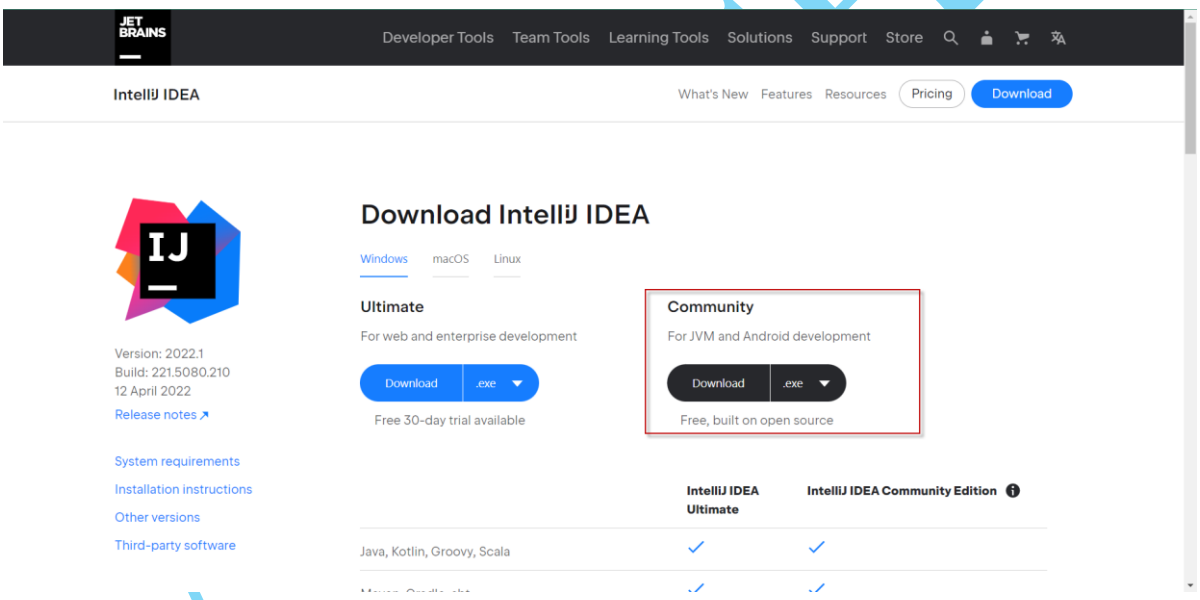
Tìm hiểu công cụ kiểm thử JUNIT5

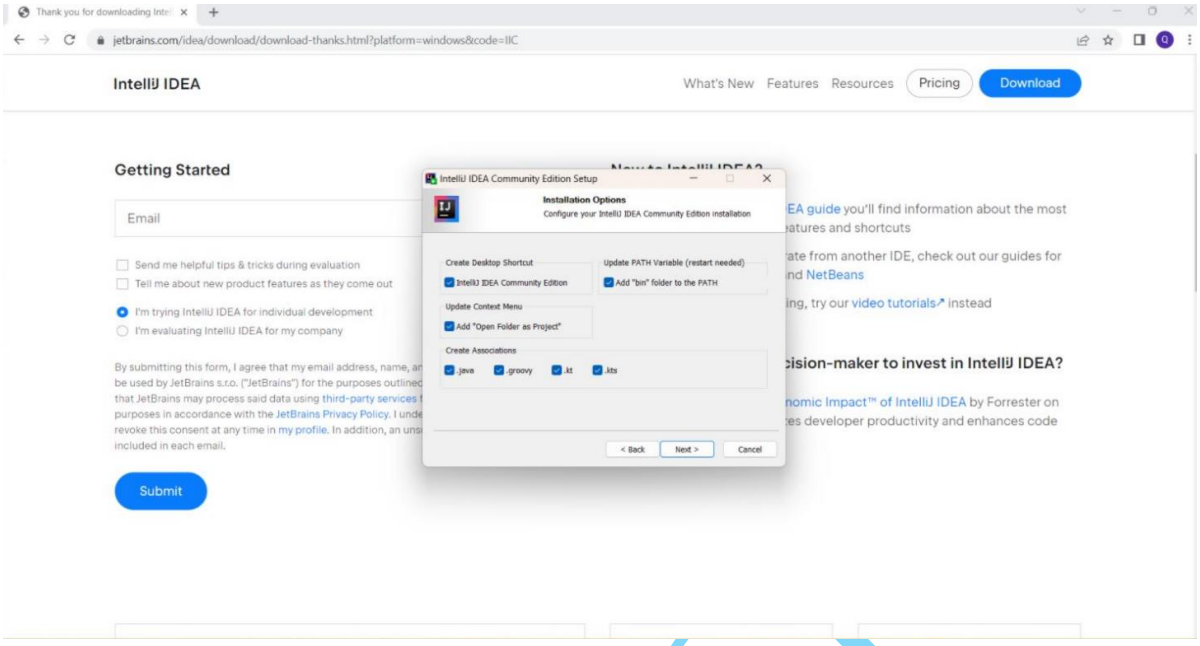
I. JUnit là gì

1. JUnit là gì?

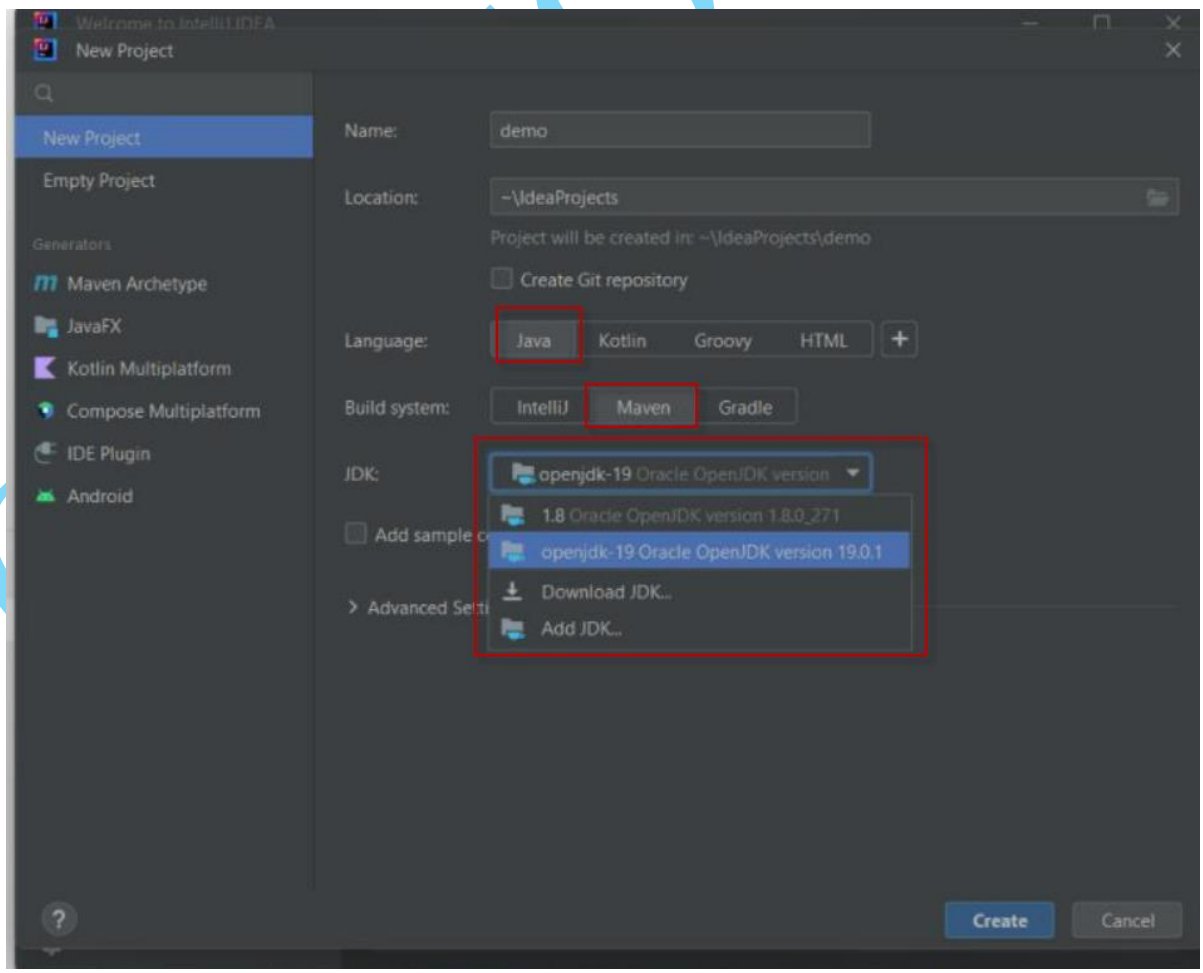
- ✓ JUnit là một framework dùng để unit test cho ngôn ngữ lập trình Java. (Một unit ở đây có thể là một hàm, phép tính, một module, một class -- thường thì sẽ sử dụng method để làm unit test)
- ✓ JUnit là một mã nguồn mở, miễn phí.
- ✓ Link download: https://www.jetbrains.com/idea/download/#section=windows



