


Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. MH: Kiểm định phần mềm MSMH:	BUỔI 3. Test Unit	
---	--------------------------	---

A. MỤC TIÊU:

- Tạo và chạy UnitTest trong Visual Studio

B. DỤNG CỤ - THIẾT BỊ THỰC HÀNH CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

1. Giới thiệu tổng quan về UnitTest

Mỗi unit test là 1 đơn vị riêng biệt, độc lập, không phụ thuộc vào unit khác. Unit Test nên test từng đối tượng riêng biệt. Mỗi unit test là 1 method trong test class, tên method cũng là tên UnitTest.

Annotation

- Annotation [TestFixture] đặt vào đầu class chứa các unit test, đánh dấu đây là một bộ các unit test.
- Annotation [SetUp] để đánh dấu hàm setup. Hàm này sẽ được chạy vào đầu mỗi unit test.
- Annotation [Test] để đánh dấu hàm bên dưới là một unit test
- ...

Mẫu phổ biến để viết UnitTest (Arrange, Act, Assert):

- Arrange của phương thức kiểm tra đơn vị khởi tạo các đối tượng và đặt giá trị của dữ liệu được truyền cho phương thức được kiểm tra
- Act gọi phương thức được thử nghiệm
- Assert xác minh rằng hành động của phương thức được kiểm tra hoạt động như mong đợi.

Một số class Assert thông dụng:

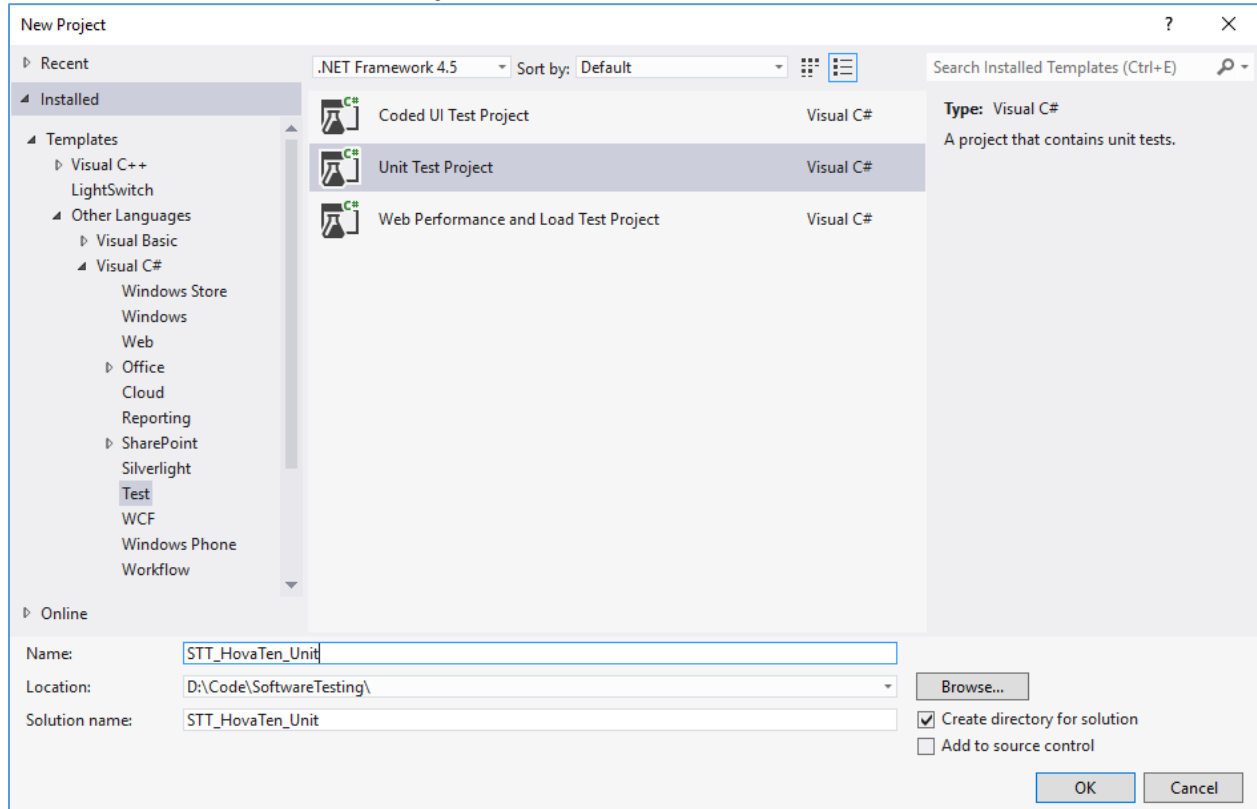
- **AreEqual(String, String, Boolean):** Kiểm tra các chuỗi được chỉ định có bằng nhau không và đưa ra một ngoại lệ nếu chúng không bằng nhau.
- **AreEqual(Double, Double, Double):** Kiểm tra số (kiểu double) đưa ra có bằng số mong đợi không
- **AreEqual(Object, Object, String):** Kiểm tra các đối tượng đã chỉ định có bằng nhau không và ném ngoại lệ nếu hai đối tượng không bằng nhau. Các loại số khác nhau được coi là không bằng nhau ngay cả khi các giá trị logic bằng nhau.
- **AreSame(Object, Object, String):** Kiểm tra xem cả hai đối tượng được chỉ định đều tham chiếu đến cùng một đối tượng và đưa ra một ngoại lệ nếu hai đầu vào không tham chiếu đến cùng một đối tượng.
- **Equals(Object, Object):** Kiểm tra xem các đối tượng được chỉ định cả hai tham chiếu đến được sử dụng để so sánh.

2. Hướng dẫn tạo và chạy UnitTest trong Visual Studio (NUnit)

Ví dụ 1: Viết UnitTest hàm tính tổng 2 số a, b:

