KIỂM ĐỊNH CHẤT LƯỢNG PHẦN MỀM

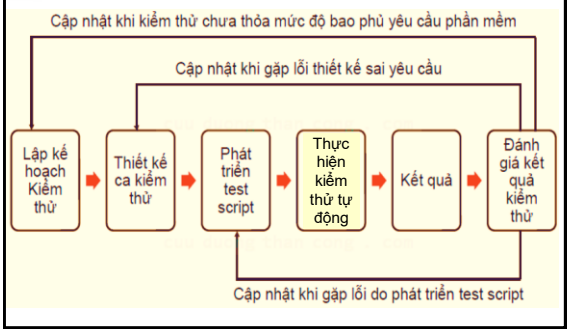
Bộ môn : Công nghệ Phần mềm
Team : Software Testing

Chương 4:
KIỂM THỬ VỚI
NUnit (*Dành cho Visual Studio*)
VÀ
JUnit (*Dành cho Java – tích hợp trong Eclipse và Net Bean*)

Kiểm thử Tự động

- **Kiểm thử tự động:** áp dụng các công cụ giúp thực hiện việc kiểm thử phần mềm.
- **Nên sử dụng công cụ tự động khi:**
  - Không đủ tài nguyên
  - Kiểm thử hồi quy
  - Kiểm tra khả năng vận hành của phần mềm trong môi trường đặc biệt.
- **Test script:** là nhóm mã lệnh đặc tả kịch bản dùng để tự động hóa một trình tự kiểm thử.
- **Test script:** có thể tạo thủ công hoặc tạo tự động dùng công cụ kiểm thử tự động.

Quy trình Kiểm thử Tự động



Quy trình Kiểm thử Tự động

1. **Tạo test script:** Giai đoạn này ta dùng test tool để ghi lại các thao tác lên Phần mềm cần kiểm tra và tự động sinh ra test script.
2. **Chỉnh sửa test script:** chỉnh sửa lại test script thực hiện kiểm tra theo đúng yêu cầu đặt ra, cụ thể là làm theo test case cần thực hiện.
3. **Chạy test script để kiểm thử tự động:** Giám sát hoạt động kiểm tra phần mềm của test script.
4. **Đánh giá kết quả:** Kiểm tra kết quả thông báo sau khi thực hiện kiểm thử tự động. Sau đó bổ sung, chỉnh sửa những sai sót.

Ưu điểm và Nhược điểm của Kiểm thử Tự động

Ưu điểm:

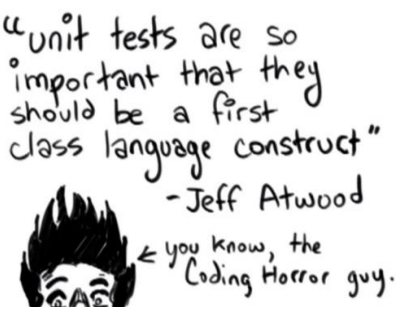
- Kiểm thử phần mềm không cần can thiệp của Tester.
- Giảm chi phí thực hiện kiểm tra số lượng lớn các test case hoặc test case lặp lại nhiều lần.
- Giả lập tình huống khó có thể thực hiện bằng tay.

Ưu điểm và Nhược điểm của Kiểm thử Tự động

Nhược điểm:

- Mất chi phí tạo các script để thực hiện kiểm thử tự động.
- Tốn chi phí dành cho bảo trì các script.
- Đòi hỏi tester phải có kỹ năng tạo và thay đổi script cho phù hợp testcase.
- Không áp dụng tìm được các lỗi mới cho phần mềm.

Unit Test



Unit Test

Một số đặc điểm của Unit Test:

- Code Unit Test phải ngắn gọn, dễ hiểu, dễ đọc.
- Mỗi Unit Test là 1 đơn vi riêng biệt, độc lập, không phụ thuộc vào Unit khác.
- Mỗi Unit Test là 1 method trong Test Class, tên method cũng là tên UnitTest. Do đó ta nên đặt tên hàm rõ ràng, nói rõ Unit Test này test cái gì (Test_***A***_Do_***B***), tên method có thể rất dài cũng không sao (Ví dụ: Test_***Case1***_Do_***Tong2SoNguyen***).

