TRƯỜNG ĐẠI HỌC KINH TẾ - ĐẠI HỌC ĐÀ NẴNG

KHOA THỐNG KÊ – TIN HỌC

Tài liệu tham khảo

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI NGÔN NGỮ C#

Đà Nẵng, 2022

MỤC LỤC

CHƯƠNG 1. KHÁI NIỆM LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Lập trình hướng đối tượng (Object-Oriented Programming – OOP) là một phương pháp lập trình sử dụng các đối tượng (Object) để xây dựng hệ thống phần mềm.

Các đặc điểm của hướng đối tượng:

- *Tính kế thừa (Inheritance)*: cho phép một lớp (class) có thể kế thừa các thuộc tính và phương thức từ các lớp khác đã được định nghĩa.
- *Tính đóng gói (Encapsulation)*: là quy tắc yêu cầu trạng thái bên trong của một đối tượng được bảo vệ và tránh truy cập được từ code bên ngoài (tức là code bên ngoài không thể trực tiếp nhìn thấy và thay đổi trạng thái của đối tượng đó).
- *Tính đa hình (Polymorphism)*: 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.
- *Tính trừu tượng (Abstract)*: là tổng quát hóa một cái gì đó lên, không cần chú ý chi tiết bên trong. Hay nói cách khác, nó là kĩ thuật giúp tạo ra bản thiết kế hoặc bộ khung để các lớp con thừa kế phải tuân theo.

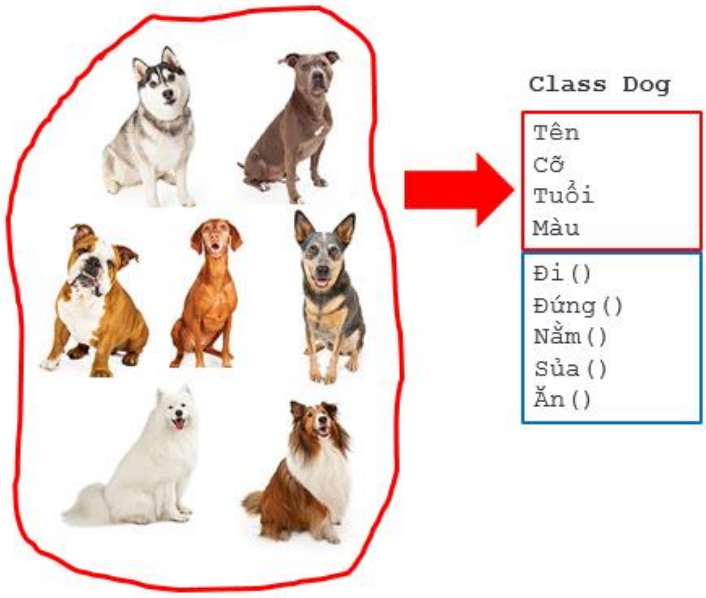
CHƯƠNG 2. LỚP & ĐỐI TƯỢNG

2.1 Lớp (Class)

Một lớp trong lập trình hướng đối tượng đại diện cho tập hợp các đối tượng có cùng các đặc điểm, hành vi, phương thức hoạt động.

Trong đó:

- đặc điểm của đối tượng được thể hiện ra bên ngoài là các thuộc tính (property) của lớp;
- các hành vi hay phương thức hoạt động của đối tượng gọi là phương thức của lớp;
- các thuộc tính và phương thức được gọi chung là thành viên của lớp.



2.1.1 Khai báo class

Khai báo một lớp bắt đầu bằng từ khóa **class** theo sau là tên của lớp, phần thân của lớp được đánh dấu bằng cặp dấu mở đóng ngoặc nhọn {}.

Dưới đây là cấu trúc tổng quát khi khai báo một lớp:

```
<access specifier> class  ClassName
{
    // member properties
    <access specifier> <data type> Property1 { get; set; }
    <access specifier> <data type> Property2 { get; set; }

    // constructors
    <access specifier> ClassName()
    {
        // constructor body
    }
    <access specifier> ClassName(parameter_list)
    {
        // constructor body
    }



    // member methods
    <access specifier> <return type> Method1(parameter_list)
    {
       // method body
    }
    <access specifier> <return type> Method2(parameter_list)
    {
       // method body
    }
}
```

Giải thích:

- `access specifier`: xác định phạm vi truy cập của lớp hoặc thành viên của lớp. Có các từ khóa xác định cấp độ truy cập như sau: `public`, `internal`, `protected`, `private`. Nếu không có từ khóa xác định phạm vi thì mặc định các thành viên của lớp là `private`, còn của lớp sẽ là `internal`.
- `data type`: kiểu dữ liệu của thuộc tính.
- `return type`: kiểu dữ liệu trả về của phương thức.
- `constructor`: phương thức khởi tạo của class. Khi không khai báo constructor thì trình biên dịch sẽ tự động tạo một constructor không có tham số.
- Tên lớp, tên thuộc tính, tên phương thức hợp lệ chỉ gồm các kí tự a-z, A-Z, 0-9 và kí tự _ (gạch nối) và không được bắt đầu bằng số. Thông thường, tên trong C# được đặt tên theo quy tắc: kí tự đầu tiên của tên bắt đầu bằng chữ HOA.

Ví dụ về khai báo class:

```
using System;
namespace Animal
{
    class Dog
    {
        public string Name{get; set;}
        public float Size {get; set;}
        public float Age {get; set;}
        public string Color{get; set;}

        public void Go()
        {
            Console.WriteLine("Going....");
        }

        public void Stay(string Place)
        {
            Console.WriteLine("I am staying at" + Place);
        }
        public void Lie (string Place)
        {
            Console.WriteLine("I am lying at" + Place);
        }

        public void Bark()
        {
            Console.WriteLine("Whoop.....");
        }
        public void Eat (string Food)
        {
            Console.WriteLine("I am eating" + Food);
        }

    }

    class Demo
    {
        public static void Main(string[] args)
        {
            Dog d = new Dog();
            d.Name = "Bull";
            d.Eat("Bone");

        }
    }
}
```

Sau khi biên dịch đoạn code trên và chạy thử, kết quả thu được:

```
I am eating Bone
```

2.1.2 Phương thức

Phương thức là tập hợp các lệnh C# dùng để thực hiện một cách trọn vẹn một nhiệm vụ nào đó.

Cú pháp khai báo một phương thức:

```
<access specifier> <return type> MethodName(<parameters>)
{
      //body of the method
}
```

Trong ví dụ ở mục 2.1.1, `Go`, `Stay`, `Lie`, `Bark`, `Eat` là phương thức.

Ví dụ về khai báo phương thức:

```
public void Eat (string Food)
{
    Console.WriteLine("I am eating" + Food);
}
```

Cú pháp để gọi một phương thức:

```
Tên_Đối_Tượng/Tên_Lớp.Tên_PhươngThức(<các tham số nếu có>);
```

Ví dụ về cách gọi một phương thức của đối tượng:

```
Dog d = new Dog();
d.Eat("Bone");
```

2.1.2.1 Phương thức khởi tạo (Constructor)

Đây là một phương thức đặc biệt của lớp. Nó được thực thi khi khởi tạo đối tượng.

Đặc điểm của phương thức khởi tạo:

- Một phương thức khởi tạo có cùng tên với lớp và không có kiểu trả về;
- Một lớp có thể có nhiều phương thức khởi tạo (khác nhau ở tham số). Trường hợp này giúp khởi tạo giá trị cho các thuộc tính của lớp;
- Một lớp cũng có thể không có phương thức thức khởi tạo. Trong trường hợp không có, trình biên dịch sẽ tự tạo ra một phương thức khởi tạo không có tham số đầu vào.

Ví dụ minh họa cho phương thức khởi tạo:

```
class Dog
    {
        public string Name{get; set;}
        public float Size {get; set;}
        public float Age {get; set;}
        public string Color{get; set;}
```

```
        public Dog(string _name)
        {
            Name = _name;
        }
        public Dog(string _name, float _size, float _age, string _color)
        {
            Name = _name;
            Size = _size;
            Age = _age;
            Color = _color;
        }

        public void Go()
        {
            Console.WriteLine("Going....");
        }

        public void Stay(string Place)
        {
            Console.WriteLine("I am staying at " + Place);
        }
        public void Lie (string Place)
        {
            Console.WriteLine("I am lying at " + Place);
        }

        public void Bark()
        {
            Console.WriteLine("Whoop.....");
        }
        public void Eat (string Food)
        {
            Console.WriteLine("I am eating " + Food);
        }

    }
```

2.1.2.2 Phương thức hủy (destructor)

Là một phương thức đặc biệt của lớp, dùng để hủy một đối tượng của lớp đó.

