Lập trình hướng đối tượng với C#

Nội dung

- *Lớp và đối tượng*
- *Phương thức*
- *Nạp chồng phương thức*
- *Constructor & Destructor*
- *Thừa kế*
- *Ghi đè phương thức trong thừa kế*
- *Giao tiếp (Interface)*
- *Thuộc tính*

Lớp và đối tượng

Lớp

- Những **thuộc tính và hành động chung của các thực thể** được nhóm lại để tạo nên 1 đơn vị duy nhất là lớp.

- **TD**: Lớp Con người có các thuộc tính và hành động sau:
  - Tên
  - Chiều cao
  - Màu tóc.
  - Nói năng
  - Viết
  - ….
- Những thuộc tính và hành động chung gọi là thành viên (member)

Đối tượng

- Một đối tượng là một trường hợp cụ thể của một lớp.
- **TD**: Đối tượng 1 con người thực tế của lớp Con người:

  - Tên: Văn Cường
  - Tuổi: 29
  - Trọng lượng: 60 kg

  Hành động:

  - Đi
  - Nói
  - Suy nghĩ

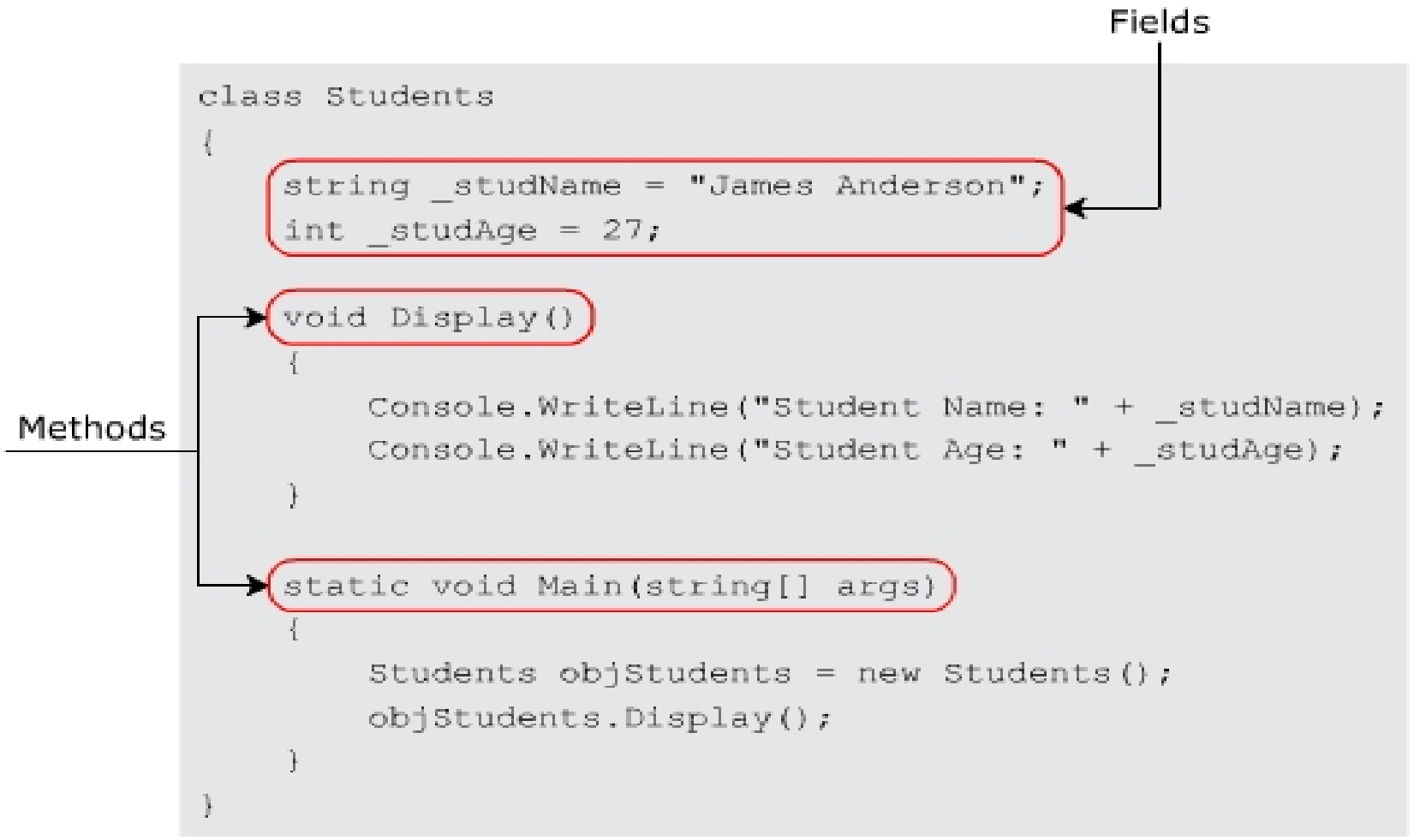
Tạo một lớp mới

- Sử dụng từ khóa class:

```
class <Tên lớp>{
        <Các thành viên của lớp>
}
```

- <Các thành viên của lớp>: dữ liệu thành viên hay phương thức

Tạo một lớp mới



Tạo đối tượng từ lớp

- Việc tạo đối tượng chính là việc **khai báo và khởi tạo** 01 biến từ lớp.
- Thí dụ: Tạo đối tượng p thuộc lớp Students:

```
Students p;
p = new Students();
//hoặc
Students p = new Students ();
```

Phương thức (Method)

Phương thức [1]

- Các hành động của các đối tượng của 1 lớp được thể hiện qua các phương thức (hàm).
- Khai báo phương thức:

```
[<Phạm vi truy cập>]  <Kiểu trả về> <Tên phương thức>
            (<Khai báo các tham số>)
{
            <Các câu lệnh định nghĩa phương thức>
}
```

- Khai báo các tham số giống như khai báo biến.