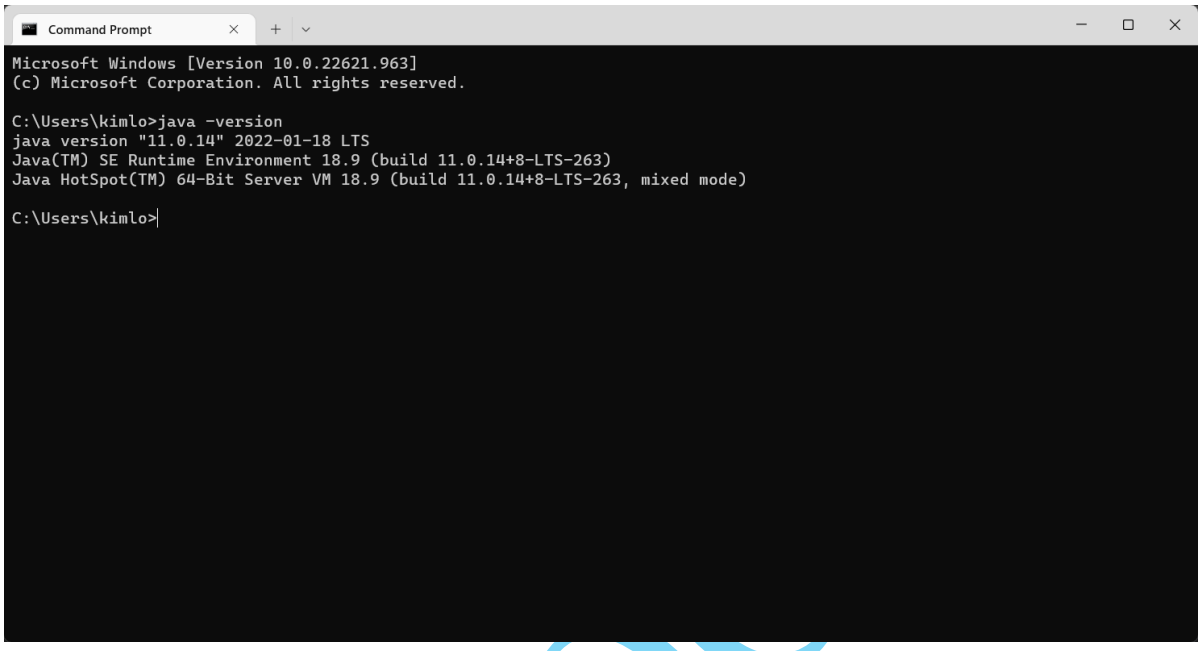
Bấm Next trong quá trình cài, và tick chọn như bên dưới:



Sau khi cài xong, tạo Project với lựa chọn là Java + Maven, chọn JDK trên máy, nếu máy chưa cài JDK thì chọn download JDK:



Vào cmd: java -version để check JDK vừa cài đặt.



Nếu cài JDK rồi vẫn bị lỗi

```
'java' is not recognized as an internal or external command, operable program or batch file.
```

thì vào link bên dưới để setup JAVA_HOME:

https://shareprogramming.net/cach-dat-bien-moi-truong-java_home-trong-windows-10/

2. Các tính năng của Junit:

- ✓ JUnit là một khung kiểm tra nguồn mở, được sử dụng để viết và chạy test.
- ✓ Cung cấp annotation để định dạng các test method
- ✓ Cung cấp assertion để kiểm thử các kết quả mong đợi
- ✓ Cung cấp các trình chạy để chạy test
- ✓ Cho phép bạn viết code nhanh hơn, cải thiện chất lượng
- ✓ JUnit khá đơn giản. Nó ít phức tạp hơn và tốn ít thời gian hơn
- ✓ JUnit có thể được chạy tự động và tự kiểm tra kết quả, cung cấp phản hồi nhanh chóng. Không cần thiết phải xem xét thủ công các báo cáo về kết quả kiểm thử.

- ✓ Các bài test JUnit có thể được tổ chức thành các bộ kiểm thử chứa các trường hợp kiểm thử, thậm chí là chứa các bộ kiểm thử khác.
- ✓ JUnit thể hiện tiến độ kiểm thử trên một thanh. Nếu thanh có màu xanh, bài test đang chạy êm ả. Ngược lại, nếu thanh chuyển đỏ, tức là bài test thất bại.

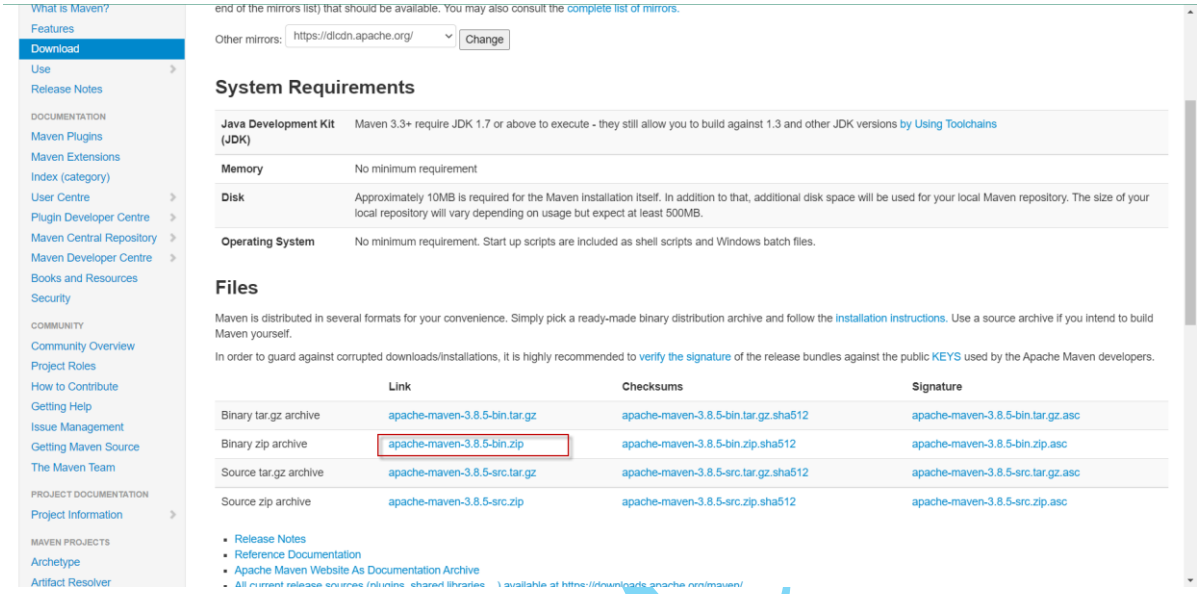
3. Maven:

- ✓ Apache maven là một chương trình quản lý dự án cho phép các developers có thể quản lý về version, các dependencies (các thư viện sử dụng trong dự án), quản lý build, tự động download javadoc & source, …
- ✓ Vì sao phải sử dụng maven?