Đặc điểm:

- Destructor có cùng tên với tên lớp nhưng có dấu ngã (~) phía trước
- Không trả về giá trị
- Không có tham số
- Không được phép chạy các lệnh truy cập dữ liệu

Ví dụ về destructor:

```
class Dog
{
    public string Name{get; set;}

    public Dog(string _name)
    {
        Name = _name;
    }

    public void Go()
    {
        Console.WriteLine("Going....");
    }

    ~Dog()
    {
        Console.WriteLine("Object is being deleted.");
    }
}
```

2.1.3 Thành viên tĩnh của lớp

Thuộc tính hoặc phương thức trở thành dạng tĩnh nếu có từ khóa `static` trước phần khai báo thuộc tính hoặc phương thức. Nó là thành viên của lớp chứ không phải của đối tượng.

Thành viên tĩnh được khởi tạo khi khai báo lớp. Thành viên tĩnh được sử dụng để lưu trữ và chia sẻ giá trị dùng chung giữa tất cả đối tượng được tạo từ lớp.

2.1.3.1 Thuộc tính tĩnh

Ví dụ sau đây minh họa cho việc sử dụng thuộc tính tĩnh.

```
namespace Animal
{
    class Dog
{
    public string Name{get; set;}
    public static int Count;

    public Dog(string _name)
    {
        Name = _name;
        Count++;
    }

    public void Go()
    {
```

```
        Console.WriteLine("Going....");
    }

    ~Dog()
    {
        Console.WriteLine("Object is being deleted.");
    }
}

    class Demo
    {
        public static void Main(string[] args)
        {
            Dog d1 = new Dog("Bull");
            Dog d2 = new Dog("Poodle");
            Console.WriteLine(Dog.Count);

        }
    }
}
```

Kết quả thu được khi chạy code:

```
2
```

Chú ý:

Truy cập thuộc tính tĩnh của lớp thì phải sử dụng tên lớp chứ không sử dụng tên đối tượng (xem đoạn highlight màu vàng trong ví dụ trên).

2.1.3.2 Phương thức tĩnh

Phương thức tĩnh chỉ được phép truy cập vào các thuộc tính tĩnh của lớp.

Ví dụ về phương thức tĩnh:

```
namespace Animal
{
    class Dog
    {
        public string Name{get; set;}
        static public int Count;

        public Dog(string _name)
        {
            Name = _name;
            Count++;
        }
        public void Go()
        {
            Console.WriteLine("Going....");
```

```
        }
        public static int GetNum()
        {
            return Count;
        }
        ~Dog()
        {
            Console.WriteLine("Object is being deleted.");
        }
    }

    class Demo
    {
        public static void Main(string[] args)
        {
            Dog d1 = new Dog("Bull");
            Dog d2 = new Dog("Poodle");
            int NumOfDogs = Dog.GetNum();
            Console.WriteLine(NumOfDogs);
        }
    }
}
```

2.2 Đối tượng

Là một đối tượng trong thế giới thực, có thuộc tính và hành vi. Trong C#, đối tượng là một thể hiện của lớp. Sử dụng từ khóa `new` để khởi tạo một đối tượng của lớp.

Trong ví dụ của phần 2.1.3.2, `d1` và `d2` là 2 đối tượng của lớp `Dog`, và để khởi tạo 2 đối tượng này, chúng ta dùng từ khóa `new`.

Ví dụ:

```
Dog d1 = new Dog("Bull");
Dog d2 = new Dog("Poodle");
```

CHƯƠNG 3. CÁC TÍNH CHẤT CỦA HƯỚNG ĐỐI TƯỢNG

3.1 Tính đóng gói (Encapsulation)

Là khả năng che dấu thông tin của đối tượng với môi trường bên ngoài. Tính chất này giúp đảm bảo tính toàn vẹn và bảo mật của đối tượng.

Tính đóng gói được thể hiện thông qua các từ khóa xác định phạm vi sử dụng:

- `public`: cho phép truy cập khi đứng ở ngoài lớp
- `private`: chỉ cho phép truy cập khi đứng ở trong lớp
- `protected`: chỉ cho phép truy cập khi đứng ở trong lớp hoặc ở lớp con
- `internal`: được truy cập từ các lớp nằm trong cùng một assembly
- `protected internal`:

3.2 Tính thừa kế (Inheritance)

Thừa kế là việc tái sử dụng lại một số thuộc tính, phương thức của một lớp đã có sẵn. Lớp mới là lớp con (lớp dẫn xuất), lớp đã có sẵn là lớp cha (lớp cơ sở). Lớp con thừa kế các thuộc tính hoặc phương thức được định nghĩa ở lớp cha.