- Lời gọi phương thức là một biểu thức:

```
<Tên phương thức> (<Danh sách các tham số thực tế>)
```

Phương thức [2]

```
class Book
{
      string _bookName;
      public string Print()
      {
            return _bookName;
      }

      public void Input(string _bName)
      {
            _bookName=_bName;
      }

      static void Main()
      {
            Book b = new Book();
            b.Input("Inside C#");
            Console.WriteLine(b.Print());
            Console.ReadLine();
      }
}
```

Phạm vi truy cập

- private
  - Chỉ truy cập được từ trong lớp khai báo.
- protected
  - Truy cập được từ trong lớp khai báo và các lớp con của lớp khai báo.
- public
  - Truy cập được từ mọi nơi.
- Mặc định là private

Từ khóa static

- Từ khoá static
  - Được dùng với phương thức và dữ liệu thành viên.
  - Dữ liệu thành viên static: dùng chung cho mọi đối tượng của lớp, được truy cập qua qua tên lớp.

    private static char TAB = '\t';
  - Phương thức static: là phương thức chỉ được phép truy cập tới các biến static của lớp, có thể gọi ngay cả khi chưa có đối tượng nào của lớp.

    public static void Welcome() {...}

Phương thức static

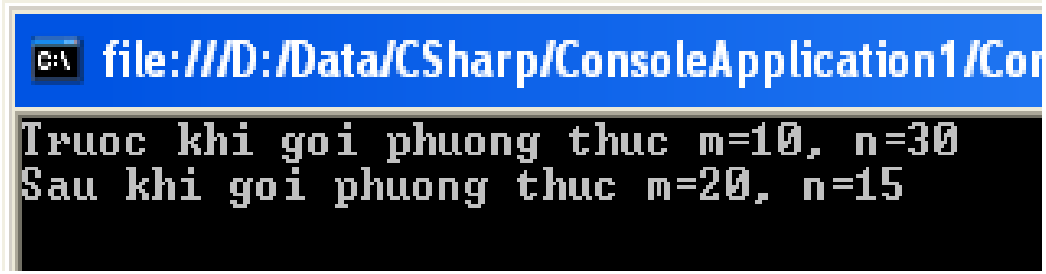
```
class Calculate
{
   public static void Addition(int val1, int val2)
   {
      Console.WriteLine(val1 + val2);
   }
   public void Multiply(int val1, int val2)
   {
      Console.WriteLine(val1 * val2);
   }
}
class StaticMethods
{
   static void Main(string [] args)
   {
      Calculate.Addition(10, 50);
      Calculate objCal = new Calculate();
      objCal.Multiply(10, 20);
   }
}
```