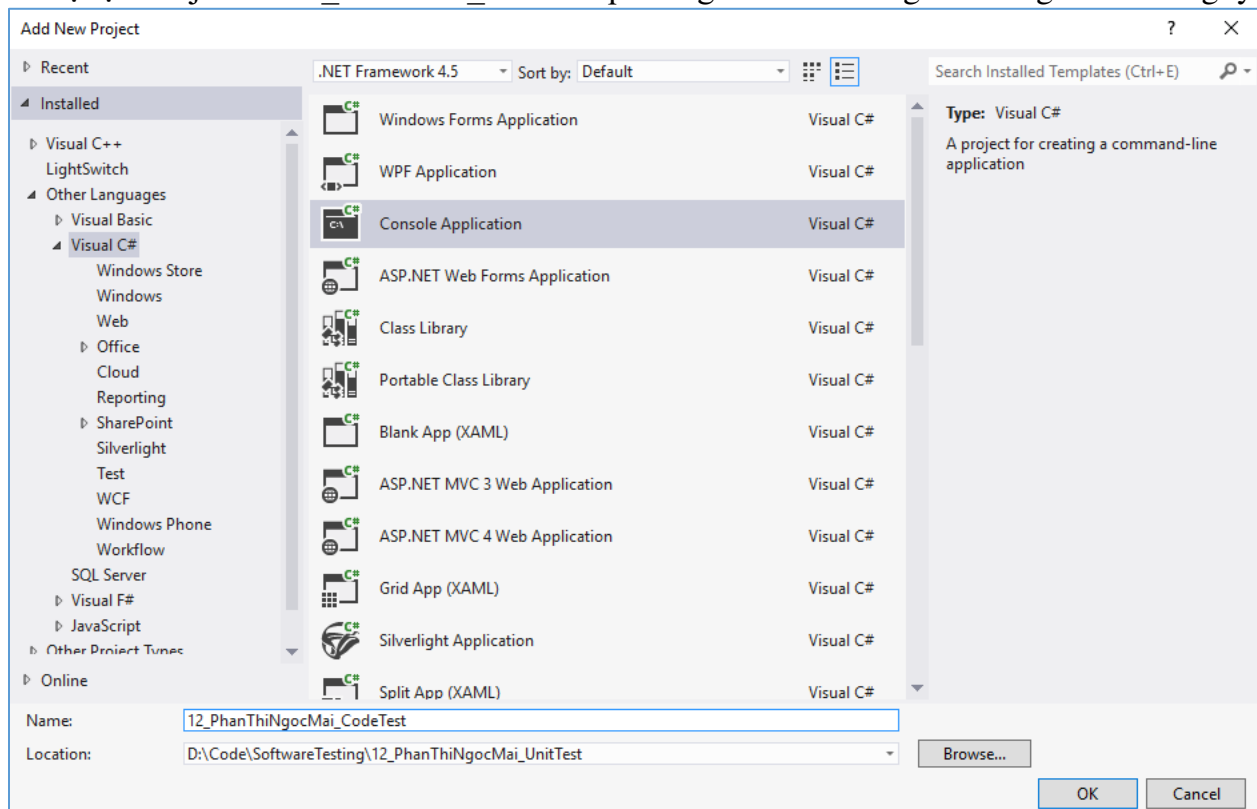
Bước 1: Tạo UnitTest

- (1) Click chuột phải Solution chọn Add – New Project → Test → Unit Test Project
- (2) Đặt tên UnitTest Project (Ví dụ: STT_HoVaTen_Unit)



Bước 2: Tạo project code và các phương thức cần kiểm tra

Ví dụ tạo Project: STT_HoVaTen_Code và phương thức tínhTong tính tổng của 2 số nguyên:

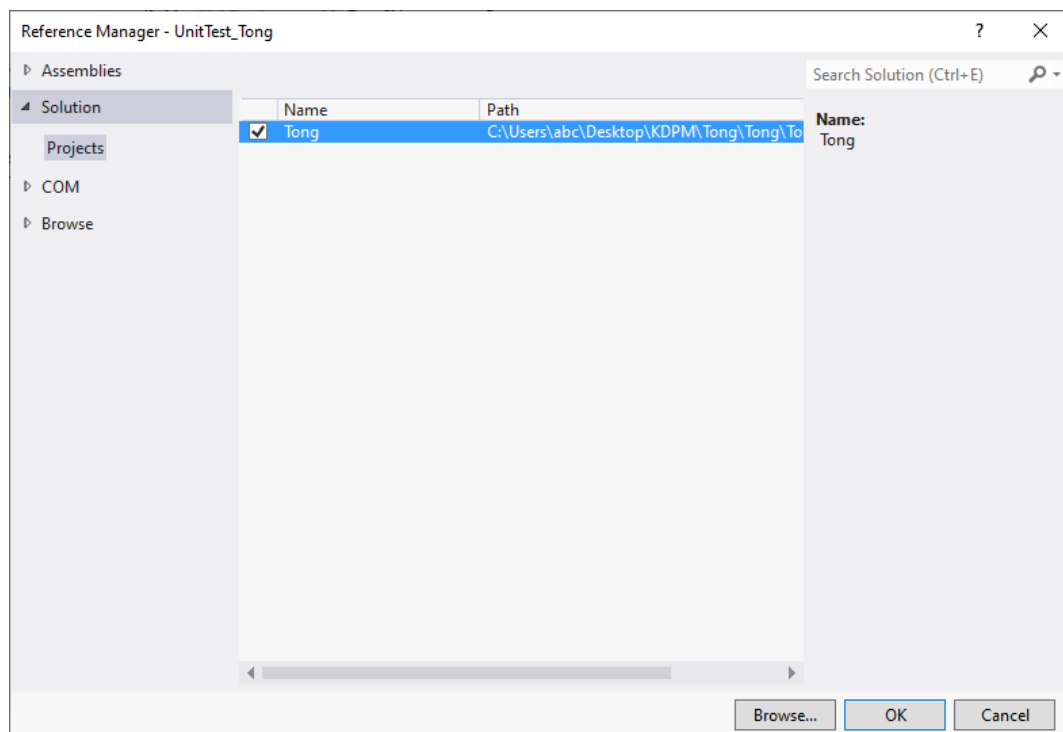
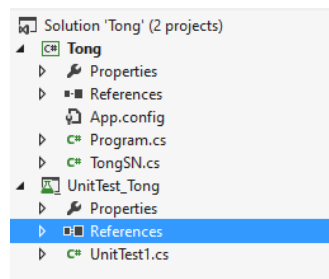


```
//Tính tổng 2 số nguyên
public int tinhTong(int a, int b)
{
    return a + b;
}
```

(3) Xây dựng bảng TestData

No.	Test Case	Test data	Result Expected	Ghi chú
1	TC01	5,7	12	Tính tổng 2 số: 5+7= 12 dùng assertEquals. Passed
2	TC02	2,7	12	Tính tổng 2: 2+7=12 dùng assertEquals. Failed

Bước 3: Trong Project UnitTest vừa tạo (UnitTest_Tong) add reference đến project cần test



Bước 4: Code UnitTest hàm tính tổng 2 số nguyên

Testcase 1:

```
[TestMethod]
public void TC1_testTong2SoNguyen()
{
    Buoil6 sn = new Buoil6();
    int Result_Actual = sn.tong2SoNguyen(5, 7);
    int Result_Expect = 12;
    Assert.AreEqual(Result_Expect, Result_Actual);
}
```

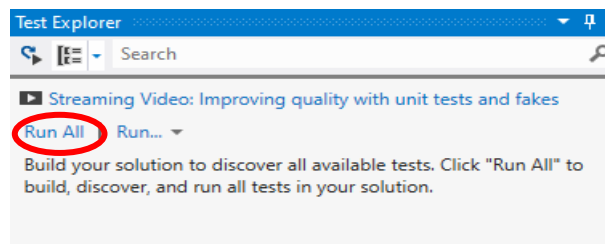
Testcase 2:

```
[TestMethod]
public void TC2_testTong2SoNguyen()
{
    Buoil6 sn = new Buoil6();
    int Result_Actual = sn.tong2SoNguyen(2, 7);
    int Result_Expect = 12;
    Assert.AreEqual(Result_Expect, Result_Actual);
}
```

