


Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm MH: TH Kiểm định phần mềm MSMH:	BUỔI 4: JUnit	
---	----------------------	---

A. MỤC TIÊU:

1. Thực hành các thao tác kiểm định phần mềm trên JUnit

- Cài đặt JUnit
- Hướng dẫn sử dụng JUnit trong Eclipse
- Bài tập thực hành
- Bài tập về nhà

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	

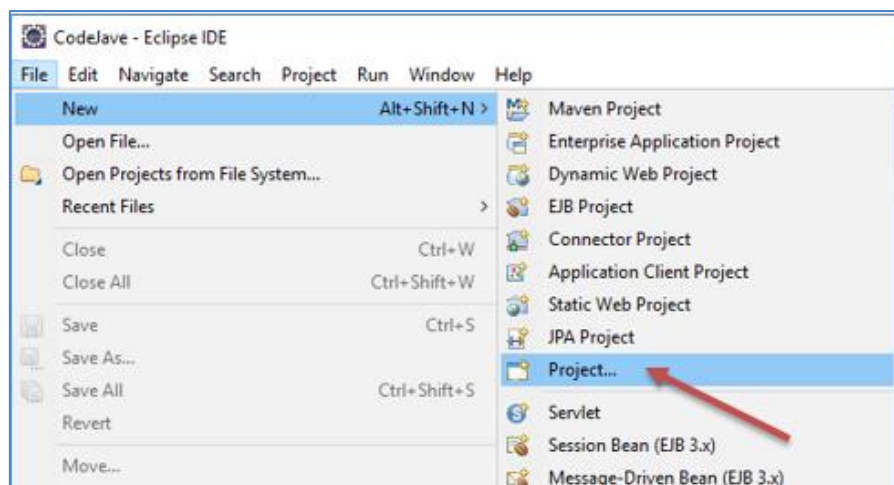
C. NỘI DUNG THỰC HÀNH

1. Hướng dẫn sử dụng JUnit trong Eclipse

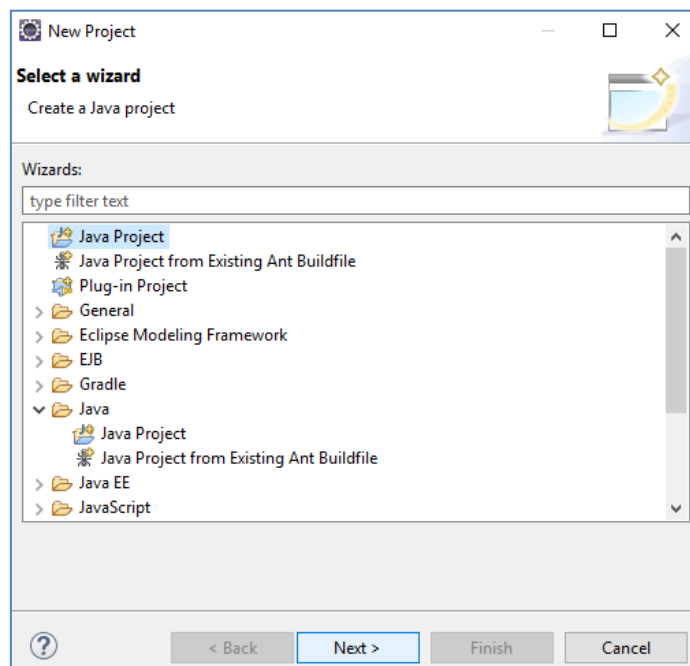
–JUnit đã được tích hợp sẵn trong các IDE thông dụng (Netbean, Eclipse) hỗ trợ lập trình java. Nếu công cụ đang sử dụng không hỗ trợ thì có thể download tại trang web <http://www.junit.org> và cài đặt.

