

Lab 02

Statements and Operators

C# Programming Constructs

Mục tiêu

- Hiểu về câu lệnh và sử dụng các toán tử
- Sử dụng các cấu trúc lập trình trong c# (if, switch, for, do, while, foreach)

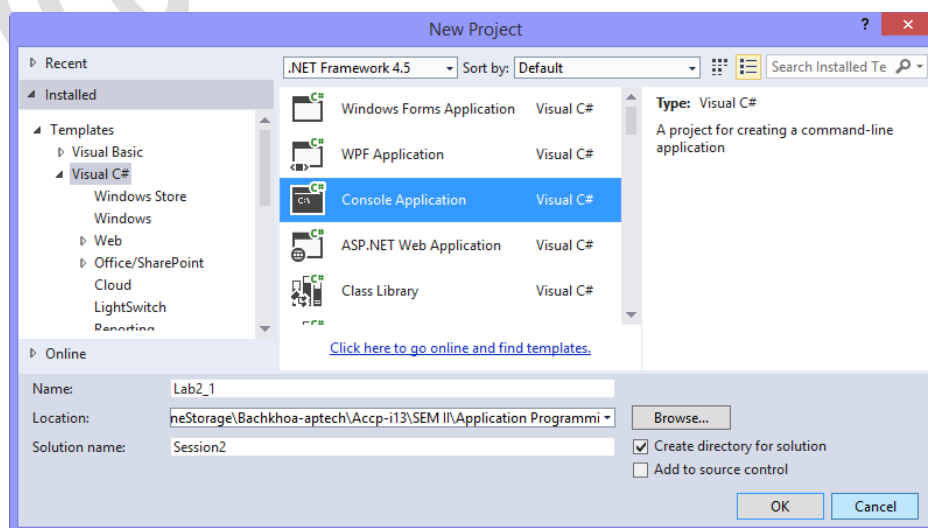
Phần I Bài tập step by step

Bài 2.1

Nhập vào tên thuê bao, số điện trên công tơ sau đó tính cước theo quy ước sau.

- Nếu số phút gọi ≤ 30 số thì cước phí tính 30\$.
- Nếu từ 30-50 số thì số trong khoảng (30-50] sẽ tính theo 1.2\$ cho mỗi số.
- Nếu trên 50 số thì số trên 50 được tính theo 1.5\$ cho mỗi số
- In ra màn hình toàn bộ thông tin.

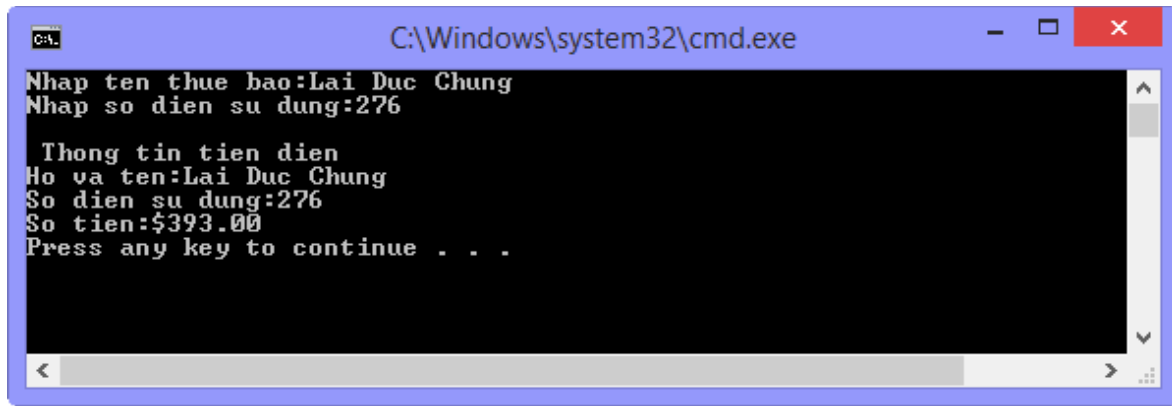
Bước 1: Mở Visual Studio 2013, vào menu File -> New -> Project -> chọn loại project “Console Application”, nhập tên project, tên solution -> OK.



Bước 2: Mở tệp Program.cs và code cho hàm Main theo gợi ý sau:

```
/// <summary>
/// Chương trình tính tiền điện
/// </summary>
/// <param name="args"></param>
static void Main(string[] args)
{
    //khai báo biến
    string name;
    int number;
    double money = 0;
    //Nhập thông tin
    Console.Write("Nhập ten thuê bao:");
    name = Console.ReadLine();
    Console.Write("Nhập số diện sử dụng:");
    number = Convert.ToInt32(Console.ReadLine());
    //tính toán số tiền
    if (number <= 30)
        money = 30;
    else if (number > 30 && number <= 50)
        money = 30 + (number - 30) * 1.2;
    else if (number > 50)
        money = 30 + 20 * 1.2 + (number - 50) * 1.5;
    //in thông tin
    Console.WriteLine("\n Thông tin tiền điện");
    Console.WriteLine("Ho va ten:{0}", name);
    Console.WriteLine("So dien su dung:{0}", number);
    Console.WriteLine("So tien:{0:C}", money);
}
```

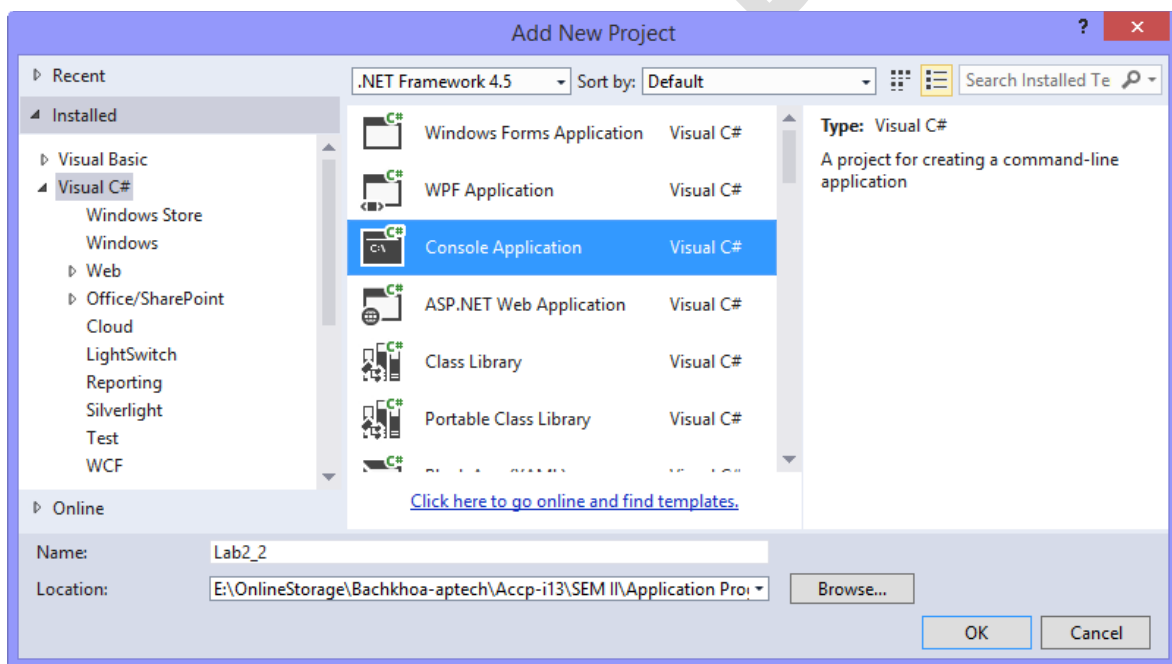
Bước 3: Nhấn Ctrl+F5 để chạy và xem kết quả



Bài 2.2

Nhập vào một ký tự, thông báo ra màn hình đó là nguyên âm hay phụ âm.