Bước 3: Run UnitTest

Test – Windows – Test Explorer

Run Unit Test



Kết quả

Testcase 1:

```
[TestMethod]
public void TC1_testTong2SoNguyen()
{
    Buoil6 sn = new Buoil6();
    int Result_Actual = sn.tong2SoNguyen(5, 7);
    int Result_Expect = 12;
    Assert.AreEqual(Result_Expect, Result_Actual);
}
```

✓ Test Passed - TC1_testTong2SoNguyen
Elapsed time: 7 ms

Testcase 2:

```
[TestMethod]
public void TC2_testTong2SoNguyen()
{
    Buoil6 sn = new Buoil6();
    int Result_Actual = sn.tong2SoNguyen(2, 7);
    int Result_Expect = 12;
    Assert.AreEqual(Result_Expect, Result_Actual);
}
```

✗ Test Failed - TC2_testTong2SoNguyen
Message: Assert.AreEqual failed. Expected:<12>. Actual:<9>.
Elapsed time: 5 ms

Ví dụ 2: Viết UnitTest hàm tìm giá trị max, min 3 số

Tạo project tìm max, min 3 số.

Bước 1: Tạo Unit Test

Bước 2: Code Unit Test

TC1

```
public void TC1_TimMaxMin()
{
    FMinMax mm = new FMinMax();
    int resultMax, resultMin;
    mm.timMaxMin(3, 7, 2, out resultMax, out resultMin);
    Assert.AreEqual(7, resultMax);
    Assert.AreEqual(3, resultMin);
}
```

TC2

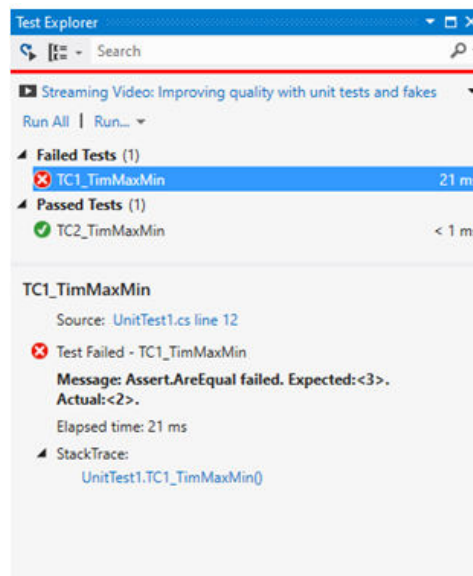
```
public void TC2_TimMaxMin()
{
    FMinMax mm = new FMinMax();
    int resultMax, resultMin;
    mm.timMaxMin(3, 7, 2, out resultMax, out resultMin);
    Assert.AreEqual(7, resultMax);
    Assert.AreEqual(2, resultMin);
}
```

Tương tự viết hàm TCn

Bảng TestData

No.	Test Case	Test data	Result Expected	Ghi chú
1	TC01	3,7,2	Max=7 Min=3	Tìm max, min 3,7,2 dùng areEqual. Thực sự Max=7, Min=2. Failed
2	TC02	3,7,2	Max=7 Min=2	Tìm max, min 3,7,2 dùng areEqual. Max=7, Min=2. Passed
...

Kết quả

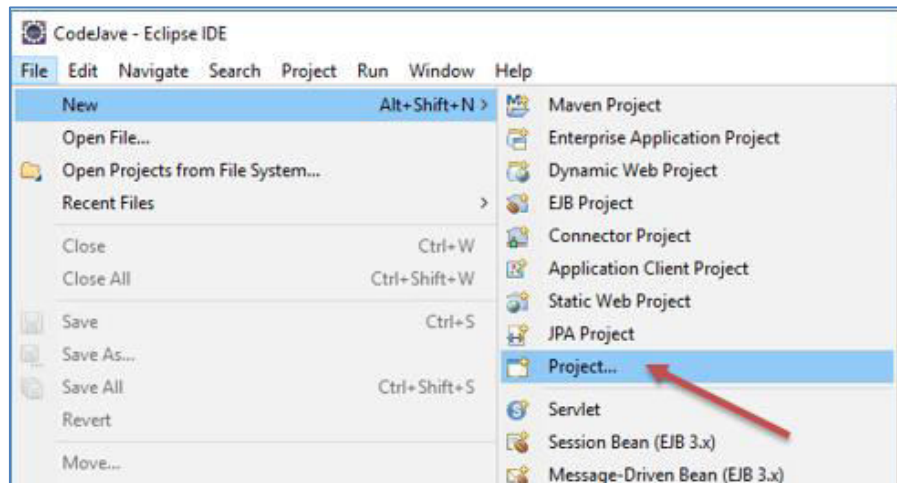


3. Hướng dẫn tạo và chạy UnitTest trong Eclipse (JUnit)

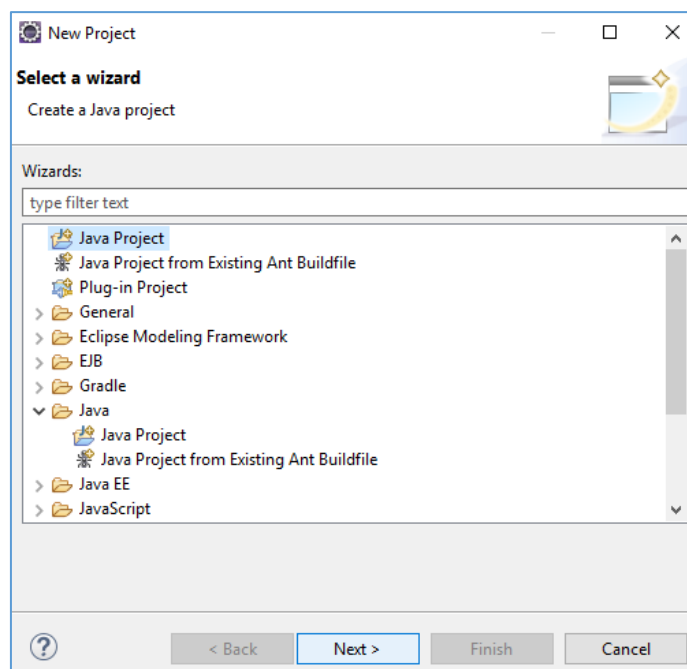
–JUnit đã được tích hợp sẵn trong các IDE thông dụng (Netbean, Eclipse) hỗ trợ lập trình Java. Nếu công cụ đang sử dụng không hỗ trợ thì có thể download tại trang web <http://www.junit.org> và cài đặt.