1.1. Tạo Project

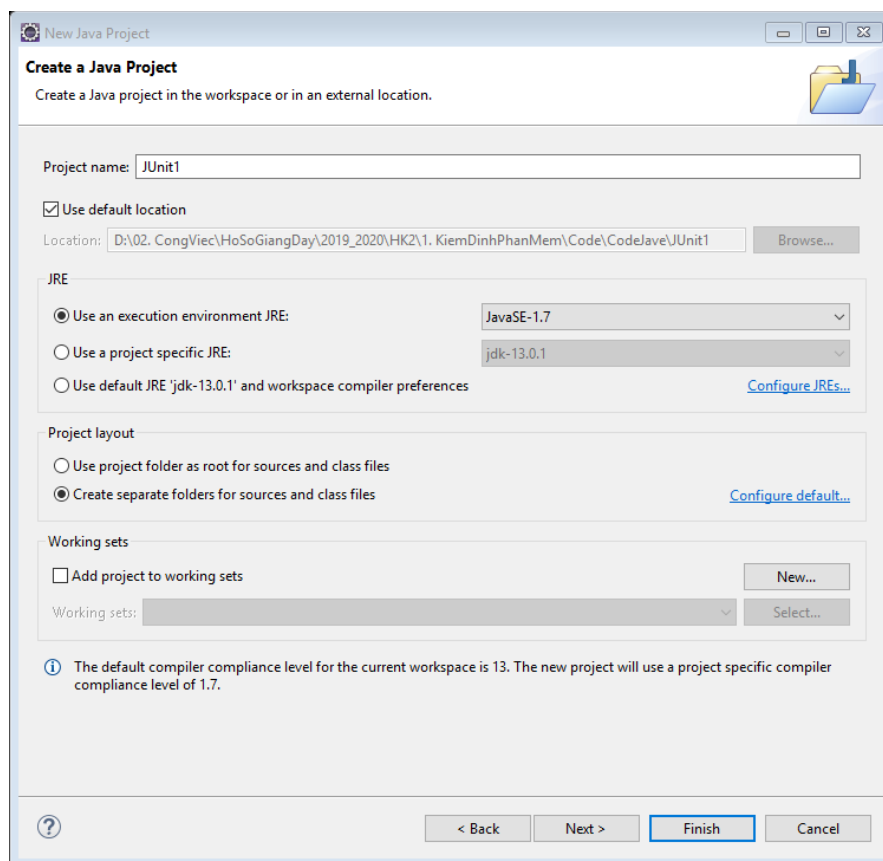
–**Bước 1:** Click chuột vào File → New → Project...



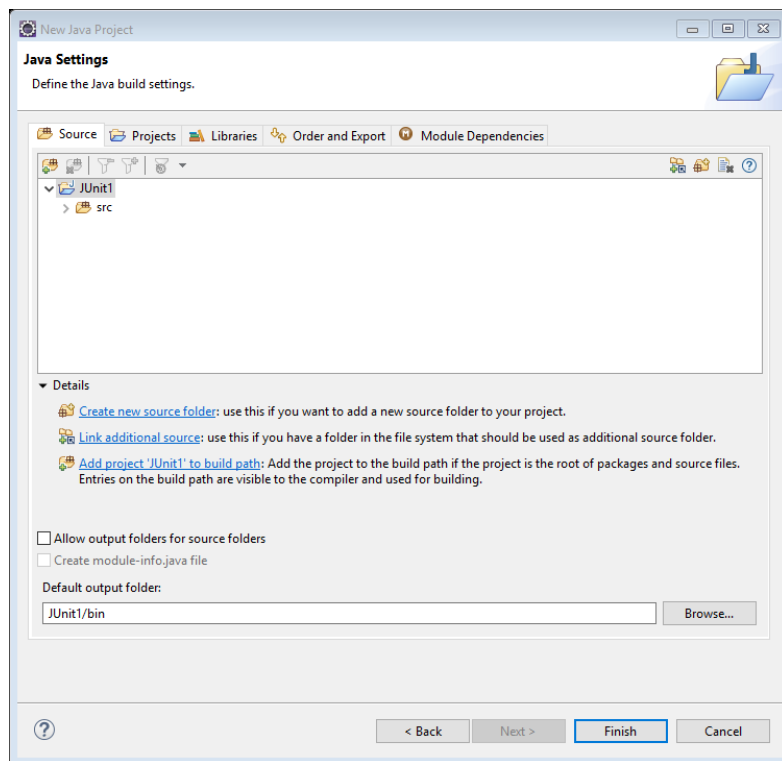
–**Bước 2:** Chọn Java Project trong hộp thoại New Project và nhấn Next