Bước 1: Kích chuột phải vào Solution “Session2” chọn Add -> New Project ->nhập tên.

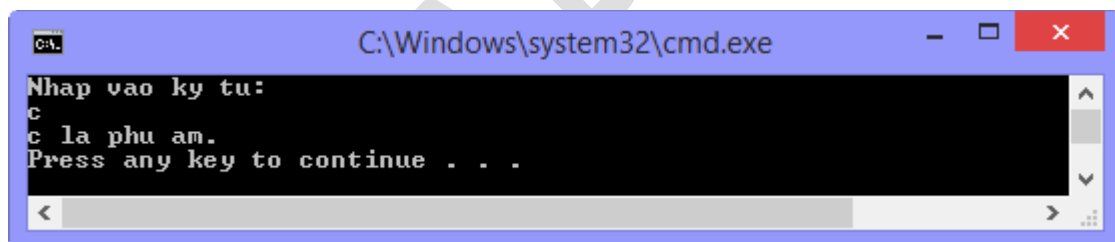


Bước 2: Mở tệp Program.cs và code cho hàm Main theo gợi ý sau:

```
/// <summary>  
/// Nhập vào 1 ký tự, in ra ký tự đó là nguyên âm hay phụ âm  
/// </summary>  
/// <param name="args"></param>  
public static void Main(String[] args)  
{
```

```
char ch;
Console.WriteLine("Nhap vao ky tu:");
ch = (char)Console.Read();
switch (ch)
{
    case 'a':
    case 'o':
    case 'e':
    case 'u':
    case 'i': Console.WriteLine("{0} la nguyen am.", ch);
    break;
    default: Console.WriteLine("{0} la phu am.", ch);
    break;
}
```

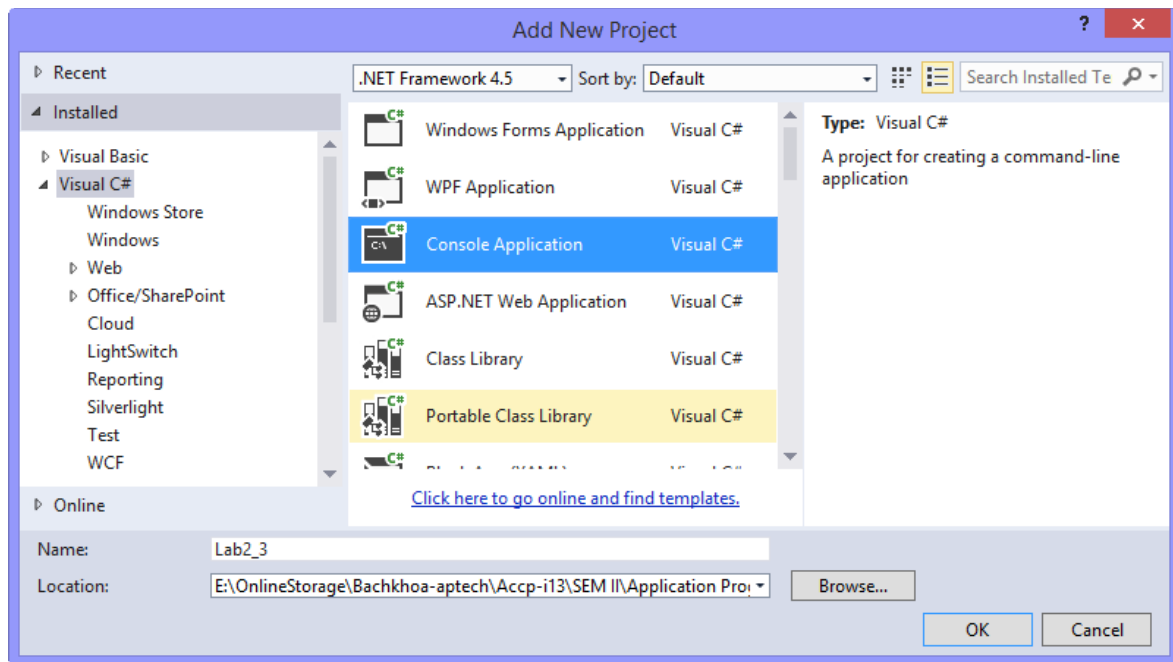
Bước 3: Nhấn Ctrl+F5 để chạy và xem kết quả



Bài 2.3

Viết chương trình C# để giải phương trình bậc 2 (lưu ý a phải luôn !=0) .

Bước 1: Kích chuột phải vào Solution "Session2" chọn Add -> New Project -> nhập tên.



Bước 2: Mở tệp Program.cs và code cho hàm Main theo gợi ý sau:

```

/// <summary>
/// Chương trình giải phương trình bậc 2
/// </summary>
/// <param name="args"></param>
static void Main(string[] args)
{
    //Khai báo các biến
    double a, b, c, delta, x1, x2;
    //Nhập a,b,c
    Console.Write("a=");
    //a phải !=0
    do
    {
        a = Convert.ToInt32(Console.ReadLine());
    }
    while (a == 0);
    Console.Write("b=");
    b = Convert.ToInt32(Console.ReadLine());
    Console.Write("c=");
    c = Convert.ToInt32(Console.ReadLine());
}

