



T3H

IT-Institute

# Lập trình Java – Unit Testing

---

Ths. Vũ Duy Khương

- 1 **Unit Test là gì?**
- 2 **Vòng đời Unit Test**
- 3 **Tạo sao phải thực hiện UT**
- 4 **Mock Object**
- 5 **JUnit là gì?**
- 6 **Ví dụ JUnit trên Eclipse**
- 7 **Các tính năng của JUnit Test Framework**
- 8 **Bài tập thực hành**

# Unit Test là gì?

- - Unit Test có nghĩa là kiểm thử đơn vị, một bước trong kiểm thử phần mềm. Với Unit Test, chỉ có những đơn vị hay những thành phần riêng lẻ của phần mềm được kiểm thử. Mục đích là để xác định rằng mỗi đơn vị của phần mềm đều hoạt động đúng như kỳ vọng.
  - Unit Testing được tiến hành trong quá trình phát triển (lập trình) một phần mềm. Unit Test cô biệt một phần của các mã code và đánh giá sự chính xác của chúng. Một đơn vị có thể là một hàm (function), một phương thức (method), một quy trình (procedure), một mô-đun hay một đối tượng.

# Unit Test là gì?

---

- Unit test là test do developer viết, được chạy để kiểm tra các hàm do developer viết ra có sai hay không. UT thường được chạy mỗi khi build để đảm bảo các hàm đều chạy đúng sau khi ta sửa code

# Vòng đời của Unit Test

---

- Unit test có 3 trạng thái cơ bản:
  - Fail (trạng thái lỗi)
  - Ignore (tạm ngừng thực hiện)
  - Pass (trạng thái làm việc)
- Toàn bộ UT được vận hành trong một hệ thống tách biệt. Có rất nhiều phần mềm hỗ trợ thực thi UT với giao diện trực quan. Thông thường, trạng thái của UT được biểu hiện bằng các màu khác nhau: màu xanh (pass), màu vàng (ignore) và màu đỏ (fail).

# Vòng đời của Unit Test

---

- UT chỉ thực sự đem lại hiệu quả khi:
  - Được vận hành lặp lại nhiều lần.
  - Tự động hoàn toàn.
  - Độc lập với các UT khác.

# Ưu điểm của Unit Test

---

- Giúp sửa bug sớm trong chu trình phát triển sản phẩm và tiết kiệm chi phí
- Giúp các lập trình viên hiểu được nền tảng mã kiểm thử và cho phép họ đưa ra các thay đổi nhanh chóng
- Có thể được sử dụng như các ghi chép về dự án, nếu hiệu quả
- Tái sử dụng code. Kết hợp cả code của bạn và kiểm thử của bạn cho dự án mới. Thay đổi code cho đến khi kiểm thử chạy được

# Tại sao phải kiểm thử đơn vị?

- Lập trình viên sẽ tiến hành kiểm thử đơn vị một cách đơn giản để tiết kiệm thời gian. Kiểm thử đơn vị không được tiến hành kỹ lưỡng sẽ dẫn đến chi phí sửa bug cao hơn khi kiểm thử hệ thống, kiểm thử tích hợp. Thậm chí, có thể vẫn còn sót lỗi cho đến tận khi tiến hành kiểm thử beta, một phương pháp kiểm thử xác thực.



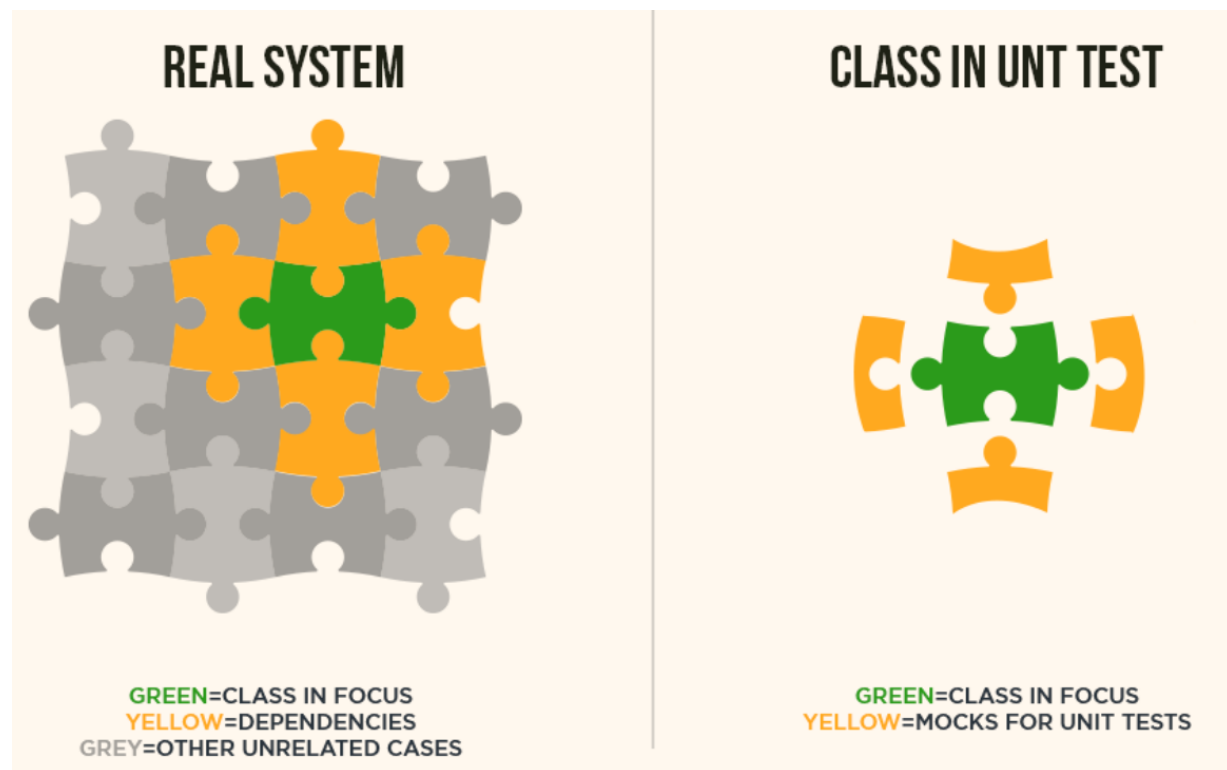
# Xây dựng UT với mô hình đối tượng ảo (Mock Object)

---

- Trong UT, mỗi một đối tượng hay một phương thức riêng lẻ được kiểm tra tại một thời điểm và chúng ta chỉ quan tâm đến các trách nhiệm của chúng có được thực hiện đúng hay không. Tuy nhiên trong các dự án phần mềm phức tạp thì UT không còn là quy trình riêng lẻ, nhiều đối tượng (đơn vị chương trình) không làm việc độc lập mà tương tác với các đối tượng khác như kết nối mạng, cơ sở dữ liệu hay dịch vụ web.

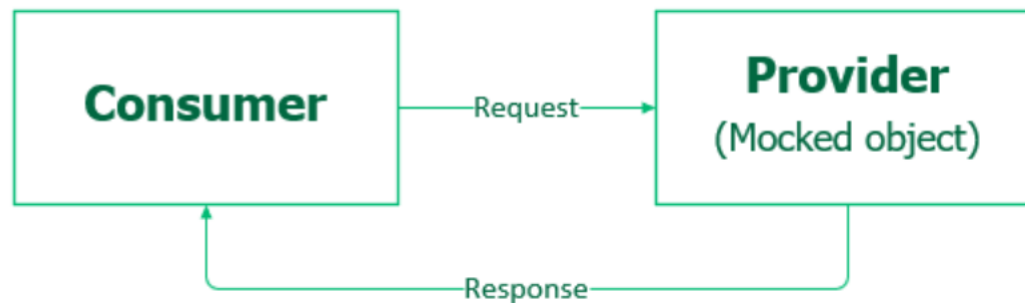
# Xây dựng UT với mô hình đối tượng ảo (Mock Object)

➔ Như vậy công việc kiểm nghiệm có thể bị trì hoãn gây tác động xấu đến quy trình phát triển chung. Để giải quyết các vấn đề này người ta đưa ra mô hình “Mock Object” hay đối tượng ảo (hoặc đối tượng giả).



# Mock Object (MO) là gì ?

- Mock object (MO) là một đối tượng ảo, mô phỏng các tính chất và hành vi giống hệt như đối tượng thực được truyền vào bên trong khối mã đang vận hành nhằm kiểm tra tính đúng đắn của các hoạt động bên trong.
- Thay vì lấy data từ một real service, chúng ta sử dụng một bộ test data mà input và output đã được định nghĩa rõ ràng từ một Mock Object và chúng ta có thể dùng nó cho đối tượng muốn test.



# Đặc điểm của Mock Object

---

- Đơn giản hơn đối tượng thực nhưng vẫn giữ được sự tương tác với các đối tượng khác
- Không lặp lại nội dung đối tượng thực
- Cho phép thiết lập các trạng thái riêng trợ giúp kiểm tra

# Lợi ích của Mock Object

- Đảm bảo công việc kiểm nghiệm không bị gián đoạn bởi các yếu tố bên ngoài, giúp các người viết tập trung vào một chức năng nghiệp vụ cụ thể, từ đó tạo ra UT vận hành nhanh hơn
- Giúp tiếp cận hướng đối tượng tốt hơn. Nhờ MO chúng ta có thể phát hiện interface cần tách ở một số lớp
- Dễ dàng cho việc kiểm nghiệm. Thay vì gọi các đối tượng thực vận hành nặng nề, chúng ta có thể gọi các MO đơn giản hơn để kiểm tra nhanh liên kết giữa các thủ tục, công việc kiểm nghiệm có thể tiến hành nhanh hơn

# Phạm vi sử dụng của Mock Object

---

MO được sử dụng trong các trường hợp sau:

- Khi cần lập trạng thái giả của một đối tượng thực trước khi các UT có liên quan được đưa vào vận hành (ví dụ kết nối cơ sở dữ liệu, giả định trạng thái lỗi server, ...)
- Khi cần lập trạng thái cần thiết cho một số tính chất nào đó của đối tượng đã bị khoá quyền truy cập (các biến, thủ tục, hàm, thuộc tính riêng được khai báo private).

# Phạm vi sử dụng của Mock Object

---

MO được sử dụng trong các trường hợp sau:

- Khi cần kiểm tra một số thủ tục hoặc các biến của thành viên bị hạn chế truy cập.
- Khi cần loại bỏ các hiệu ứng phụ của một đối tượng nào đó không liên quan đến UT
- Khi cần kiểm nghiệm mã vận hành có tương tác với hệ thống bên ngoài

# Thiết kế Mock Object

Việc thiết kế MO gồm 3 bước chính sau đây:

- Đưa ra interface để mô tả đối tượng. Tất cả các tính chất và thủ tục quan trọng cần kiểm tra phải được mô tả trong interface
- Viết nội dung cho đối tượng thực dựa trên interface như thông thường
- Trích interface từ đối tượng thực và triển khai MO dựa trên interface đó

**Chú ý:** Một cách làm khác là kế thừa một đối tượng đang tồn tại và cố gắng mô phỏng các hành vi càng đơn giản càng tốt, như trả về một dữ liệu giả chẳng hạn.