Unit Test

Một số đặc điểm của Unit Test:

- Unit Test phải nhanh, vì nó sẽ được chạy để kiểm định lỗi mỗi lần Build. Do đó trong Unit Test nên hạn chế các task tốn thời gian như gọi I/O, database, network,...
- Unit Test nên test từng đối tượng riêng biệt. Ví dụ: Unit Test cho **Business Class** thì chỉnh test chính BusinessClass đó, không nên đụng tới các class móc nối với nó (**DataAccess Class** chẳng hạn).

Kiểm thử Tự động với NUnit

- **NUnit là một Framework miễn phí được sử dụng khá rộng rãi trong Unit Testing đối với ngôn ngữ .Net.**
- **NUnit được viết hoàn toàn bằng C# và đã được hoàn toàn thiết kế lại để tận dụng lợi thế của nhiều người.**

Tạo và chạy unit test trong Visual studio

1. Tạo một Project cần kiểm tra đặt tên **CodeTest**
2. Click vào **File** menu, chọn **Add**, và chọn **New Project ....**
3. Trong New Project dialog box, chọn **Installed**, chọn **Visual C#**, và chọn **Test**.
4. Trong danh sách, chọn **Unit Test Project**.
5. Trong **Name** box, đặt tên là **UnitTest**, và chọn **OK**.
6. Trong **UnitTest** project, add reference đến **CodeUnit** (project bạn muốn viết Unit test) trong solution.

Tạo một test class


```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using CodeTest;
namespace UnitTest
{
  [TestClass]
  public class TestsDemo
  {
    [TestMethod]
    public void TestMethod1()
    {
    }
  }
}
```

Yêu cầu của một Test class:

attribute [TestClass] là bắt buộc trong Microsoft unit testing framework để được quản lý cho bất kỳ class nào chứa unit test methods.

Mỗi test method phải chứa attribute [TestMethod].

Phương thức tĩnh hay dùng trong thư viện NUnit.Framework.*Assert*

Trong lớp Assert của thư viện NUnit.Framework có một số phương thức tĩnh để có thể khẳng định tính đúng đắn cho một số điểm trong bài test:

- **Assert.AreEqual(object, object):** Là kiểm tra sự bằng nhau bởi cách gọi phương thức Equal trên đối tượng.
- **Assert.AreEqual(int, int):** Là so sánh giá trị hai số nguyên để kiểm tra bằng nhau. Test chấp nhận nếu các giá trị bằng nhau.

Phương thức tĩnh hay dùng trong thư viện NUnit.Framework.*Assert*

- **Assert.AreEqual(float, float, float):** *tương tự*
- **Assert.AreEqual(double, double, double):** *tương tự*
- **Assert.Fail():** Là hữu ích khi có một bộ test. Test sẽ chấp nhận nếu biểu thức sai.
- **Assert.IsTrue(bool):** Đánh giá một biểu thức luận lý. Test chấp nhận nếu biểu thức đúng.
- **Assert.IsFalse(bool):** Đánh giá một biểu thức luận lý. Test chấp nhận nếu biểu thức sai.

Phương thức tĩnh hay dùng trong thư viện NUnit.Framework.*Assert*

- **Assert.IsNull(bool):** So sánh tham chiếu của đối tượng với giá trị null. Test sẽ được chấp nhận nếu tham chiếu là null.
- **Assert.IsNotNull(bool):** So sánh tham chiếu của một đối tượng null. Test sẽ chấp nhận nếu biểu thức tham chiếu đối tượng khác null.
- **Assert.AreSame(object, object):** Thực thi một tham chiếu bảng trên hai đối tượng là điểm giống như đối tượng.

