Trường Đại học Công nghiệp Thực phẩm Tp. Hồ Chí Minh

Bộ môn Công nghệ Phần mềm

CHƯƠNG 05

KẾ THỪA VÀ ĐA HÌNH

16/03/2023

ĐẠI HỌC CHÍNH QUY

Nội dung

Đặc biệt hóa (Chuyên biệt hóa) – Tổng quát hóa

5.1. Kế thừa (Inheritance)

5.2. Đa hình (Polymorphism)

5.3. Lớp trừu tượng (Abstract class)

5.4. Interface

Mục tiêu

- ✓ Tìm hiểu mối quan hệ giữa các đối tượng trong thế giới thực.
- ✓ Cách thức mô hình hóa các mối quan hệ này trong chương trình.

Đặc biệt hóa – Tổng quát hóa

✓ **Là quan hệ giữa *cái cụ thể* và *cái chung***

Ví dụ:

- Quan hệ giữa Tam giác cân, Tam giác vuông và Tam giác (chứa 3 điểm A, B, C).
- Tam giác cân, Tam giác vuông **có tất cả các đặc tính chung** của một Tam giác.
- Tam giác cân, Tam giác vuông có **thêm các đặc tính riêng** được xác định.

Đặc biệt hóa – Tổng quát hóa

- ✓ **Là quan hệ có tính *đối ngẫu***

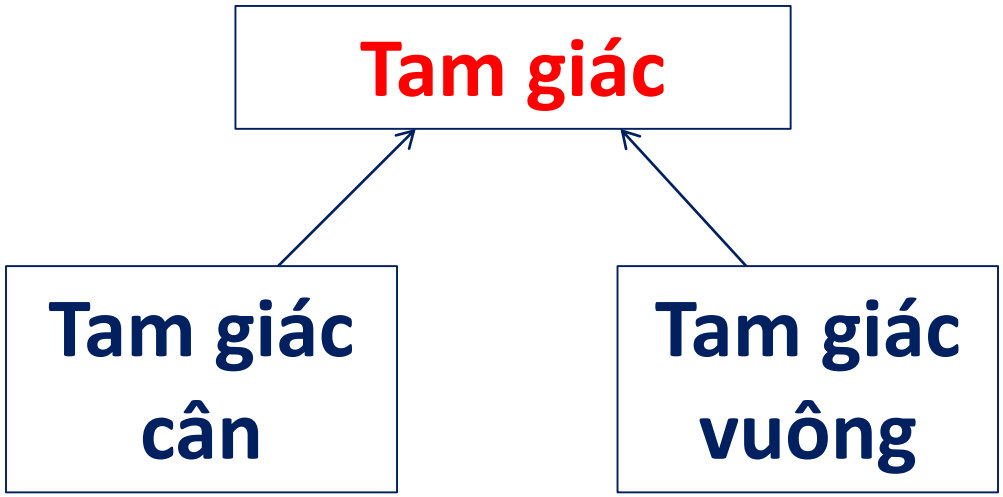
Ví dụ:

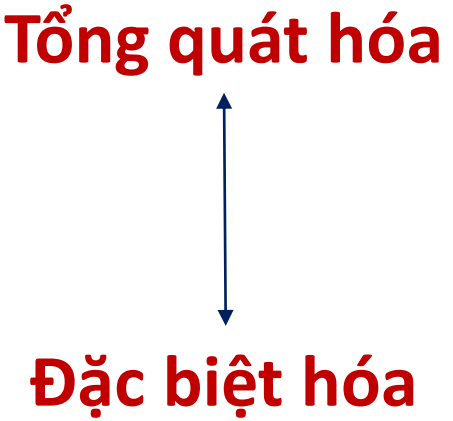
- Tam giác cân, Tam giác vuông là **trường hợp đặc biệt** của Tam giác.
- Ngược lại Tam giác là **trường hợp tổng quát** của Tam giác cân, Tam giác vuông .

Đặc biệt hóa – Tổng quát hóa

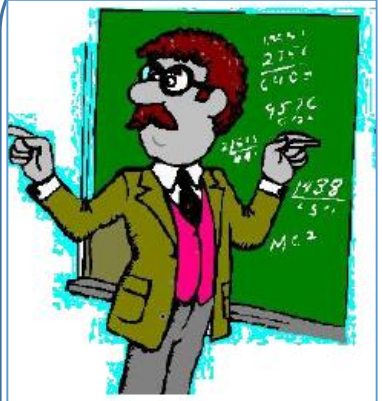
- ✓ **Là quan hệ có tính *phân cấp*** *vì* tạo ra một cây quan hệ.

Ví dụ:





Đặc biệt hóa – Tổng quát hóa



Giáo viên

- ✓ **Thuộc tính:**
  - Họ Tên
  - Mức lương
  - Số ngày nghỉ
- ✓ **Thao tác:**
  - Giảng dạy
  - Tính lương

- ✓ **Thuộc tính:**
  - Họ Tên
  - Mức lương
  - Số ngày nghỉ
  - **Lớp chủ nhiệm**
- ✓ **Thao tác:**
  - Giảng dạy
  - Tính lương
  - **Sinh hoạt CN**

GVCN

Nội dung

Đặc biệt hóa (Chuyên biệt hóa) – Tổng quát hóa

5.1. Kế thừa (Inheritance)

5.2. Đa hình (Polymorphism)

5.3. Lớp trừu tượng (Abstract class)

5.4. Interface

5.1. Kế thừa

- Trong C#, quan hệ đặc biệt hóa, tổng quát hóa thường được thể hiện thông qua **kế thừa**.
- **Xây dựng lớp mới trên cơ sở lớp đã tồn tại.**
- Xử lý các trường hợp trong quan hệ:
  - Đặc biệt hóa – Tổng quát hóa
- Giải quyết vấn đề dư thừa thông tin.

