

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C#

Biên soạn: **ThS. Nguyễn Thị Anh Thư**

1

NỘI DUNG

1. Tính kế thừa
2. Tính đa hình
3. Nạp chồng toán tử
4. Interface

1. TÍNH KẾ THỪA

- Một trong những khái niệm quan trọng nhất trong lập trình hướng đối tượng là **Tính kế thừa (Inheritance)**.
 - Tính kế thừa cho phép chúng ta định nghĩa một lớp trong điều kiện một lớp khác, **giúp cho quá trình tạo và duy trì một ứng dụng trở nên dễ dàng hơn**.
 - Điều này cũng **cung cấp một cơ hội để tái sử dụng tính năng code và thời gian thực thi nhanh hơn**.
- Khi tạo một lớp, thay vì viết toàn bộ các thành viên dữ liệu và các hàm thành viên mới, lập trình viên có thể kế thừa các thành viên của một lớp đang tồn tại. Lớp đang tồn tại này được gọi là **Base Class - lớp cơ sở** và lớp mới được xem như là **Derived Class – lớp thừa kế**.

1. TÍNH KẾ THỪA

Lớp cơ sở (Base Class) và Lớp thừa kế (Derived Class) trong C#

- Một lớp có thể được kế thừa từ hơn một lớp khác, nghĩa là, nó có thể kế thừa dữ liệu và hàm từ nhiều Lớp hoặc Interface cơ sở.
- Cú pháp để tạo lớp kế thừa trong C# là:

```
<access-specifier> class <base_class>
{
    ...
}
class <derived_class> : <base_class>
{
    ...
}
```

1. TÍNH KẾ THỪA

Xét một **lớp cơ sở Shape** và **lớp kế thừa Rectangle**

- Tạo 3 lớp có tên lần lượt là **Shape**, **Rectangle**, **TestCsharp** trong đó:
 - Lớp **Shape** là lớp cơ sở
 - Lớp **Rectangle** là lớp kế thừa
 - Lớp **TestCsharp** chứa phương thức **main()** để thao tác trên đối tượng **Rectangle**

1. TÍNH KẾ THỪA

- Lớp **Shape** là lớp cơ sở, lớp **Rectangle** là lớp kế thừa.

```
class Shape
{
    protected int chieu_rong;
    protected int chieu_cao;
    1 reference
    public void setChieuRong(int w)
    {
        chieu_rong = w;
    }
    1 reference
    public void setChieuCao(int h)
    {
        chieu_cao = h;
    }
}
```

```
class Rectangle : Shape
{
    1 reference
    public int tinhDienTich()
    {
        return (chieu_cao * chieu_rong);
    }
}
```

1. TÍNH KẾ THỪA

- Lớp **TestCsharp** chứa phương thức **main()** để thao tác trên đối tượng **Rectangle**

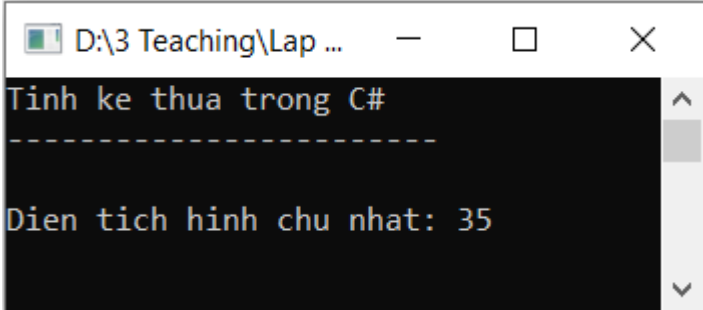
```
public class TestCsharp
{
    0 references
    public static void Main(string[] args)
    {
        Console.WriteLine("Tinh ke thua trong C#");
        Console.WriteLine("-----\n");

        //tao doi tuong HinhChuNhat
        Rectangle hcn = new Rectangle();

        hcn.setChieuRong(5);
        hcn.setChieuCao(7);

        // in dien tich cua doi tuong.
        Console.WriteLine("Dien tich hinh chu nhat: {0}", hcn.tinhDienTich());

        Console.ReadKey();
    }
}
```

A screenshot of a Windows console window titled "D:\3 Teaching\Lap ...". The window has standard Windows window controls (minimize, maximize, close). The console output is as follows:
Tinh ke thua trong C#

Dien tich hinh chu nhat: 35

1. TÍNH KẾ THỪA

Khởi tạo Lớp cơ sở (Base Class) trong C#

- **Lớp kế thừa (Derived Class)** trong C# kế thừa các biến thành viên và các phương thức thành viên từ lớp cơ sở. Vì thế, **đối tượng của lớp cha nên được tạo trước khi lớp phụ được tạo.**
- Tạo 3 lớp có tên lần lượt là **HinhChuNhat**, **ChiPhiXayDung**, **TestCsharp** như sau:
 - Lớp **HinhChuNhat** là lớp cơ sở
 - Lớp **ChiPhiXayDung** kế thừa lớp **HinhChuNhat**
 - Lớp **TestCsharp** chứa phương thức **main()** để thao tác trên đối tượng **ChiPhiXayDung**