–**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



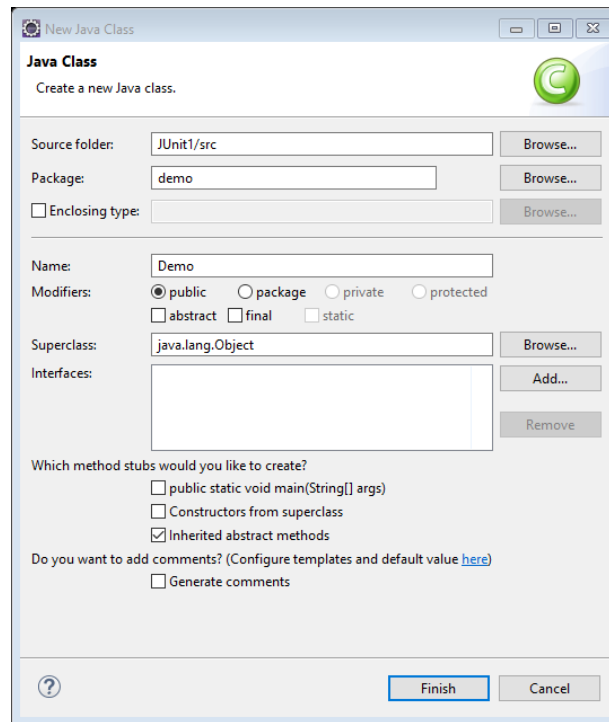
1.2. Tạo lớp trong java (Demo)

–**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

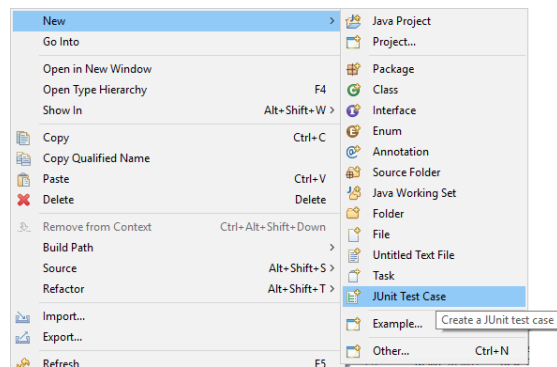
```

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 }

```

1.3. Tạo Junit Test case

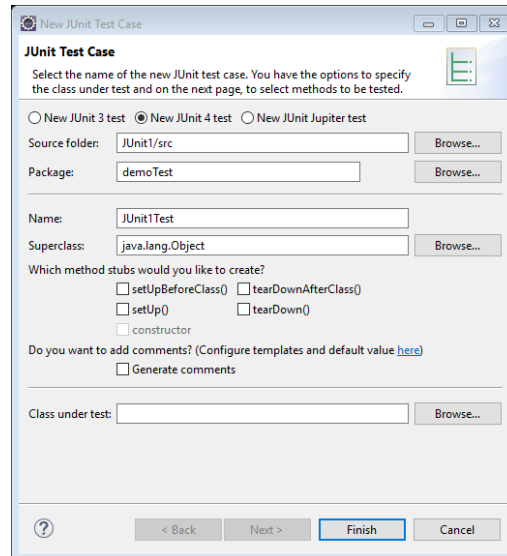
–**Bước 1:** Chuột phải vào Project → chọn New → chọn Junit Test case



–**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.