5.1. Kế thừa

- Trong thực tế, **kế thừa** *là việc thừa hưởng lại những gì mà người khác để lại*. **Ví dụ**: con kế thừa tài sản của cha,...
- Trong lập trình, **kế thừa** là cách một lớp có thể thừa hưởng lại những thuộc tính, phương thức từ một lớp khác và sử dụng chúng như là của bản thân mình.

5.1. Kế thừa

Một cách trừu tượng:

- **Kế thừa** là một đặc điểm của ngôn ngữ hướng đối tượng dùng để biểu diễn mối quan hệ **đặc biệt hoá – tổng quát hoá** giữa các lớp.

5.1. Kế thừa

- Kế thừa biểu diễn mối liên hệ **“Cha/con”** giữa các lớp đối tượng.
- Lớp cha (lớp đã tồn tại) được gọi là lớp cơ sở (**Base class**), lớp con được gọi là lớp dẫn xuất (lớp thừa kế) (**Devired class**).

5.1. Kế thừa

Ý nghĩa của tính kế thừa:

- Tính nhất quán: những đối tượng có chung nguồn gốc thì sẽ có những đặc điểm chung giống nhau.
- *Tái sử dụng, chia sẽ thông tin chung và giúp dễ dàng nâng cấp, dễ dàng bảo trì*.
- Lớp kế thừa sẽ có tất cả những thuộc tính và phương thức với từ khóa truy xuất **không phải private** của lớp cha.

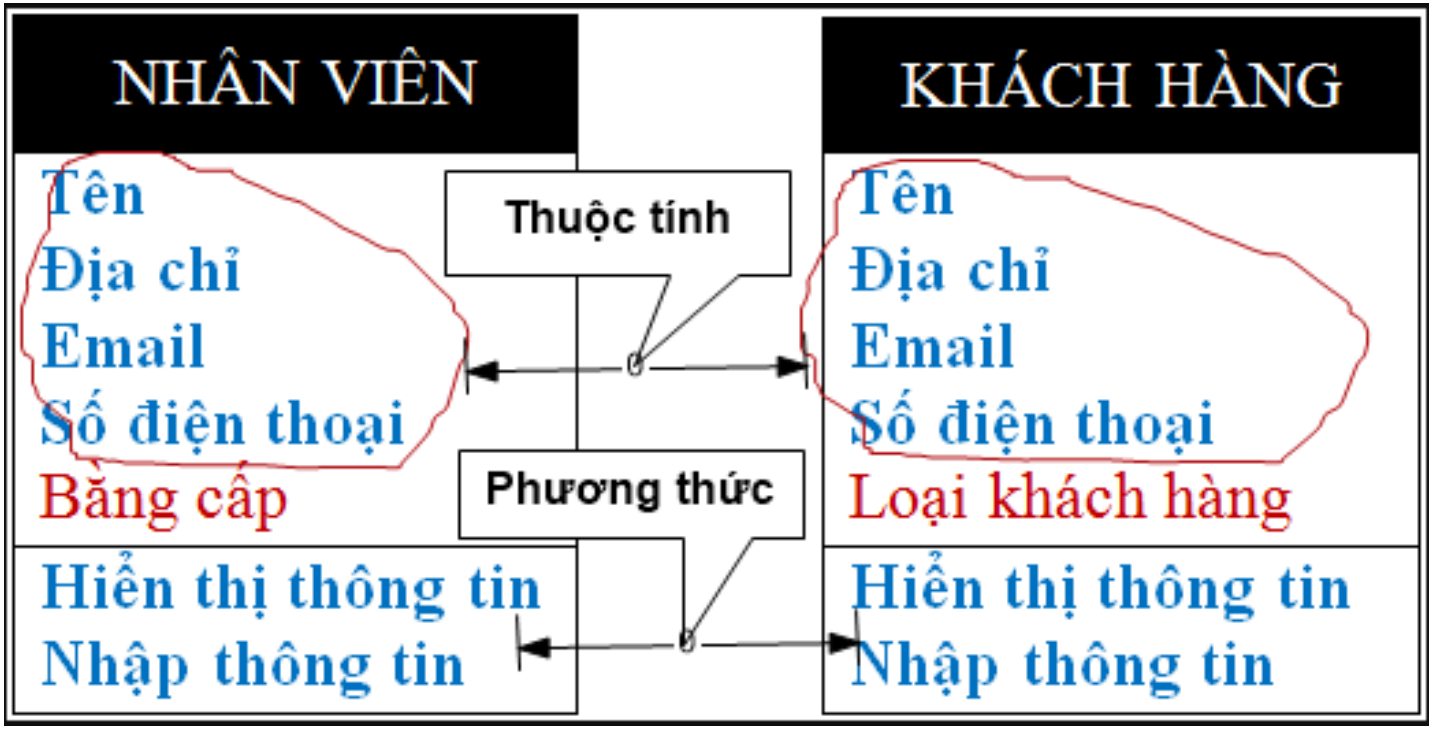
5.1. Kế thừa

Ý nghĩa của tính kế thừa:

- ***Tái sử dụng:***
  - Tái sử dụng code: những phần code chung chỉ cần định nghĩa một lần tại lớp cha (*Base Class*), các lớp con(*Derived Class*) đều có thể sử dụng mà không cần viết lại.
  - Thuận tiện trong việc bảo trì và phát triển. Khi sửa lỗi hay nâng cấp chỉ cần định nghĩa lại ở lớp cha.

