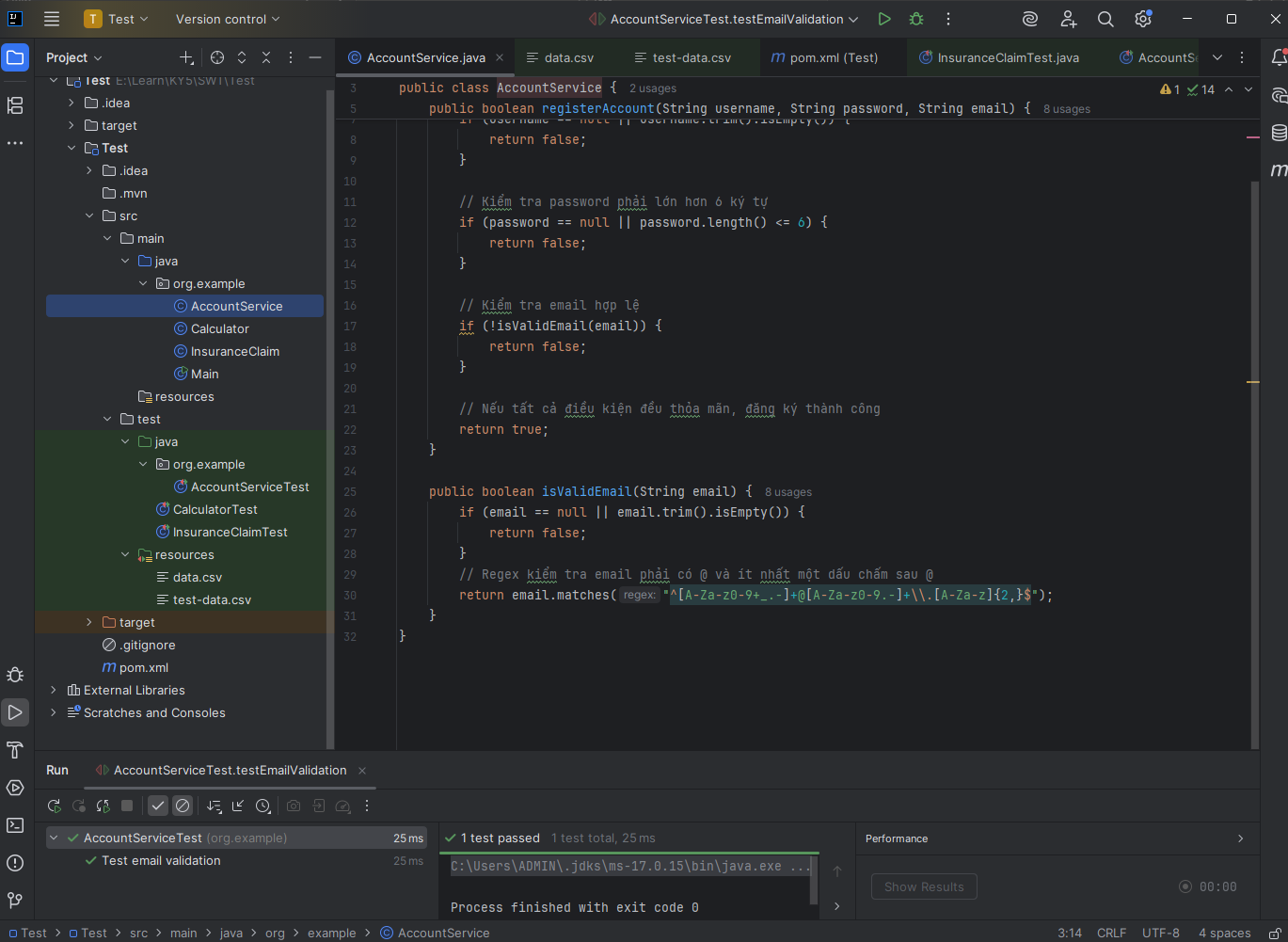
**2.Class InsuranceClaimTest**

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 org.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("Constructor throws exception for null claim ID")  
 void testConstructorNullClaimId() {  
 assertThrows(IllegalArgumentException.class, () -> new InsuranceClaim(null, 1000.0));  
 }  
  