  Hãy thử tượng nếu dự án của chúng ta sử dụng rất nhiều thư viên thứ 3: struts, hibernate, spring, … Việc import thư viện và các dependency (dịch hiểu là "sự phụ thuộc") là rất vất vả, chưa kể đến việc version của các thư viện có thể conflict với nhau, việc import đầy đủ các thư viện là cả một vấn đề. VD như bạn muốn sử dụng struts, thì điều bắt buộc là chúng ta phải import cả thư viện servlet.
- ✓ Maven sẽ giúp chúng ta tự động hóa dự án → xây dựng dự án thành một file jar có thể thực thi và quản lý các *dependencies*.

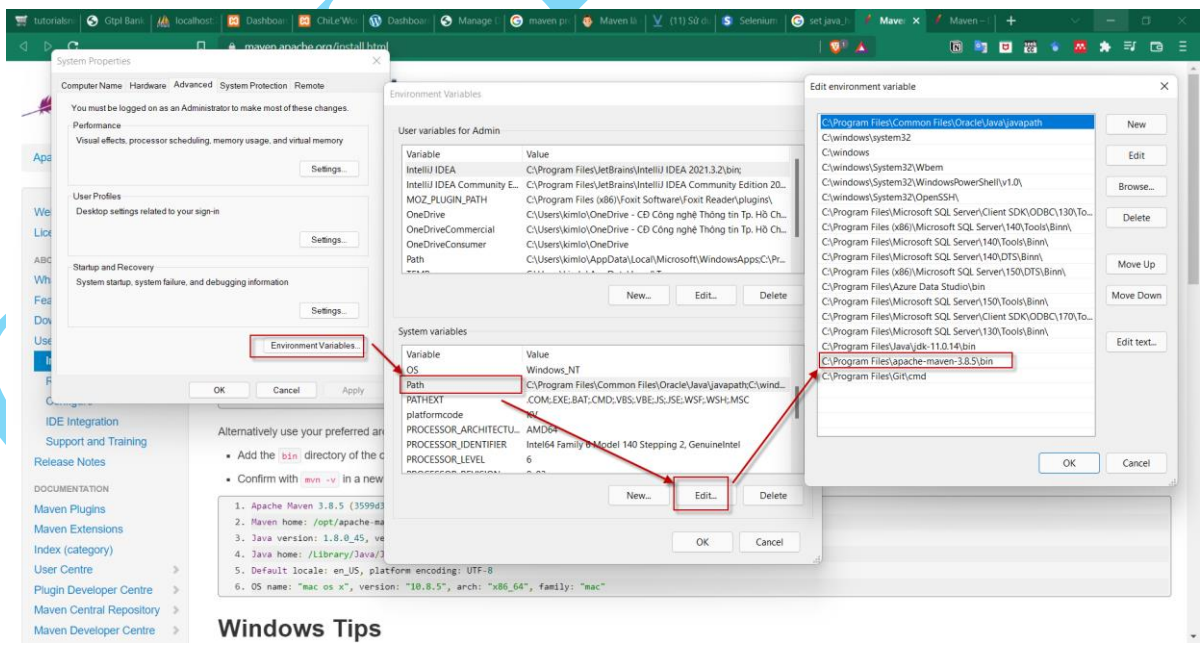
4. Cài đặt Maven:

✓ Download Maven: https://maven.apache.org/download.cgi

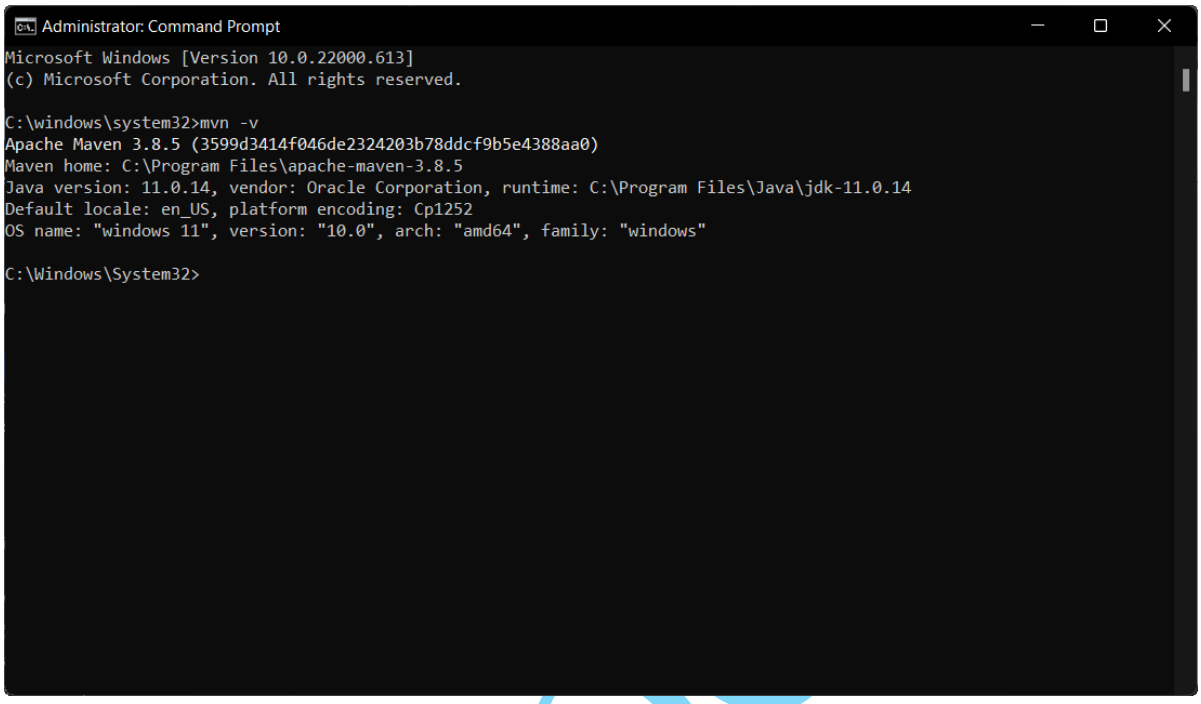


✓ Vào đường dẫn https://maven.apache.org/install.html để xem cách cài đặt Maven project.