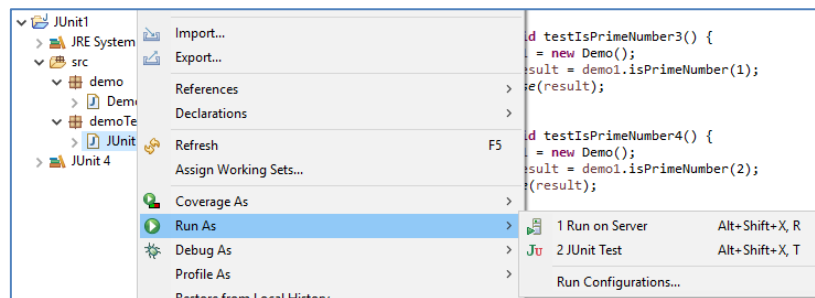
```
1 package demoTest;
2 import static org.junit.Assert.*;
3 import org.junit.Test;
4 import demo.Demo;
5 public class JUnit1Test {
6     @Test
7     public void testIsPrimeNumber1() {
8         Demo demo1 = new Demo();
9         boolean result = demo1.isPrimeNumber(-1);
10        assertFalse(result);
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

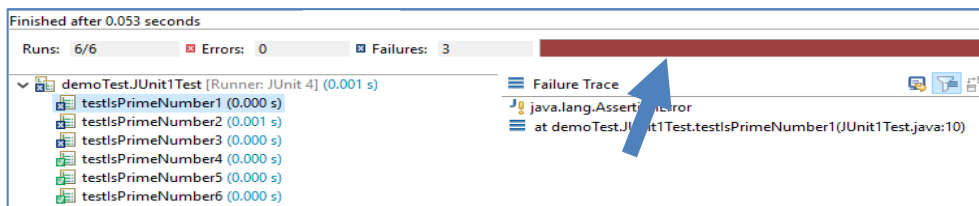
No.	Test Case	Test data	Result Expected	Ghi chú
1	TC01	-1	false	Hàm thứ nhất đầu vào là số âm '-1' nên kết quả mong đợi sẽ là false nên dùng assertFalse
2	TC02	0	false	Hàm thứ hai đầu vào là số 0, 0 không phải là số nguyên tố nên kết quả mong đợi sẽ là false
3	TC03	1	false	Hàm thứ ba đầu vào là số 1, 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

1.4. 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:

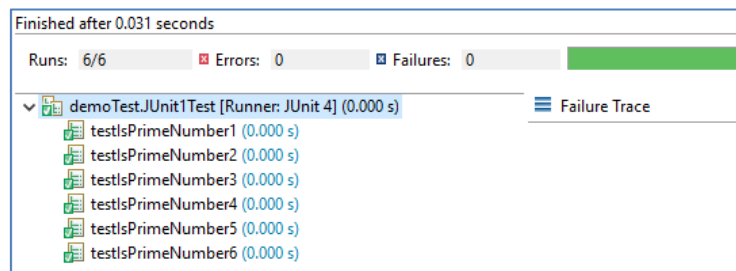


Chạy 6 test case trên có tổng thời gian 0.053 giây, thanh trạng thái màu đỏ tức là có test case không pass (3 test case thất bại) → trên hình 3 test case đầu tiên bị thất bại tức là method `isPrimeNumber` đang bị sai cho những trường hợp đó, do quên chưa kiểm tra trường hợp bằng 0 và 1 và nhỏ hơn không.

→ 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.

1.5. 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

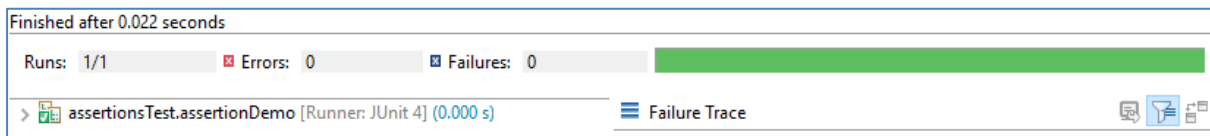
a. assertEquals():

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);
}
```

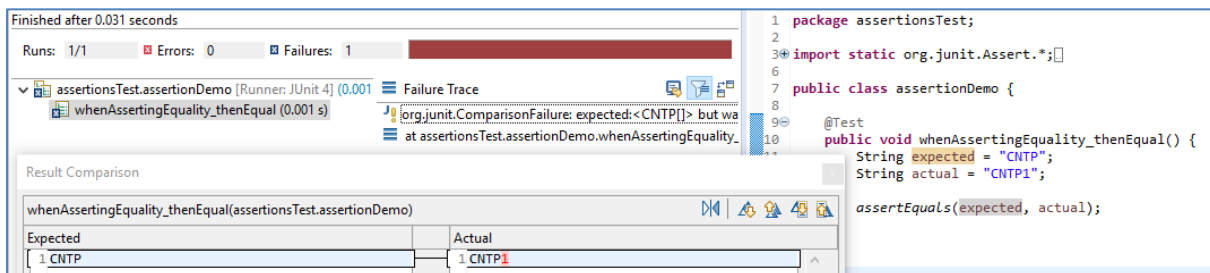
Kết quả:



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:



b. 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);
}
```


c. **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);
}
```

d. **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);
}
```

e. **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);
}
```

```
int[] actual = null;
assertArrayEquals(expected, actual);
}
```

f. **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"));
}
```

g. **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());
    }
}
```

1.6. 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 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.

