Buổi 4.2
Tính đa hình

I. Tính đa hình

Xét về ngữ nghĩa, "đa hình" có nghĩa là "nhiều hình dạng khác nhau". Trong lập trình, tính đa hình nghĩa là cùng 1 thứ nhưng có thể có nhiều cách cài đặt, thể hiện khác nhau.

Có 2 cách để thể hiện tính đa hình:

- Đa hình tĩnh (static polymorphism): Còn gọi là tính đa hình tại thời điểm biên dịch (compile-time). Kiểu đa hình này cho phép dùng chung 1 tên phương thức để cài đặt theo nhiều cách khác nhau thông qua 2 kỹ thuật:
  - Nạp chồng phương thức (**Method Overloading**).
  - Nạp chồng toán tử (**Operator Overloading**).
- Đa hình động (dynamic polymorphism): Còn gọi là tính đa hình tại thời điểm thực thi (run-time). Kiểu đa hình này cho phép dùng chung 1 kiểu dữ liệu để có thể tạo và truy cập đối tượng thuộc nhiều kiểu khác nhau thông qua kỹ thuật Ghi đè phương thức (**Method Overriding**).

Tính đa hình (**Polymorphism**) là 1 tính chất của lập trình hướng đối tượng. Khi nhắc đến tính đa hình trong lập trình hướng đối tượng, thường chúng ta sẽ ám chỉ đa hình động.

II. Đa hình tĩnh

Loại đa hình này có tên là đa hình tĩnh vì trình biên dịch có thể nhận biết được tính đa hình tại thời điểm biên dịch. Nói cách khác, chỉ cần dựa vào cú pháp, mã nguồn để có thể xác định đây là tính đa hình.

1. Nạp chồng phương thức

Trong lập trình, mỗi hàm xử lý đều phải có tên riêng. Do đó, đối với những chương trình lớn, số lượng hàm nhiều, sẽ dẫn đến việc cần phải có rất nhiều tên hàm khác nhau, mặc dù có những hàm thực hiện các công việc với mục đích tương tự nhau.

Kỹ thuật Nạp chồng hàm (**Function overloading**) là 1 kỹ thuật trong lập trình cho phép cài đặt các hàm có tên giống nhau và danh sách tham số khác nhau về số lượng, kiểu dữ liệu, thứ tự của tham số. Các hàm này được tính là các hàm khác nhau. Khi gọi hàm, tùy thuộc vào giá trị tham số truyền vào mà trình biên dịch sẽ gọi hàm tương ứng.

Ví dụ: Tất cả các hàm sau đây đều khác nhau mặc dù có tên giống nhau:

```
// Khác nhau về số lượng tham số
void Func() { }
void Func(int x) { }

// Khác nhau về kiểu dữ liệu của tham số
void Func(int x) { }
void Func(string s) { }

// Khác nhau về thứ tự tham số
void Func(int x, double y) { }
void Func(double x, int y) { }
```

Lưu ý: Tên tham số và kiểu trả về không dùng để phân biệt 2 hàm cùng tên. Tất cả các hàm cùng tên trong ví dụ sau đều **không thể phân biệt** vì có danh sách tham số giống nhau:

```
void Func(int x) { }
int Func(int x) { }       // Chỉ khác kiểu trả về
void Func(int a) { }      // Chỉ khác tên tham số
```

Đối với C# là 1 ngôn ngữ thuần hướng đối tượng thì không tồn tại hàm, mà tất cả các hàm đều được cài đặt trong class và gọi là phương thức, do đó kỹ thuật này đối với C# được gọi là Nạp chồng phương thức (**Method Overloading**).