Tạo Project

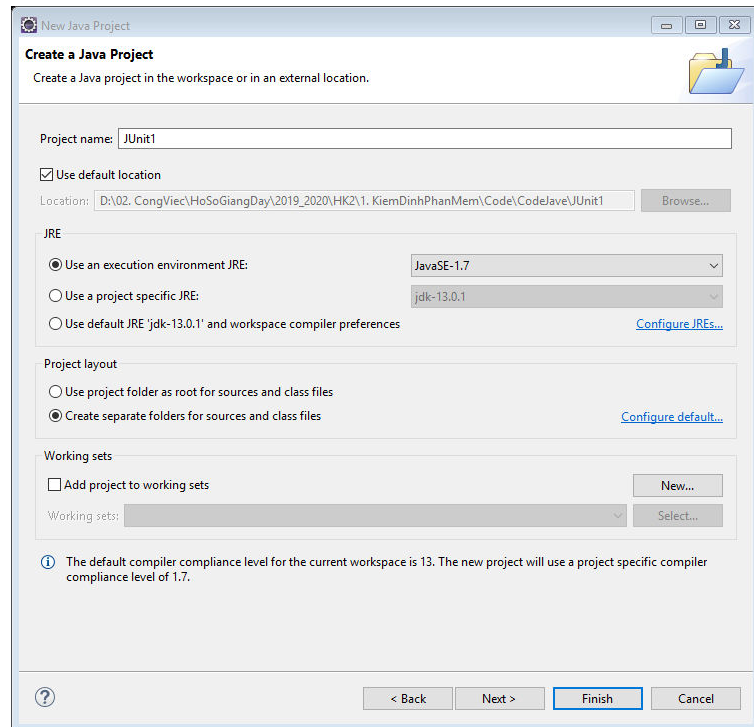
–**Bước 1:** Click chuột vào File → New → Project...



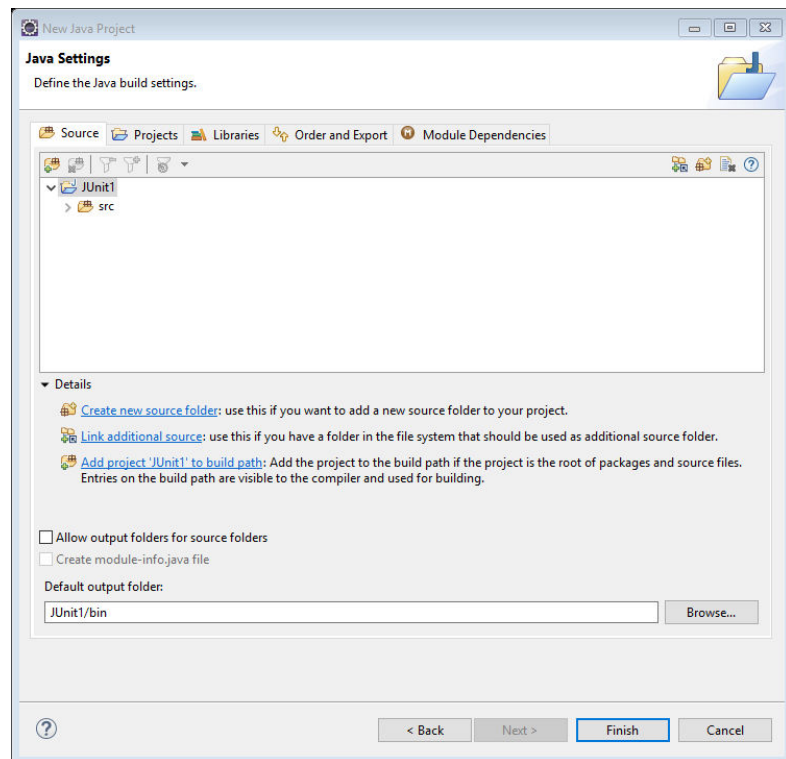
–**Bước 2:** Chọn Java Project trong hộp thoại New Project và nhấn Next



–**Bước 3:** Trong hộp thoại New Java Project điền thông tin: *Project name*: tên của project; *Chọn jre*: phiên bản của thư viện và máy ảo java và chọn *Next*

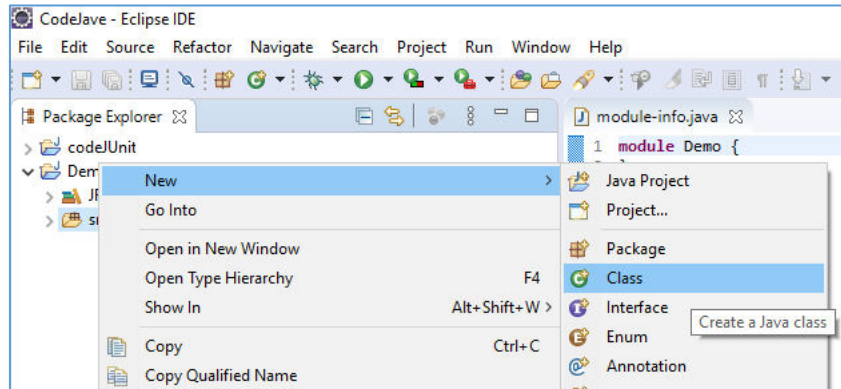


–**Bước 4:** Cấu hình thư viện (nếu không có thì để mặc định) và nhấn Finish



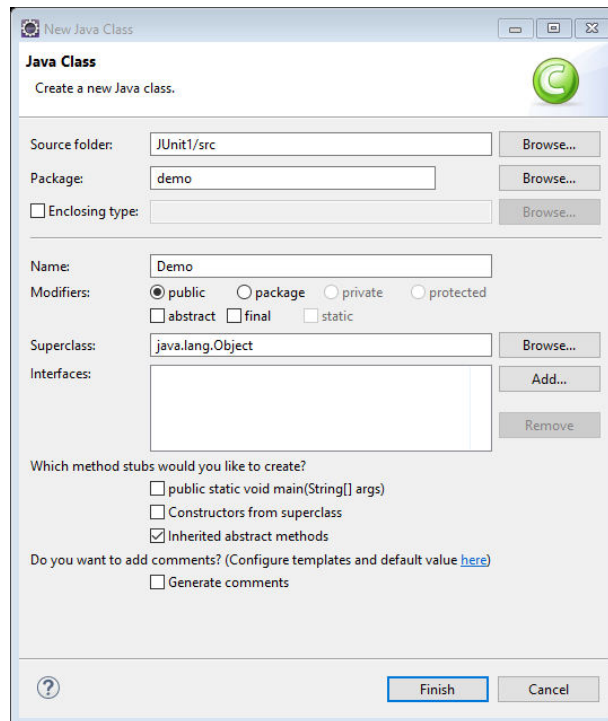
Tạo lớp trong java (Demo)

