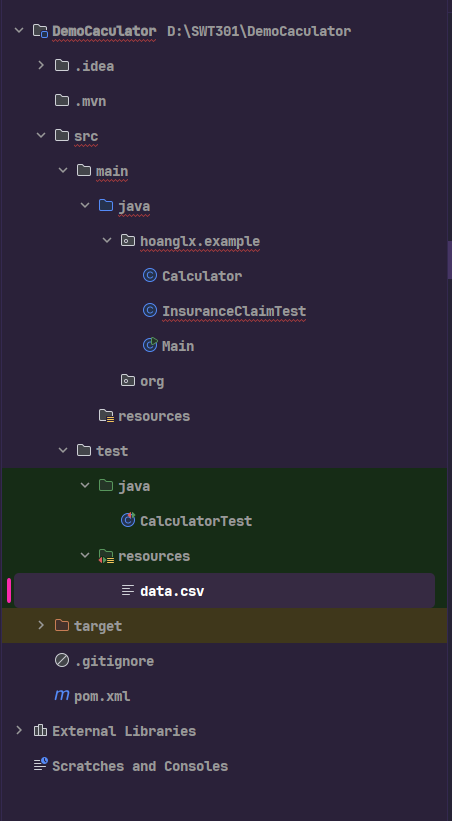
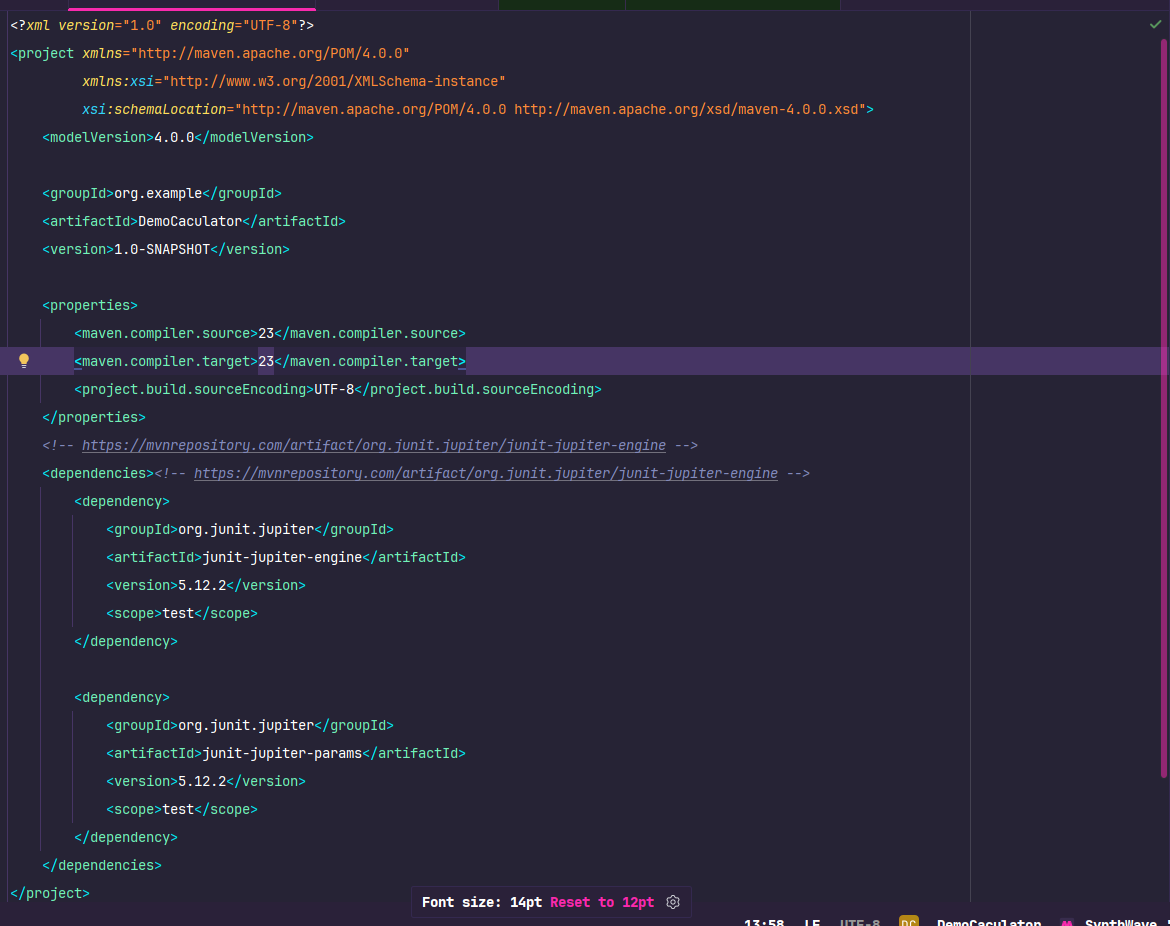
**Phương thức Calculator:**