Kỹ thuật này đã được sử dụng để cài đặt các phương thức khởi tạo ở phần III.2, buổi 3. Do phương thức khởi tạo bắt buộc phải có tên trùng với tên class, dẫn đến tất cả phương thức khởi tạo phải có tên giống nhau, nên nếu không sử dụng kỹ thuật nạp chồng phương thức thì sẽ không thể cài đặt được:

```
class SinhVien
{
      public SinhVien() { … }
      public SinhVien(SinhVien sv) { … }
      public SinhVien(string hoTen, int tuoi, double diemLT, double
diemTH) { … }
}
```

2. Nạp chồng toán tử

C# cung cấp khá nhiều toán tử để thực hiện các phép toán số học, so sánh, logic… Lập trình viên hoàn toàn có thể cài đặt các toán tử này để làm việc với class của mình. Kỹ thuật này được gọi là Nạp chồng toán tử (**Operator Overloading**).

Cài đặt toán tử tương tự như cài đặt phương thức, nhưng kèm theo từ khóa `operator` và `static`. Ví dụ sau đây cài đặt toán tử + cho class `PhanSo`:

```
class PhanSo
{
      public int TuSo { get; set; }

      public int MauSo { get; set; }

      public static PhanSo operator+(PhanSo a, PhanSo b)
      {
            PhanSo kq = new PhanSo();
            kq.TuSo = a.TuSo * b.MauSo + a.MauSo * b.TuSo;
            kq.MauSo = a.MauSo * b.MauSo;
            return kq;
      }
}
```

Sau khi đã cài đặt toán tử trên, chúng ta có thể dùng nó để thực hiện phép cộng giữa các đối tượng kiểu PhanSo:

```
static void Main(string[] args)
{
      PhanSo p1 = new PhanSo(1, 2);  // Tạo phân số 1/2
      PhanSo p2 = new PhanSo(1, 4);  // Tạo phân số 1/4
      (p1 + p2).Xuat();              // Kết quả: 6/8
}
```

3. Thành phần static

Khi 1 thành phần của class được khai báo static, thành phần đó được sử dụng chung cho cả class (thay vì là thành phần riêng của các đối tượng), và chỉ có thể được gọi thông qua tên class (thay vì tên đối tượng).

a) Thuộc tính và property static

Ví dụ: Class Nhân viên có thuộc tính static là Tên công ty và property static là Địa chỉ, dùng chung cho tất cả đối tượng thuộc class này:

```
class NhanVien
{
      public static string tenCongty = "Microsoft";

      public static string DiaChi { get; set; } = "Washington, USA";
}

class Program
{
      static void Main(string[] args)
      {
            Console.WriteLine(NhanVien.tenCongty);
            Console.WriteLine(NhanVien.DiaChi);
```

Gọi thành phần static thông qua tên class

```
            NhanVien.tenCongTy = "Apple";
            NhanVien.DiaChi = "California, USA";
        }
}
```

Một ứng dụng của thuộc tính và property static đó là dùng để đếm số lượng đối tượng đã được khởi tạo:

```
class SinhVien
{
        public static int soLuong = 0;

        public SinhVien()
        {
                soLuong++;
        }
}

class Program
{
        static void Main(string[] args)
        {
                SinhVien sv1 = new SinhVien();
                SinhVien sv2 = new SinhVien();
                SinhVien sv3 = new SinhVien();

                Console.WriteLine(SinhVien.soLuong);      // Kết quả: 3
        }
}
```

b) Phương thức khởi tạo static

Phương thức khởi tạo static được dùng để khởi tạo giá trị cho các thuộc tính, property static và để cài đặt các câu lệnh chạy 1 lần duy nhất cho class.

Phương thức khởi tạo static không đi kèm access modifier và không có tham số:

```
class NhanVien
{
        public static string tenCongty;

        public static string DiaChi { get; set; };

        static NhanVien()
        {
                tenCongTy = "Microsoft";
                DiaChi = "Washington, USA";
        }
}
```

Phương thức khởi tạo static không ảnh hưởng đến phương thức khởi tạo mặc định (nghĩa là class vẫn sẽ có thể có phương thức khởi tạo mặc định mà không vi phạm quy tắc Nạp chồng phương thức).