# Unit Test trong Java

- JUnit là một framework được sử dụng cho mục đích kiểm tra nguồn mở để kiểm thử đơn vị dành cho ngôn ngữ lập trình Java. Nó đóng một vai trò quan trọng trong sự phát triển theo hướng kiểm thử.
- JUnit là một “thành viên” của gia đình khung kiểm thử cho kiểm thử đơn vị xUnit.
- Ý tưởng của JUnit là “test trước code sau”. Cách tiếp cận này giống như “test một ít, code một ít, rồi lại test một ít và code một ít” → giúp cải thiện năng suất của lập trình viên và sự ổn định của các mã lập trình

# JUnit là gì?

---

- **JUnit là một framework được sử dụng cho mục đích kiểm thử đơn vị (unit testing) cho ngôn ngữ lập trình Java.** Nó đóng một vai trò quan trọng trong phát triển phần mềm dựa trên kiểm thử, và là một trong những framework kiểm thử đơn vị được gọi chung là xUnit.

# Các tính năng của JUnit

---

- JUnit là một framework mã nguồn mở, được sử dụng để viết và chạy kiểm thử.
- Cung cấp các annotation để định nghĩa các phương thức kiểm thử.
- Cung cấp các assertions để kiểm tra kết quả mong đợi.
- Cung cấp các test runners để thực thi các test scripts.
- JUnit cho phép bạn viết code nhanh hơn.

# Các tính năng của JUnit

---

- JUnit là đơn giản để sử dụng.
- Kiểm thử JUnit có thể được chạy tự động.
- Kiểm thử JUnit có thể được tổ chức thành các test suites có chứa các test cases.
- JUnit hiển thị tiến trình kiểm thử trong thanh màu xanh nếu nếu kiểm thử không có lỗi, và nó chuyển sang màu đỏ khi có bất kỳ một kiểm thử nào bị lỗi.

# Các điểm đặc biệt của JUnit

---

- a. **Fixture:** Mục đích của fixture là đảm bảo rằng có một môi trường phổ biến và cố định để chạy test sao cho các kết quả lặp lại. Fixture bao gồm:
  - setUp() method: chạy trước khi gọi kiểm tra
  - tearDown() method: chạy sau mỗi method kiểm thử
- b. **Test Suite** (bộ kiểm thử): Test Suite bao gồm một hệ thống các test case (trường hợp kiểm thử) và chạy chúng cùng một lúc
- c. **Test Runner** (Trình chạy kiểm thử): Dùng để chạy các test case

## Các điểm đặc biệt của JUnit

---

d. **JUnit Class** (Lớp JUnit): Lớp JUnit là những lớp quan trọng, dùng để viết và chạy kiểm thử. Một số lớp quan trọng là:

Assert: Bao gồm hệ thống các method assert

Test Case: Bao gồm một test case để định nghĩa fixture chạy nhiều lần

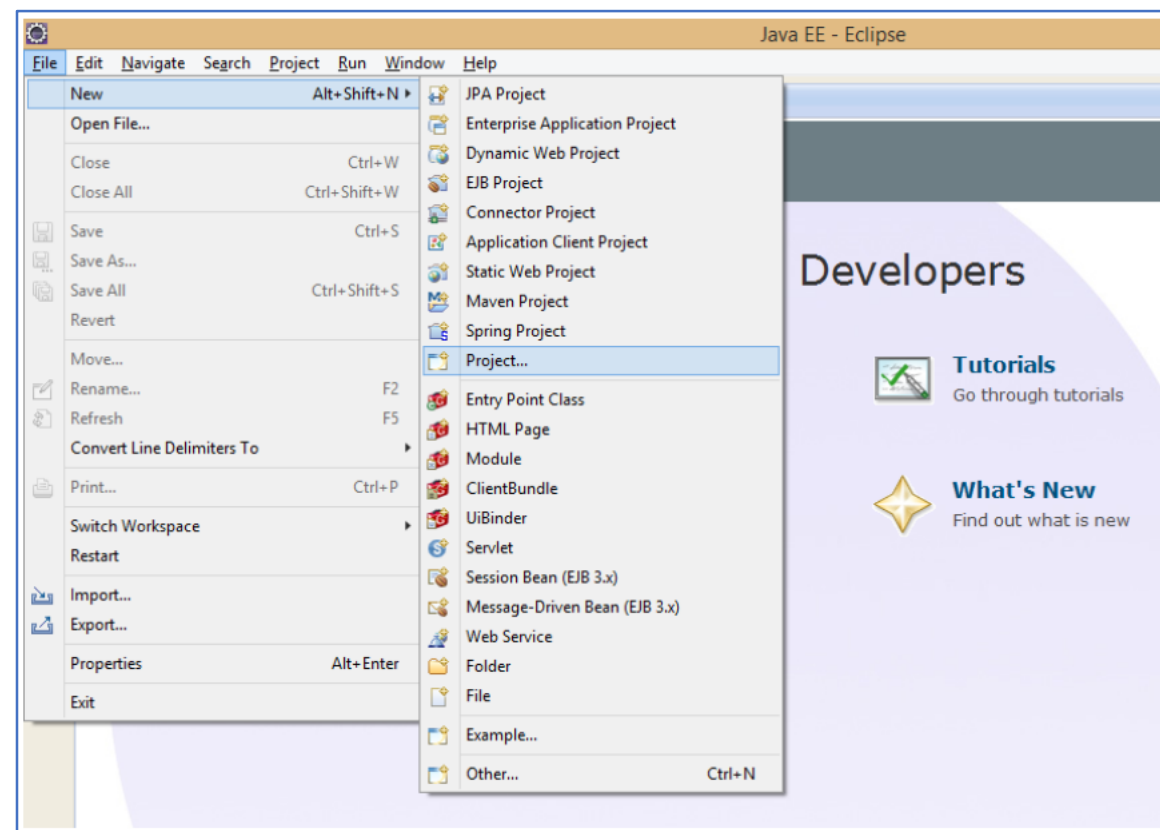
Test Result: Bao gồm các method dùng để thu thập kết quả từ việc chạy một test case