5.1. Kế thừa

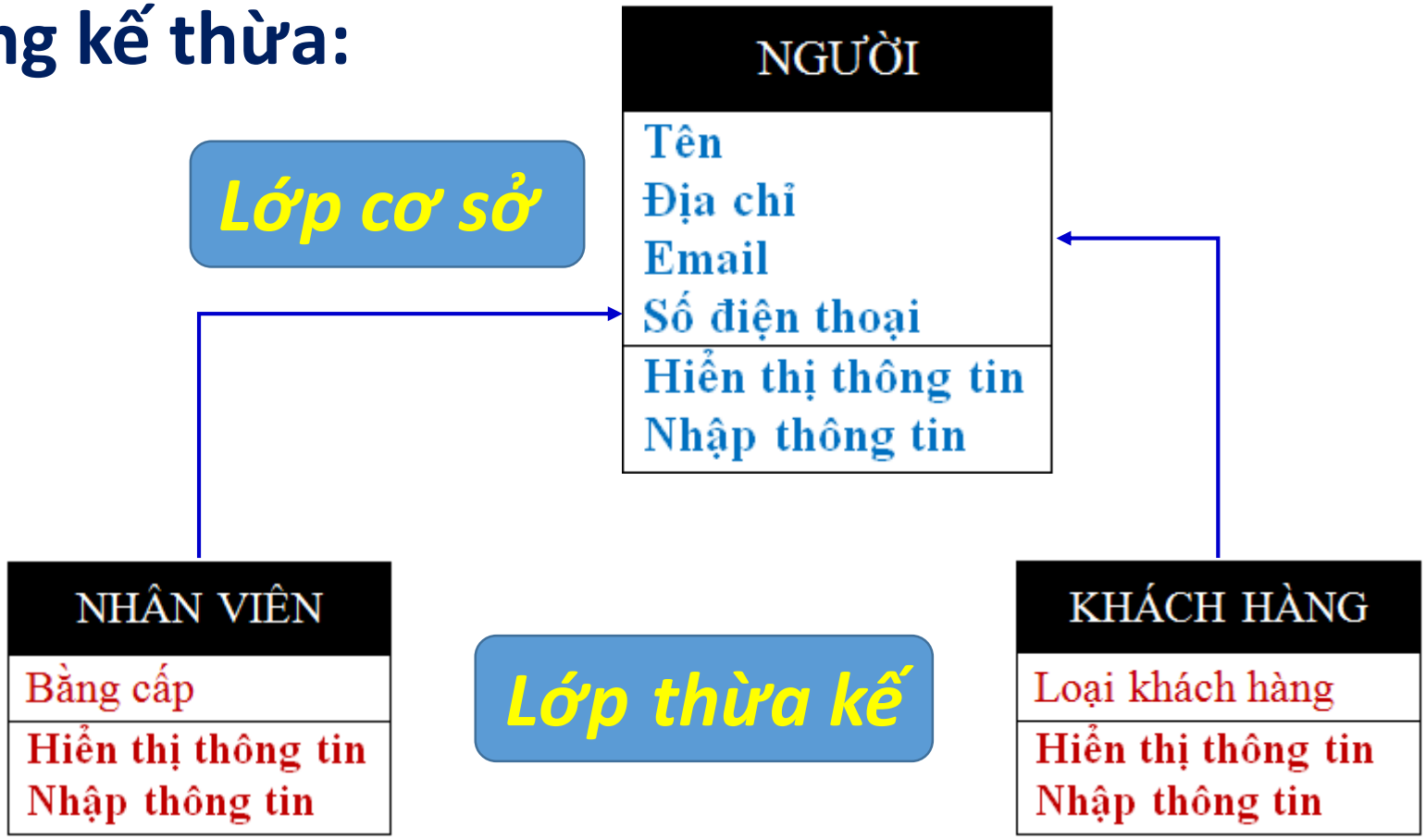
Ví dụ 1: Yêu cầu tạo 2 lớp nhân viên và khách hàng có các thuộc tính và phương thức như hình.



Không kế thừa

5.1. Kế thừa

Ý tưởng kế thừa:



5.1. Kế thừa

Cú pháp khai báo:

```
[<quyen_truy_cap>] class <Lop_cha>
{
    ...
}
[<quyen_truy_cap>] class <Lop_con> : <Lop_cha>
{
    ...
}
```

5.1. Kế thừa

Ví dụ 2:

```
class Nguoi
{
    //Thân lớp
}

class NhanVien:Nguoi
{
    //Thân lớp
}
```

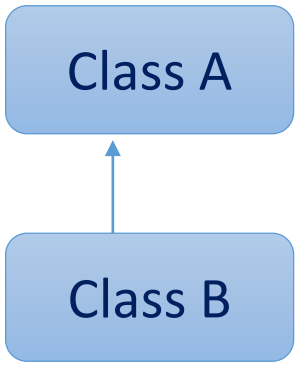
5.1. Kế thừa

Lưu ý:

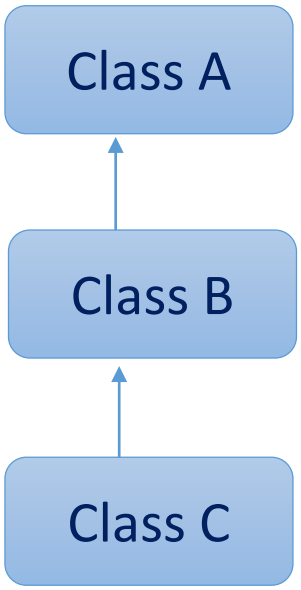
- Lớp con chỉ kế thừa được, chỉ sử dụng được các thuộc tính và phương thức có phạm vi truy cập **không là private** (non-private) của lớp cha.
- Mọi lớp trong .NET đều kế thừa từ lớp Object.