Phương thức khởi tạo static sẽ được gọi tự động ngay trước khi đối tượng đầu tiên của class được tạo ra, hoặc ngay trước khi thuộc tính, property static được truy cập:

```
class Program
{
    static void Main(string[] args)
    {
        //
        Console.WriteLine(NhanVien.tenCongty);
    }
}
```

Phương thức khởi tạo static được gọi tự động

c) Lớp đối tượng static

Class static tương tự như 1 class bình thường với các đặc điểm khác biệt sau:

- Chỉ có thể chứa các thành phần static.
- Không thể tạo ra đối tượng, không thể có phương thức khởi tạo bình thường.
- Không thể được kế thừa (tương đương với `sealed`[1]).

Class static thường dùng để xây dựng các class tính toán, xử lý công việc phổ thông mà không cần đối tượng. Một ví dụ điển hình cho class static là class `Math` của .NET Framework.

Ví dụ: Xây dựng class `MyMath`:

```
static class MyMath
{
    public static double PI { get; } = 3.14159265359;
    public static double Square(double d)
    {
        return d * d;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Số Pi = " + MyMath.PI);
        Console.WriteLine("5^2 = " + MyMath.Square(5));
    }
}
```

[1] Xem thêm về từ khóa sealed ở phần VI, buổi 6.

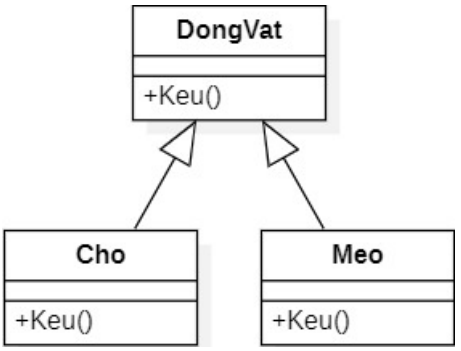
III. Đa hình động

Loại đa hình này có tên là đa hình động vì trình biên dịch có thể nhận biết được tính đa hình tại thời điểm thực thi. Nói cách khác, nếu chỉ dựa vào cú pháp, mã nguồn thì không đủ để xác định đây là tính đa hình mà phải đợi đến khi chương trình được thực thi.

1. Tạo đối tượng lớp con từ lớp cha

Giả sử chúng ta cần mở rộng bài toán về Động vật và Chó như ở phần I, buổi 6 như sau:

- Bổ sung class Mèo.
- Với mỗi class, bổ sung phương thức mô tả tiếng kêu của loài động vật đó.
- Tạo 1 danh sách gồm nhiều con vật khác nhau (chó hoặc mèo), sau đó duyệt danh sách này và cho mỗi con vật phát ra tiếng kêu của nó.



Trong ví dụ này, chúng ta sẽ bỏ qua các thuộc tính mà chỉ quan tâm đến các phương thức như sau:

```
class DongVat
{
      public void Keu() => Console.WriteLine("Kêu gì bây giờ?");
}

class Cho : DongVat
{
      public void Keu() => Console.WriteLine("Gâu Gâu");
}

class Meo : DongVat
{
      public void Keu() => Console.WriteLine("Meo Meo");
}
```

Với yêu cầu tạo danh sách con vật, dù là dùng dạng `List` hay là mảng (array) thì cũng yêu cầu các phần tử phải có cùng kiểu dữ liệu với nhau. Do đó, chúng ta không thể thêm phần tử thuộc 2 kiểu dữ liệu là `Cho` và `Meo` vào cùng 1 danh sách.

Để giải quyết vấn đề trên, lập trình hướng đối tượng cho phép 1 đối tượng thuộc lớp cha có thể tạo ra được 1 đối tượng thuộc lớp con như sau:

```
static void Main(string[] args)
{
      DongVat c = new Cho();    // DongVat tạo ra đối tượng kiểu Cho
      DongVat m = new Meo();    // DongVat tạo ra đối tượng kiểu Meo
}
```

Với cách này, chúng ta có thể dùng chung 1 kiểu dữ liệu là Động vật để có thể tạo ra được cả đối tượng kiểu Chó và kiểu Mèo. Nhờ đó, chúng ta có thể tạo ra 1 danh sách gồm cả 2 loài Chó và Mèo, miễn là danh sách này được khai báo với kiểu dữ liệu là Động vật.