 @Test  
 @DisplayName("processClaim updates status if Pending")  
 void testProcessClaimWhenPending() {  
 boolean result = claim.processClaim("Approved");  
 assertTrue(result);  
 assertEquals("Approved", claim.getClaimStatus());  
 }  
  
 @Test  
 @DisplayName("processClaim throws exception for null input")  
 void testProcessClaimNullInput() {  
 assertThrows(IllegalArgumentException.class, () -> claim.processClaim(null));  
 }  
  
 @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'"));  
 }  
}

**KẾT QUẢ:**

**  
  
Viết unit test cho chức năng Đăng ký tài khoản.**

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

****

**package org.example;**

**public class AccountService {**

**public boolean registerAccount(String *username*, String *password*, String *email*) {**

***// Kiểm tra username không được null hoặc rỗng***

**if (*username* == null || *username*.trim().isEmpty()) {**

**return false;**

**}**

***// Kiểm tra password phải lớn hơn 6 ký tự***

**if (*password* == null || *password*.length() <= 6) {**

**return false;**

**}**

***// Kiểm tra email hợp lệ***

**if (!isValidEmail(*email*)) {**

**return false;**

**}**

***// Nếu tất cả điều kiện đều thỏa mãn, đăng ký thành công***

**return true;**

**}**

**public boolean isValidEmail(String *email*) {**

**if (*email* == null || *email*.trim().isEmpty()) {**

**return false;**

**}**

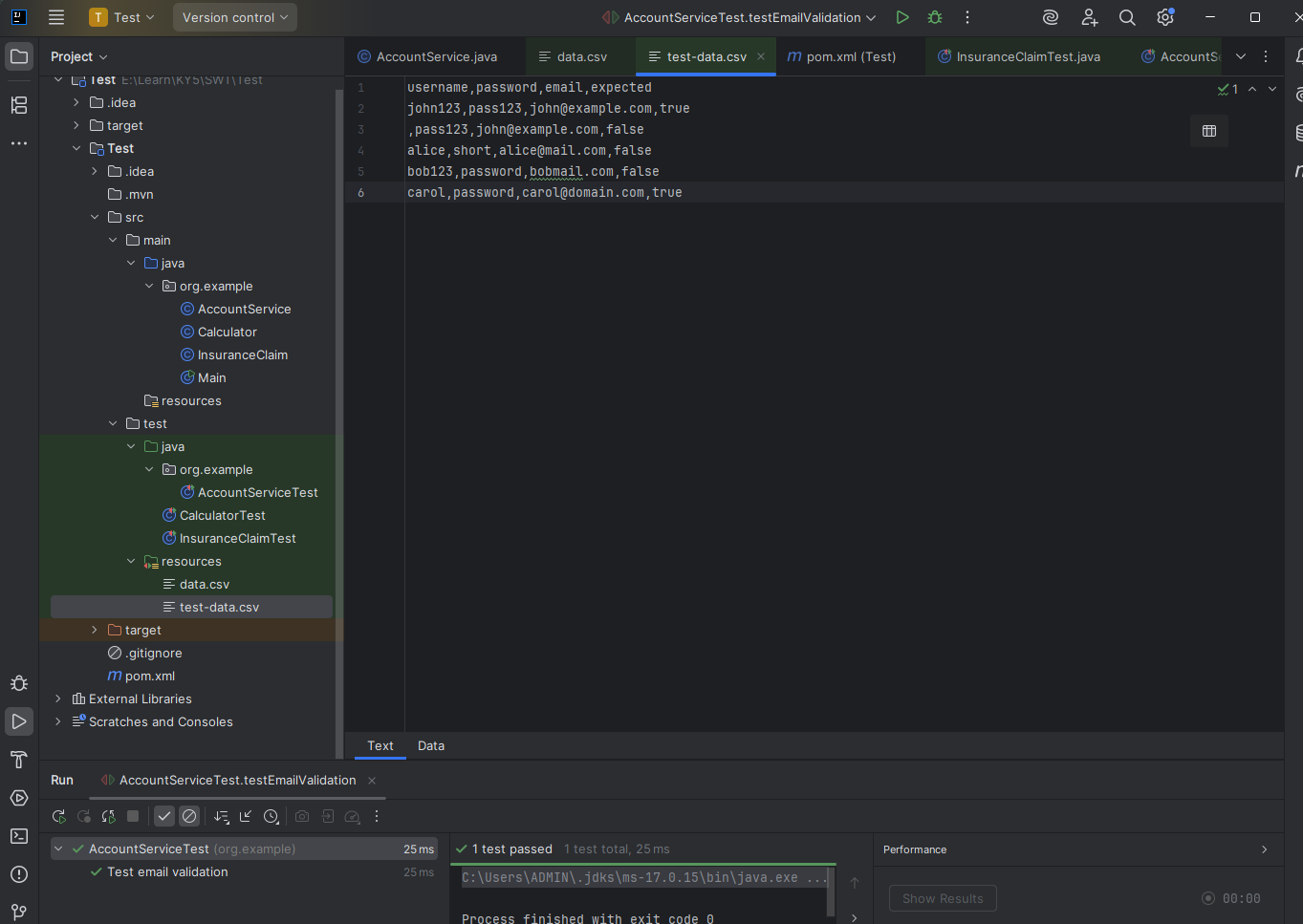
***// Regex kiểm tra email phải có @ và ít nhất một dấu chấm sau @***