1. TÍNH KẾ THỪA

- Lớp **HìnhChuNhat** là lớp cơ sở.

```
class HìnhChuNhat
{
    //cac bien thanh vien
    protected double chieu_dai;
    protected double chieu_rong;
    // constructor
    1 reference
    public HìnhChuNhat(double l, double w)
    {
        chieu_dai = l;
        chieu_rong = w;
    }
    //phuong thuc
    2 references
    public double tinhDienTich()
    {
        return chieu_dai * chieu_rong;
    }

    1 reference
    public void Display()
    {
        Console.WriteLine("Chieu dai: {0}", chieu_dai);
        Console.WriteLine("Chieu rong: {0}", chieu_rong);
        Console.WriteLine("Dien tich: {0}", tinhDienTich());
    }
}
```

1. TÍNH KẾ THỪA

- Lớp **ChiPhiXayDung** kế thừa lớp **HinhChuNhat**

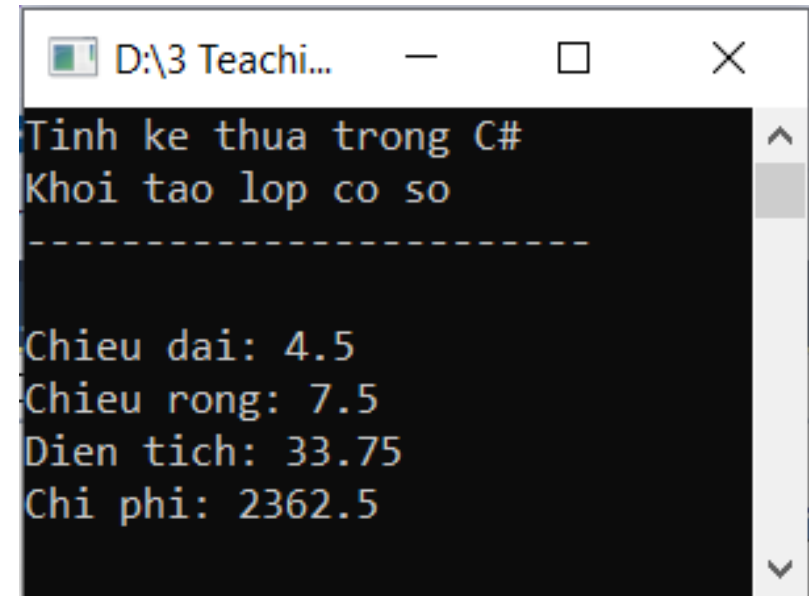
```
class ChiPhiXayDung : HinhChuNhat
{
    private double cost;
    1 reference
    public ChiPhiXayDung(double l, double w) : base(l, w)
    { }
    1 reference
    public double tinhChiPhi()
    {
        double chi_phi;
        chi_phi = tinhDienTich() * 70;
        return chi_phi;
    }
    1 reference
    public void hienThiThongTin()
    {
        base.Display();
        Console.WriteLine("Chi phi: {0}", tinhChiPhi());
    }
}
```

1. TÍNH KẾ THỪA

- Lớp **TestCsharp** chứa phương thức **main()** để thao tác trên đối tượng **ChiPhiXayDung**.

```
public class TestCsharp
{
    0 references
    public static void Main(string[] args)
    {
        Console.WriteLine("Tinh ke thua trong C#");
        Console.WriteLine("Khoi tao lop co so");
        Console.WriteLine("-----\n");
        //tao doi tuong ChiPhiXayDung
        ChiPhiXayDung t = new ChiPhiXayDung(4.5, 7.5);
        t.hienThiThongTin();
        Console.ReadLine();

        Console.ReadKey();
    }
}
```



```
Tinh ke thua trong C#
Khoi tao lop co so
-----

Chieu dai: 4.5
Chieu rong: 7.5
Dien tich: 33.75
Chi phi: 2362.5
```

1. TÍNH KẾ THỪA

Đa kế thừa trong C#

- C# không hỗ trợ đa kế thừa. Tuy nhiên, chúng ta có thể sử dụng **Interface** để triển khai đa kế thừa.
- Ví dụ sau minh họa cách sử dụng **Interface** để triển khai đa kế thừa trong C#:
 - Tạo 3 lớp có tên lần lượt là **Shape**, **HinhChuNhat**, **TestCsharp**.
 - Một **interface** có tên là **ChiPhiSon**.

1. TÍNH KẾ THỪA

- Lớp **Shape** là lớp cơ sở, interface **ChiPhiSon**.

```
class Shape
{
    protected int chieu_rong;
    protected int chieu_cao;
    1 reference
    public void setChieuRong(int w)
    {
        chieu_rong = w;
    }
    1 reference
    public void setChieuCao(int h)
    {
        chieu_cao = h;
    }
}
```

```
public interface ChiPhiSon
{
    2 references
    int tinhChiPhi(int dien_tich);
}
```

1. TÍNH KẾ THỪA

- Lớp **HìnhChuNhat** là lớp kế thừa lớp **Shape** và interface **ChiPhiSon**.