Phương thức tĩnh hay dùng trong thư viện NUnit.Framework.*Assert*

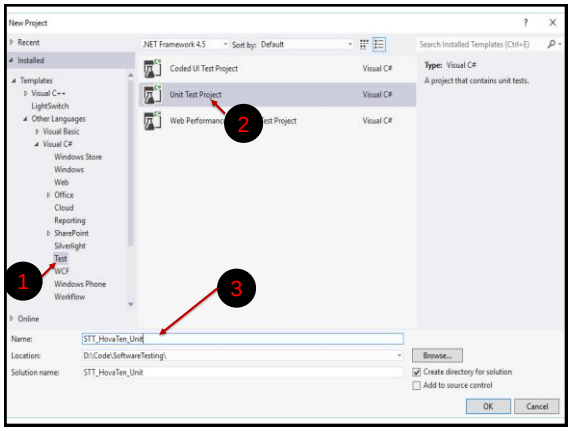
- **Assert.Ignore()**: dùng để test trạng thái nghi ngờ
- **Assert.IsEmpty():** Khẳng định một mảng, danh sách hay một bộ nào đó là rỗng.
- **Assert.IsNotEmpty():** Khẳng định một mảng, danh sách, hay một bộ nào đó là không rỗng.

Tạo và chạy UnitTest trong Visual Studio

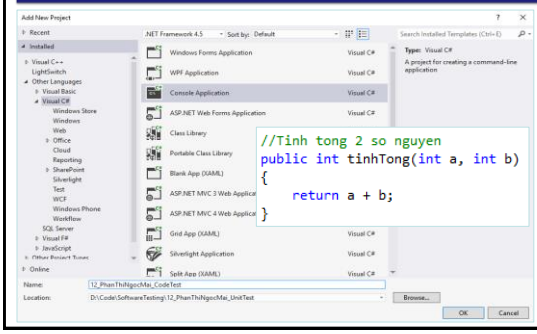
Viết UnitTest hàm tính tổng 2 số a, b

Bước 1: Tạo UnitTest

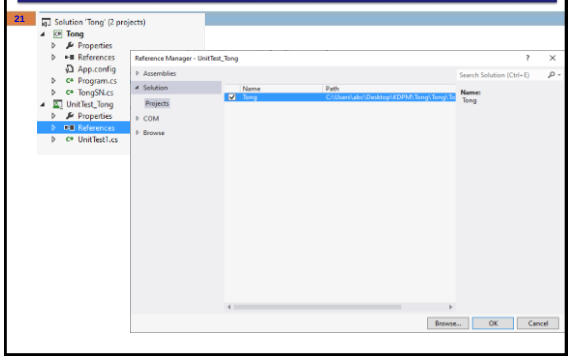
- ✓ Click chuột phải Solution chọn Add – New Project ➔ Test ➔ Unit Test Project
- ✓ Đặt tên UnitTest Project: (Ví dụ: STT_HoVaTen_Unit)



Bước 2: Tạo Project code và các phương thức cần kiểm tra. Ví dụ tạo Project: **STT_HovaTen_Code** và phương thức **tinhTong** tính tổng của 2 số nguyên



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

22 BM. CNPM

Testcase:

Failed

Passed

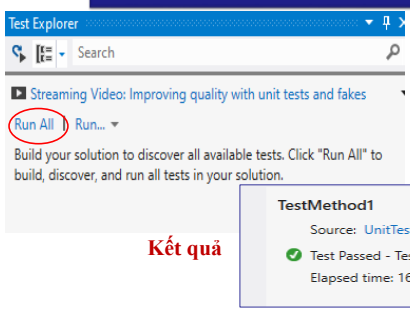
```
//Tinh tong 2 so
[TestClass]
public class UnitTest_Buoi6
{
    [TestMethod]
    public void TH1_TestTinhTong()
    {
        CodeUnit kq = new CodeUnit();
        int Result_Actual = kq.tinhTong(5, 7);
        int Result_Expect = 10;
        Assert.AreEqual(Result_Expect, Result_Actual);
    }
    [TestMethod]
    public void TH2_TestTinhTong()
    {
        CodeUnit kq = new CodeUnit();
        int Result_Actual = kq.tinhTong(5, 7);
        int Result_Expect = 12;
        Assert.AreEqual(Result_Expect, Result_Actual);
    }
```

Bước 5: Run UnitTest

Test – Windows – Test Explorer

Run Unit Test

23 BM. CNPM



Kết quả

Bài tập ứng dụng 1

24 BM. CNPM

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.