1.7. 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.

a. 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 {

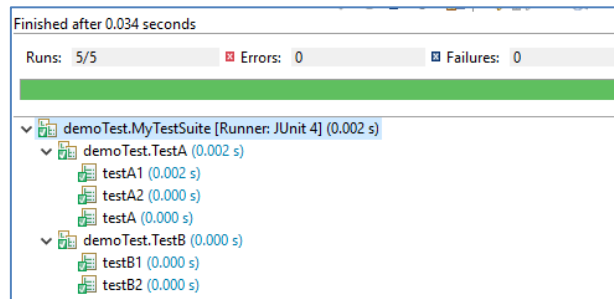
    @Test
    public void testB1() {
        System.out.println("testB1");
        assertTrue(true);
    }

    @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



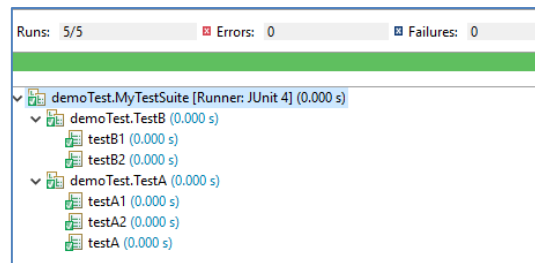
Ngược lại: khai báo class TestB.java trước TestA.java

```

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:



b. 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)

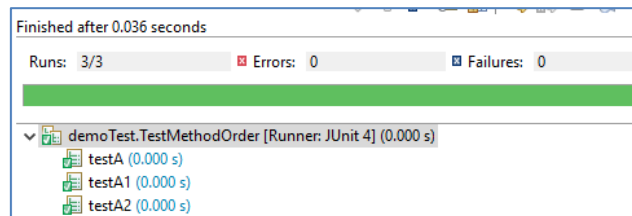
Ví dụ: Thực hiện chạy các method test theo thứ tự tên method:

```

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:



2. Bài tập thực hành trên lớp

Tạo Project đặt tên *STT_HoVaTen_Buoi4*. Trong Project *STT_HoVaTen_Buoi4*, tạo hai package: demo và test

Bài 1: Trong package *demo* tạo lớp *DaysInMonth*, định nghĩa phương thức kiểm tra xem tháng, năm đó có bao nhiêu ngày. Trong đó số ngày trong tháng được định nghĩa:

- Các tháng có 31 ngày là: 1, 3, 5, 7, 8, 10, 12.
- Các tháng có 30 ngày là: 4, 6, 9, 11.
- Riêng tháng hai nếu là năm nhuận sẽ có 29 ngày, ngược lại nếu không nhuận thì có 28 ngày. Trong đó, một năm được gọi là nhuận nếu số năm đó chia hết cho 4 mà không chia hết cho 100; Hoặc năm đó chia hết cho 400.

Trong package *test* tạo lớp *testDaysInMonth* và các test case để kiểm tra tính đúng đắn của các phương thức ứng trong tất cả các trường hợp.

Bài 2: Trong package *demo* tạo lớp *VeMayBay*, định nghĩa phương thức kiểm tra vé máy bay theo mô tả như sau:

- ✓ Hạng thương gia
 - Người lớn: 4.000.000 VND
 - Trẻ em: 1.500.000 VND
- ✓ Hạng thông thường
 - Người lớn : 3.000.000 VND

- Trẻ em: 700.000 VND
- ✓ Quy định về tuổi:
 - Trẻ em: < 7 tuổi
 - Người lớn: ≥ 7 tuổi

Trong package *test* tạo lớp *testVeMayBay* và các test case để kiểm tra tính đúng đắn của các phương thức ứng trong tất cả các trường hợp.

Bài 3: Xây dựng thứ tự thực thi của 2 class test: *testVeMayBay* và *testDaysInMonth*

3. Bài tập về nhà

Tạo Project đặt tên *STT_HoVaTen_Buoi4_VeNha*. Trong Project *STT_HoVaTen_Buoi4_VeNha*, tạo hai package: *demo1* và *test1*

Bài 1: Trong package *demo1* tạo lớp *tamGiac*, định nghĩa phương thức kiểm tra a, b, c có là độ dài ba cạnh của một tam giác. Nếu đúng thì cho biết đó là tam giác gì?

Trong package *test1* tạo lớp *test tamGiac* và các test case để kiểm tra tính đúng đắn của các phương thức ứng trong tất cả các trường hợp.

Bài 2: Trong package *demo1* 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 package *test1* tạo lớp *testPhuongTrinhBacHai* và các test case để kiểm tra tính đúng đắn của các phương thức ứng trong tất cả các trường hợp.