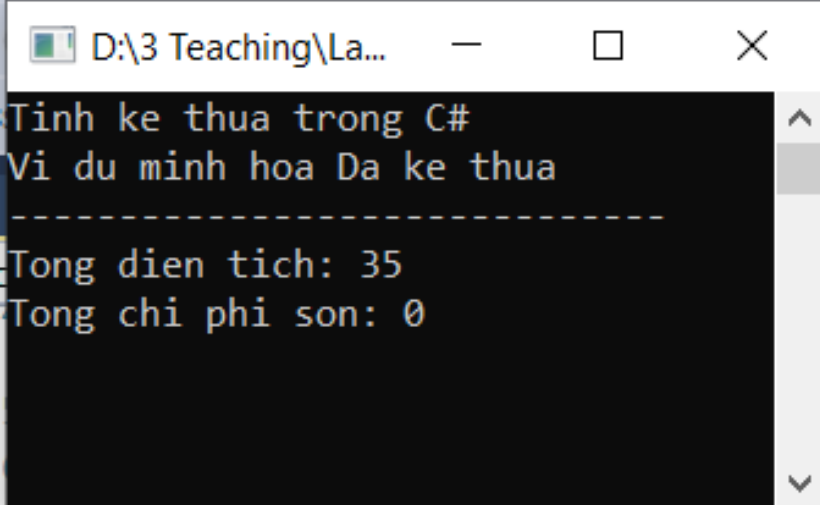
```
class HìnhChuNhat : Shape, ChiPhiSon
{
    2 references
    public int tinhDienTich()
    {
        return (chieu_rong * chieu_cao);
    }
    2 references
    public int tinhChiPhi(int dien_tich)
    {
        return dien_tich * 70;
    }
}
```

1. TÍNH KẾ THỪA

- Lớp **TestCsharp** chứa phương thức **main()** để thao tác trên đối tượng **HìnhChuNhat**.

```
public class TestCsharp
{
    0 references
    public static void Main(string[] args)
    {
        Console.WriteLine("Tinh ke thua trong C#");
        Console.WriteLine("Vi du minh hoa Da ke thua");
        Console.WriteLine("-----");
        //tao doi tuong HìnhChuNhat
        HìnhChuNhat hcn = new HìnhChuNhat();
        int dien_tich;
        hcn.setChieuRong(5);
        hcn.setChieuCao(7);
        dien_tich = hcn.tinhDienTich();

        // in dien tich va chi phi.
        Console.WriteLine("Tong dien tich: {0}", hcn.tinhDienTich());
        Console.WriteLine("Tong chi phi son: 0", hcn.tinhChiPhi(dien_tich));
        Console.ReadLine();
    }
}
```

A screenshot of a Windows console window titled "D:\3 Teaching\La...". The window has standard Windows window controls (minimize, maximize, close). The console output is as follows:
Tinh ke thua trong C#
Vi du minh hoa Da ke thua

Tong dien tich: 35
Tong chi phi son: 0
The text is displayed in a monospaced font on a black background with white text.

2. TÍNH ĐA HÌNH

Từ polymorphism (tính đa hình) nghĩa là có nhiều hình thái.

- Trong lập trình hướng đối tượng, tính đa hình thường được diễn đạt như là “**một Interface, nhiều hàm**”.
- Tính đa hình trong C# có thể là **static** hoặc **dynamic**. Trong đó:
 - Kiểu đa hình **static** có thể được gọi là **đa hình tĩnh**.
 - Kiểu đa hình **dynamic** có thể được gọi là **đa hình động**.
- Trong đa hình tĩnh, phần phản hồi tới một hàm được xác định tại compile time. Trong khi đó với đa hình động, nó được quyết định tại runtime.

2. TÍNH ĐA HÌNH

Đa hình static trong C#

- Kỹ thuật liên kết một hàm với một đối tượng trong thời gian biên dịch được gọi là Early Binding. Nó cũng được gọi là Static Binding. C# cung cấp hai kỹ thuật để triển khai đa hình tĩnh. Chúng là:
 - **Nạp chồng hàm (Function overloading)**
 - **Nạp chồng toán tử (Operator overloading) – bàn luận sau.**