- Giải nén file và thêm thư mục bin vào PATH environment variable

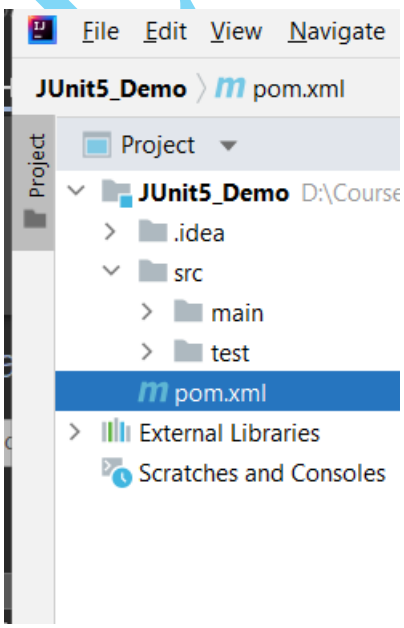


- Gõ cmd - run as administrator: *mvn -v*



5. Ví dụ JUnit với IntelIJ +Maven

✓ Tạo maven project:

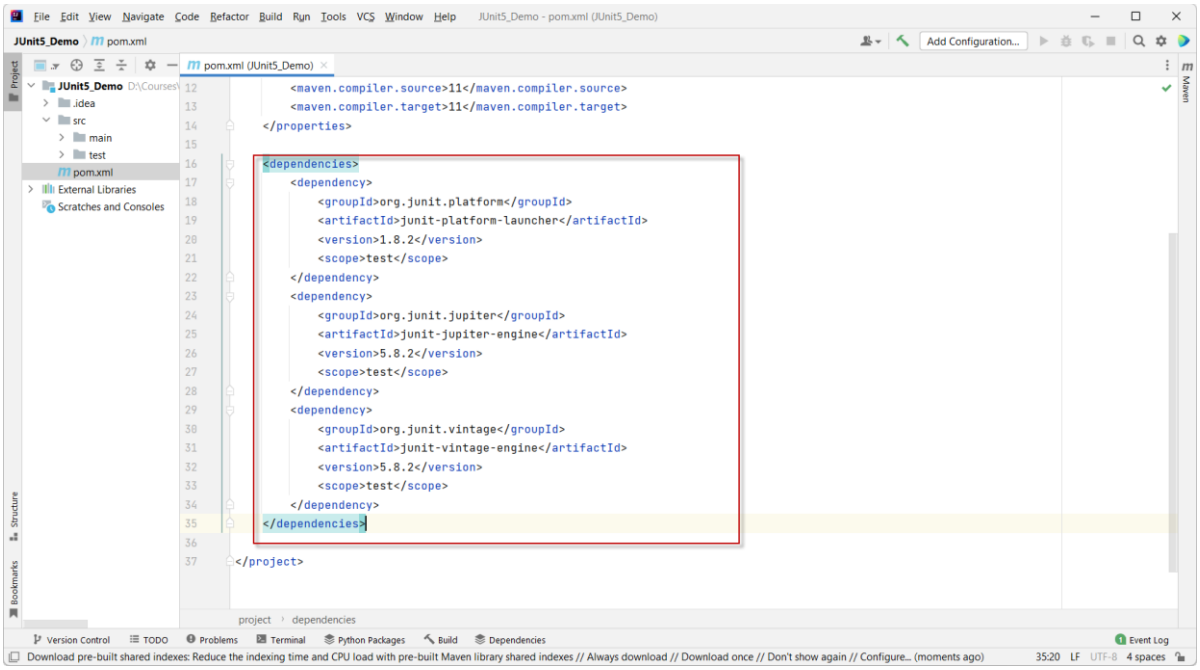


(Maven Project sẽ tự tạo source folder để viết code src/main/java và folder để viết test src/test/java)

✓ Để sử dụng thư viện **Junit5** ta khai báo:

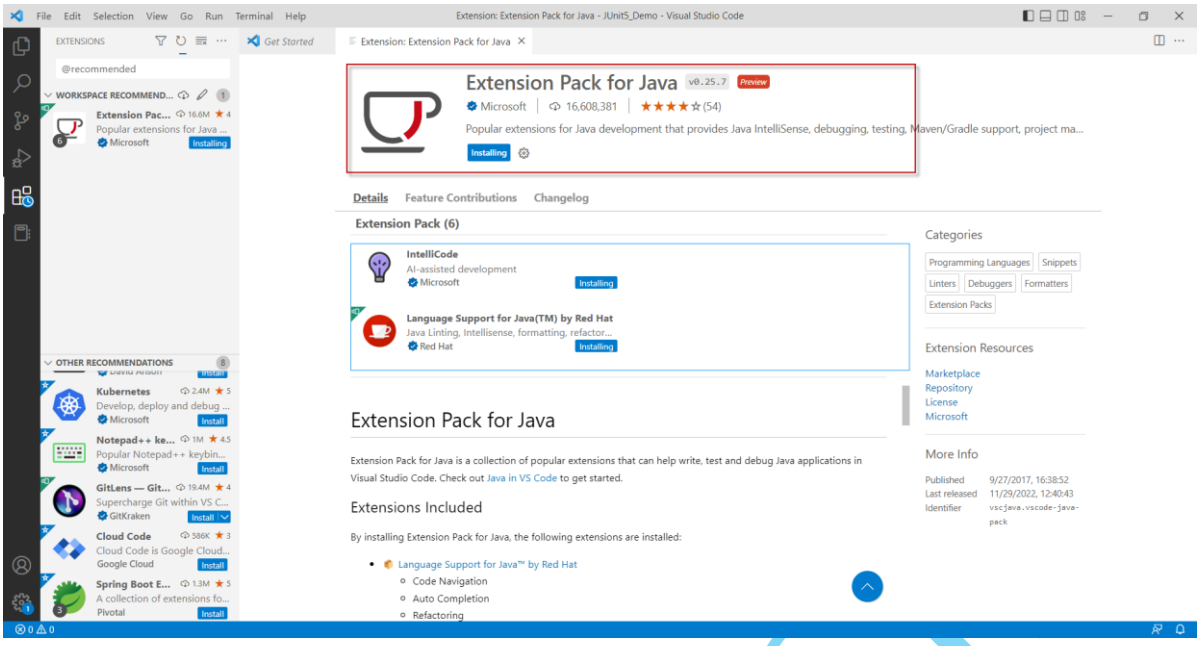
```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.10.0</version>
        <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-params -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-params</artifactId>
        <version>5.10.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.10.0</version>
        <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.junit.vintage/junit-vintage-engine -->
    <dependency>
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
        <version>5.10.0</version>
        <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.junit.platform/junit-platform-suite-engine -->
    <dependency>
```