–**Bước 1:** Click chuột phải vào thư mục src → chọn New → chọn Class



–**Bước 2:** Trong hộp thoại New Java Class điền các giá trị sau và chọn Finish

- + Package: demo → phân nhóm các lớp đối tượng
- + Name: Demo
- + SuperClass: lớp cha (nếu có)



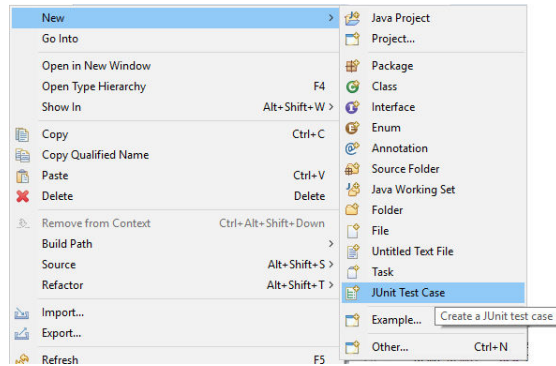
–**Bước 3:** Định nghĩa các thuộc tính, hàm

```

1 package demo;
2
3 public class Demo {
4     public boolean isPrimeNumber(int input) {
5         for (int i = 2; i < input; i++) {
6             if (input % i == 0)
7                 return false;
8         }
9         return true;
10    }
11 }
  
```

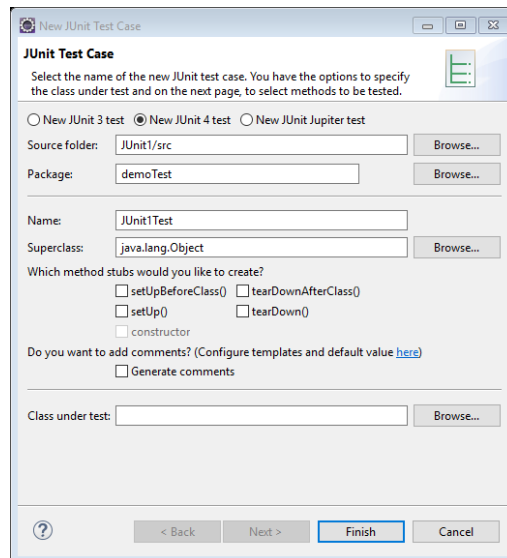
Tạo Junit Test case

–**Bước 1:** Chuột phải vào Project → chọn New → chọn Junit Test case



–**Bước 2:** Trong hộp thoại New Junit Test case chú ý các thuộc tính sau và chọn Next

- + Package: tên package (demoTest)
- + Name: tên lớp (JUnit1Test)



–**Bước 3:** chọn hàm/phương thức cần test ➔ nhấn Finish

–**Bước 4:** sử dụng JUnit để kiểm tra hàm **isPrimeNumber** với các đầu vào khác nhau.

```

1 package demoTest;
2 import static org.junit.Assert.*;
3 import org.junit.Test;
4 import demo.Demo;
5 public class JUnit1Test {
6     @Test
7     public void testIsPrimeNumber1() {
8         Demo demo1 = new Demo();
9         boolean result = demo1.isPrimeNumber(-1);
10        assertFalse(result);
11    }
12    @Test
13    public void testIsPrimeNumber2() {
14        Demo demo1 = new Demo();
15        boolean result = demo1.isPrimeNumber(0);
16        assertFalse(result);
17    }
18    @Test
19    public void testIsPrimeNumber3() {
20        Demo demo1 = new Demo();
21        boolean result = demo1.isPrimeNumber(1);
22        assertFalse(result);
23    }
24    @Test
25    public void testIsPrimeNumber4() {
26        Demo demo1 = new Demo();
27        boolean result = demo1.isPrimeNumber(2);
28        assertTrue(result);
29    }
30    @Test
31    public void testIsPrimeNumber5() {
32        Demo demo1 = new Demo();
33        boolean result = demo1.isPrimeNumber(4);
34        assertFalse(result);
35    }
36    @Test
37    public void testIsPrimeNumber6() {
38        Demo demo1 = new Demo();
39        boolean result = demo1.isPrimeNumber(5);
40        assertTrue(result);
41    }
}

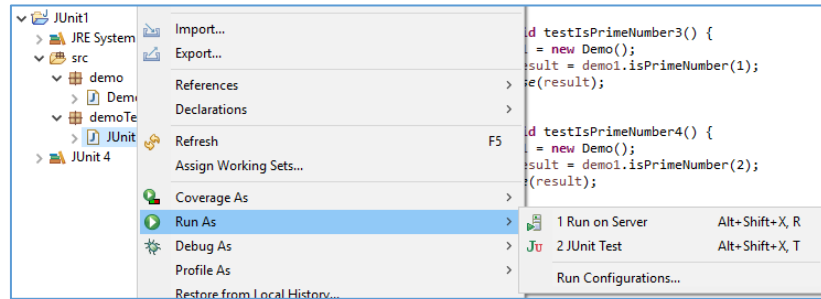
```

Bảng Testdata

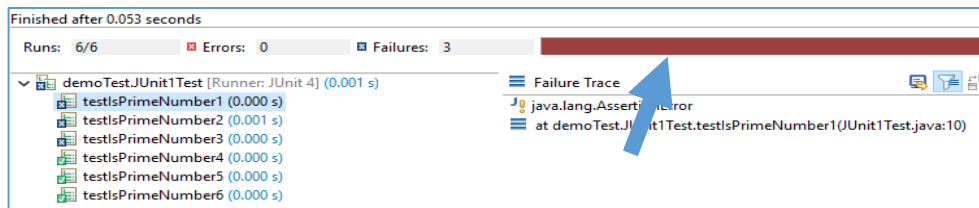
No.	Test Case	Test data	Result Expected	Ghi chú
1	TC01	-1	false	Hàm thứ nhất đầu vào là số âm '-1' nên kết quả mong đợi sẽ là false nên dùng assertFalse
2	TC02	0	false	Hàm thứ hai đầu vào là số 0, 0 không phải là số nguyên tố nên kết quả mong đợi sẽ là false
3	TC03	1	false	Hàm thứ ba đầu vào là số 1, 1 không phải là số nguyên tố nên kết quả mong đợi sẽ là false
4	TC04	2	true	Hàm thứ tư đầu vào là số '2', 2 là số nguyên tố nên kết quả mong đợi sẽ là true nên dùng assertTrue
5	TC05	4	false	Hàm thứ năm đầu vào là số '4', 4 không phải là số nguyên tố nên kết quả mong đợi sẽ là false
6	TC06	5	true	Hàm thứ sáu đầu vào là số '5', 5 là số nguyên tố nên kết quả mong đợi sẽ là true

Chạy Test case

Để chạy kiểm tra các test case trên, chúng ta sẽ chọn chuột phải trên class tương ứng cần test, sau đó chọn **Run As → Unit Test**. Tương tự, chúng ta cũng có thể thực thi test cho một phương thức hoặc cả project.