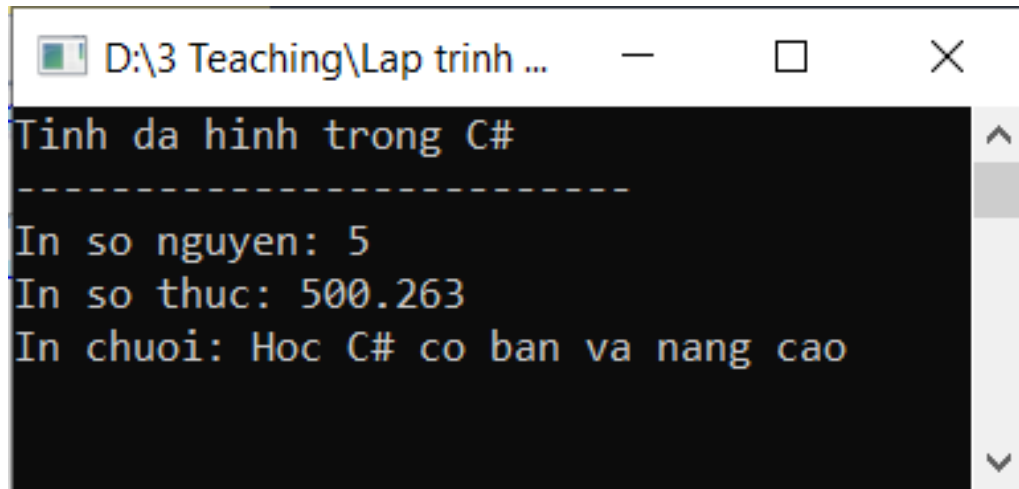
2. TÍNH ĐA HÌNH

Nạp chồng hàm trong C#

- Chúng ta có thể có nhiều định nghĩa cho cùng tên hàm trong cùng một phạm vi.
- Các định nghĩa này của hàm phải khác nhau: như kiểu hoặc số tham số trong danh sách tham số.
- Trong C#, chúng ta không thể nạp chồng các khai báo hàm mà chỉ khác nhau ở kiểu trả về.

2. TÍNH ĐA HÌNH

- Ví dụ sau minh họa cách sử dụng hàm **print()** để in các kiểu dữ liệu khác nhau trong C#:



The screenshot shows a console window titled "D:\3 Teaching\Lap trình ...". The output text is as follows:

```
Tinh da hinh trong C#
-----
In so nguyen: 5
In so thuc: 500.263
In chuoi: Hoc C# co ban va nang cao
```

```
public class TestCsharp
{
    1 reference
    void print(int i)
    {
        Console.WriteLine("In so nguyen: {0}", i);
    }
    1 reference
    void print(double f)
    {
        Console.WriteLine("In so thuc: {0}", f);
    }
    1 reference
    void print(string s)
    {
        Console.WriteLine("In chuoi: {0}", s);
    }

    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Tinh da hinh trong C#");
        Console.WriteLine("-----");
        //tao doi tuong TestCsharp
        TestCsharp p = new TestCsharp();

        // goi ham print()
        p.print(5);
        p.print(500.263);
        p.print("Hoc C# co ban va nang cao");
        Console.ReadKey();
    }
}
```

2. TÍNH ĐA HÌNH

Đa hình dynamic trong C#

- C# cho phép tạo các lớp abstract (trừu tượng) được sử dụng để cung cấp trình triển khai cục bộ lớp của một Interface. Trình triển khai (Implementation) được hoàn thành khi một lớp kế thừa kế thừa từ nó. **Các lớp Abstract chứa các phương thức abstract, được triển khai bởi lớp kế thừa.** Lớp kế thừa này có tính năng chuyên dụng hơn.
- **Một số qui tắc về các lớp abstract trong C#:**
 - Không thể tạo một Instance (sự thể hiện) của một lớp abstract.
 - Không thể khai báo một phương thức abstract ở bên ngoài một lớp abstract.

2. TÍNH ĐA HÌNH

- Ví dụ sau minh họa một lớp abstract trong C#:
 - Tạo 3 lớp có tên lần lượt là **Shape**, **HìnhChuNhat**, **TestCsharp**.
 - Lớp **Shape** là một lớp abstract, Lớp **HìnhChuNhat** là một lớp kế thừa lớp **Shape**.

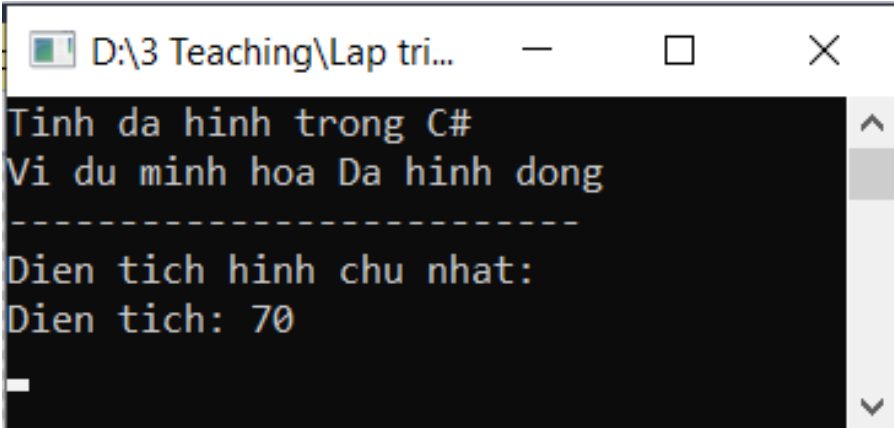
```
abstract class Shape
{
    2 references
    public abstract int tinhDienTich();
}
3 references
class HìnhChuNhat : Shape
{
    private int chieu_dai;
    private int chieu_rong;
    1 reference
    public HìnhChuNhat(int a = 0, int b = 0)
    {
        chieu_dai = a;
        chieu_rong = b;
    }
    2 references
    public override int tinhDienTich()
    {
        Console.WriteLine("Dien tich hình chu nhật:");
        return (chieu_rong * chieu_dai);
    }
}
```