5.1. Kế thừa

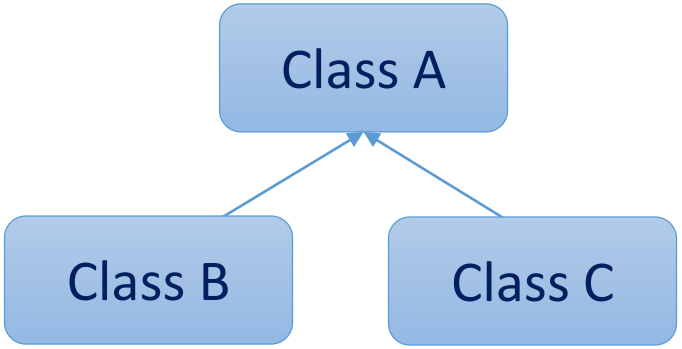
Có 3 kiểu kế thừa: *đơn kế thừa, kế thừa nhiều cấp, kế thừa thứ bậc*.



Đơn kế thừa



Kế thừa nhiều cấp



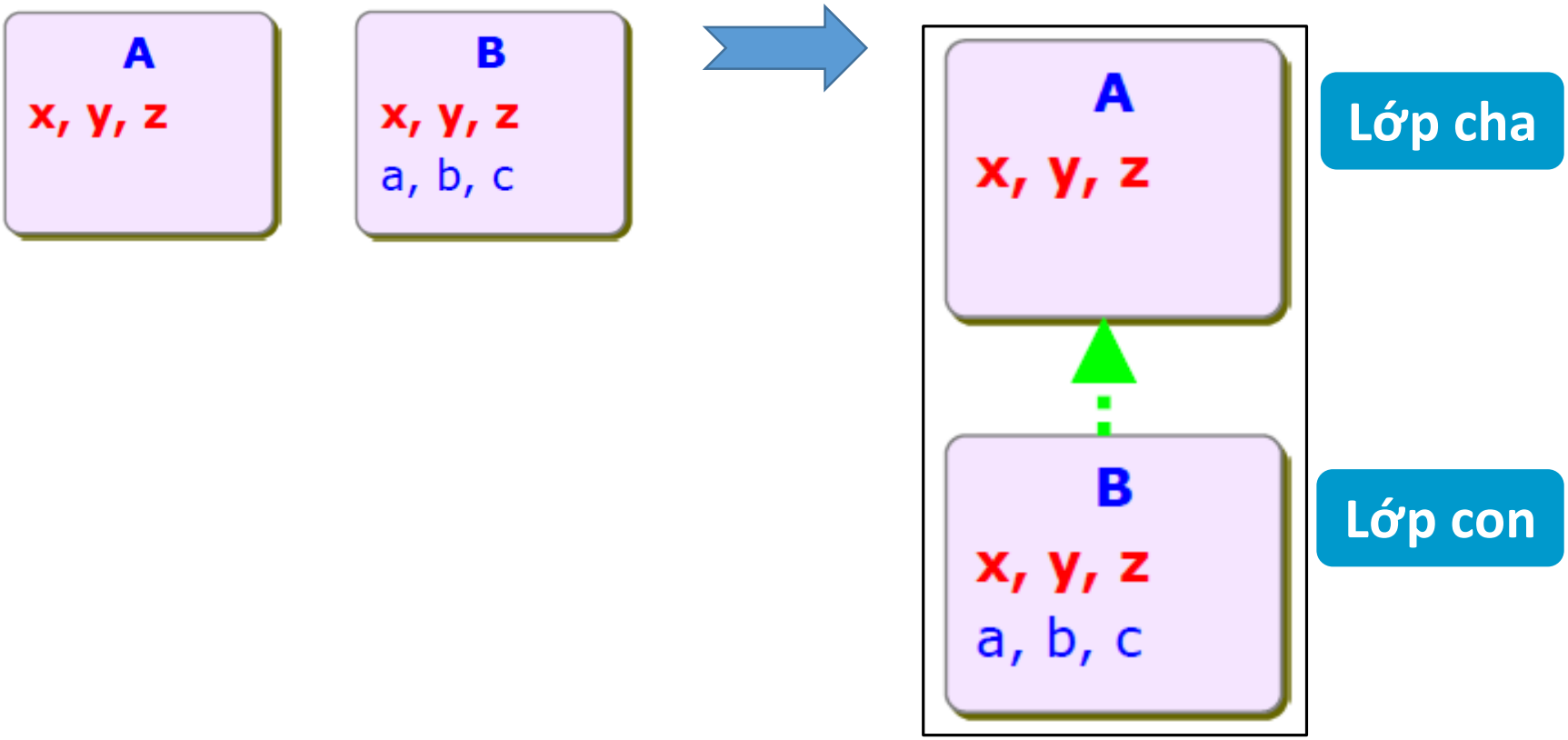
Kế thừa thứ bậc

Lưu ý: Trong C# không hỗ trợ **đa kế thừa**.