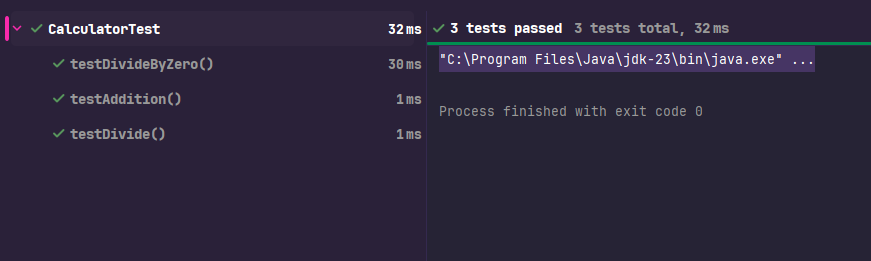
****

Pom.xml

****

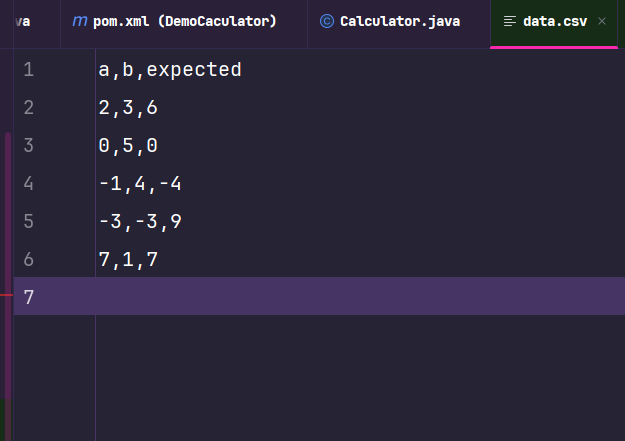
*package* hoanglx.example;  
  
*public class* Calculator {  
  
 *public int* add(*int* a, *int* b) {  
 *return* a + b;  
 }  
  
 *public int* divide(*int* a, *int* b) {  
 *if* (b == 0) *throw new* IllegalArgumentException("Cannot divide by zero");  
 *return* a / b;  
 }  
  
 *public int* multiply(*int*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

*import* org.junit.jupiter.api.Test;  
*import* hoanglx.example.Calculator;  
*import* org.junit.jupiter.params.ParameterizedTest;  
*import* org.junit.jupiter.params.provider.CsvFileSource;  
*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());  
 }  
  
}

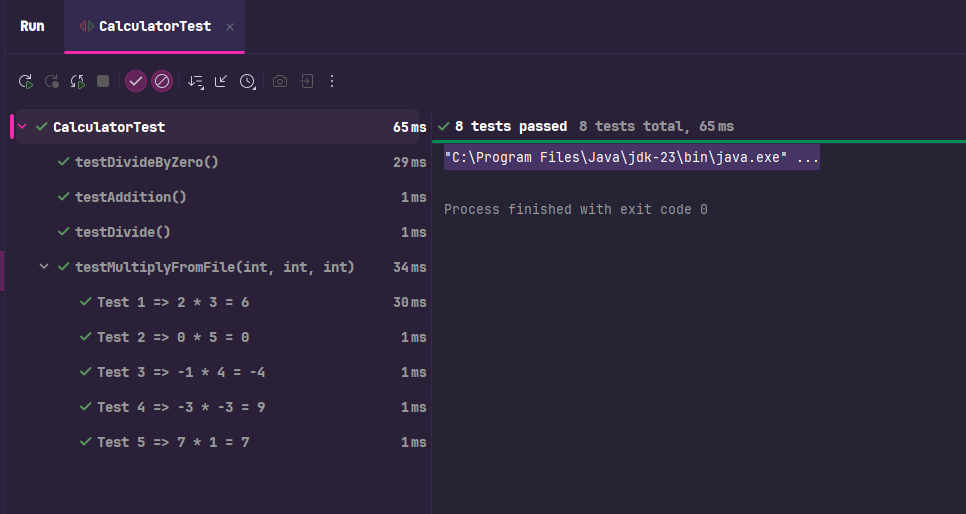
****

**Sử dụng @ParameterizedTest với @CsvSource**

File data.csv

****

*import* org.junit.jupiter.api.Test;  
*import* hoanglx.example.Calculator;  
*import* org.junit.jupiter.params.ParameterizedTest;  
*import* org.junit.jupiter.params.provider.CsvFileSource;  
  