2. TÍNH ĐA HÌNH

- Lớp **TestCsharp** chứa phương thức **main()** để thao tác trên đối tượng **HìnhChuNhat**.

```
public class TestCsharp
{
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Tinh da hinh trong C#");
        Console.WriteLine("Vi du minh hoa Da hinh dong");
        Console.WriteLine("-----");

        HìnhChuNhat r = new HìnhChuNhat(10, 7);
        double a = r.tinhDienTich();
        Console.WriteLine("Dien tich: {0}", a);
        Console.ReadKey();
    }
}
```



The screenshot shows a console window titled "D:\3 Teaching\Lap tri...". The output text is as follows:

```
Tinh da hinh trong C#
Vi du minh hoa Da hinh dong
-----
Dien tich hinh chu nhat:
Dien tich: 70
```

2. TÍNH ĐA HÌNH

- Hàm **virtual** trong C# cho phép một hàm được định nghĩa trong một lớp và được triển khai ở lớp kế thừa.
- Các hàm **virtual** có thể được triển khai một cách khác nhau trong lớp kế thừa khác nhau và việc gọi những hàm này sẽ được quyết định tại runtime.
- Đa hình động trong C# được triển khai bởi các lớp **abstract** và các hàm **virtual**.

2. TÍNH ĐA HÌNH

Ví dụ sau minh họa điều này:

- Tạo 5 lớp có tên lần lượt như sau:
 - Lớp **Shape**: lớp cơ sở
 - Lớp **HinhChuNhat**: là lớp kế thừa lớp **Shape**
 - Lớp **TamGiac**: là lớp kế thừa lớp **Shape**
 - Lớp **HienThiDuLieu**: in các dữ liệu
 - Lớp **TestCsharp**: chứa phương thức **main()** để thao tác trên các đối tượng

2. TÍNH ĐA HÌNH

- Lớp **Shape**: lớp cơ sở. Lớp **HìnhChuNhat**: là lớp kế thừa lớp **Shape**.

```
class Shape
{
    protected int chieu_rong, chieu_cao;
    2 references
    public Shape(int a = 0, int b = 0)
    {
        chieu_rong = a;
        chieu_cao = b;
    }
    3 references
    public virtual int tinhDienTich()
    {
        Console.WriteLine("Dien tich cua class cha: ");
        return 0;
    }
}
```

```
class HìnhChuNhat : Shape
{
    1 reference
    public HìnhChuNhat(int a = 0, int b = 0) : base(a, b)
    {
    }
    3 references
    public override int tinhDienTich()
    {
        Console.WriteLine("Dien tich cua class HìnhChuNhat: ");
        return (chieu_rong * chieu_cao);
    }
}
```

2. TÍNH ĐA HÌNH

- Lớp **TamGiac**: là lớp kế thừa lớp **Shape**. Lớp **HienThiDuLieu**: in các dữ liệu.

```
class TamGiac : Shape
{
    1 reference
    public TamGiac(int a = 0, int b = 0) : base(a, b)
    {
    }
    3 references
    public override int tinhDienTich()
    {
        Console.WriteLine("Dien tich cua class TamGiac: ");
        return (chieu_cao * chieu_rong / 2);
    }
}
```

```
class HienThiDuLieu
{
    2 references
    public void hienThiDienTich(Shape sh)
    {
        int a;
        a = sh.tinhDienTich();
        Console.WriteLine("Dien tich: {0}", a);
    }
}
```