# Ví dụ JUnit trên Eclipse

- Cấu hình sử dụng;
  - Eclipse Oxygen 4.7
  - JDK 1.8
  - JUnit 4.12

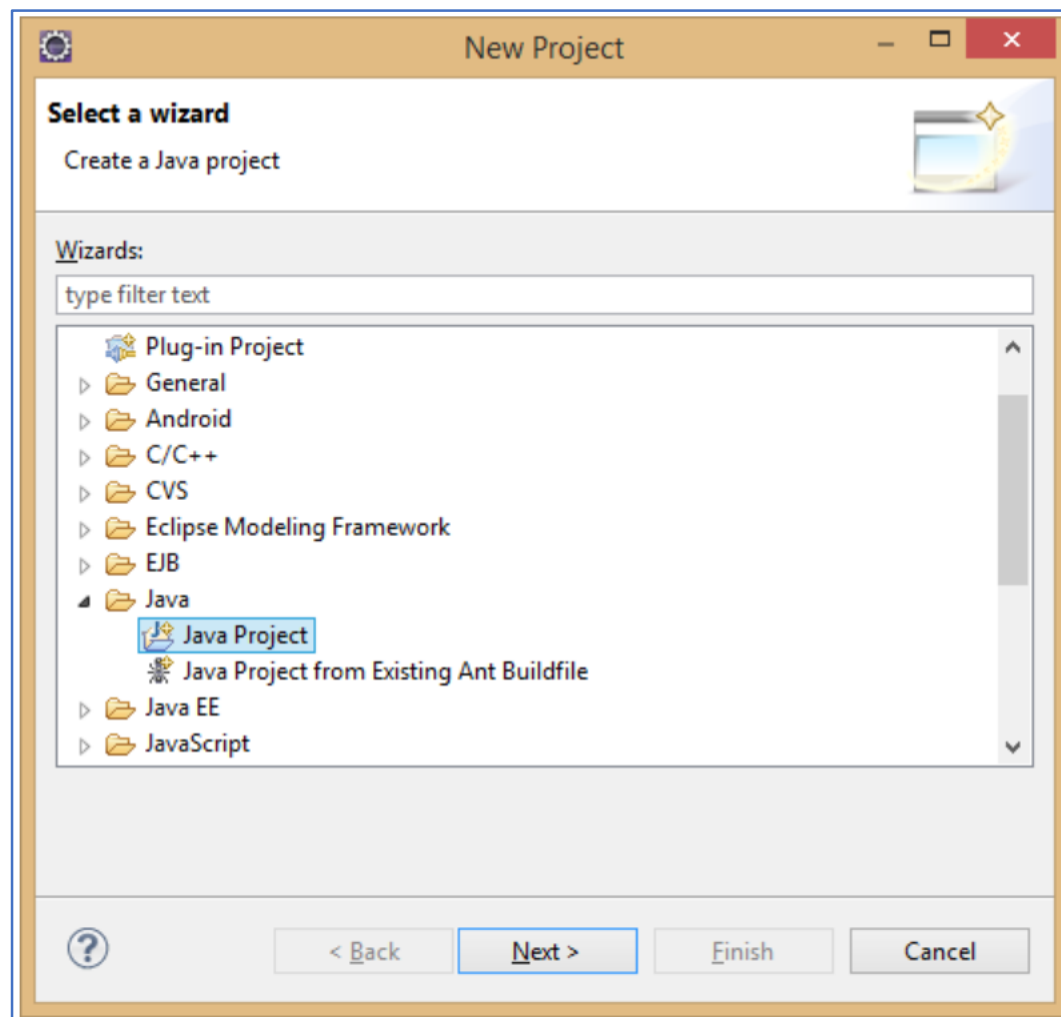
- Tao Project

**Step 1:** Menu File => New => Project...



# Ví dụ JUnit trên Eclipse

**Step 2:** Chọn Java Project trong hộp thoại New Project và nhấn Next

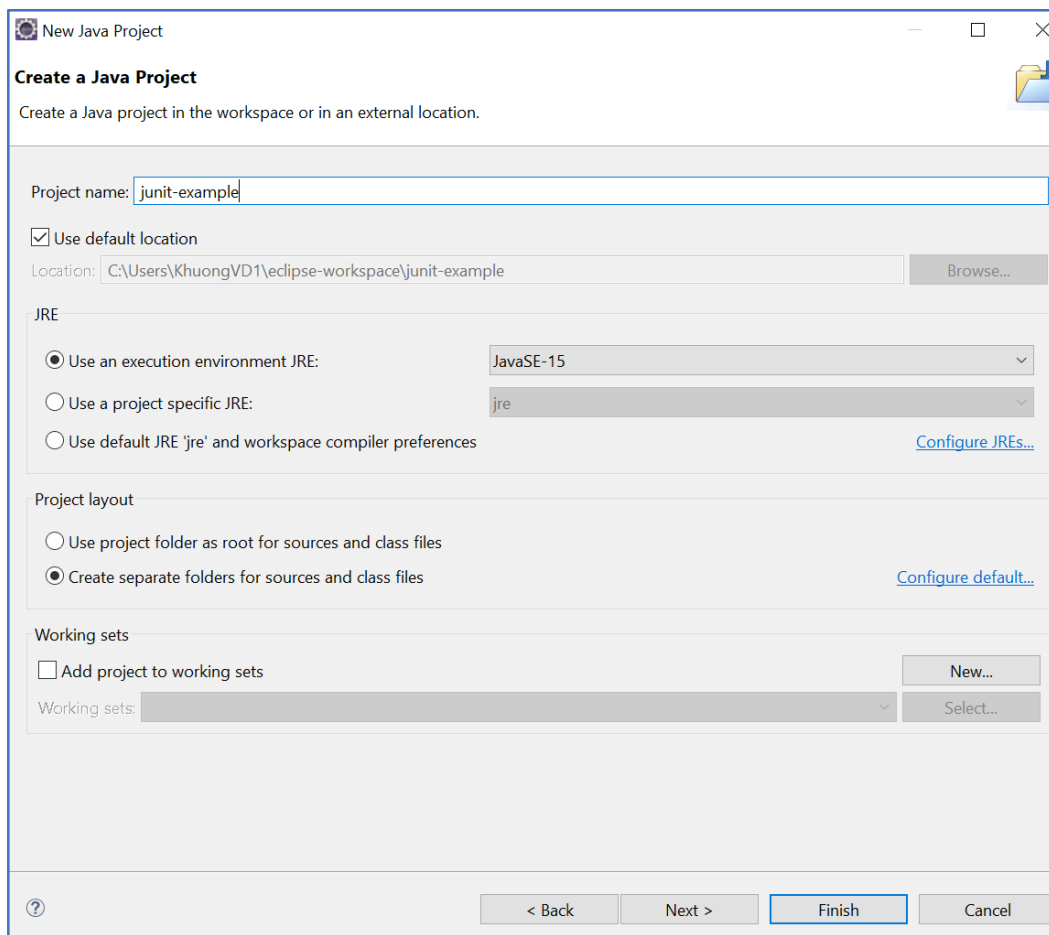




# Ví dụ JUnit trên Eclipse

## Step 3: Trong hộp thoại New Java Project điền thông tin và chọn Next

- Project name: tên của project
- Chọn jre: phiên bản của thư viện và máy ảo java



**New Java Project**

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location:  [Browse...](#)

JRE

☒ Use an execution environment JRE:  [Configure JREs...](#)

☐ Use a project specific JRE:

☐ Use default JRE 'jre' and workspace compiler preferences

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

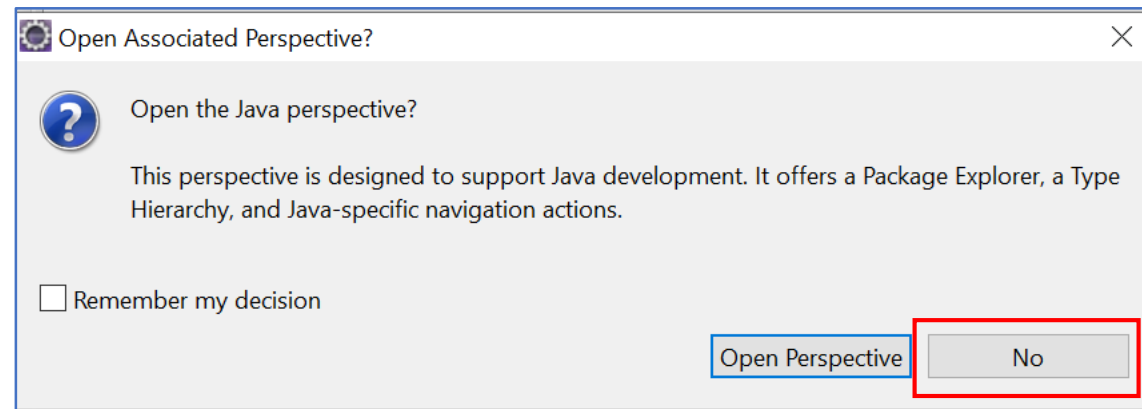
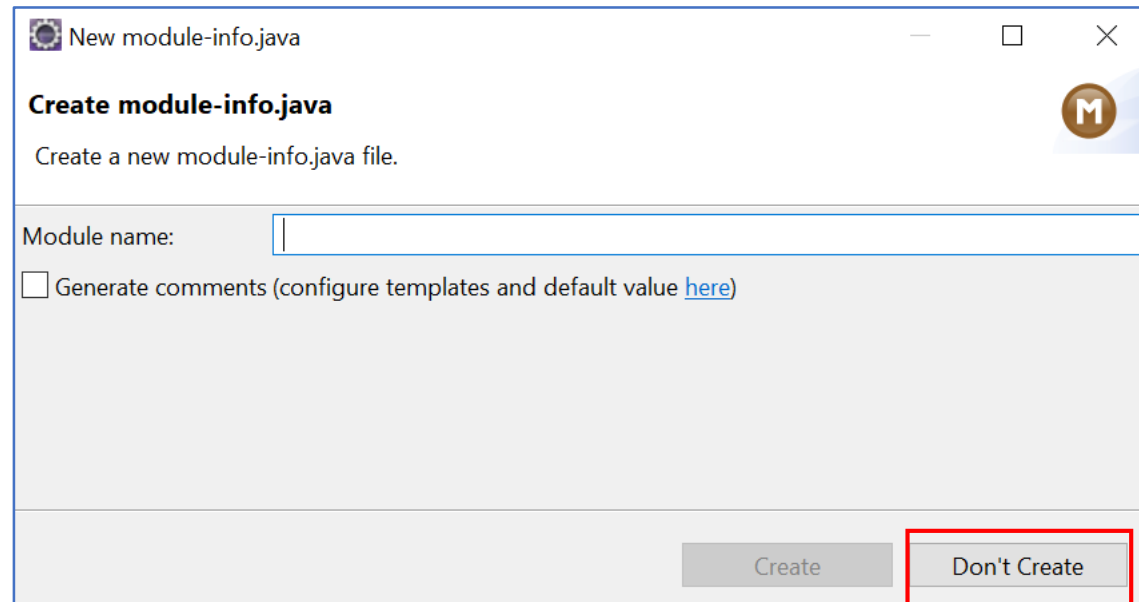
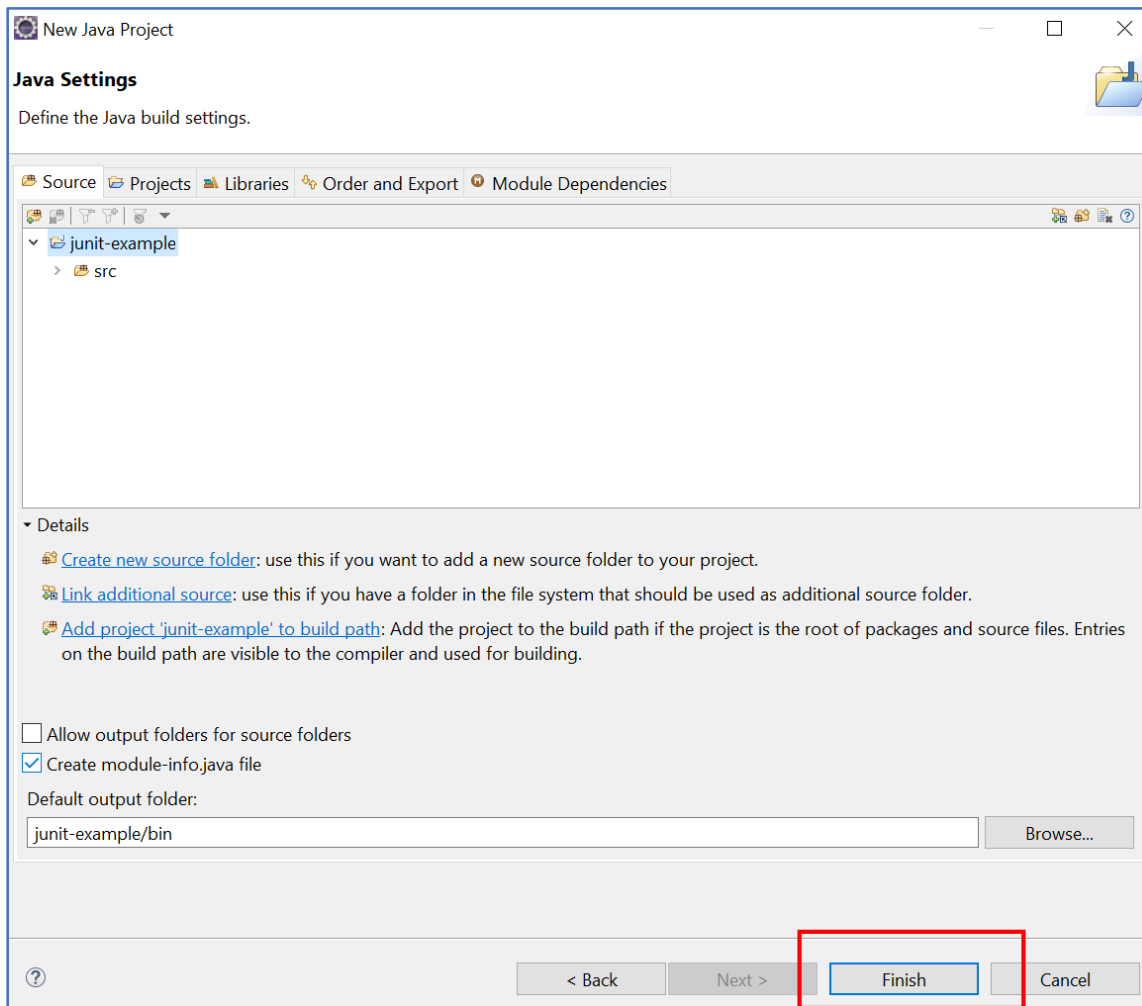
☐ Add project to working sets [New...](#)

Working sets:  [Select...](#)

[? < Back](#) [Next >](#) [Finish](#) [Cancel](#)

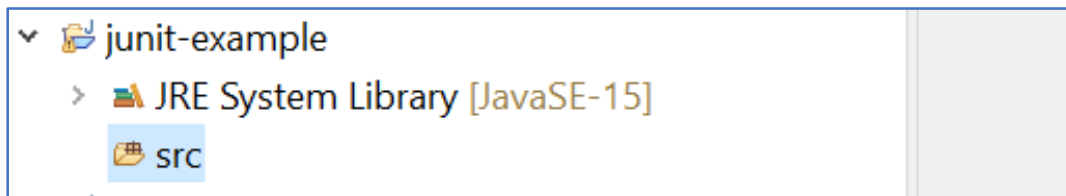
# Ví dụ JUnit trên Eclipse

## Step 4: Cấu hình thư viện (nếu không có thì để mặc định) và nhấn Finish



# Ví dụ JUnit trên Eclipse

## Step 5: Kết quả



Tạo lớp trong java (PhanSo)