```

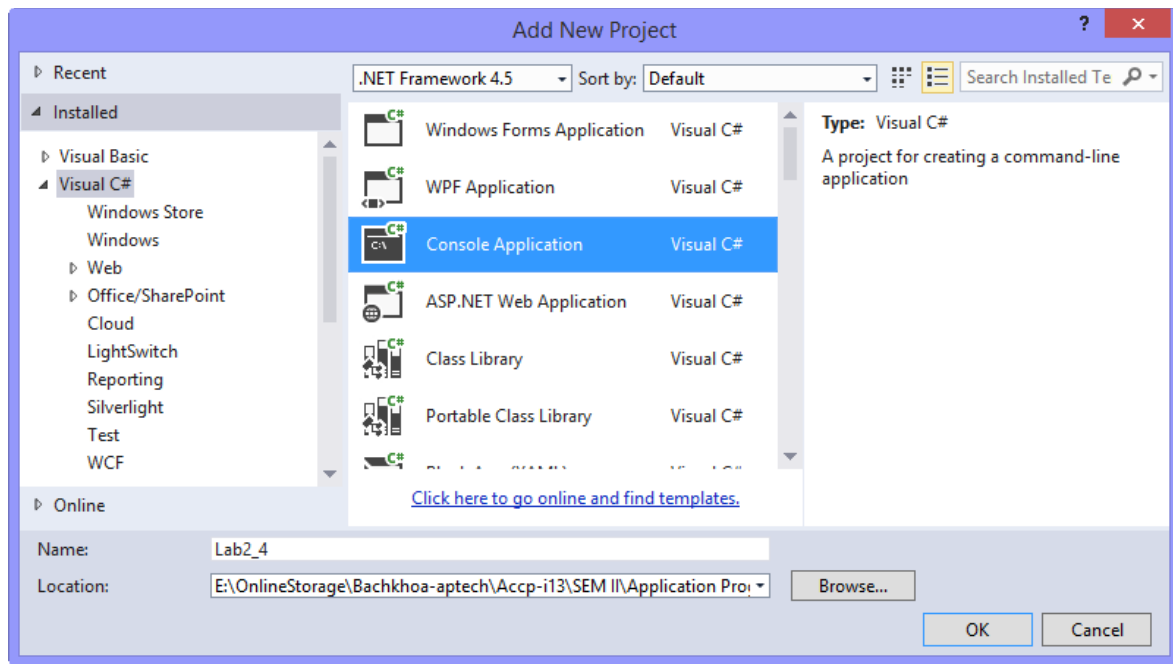
```
//tính delta
delta = b * b - 4 * a * c;
//biện luận
if (delta < 0)
    Console.WriteLine("Phương trình vô nghiệm");
else if (delta == 0)
{
    Console.WriteLine("Phương trình có nghiệm kép");
    Console.WriteLine("x1=x2={0}", -b / (2 * a));
}
else
{
    Console.WriteLine("Phương trình có 2 nghiệm");
    x1 = (-b + Math.Sqrt(delta)) / (2 * a);
    Console.WriteLine("x1={0}", x1);
    x2 = (-b - Math.Sqrt(delta)) / (2 * a);
    Console.WriteLine("x2={0}", x2);
}
}
```

Bước 3: Nhấn Ctrl+F5 để chạy và xem kết quả

Bài 2.4

Viết chương trình C# In ra các số nguyên tố từ 2-100 (chú ý: số nguyên tố là số **chỉ** chia hết cho 1 và chính nó)..

Bước 1: Kích chuột phải vào Solution “Session2” chọn Add -> New Project ->nhập tên.



Bước 2: Mở tệp Program.cs và code cho hàm Main theo gợi ý sau:

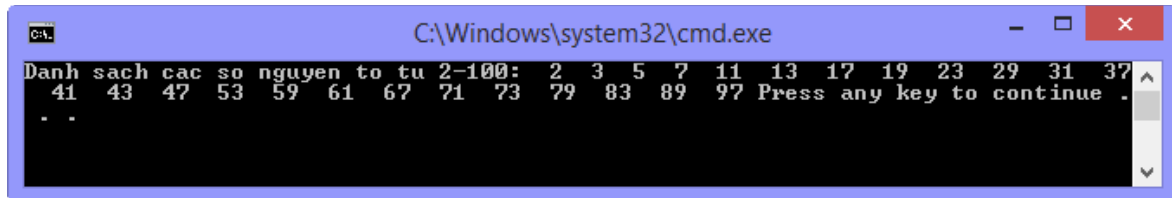
```
/// <summary>
/// Chương trình in ra các số nguyên tố từ 2-100
/// </summary>
/// <param name="args"></param>
static void Main(string[] args)
{
    //khai báo biến đánh dấu
    bool check_i;
    Console.WriteLine("Danh sach cac so nguyen to tu 2-100: ");
    //duyet từ 2-100
    for (int i = 2; i <= 100; i++)
    {
        check_i = true; //giả sử i là số nguyên tố
        for (int j = 2; j <= i / 2; j++) //duyet từ 2-j/2
        {
            if (i % j == 0) //nếu i chia hết cho j thì
            {
                check_i = false; //kết luận không là số nguyên tố
                break; //thoát khỏi vòng lặp hiện tại
            }
        }
    }
}
```

```

    }
    if (check_i) //nếu giả sử vẫn đúng -> i là số nguyên tố
        Console.WriteLine(" {0} ", i);
    }
}

```

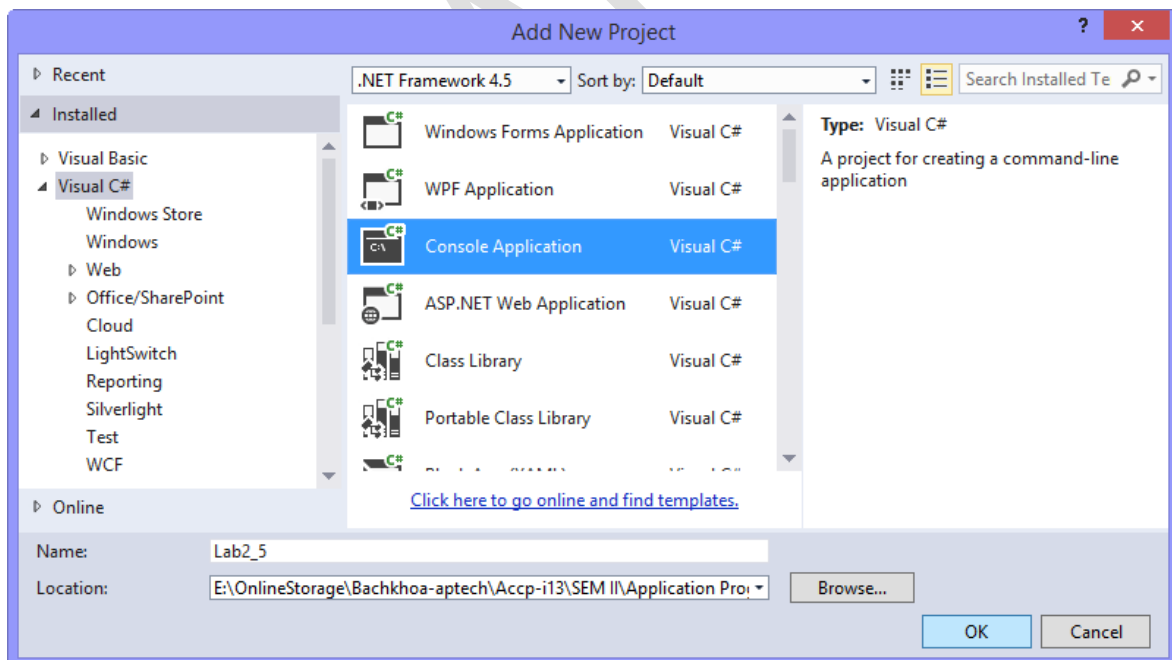
Bước 3: Nhấn Ctrl+F5 để chạy và xem kết quả



Bài 2.5

Viết chương trình C# tính tổng các số chẵn và không chia hết cho 3 từ 1-100

Bước 1: Kích chuột phải vào Solution “Session2” chọn Add -> New Project ->nhập tên.