Kết quả:



Chạy 6 test case trên có tổng thời gian 0.053 giây, thanh trạng thái màu đỏ tức là có test case không pass (3 test case thất bại) → trên hình 3 test case đầu tiên bị thất bại tức là method `isPrimeNumber` đang bị sai cho những trường hợp đó, do quên chưa kiểm tra trường hợp bằng 0 và 1 và nhỏ hơn không.

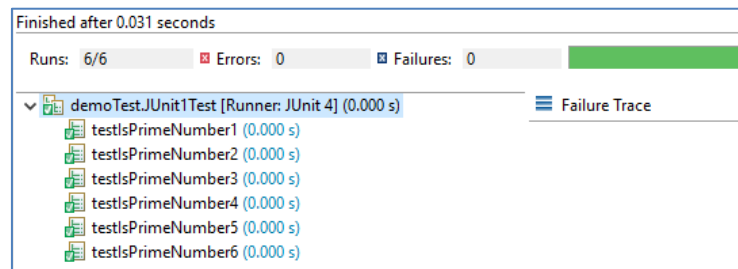
→ Sửa lại method `isPrimeNumber` để fix các lỗi:

```

1 package demo;
2 public class Demo {
3     public boolean isPrimeNumber(int input) {
4         if(input<2)
5             return false;
6         for (int i = 2; i < input; i++) {
7             if (input % i == 0)
8                 return false;
9         }
10        return true;
11    }
12 }

```

Chạy lại:



Tất cả các test case đều pass, thanh trạng thái chuyển màu xanh → Các test case của JUnit không cần phải sửa gì cả, và nó cũng không cần quan tâm các method cần kiểm tra thực hiện gì mà chỉ cần quan tâm đầu vào và đầu ra.

Assertions JUnit

Trong đầu tiên 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...

Assert được sử dụng để kiểm tra kết quả mong đợi cho các primitive type, Object, array primitive, array Object

Method assert:

assert[kiểu so sánh](expecteds_value, actuals_value)

hoặc

assert[kiểu so sánh](message, expecteds_value, actuals_value)

Trong đó: message là dữ liệu in ra nếu assert thất bại

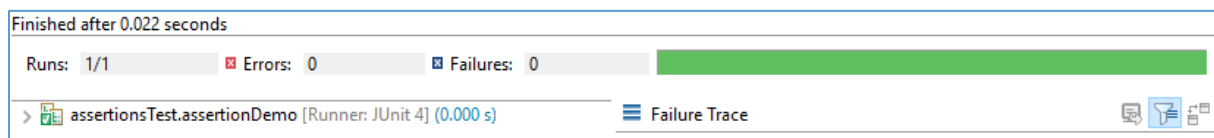
`assertEquals()` :

So sánh 2 giá trị để kiểm tra bằng nhau. Test sẽ được chấp nhận nếu các giá trị bằng nhau

Ví dụ:

```
@Test
public void whenAssertingEquality_thenEqual() {
    String expected = "DH_CNTP";
    String actual = "DH_CNTP";
    assertEquals(expected, actual);
}
```

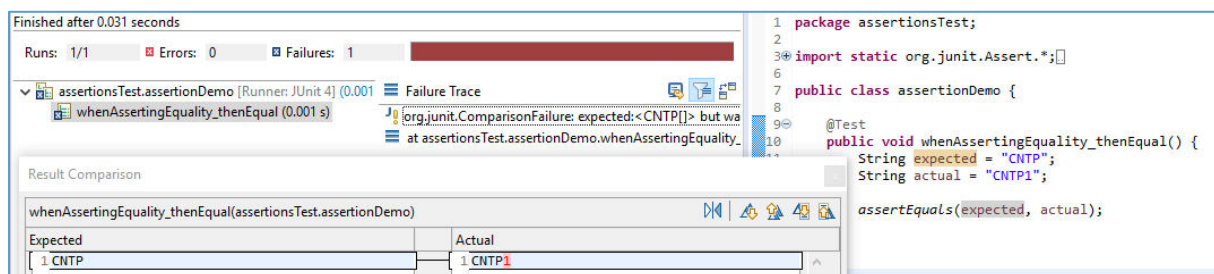
Kết quả:



Có thể chỉ định một thông báo sẽ hiển thị khi xác nhận thất bại

`assertEquals("failure - strings are not equal", expected, actual);`

Trường hợp không bằng nhau:



`assertTrue` and `assertFalse`

- **assertTrue()**: Đánh giá một biểu thức luận lý. Test sẽ được chấp nhận nếu biểu thức đúng.
- **assertFalse()**: Đánh giá biểu thức luận lý. Test sẽ được chấp nhận nếu biểu thức sai.

Ví dụ:

```
@Test
public void whenAssertingConditions_thenVerified() {
```

```
assertTrue("5 is greater then 4", 5 > 4);
assertFalse("5 is not greater then 6", 5 > 6);
}
```

assertNotNull and assertNull

- **assertNull()**: So sánh tham chiếu của một đối tượng với giá trị null. Test sẽ được chấp nhận nếu tham chiếu là null.
- **assertNotNull()**: So sánh tham chiếu của một đối tượng với null. Test sẽ được chấp nhận nếu tham chiếu đối tượng khác null.

Ví dụ:

```
@Test
public void whenAssertingNull_thenTrue() {
    Object car = null;
    assertNull("The car should be null", car);
}
```

assertNotSame and assertEquals

- **assertSame()**: So sánh địa chỉ vùng nhớ của 2 tham chiếu đối tượng bằng cách sử dụng toán tử ==. Test sẽ được chấp nhận nếu cả 2 đều tham chiếu đến cùng một đối tượng.
- **assertNotSame()**: So sánh địa chỉ vùng nhớ của 2 tham chiếu đối tượng bằng cách sử dụng toán tử ==. Test sẽ được chấp nhận nếu cả 2 đều tham chiếu đến các đối tượng khác nhau.