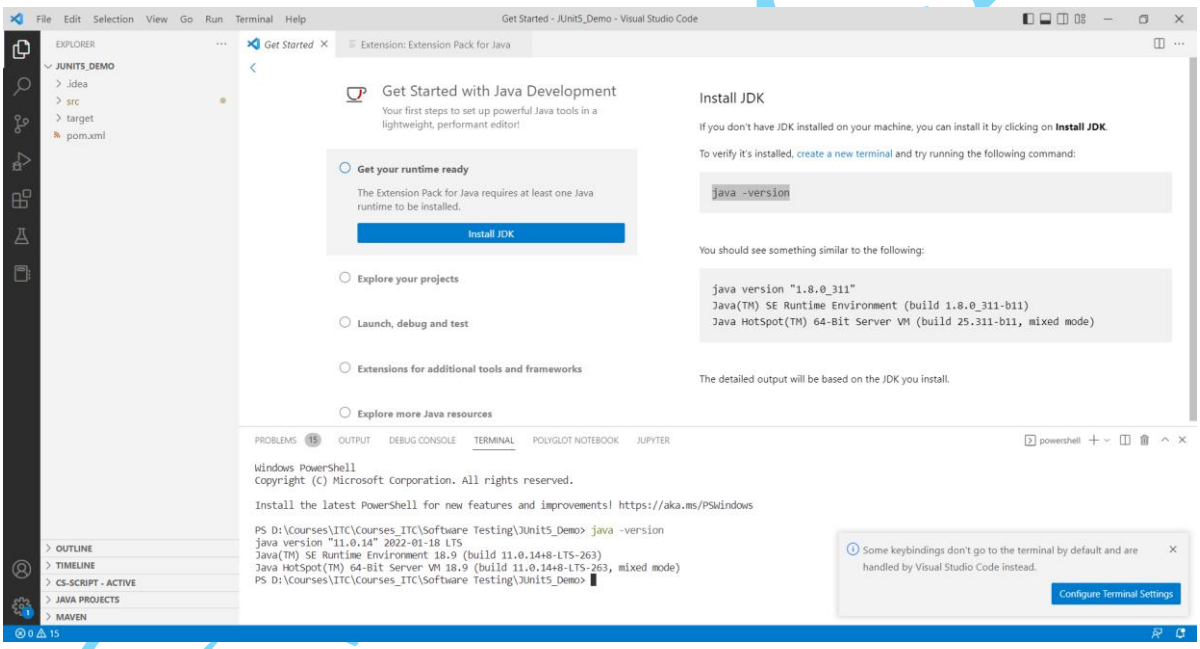
```
        <groupId>org.junit.platform</groupId>
        <artifactId>junit-platform-suite-engine</artifactId>
        <version>1.10.0</version>
        <scope>test</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.junit.platform/junit-platform-launcher -->
    <dependency>
        <groupId>org.junit.platform</groupId>
        <artifactId>junit-platform-launcher</artifactId>
        <version>1.10.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```



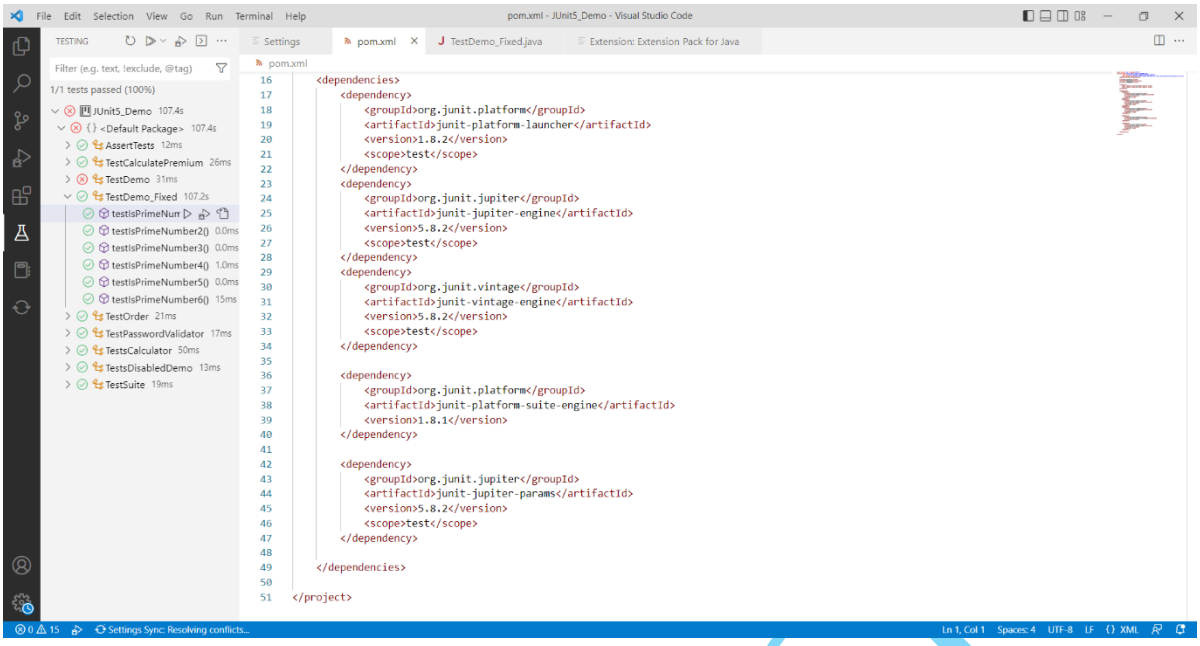
Mục tiêu của JUnit là kiểm tra kết quả của 1 unit có trả về kết quả giống như mong muốn hay không.

Lưu ý: nếu sử dụng IDE là Visual Studio Code thì cài đặt extension "**Extension Pack For Java**".





- ✓ Tham khảo link https://code.visualstudio.com/docs/java/java-build để tạo Maven project.
- ✓ Sau đó copy code thư viện Junit5 vào pom.xml để thực hiện việc test như hướng dẫn bên trên.



6. Ví dụ:

Ví dụ: method isPrimeNumber để kiểm tra một số có phải là số nguyên tố hay không

(Số nguyên tố là số tự nhiên chỉ có hai ước số dương phân biệt là 1 và chính nó. (Số 1 chỉ có một ước số dương là chính nó nên nó không phải là số nguyên tố))

```
public class Demo {

    public  boolean  isPrimeNumber(int input)  {
        for  (int i = 2; i < input; i++)  {
            if  (input % i == 0)
            return false;
        }
        return true;
    }
}
```

Bây giờ ta sẽ sử dụng JUnit để kiểm tra hàm trên với các đầu vào khác nhau:

- ✓ Code Test:

Ở đây ta có 6 test case tương ứng với 6 method

```
public class TestDemo {

    @Test
    public  void  testIsPrimeNumber1()  {
        Demo demo1 = new  Demo();
        boolean result = demo1.isPrimeNumber(-1);
        assertFalse(result);
    }

    @Test
    public  void  testIsPrimeNumber2()  {
        Demo demo1 = new  Demo();
        boolean result = demo1.isPrimeNumber(0);
        assertFalse(result);
    }
    @Test
    public  void  testIsPrimeNumber3()  {
        Demo demo1 = new  Demo();
        boolean result = demo1.isPrimeNumber(1);
        assertFalse(result);
    }

    @Test
    public  void  testIsPrimeNumber4()  {
        Demo demo1 = new  Demo();
        boolean result = demo1.isPrimeNumber(2);
        assertTrue(result);
    }
    @Test
    public  void  testIsPrimeNumber5()  {
        Demo demo1 = new  Demo();
        boolean result = demo1.isPrimeNumber(4);
        assertFalse(result);
    }
```