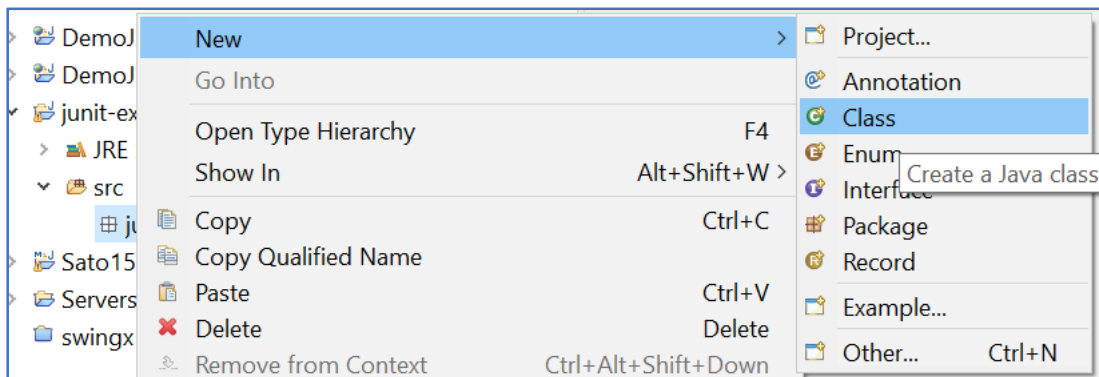
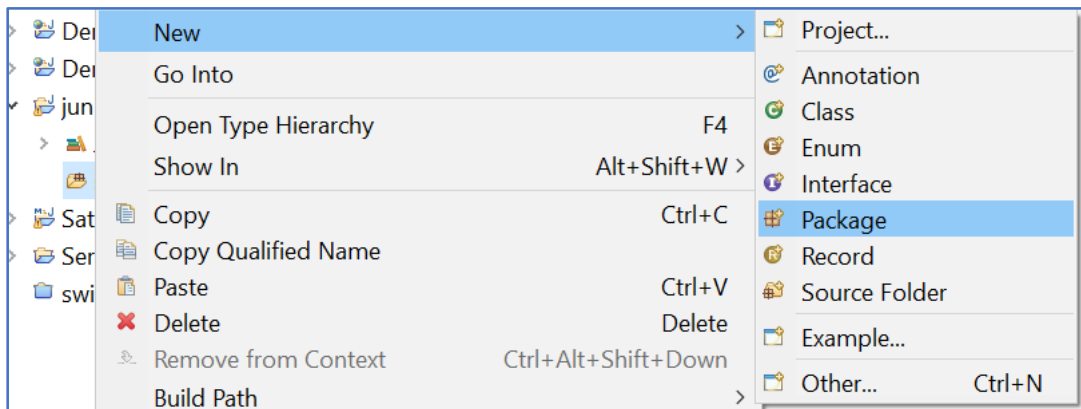
### Step 5.1:

Click chuột phải vào thư mục src => chọn New => chọn Package

Click chuột phải vào thư mục src => chọn New => chọn Class

# Ví dụ JUnit trên Eclipse

## Step 5.1:



# Ví dụ JUnit trên Eclipse

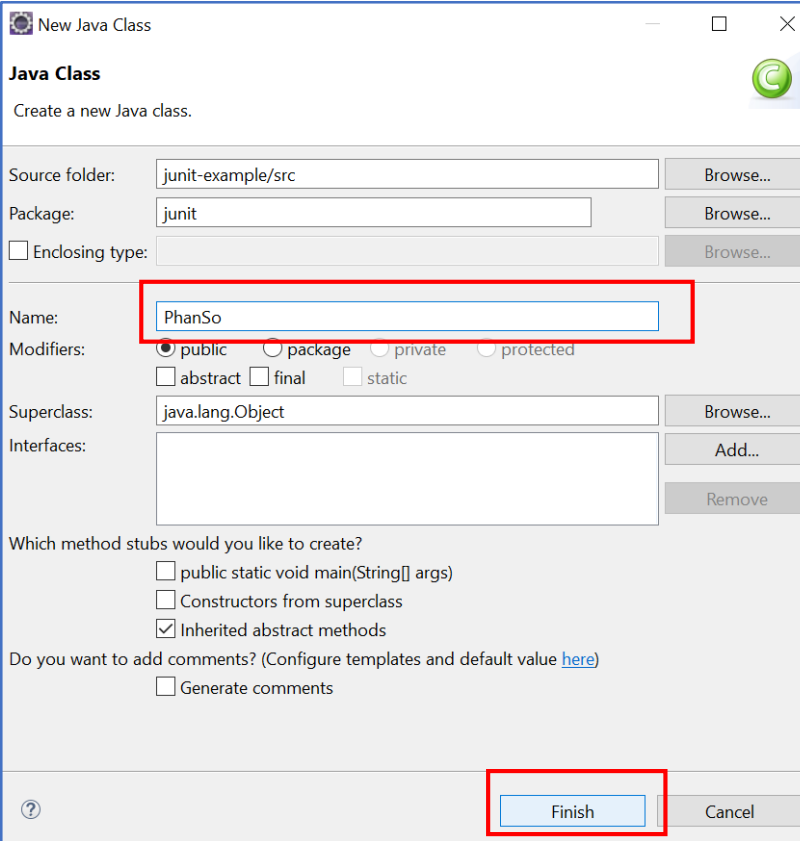
## Step 5.2:

Trong hộp thoại New Java Class điền các giá trị sau và chọn FinishPackage:  
demo => phân nhóm các lớp đối tượng

Name: PhanSo

SuperClass: lớp cha (nếu có)

Interfaces: các interface cần thực thi  
(nếu có)

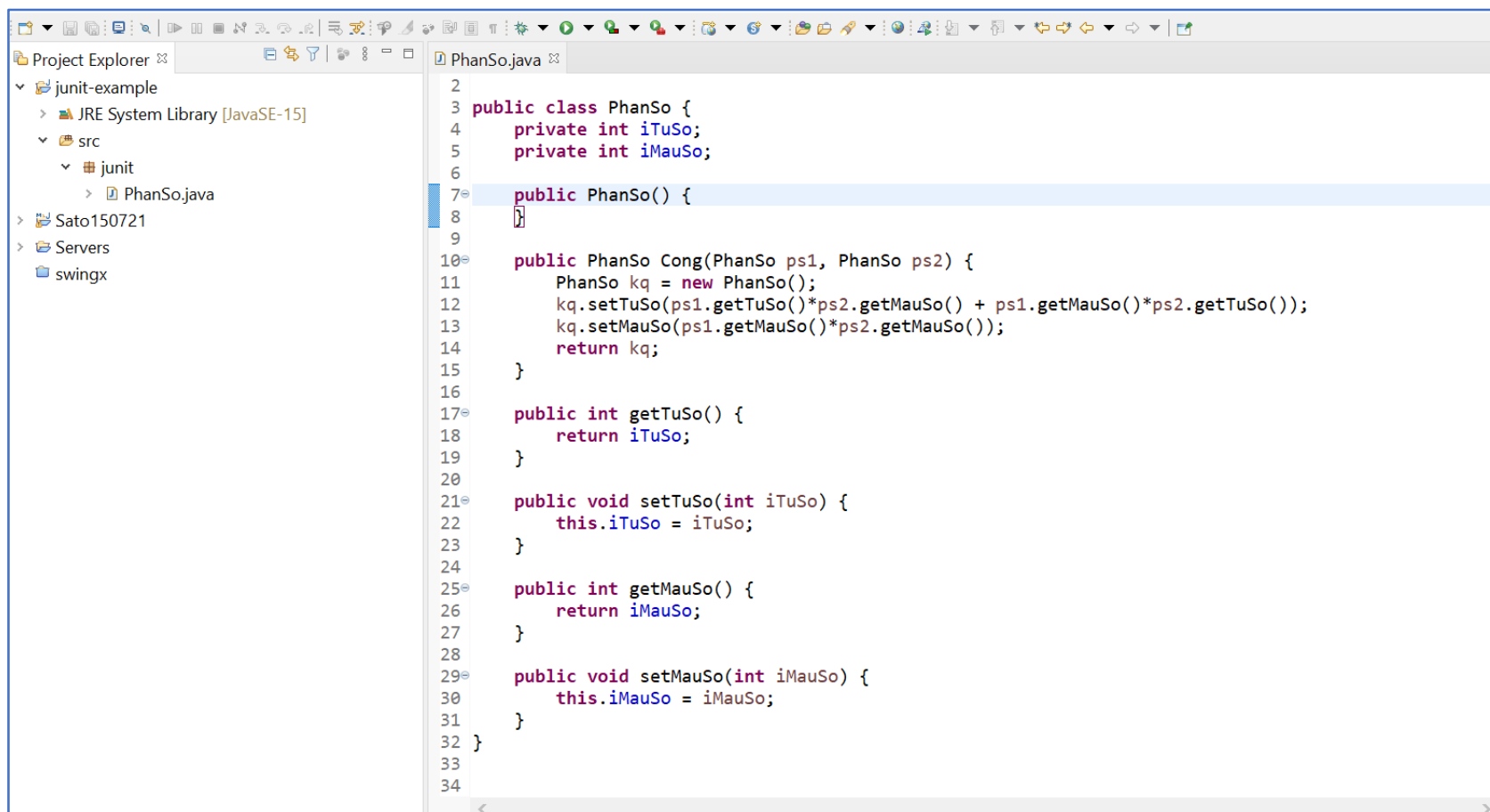


The screenshot shows the 'New Java Class' dialog box in Eclipse. The 'Name' field is highlighted with a red box and contains the text 'PhanSo'. The 'Superclass' field contains 'java.lang.Object'. The 'Finish' button at the bottom right is also highlighted with a red box. The dialog includes fields for 'Source folder', 'Package', and 'Enclosing type', all with 'Browse...' buttons. It also has checkboxes for 'public', 'package', 'private', 'protected', 'abstract', 'final', and 'static' under 'Modifiers'. At the bottom, there are checkboxes for 'Which method stubs would you like to create?' (public static void main, Constructors from superclass, Inherited abstract methods) and 'Do you want to add comments?' (Generate comments).

