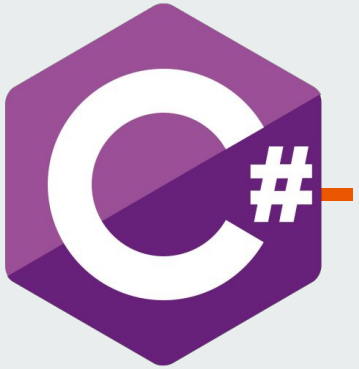


Bài 5

Lớp và đối tượng

Mục tiêu

- Lớp và khai báo lớp
- Tạo đối tượng
- Thuộc tính
- Phương thức
- Truy xuất các thành phần bên trong đối tượng
- Biến tĩnh và phương thức tĩnh
- Truyền tham số cho phương thức

Lớp và khai báo lớp

- Class là một khái niệm trong lập trình hướng đối tượng mô tả cho những thực thể có chung tính chất và hành vi. Class định nghĩa những thuộc tính và hành vi dùng chung cho những đối tượng của lớp đó.
- Khai báo lớp

```
<access_modifier> class <Class_name>
{
    //khai báo các thành viên của lớp
}
```

Các loại thành viên của lớp

Thành viên	Mô tả
Fields	Là các trường dữ liệu, là các biến thể hiện
Constants	Là các hằng số
Methods	Là các phương thức, các hành vi
Properties	Là các thuộc tính bao bọc các trường dữ liệu
Contructors	Là các phương thức khởi tạo
Indexers	Là các chỉ mục, cho phép đối tượng được truy cập giống như mảng
Operators	Là các toán tử được ghi đè, giống như các toán tử +,-,…
Events	Là các sự kiện
Destructor	Là phương thức hủy
Types	Là các kiểu, tức là các lớp inner

Một số lưu ý khi khai báo lớp

- Phạm vi của lớp có thể là public (sử dụng ở bất kỳ đâu) hoặc để mặc định là internal (sử dụng trong cùng một assembly)
- Tên các lớp phải là danh từ
- Ký tự đầu của mỗi từ phải là chữ hoa
- Tên phải đơn giản và có ý nghĩa
- Tên không được trùng với từ khóa của C#
- Tên không có khoảng trắng
- Tên không thể bắt đầu với ký tự số, tuy nhiên có thể bắt đầu với “@” hoặc “_”

Ví dụ

```
public class Category
{
    //khai báo trường
    private int id;
    private string name;
    //khai báo các thuộc tính
    public int Id
    {
        get { return id; }
        set { id = value; }
    }
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    //khai bao phương thức
    public void Display()
    {
        Console.WriteLine("Id:" + id);
        Console.WriteLine("Name:" + name);
    }
}
```

Đối tượng và cách tạo

- Đối tượng là một thực thể trong thế giới thực, nó là thể hiện của một lớp.
- Tạo đối tượng

  <Type> object_name=new <Type>();

- Ví dụ

```
Category c = new Category();
```

- Khởi tạo nhanh đối tượng

```
//Khởi tạo nhanh đối tượng
Student st = new Student { Id = "B04850", Name = "Nguyen Hong Nhung", Mark1 = 16, Mark2 = 18, Mark3 = 20 };
```

Thuộc tính (properties) 1-5

- Trong C# thuộc tính là thành phần được sử dụng để truy xuất đến các trường private được khai báo bên trong lớp
- Mỗi thuộc tính truy xuất đến một biến thành viên duy nhất
- **Cú pháp định nghĩa thuộc tính**

```
class <Class_name>
{
 //khai báo thuộc tính
 [access_modifier] DataType PropertyName
 {
        get{ return field_name;}
        set{ field_name=value;}
 }
 // khai báo tiếp
}
```