Bước 2: Mở tệp Program.cs và code cho hàm Main theo gợi ý sau:

```

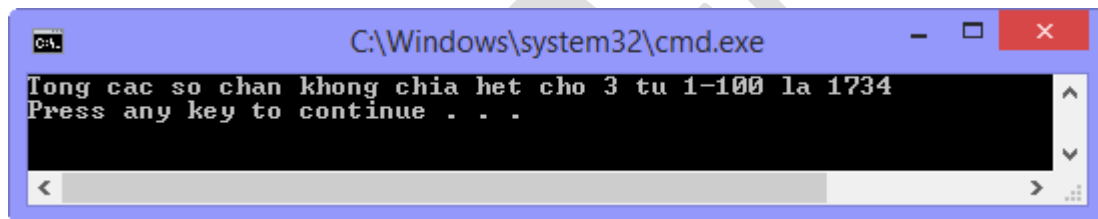
static void Main(string[] args)
{
    //khai báo biến tổng

```



```
int sum = 0;
//duyet từ 1-100
for (int i = 1; i <= 100; i++)
{
    if (i % 2 == 0 && i % 3 != 0)
    {
        sum += i;
    }
}
//in kết quả
Console.WriteLine("Tong cac so chan khong chia het cho 3 tu 1-100 la {0}", sum);
}
```

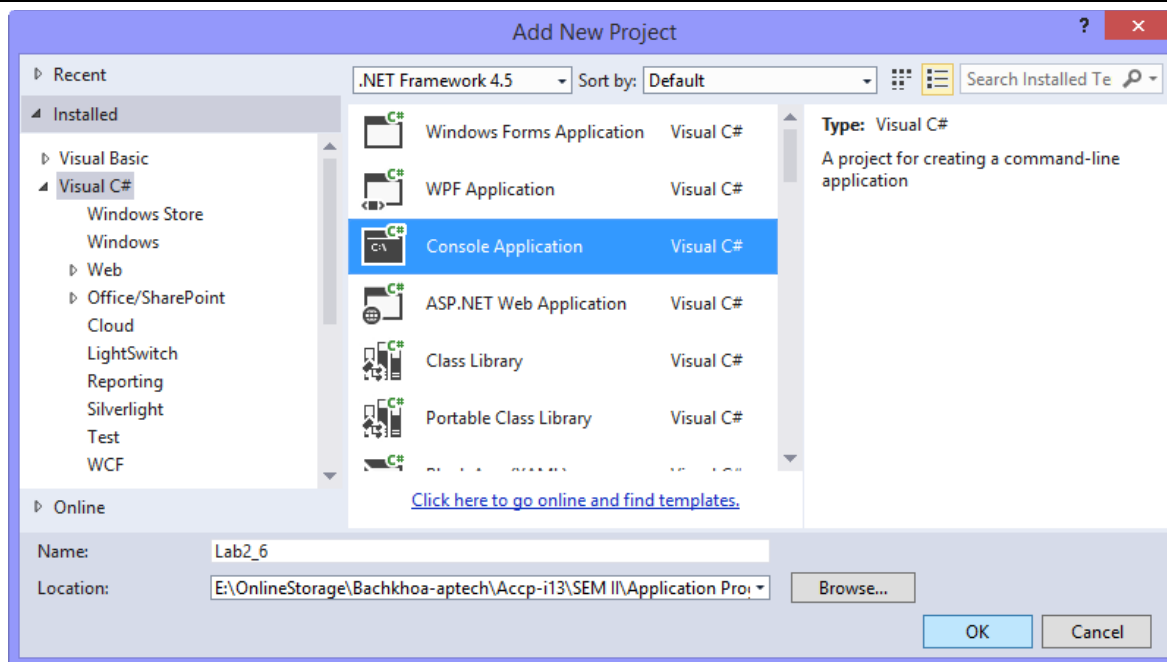
Bước 3: Nhấn Ctrl+F5 để chạy và xem kết quả



Bài 2.6

Viết chương trình C# khai báo và khởi tạo một mảng tên, dùng foreach để in mỗi tên trên một dòng

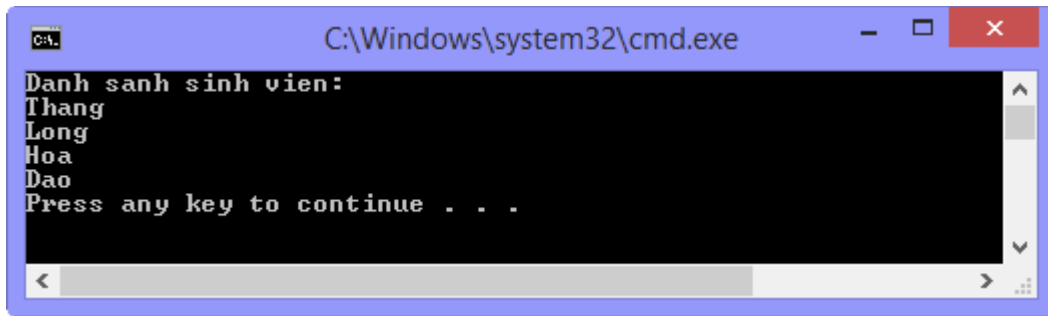
Bước 1: Kích chuột phải vào Solution "Session2" chọn Add -> New Project -> nhập tên.



Bước 2: Mở tệp Program.cs và code cho hàm Main theo gợi ý sau:

```
/// <summary>
/// Sử dụng foreach in ra danh sách các phần tử của mảng
/// </summary>
/// <param name="args"></param>
static void Main(string[] args)
{
    //Khai báo và khởi tạo mảng tên
    string[] names = {"Thang", "Long", "Hoa", "Dao" };
    Console.WriteLine("Danh sanh sinh vien:\n");
    foreach (var n in names)
    {
        Console.WriteLine(n);
    }
}
```

Bước 3: Nhấn Ctrl+F5 để chạy và xem kết quả



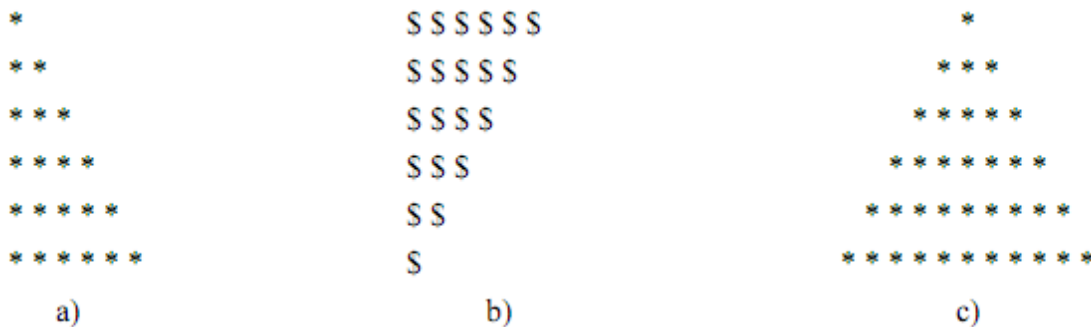
Phần II Bài tập tự làm

Bài 2.1: Viết chương trình nhập vào năm x tháng y sau đó in ra số ngày trong tháng năm đó.