Ví dụ:

```
@Test
public void whenAssertingNotSameObject_thenDifferent() {
    Object cat = new Object();
    Object dog = new Object();

    assertNotSame(cat, dog);
}
```

assertArrayEquals () :

- So sánh 2 mảng để kiểm tra bằng nhau. Test sẽ được chấp nhận nếu các giá trị của 2 mảng là bằng nhau.

```
@Test
public void whenAssertingArraysEquality_thenEqual() {
    char[] expected = {'J','u','n','i','t'};
    char[] actual = "Junit".toCharArray();

    assertEquals(expected, actual);
}
```

- Nếu cả hai mảng đều null, khẳng định sẽ coi chúng bằng nhau:

```
@Test
public void givenNullArrays_whenAssertingArraysEquality_thenEqual() {
    int[] expected = null;
    int[] actual = null;
    assertEquals(expected, actual);
}
```

```
}
```

assertThat() :

- So sánh một giá trị thực tế có thỏa mãn với 1 **org.hamcrest.Matcher** được xác định hay không. Với matchers có thể kiểm tra kết quả của một string, number, collections...

```
@Test
public void testAssertThatHasItems() {
    assertThat(
        Arrays.asList("Java", "Kotlin", "Scala"),
        hasItems("Java", "Kotlin"));
}
```

fail() :

- Phương thức này làm cho test hiện hành thất bại, phương thức này thường được sử dụng khi xử lý các ngoại lệ.

```
@Test
public void whenCheckingExceptionMessage_thenEqual() {
    try {
        methodThatShouldThrowException();
        fail("Exception not thrown");
    } catch (UnsupportedOperationException e) {
        assertEquals("Operation Not Supported", e.getMessage());
    }
}
```

Annotation

JUnit cung cấp một số Annotation để viết Test

– **@Before:** Phương thức được đánh dấu với Annotation này sẽ được gọi trước mỗi khi phương thức **@Test** được gọi và được sử dụng để khởi tạo dữ liệu trước khi thực thi một phương thức **@Test**.

– **@After:** Phương thức được đánh dấu với Annotation này sẽ được gọi sau mỗi khi phương thức **@Test** được gọi và được sử dụng để dọn dẹp bộ nhớ sau khi thực thi một phương thức **@Test**.

– **@BeforeClass:** Phương thức được đánh dấu với Annotation này sẽ được gọi trước khi thực thi tất cả các phương thức **@Test** được gọi trong một Test class. Phương thức này chỉ được gọi một lần duy nhất. Phương thức đánh dấu Annotation này phải là static. Nó thường được sử dụng để khởi tạo dữ liệu cho việc thực thi một Test class.

– **@AfterClass:** Tương tự như **@BeforeClass**, nhưng nó được gọi sau khi kết thúc thực thi các phương thức **@Test**. Phương thức này chỉ được gọi một lần duy nhất. Phương thức đánh dấu Annotation này phải là static. Nó thường được sử dụng để dọn dẹp bộ nhớ sau khi thực thi tất cả các phương thức **@Test** trong một Test class.

– **@Test:** Được sử dụng để đánh dấu đây là một phương thức [test.@Test\(timeout=500\)](#): Được sử dụng khi cần giới hạn thời gian thực thi của một phương thức. Nếu vượt quá thời gian này thì phương thức sẽ fail.

– **@Test(expected= XxxException.class):** Được sử dụng khi cần test một ngoại lệ được throw ra từ phương thức được test. Nếu ngoại lệ không được throw thì phương thức sẽ fail.

– **@Ignore**: Được sử dụng để đánh dấu phương thức này để được bỏ qua (ignore/ disable), không cần thực thi test. Nó có thể sử dụng cho một phương thức test hay một class từ một test suite.

– **@FixMethodOrder**: Annotation này cho phép user có thể chọn thứ tự thực thi các phương thức @Test trong một test class.

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 @SuiteClasses.

Ví dụ có 2 class test sau:

TestA:

```
import org.junit.Test;
import static org.junit.Assert.assertTrue;
public class TestA {

    @Test
    public void testA2() {
        System.out.println("testA2");
        assertTrue(true);
    }

    @Test
    public void testA() {
        System.out.println("testA");
        assertTrue(true);
    }

    @Test
    public void testA1() {
        System.out.println("testA1");
        assertTrue(true);
    }
}
```

Test B:

```
import static org.junit.Assert.assertTrue;
import org.junit.Test;
public class TestB {

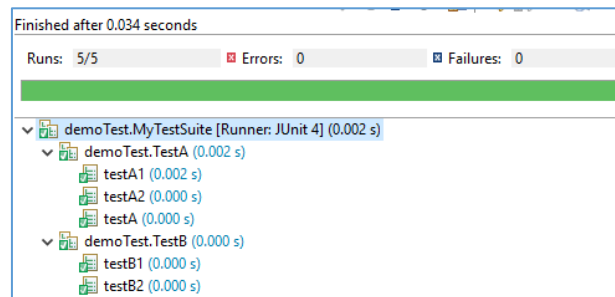
    @Test
    public void testB1() {
        System.out.println("testB1");
        assertTrue(true);
    }

    @Test
    public void testB2() {
        System.out.println("testB2");
        assertTrue(true);
    }
}
```

Khai báo class TestA.java trước TestB.java

```
1 package demoTest;
2 import org.junit.runner.RunWith;
3 import org.junit.runners.Suite;
4 import org.junit.runners.Suite.SuiteClasses;
5 @RunWith(Suite.class)
6 @SuiteClasses({TestA.class, TestB.class})
7 public class MyTestSuite {
8
9 }
```

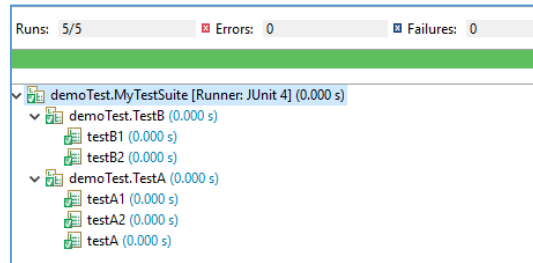
Kết quả: test A chạy trước test B



Ngược lại: khai báo class TestB.java trước TestA.java

```
1 package demoTest;
2 import org.junit.runner.RunWith;
3 import org.junit.runners.Suite;
4 import org.junit.runners.Suite.SuiteClasses;
5 @RunWith(Suite.class)
6 @SuiteClasses({TestB.class, TestA.class})
7 public class MyTestSuite {
8
9 }
```

Kết quả:



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 `@FixMethodOrder` ở đầu class. Có 3 kiểu sắp xếp:

–`@FixMethodOrder(MethodSorters.DEFAULT)`: Đây là kiểu sắp xếp mặc định nếu không khai báo `@FixMethodOrder`, tuy nhiên với kiểu này thì sẽ không thể xác định chính xác method nào sẽ được chạy trước

–`@FixMethodOrder(MethodSorters.JVM)`: Thứ tự các method test dựa theo JVM. Tuy nhiên thứ tự này có thể bị thay đổi khi chạy.

–`@FixMethodOrder(MethodSorters.NAME_ASCENDING)`: Thứ tự các method được thực thi dựa theo tên method.