Thuộc tính (properties) 2-5

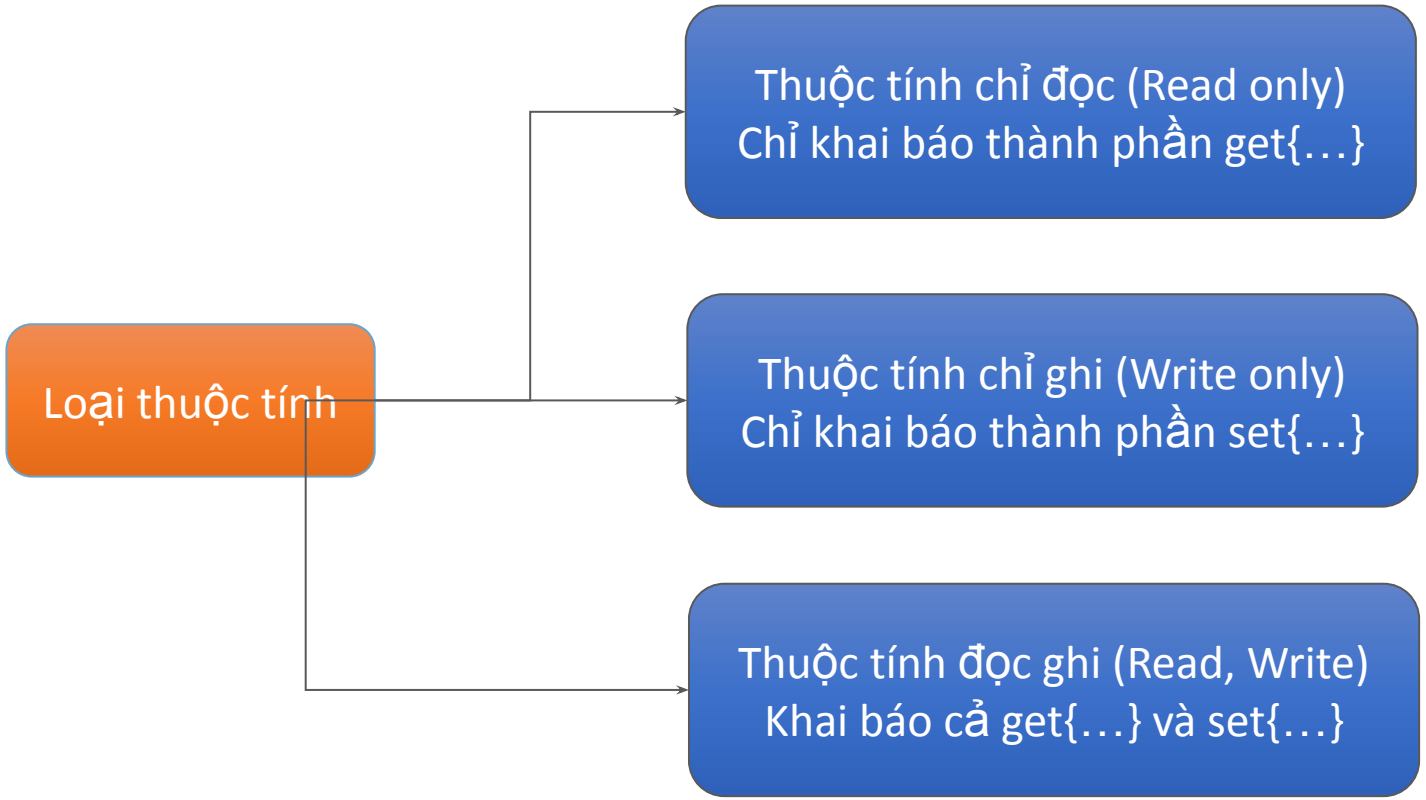
```
class Employee
{
    //khai báo trường private
    private string id;
    private string fullName;
    private int salary;

    //khai báo các thuộc tính
    public string Id
    {
        get { return id; }
        set { id = value; }
    }

    public string FullName
    {
        get { return fullName; }
        set { fullName = value; }
    }

    public int Salary
    {
        set { salary = value; }
        get { return salary; }
    }
}
```