Truyền tham số bằng tham chiếu

- Từ khóa **ref** được sử dụng khi định nghĩa phương thức và lúc gọi thực thi phương thức
- Tham số thực tế được truyền bằng tham chiếu bắt buộc phải được khởi tạo giá trị trước khi gọi thực thi phương thức

Truyền tham số bằng tham chiếu

```
class RefParameters
{
    static void Calculate(ref int a, ref int b)
    {
        a=2*a;
        b=b/2;
    }

    static void Main()
    {
        int m=10;
        int n=30;
        Console.WriteLine("Truoc khi goi phuong thuc m={0}, n={1}",m,n);
        Calculate(ref m, ref n);
        Console.WriteLine("Sau khi goi phuong thuc m={0}, n={1}",m,n);

        Console.ReadLine();
    }
}
```



Tham số đầu ra

- Tham số đầu ra trong C# cũng là hình thức tham số được truyền bằng tham chiếu.
- Từ khóa **out** được sử dụng khi định nghĩa và lúc gọi thực thi phương thức.
- Sử dụng từ khóa **out** ta không cần khởi tạo giá trị cho tham số thực tế tương ứng

Tham số đầu ra

```
class OutParameters
{
      static void Depreciation(out int val)
      {
            val = 20000;
            int dep = val * 5/100;
            int amt = val - dep;
            Console.WriteLine("Depreciation Amount: " + dep);
            Console.WriteLine("Reduced value after depreciation: "
            + amt);
      }

      static void Main(string[] args)
      {
           int value;
           Depreciation(out value);
      }
}
```

Nạp chồng phương thức (Method Overloading)

Nạp chồng phương thức - Method Overloading

- Các phương thức của 1 lớp có cùng tên nhưng khác tham số hoặc kiểu trả về.

Nạp chồng phương thức - Method Overloading

```
class MethodOverloading
{
    static int Square(int num)
    {
        return num*num;
    }

    static double Square(double num)
    {
        return num*num;
    }

    static void Main()
    {
        Console.WriteLine("Binh phuong so nguyen {0}",Square(5));
        Console.WriteLine("Binh phuong so thuc {0}",Square(5.5));
        Console.ReadLine();
    }
}
```

Từ khóa this

- Từ khóa **this** cho phép tham chiếu đến đối tượng hiện hành.
- Thông qua từ khóa **this** ta có thể truy cập đến các thành viên của đối tượng hiện hành.

Từ khóa this

```
class ThisKeyword
{
    double length, width;

    public double Area(double length, double width)
    {
        this.length = length;
        this.width = width;
        return this.length*this.width;
    }

    static void Main()
    {
        ThisKeyword obj = new ThisKeyword();

        Console.WriteLine("Dien tich la {0}",obj.Area(3.0,2.5));
        Console.ReadLine();
    }
}
```

Constructor & Destructor

Phương thức xây dựng - Constructor

- Là phương thức được gọi thực hiện khi một đối tượng của một lớp được tạo ra.
- Trong C#, phương thức xây dựng có tên trùng với tên lớp.
- Có thể nạp chồng phương thức xây dựng.

Phương thức xây dựng - Constructor

```
class Employee
{
    public string emp_Name;
    public int emp_Age;

    public Employee(string Name, int Age)
    {
        emp_Name=Name;
        emp_Age=Age;
    }
}

class Constructor
{
    static void Main()
    {
        Employee emp = new Employee("Roger Federer",29);
        Console.WriteLine("Name: " + emp.emp_Name + " - Age: " + emp.emp_Age);
        Console.ReadLine();
    }
}
```

```
class Rectangle
{
    double length, width;

    public Rectangle()
    {
        length=10.0;
        width=5.0;
    }

    public Rectangle(double length, double width)
    {
        this.length=length;
        this.width=width;
    }

    public double Area()
    {
        return this.length*this.width;
    }

    static void Main()
    {
        Rectangle rec1 = new Rectangle();
        Console.WriteLine("Dien tich hinh chu nhat la {0}",rec1.Area());
        Rectangle rec2 = new Rectangle(5.0,2.5);
        Console.WriteLine("Dien tich hinh chu nhat la {0}",rec2.Area());
        Console.ReadLine();
    }
}
```

Phương thức hủy - Destructor

- Là phương thức được gọi thực thi mỗi khi 1 đối tượng bị thu hồi.
- Trong C#, phương thức hủy có tên là ký tự ~ theo sau là tên lớp.