5.1. Kế thừa

Có 2 trường hợp kế thừa:

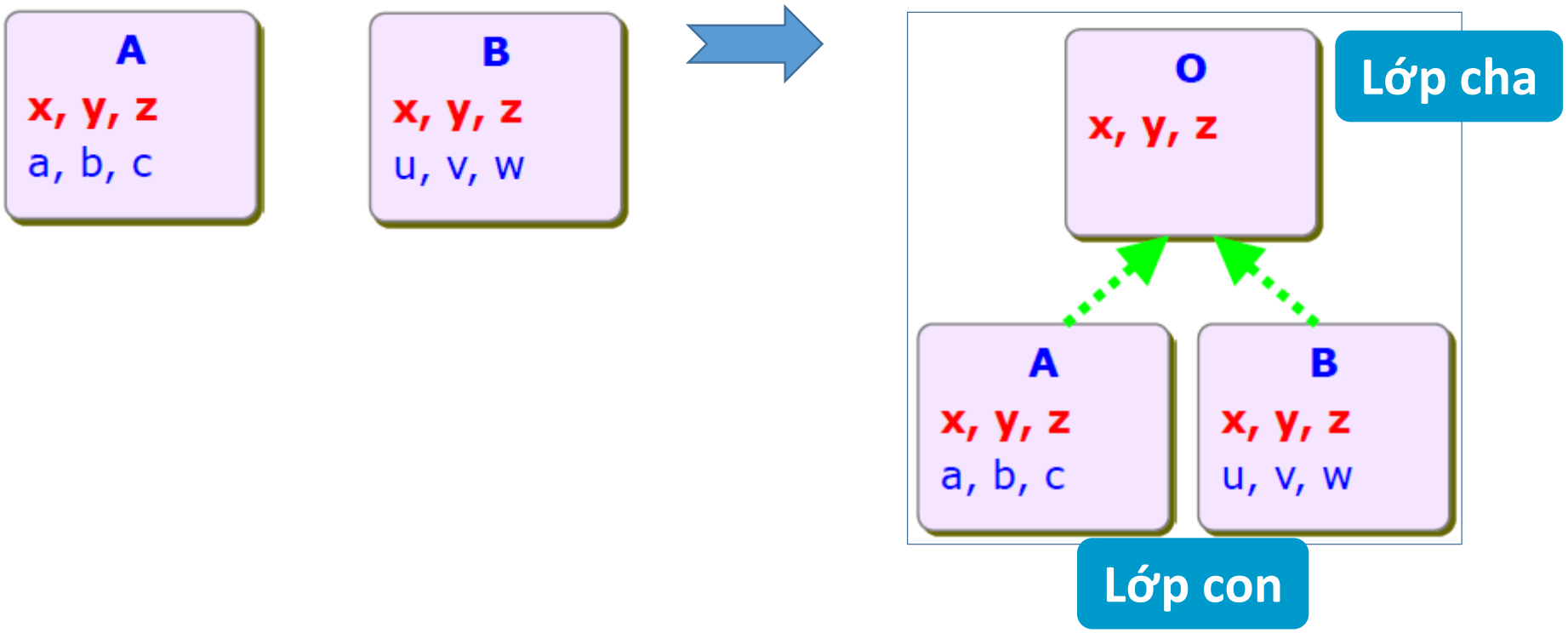
TH1: Thông tin lớp này chứa lớp kia



5.1. Kế thừa

Có 2 trường hợp kế thừa:

TH2: Phần giao của các lớp khác rỗng



5.1. Kế thừa – Phương thức khởi tạo

- ✓ Phương thức khởi tạo (constructor) và phương thức hủy (destructor) **không được kế thừa**.
- ✓ **Nếu không có lời gọi tường minh** đến constructor của lớp cha thì trình biên dịch sẽ **tự động gọi constructor mặc định của lớp cha trước khi thực hiện các lệnh trong constructor của lớp con**.

5.1. Kế thừa – Phương thức khởi tạo

Ví dụ 3:

```
public class Base_class
{
    protected int _x;
    public Base_class()
    {
        _x = 0;
        Console.WriteLine("Khoi tao mac dinh cua Base duoc goi");
    }
    public void Xuat()
    {
        Console.WriteLine("Gia tri _x = {0}", _x);
    }
}
```