Thuộc tính (properties) 3-5



Thuộc tính (properties) 4-5

```
class Staff
{
    //khai báo trường private
    private string id;
    private string fullName;
    private int salary;
    //khai báo các thuộc tính đọc ghi
    public string Id
    {
        get { return id; }
        set { id = value; }
    }
    //thuộc tính chỉ đọc
    public string FullName
    {
        get { return fullName; }
    }
    //thuộc tính chỉ ghi
    public int Salary
    {
        set { salary = value; }
    }
}
```

Thuộc tính (properties) 5-5

- **Khai báo thuộc tính tự động**

```
class <Class_name>
{
    public DataType PropertyName{ get; set; }
}
```

- Một trường ẩn của thuộc tính sẽ tự động phát sinh khi nó sử dụng
- Có thể thêm từ khóa private vào trước set hoặc get để quy định là read only hoặc write only

```
class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Quantity { get; private set; } //read only
    public int Price { private get; set; } //write onley
}
```

Phương thức (method) 1-3

- Phương thức là các chức năng khai báo bên trong lớp để thực hiện 1 chức cụ thể, phương thức có thể có hoặc không có tham số , phương thức có thể có hoặc không có giá trị trả về

Quy ước đặt tên phương thức

- Không thể là từ khóa trong C#
- Không chứa khoảng trắng
- Không bắt đầu là số
- Có thể bắt đầu với ký tự _ hoặc @

Phương thức (method) 2-3

- Cú pháp

```
<phạm_vi> <kiểu_trả_về> <tên_phương_thức>([tham_số])
{
    //các lệnh
}
```

Phương thức (method) 3-3

```
class Student
{
    private string id;
    private string name;
    private double mark1, mark2, mark3;
    //định nghĩa thuộc tính
    public string Id...
    public string Name...
    public double Mark1...
    public double Mark2...
    public double Mark3...
    //phương thức không tham số và không trả về giá trị
    public void Display()
    {
        Console.WriteLine("Id:" + id);
        Console.WriteLine("Name:" + id);
        Console.WriteLine(" Makr1 {0} \n Mark2 {1} \n Mark3 {2}", mark1, mark2, mark3);
    }
    //phương thức không tham số và trả về giá trị kiểu double
    public double Average()
    {
        return (mark1 + mark2 + mark3) / 3;
    }
    //phương thức có tham số và trả về giá trị kiểu số thực
    public double CalculateAverage(double m1, double m2, double m3)
    {
        mark1 = m1;
        mark2 = m2;
        mark3 = m3;
        return (m1 + m2 + m3) / 3;
    }
}
```

Truy xuất vào thành phần bên trong đối tượng 1-2

- **Truy xuất vào trường**

  <object_name>.<fieldName>;

- **Truy xuất vào thuộc tính**

  <object_name >.<PropertyName>;

- **Truy xuất vào phương thức**

  <object_name >.<MethodName>([value,value2,....]);

Truy xuất vào thành phần bên trong đối tượng 2-2

```
class Program
{
    static void Main(string[] args)
    {
        //Tạo đối tượng
        Student st = new Student();
        //Gán các thuộc tính
        st.Id = "B8450";
        st.Name = "Nguyen Hong Nhung";
        st.Mark1 = 16;
        st.Mark2 = 18;
        st.Mark3 = 20;
        //gọi phương thức
        st.Display();
        //gọi phương thức
        double avg = st.Average();
        Console.WriteLine("Average: "+ avg);
        //Ctrl+F5 để xem kết quả
    }
}
```

Phạm vi truy xuất (Access Modifiers) 1-2

- Trong lập trình hướng đối tượng cho phép bạn có thể hạn chế việc truy xuất tới các thành viên dữ liệu được định nghĩa trong lớp
- Để hạn chế việc truy xuất, c# cung cấp các bổ từ sau:
  - public
  - private
  - protected
  - internal