Trong ví dụ dưới đây, chúng ta tạo ra 1 danh sách gồm 3 phần tử, phần tử [0] và [2] thuộc kiểu Chó, phần tử [1] thuộc kiểu Mèo:

```
static void Main(string[] args)
{
      List<DongVat> l = new List<DongVat>();
      l.Add(new Cho());
      l.Add(new Meo());
      l.Add(new Cho());
}
```

Tiếp theo, chúng ta duyệt danh sách ở trên và cho từng con vật kêu:

```
static void Main(string[] args)
{
      foreach (var item in l)
      {
            item.Keu();
      }
}
```

Kết quả xuất ra màn hình như sau:

```
Kêu gì bây giờ?
Kêu gì bây giờ?
Kêu gì bây giờ?
```

Đây rõ ràng không phải là kết quả chúng ta đang cần. Với kết quả này, chúng ta có thể thấy rằng mặc dù tạo ra đối thượng thuộc kiểu của lớp con (Chó, Mèo) nhưng phương thức `Keu()` được gọi vẫn là phương thức của lớp cha (Động vật). Để đối tượng của lớp con có thể gọi đúng phương thức của nó, cần khai báo đa hình cho phương thức `Keu()`.

2. Khai báo đa hình

Chúng ta sẽ khai báo phương thức ở lớp cha kèm theo từ khóa `virtual`, và phương thức tương ứng ở các lớp con kèm theo từ khóa `override`:

```
class DongVat
{
      public virtual void Keu() => Console.WriteLine("Kêu gì bây giờ?");
}
class Cho : DongVat
{
      public override void Keu() => Console.WriteLine("Gâu Gâu");
}
```

```
class Meo : DongVat
{
	public override void Keu() => Console.WriteLine("Meo Meo");
}
```

Sau khi cài đặt tính đa hình, giờ đây mỗi đối tượng có thể gọi chính xác được phương thức `Keu()` của lớp đó:

```
static void Main(string[] args)
{
	foreach (var item in l)
	{
		item.Keu();
	}
}
```

Kết quả xuất ra màn hình như sau:

```
Gâu Gâu
Meo Meo
Gâu Gâu
```

3. Cài đặt phương thức ToString()

Về bản chất, tất cả các kiểu dữ liệu, bao gồm kiểu cơ bản và kiểu tự định nghĩa (class, struct) đều kế thừa từ class `Object`[2]. Đây là lớp cơ sở của tất cả các class khác, đứng đầu trong hệ thống phân cấp class của .NET Framework. Mối quan hệ kế thừa này là ngầm định, không cần phải khai báo.

Class `Object` cung cấp một số phương thức `virtual`, làm cơ sở cho tất cả các class khác kế thừa và cài đặt lại, trong đó có phương thức `ToString()`, dùng để chuyển 1 đối tượng về kiểu chuỗi. Để cài đặt phương thức này cho 1 class, chúng ta sẽ sử dụng kỹ thuật Method overriding như đã trình bày ở trên.

Ví dụ: Cài đặt phương thức `ToString()` cho class `PhanSo`, sau đó xuất kết quả cộng 2 phân số ra màn hình:

```
class PhanSo
{
	public int TuSo { get; set; }
	public int MauSo { get; set; }
	public override string ToString()
	{
		return $"{TuSo}/{MauSo}";
	}
}
```

[2] Alias là `object`

```
class Program
{
      static void Main(string[] args)
      {
            PhanSo p1 = new PhanSo { TuSo = 1, MauSo = 2 };
            PhanSo p2 = new PhanSo { TuSo = 1, MauSo = 4 };
            Console.WriteLine((p1 + p2).ToString()); // Cách đầy đủ
            Console.WriteLine(p1 + p2);              // Cách rút gọn
      }
}
```

IV. Bài tập

Thực hiện các bài tập sau, mỗi bài nằm trong 1 project của solution **OOP_Buoi06**.