```
    @Test
    public void testIsPrimeNumber6() {
        Demo demo1 = new Demo();
        boolean result = demo1.isPrimeNumber(5);
        assertTrue(result);
    }
}
```

- Hàm thứ nhất đầu vào là số âm '-1' nên kết quả mong đợi sẽ là false nên dùng assertFalse
- Hàm thứ hai đầu vào là số 0, 0 không phải là số nguyên tố nên kết quả mong đợi sẽ là false
- Hàm thứ ba đầu vào là số 1, 1 không phải là số nguyên tố nên kết quả mong đợi sẽ là false
- Hàm thứ tư đầu vào là số '2', 2 là số nguyên tố nên kết quả mong đợi sẽ là true nên mình dùng assertTrue
- Hàm thứ năm đầu vào là số '4', 4 không phải là số nguyên tố nên kết quả mong đợi sẽ là false
- Hàm thứ sáu đầu vào là số '5', 5 là số nguyên tố nên kết quả mong đợi sẽ là true

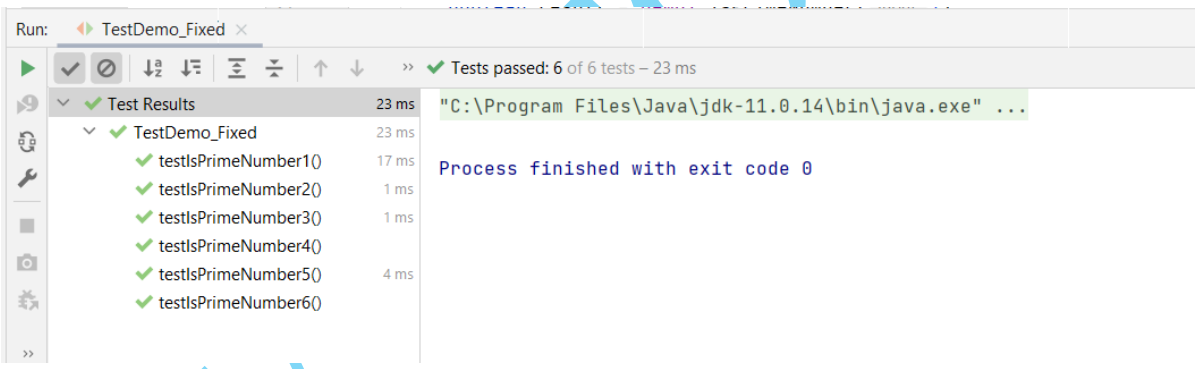
Kết quả chạy:

Như chúng ta thấy trên hình 3 test case đầu tiên bị thất bại tức là method isPrimeNumber đang bị sai cho những trường hợp đó (do chưa kiểm tra trường hợp bằng 0, 1 và <0)

Bây giờ sửa lại method isPrimeNumber để fix các lỗi đó:

```
public class Demo {
   public  boolean  isPrimeNumber(int input)  {
      if  (input < 2)  {
       return false;
      }
      for  (int i = 2; i < input; i++)  {
       if  (input % i == 0)
       return false;
      }
      return true;
   }
}
```

Chạy lại các test case:



Tất cả các test case đều pass.

II. Assertions:

Assertions JUnit Ở ví dụ trên chúng ta dùng assertFalse,assertTrue để kiểm tra kết quả cho từng trường hợp với mong muốn kết quả trả về là false, hoặc true.

Vậy với những trường hợp kết quả mong muốn không phải là boolean (true, false) mà là String, byte, Object… thì ta sẽ dùng assertEquals, assertArrayEquals…

Đó chính là assertions trong JUnit, assertions chính là những method dùng để kiểm tra kết quả của đơn vị cần test có đúng với mong đợi không.

Với mỗi loại kết quả đầu ra ta có một method assert tương ứng.Như so sánh đối tượng, so sánh mảng, kiểm tra null…

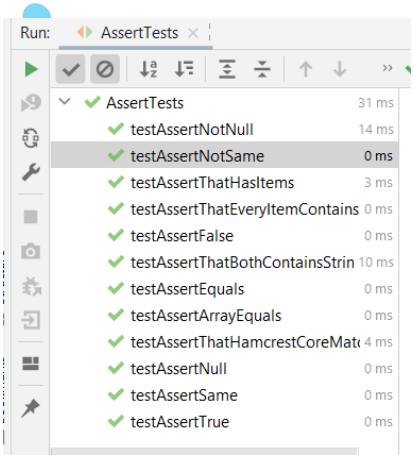
Code ví dụ assertions JUnit của các method assert sẽ là assert[kiểu so sánh](expecteds_value, actuals_value) hoặc assert[kiểu so sánh](message, expecteds_value, actuals_value) với message là dữ liệu in ra nếu assert thất bại

```
import static org.hamcrest.CoreMatchers.allOf;
import static org.hamcrest.CoreMatchers.anyOf;
import static org.hamcrest.CoreMatchers.both;
import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.CoreMatchers.everyItem;
import static org.hamcrest.CoreMatchers.hasItems;
import static org.hamcrest.CoreMatchers.not;
import static org.hamcrest.CoreMatchers.sameInstance;
import static org.hamcrest.CoreMatchers.startsWith;
import static org.junit.Assert.assertArrayEquals;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertNotSame;
import static org.junit.Assert.assertNull;
import static org.junit.Assert.assertSame;
import static org.junit.Assert.assertThat;
import static org.junit.Assert.assertTrue;
import java.util.Arrays;
import org.hamcrest.core.CombinableMatcher;
import org.junit.Test;
public class AssertTests {
  @Test
  public void testAssertArrayEquals() {
    byte[] expected = "trial".getBytes();
    byte[] actual = "trial".getBytes();
    assertArrayEquals("failure - byte arrays not same", expected, actual);
```

```
}
@Test
public void testAssertEquals() {
  assertEquals("failure - strings are not equal", "text", "text");
}
@Test
public void testAssertFalse() {
  assertFalse("failure - should be false", false);
}
@Test
public void testAssertNotNull() {
  assertNotNull("should not be null", new Object());
}
@Test
public void testAssertNotSame() {
  assertNotSame("should not be same Object", new Object(), new Object());
}
@Test
public void testAssertNull() {
  assertNull("should be null", null);
}
@Test
public void testAssertSame() {
  Integer aNumber = Integer.valueOf(768);
  assertSame("should be same", aNumber, aNumber);
}
// JUnit Matchers assertThat
@Test
public void testAssertThatBothContainsString() {
  assertThat("albumen", both(containsString("a")).and(containsString("b")));
}
@Test
public void testAssertThatHasItems() {
  assertThat(Arrays.asList("one", "two", "three"), hasItems("one", "three"));
```

```
  }
  @Test
  public void testAssertThatEveryItemContainsString() {
    assertThat(Arrays.asList(new String[] { "fun", "ban", "net" }),
everyItem(containsString("n")));
  }
  // Core Hamcrest Matchers with assertThat
  @Test
  public void testAssertThatHamcrestCoreMatchers() {
    assertThat("good", allOf(equalTo("good"), startsWith("good")));
    assertThat("good", not(allOf(equalTo("bad"), equalTo("good"))));
    assertThat("good", anyOf(equalTo("bad"), equalTo("good")));
    assertThat(7, not(CombinableMatcher.<Integer>either(equalTo(3)).or(equalTo(4))));
    assertThat(new Object(), not(sameInstance(new Object())));
  }
  @Test
  public void testAssertTrue() {
    assertTrue("failure - should be true", true);
  }
}
```

Kết quả:

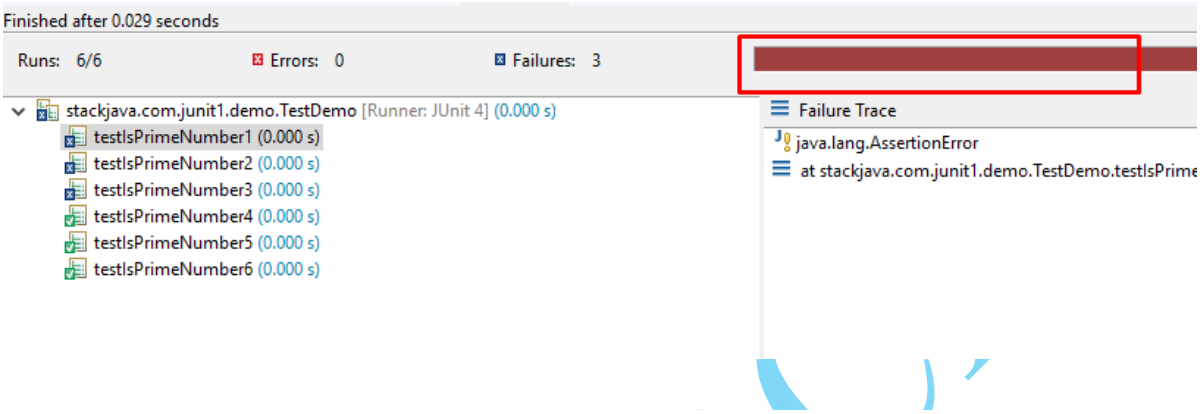


Tham khảo thêm ở link: https://junit.org/junit5/docs/snapshot/user-guide/

III. Test runner và Test suite:

1. Test runners:

Bình thường, các IDE như NetBeans, Eclipse đều có sẵn trình chạy (runner) cho JUnit để hiển thị kết quả các test case, ví dụ:



2. Annotation @Suite

Sử dụng @Suite để chạy nhiều class JUnit cùng lúc.

3. Test Suite - Tạo bộ test với Junit:

Thông thường 1 class test sẽ sử dụng để test cho một chức năng, một unit. Vậy nếu muốn chạy nhiều class test để xem kết quả thì như thế nào? Câu trả lời là test suite, ta sẽ tạo một bộ gồm nhiều class để thực hiện test và xem kết quả sau một lần chạy.

- ✓ Insert dependency vào file pom.xml