25

Tạo lớp kiểm tra tam giác

Định nghĩa phương thức kiểm tra a, b, c có là độ dài ba cạnh của một tam giác. Nếu là tam giác thì cho biết đó là tam giác gì (*thường, cân, vuông, vuông cân, đều*)?

```
//La tam giac
public Boolean laTamGiac(int a, int b, int c)
{
    Boolean kq = true;
    if ((a + b < c) || (a + c < b) || (a + b < c) || (a < 0) || (b < 0) || (c < 0)
        kq = false;
    return kq;
}
//La tam giac thuong, can, deu, vuong, vuong can
public String tamGiac(int a, int b, int c)
{
    String kq = ""; //E2
    if (laTamGiac(a, b, c))
    {
        if (a * a == b * b + c * c)
        {
            if (b == c)
                kq = "Tam giac vuong can tai a"; //E10
            else
....
```

26

UnitTest tạo các testcase

```
//Tam giac
[TestMethod]
public void TH1_TestTamGiac()
{
    CodeUnit kq = new CodeUnit();
    String Result_Actual = kq.tamGiac(-1, 4, 3);
    String Result_Expect = "Tam giac thuong";
    Assert.AreEqual(Result_Expect, Result_Actual);
}
[TestMethod]
public void TH2_TestTamGiac()
{
    CodeUnit kq = new CodeUnit();
    String Result_Actual = kq.tamGiac(1, 4, 5);
    String Result_Expect = "Tam giac thuong";
    Assert.AreEqual(Result_Expect, Result_Actual);
}
```

Bài tập ứng dụng 2

27 BM. CNPM

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

Kiểm thử Tự động với JUnit

28 BM. CNPM

- **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 người ta sẽ sử dụng method để làm Unit Test***).**
- **JUnit là một mã nguồn mở, miễn phí, được sử dụng để viết và chạy kiểm thử.**
- ***Lưu ý:*** *Hiện nay Eclipse và Net Bean đã tích hợp sẵn nên ta có thể sử dụng mà không cần phải cài đặt thêm gói JUnit.*

Các tính năng của JUnit

29 BM. CNPM

- Cung cấp các **annotation** để định nghĩa các phương thức kiểm thử.
- Cung cấp các **Assertion** để kiểm tra kết quả mong đợi.
- Cung cấp các **test runner** để thực thi các test script.
- Test case JUnit có thể được chạy tự động.
- Test case JUnit có thể được tổ chức thành các test suite.
- JUnit cho thấy kết quả test một cách trực quan: **pass** (không có lỗi) là **màu xanh** và **fail** (có lỗi) là **màu đỏ**.

Các tính năng của JUnit

30 BM. CNPM

- **Assertions JUnit:**
  - 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...

Các tính năng của JUnit

- **Định dạng của các method assert sẽ là:**

assert[kiểu so sánh] (expecteds_value, actuals_value)

Hoặc:

assert[kiểu so sánh] (expecteds_value, actuals_value, message) với message là dữ liệu in ra nếu assert thất bại

Các tính năng của JUnit

- **Ví dụ:**

```
@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());
}
```

Các tính năng của JUnit

- **Test runners:**
  - Các IDE như NetBeans, Eclipse đều có sẵn trình chạy (runner) cho JUnit để hiển thị kết quả các test case.
  - Để chỉ rõ trình runner ta có thể sử dụng **@RunWith**. Ví dụ thường dùng nhất là **@RunWith(Suite.class)** để chạy nhiều class JUnit cùng lúc.

Các tính năng của JUnit

Test Suite – Tạo bộ test với JUnit

- Thông thường 1 class test sẽ sử dụng để test cho một chức năng, một unit.
- Muốn tạo một bộ gồm nhiều class để thực hiện test và xem kết quả sau một lần chạy thì phải sử dụng test suite.

Các tính năng của JUnit

Test Suite – Tạo bộ test với JUnit

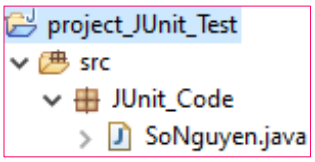
- Để tạo test suite ta sử dụng:
  - @RunWith(Suite.class) và
  - @SuiteClasses(TestClass1.class, ...).
  - Bên trong **@SuiteClasses** sẽ là các class test được chạy.