*import static* org.junit.jupiter.api.Assertions.*assertEquals*;  
*import static* org.junit.jupiter.api.Assertions.*assertThrows*;  
  
*public class* CalculatorTest {  
 Calculator calculator = *new* Calculator();  
 @ParameterizedTest(name = "Test {index} => {0} \* {1} = {2}")  
 @CsvFileSource(resources = "/data.csv", numLinesToSkip = 1)  
 *void* testMultiplyFromFile(*int* a, *int* b, *int* expected) {  
 *int* result = calculator.multiply(a, b);  
 *assertEquals*(expected, result, () -> a + " \* " + b + " should be " + expected);  
 }  
  
 @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());  
 }  
  
}

****

**Sử dụng các annotation @BeforeAll và @AfterAll**

*import* org.junit.jupiter.api.AfterAll;  
*import* org.junit.jupiter.api.BeforeAll;  
*import* org.junit.jupiter.api.DisplayName;  
*import* org.junit.jupiter.api.Test;  
*import* hoanglx.example.Calculator;  
*import* org.junit.jupiter.params.ParameterizedTest;  
*import* org.junit.jupiter.params.provider.CsvFileSource;  
  
*import static* org.junit.jupiter.api.Assertions.*assertEquals*;  
*import static* org.junit.jupiter.api.Assertions.*assertThrows*;  
  
*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");  
 }  
}

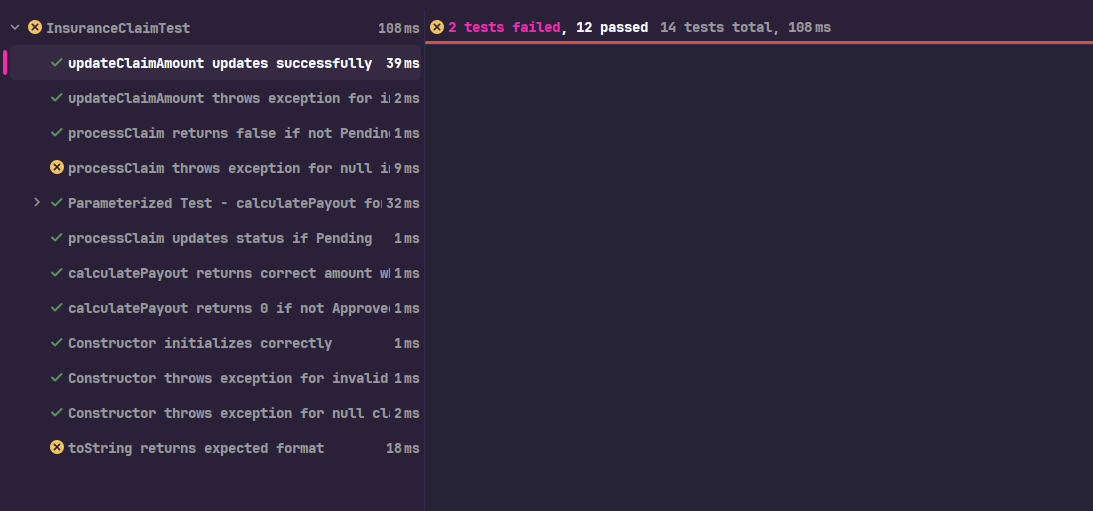
**Phương Thức InsuranceClaim:**

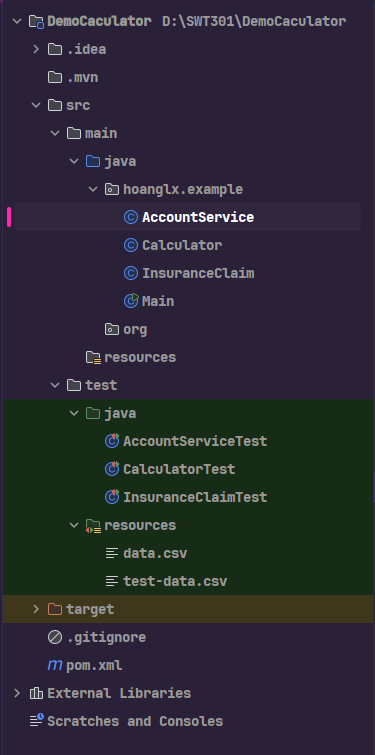
Lớp InsuranceClaim

*package* hoanglx.example;  
  