Bài 2.2: Viết chương trình nhập vào số nguyên dương là số giây, in ra định dạng sau hh:mm:ss . Ví dụ số giây nhập vào là 350 thì in ra là 00:05:50.

Bài 2.3: Viết chương trình in ra các số tổng 3 ký số là chẵn từ 100-999.

Bài 2.4: Viết chương trình in ra các hình sau.



Bài 2.5: Viết chương trình nhập vào 3 số nguyên a, b, c.

- Xét xem a,b,c có tạo thành độ dài 3 cạnh của một tam giác không?
- Nếu là a,b,c là độ dài 3 cạnh của tam giác thì xét xem tam giác gì.

Bài 2.6: Viết chương trình csharp hiển thị ra hình sau:

1
2 3 2
3 4 5 4 3
4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5
6 7 8 9 0 1 0 9 8 7 6
7 8 9 0 1 2 3 2 1 0 9 8 7
8 9 0 1 2 3 4 5 4 3 2 1 0 9 8
9 0 1 2 3 4 5 6 7 6 5 4 3 2 1 0 9
0 1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1 0

Bài 2.7: Viết chương trình Csharp với các yêu cầu sau.

- Nhập vào số tiền gửi, lãi suất ngân hàng (tính theo năm) và số tháng gửi.
- Tính và xuất số dư cuối kỳ và tiền lãi cuối kỳ, biết rằng:
 - o $\text{Lãi suất tháng} = (\text{Lãi suất năm} / 12) / 100$
 - o $\text{Tiền lãi tháng} = \text{Tiền gốc} * \text{Lãi suất tháng}$
 - o Tiền lãi mỗi tháng sẽ được gộp chung vào tiền gốc
 - o Tiền lãi cuối kỳ sẽ được tích lũy tiến từ tiền lãi mỗi tháng + tiền gốc.

Ví dụ: Nếu bạn gửi 10 triệu với lãi suất 12%/năm và gửi trong 4 tháng, thì tiền lãi cuối kỳ được tính như sau:

- o $\text{Lãi suất tháng} = (12 / 12) / 100 = 0.01$

Tháng	Tiền gốc + lãi	Tiền lãi tháng
1	10,000,000	100,000
2	10,100,000	101,000
3	10,201,000	102,010
4	10,303,010	103,030.1

Bài 2.8: Đọc các nội dung tham khảo sau để nâng cấp level

Giới thiệu về lập trình hướng đối tượng (Object Oriented Programming)

Lập trình hướng đối tượng (OOP) là một trong những kỹ thuật lập trình rất quan trọng hiện nay. Nó được áp dụng ở hầu hết các ứng dụng thực tế xây dựng tại các doanh nghiệp. Hầu hết các ngôn ngữ lập trình và framework lập trình phổ biến hiện nay như Java, PHP, .NET đều hỗ trợ lập trình hướng đối tượng. Các lập trình viên đa phần đã được học về lập trình hướng đối tượng ở trường đại học nhưng các nguyên lý cơ bản của lập trình hướng đối tượng đôi khi lại không nắm rõ dẫn đến sử dụng sai, không đúng triết lý của lập trình hướng đối tượng.

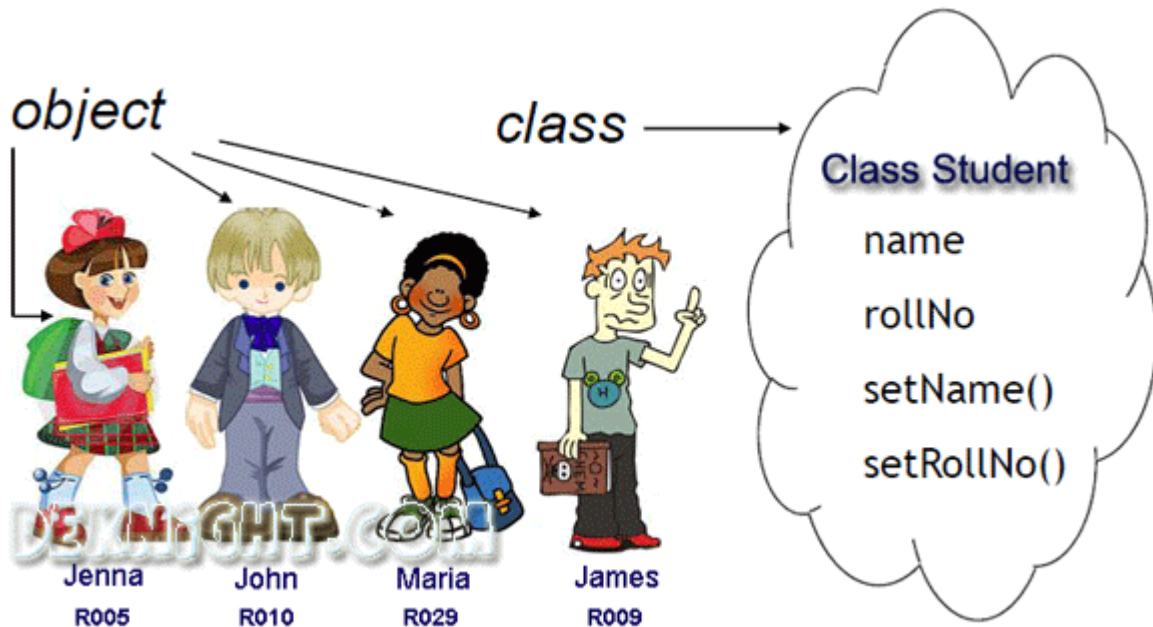
Trong bài viết này, tôi sẽ tóm lược lại các nguyên lý cơ bản của lập trình hướng đối tượng nhằm giúp các bạn có được một cái nhìn tổng quát về OOP cũng như cách áp dụng nó.

Lập trình hướng đối tượng là gì?

Lập trình hướng đối tượng là một kỹ thuật lập trình cho phép lập trình viên tạo ra các đối tượng trong code **trừu tượng hóa** các đối tượng thực tế trong cuộc sống. Hướng tiếp cận này hiện đang rất thành công và đã trở thành một trong những khuôn mẫu phát triển phần mềm, đặc biệt là các phần mềm cho doanh nghiệp.

Khi phát triển ứng dụng sử dụng OOP, chúng ta sẽ định nghĩa các lớp (**class**) để mô hình các đối tượng thực tế. Trong ứng dụng **các lớp** này sẽ được **khởi tạo** thành các **đối tượng** và trong suốt thời gian ứng dụng chạy, các phương thức (**method**) của những đối tượng này sẽ được gọi.