Sử dụng JUnit trong Eclipse

Tạo Project

Bước 1: Tạo một project trong java có tên **project_JUnit_Test,** một package có tên **JUnit_Code**, và một lớp có tên **SoNguyen** như sau:

Ví dụ:



Sử dụng JUnit trong Eclipse

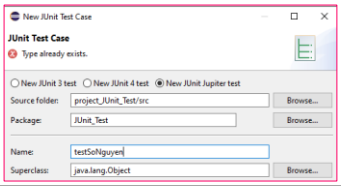
Bước 2: Trong lớp **SoNguyen** ta viết một phương thức kiểm tra một số nguyên n có phải là số nguyên tố hay không?

```
package JUnit_Code;

public class SoNguyen {
    public SoNguyen()
    {
    }

    public boolean ktSNT(int n)
    {//Hàm kiểm tra số n có phải là số nguyên tố?
        int cn = (int)Math.sqrt(n);
        for(int i = 2; i <= cn; i++)
            if(n % i == 0)
                return false;
        return true;
    }
}
```

Sử dụng JUnit trong Eclipse

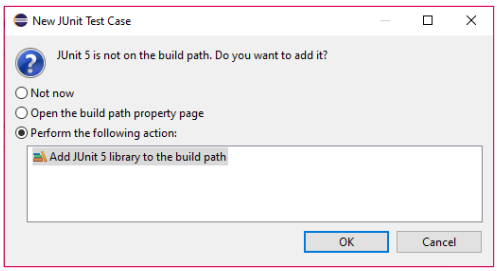
Bước 3: Trong project **project_JUnit_Test**, tạo một package có tên **JUnit_Test.**

Bước 4: Trong package **JUnit_Test** Tạo một JUnit Test có tên **testSoNguyen** dạng **JUnit Test Case** như sau:



Sử dụng JUnit trong Eclipse

Chọn như sau → nhấn nút **OK**:



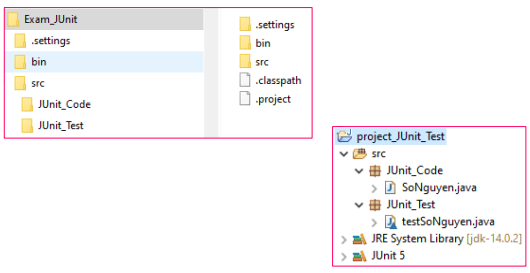
Sử dụng JUnit trong Eclipse

Ta có kết quả như sau:

```
package JUnit_Test;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class testSoNguyen {

    @Test
    void test() {
        fail("Not yet implemented");
    }

}
```

Sử dụng JUnit trong Eclipse

Ta sẽ có cấu trúc cây thư mục và Project như sau:



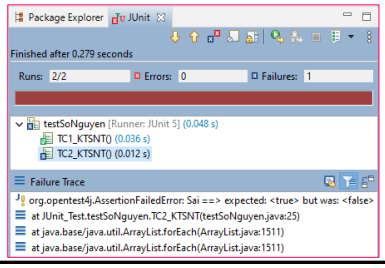
Sử dụng JUnit trong Eclipse (tt)

Ví dụ như ta viết code một số phương thức để test như sau:

```
@Test
void TC1_KTSNT() {
    int n = 4;
    SoNguyen sn = new SoNguyen();
    boolean rsActual = sn.ktSNT(n);
    boolean rsExpected = false;
    assertEquals(rsExpected, rsActual, "Đúng");
}

@Test
void TC2_KTSNT() {
    int n = 4;
    SoNguyen sn = new SoNguyen();
    boolean rsActual = sn.ktSNT(n);
    boolean rsExpected = true;
    assertEquals(rsExpected, rsActual, "Sai");
}
```

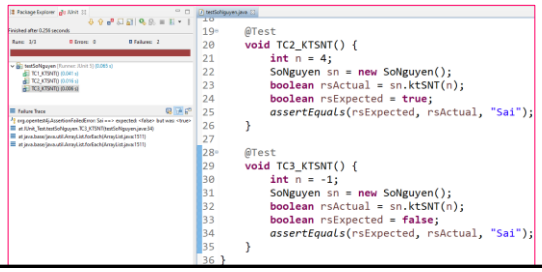
Sử dụng JUnit trong Eclipse (tt)