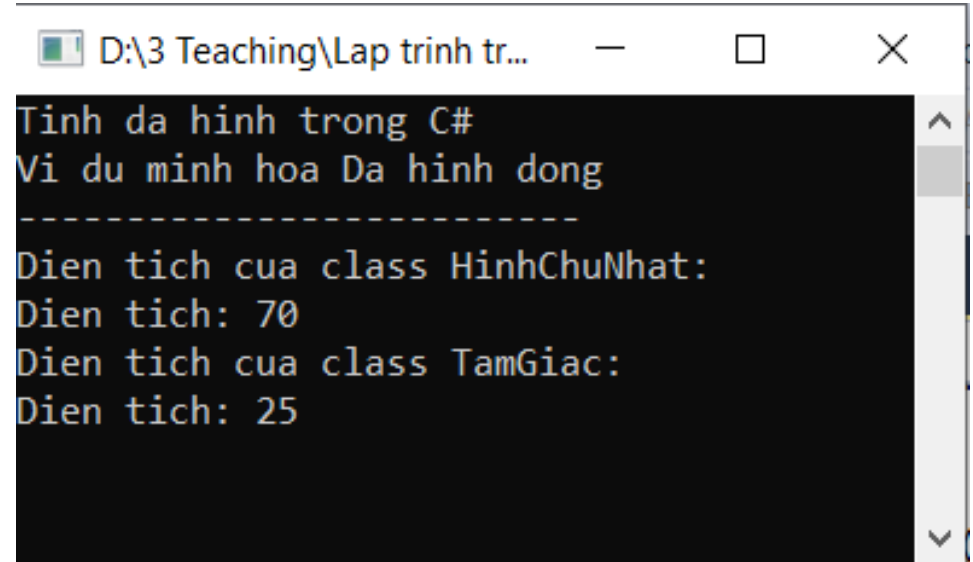
2. TÍNH ĐA HÌNH

- Lớp **TestCsharp**: chứa phương thức **main()** để thao tác trên các đối tượng.

```
public class TestCsharp
{
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Tinh da hinh trong C#");
        Console.WriteLine("Vi du minh hoa Da hinh dong");
        Console.WriteLine("-----");

        HienThiDuLieu c = new HienThiDuLieu();
        HinhChuNhat r = new HinhChuNhat(10, 7);
        TamGiac t = new TamGiac(10, 5);
        c.hienThiDienTich(r);
        c.hienThiDienTich(t);

        Console.ReadKey();
    }
}
```

A screenshot of a Windows console window titled "D:\3 Teaching\Lap trinh tr...". The console displays the output of the C# program. It shows the title "Tinh da hinh trong C#", followed by "Vi du minh hoa Da hinh dong", and a dashed line separator. Below the separator, it shows the area calculation for a rectangle: "Dien tich cua class HinhChuNhat:" followed by "Dien tich: 70", and then for a triangle: "Dien tich cua class TamGiac:" followed by "Dien tich: 25".

```
Tinh da hinh trong C#
Vi du minh hoa Da hinh dong
-----
Dien tich cua class HinhChuNhat:
Dien tich: 70
Dien tich cua class TamGiac:
Dien tich: 25
```

3. NẠP CHỒNG TOÁN TỬ

Operator Overloading là Nạp chồng toán tử

- Có thể tái định nghĩa hoặc nạp chồng hầu hết các toán tử có sẵn trong C#. Vì thế, một lập trình viên có thể sử dụng các toán tử với các kiểu tự định nghĩa (user-defined).
- Các toán tử được nạp chồng trong C# là các hàm với các tên đặc biệt: từ khóa **operator** được theo sau bởi biểu tượng cho toán tử đang được định nghĩa.
- Tương tự như bất kỳ hàm nào khác, một toán tử được nạp chồng có một kiểu trả về và một danh sách tham số.

3. NẠP CHỒNG TOÁN TỬ

Operator Overloading là Nạp chồng toán tử

- Mỗi loại phép toán sẽ có cách nạp chồng riêng. Tuy nhiên, cú pháp chung là:

```
public static <return_type> operator <operator>(<parameters>) { ... }
```

- Các toán tử có thể nạp chồng:

`+, -, !, ~, ++, --, +, -, *, /, %, ==, !=, <, >, <=, >=`

3. NẠP CHỒNG TOÁN TỬ

Triển khai Nạp chồng toán tử trong C#

- Ví dụ dưới đây minh họa cách triển khai nạp chồng toán tử trong C#:
 - Tạo hai lớp có tên lần lượt là **Box**, **TestCsharp** như sau:
 - Lớp **Box**: chứa các thuộc tính và phương thức.
 - Lớp **TestCsharp**: chứa phương thức **main()** để thao tác trên đối tượng **Box**.

```

class Box
{
    private double chieu_dai;
    private double chieu_rong;
    private double chieu_cao;
    3 references
    public double tinhTheTich()
    {
        return chieu_dai * chieu_rong * chieu_cao;
    }
    2 references
    public void setChieuDai(double len)
    {
        chieu_dai = len;
    }
    2 references
    public void setChieuRong(double bre)
    {
        chieu_rong = bre;
    }
    2 references
    public void setChieuCao(double hei)
    {
        chieu_cao = hei;
    }
    // nap chong toan tu + de cong hai doi tuong Box.
    1 reference
    public static Box operator +(Box b, Box c)
    {
        Box box = new Box();
        box.chieu_dai = b.chieu_dai + c.chieu_dai;
        box.chieu_rong = b.chieu_rong + c.chieu_rong;
        box.chieu_cao = b.chieu_cao + c.chieu_cao;
        return box;
    }
}