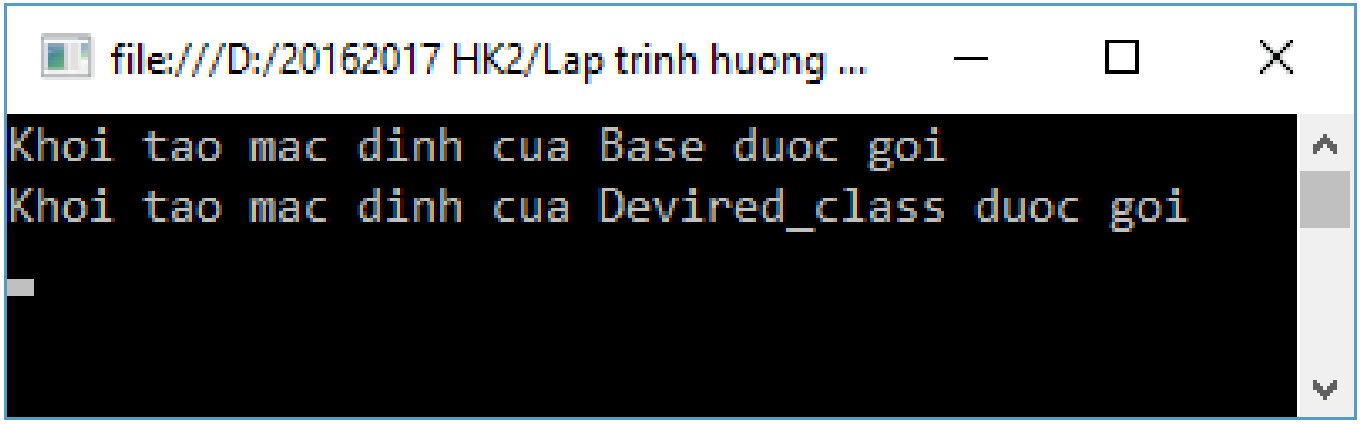
5.1. Kế thừa – Phương thức khởi tạo

Ví dụ 3:

```
public class Devired_class : Base_class
{
    int _y;
    public Devired_class()
    {
        _y = 0;
        Console.WriteLine("Khoi tao mac dinh cua
                           Devired_class duoc goi");
    }
}
```

5.1. Kế thừa – Phương thức khởi tạo

```
static void Main()
{
    Devired_class dc_o = new Devired_class();
    Console.ReadLine();
}
```



5.1. Kế thừa – Phương thức khởi tạo

- ✓ Nếu **lớp cha có phương thức khởi tạo có tham số** thì **phương thức khởi tạo có tham số tương ứng** của lớp con được tạo theo cú pháp:

```
public <Ten_lop_con>(DSTS của lớp con):
                              base(DSTS của lớp cha)
{
    ...
}
```

5.1. Kế thừa – Phương thức khởi tạo

Ví dụ 3:

```
public Base_class(int x)
{
    this._x = x;
    Console.WriteLine("Khoi tao co tham so cua Base duoc goi");
}
```

```
public Devired_class(int x, int y)
{
   Console.WriteLine("Khoi tao 2 tham so cua
                                Devired_class duoc goi");

}
```

5.1. Kế thừa – Phương thức khởi tạo

Ví dụ 3:

```
public Base_class(int x)
{
    this._x = x;
    Console.WriteLine("Khoi tao co tham so cua Base duoc goi");
}
```

```
public Devired_class(int x, int y) : base(x)
{
   Console.WriteLine("Khoi tao 2 tham so cua
                                Devired_class duoc goi");
   this._y = y;
}
```

5.1. Kế thừa – Phương thức khởi tạo

```
//Phuong thuc xuat cua Devired_class
public void Xuat()
{
    Console.WriteLine("Gia tri _y = {0}", _y);
}
```

```
static void Main()
{
    Devired_class dc_o = new Devired_class(8,6);
    dc_o.Xuat();
    Console.ReadLine();
}
```

5.1. Kế thừa – Phương thức khởi tạo

- ✓ Nếu **phương thức khởi tạo của lớp con không chỉ định :base(DSTS của lớp cha)** thì sẽ gọi phương thức khởi tạo mặc định của lớp cha.

Ví dụ:

```
public Devired_class(int y)
{
    this._y = y;
    Console.WriteLine("Khoi tao 1 tham so
                cua Devired_class duoc goi");
}
```

5.1. Kế thừa – Phương thức khởi tạo

```
//Phuong thuc xuat cua Devired_class
public void Xuat()
{
    Console.WriteLine("Gia tri _y = {0}", _y);
}
```

```
static void Main()
{
    Devired_class dc_o = new Devired_class(8);
    dc_o.Xuat();
    Console.ReadLine();
}
```

Bài tập 1

Công ty du lịch ABC quản lý thông tin 2 loại chuyến xe gồm:

- Chuyến xe nội thành: Mã số chuyến, Họ tên tài xế, số xe, số tuyến, số km đi được, doanh thu.
- Chuyến xe ngoại thành: Mã số chuyến, Họ tên tài xế, số xe, nơi đến, số ngày đi được, doanh thu.

Yêu cầu: Xây dựng các lớp với chức năng thừa kế (với các phương thức: khởi tạo mặc định, khởi tạo đầy đủ tham số, nhập, xuất).

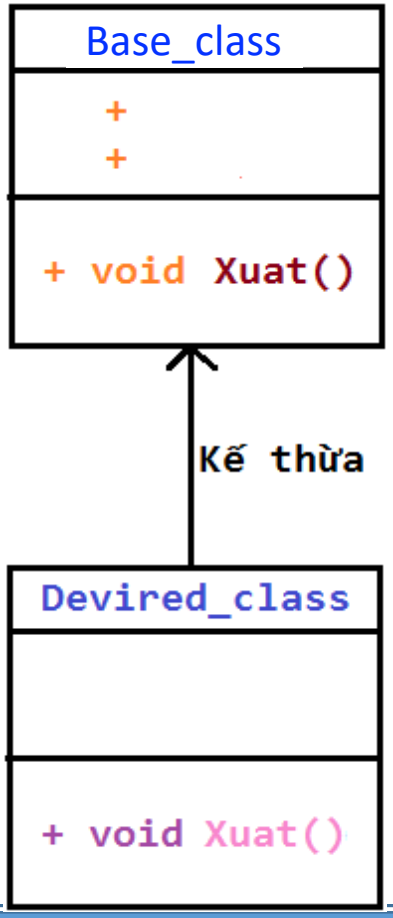
5.1. Kế thừa - Phương thức trùng tên

- ✓ Giả sử lớp Base_class có phương phức Xuat(). Lớp Devired_class kế thừa nên cũng sẽ nhận được phương thức này.
- ✓ Trong lớp Devired_class ta cũng định nghĩa 1 phương thức tên Xuat().
- ✓ **Vậy câu lệnh sau sẽ gọi phương thức Xuat() nào?**

```
Devired_class dc_o = new Devired_class(8);
dc_o.Xuat();
```