Phạm vi truy xuất 2-2

Cấp độ

Bổ từ truy xuất

	Sử dụng ở bất kỳ đâu trong ứng dụng	Sử dụng trong lớp hiện tại	Sử dụng trong lớp dẫn xuất
public	✓	✓	✓
private	□	✓	□
protected	□	✓	✓
internal	□	✓	✓

Biến tĩnh

- Biến tĩnh là một biến đặc biệt được truy cập thông qua tên lớp mà không cần sử dụng đối tượng của lớp. Biến tĩnh được khai báo với từ khóa static, khi biến tĩnh được tạo, nó tự động khởi tạo trước khi được sử dụng, tất cả các đối tượng của lớp đều chia sẻ cùng một biến tĩnh. Đối tượng của lớp không thể truy cập vào biến tĩnh.

```
class Program
{
    static double rate = 10.4;
    static string name = "CHUNGLD";

    static void Main(string[] args)
    {
        Console.WriteLine("Rate {0}",rate);
        Console.WriteLine("Name {0}", Program.name);
    }
}
```

Phương thức tĩnh 1-2

- Mặc định một phương thức được gọi bằng cách sử dụng tên đối tượng của lớp (phương thức thể hiện), tuy nhiên bạn có thể gọi phương thức mà không cần tạo bất kỳ đối tượng nào, đó là phương thức tĩnh. Khi khai báo phương sử dụng từ khóa static
- Phương thức tĩnh có thể truy cập trực tiếp tới các biến tĩnh, tuy nhiên chúng không thể truy cập trực tiếp tới biến không tĩnh.

Phương thức tĩnh 2-2

```
class Calculator
{
    public static int Add(int a, int b)
    {
        return (a + b);
    }
    public static int Sub(int a, int b)
    {
        return (a - b);
    }
    public int Mul(int a, int b)
    {
        return a * b;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        //cách gọi phương thức tĩnh
        int sum = Calculator.Add(4, 5);
        int sub = Calculator.Sub(4, 5);
        //cách gọi phương thức không tĩnh
        Calculator cal = new Calculator();
        int mul = cal.Mul(4, 5);
        Console.WriteLine(sum);
        Console.WriteLine(sub);
        Console.WriteLine(mul);
    }
}
```

Truyền tham số cho phương thức 1-2

- **Truyền tham trị**: là cách truyền bản sao giá trị của tham số thực cho tham số hình thức, mọi thay đổi của tham số hình thức trong phương thức sẽ không ảnh hưởng tới tham số thực.
- **Truyền tham chiếu**: sử dụng từ khóa **ref** hoặc **out**, mọi thay đổi của tham số hình thức sẽ ảnh hưởng tới tham số thực.
  - Với từ khóa ref thì giá trị của tham số hình thức sẽ giữ lại khi kết thúc phương thức
  - Với từ khóa out giống từ khóa ref nhưng tham số thực sự không cần khởi tạo giá trị ban đầu.

Truyền tham số cho phương thức 2-2

```
class NumberUtility
{
    public static void Swap(int a, int b)
    {
        int tg = a;
        a = b;
        b = tg;
    }
    public static void Swap(ref int a, ref int b)
    {
        int tg = a;
        a = b;
        b = tg;
    }
    public static void Cal(out double area, double r)
    {
        area = 2 * 3.14 * r;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        int a = 10, b = 20;
        double area;
        //truyền tham trị
        NumberUtility.Swap(a, b);
        Console.WriteLine("a={0},b={1}", a, b);
        //truyền tham chiếu sử dụng từ khóa ref
        NumberUtility.Swap(ref a, ref b);
        Console.WriteLine("a={0},b={1}", a, b);
        //truyền tham chiếu sử dụng từ khóa out
        NumberUtility.Cal(out area, 5);
        Console.WriteLine("area={0}", area);
    }
}
```

HỎI ĐÁP

TRẢI NGHIỆM THỰC HÀNH