```

```

public class TestCsharp
{
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Nap chong toan tu trong C#");
        Console.WriteLine("Vi du minh hoa nap chong toan tu");
        Console.WriteLine("-----");
        //tao cac doi tuong Box1, Box2 va Box3
        Box Box1 = new Box();
        Box Box2 = new Box();
        Box Box3 = new Box();
        double the_tich = 0.0;

        // nhap thong tin Box1
        Box1.setChieuDai(6.0);
        Box1.setChieuRong(7.0);
        Box1.setChieuCao(5.0);

        // nhap thong tin Box2
        Box2.setChieuDai(12.0);
        Box2.setChieuRong(13.0);
        Box2.setChieuCao(10.0);

        // tinh va hien thi the tich Box1
        the_tich = Box1.tinhTheTich();
        Console.WriteLine("The tich cua Box1 la: {0}", the_tich);

        // tinh va hien thi the tich Box2
        the_tich = Box2.tinhTheTich();
        Console.WriteLine("The tich cua Box2 la: {0}", the_tich);

        // con hai doi tuong
        Box3 = Box1 + Box2;

        // tinh va hien thi the tich Box3
        the_tich = Box3.tinhTheTich();
        Console.WriteLine("The tich cua Box3 la: {0}", the_tich);

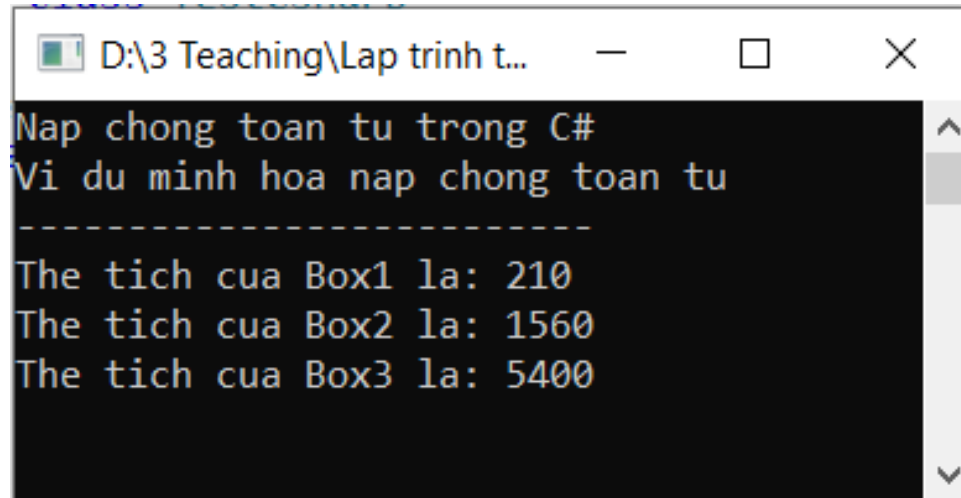
        Console.ReadKey();
    }
}

```

3. NẠP CHỒNG TOÁN TỬ

Triển khai Nạp chồng toán tử trong C#

- Ví dụ dưới đây minh họa cách triển khai nạp chồng toán tử trong C#:
 - Tạo hai lớp có tên lần lượt là **Box**, **TestCsharp** như sau:
 - Lớp **Box**: chứa các thuộc tính và phương thức.
 - Lớp **TestCsharp**: chứa phương thức **main()** để thao tác trên đối tượng **Box**.



```
D:\3 Teaching\Lap trình t...
Nạp chồng toán tử trong C#
Ví dụ minh họa nạp chồng toán tử
-----
The tích của Box1 là: 210
The tích của Box2 là: 1560
The tích của Box3 là: 5400
```


4. INTERFACE

- Một Interface được định nghĩa như là một giao ước có tính chất cú pháp (syntactical contract) mà tất cả lớp kế thừa Interface đó nên theo.
- Interface định nghĩa phần “Là gì” của giao ước và các lớp kế thừa định nghĩa phần “Cách nào” của giao ước đó.
- Các Interface chỉ chứa khai báo của các thành viên. Việc định nghĩa các thành viên là trách nhiệm của lớp kế thừa. Nó thường **giúp ích trong việc cung cấp một Cấu trúc chuẩn mà các lớp kế thừa nên theo.**

4. INTERFACE

Khai báo Interface trong C#

- Các **Interface** được khai báo bởi sử dụng từ khóa **interface** trong C#. Nó tương tự như khai báo lớp.
- Theo mặc định, các lệnh **Interface** là **public**.
- Ví dụ sau minh họa một khai báo **Interface** trong C#:

```
public interface ITransactions
{
    // các thành viên của interface

    //các phương thức
    0 references
    void hienThiThongTinGiaoDich();
    0 references
    double laySoLuong();
}
```

4. INTERFACE

- Ví dụ: tạo 2 lớp có tên lần lượt là **GiaoDichHangHoa**, **TestCsharp** và một interface có tên là **GiaoDich**.

```
public interface GiaoDich
{
    // cac thanh vien cua interface

    //cac phuong thuc
    3 references
    void hienThiThongTinGiaoDich();
    2 references
    double laySoLuong();
}
```

```
class GiaoDichHangHoa : GiaoDich
{
    private string ma_hang_hoa;
    private string ngay;
    private double so_luong;
    0 references
    public GiaoDichHangHoa()
    {
        ma_hang_hoa = " ";
        ngay = " ";
        so_luong = 0.0;
    }

    2 references
    public GiaoDichHangHoa(string c, string d, double a)
    {
        ma_hang_hoa = c;
        ngay = d;
        so_luong = a;
    }

    2 references
    public double laySoLuong()
    {
        return so_luong;
    }

    3 references
    public void hienThiThongTinGiaoDich()
    {
        Console.WriteLine("Ma hang hoa: {0}", ma_hang_hoa);
        Console.WriteLine("Ngay giao dich: {0}", ngay);
        Console.WriteLine("So luong: {0}", laySoLuong());
    }
}
```