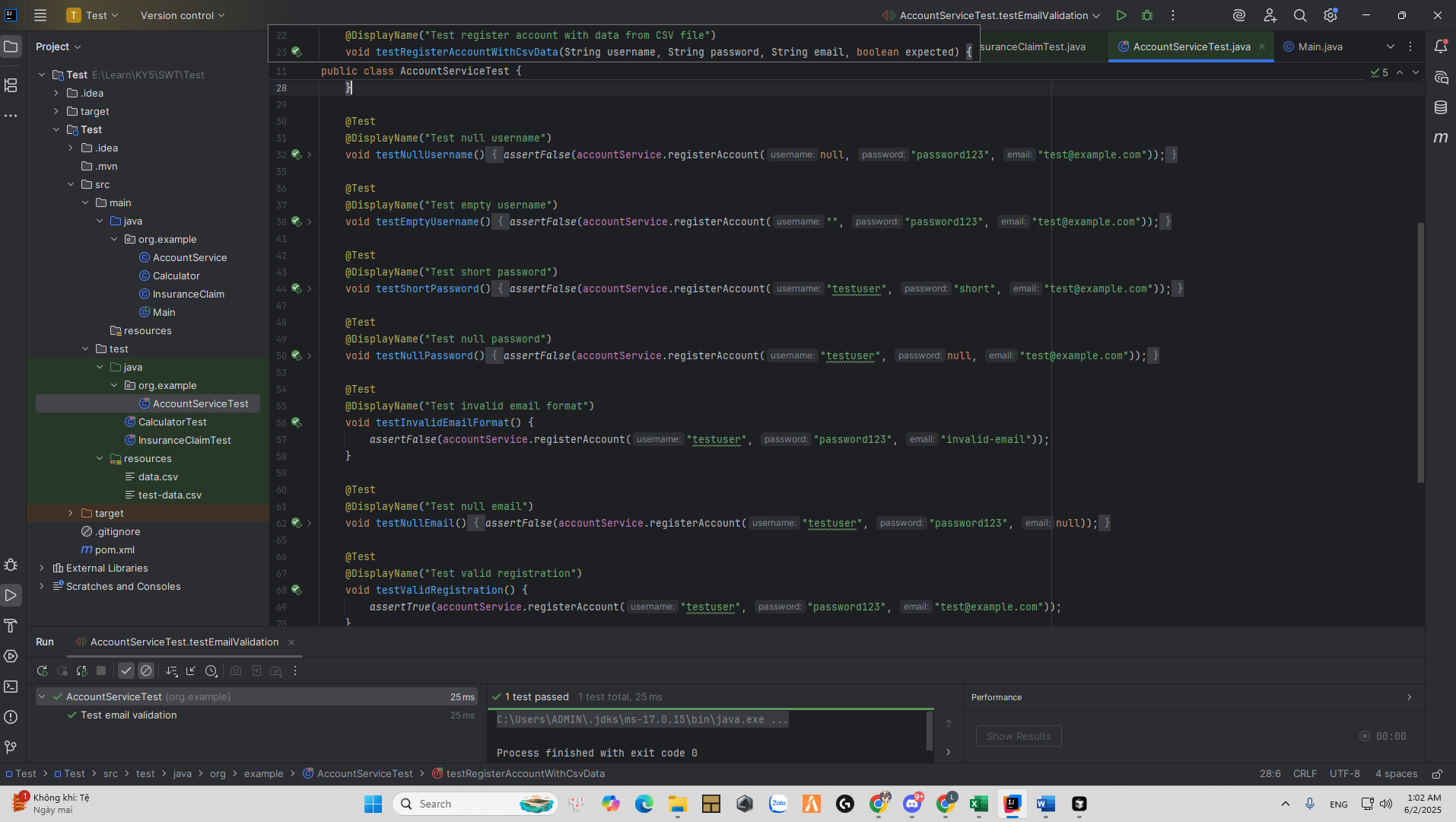
**return *email*.matches("^[A-Za-z0-9+\_.-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$");**