*public class* InsuranceClaim {  
  
 *private* String claimId;  
 *private double* amount;  
 *private* String claimStatus;  
  
 *private static final double* PAYOUT\_RATE = 0.85;  
  
 */\*\*  
 \* Constructor to initialize the insurance claim.  
 \** ***@param id*** *Claim ID  
 \** ***@param claimAmount*** *Amount claimed  
 \** ***@throws*** *IllegalArgumentException if amount is zero or negative  
 \*/  
 public* InsuranceClaim(String id, *double* claimAmount) {  
 *if* (claimAmount <= 0) {  
 *throw new* IllegalArgumentException("Claim amount must be positive.");  
 }  
 *this*.claimId = id;  
 *this*.amount = claimAmount;  
 *this*.claimStatus = "Pending";  
 *if* (id == *null* || id.isEmpty()) {  
 *throw new* IllegalArgumentException("Claim ID cannot be null or empty");  
 }  
 }  
  
 */\*\*  
 \* Processes the claim by updating its status if currently pending.  
 \** ***@param statusUpdate*** *New status to apply  
 \** ***@return*** *true if update was applied, false otherwise  
 \*/  
 public boolean* processClaim(String statusUpdate) {  
 *if* ("Pending".equals(*this*.claimStatus)) {  
 *this*.claimStatus = statusUpdate;  
 *return true*;  
 }  
 *return false*;  
 }  
  
 */\*\*  
 \* Calculates payout based on approval.  
 \** ***@return*** *payout amount or 0 if not approved  
 \*/  
 public double* calculatePayout() {  
 *if* ("Approved".equals(*this*.claimStatus)) {  
 *return* amount \* PAYOUT\_RATE;  
 } *else* {  
 *return* 0;  
 }  
 }  
  
 */\*\*  
 \* Updates the amount of the claim.  
 \** ***@param newAmount*** *new claim amount  
 \*/  
 public void* updateClaimAmount(*double* newAmount) {  
 *if* (newAmount <= 0) {  
 *throw new* IllegalArgumentException("New amount must be positive.");  
 }  
 *this*.amount = newAmount;  
 }  
  
 *// Getters  
 public* String getClaimId() {  
 *return* claimId;  
 }  
  
 *public double* getAmount() {  
 *return* amount;  
 }  
  
 *public* String getClaimStatus() {  
 *return* claimStatus;  
 }  
  
 @Override  
 *public* String toString() {  
 *return* "InsuranceClaim{" +  
 "claimId='" + claimId + '\'' +  
 ", amount=" + amount +  
 ", claimStatus='" + claimStatus + '\'' +  
 '}';  
 }  
  
}

lớp 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* hoanglx.example.InsuranceClaim;  
  
*import static* org.junit.jupiter.api.Assertions.\*;  
  
*class* InsuranceClaimTest {  
  
 *private* InsuranceClaim claim;  
 @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*));  
 }  
  
 @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("InsuranceTest"));  
 *assertTrue*(output.contains("claimId='C001'"));  
 *assertTrue*(output.contains("amount=1000.0"));  
 *assertTrue*(output.contains("claimStatus='Pending'"));  
 }  
}



Phương thức AccountService test chức năng đăng kí tài khoản

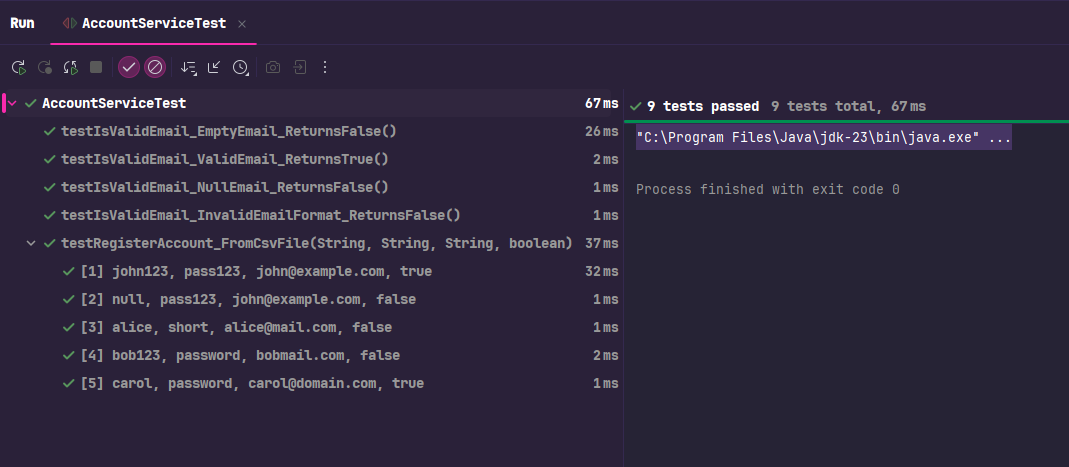
Class AccountService

*package* hoanglx.example;  
  