(Kiểu `MethodSorters.NAME_ASCENDING` thì chắc chắn biết trước thứ tự các method sẽ chạy)

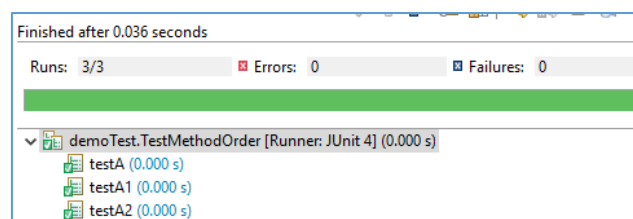
Ví dụ: Thực hiện chạy các method test theo thứ tự tên method:

```

2 import org.junit.Test;
3 import org.junit.FixMethodOrder;
4 import org.junit.runners.MethodSorters;
5 import static org.junit.Assert.assertTrue;
6
7 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
8
9 public class TestMethodOrder {
10
11     @Test
12     public void testA2() {
13         System.out.println("testA2");
14         assertTrue(true);
15     }
16
17     @Test
18     public void testA() {
19         System.out.println("testA");
20         assertTrue(true);
21     }
22
23     @Test
24     public void testA1() {
25         System.out.println("testA1");
26         assertTrue(true);
27     }
28 }

```

Kết quả:



4. Bài tập thực hành trên lớp:

SV thực hiện tự chọn thực hiện test unit cho các bài tập này bằng NUnit hoặc JUnit

Tạo Project đặt tên *STT_HoVaTen_BuoiX*. Trong Project *STT_HoVaTen_BuoiX*, tạo hai Project: Code và TestUnit

Bài 1: Trong Project demo tạo lớp *kiemTraTamGiac*: định nghĩa phương thức kiểm tra a, b, c có là độ dài ba cạnh của một tam giác. Nếu có thì cho biết đó là tam giác gì (*thường, cân, vuông, vuông cân, đều*)? Trong *UnitTest* tạo các testcase để kiểm tra tính đúng của các kiểu tam giác trên.

Bài 2: Trong Project demo tạo lớp *tinhTienDien*: định nghĩa phương thức số điện sinh hoạt (CSM – CSC). Từ số điện sinh hoạt tính tiền điện sinh hoạt dựa vào bảng sau



Trong *UnitTest* tạo các testcase để kiểm tra tính đúng của các điều kiện trên.

Bài 3: Trong Project demo tạo lớp *AmLich*: nhập vào năm dương lịch, xuất ra tên âm lịch.

Dựa theo bảng quy ước số thứ tự cho can – chi năm sinh:

BẢNG QUY ƯỚC SỐ THỨ TỰ CHO CAN CHI NĂM SINH												
Thiên Can	Canh	Tân	Nhâm	Quý	Giáp	Ất	Bính	Đinh	Dậu	Kỷ		
Địa Chi	Tý	Sửu	Dần	Mão	Thìn	Tị	Ngọ	Mùi	Thân	Dậu	Tuất	Hợi
Số tương ứng	0	1	2	3	4	5	6	7	8	9	10	11

Ghép Can- Chi lại với nhau, ta được hệ đếm gồm 60 đơn vị với các tên như:

Ví dụ: nhập 2020 xuất ra Canh Tý.

Trong *UnitTest* tạo các testcase để kiểm tra tính đúng của các điều kiện trên.

5. Bài tập thực hành về nhà:

Bài 4: Trong Project demo tạo lớp loạiHS.

Điều kiện đánh giá và xếp loại học sinh như sau:

- Học sinh thi 3 môn Toán, Lý, Hoá được đánh giá học lực trên thang điểm 10. Thang xếp loại học lực từ cao đến thấp thứ tự như sau: Giỏi, Khá, Trung Bình, Yếu.
- Học sinh cũng được đánh giá rèn luyện trên thang điểm 10. Thang xếp loại tương tự thang xếp loại học lực: Tốt, Khá, Trung Bình, Yếu. Xếp loại phải tương ứng với năng lực.
- Điểm trung bình chung nhỏ hơn 5.0 thì xếp loại ban đầu Yếu.
- Điểm trung bình chung từ 5.0 đến dưới 6.5 thì xếp loại ban đầu Trung Bình.
- Điểm trung bình chung từ 6.5 đến dưới 8.0 thì xếp loại ban đầu Khá.
- Điểm trung bình chung từ 8.0 trở lên thì xếp loại ban đầu Giỏi.
- Học sinh xếp loại ban đầu Khá, Giỏi chỉ khi có điểm rèn luyện tương đương và điểm thành phần 3 môn có xếp loại tương ứng từ Trung Bình, Khá trở lên. Nếu không thì giảm một bậc.
- Trong các điểm thành phần, nếu có bất kì điểm nào nhỏ hơn 3.0, thì lập tức xếp loại Yếu.

Yêu cầu:

Viết một hàm đánh giá bằng ngôn ngữ C, đầu vào điểm thi 3 môn và điểm rèn luyện, đầu ra là một chuỗi đánh giá xếp loại thuộc bộ {4, 3, 2, 1, 0} tương ứng thang xếp loại trong đề bài và 0 là không đánh giá được.

Ví dụ:

Toán	Lý	Hoá	TBC	R.Luyện	X.Loại	Ghi chú
11.0	9.0	9.0	-	9.0	0	Dữ liệu sai
9.0	9.0	9.0	9.0	9.0	4	Giỏi
7.0	7.0	7.0	7.0	7.0	3	Khá
6.0	6.0	6.0	6.0	6.0	2	TB
3.5	4.0	6.0	4.5	6.5	1	Yếu
2.5	9.0	8.0	6.5	6.0	1	Khá > Yếu, có điểm chết, RL ko đủ
6.0	9.0	9.0	8.0	8.0	3	Giỏi > Khá
7.0	8.0	9.0	8.0	4.5	1	Giỏi > Yếu, RL ko đủ
4.0	9.0	8.0	7.0	6.0	2	Khá > TB, điểm và RL ko đủ

Trong UnitTest tạo các testcase để kiểm tra tính đúng của các điều kiện trên.

Bài 5: Trong Project demo tạo lớp giaiPhuongTrinhBacHai ($ax^2+bx+c=0$), định nghĩa phương thức kiểm tra các trường hợp.

- Nếu $a = 0$: Phương trình trở thành: $bx + c = 0$, khi đó:
 - + Nếu $b \neq 0$, phương trình $\Leftrightarrow x = -\frac{c}{b}$, do đó phương trình có nghiệm duy nhất $x = -\frac{c}{b}$.
 - + Nếu $b = 0$, phương trình trở thành $0x + c = 0$, ta tiếp tục xét 2 trường hợp:
Trường hợp 1: Với $c = 0$, phương trình nghiệm đúng với mọi $x \in \mathbb{R}$.
Trường hợp 2: Với $c \neq 0$, phương trình vô nghiệm.
- Nếu $a \neq 0$: xét $\Delta = b^2 - 4ac$:
 - + Trường hợp 1: Nếu $\Delta > 0$, phương trình có hai nghiệm phân biệt $x = \frac{-b \pm \sqrt{\Delta}}{2a}$.
 - + Trường hợp 2: Nếu $\Delta = 0$, phương trình có nghiệm kép $x = -\frac{b}{2a}$.
 - + Trường hợp 3: Nếu $\Delta < 0$, phương trình vô nghiệm.

Trong UnitTest tạo các testcase để kiểm tra tính đúng của các điều kiện trên.

-- HẾT--