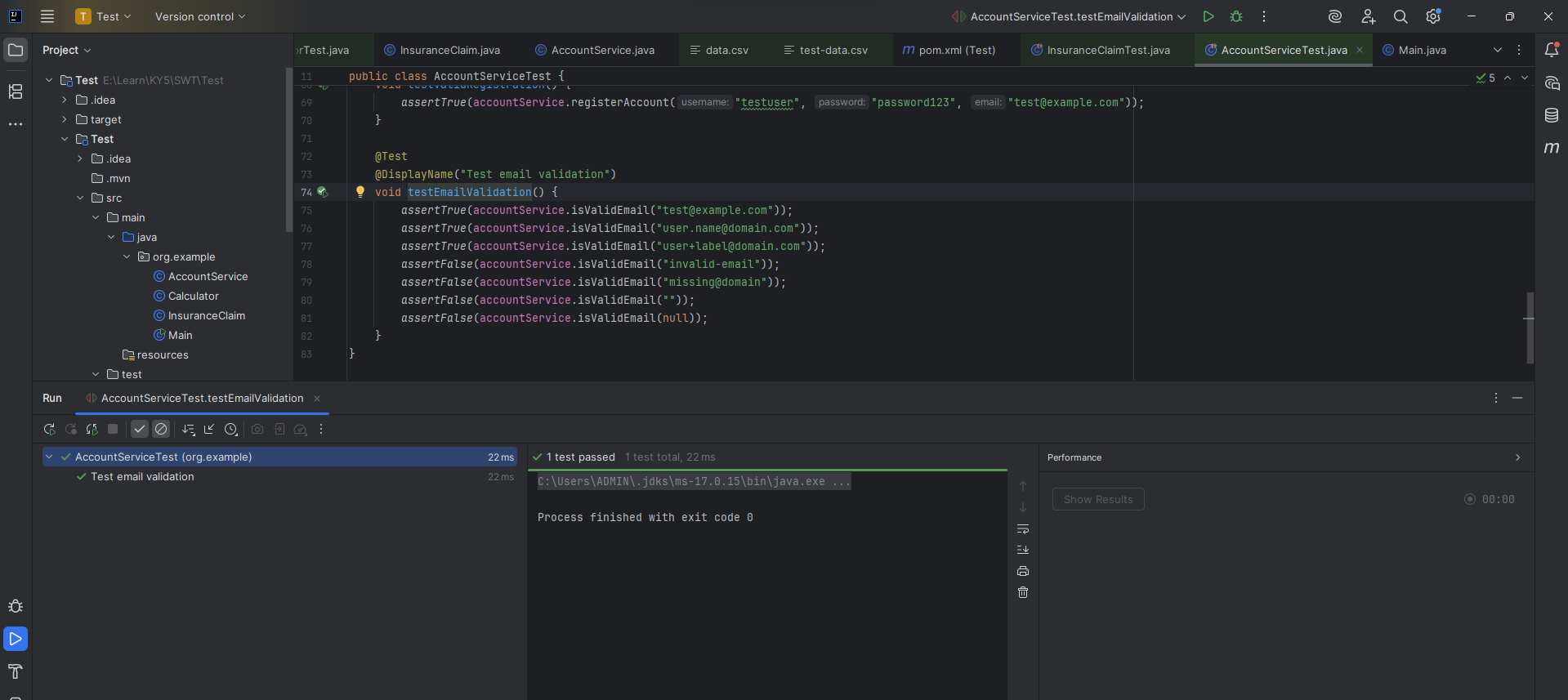
**}**

**}**

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

****

**Test:  
package org.example;  
  
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.CsvFileSource;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class AccountServiceTest {  
   
 private AccountService accountService;  
   
 @BeforeEach  
 void setUp() {  
 accountService = new AccountService();  
 }  
   
 @ParameterizedTest(name = "Test case {index}: username={0}, password={1}, email={2}, expected={3}")  
 @CsvFileSource(resources = "/test-data.csv", numLinesToSkip = 1)  
 @DisplayName("Test register account with data from CSV file")  
 void testRegisterAccountWithCsvData(String username, String password, String email, boolean expected) {  
 boolean result = accountService.registerAccount(username, password, email);  
 *assertEquals*(expected, result,   
 String.*format*("Registration should %s for username=%s, password=%s, email=%s",  
 expected ? "succeed" : "fail", username, password, email));  
 }  
   
 @Test  
 @DisplayName("Test null username")  
 void testNullUsername() {  
 *assertFalse*(accountService.registerAccount(null, "password123", "test@example.com"));  
 }  
   
 @Test  
 @DisplayName("Test empty username")  
 void testEmptyUsername() {  
 *assertFalse*(accountService.registerAccount("", "password123", "test@example.com"));  
 }  
   
 @Test  