Bài 3: Trong package *demo1* tạo lớp *tinhPhiBaoHiemXeHoi*, định nghĩa phương thức kiểm tra tính phí bảo hiểm xe hơi như sau:

- Đối với nữ < 65 tuổi, phí bảo hiểm là 500\$
- Đối với Nam < 25 tuổi, phí bảo hiểm là 3000\$
- Đối với nam từ 25 đến 64 tuổi, phí bảo hiểm là 1000\$

- Đối với tuổi ≥ 65 , phí bảo hiểm là 1500\$

Trong package *test1* tạo lớp *testPhiBaoHiemXeHoi* và các test case để kiểm tra tính đúng đắn của các phương thức ứng trong tất cả các trường hợp.

Bài 4: Viết chương trình giúp Arthur ước lượng được khả năng thắng bại của mỗi hiệp sĩ trước mỗi cuộc chiến.

Cài đặt và sử dụng Junit, thiết kế testcase cho các trường hợp bên dưới.

VUA ARTHUR VÀ CÁC HIỆP SĨ BÀN TRÒN

A. Giới thiệu



Vua Arthur là một vị vua huyền thoại trong lịch sử nước Anh, là người dẫn dắt các hiệp sĩ Bàn Tròn (Knights of the Round Table) chuyên chiến đấu để giữ gìn và bảo vệ công lý. Gần như bất khả chiến bại trên chiến trường, vua Arthur và các hiệp sĩ Bàn Tròn chỉ gặp các đối thủ xứng tầm khi đương đầu cùng các chiến binh Saxon, được dẫn dắt bởi vua Cerdic, trong cuộc chiến quyết định ngai vàng của nước Anh. Theo tình thần thượng võ, họ quyết định sẽ tổ chức các cuộc đấu tay đôi giữa các đại diện của hai bên để giải quyết mọi vấn đề tranh chấp.

Các hiệp sĩ Bàn Tròn và các chiến binh Saxon đều là các dũng sĩ thiện chiến, mỗi trận đánh tay đôi giữa họ luôn là một cuộc chiến khốc liệt mà sự thắng bại đôi khi không chỉ quyết định bởi tài nghệ của các đấu sĩ mà còn bởi các yếu tố khác như vũ khí, áo giáp, địa hình, v.v... Thông thường, một trận chiến như vậy chỉ kết thúc khi một trong hai đấu sĩ tử thương.

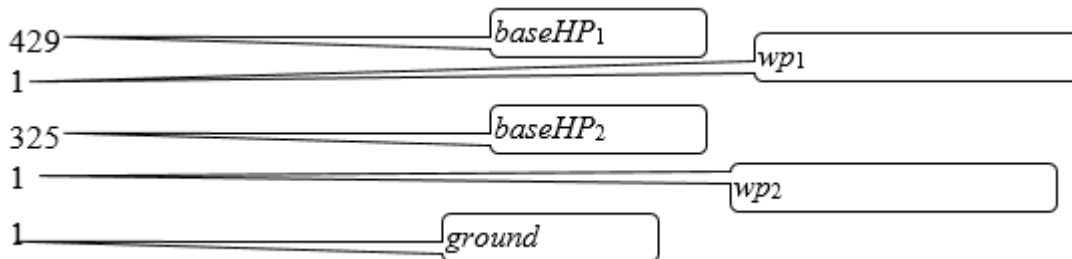
Do vậy, trước mỗi trận chiến tay đôi, vua Arthur luôn lo lắng cho số phận của hiệp sĩ Bàn Tròn anh em của mình. Vua Arthur không biết rằng vào khoảng 1500 năm sau, máy tính có thể giúp ông ước lượng được khả năng thắng bại của mỗi hiệp sĩ trước mỗi cuộc chiến.

B. Yêu cầu

Trong bài tập này, sinh viên sẽ được cung cấp chứa dữ liệu nhập, bao gồm các thông số cho một trận chiến tay đôi giữa một hiệp sĩ Bàn Tròn và một chiến binh Saxon. Chương trình sẽ tính toán và in ra màn hình xác suất chiến thắng của hiệp sĩ Bàn Tròn.