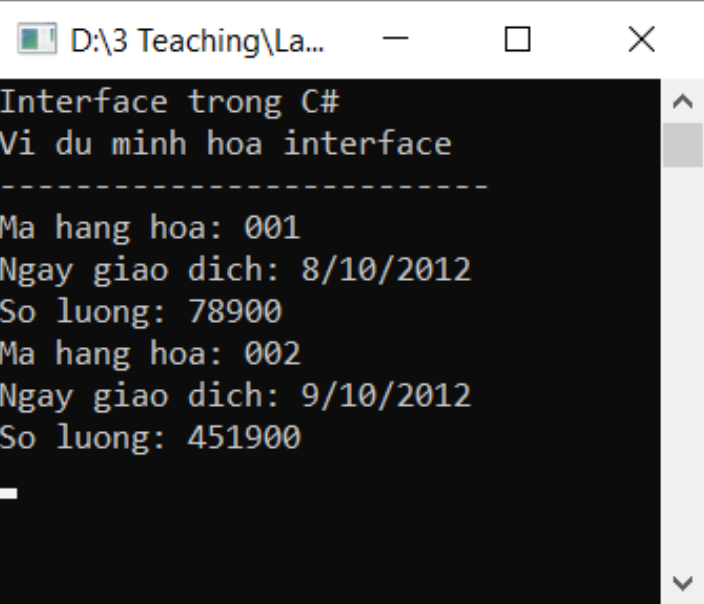
4. INTERFACE

- Ví dụ: tạo 2 lớp có tên lần lượt là **GiaoDichHangHoa**, **TestCsharp** và một interface có tên là **GiaoDich**.

```
public class TestCsharp
{
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Interface trong C#");
        Console.WriteLine("Vi du minh hoa interface");
        Console.WriteLine("-----");

        //tao cac doi tuong GiaoDichHangHoa
        GiaoDichHangHoa t1 = new GiaoDichHangHoa("001", "8/10/2012", 78900.00);
        GiaoDichHangHoa t2 = new GiaoDichHangHoa("002", "9/10/2012", 451900.00);
        t1.hienThiThongTinGiaoDich();
        t2.hienThiThongTinGiaoDich();

        Console.ReadKey();
    }
}
```



```
D:\3 Teaching\La...
Interface trong C#
Vi du minh hoa interface
-----
Ma hang hoa: 001
Ngay giao dich: 8/10/2012
So luong: 78900
Ma hang hoa: 002
Ngay giao dich: 9/10/2012
So luong: 451900
```

BÀI TẬP 1

- Một đơn vị sản xuất gồm có các cán bộ là ***công nhân, kỹ sư, nhân viên***.
 - Mỗi **cán bộ** cần quản lý các thuộc tính: Họ tên, năm sinh, giới tính, địa chỉ.
 - Các **công nhân** cần quản lý: Bậc (công nhân bậc 3/7, bậc 4/7, ...).
 - Các **kỹ sư** cần quản lý: Ngành đào tạo.
 - Các **nhân viên phục vụ** cần quản lý thông tin: công việc.
- **Yêu cầu:**
 1. Xây dựng các lớp **NhanVien**, **CongNhan**, **KySu** kế thừa từ lớp **CanBo**.
 2. Xây dựng các hàm để truy nhập, hiển thị thông tin và kiểm tra về các thuộc tính của các lớp.
 3. Xây dựng lớp **QLCB** cài đặt các phương thức thực hiện các chức năng sau:
 - Nhập thông tin mới cho cán bộ.
 - Tìm kiếm theo họ tên.
 - Hiển thị thông tin về danh sách các cán bộ.
 - Thoát khỏi chương trình.

BÀI TẬP 2

- Một thư viện cần quản lý các **tài liệu** bao gồm: **Sách, Tạp chí, Báo**.
 - Mỗi **tài liệu** có các thuộc tính: Mã tài liệu, Tên nhà xuất bản, Số bản phát hành.
 - Các loại **sách** cần quản lý: Tên tác giả, số trang.
 - Các **tạp chí** cần quản lý: Số phát hành, tháng phát hành.
 - Các **báo** cần quản lý: ngày phát hành.
- **Yêu cầu:**
 1. Xây dựng các lớp để quản lý các loại tài liệu trên sao cho việc sử dụng lại được nhiều nhất.
 2. Xây dựng lớp **QuanLySach** cài đặt các phương thức thực hiện các công việc sau:
 - Nhập thông tin về các tài liệu.
 - Hiển thị thông tin về các tài liệu.
 - Tìm kiếm tài liệu theo loại.
 - Thoát khỏi chương trình.



Q & A