Bài 6.13: Tính chất đa hình

- ✓ Tổng quan về polymorphism
- ✓ Các phương thức virtual
- ✓ Ẩn thành phần lớp cha với new
- ✓ Cấm lớp con override các thành phần virtual
- ✓ Ví dụ minh họa và bài tập

Tổng quan về tính chất đa hình

- ✓Đa hình-polymorphism là 1 trong 4 tính chất của lập trình hướng đối tượng.
- ✓Đa hình là tính chất sinh ra từ tính chất kế thừa. Nếu không có kế thừa thì không có đa hình.
- ✓Đa hình tức là có nhiều vai trò, nhiều hình dạng. Ví dụ ở nhà bạn là con của ba mẹ, nhưng là cháu của ông bà; là anh/chị của các em; là em của các anh chị; đến trường bạn là học sinh của thầy cô; đi mua hàng bạn là khách hàng…
- ✓Tính đa hình thể hiện ở 2 khía cạnh riêng biệt:
  - ✓ Tại thời điểm chương trình đang chạy, đối tượng của kiểu dẫn xuất sẽ được coi như đối tượng của kiểu cha của nó khi đối tượng đó là các tham số, trong tập hợp, mảng.
  - ✓ Lớp cha có thể khai báo và triển khai các phương thức virtual và sau đó các lớp con của nó có thể ghi đè lại cho phù hợp với ngữ cảnh sử dụng. Khi chương trình chạy, hệ thống sẽ dựa vào kiểu của đối tượng đang thực hiện lời gọi để gọi đúng phương thức override của phương thức virtual. Bạn có thể thực hiện lời gọi từ biến của kiểu lớp cha nhưng phương thức được gọi có thể là phương thức được override trong lớp con.

Các phương thức virtual

- ✓Các phương thức virtual cho phép ta làm việc với một nhóm các đối tượng có liên quan trong quan hệ kế thừa theo cách thức tương tự nhau.
- ✓Ví dụ: xét các loại hình học 2D. Tất cả chúng đều có hành động tính chu vi. Mỗi loại hình học(tròn, vuông, tam giác, thang…) lại có công thức tính riêng. Do đó ở lớp cha Shape ta tạo phương thức virtual Perimeter. Các lớp con sẽ override lại phương thức virtual này.
- ✓Sau đó ta tạo một mảng chứa các đối tượng của kiểu Shape. Trong đó lưu các đối tượng của các kiểu con(Circle, Rectangle, Triangle).
- ✓Cuối cùng thông qua vòng lặp foreach để gọi và tính chu vi của lần lượt từng đối tượng. Trong đó đối tượng thực hiện lời gọi có kiểu Shape.
- ✓Lúc này dựa vào kiểu của đối tượng thực tế hiện tại mà phương thức Perimeter tương ứng trong các lớp con sẽ được gọi ra để thực hiện.

Ví dụ minh họa

```
class Shape
{
    0 references
    public int X { get; set; }
    0 references
    public int Y { get; set; }
    // phương thức virtual
    4 references
    public virtual double Perimeter() => 0;
}

4 references
class Triangle : Shape
{
    3 references
    public double EdgeA { get; set; }
    3 references
    public double EdgeB { get; set; }
    3 references
    public double EdgeC { get; set; }
    3 references
    public Triangle(double a, double b, double c)
    {
        EdgeA = a;
        EdgeB = b;
        EdgeC = c;
    }
    // ghi đè phương thức tính chu vi của lớp cha
    2 references
    public override double Perimeter() => EdgeA + EdgeB + EdgeC;
    0 references
    public override string ToString() =>
        $"Triangle[EdgeA={EdgeA}, EdgeB={EdgeB}, EdgeC={EdgeC}]";
}
```

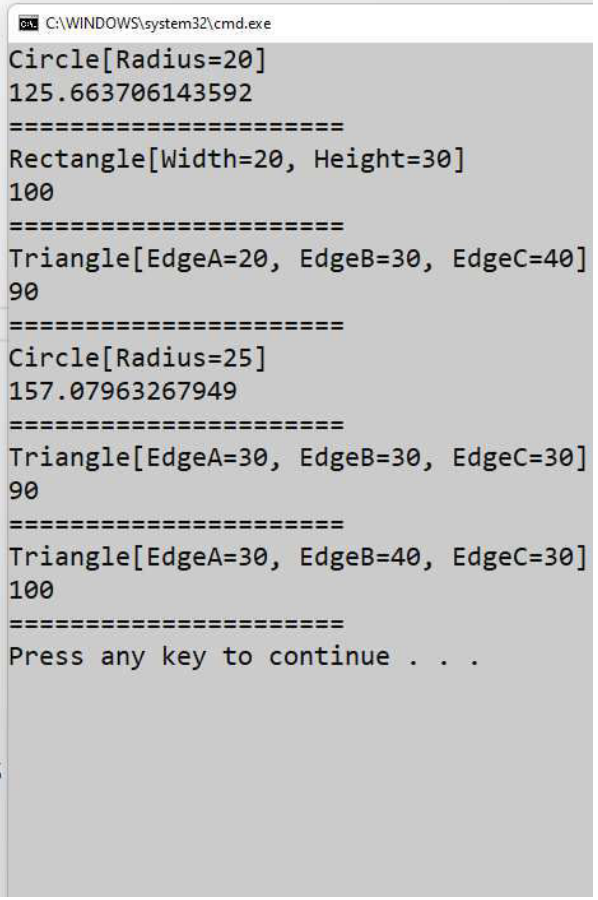
```
class Rectangle : Shape
{
    3 references
    public double Width { get; set; }
    3 references
    public double Height { get; set; }
    1 reference
    public Rectangle(double width, double height)
    {
        Width = width;
        Height = height;
    }
    2 references
    public override double Perimeter() => 2 * (Width + Height);
    0 references
    public override string ToString() =>
        $"Rectangle[Width={Width}, Height={Height}]";
}

3 references
class Circle : Shape
{
    3 references
    public double Radius { get; set; }
    2 references
    public Circle(double radius)
    {
        Radius = radius;
    }
    2 references
    public override double Perimeter() => Math.PI * Radius * 2;
    0 references
    public override string ToString() => $"Circle[Radius={Radius}]";
}
```