 @DisplayName("Test short password")  
 void testShortPassword() {  
 *assertFalse*(accountService.registerAccount("testuser", "short", "test@example.com"));  
 }  
   
 @Test  
 @DisplayName("Test null password")  
 void testNullPassword() {  
 *assertFalse*(accountService.registerAccount("testuser", null, "test@example.com"));  
 }  
   
 @Test  
 @DisplayName("Test invalid email format")  
 void testInvalidEmailFormat() {  
 *assertFalse*(accountService.registerAccount("testuser", "password123", "invalid-email"));  
 }  
   
 @Test  
 @DisplayName("Test null email")  
 void testNullEmail() {  
 *assertFalse*(accountService.registerAccount("testuser", "password123", null));  
 }  
   
 @Test  
 @DisplayName("Test valid registration")  
 void testValidRegistration() {  
 *assertTrue*(accountService.registerAccount("testuser", "password123", "test@example.com"));  
 }  
   
 @Test  
 @DisplayName("Test email validation")  
 void testEmailValidation() {  
 *assertTrue*(accountService.isValidEmail("test@example.com"));  
 *assertTrue*(accountService.isValidEmail("user.name@domain.com"));  
 *assertTrue*(accountService.isValidEmail("user+label@domain.com"));  
 *assertFalse*(accountService.isValidEmail("invalid-email"));  
 *assertFalse*(accountService.isValidEmail("missing@domain"));  
 *assertFalse*(accountService.isValidEmail(""));  
 *assertFalse*(accountService.isValidEmail(null));  
 }  
}   
**

****