*public class* AccountService {  
 *public boolean* registerAccount(String username, String password, String email) {  
 *if* (username == *null* || username.isEmpty()) {  
 *return false*;  
 }  
 *if* (password == *null* || password.length() <= 6) {  
 *return false*;  
 }  
 *if* (!isValidEmail(email)) {  
 *return false*;  
 }  
 *return true*;  
 }  
  
 *public boolean* isValidEmail(String email) {  
 *if* (email == *null* || email.isEmpty()) {  
 *return false*;  
 }  
 *// Sửa regex để yêu cầu domain có ít nhất một dấu chấm* String emailRegex = "^[A-Za-z0-9+\_.-]+@[A-Za-z0-9-]+\\.[A-Za-z]{2,}$";  
 *return* email.matches(emailRegex);  
 }  
}

Class AccountServiceTest

*import* hoanglx.example.AccountService;  
*import* org.junit.jupiter.api.BeforeEach;  
*import* org.junit.jupiter.api.Test;  
*import* org.junit.jupiter.params.ParameterizedTest;  
*import* org.junit.jupiter.params.provider.CsvFileSource;  
*import* org.junit.jupiter.params.provider.CsvSource;  
*import static* org.junit.jupiter.api.Assertions.\*;  
  
  
*public class* AccountServiceTest {  
 *private* AccountService accountService;  
  
 @BeforeEach  
 *void* setUp() {  
 accountService = *new* AccountService();  
 }  
  
 *// Test từ dữ liệu CSV* @ParameterizedTest  
 @CsvFileSource(resources = "/test-data.csv", numLinesToSkip = 1) *// Đọc file test-data.csv, bỏ qua dòng tiêu đề  
 void* testRegisterAccount\_FromCsvFile(String username, String password, String email, *boolean* expected) {  
 *// Gọi phương thức registerAccount với các tham số từ CSV  
 boolean* result = accountService.registerAccount(username, password, email);  
 *// So sánh kết quả thực tế với kết quả kỳ vọng từ file CSV  
 // Nếu không khớp, hiển thị thông báo lỗi với thông tin chi tiết  
 assertEquals*(expected, result,  
 String.*format*("Registration failed for username=%s, password=%s, email=%s",  
 username, password, email));  
 }  
  
 *// Test kiểm tra isValidEmail với email hợp lệ* @Test  
 *void* testIsValidEmail\_ValidEmail\_ReturnsTrue() {  
 *// Gọi isValidEmail với email hợp lệ  
 boolean* result = accountService.isValidEmail("test@example.com");  
 *// Kiểm tra kết quả phải là true, nếu không thì hiển thị thông báo lỗi  
 assertTrue*(result, "Valid email should return true");  
 }  
  
 *// Test kiểm tra isValidEmail với email null* @Test  
 *void* testIsValidEmail\_NullEmail\_ReturnsFalse() {  
 *// Gọi isValidEmail với email null  
 boolean* result = accountService.isValidEmail(*null*);  
 *// Kiểm tra kết quả phải là false, nếu không thì hiển thị thông báo lỗi  
 assertFalse*(result, "Null email should return false");  
 }  
  
 *// Test kiểm tra isValidEmail với email rỗng* @Test  
 *void* testIsValidEmail\_EmptyEmail\_ReturnsFalse() {  
 *// Gọi isValidEmail với email rỗng  
 boolean* result = accountService.isValidEmail("");  
 *// Kiểm tra kết quả phải là false, nếu không thì hiển thị thông báo lỗi  
 assertFalse*(result, "Empty email should return false");  
 }  
  
 *// Test kiểm tra isValidEmail với email không hợp lệ* @Test  
 *void* testIsValidEmail\_InvalidEmailFormat\_ReturnsFalse() {  
 *// Gọi isValidEmail với email không hợp lệ (thiếu dấu chấm trong domain)  
 boolean* result = accountService.isValidEmail("invalid.email@com");  
 *// Kiểm tra kết quả phải là false, nếu không thì hiển thị thông báo lỗi  
 assertFalse*(result, "Invalid email format should return false");  
 }  
}

Chạy các hàm test



File test-data.csv