C. Dữ liệu nhập

Thông tin về hiệp sĩ Bàn Tròn và chiến binh Saxon tham gia vào trận chiến tay đôi. Cấu trúc như sau:



Giải thích:

- **baseHP₁**: Chỉ số sức mạnh cơ bản của hiệp sĩ Bàn Tròn, là một số nguyên từ 99 đến 999
- **wp₁**: Thông tin về vũ khí của hiệp sĩ Bàn Tròn, nhận một trong các giá trị 0,1,2,3
- **baseHP₂**: Chỉ số sức mạnh cơ bản của chiến binh Saxon, là một số nguyên từ 1 đến 888
- **wp₂**: Thông tin về vũ khí của chiến binh Saxon, nhận một trong các giá trị 0,1,2,3
- **ground**: Thông tin về địa hình nơi diễn ra cuộc chiến tay đôi, là một số nguyên từ 1 đến 999

E. Dữ liệu xuất

Chương trình sẽ xuất trực tiếp ra màn hình giá trị xác suất $p(R)$ dự đoán về khả năng chiến thắng của hiệp sĩ Bàn Tròn trong trận chiến tay đôi. Giá trị $p(R)$ sẽ được tính bằng công thức sau:

$$p(R) = \frac{realHP_1 - realHP_2 + 999}{2000}$$

Trong đó $realHP_1$ và $realHP_2$ là chỉ số sức mạnh thật của hiệp sĩ Bàn Tròn và chiến binh Saxon khi chiến đấu. Chỉ số sức mạnh thật này sẽ được tính dựa trên chỉ số sức mạnh cơ bản và vũ khí được sử dụng, được mô tả như sau:

- i) **wp_i = 1**: vũ khí bình thường được sử dụng. Khi đó $realHP_i = baseHP_i$
- ii) **wp_i = 0**: đấu sĩ bỏ quên vũ khí và phải chiến đấu bằng tay không. Khi đó $realHP_i = baseHP_i / 10$.

Ngoài ra đấu sĩ sẽ có thêm lợi thế nếu được chiến đấu trên địa hình quen thuộc, được mô tả như sau:

iii) Nếu $\text{ground} = \text{baseHPi}$, thì realHPi sẽ được tăng thêm 10% sau khi đã tính điểm vũ khí như đã mô tả ở mục i và ii. Tuy nhiên nếu realHPi vượt quá 999 thì sẽ được tự động giảm xuống giá trị 999

–**Ví dụ 1(Testcase1)**: Nếu $\text{baseHP1} = 450$, $\text{wp1} = 1$, $\text{baseHP2} = 150$, $\text{wp2} = 1$, $\text{ground} = 302$, giá trị in ra màn hình sẽ là $(450-150+999)/2000 = 0.65$.

–**Ví dụ 2(Testcase2)**: Nếu $\text{baseHP1} = 807$, $\text{wp1} = 0$, $\text{baseHP2} = 750$, $\text{wp2} = 1$, $\text{ground} = 156$, giá trị in ra màn hình sẽ là $(80-750+999)/2000 = 0.16$.

–**Ví dụ 3(Testcase3)**: Nếu $\text{baseHP1} = 417$, $\text{wp1} = 1$, $\text{baseHP2} = 416$, $\text{wp2} = 0$, $\text{ground} = 417$, giá trị in ra màn hình sẽ là $(417*1.1-41+999)/2000 = 0.71$.

–**Ví dụ 4(Testcase4)**: Nếu $\text{baseHP1} = 235$, $\text{wp1} = 1$, $\text{baseHP2} = 624$, $\text{wp2} = 0$, $\text{ground} = 624$, giá trị in ra màn hình sẽ là $(235-62*1.1+999)/2000 = 0.58$.

–**Ví dụ 5(Testcase5)**: Nếu $\text{baseHP1} = 998$, $\text{wp1} = 1$, $\text{baseHP2} = 517$, $\text{wp2} = 1$, $\text{ground} = 998$, giá trị in ra màn hình sẽ là $(999-517+999)/2000 = 0.74$.

Ngoài ra chương trình còn chấp nhận một số trường hợp dữ liệu đặc biệt như sau:

iv) Nếu $\text{wpi} = 2$, vũ khí được sử dụng bao gồm một áo giáp đặc biệt được làm từ mithril. Không vũ khí nào có thể xuyên thủng được áo giáp này, vì vậy đấu sĩ mặc áo giáp này sẽ không bao giờ thua trận. Trong trường hợp này realHPi của đấu sĩ vẫn được tính qua các mô tả ở mục i và mục iii và kết quả in ra màn hình vẫn được tính như cũ; tuy nhiên nếu realHP1 thấp hơn chỉ số tương ứng của đối thủ, giá trị in ra màn hình sẽ là 0.50 (trận đấu hoà).

–**Ví dụ 6**: Nếu $\text{baseHP1} = 238$, $\text{wp1} = 2$, $\text{baseHP2} = 113$, $\text{wp2} = 1$, $\text{ground} = 145$, giá trị in ra màn hình sẽ là $(238-113+999)/2000 = 0.56$.