```
<dependency>
   <groupId>org.junit.platform</groupId>
   <artifactId>junit-platform-suite-engine</artifactId>
   <version>1.8.1</version>
</dependency>
```

```
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
            <version>5.8.2</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-suite-engine</artifactId>
            <version>1.8.1</version>
        </dependency>
    </dependencies>

</project>
```

✓ Để tạo test suite ta sử dụng:

@Suite(Suite.class)và @SelectClasses (TestClass1.class, ...). Bên trong @ SelectClasses sẽ là các class test được chạy.

Ví dụ:

Tạo class java PasswordValidator:

```
public class PasswordValidator {
   public boolean isPasswordStrong (String password) throws Exception{
      if (password.length() >12)
         throw new Exception("Password is too long!");
      return password.length()>7;
   }
}
```

Class TestPasswordValidator:

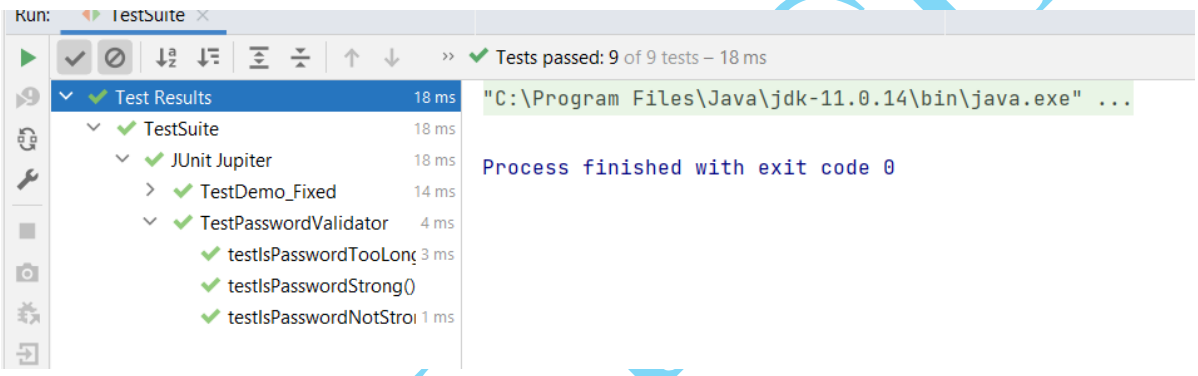
```
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.FixMethodOrder;
import org.junit.Test;
```

```
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.runners.MethodSorters;

import static org.junit.Assert.*;

//@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class TestPasswordValidator {

    @Test
    public void testIsPasswordStrong() throws Exception{
        PasswordValidator pv = new PasswordValidator();
        boolean isStrong = pv.isPasswordStrong("12345678");
        assertTrue(isStrong);
    }

    @Test
    public void testIsPasswordNotStrong() throws Exception{
        PasswordValidator pv = new PasswordValidator();
        boolean isStrong = pv.isPasswordStrong("1234");
        assertFalse(isStrong);
    }

    @Test
    public void testIsPasswordTooLong() throws Exception{
        PasswordValidator pv = new PasswordValidator();
        assertThrows(Exception.class, ()-> pv.isPasswordStrong("123456vhvjhjhfhj"));
    }
}
```

Tạo bộ test gồm 2 class test là TestDemo_Fixed.java và TestPasswordValidator.java

```
import org.junit.platform.suite.api.SelectClasses;
import org.junit.platform.suite.api.Suite;

@SelectClasses({TestDemo_Fixed.class, TestPasswordValidator.class})
@Suite
public class TestSuite {
}
```

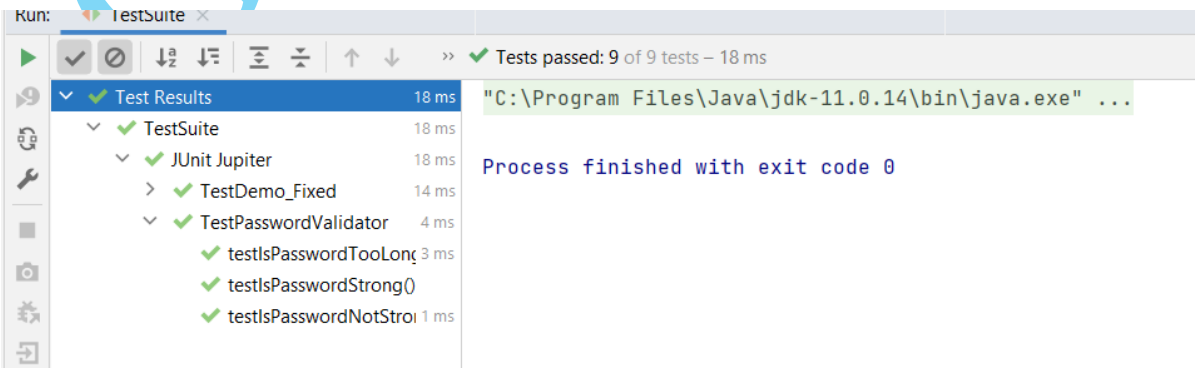
Kết quả:



IV. Thứ tự thực thi các class test trong một test suite:

Thứ tự thực thi các class test trong một test suite chính là thứ tự khai báo các class đó trong annotation @ SelectClasses

Ví dụ khai báo class TestDemo_Fixed.java trước class TestPasswordValidator.java trong test suite thì nó sẽ thực hiện chạy class TestDemo_Fixed.java trước



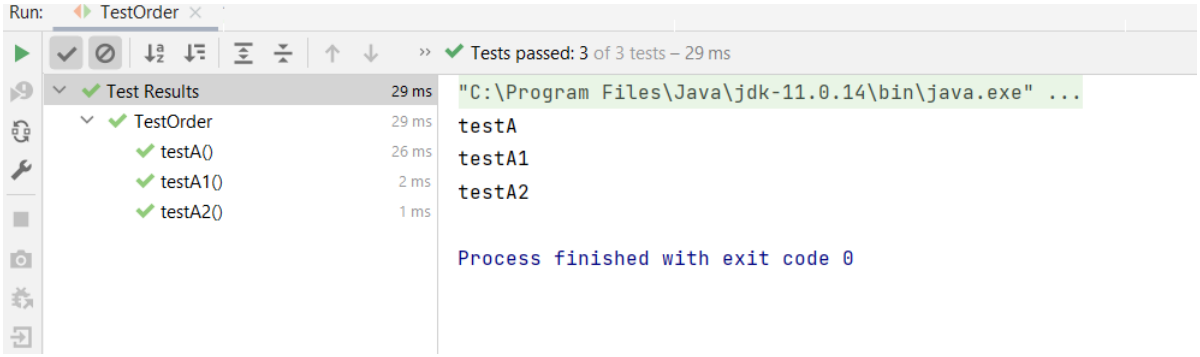
Nếu mình đổi thứ tự lại thành khai báo class TestPasswordValidator.java trước thì nó sẽ chạy class TestPasswordValidator.java trước.

V. Thứ tự thực thi các method trong một class test:

Để chỉ rõ thứ tự chạy của các method trong class test ta dùng annotation @TestMethodOrder ở đầu class.

Ví dụ: Thực hiện chạy các method test theo thứ tự tên method:

```
import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;
import static org.junit.jupiter.api.Assertions.*;

@TestMethodOrder(OrderAnnotation.class)
public class TestOrder {
    @Test
    @Order(3)
    public void testA2()  {
        System.out.println("testA2");
        assertTrue(true);
    }
    @Test
    @Order(1)
    public void testA()  {
        System.out.println("testA");
        assertTrue(true);
    }
    @Test
    @Order(2)
    public void testA1()  {
        System.out.println("testA1");
        assertTrue(true);
    }
}
```



VI. JUnit Expected Exceptions:

Ở những bài trước chúng ta đã tìm hiểu cách test method bằng việc so sánh các giá trị trả về của test case. Vậy với những trường hợp method không trả về giá trị hoặc xảy ra exception thì chúng ta thực hiện unit test như nào?

✓ Sử dụng expect exception:

Với những test case cho unit (method, đoạn code) ứng với trường hợp xảy ra exception thì chúng ta expect kết quả là test case đó sẽ trả về / xảy ra exception chứ không phải một giá trị cụ thể.

Ví dụ:

```
@Test
public void testIsPasswordTooLong() throws Exception{
    PasswordValidator pv = new PasswordValidator();
    assertThrows(Exception.class, ()-> pv.isPasswordStrong("123456vhvjhjhfhj"));
}
```

VII. Disabled tests:

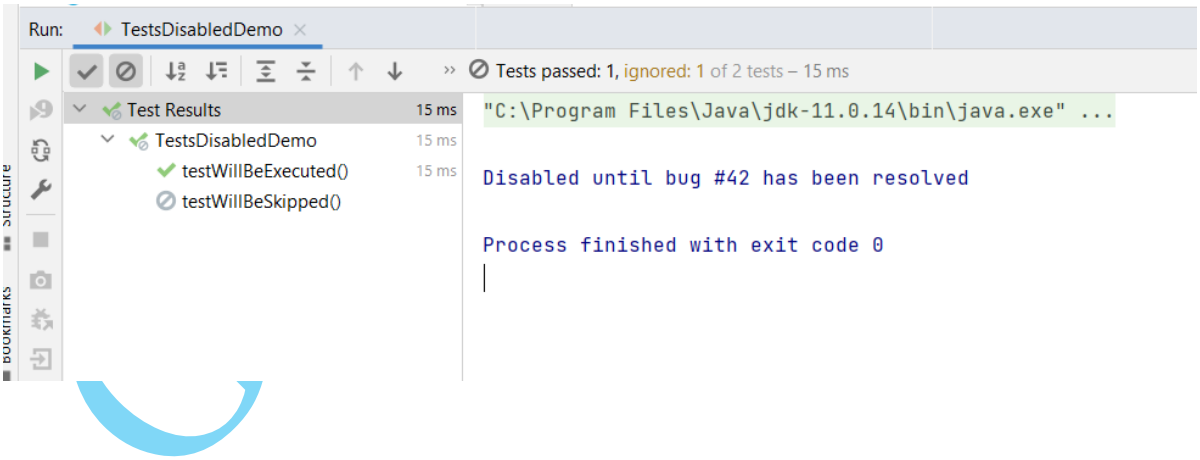
Vì một lý do nào đó, bạn muốn tạm thời vô hiệu hóa test case (bỏ qua không chạy test case đó).

Thông thường ta sẽ xóa hoặc comment annotation @Test như thế trình test runner sẽ bỏ qua method đó nhưng đồng thời test case đó cũng sẽ không được report (bạn có thể quên mất là có test case đó).

Biện pháp thay thể là sử dụng annotation @ Disabled ở trước hoặc sau annotation @Test, sau khi chạy JUnit test, nó vẫn thông báo là có test case đó nhưng đang bị disable.

Ví dụ:

```
import org.junit.Ignore;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;

public class TestsDisabledDemo {
    @Disabled("Disabled until bug #42 has been resolved")
    @Test
    void testWillBeSkipped() {
    }
    @Test
    void testWillBeExecuted() {
    }
}
```



VIII. Tổng quan về các chú thích quan trọng nhất trong JUnit 5:

Chú thích - Annotations

Chú thích	Sự miêu tả
@Test	Xác định một phương pháp như một phương pháp thử nghiệm.
@Disabled("reason")	Vô hiệu hóa một phương pháp kiểm tra với một lý do tùy chọn.
@BeforeEach	Thực hiện trước mỗi bài kiểm tra. Được sử dụng để chuẩn bị môi trường thử nghiệm, ví dụ: khởi tạo các trường trong lớp thử nghiệm, cấu hình môi trường, v.v.
@AfterEach	Thực hiện sau mỗi lần kiểm tra. Được sử dụng để dọn dẹp môi trường thử nghiệm, ví dụ: xóa dữ liệu tạm thời, khôi phục giá trị mặc định, dọn dẹp cấu trúc bộ nhớ đắt tiền.
@DisplayName("<Name>")	<Tên> sẽ được hiển thị bởi người chạy thử nghiệm. Ngược lại với tên phương thức, tên có thể chứa khoảng trắng để cải thiện khả năng đọc.
@RepeatedTest(<Number>)	Tương tự như @Testnhưng lặp lại kiểm tra một <Số> lần
@BeforeAll	Chú thích một phương thức tĩnh được thực thi một lần, trước khi bắt đầu tất cả các thử nghiệm. Nó được sử dụng để thực hiện các hoạt động tốn nhiều thời gian, ví dụ, để kết nối với cơ sở dữ liệu. Các phương thức được đánh dấu bằng chú thích này cần được xác định là statichoạt động với JUnit.
@AfterAll	Chú thích một phương thức tĩnh được thực thi một lần, sau khi tất cả các thử nghiệm đã kết thúc. Nó được sử dụng để thực hiện các hoạt động dọn dẹp, ví dụ, để ngắt kết nối khỏi cơ sở dữ liệu. Các phương thức được chú thích bằng chú thích này cần được xác định là statichoạt động với JUnit.

Chú thích - Annotations

Chú thích	Sự miêu tả
@TestFactory	Chú thích một phương pháp là một Nhà máy để tạo các thử nghiệm động
@Nested	Cho phép bạn lồng các lớp kiểm tra bên trong để nhóm các bài kiểm tra của bạn và có thêm các phương thức @BeforeEach và @AfterEach.
@Tag("<TagName>")	Thẻ một phương pháp kiểm tra, các bài kiểm tra trong JUnit 5 có thể được lọc theo thẻ. Ví dụ: chỉ chạy các bài kiểm tra được gắn thẻ "nhanh".
@ExtendWith	Cho phép bạn đăng ký một lớp Mở rộng bổ sung chức năng cho các bài kiểm tra

Thực hành:

1. Tạo các script test như đã hướng dẫn trong bài học.
2. Tạo code Calculator đơn giản cho phép tính +,-,*,/
3. Tạo file csv gồm 3 cột a,b, result, lưu vào thư mục /src/main/resources
4. Tạo class script test cho hàm calculator đọc giá trị từ file csv.