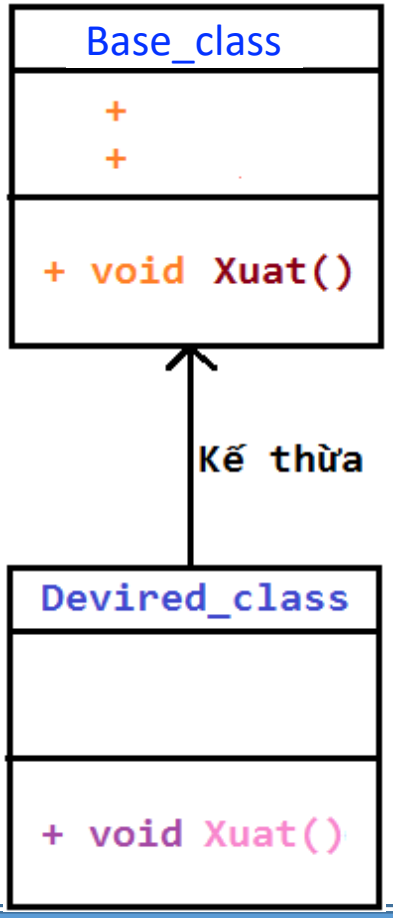
5.1. Kế thừa - Phương thức trùng tên

```
Devired_class dc_o = new Devired_class(8);
dc_o.Xuat();
```



5.1. Kế thừa - Phương thức trùng tên

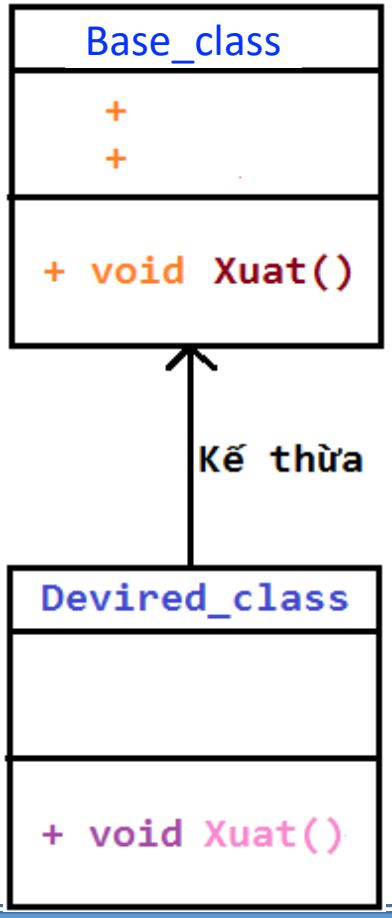
```
Devired_class dc_o = new Devired_class(8);
dc_o.Xuat();
```



- C# sẽ gọi phương thức **Xuat()** của lớp Devired_class định nghĩa.
- Đồng thời cũng đưa ra 1 cảnh báo khi biên dịch.
- C# hỗ trợ từ khoá **new** nhằm đánh dấu đây là 1 hàm mới và **hàm kế thừa từ lớp cha sẽ bị che**.

5.1. Kế thừa - Phương thức trùng tên

```
Devired_class dc_o = new Devired_class(8);
dc_o.Xuat();
```



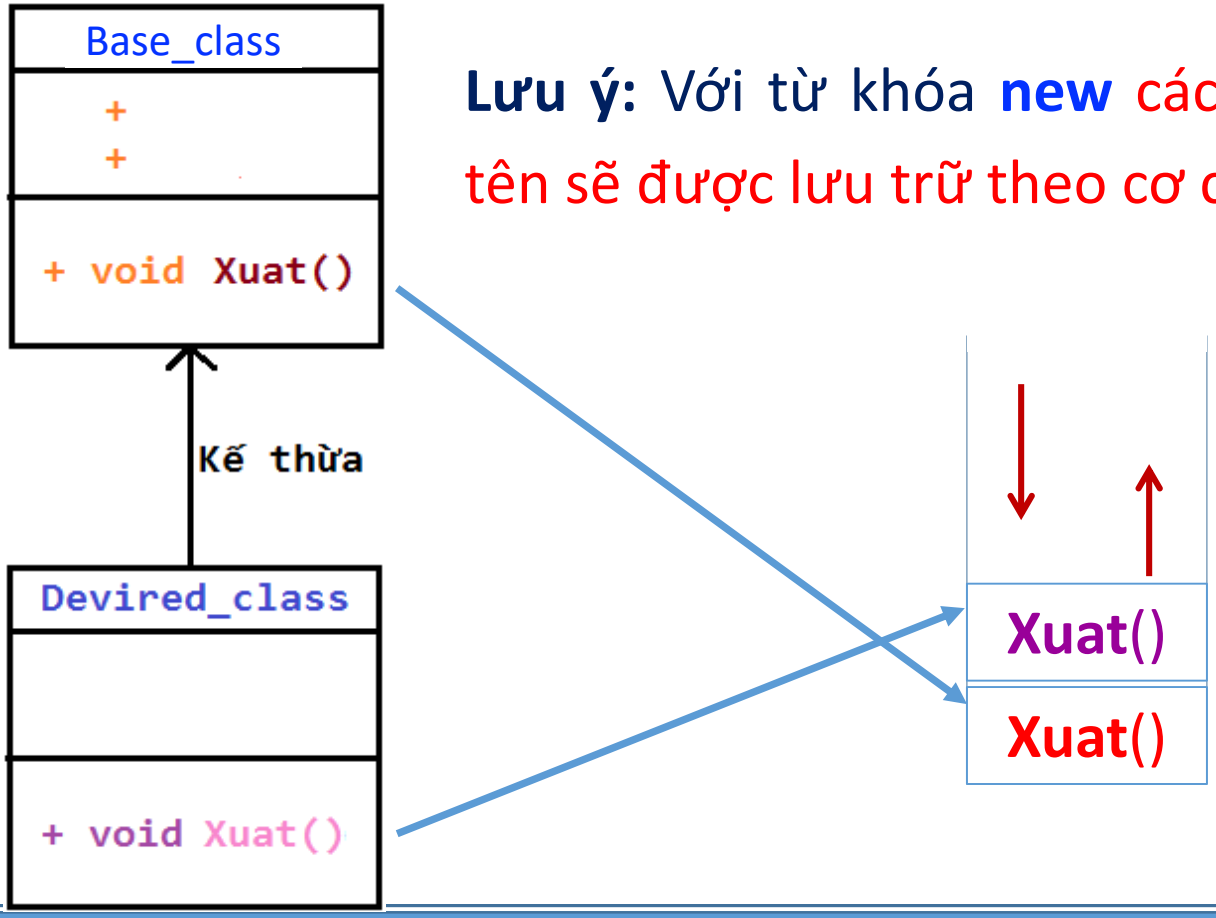

```
//Xuat() của Devired_class
public new void Xuat()
{

}
```