Lớp định nghĩa đối tượng sẽ như thế nào: gồm những **phương thức** và thuộc tính (**property**) gì. Một đối tượng chỉ là một thể hiện của lớp. Các lớp tương tác với nhau bởi các **public API**: là tập các **phương thức, thuộc tính public** của các lớp.



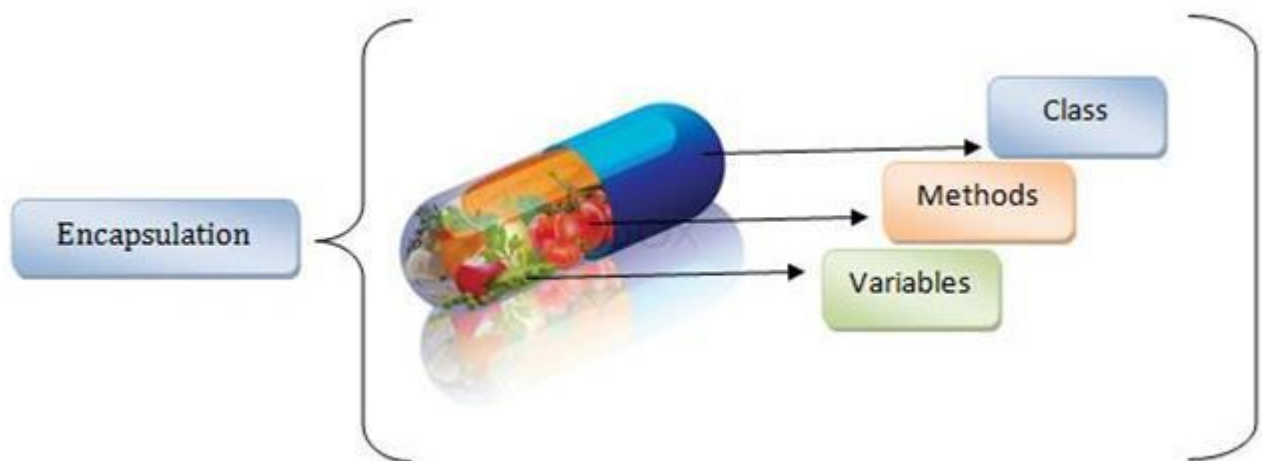
OOP có **3 nguyên lý cơ bản** chúng ta sẽ cùng tìm hiểu chi tiết sau đây đó là:

Tính đóng gói (Encapsulation)

Tính đóng gói tức là quy tắc yêu cầu trạng thái bên trong của một đối tượng được bảo vệ và tránh truy cập được từ các code bên ngoài (tức là các code bên ngoài không thể trực tiếp nhìn thấy và thay đổi trạng thái của một đối tượng). Bất cứ truy cập nào tới trạng thái bên trong này bắt buộc phải thông qua một public API để đảm bảo trạng thái của đối tượng luôn hợp lệ bởi vì các public API đảm bảo tất cả các quy tắc kiểm tra tính hợp lệ cũng như trình tự thực hiện được áp dụng mỗi khi thay đổi trạng thái đó.

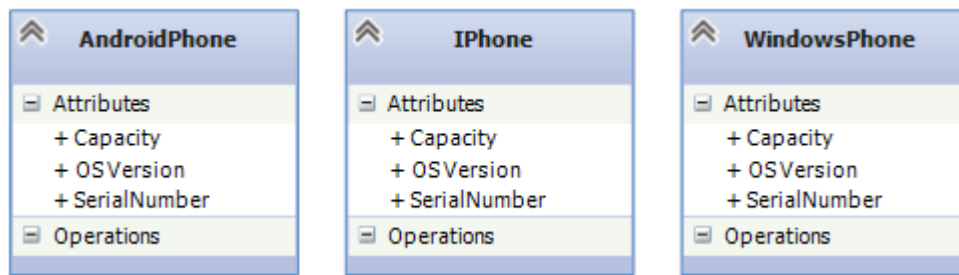
Vì trạng thái đối tượng không hợp lệ thường do: chưa được kiểm tra tính hợp lệ, các bước thực hiện không đúng trình tự hoặc bị bỏ qua nên **trong OOP có một quy tắc quan trọng cần nhớ đó là phải luôn khai báo các trạng thái bên trong của đối tượng là private và chỉ cho truy cập qua các public, protected method/property**. Khi sử dụng các đối tượng ta không cần biết bên trong nó làm việc như thế nào, ta chỉ cần biết các public API là gì và điều này đảm bảo những gì thay đổi đối tượng sẽ được kiểm tra bởi các quy tắc logic bên trong, tránh đối tượng bị sử dụng không chính xác.

Cũng giống như viên thuốc, chúng ta chỉ biết nó chữa bệnh này, bệnh kia và một số thành phần chính còn cụ thể bên trong nó có những gì thì hoàn toàn không biết.



Tính kế thừa (Inheritance)

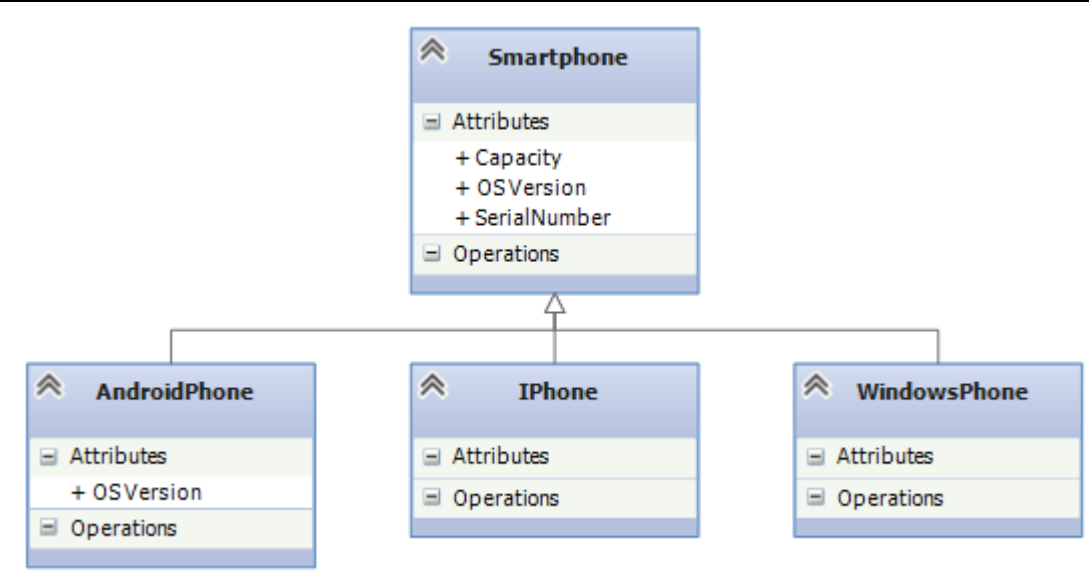
Khi bắt đầu xây dựng ứng dụng bởi các lớp, chúng ta thường thấy trường hợp một số lớp dường như có quan hệ với những lớp khác, chúng khá tương đồng. VD: 3 lớp AndroidPhone, iPhone, WindowsPhone ở hình dưới:



Mỗi lớp đều đại diện cho một loại smartphone khác nhau nhưng lại có những thuộc tính giống nhau. Thay vì sao chép những thuộc tính này, sẽ hay hơn nếu ta đặt chúng ở một nơi có thể dùng bởi những lớp khác. Điều này được thực hiện bởi tính kế thừa trong OOP: chúng ta có thể định nghĩa lớp cha – base class (trong trường hợp này là Smartphone) và có những lớp con kế thừa từ nó (derived class), tạo ra một mối quan hệ cha/con như hình dưới:



Bây giờ, các lớp con có thể kế thừa 3 thuộc tính từ lớp cha. Nếu các chức năng của lớp cha đã được định nghĩa đầy đủ thì lập trình viên sẽ không phải làm bất cứ việc gì với lớp con. Còn nếu một lớp con muốn chức năng khác so với định nghĩa ở lớp cha thì nó có thể dễ dàng ghi đè (**override**) chức năng đã được định nghĩa trên lớp cha này.



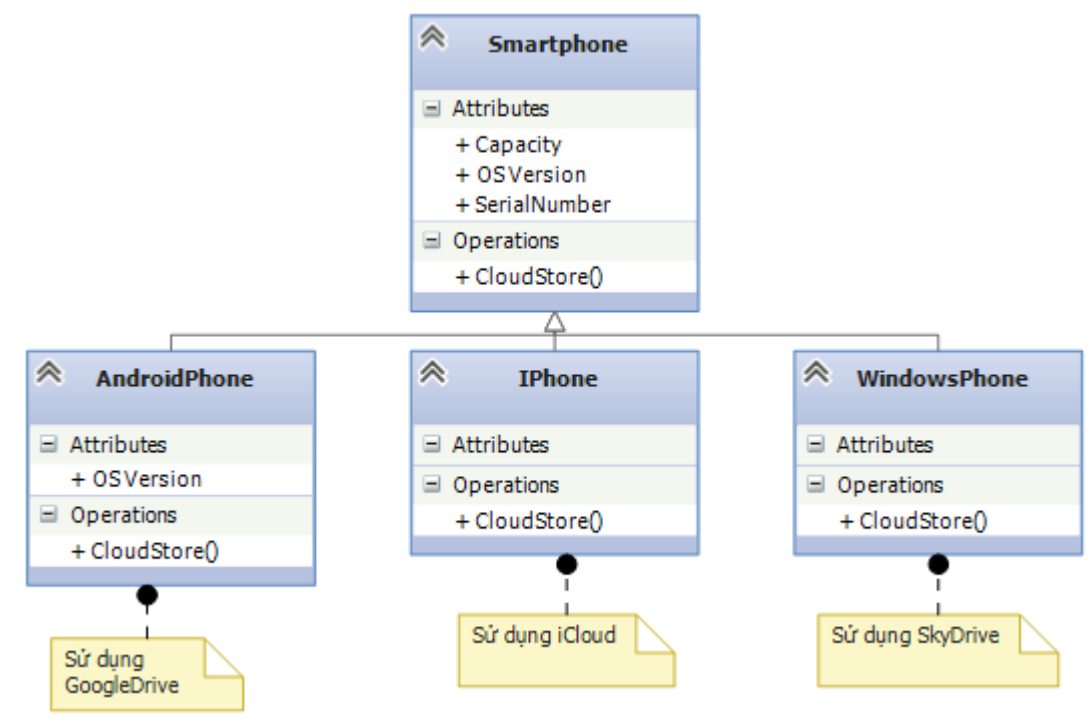
Như hình trên bây giờ nếu gọi thuộc tính **OSVersion** trên lớp **AndroidPhone** thì nó sẽ dùng định nghĩa được khai báo ở lớp này, trong khi hai lớp Iphone & Windows Phone vẫn dùng định nghĩa được khai báo ở lớp Smartphone.

Tính đa hình (Polymorphism)

Với đa số lập trình viên thì tính Kế thừa và Đóng gói trong OOP khá dễ hiểu còn tính Đa hình khi mới tiếp cận sẽ thấy khó hiểu hơn một chút. Tuy nhiên đây lại là một tính chất có thể nói là chứa đựng hầu hết sức mạnh của lập trình hướng đối tượng. Hiểu một cách đơn giản: **Đa hình là khái niệm mà hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau.**

Ví dụ như ở phần trên, mỗi một smartphone kế thừa từ lớp Smartphone nhưng có thể lưu trữ dữ liệu trên cloud theo những cách khác nhau:

- AndroidPhone lưu trữ bằng Google Drive
- Iphone lưu trên iCloud
- WindowsPhone sử dụng SkyDrive.



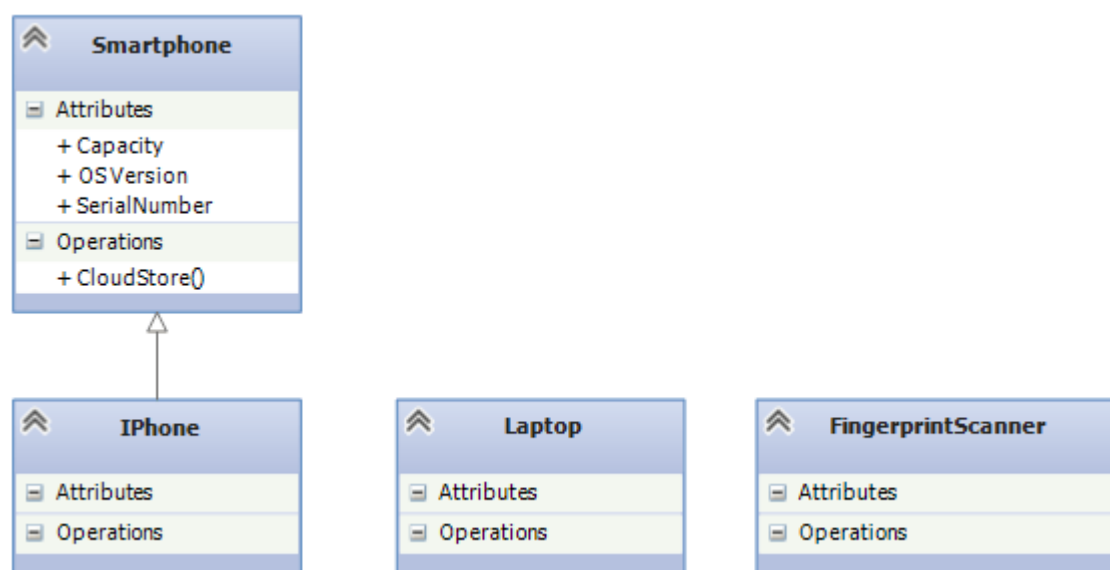
Mặc dù tất cả đều là Smartphone và nếu ta viết một hàm dùng kiểu Smartphone làm tham số thì khi gọi hàm có thể truyền vào một đối tượng kiểu AndroidPhone, Iphone hoặc WindowsPhone bởi vì chúng đều kế thừa từ lớp Smartphone nên được chấp nhận (hiểu nôm na một AndroidPhone, Iphone, WindowsPhone cũng là một Smartphone).