3.2.1 Cú pháp khai báo thừa kế

```
<acess-specifier> class <base_class>
{
    ...
}

class <derived_class> : <base_class>
{
    ...
}
```

Giải thích:

- Base_class (lớp cơ sở): khai báo như cách khai báo một lớp thông thường
- Derived_class (lớp dẫn xuất): để khai báo thừa kế từ lớp cơ sở, cần sử dụng dấu hai chấm (:) sau đó đến tên lớp cơ sở.

Mục đích của thừa kế:

- *Tái sử dụng code*. Khi tạo một lớp mới, không cần thiết phải viết lại toàn bộ thuộc tính hoặc phương thức mà sử dụng thuộc tính hoặc phương thức của lớp đã có sẵn.

Ví dụ về thừa kế:

```
public class Animal
{
    protected string Name{get; set;}
    protected float Age{get; set;}
    public void SayHi()
    {
        Console.WriteLine("I'm " + Name + ". Hello world!");
    }
}
public class Dog: Animal
{
    private string Color{get; set;}

    public Dog(string _name, string _color)
    {
        Name = _name;
        Color = _color;
    }
    public void Go()
    {
        Console.WriteLine("Dog "+ Name + " going....");
    }

    ~Dog()
    {
        Console.WriteLine("Object is being deleted.");
    }
}

public class DogDemo
{
    public static void Main(string[] args)
    {
        Dog d1 = new Dog("Bull", "White");
        d1.SayHi();
        d1.Go();
    }
}
```

Kết quả sau khi chạy code:

```
I'm Bull. Hello world!
Dog Bull going....
```

Giải thích ví dụ:

- Lớp Dog thừa kế các thuộc tính `Name`, `Age` và phương thức `SayHi` từ lớp cha Animal.

- Khi khai báo đối tượng `d1` của lớp `Dog`, có thể gọi các phương thức của riêng lớp `Dog` là `Go` và phương thức thừa kế từ lớp cha (`SayHi`).

3.2.2 Phạm vi của thừa kế

Lớp con không phải lúc nào cũng thừa kế được mọi thành viên của lớp cha. Các thành viên sau đây không được thừa kế:

- Static constructor – được dùng để khởi tạo các thuộc tính static;
- Instance constructor – phương thức khởi tạo có đối số để khởi tạo các giá trị cho thuộc tính của lớp;
- Destructor – phương thức hủy;

Những thành viên khác của lớp cha đều được thừa kế bởi lớp con nếu thành viên (thuộc tính/phương thức) có phạm vi sử dụng protected trở lên.

3.2.3 Đa thừa kế

C# không hỗ trợ đa thừa kế nhưng cho phép thừa kế từ một class và triển khai (implement) nhiều interface.

3.3 Tính đa hình (Polymorphism)

Là khả năng một đối tượng có thể thực hiện một tác vụ theo nhiều cách khác nhau. Có 2 loại đa hình: tĩnh và động.

3.3.1 Đa hình tĩnh (overloading)

Gọi là tĩnh vì tính đa hình được thực hiện lúc biên dịch code.

C# cung cấp 2 kĩ thuật để thực hiện đa hình tĩnh:

- Nạp chồng phương thức: là các phương thức có cùng tên nhưng khác nhau về mặt tham số.
- Nạp chồng toán tử: là định nghĩa lại hoặc nạp chồng các toán tử tích hợp sẵn trong C#.