Phương thức hủy - Destructor

```
class <ClassName>
{
        <ClassName>() //constructor
        {
            //Initialization code
        }
        ~<ClassName>() //destructor
        {
            //De-allocation of objects
        }
}
```

Thừa kế

Thừa kế

- Thừa kế là khả năng một lớp (lớp con) thừa hưởng những thành viên từ 1 lớp đã có (lớp cha).
- **Thí dụ**:
  - Hình vuông kế thừa từ hình chữ nhật
  - Con ếch kế thừa từ loài động vật dưới nước
- Cú pháp:

  class <*Tên lớp con*>**:**<*Tên lớp cha*>{

  }
- C# không hỗ trợ đa thừa kế (một lớp thừa kế từ nhiều lớp).

Thừa kế

```
class Animal
{
    public void Eat()
    {
        Console.WriteLine("Every animal eats something");
    }

    public void Do()
    {
        Console.WriteLine("Every animal does something");
    }
}

class Cat:Animal
{
    static void Main()
    {
        Cat obj = new Cat();
        obj.Eat();
        obj.Do();
        Console.ReadLine();
    }
}
```

Ghi đè phương thức trong thừa kế

Ghi đè phương thức – Method Overriding

- Ghi đè phương thức là việc lớp con có thể định nghĩa lại phương thức đã có của lớp cha.
- Để thực hiện ghi đè phương thức, phương thức của lớp cha ta sử dụng từ khóa **virtual**, phương thức tương ứng của lớp con ta sử dụng từ khóa **override**.

Ghi đè phương thức - Method Overriding

```
class Animal
{
    public virtual void Eat()
    {
        Console.WriteLine("Every animal eats something");
    }

    public void Do()
    {
        Console.WriteLine("Every animal does something");
    }
}

class Cat:Animal
{
    public override void Eat()
    {
        base.Eat();
        Console.WriteLine("Cat eat mice");
    }
    static void Main()
    {
        Cat obj = new Cat();
        obj.Eat();
        obj.Do();
        Console.ReadLine();
    }
}
```

Từ khóa base

- Cho phép truy cập đến các thành viên của lớp cha từ nội bộ lớp con.

Giao tiếp (Interface)

Giao tiếp - Interface

- Giao tiếp chỉ ra các "tính chất" mà một đối tượng có thể có, trong một "ngữ cảnh" nào đó.
  - Một người có thể khi ở nhà là một người con, ở trường là một sinh viên, ở lớp là một người bạn.
- Giao tiếp trong C# có thể được dùng để thể hiện sự đa kế thừa như trong C++.

Giao tiếp – Interface

- Khai báo giao tiếp

  interface Name {...}

- Một giao tiếp thường chỉ chứa khai báo của các phương thức (không có mã lệnh để cài đặt).
- Một giao tiếp có thể thừa kế một giao tiếp khác.
- Một lớp có thể cài đặt một hay nhiều giao tiếp nhưng chỉ có thể thừa kế từ một lớp.