# Ví dụ JUnit trên Eclipse

## Step 5.3:

Trong lớp PhanSo, định nghĩa các thuộc tính, hàm khởi tạo mặc định và hàm cộng



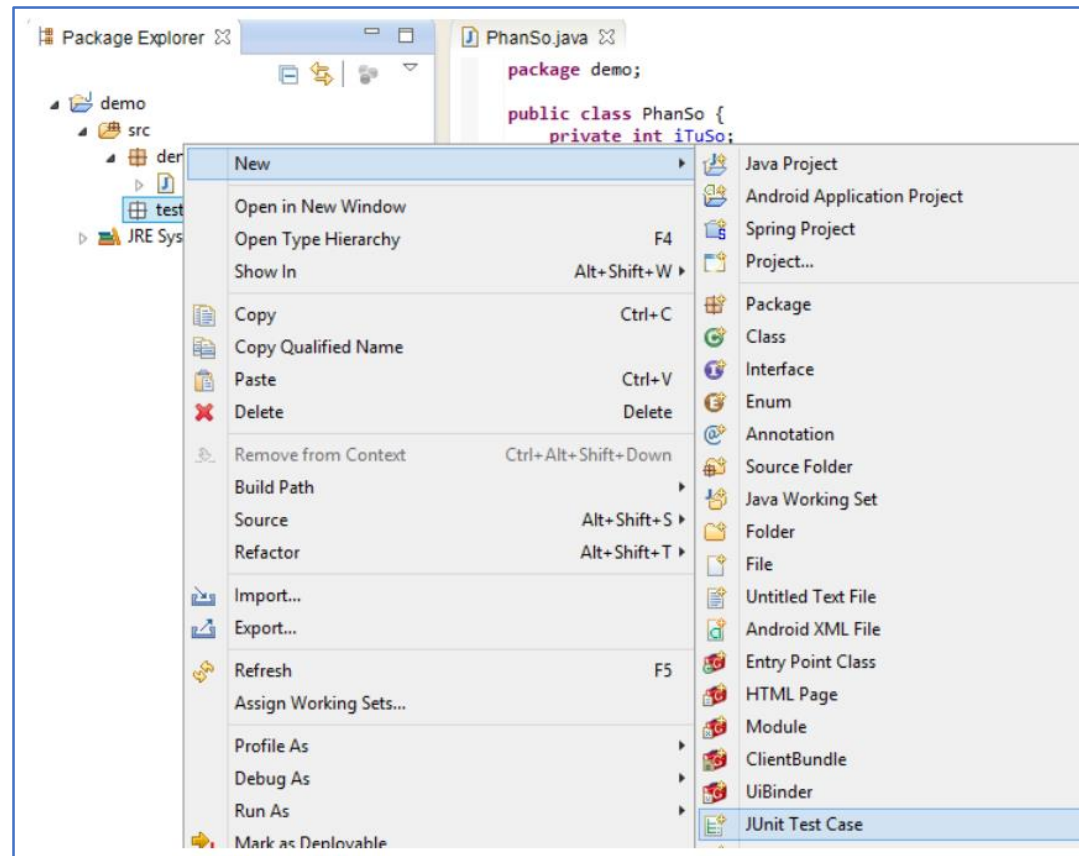
```
1
2
3 public class PhanSo {
4     private int iTuSo;
5     private int iMauSo;
6
7     public PhanSo() {
8
9
10
11     public PhanSo Cong(PhanSo ps1, PhanSo ps2) {
12         PhanSo kq = new PhanSo();
13         kq.setTuSo(ps1.getTuSo()*ps2.getMauSo() + ps1.getMauSo()*ps2.getTuSo());
14         kq.setMauSo(ps1.getMauSo()*ps2.getMauSo());
15         return kq;
16     }
17
18     public int getTuSo() {
19         return iTuSo;
20     }
21
22     public void setTuSo(int iTuSo) {
23         this.iTuSo = iTuSo;
24     }
25
26     public int getMauSo() {
27         return iMauSo;
28     }
29
30     public void setMauSo(int iMauSo) {
31         this.iMauSo = iMauSo;
32     }
33 }
34
```

# Ví dụ JUnit trên Eclipse

Tạo lớp PhanSoTest trong test package

## Step 5.4:

Chuột phải vào Project => chọn New => chọn Junit Test case



# Ví dụ JUnit trên Eclipse

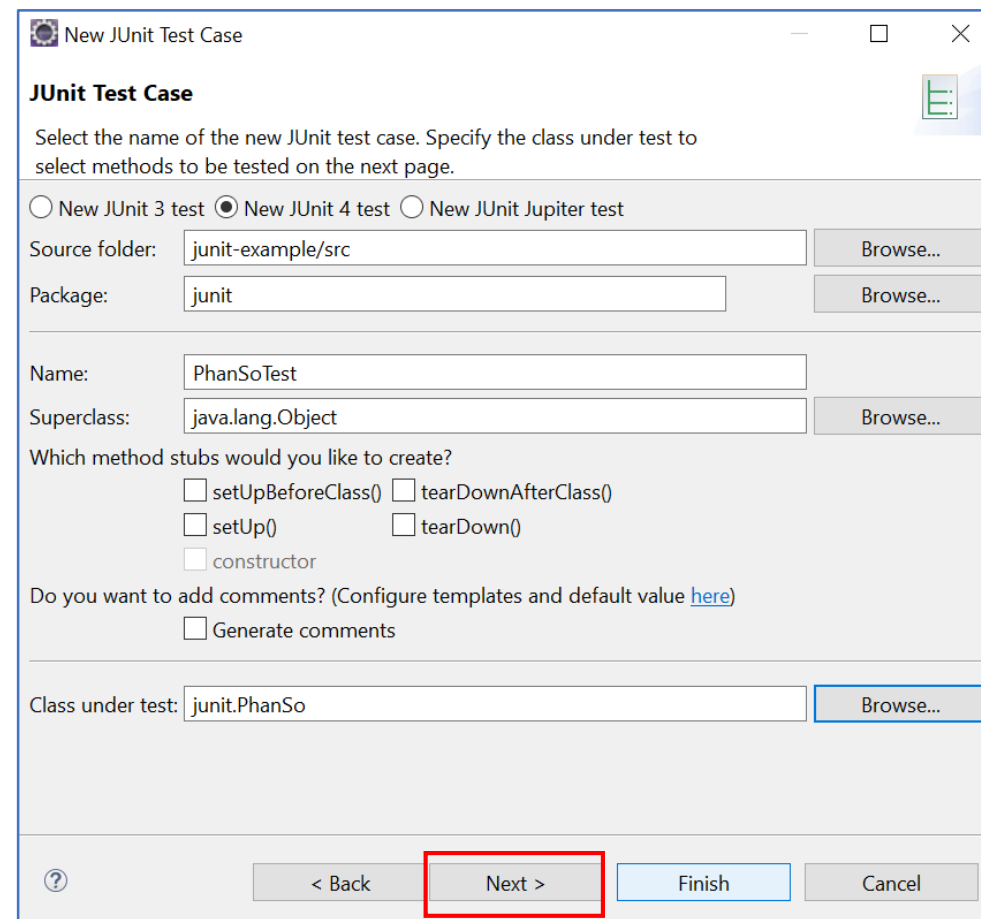
## Step 5.5:

Trong hộp thoại New JUnit Test case chú ý các thuộc tính sau và chọn

NextPackage: tên package

Name: tên lớp

Class under Test: tên lớp được test

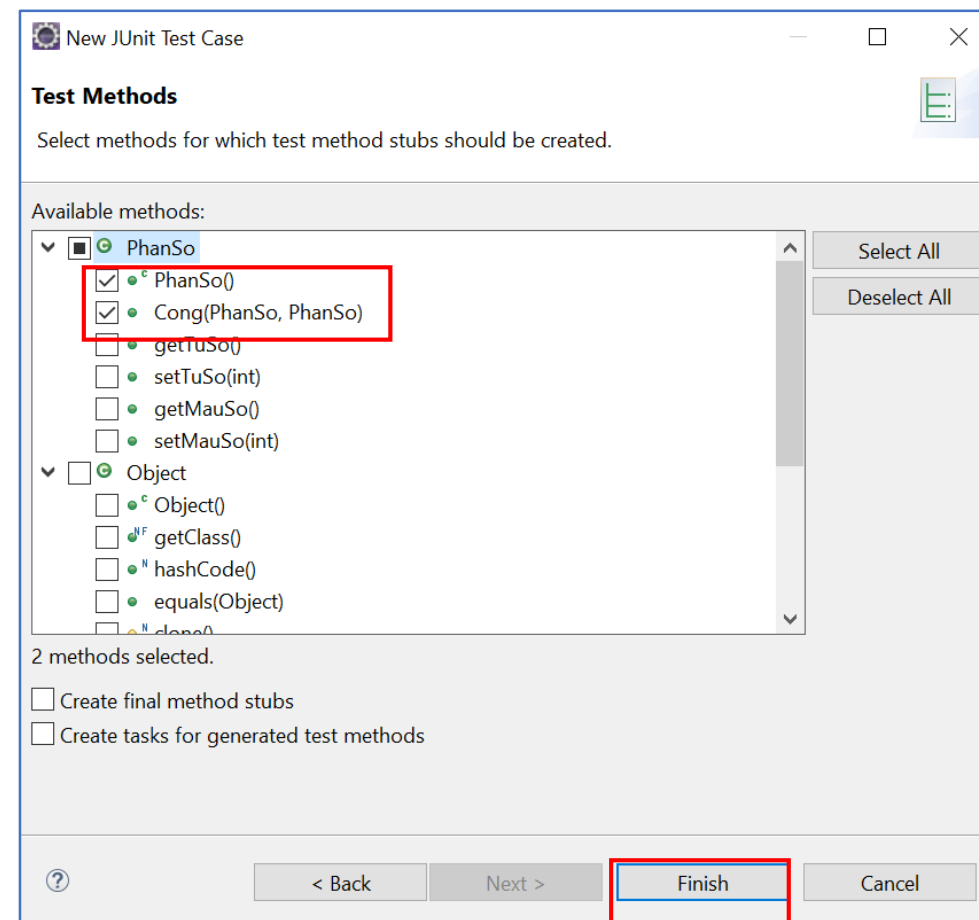




# Ví dụ JUnit trên Eclipse

## Step 5.6:

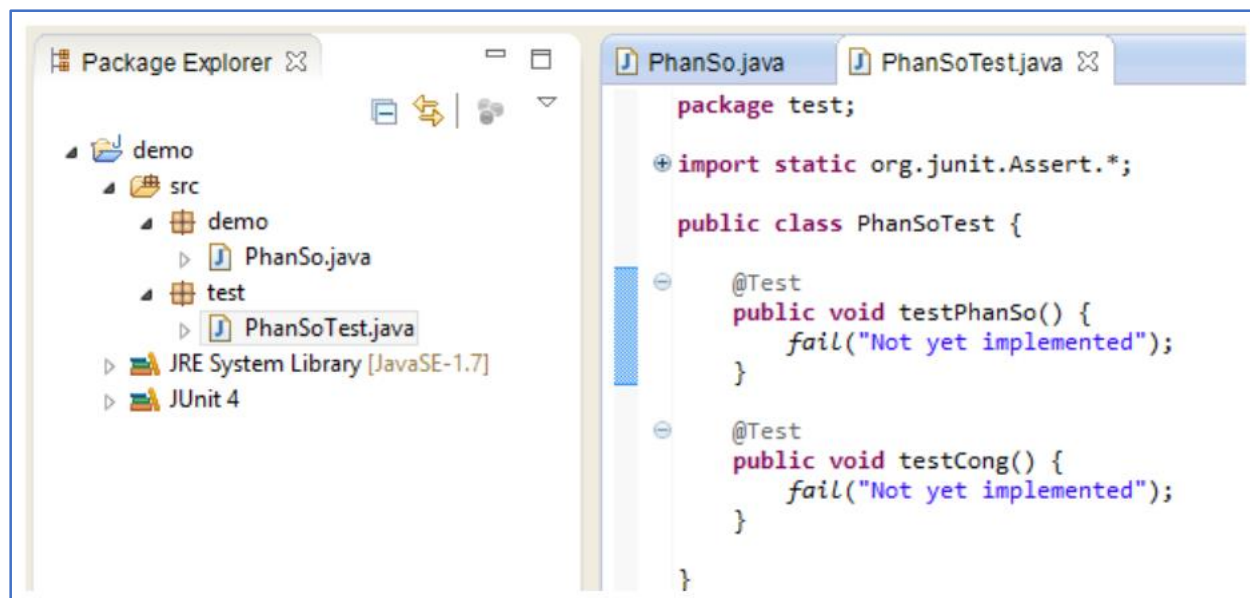
Muốn test hàm/phương thức nào thì chọn và nhấn Finish