Ví dụ minh họa

```
class Lesson613
{
    0 references
    static void Main()
    {
        Shape[] shapes = new Shape[]
        {
            new Circle(20),
            new Rectangle(20, 30),
            new Triangle(20, 30, 40),
            new Circle(25),
            new Triangle(30, 30, 30),
            new Triangle(30, 40, 30)
        };
        ShowResult(shapes);
    }

    1 reference
    static void ShowResult(Shape[] shapes)
    {
        foreach (var item in shapes)
        {
            Console.WriteLine(item);
            Console.WriteLine(item.Perimeter());
            Console.WriteLine("======================");
        }
    }
}
```



Các thành phần virtual

- ✓Các thành phần virtual của lớp cha có thể được kế thừa lại và không có thay đổi gì.
- ✓Các lớp con có thể override lại các thành phần virtual để triển khai hành vi theo cách riêng của lớp con.
- ✓Các lớp con có thể vừa kế thừa các thành phần virtual của lớp cha gần nhất và tiếp tục cho phép các lớp con của nó override lại các thành phần virtual này.
- ✓Lớp con có thể override các thành phần được khai báo abstract, virtual của lớp cha.
- ✓Các lớp con khi ghi đè các phương thức trên phải chỉ rõ là override để tường minh chỉ ra rằng thành phần đó là ứng viên để gọi thực thi trong lời gọi virtual.

Ẩn thành phần của lớp cha với keyword new

- ✓Trong lớp dẫn xuất, ta có thể tạo mới các thuộc tính, trường dữ liệu, phương thức cùng tên với tên của các thành phần đó trong lớp cha và ẩn các thành phần của lớp cha kia đi bằng cách sử dụng keyword new.
- ✓Cú pháp: đặt keyword new ngay trước kiểu trả về của thành phần muốn định nghĩa.

Ví dụ

```
2 references
class BaseClass
{
    0 references
    public string Name { get; set; }
    0 references
    protected long Salary { get; set; }

    1 reference
    public virtual void DoSomething()
    {
        Console.WriteLine("DoSomething in BaseClass.");
    }
}

2 references
class DerivedClass : BaseClass
{
    0 references
    public new string Name { get; set; }
    0 references
    public new long Salary { get; set; }

    1 reference
    public new void DoSomething()
    {
        Console.WriteLine("DoSomething in DerivedClass.");
    }
}
```

```
0 references
class Lesson613
{
    0 references
    static void Main()
    {
        DerivedClass derived = new DerivedClass();
        derived.DoSomething(); // DoSomething in DerivedClass.
        BaseClass baseObj = derived;
        baseObj.DoSomething(); // DoSomething in BaseClass.
    }
}
```

Cấm lớp con override các thành phần virtual

- ✓Các thành phần virtual vẫn là virtual với các lớp con cháu dưới nó.
- ✓Để các lớp con sau một số phân cấp kế thừa nào đó không override phương thức virtual này nữa, ta đặt keyword sealed trước override trong lớp muốn các lớp con của nó ngừng override.

```
class A
{
    2 references
    public virtual void DoSomething()
    {
        Console.WriteLine("DoSomething in class A.");
    }
}

1 reference
class B : A
{
    1 reference
    public sealed override void DoSomething()
    {
        Console.WriteLine("DoSomething in class B.");
    }
}

0 references
class C : B
{
    2 references
    public override void DoSomething() // cannot continue override this
    {
        Console.WriteLine("DoSomething in class B.");
    }
}
```

Cấm lớp con override các thành phần virtual

✓Lúc này lớp con có thể ẩn thành phần override của lớp cha đi bằng keyword new:

```
class A
{
    1 reference
    public virtual void DoSomething()
    {
        Console.WriteLine("DoSomething in class A.");
    }
}

1 reference
class B : A
{
    1 reference
    public sealed override void DoSomething()
    {
        Console.WriteLine("DoSomething in class B.");
    }
}

0 references
class C : B
{
    0 references
    public new void DoSomething() // this is OK
    {
        Console.WriteLine("DoSomething in class B.");
    }
}
```

Gọi thành phần virtual của lớp cha từ lớp con

- ✓Để truy cập tới thành phần virtual của lớp cha khi đã override lại thành phần đó trong lớp con, sử dụng keyword base.
- ✓Cú pháp: base.MemberName
- ✓Ví dụ:

```
1 reference
class A
{
    2 references
    public virtual void DoSomething()
    {
        Console.WriteLine("DoSomething in class A.");
    }
}

1 reference
class B : A
{
    2 references
    public sealed override void DoSomething()
    {
        base.DoSomething(); // gọi tới phương thức virtual của lớp cha
        Console.WriteLine("DoSomething in class B.");
    }
}
```

Nội dung tiếp theo

Các phương thức mở rộng(extension methods)