Giao tiếp - Interface

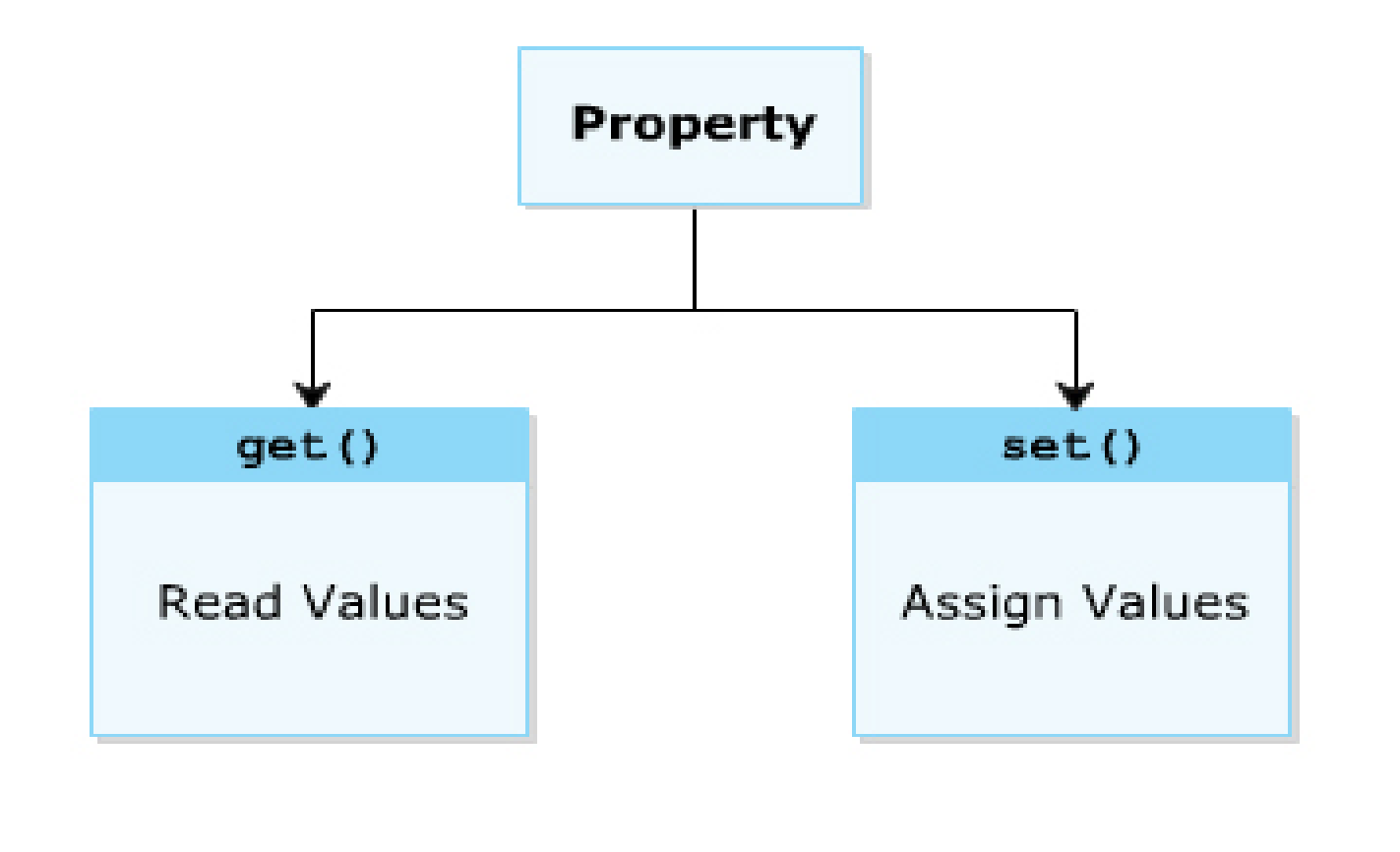
```
interface ITerrestrialAnimal
{
    void Eat();
}
interface IMarineAnimal
{
    void Swim();
}

class Crocodile : ITerrestrialAnimal, IMarineAnimal
{
    public void Eat()
    {
      Console.WriteLine("The Crocodile eats flesh");
    }

    public void Swim()
    {
      Console.WriteLine("The Crocodile can swim four
times faster than an Olympic swimmer");
    }
    static void Main(string[] args)
    {
      Crocodile objCrocodile = new Crocodile();
      objCrocodile.Eat();
      objCrocodile.Swim();
    }
}
```

Thuộc tính (Properties)

Thuộc tính - Properties

- Thuộc tính cho phép bảo vệ các dữ liệu thành viên của 1 lớp bằng các thao tác đọc/ghi dữ liệu thành viên thông qua khai báo thuộc tính.
- Thuộc tính có thể cho phép kiểm tra sự hợp lệ của dữ liệu thành viên

Thuộc tính - Properties



```
class Employee
{
    public string strName;
    public int emp_Age;

    public string emp_Name
    {
        get
        {
            return strName;
        }
        set
        {
            strName=value;
        }
    }

    public Employee(string Name, int Age)...
}

class Constructor
{
    static void Main()
    {
        Employee emp = new Employee("Roger Federer",29);
        Console.WriteLine("Name: " + emp.emp_Name + " - Age: " + emp.emp_Age);
        Console.ReadLine();
    }
}
```

Thuộc tính - Properties