# Ví dụ JUnit trên Eclipse

## Step 5.7:

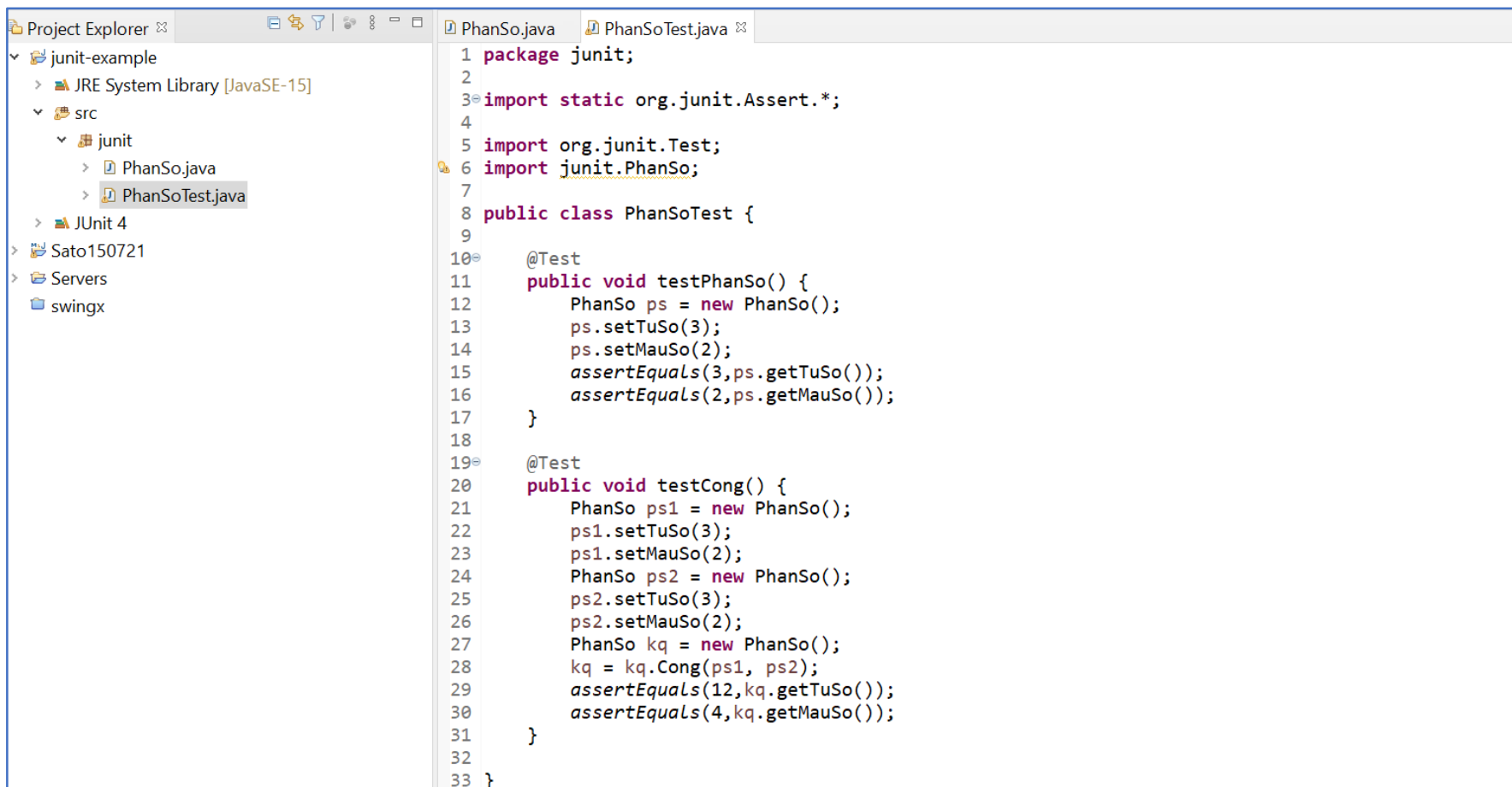
## Kết quả



# Ví dụ JUnit trên Eclipse

## Step 5.8:

Sử dụng các hàm AssertXXX để so sánh kết quả



```

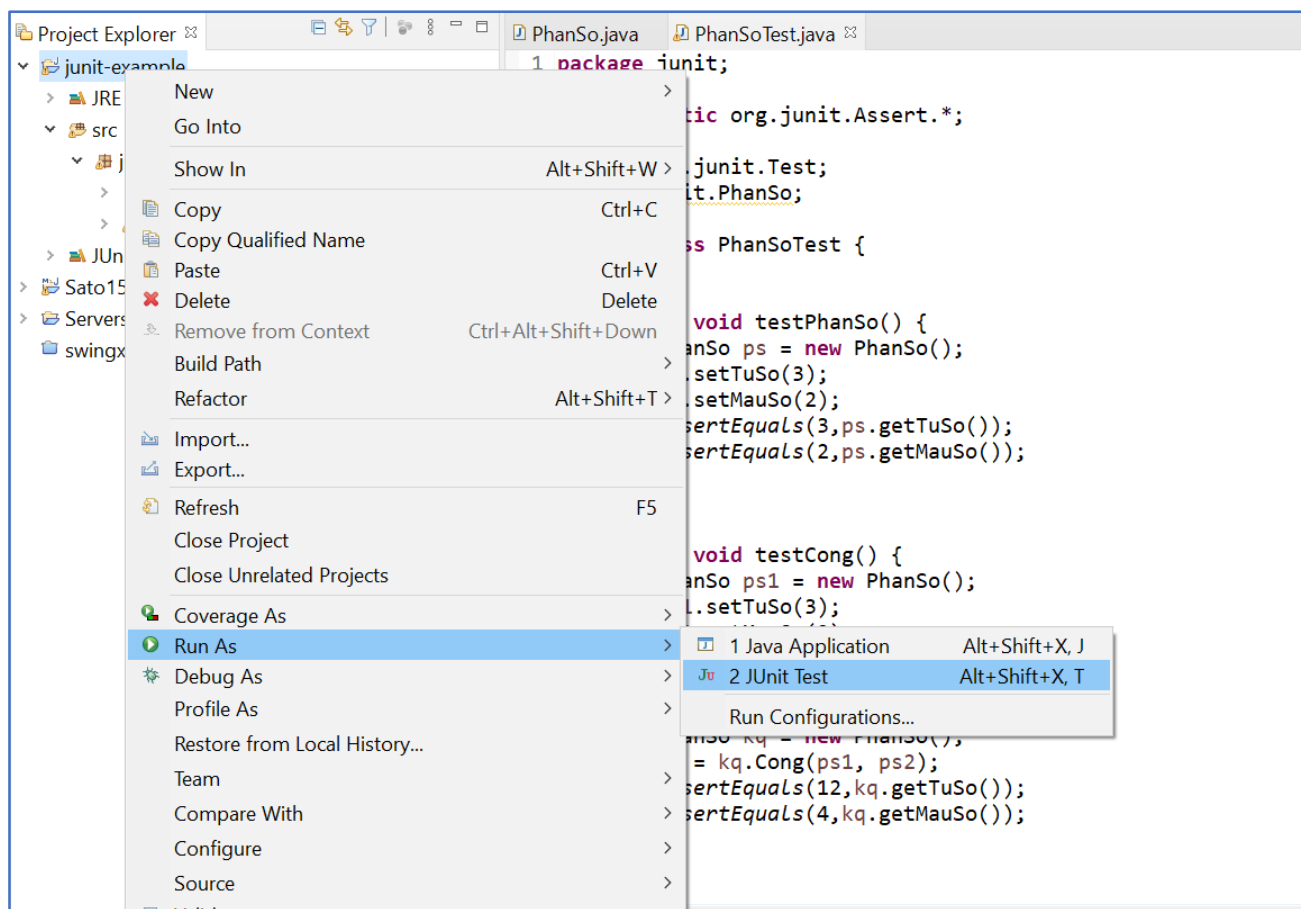
1 package junit;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Test;
6 import junit.PhanSo;
7
8 public class PhanSoTest {
9
10     @Test
11     public void testPhanSo() {
12         PhanSo ps = new PhanSo();
13         ps.setTuSo(3);
14         ps.setMauSo(2);
15         assertEquals(3, ps.getTuSo());
16         assertEquals(2, ps.getMauSo());
17     }
18
19     @Test
20     public void testCong() {
21         PhanSo ps1 = new PhanSo();
22         ps1.setTuSo(3);
23         ps1.setMauSo(2);
24         PhanSo ps2 = new PhanSo();
25         ps2.setTuSo(3);
26         ps2.setMauSo(2);
27         PhanSo kq = new PhanSo();
28         kq = kq.Cong(ps1, ps2);
29         assertEquals(12, kq.getTuSo());
30         assertEquals(4, kq.getMauSo());
31     }
32
33 }

```

# Ví dụ JUnit trên Eclipse

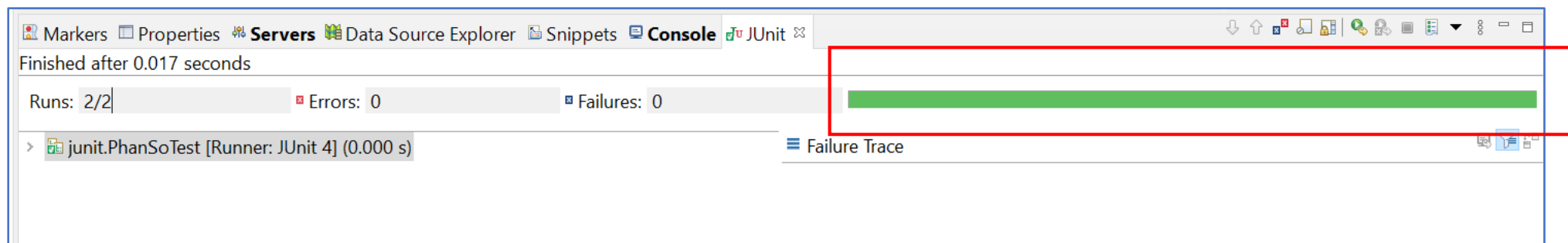
## Step 5.9:

Chạy Junit bằng cách click chuột phải vào project => chọn Run as => chọn Junit



# Ví dụ JUnit trên Eclipse

## Step 5.10: Kết quả chạy Junit

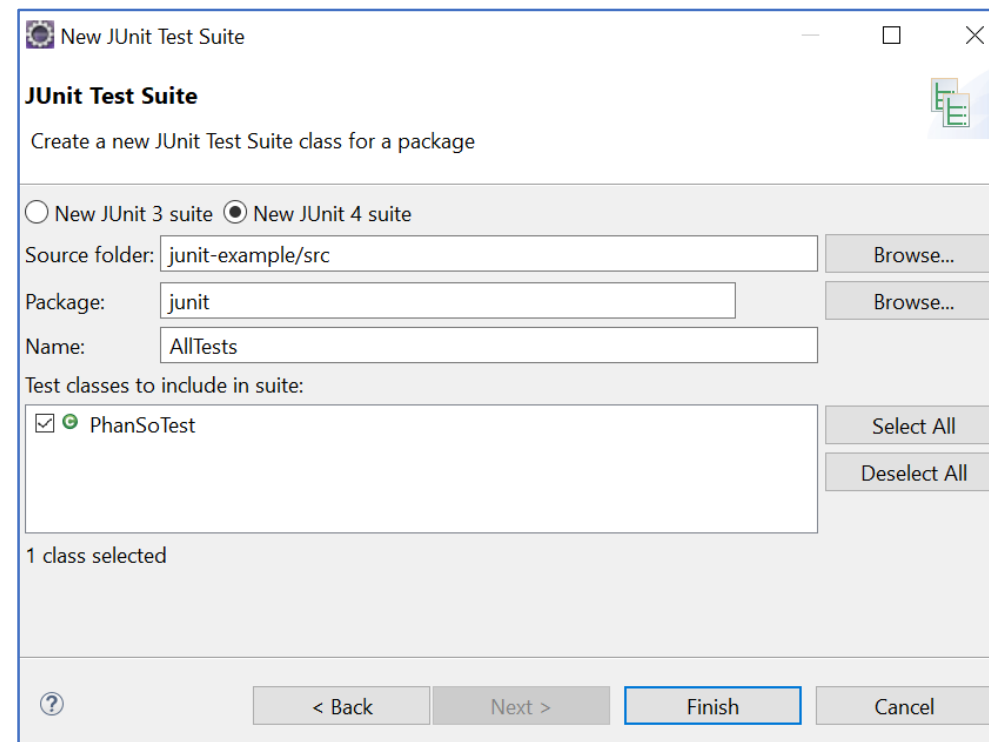
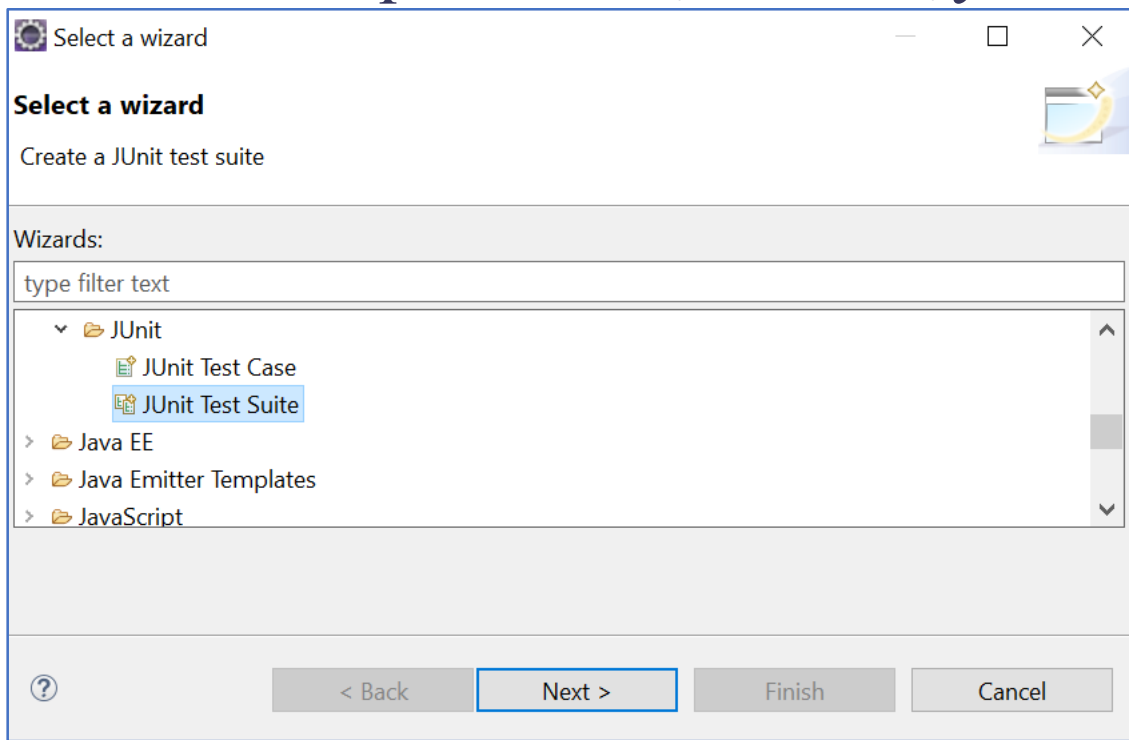


# Ví dụ JUnit trên Eclipse

## Step 6:

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 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.



## Ví dụ JUnit trên Eclipse

Step 6: Để tạo test suite ta sử dụng:

**@RunWith(Suite.class)**

**@SuiteClasses({ PhanSoTest.class })**

Bên trong @SuiteClasses sẽ là các class test được chạy.

# Ví dụ JUnit trên Eclipse

## Step 6:

### Ví dụ: TestSuite

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class Test1 {
    @Test
    public void test1() {
        assertEquals("hello", "hello");
    }
}
```

```
import static org.junit.Assert.assertTrue;

import org.junit.Test;

public class Test2 {
    @Test
    public void test1() {
        assertTrue(true);
    }
}
```

Tạo bộ test gồm 2 class test là Test1.java và Test2.java

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({Test1.class, Test2.class})
public class MyTestSuite {

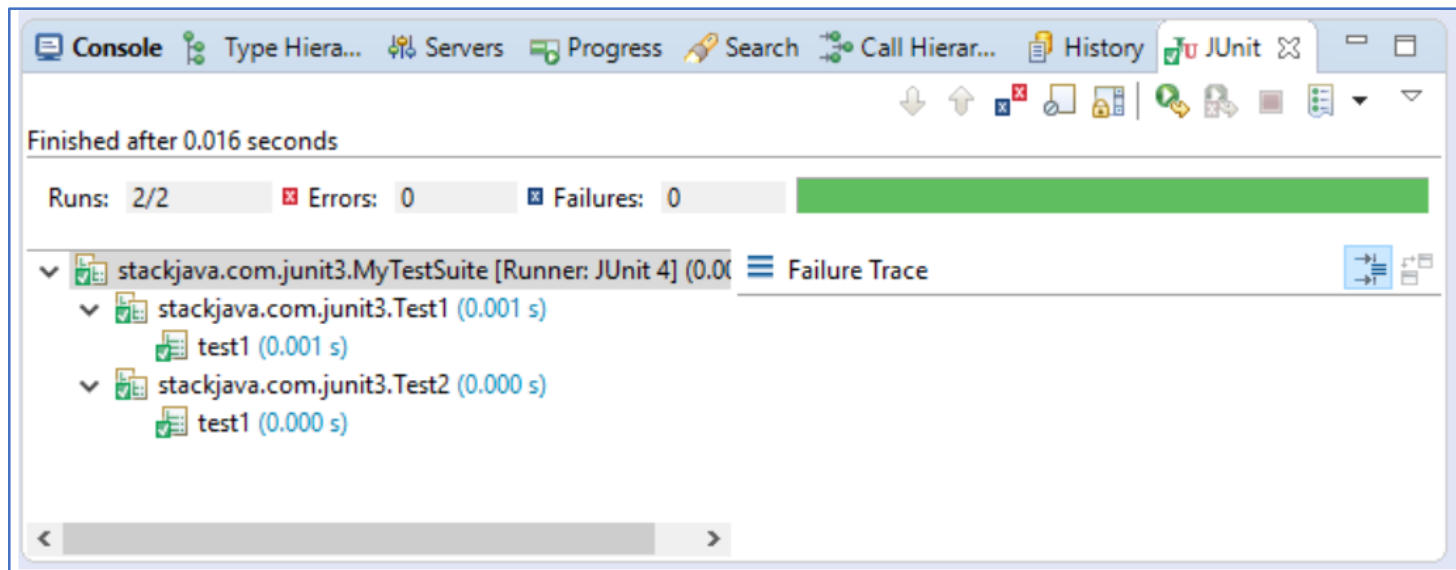
}
```



# Ví dụ JUnit trên Eclipse

## Step 6:

Kết quả: (Chuột phải vào class MyTestSuite.java và chọn Run As JUnit Test)

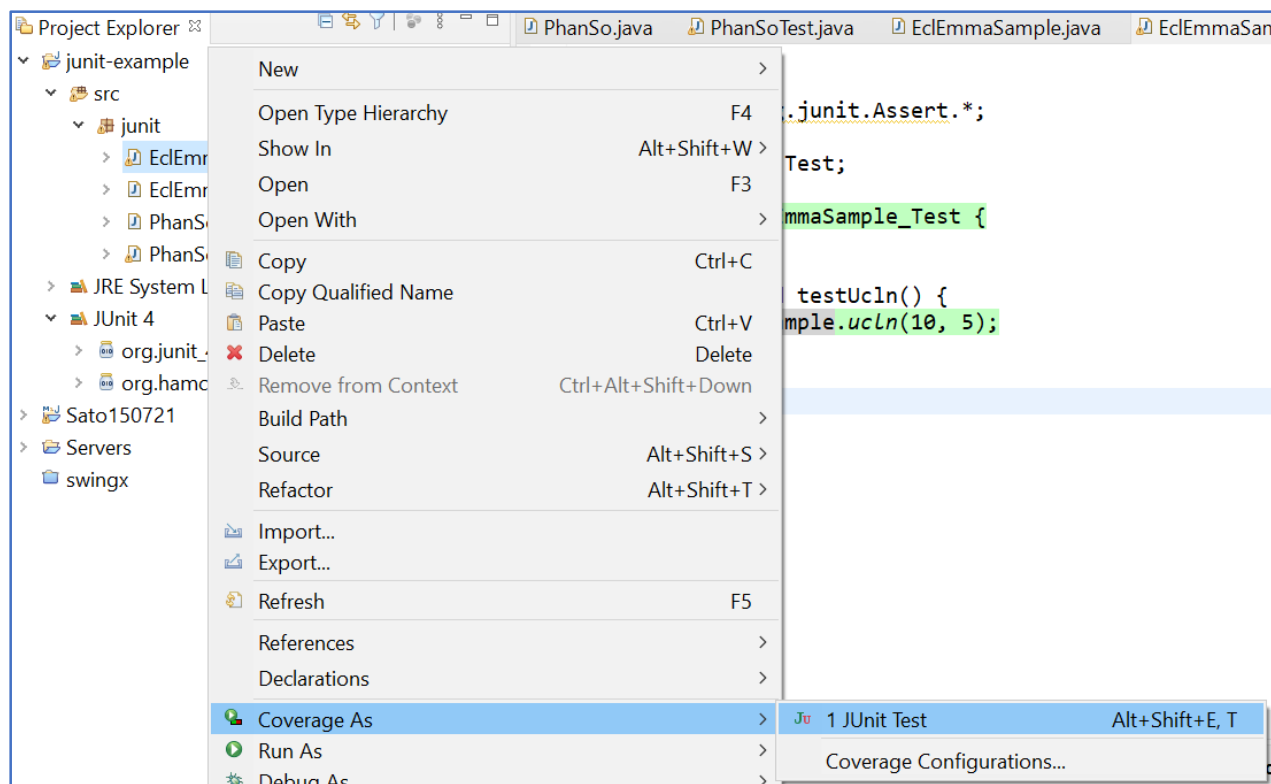


# Ví dụ JUnit trên Eclipse

## Step 7: Kiểm tra độ bao phủ của Unit Test

Nhấn chuột phải lên project hoặc package hoặc class JUnit Test bất kỳ → chọn **Coverage As** → chọn **JUnit**

Sau khi chạy ta được kết quả như sau:



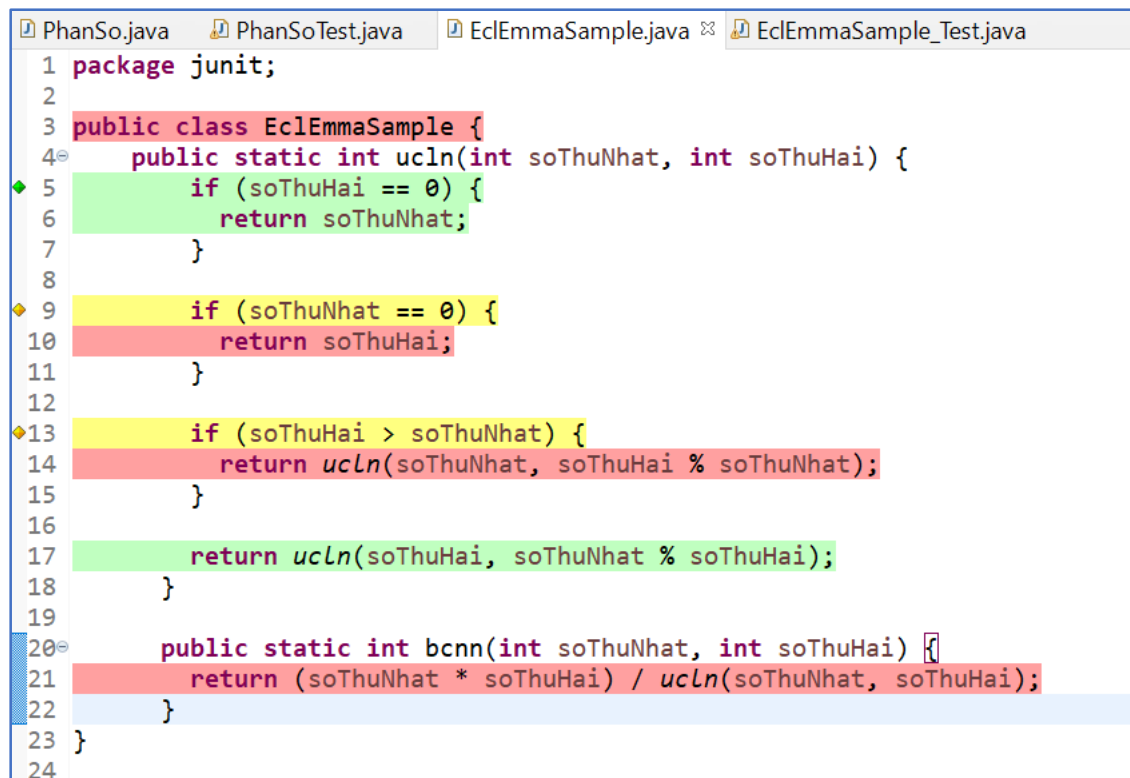
```

1 package junit;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Test;
6
7 public class EclEmmaSample_Test {
8
9     @Test
10     public void testUcln() {
11         EclEmmaSample.ucln(10, 5);
12     }
13 }
14

```

# Ví dụ JUnit trên Eclipse

**Step 7:** Kiểm tra độ bao phủ của Unit Test  
Sau khi chạy ta được kết quả như sau:



```

1 package junit;
2
3 public class EclEmmaSample {
4     public static int ucln(int soThuNhat, int soThuHai) {
5         if (soThuHai == 0) {
6             return soThuNhat;
7         }
8
9         if (soThuNhat == 0) {
10            return soThuHai;
11        }
12
13        if (soThuHai > soThuNhat) {
14            return ucln(soThuNhat, soThuHai % soThuNhat);
15        }
16
17        return ucln(soThuHai, soThuNhat % soThuHai);
18    }
19
20    public static int bcnn(int soThuNhat, int soThuHai) {
21        return (soThuNhat * soThuHai) / ucln(soThuNhat, soThuHai);
22    }
23 }
24

```







## Chú ý:

- **Màu đỏ:** code JUnit chưa bao phủ (chưa kiểm tra) được dòng code này.
- **Màu vàng:** chưa bao phủ hết trường hợp. Như ví dụ trên: cần bổ sung test trường hợp  $soThuNhat = 0$  và  $soThuHai > soThuNhat$
- **Màu xanh:** đã được kiểm tra.

# Ví dụ JUnit trên Eclipse

## Step 8: Xem kết quả của độ bao phủ

Xem kết quả báo cáo về độ bao phủ của code JUnit Test: thông thường sau khi chạy JUnit, cửa sổ Coverage sẽ được mở. Nếu không thấy, ta có thể check: vào menu Window -> Show View -> Other... -> Coverage

EclEmmaSample_Test (Jul 25, 2021 11:58:48 AM)					
Element	Coverage	Covered Ins...	Missed Instr...	Total Instruc...	
▼ src	 15.4 %	23	126	149	
▼ junit	 15.4 %	23	126	149	
> PhanSoTest.java	 0.0 %	0	64	64	
> PhanSo.java	 0.0 %	0	43	43	
> EclEmmaSample.java	 44.1 %	15	19	34	
> EclEmmaSample_Test.j	 100.0 %	8	0	8	

# Các tính năng của JUnit Test Framework

---

- **JUnit là một Framework kiểm thử hồi quy** được sử dụng bởi các nhà phát triển để thực hiện kiểm thử đơn vị trong Java và tăng tốc độ lập trình và tăng chất lượng code.
- **JUnit Test Framework** cung cấp các tính năng quan trọng sau đây:
  - Fixture
  - Test Suite
  - Test Runner
  - Các lớp JUnit

# Fixture trong JUnit Test Framework

- **Fixture** là một trạng thái cố định của một tập các đối tượng được sử dụng cho một phiên bản baseline cho việc chạy các test. Mục đích của cố định kiểm thử là để đảm bảo rằng có một môi trường cố định trong đó các kiểm thử được thực hiện sao cho kết quả có thể lặp lại. Nó bao gồm:
  - **setUp()**, phương thức này được thực thi trước phương thức khi kiểm thử.
  - **tearDown()**, phương thức này được thực thi sau phương thức kiểm thử.

# Fixture trong JUnit Test Framework

## ■ Ví dụ:

```
1  import junit.framework.TestCase;
2
3  public class JavaTest extends TestCase {
4      protected int value1;
5      protected int value2;
6
7      /**
8       * Khởi tạo các giá trị
9       */
10     protected void setUp() {
11         value1 = 3;
12         value2 = 3;
13     }
14
15     /**
16      * test Add method
17      */
18     public void testAdd() {
19         double result = value1 + value2;
20         assertTrue(result == 6);
21     }
22 }
```

# Test Suite trong JUnit Test Framework

- **Test Suite** được sử dụng để nhóm nhiều kiểm thử đơn vị và chạy chúng với nhau. Trong JUnit, ghi chú `@RunWith` và `@Suite` được sử dụng để chạy một bộ các kiểm thử.

Dưới đây là ví dụ test suite cho các lớp kiểm thử `TestJUnit1` và `TestJUnit2`.

• File: **JUnitTestSuite.java**

```

2
3 import org.junit.runner.RunWith;
4 import org.junit.runners.Suite;
5
6 //JUnit Suite Test
7 @RunWith(Suite.class)
8
9 @Suite.SuiteClasses({
10     TestJUnit1.class,
11     TestJUnit2.class
12 })
13
14 public class JUnitTestSuite {
15 }

```

File: **TestJUnit1.java**

```

7 public class TestJUnit1 {
8
9     @Test
10     public void testAdd() {
11         Operation myUnit = new Operation();
12         int result = myUnit.add(2, 3);
13         assertEquals(result, 5);
14     }
15 }

```

File: **TestJUnit2.java**

```

7 public class TestJUnit2 {
8
9     @Test
10     public void testSubtract() {
11         Operation myUnit = new Operation();
12         int result = myUnit.subtract(6, 4);
13         assertEquals(result, 2);
14     }
15 }

```



# Test runner trong JUnit Test Framework

- **Test runner** được sử dụng để thực hiện các trường hợp kiểm thử. Dưới đây là một ví dụ:

File: **TestRunner.java**

```
3 import org.junit.runner.JUnitCore;
4 import org.junit.runner.Result;
5 import org.junit.runner.notification.Failure;
6
7 public class TestRunner {
8     public static void main(String[] args) {
9         Result result = JUnitCore.runClasses(TestOperation.class);
10
11         for (Failure failure : result.getFailures()) {
12             System.out.println(failure.toString());
13         }
14
15         System.out.println(result.wasSuccessful());
16     }
17 }
```

File: **TestOperation.java**

```
3 import static org.junit.Assert.assertEquals;
4 import org.junit.Test;
5
6 public class TestOperation {
7
8     @Test
9     public void testAdd() {
10         Operation myUnit = new Operation();
11         int result = myUnit.add(2, 3);
12         assertEquals(result, 5);
13     }
14
15     @Test
16     public void testSubtract() {
17         Operation myUnit = new Operation();
18         int result = myUnit.subtract(6, 4);
19         assertEquals(result, 2);
20     }
21 }
```

# Các lớp JUnit

- **Các lớp JUnit** là các lớp quan trọng, được sử dụng để viết và kiểm thử JUnit. Một số các lớp quan trọng là:
  - **Assert** - Chứa một tập hợp các phương thức assert .
  - **TestCase** - Chứa một trường hợp kiểm thử mà định nghĩa các đối tượng cố định để thực hiện sao cho kết quả có thể lặp lại.
  - **TestResult** - Bao gồm các phương thức để thu thập các kết quả của việc thực hiện một trường hợp kiểm thử.

# Bài tập JUnit

**Bài 1:** Viết chương trình java kiểm tra 3 cạnh có phải là tam giác không, nếu có hãy kiểm tra tam giác đều, tam cân, tam giác vuông hay tam giác thường. Hãy thực hiện Junit test cho class Triangle đó

Gợi ý: Build 1 class Triangle, có hàm constructor Triangle(int a, int b, int c) Và hàm int CheckTriangle()

Bảng quyết định:

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9	Rule 10	Rule 11
C1: $a < b + c$ ?	F	T	T	T	T	T	T	T	T	T	T
C2: $b < a + c$ ?	-	F	T	T	T	T	T	T	T	T	T
C3: $c < a + b$ ?	-	-	F	T	T	T	T	T	T	T	T
C4: $a = b$ ?	-	-	-	T	T	T	T	F	F	F	F
C5: $a = c$ ?	-	-	-	T	T	F	F	T	T	F	F
C6: $b = c$ ?	-	-	-	T	F	T	F	T	F	T	F
A1: Not a triangle	X	X	X								
A2: Scalene											X
A3: Isosceles							X		X	X	
A4: Equilateral				X							
A5: Impossible					X	X		X			

# Bài tập JUnit

**Bài 2**: Viết chương trình giải phương trình bậc 2 và thực hiện Junit Test cho class đó

**Bài 3**: Thực hiện JUnit Test cho chương trình dưới đây:  
Shopping Cart: gồm 3 file đính kèm:



Product.java



ProductNotFoundException.java



ShoppingCart.java

**Bài 4**: Viết chương trình tìm UCLN, BCNN và thực hiện JUnit cho class đó





T3H

*IT-Institute*

**THANK YOU**