Nhấn chuột phải lên vùng code → Run As → JUnit Test → Kết quả chạy để kiểm thử như sau:



Sử dụng JUnit trong Eclipse (tt)

Nhưng TC3_KTSNT thì báo sai, vì **-1** không là số nguyên tố, nhưng hàm ktSNT lại không kiểm tra lỗi này.:



```
@Test
void TC2_KTSNT() {
    int n = 4;
    SoNguyen sn = new SoNguyen();
    boolean rsActual = sn.ktSNT(n);
    boolean rsExpected = true;
    assertEquals(rsExpected, rsActual, "Sai");
}

@Test
void TC3_KTSNT() {
    int n = -1;
    SoNguyen sn = new SoNguyen();
    boolean rsActual = sn.ktSNT(n);
    boolean rsExpected = false;
    assertEquals(rsExpected, rsActual, "Sai");
}
```

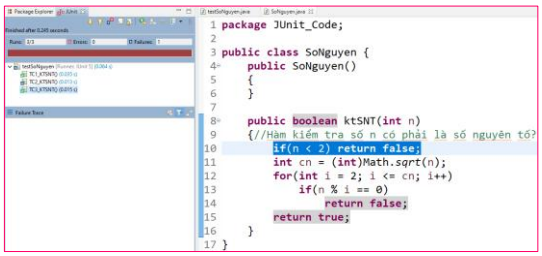
Sử dụng JUnit trong Eclipse (tt)

Nên cần phải sửa code lại như sau:

```
public boolean ktSNT(int n)
{//Hàm kiểm tra số n có phải là số nguyên tố?
    if(n < 2) return false;
    int cn = (int)Math.sqrt(n);
    for(int i = 2; i <= cn; i++)
        if(n % i == 0)
            return false;
    return true;
}
```

Sử dụng JUnit trong Eclipse (tt)

Kết quả chạy lại sẽ thì TC3_KTSNT sẽ báo đúng.



```
package JUnit_Code;

public class SoNguyen {
    public SoNguyen()
    {
    }

    public boolean ktSNT(int n)
    {//Hàm kiểm tra số n có phải là số nguyên tố?
        if(n < 2) return false;
        int cn = (int)Math.sqrt(n);
        for(int i = 2; i <= cn; i++)
            if(n % i == 0)
                return false;
        return true;
    }
}
```

Bài tập ứng dụng 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 đắng của các phương thức ứng trong tất cả các trường hợp.

Bài tập ứng dụng 2

Trong package demo1 tạo lớp giaiPhuongTrinhBacHai ($ax^2 + bx + c = 0$), định nghĩa phương thức kiểm tra các trường hợp.

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

Bài tập ứng dụng 2 (tt)

• 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 $x \in R$.

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.

Bài tập ứng dụng 3

Xây dựng chương trình tính xếp loại học tập của sinh viên dựa vào bảng qui định xếp loại và tính điểm số theo thang điểm 10 bên dưới, để kiểm chứng tính chính xác của chương trình, anh (chị) hãy:

Điểm số (thang điểm 10)	Kết quả	Điểm chữ
Điểm >=8.5 và Điểm<=10	Đạt	A
Điểm >=7.0 và Điểm<8.5		B
Điểm >=5.5 và Điểm<7.0		C
Điểm >=4.0 và Điểm<5.5		D
Điểm <4.0	Không đạt	F

Bài tập ứng dụng 3

Tạo Project đặt tên ***STT_HoVaTen_KiemTra***. Trong Project ***STT_HoVaTen_KiemTra*** hãy tạo hai package: code_XepLoaiHocTap và test_ XepLoaiHocTap.

Trong package: code_KetQuaHocTap tạo lớp xetKetQuaHocTap và Định nghĩa phương thức xét kết quả học tập dựa vào điểm trung bình tích luỹ của sinh viên.

Trong package: test_ XepLoaiHocTap tạo lớp testXepLoaiHocTap và các test case để kiểm tra tính đúng đắng của các phương thức trong tất cả các trường hợp.

KIỂM ĐỊNH CHẤT LƯỢNG PHẦN MỀM

Thank for your attention!