Bài 1: Xây dựng lớp đối tượng Phân số, thông tin là Tử số và Mẫu số kiểu số nguyên.

Cài đặt các phương thức và toán tử sau:

Toán tử số học: `+ - * /`	Toán tử so sánh: `== != > >= < <=`
Property `Value` tính giá trị thực của phân số	Phương thức `ToString()`

Bài 2: Cho 4 loại hình học sau:

- Hình chữ nhật: Thông tin gồm có chiều dài và chiều rộng.
- Hình vuông: Thông tin gồm có độ dài cạnh.
- Hình elip: Thông tin gồm có độ dài bán trục lớn và bán trục nhỏ.
- Hình tròn: Thông tin gồm có độ dài bán kính.

Yêu cầu:

- Viết chương trình cho phép người dùng nhập danh sách các hình học. Mỗi hình thuộc 1 trong 4 loại trên.
- Xuất ra danh sách tất cả các hình vừa nhập, mỗi hình cần xuất ra thông tin của nó kèm theo chu vi và diện tích.
- Xuất ra tổng chu vi và tổng diện tích của tất cả các hình.
- Cho biết hình có chu vi lớn nhất và hình có diện tích lớn nhất.

Gợi ý: Công thức tính chu vi và diện tích của hình elip lần lượt là:

Chu vi	Diện tích
$C_{elip} = 2\pi\sqrt{\frac{r_1^2 + r_2^2}{2}}$	$S_{elip} = \pi r_1 r_2$

Bài 3: Trong công ty ABC có 3 loại nhân viên sau:

- Nhân viên thường: có các thông tin Họ tên, Tuổi, Số ngày công, Lương cơ bản. Lương của nhân viên được tính như sau:

  Số ngày công × Lương cơ bản.

- Trưởng phòng: có các thông tin Họ tên, Tuổi, Số ngày công, Lương cơ bản, Phụ cấp chức vụ. Lương của trưởng phòng được tính như sau:

  Số ngày công × Lương cơ bản + Phụ cấp chức vụ

- Giám đốc: có các thông tin Họ tên, Tuổi, Số ngày công, Lương cơ bản, Phụ cấp chức vụ và thâm niên. Lương của Giám đốc được tính như sau:

  30.000.000 + Phụ cấp chức vụ + Thưởng

(Nếu thâm niên ≥ 10 năm thì thưởng 10.000.000. Ngược lại thưởng 3.000.000)

Yêu cầu:

- Viết chương trình cho phép người dùng nhập danh sách nhân viên trong công ty.
- Xuất ra màn hình danh sách nhân viên vừa nhập và mức lương của mỗi người.
- Cho biết tổng lương của tất cả nhân viên.
- Cho biết nhân viên có lương cao nhất.

Bài 4: Công ty điện lực ABC cần quản lý thông tin khách hàng thuộc 1 trong 2 loại sau:

- Khách hàng Việt Nam: có các thông tin Mã khách hàng, Họ tên, Ngày lập hóa đơn, Đối tượng khách hàng (Sinh hoạt, Sản xuất, Kinh doanh), Số điện tiêu thụ (kW), Đơn giá (đ/kW), Định mức (kW). Cách tính tiền điện:
  - Nếu số điện ≤ Định mức: Thành tiền = Số điện × Đơn giá
  - Ngược lại: Mỗi kW vượt định mức tính giá gấp 2.5 lần.
- Khách hàng nước ngoài: có các thông tin Mã khách hàng, Họ tên, Ngày lập hóa đơn, Quốc tịch, Số điện tiêu thụ (kW), Đơn giá (đ/kW). Cách tính tiền điện:

  Thành tiền = Số điện × Đơn giá × 2.5

Yêu cầu:

- Viết chương trình cho phép người dùng nhập danh sách khách hàng của công ty.
- Xuất ra màn hình danh sách khách hàng vừa nhập và tiền điện của mỗi người.
- Cho biết tổng tiền điện của tất cả khách hàng.
- Cho biết khách hàng có số điện tiêu thụ thấp nhất.