–**Ví dụ 7**: Nếu $\text{baseHP1} = 738$, $\text{wp1} = 1$, $\text{baseHP2} = 45$, $\text{wp2} = 2$, $\text{ground} = 26$, giá trị in ra màn hình sẽ là 0.50.

–**Ví dụ 6(Testcase6)**: Nếu $\text{baseHP1} = 238$, $\text{wp1} = 2$, $\text{baseHP2} = 113$, $\text{wp2} = 1$, $\text{ground} = 145$, giá trị in ra màn hình sẽ là $(238-113+999)/2000 = 0.01$.

–**Ví dụ 7(Testcase7)**: Nếu $\text{baseHP1} = 738$, $\text{wp1} = 1$, $\text{baseHP2} = 45$, $\text{wp2} = 2$, $\text{ground} = 26$, giá trị in ra màn hình sẽ là 0.50.

v) Nếu $\text{wpi} = 3$, vũ khí sử dụng là gươm Excalibur. Đối với một chiến binh Saxon, thanh gươm này cũng chỉ là một vũ khí bình thường, vì vậy chỉ số realHP2 vẫn được tính bình thường sử dụng mô tả ở mục i và iii. Nếu một hiệp sĩ Bàn Tròn sử dụng thanh gươm này, chỉ số realHP1 sẽ được nhân đôi sau khi đã tính như đã mô tả ở mục i và iii. Tuy nhiên nếu chỉ số realHP1 sau khi tính toán cao hơn 999, nó sẽ được tự động giảm xuống 999.

–**Ví dụ 8(Testcase8)**: Nếu $\text{baseHP1} = 414$, $\text{wp1} = 1$, $\text{baseHP2} = 415$, $\text{wp2} = 3$, $\text{ground} = 199$, giá trị in ra màn hình sẽ là $(414-415+999)/2000 = 0.50$.

–**Ví dụ 9(Testcase9)**: Nếu $\text{baseHP1} = 221$, $\text{wp1} = 3$, $\text{baseHP2} = 600$, $\text{wp2} = 1$, $\text{ground} = 221$, giá trị in ra màn hình sẽ là $(221*1.1*2-600+999)/2000 = 0.44$.

–Ví dụ 10(Testcase10): Nếu $\text{baseHP1} = 612$, $\text{wp1} = 3$, $\text{baseHP2} = 800$, $\text{wp2} = 1$, $\text{ground} = 800$, giá trị in ra màn hình sẽ là $(999 - 800 * 1.1 + 999) / 2000 = 0.56$.

vi) Nếu $\text{baseHP1} = 999$, đích thân Vua Arthur ra trận. Kết quả là Arthur luôn luôn thắng bất chấp đối thủ là ai và sử dụng vũ khí gì. Kết quả in ra màn hình là 1.

(Testcase11)

vii) Nếu $\text{baseHP2} = 888$, đích thân Cerdic ra trận. Không hiệp sĩ Bàn Tròn nào đủ sức chống lại Cerdic, dù sử dụng bất kỳ vũ khí gì. Trong trường hợp này kết quả in ra màn hình là 0.00. Tuy nhiên nếu đối thủ của Cerdic là Vua Arthur, kết quả sẽ được xử lý như mô tả ở mục vi.

(Testcase13)

(Testcase14)

viii) Nếu baseHPi là một số nguyên tố, đầu sĩ tương ứng thực chất là một Paladin cải trang. Vì Paladin có khả năng dùng phép thuật khi giao tranh, nên chắc chắn sẽ luôn chiến thắng bất chấp đối phương dùng vũ khí gì. Paladin chỉ thua khi gặp Arthur, Cerdic hoặc một Paladin có chỉ số baseHP cao hơn. Nếu Paladin chiến thắng là một hiệp sĩ Bàn Tròn, màn hình sẽ in ra giá trị 0.99; nếu Paladin chiến thắng là một chiến binh Saxon, màn hình sẽ in ra giá trị 0.01. Trường hợp hai Paladin có chỉ số baseHP bằng nhau, màn hình sẽ in ra giá trị 0.50. Trường hợp Paladin gặp Arthur hoặc Cerdic, kết quả được in ra như mô tả ở mục vi và vii.

(Testcase15)

(Testcase16)

(Testcase17)

(Testcase18)

--HẾT--