Ví dụ về đa hình tĩnh:

```
    public class Dog
    {
        public string Name{get; set;}
        public string Color{get; set;}
```

```
        public Dog(string _name, string _color)
        {
            Name = _name;
            Color = _color;
        }
        public void Go()
        {
            Console.WriteLine("Dog "+ Name + " going....");
        }
        public void Go(string place)
        {
            Console.WriteLine("Dog " + Name + " going to " + place);
        }
        public static Dog operator +(Dog a, Dog b) //định nghĩa lại phép cộng
        {
            return new Dog(a.Name+b.Name, a.Color+b.Color);
        }
        ~Dog()
        {
            Console.WriteLine("Object is being deleted.");
        }
    }

    public class DogDemo
    {
        public static void Main(string[] args)
        {
            Dog d1 = new Dog("Bull", "White");
            Dog d2 = new Dog ("Poodle", "Yellow");
            d1.Go();
            d1.Go("garden");

            Dog baby = d1 + d2;
            Console.WriteLine("Baby name: " + baby.Name + ", color: " +
baby.Color);
        }
    }
```

Kết quả thực hiện code:

```
Dog Bull going....

Dog Bull going to garden

Baby name: BullPoodle, color: WhiteYellow
```

Giải thích ví dụ:

- Trong lớp `Dog` có 2 phương thức `Go`, mặc dù cùng tên nhưng tham số khác nhau → đây là nạp chồng phương thức

- Đoạn highlight màu vàng là phần định nghĩa phép tính cộng cho lớp `Dog` → nạp chồng toán tử.

3.3.2 Đa hình động (overriding)

Gọi là đa hình động vì tính đa hình được thực thi khi chạy code (runtime).

```
namespace Animal
{
    public abstract class Animal
    {
        public abstract void Speak();
    }
    public class Dog:  Animal
    {

        public override void Speak()
        {
            Console.WriteLine("Whoop whoop...");
        }

    }

    public class Cat: Animal
    {
        public override void Speak()
        {
            Console.WriteLine("Meoooo meooo....");
        }
    }
    public class DogDemo
    {
        public static void Main(string[] args)
        {
            Animal a1 = new Dog();
            Animal a2 =  new Cat();
            a1.Speak();
            a2.Speak();
        }
    }
}
```

Kết quả sau khi chạy code:

```
Whoop whoop...

Meoooo meooo....
```

Giải thích ví dụ:

- Lớp `Animal` là lớp trừu tượng và nó có một phương thức trừu tượng là `Speak`

- Lớp `Cat` và `Dog` thừa kế từ lớp `Animal`, ở mỗi lớp đều định nghĩa phương thức `Speak` theo cách riêng của mỗi loài
- Tuy 2 đối tượng `a1` và `a2` đều có kiểu `Animal` nhưng khởi tạo theo lớp `Dog` và `Cat` nên khi thực thi phương thức `Speak` thì nó kêu theo tiếng kêu của loài đó.

3.4 Tính trừu tượng (abstract)

Là khả năng ẩn chi tiết triển khai, chỉ cung cấp thông tin tính năng tới người dùng. Tính trừu tượng được thực hiện thông qua abstract class và interface.

3.4.1 Triển khai trừu tượng bằng abstract class

Đặt từ khóa abstract trước khai báo lớp thì lớp đó sẽ trở thành abstract class.

Đặc điểm của một abstract class:

- Không thể tạo thể hiện (object) cho class
- Có thể chứa phương thức thông thường, phương thức ảo (virtual method) và phương thức trừu tượng (abstract method)
- Không thể sử dụng từ khóa sealed cho abstract class vì nó không thể ngăn một lớp thừa kế từ lớp abstract.

Ví dụ: (xem phần Đa hình động 3.3.2)

3.4.2 Triển khai trừu tượng bằng interface

Hiểu một cách nôm na, interface là một hợp đồng hay một khuôn mẫu mà tất cả các lớp triển khai phải tuân theo. Trong interface sẽ định nghĩa “cái gì”, còn trong các lớp thực thi sẽ triển khai “làm thế nào”.

```
namespace Animal
{
    interface Animal
    {
        void Speak();
    }
    public class Dog:  Animal
    {

        public  void Speak()
        {
            Console.WriteLine("Whoop whoop...");
        }

    }
```

```
    public class Cat: Animal
    {
        public void Speak()
        {
            Console.WriteLine("Meoooo meooo....");
        }
    }
    public class DogDemo
    {
        public static void Main(string[] args)
        {
            Animal a1 = new Dog();
            Animal a2 =  new Cat();
            a1.Speak();
            a2.Speak();
        }
    }
}
```

Kết quả sau khi thực thi code:

```
Whoop whoop...

Meoooo meooo....
```

Giải thích ví dụ:

- Lớp `Dog` và `Cat` thừa kế interface `Animal` nên nó buộc phải định nghĩa phương thức `Speak` trong phần thân của nó.