Hàm trên thậm chí không cần quan tâm smartphone nào được truyền vào do chúng thừa kế từ lớp Smartphone nên mọi thứ cần thiết đã có: ở đây chính là những public method/property được định nghĩa trên lớp cha Smartphone. **Nếu các lớp con không định nghĩa lại (overrides) phương thức CloudStore() thì phương thức CloudStore() trên lớp cha (Smartphone) sẽ được gọi.** Còn nếu lớp con override lại phương thức CloudStore() của lớp cha như ở hình trên thì phương thức CloudStore() trên lớp con sẽ được gọi mặc dù code trong hàm đang thao tác với đối tượng kiểu Smartphone nói chung.

Tính Đa hình như trên là một tính chất hết sức mạnh mẽ bởi vì nó mang lại cho code khả năng tổng quát hóa. Chúng ta không cần tạo ra phương thức cho mỗi kiểu kế thừa từ lớp cha Smartphone mà chỉ cần nhận một biến kiểu Smartphone và có thể làm việc với bất cứ lớp nào kế thừa từ nó. Điều duy nhất không làm được ở đây là sử dụng những phương thức mà chỉ được khai báo trên các lớp con. VD: nếu ta có một phương thức trên lớp iPhone gọi là **OpenSiri()** nhưng không được khai báo trên lớp Smartphone, khi đó muốn gọi nó sẽ bắt buộc phải ép kiểu từ Smartphone sang iPhone trước khi gọi.

Interface

Đa hình dựa trên Kế thừa không phải bao giờ cũng là lựa chọn tốt nhất. VD: Ta có biểu đồ lớp như sau

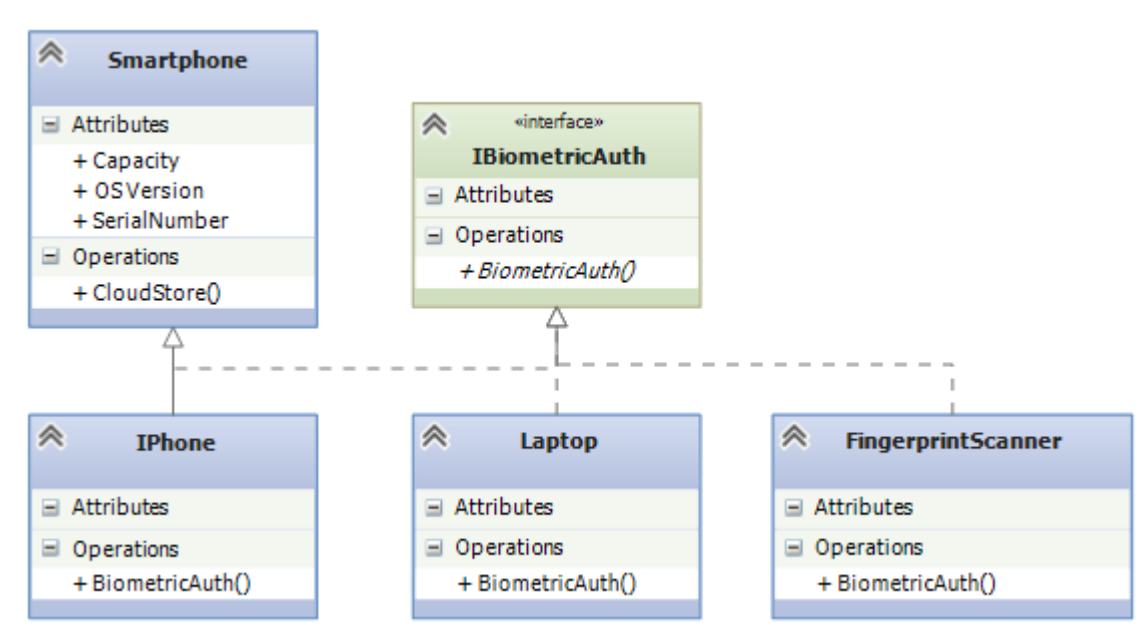


Ta thấy rõ ràng rằng 3 lớp bên dưới (Iphone, Laptop, FingerprintScanner) đều là những thứ có thể truy cập được bằng vân tay nhưng chúng thực hiện theo những cách khác nhau. Những lớp này có chung một hành động tạm gọi là định danh bằng sinh trắc học – **BiometricAuth()**. Nếu ta cố gắng gộp cả 3 lớp này vào thành 1 lớp

chung sẽ không hay vì rất khó để tìm ra điểm chung tổng quát của chúng ngoài việc có thể truy cập bằng vân tay.

Do vậy thay vì sử dụng Kế thừa ở đây, ta có thể sử dụng một kỹ thuật khác đó là Interface. **Interface đơn giản là một giao kèo chỉ ra rằng code của bạn sẽ thực thi và hỗ trợ một public API cụ thể nào đó.** Tuy nhiên các public API này được thực hiện như thế nào thì không được chỉ ra trên Interface mà sẽ được chỉ ra trên lớp thực thi interface này. Về cơ bản giao kèo là một danh sách các public method/property mà chắc chắn sẽ được thực thi trong lớp của bạn.

Áp vào VD trên, ta có thể tạo ra một interface là **IBiometricAuth** với một phương thức là **BiometricAuth()**. Tiếp theo cho các lớp Iphone, Laptop, FingerprintScanner thực thi interface **IBiometricAuth** này như hình sau



Vì mỗi lớp trên đều thực thi interface **IBiometricAuth** nên ta có thể đảm bảo rằng chúng đều có phương thức **BiometricAuth()** và khai báo của phương thức sẽ giống y như được định nghĩa trên interface **IBiometricAuth**. Tương tự Kế thừa dựa trên Đa

hình, sử dụng Interface cho phép chúng ta khai báo phương thức nhận tham số kiểu **IBiometricAuth** nhưng chấp nhận bất cứ đối tượng nào truyền vào mà kiểu của nó thực thi interface **IBiometricAuth** này. Những lớp thực thi **IBiometricAuth** không cần phải có chung lớp cha ngoại trừ interface **IBiometricAuth**. Trong phương thức ở trên, ta có thể gọi bất cứ phương thức đã được định nghĩa trên interface **IBiometricAuth** của đối tượng truyền vào mà không cần quan tâm kiểu thực sự của nó là gì: Không cần quan tâm nó là Iphone, Laptop hay FingerprintScanner, chỉ cần biết nó hỗ trợ interface **IBiometricAuth** vì thế có thể gọi được phương thức **BiometricAuth()**.

HẾT