Lưu ý: Từ khoá này **chỉ làm tường minh** khai báo của hàm **Xuat()** mà không làm thay đổi kết quả khi chạy chương trình.

5.1. Kế thừa - Phương thức trùng tên

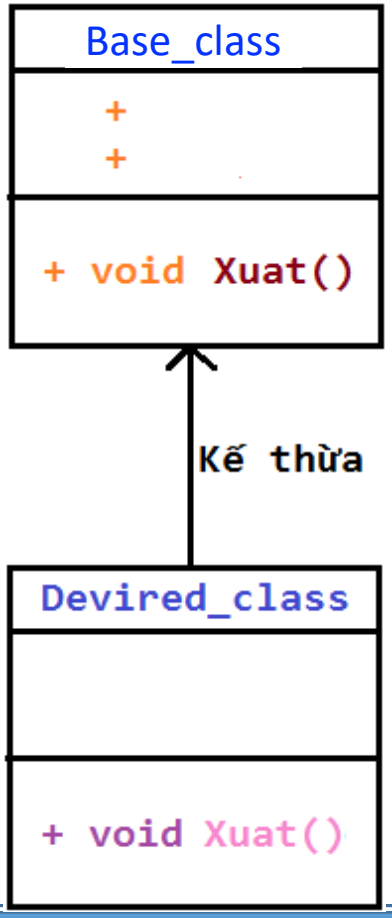
```
Devired_class dc_o = new Devired_class(8);
dc_o.Xuat();
```



Lưu ý: Với từ khóa **new** các phương thức cùng tên sẽ được lưu trữ theo cơ chế Stack.

5.1. Kế thừa - Phương thức trùng tên

```
Devired_class dc_o = new Devired_class(8);
dc_o.Xuat();
```



Khi đó: Có thể gọi phương thức Xuat() của lớp cha bên trong lớp con.

```
//Phuong thuc xuat cua Devired_class
public new void Xuat()
{
    base.Xuat();
    Console.WriteLine("Gia tri _y = {0}",_y);
}
```

5.1. Kế thừa - Cấp phát vùng nhớ

- **Không có quan hệ kế thừa**

```
Base_class bs = new Devired_class();→ Lỗi
```

- **Nếu có quan hệ kế thừa có thể khai báo**

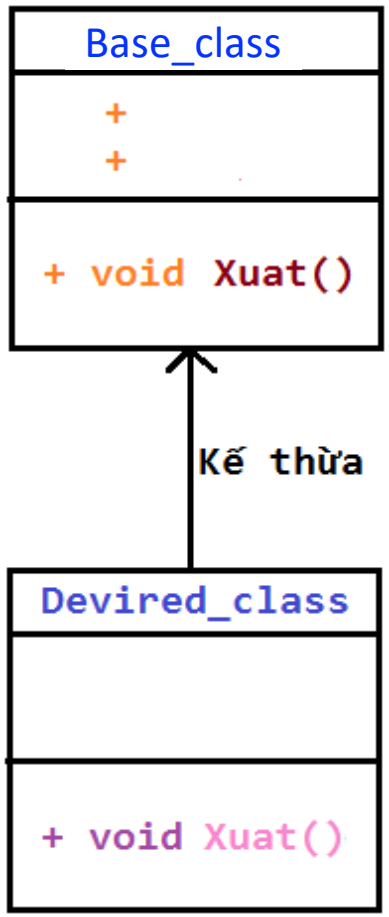
```
Base_class bs = new Devired_class();
```

nhưng nếu khai báo ngược lại sẽ **LỖI**:

```
Devired_class bs = new Base_class();
```

Ghi nhớ: *"Một đối tượng thuộc lớp cha có thể tham chiếu đến vùng nhớ của đối tượng thuộc lớp con nhưng ngược lại thì không"*

5.1. Kế thừa - Cấp phát vùng nhớ




```
Base_class dc_o = new Devired_class();
dc_o.Xuat();
```

Câu lệnh trên sẽ gọi phương thức Xuat() của lớp nào?

Câu trả lời → Phần tiếp theo

Nội dung

Đặc biệt hóa (Chuyên biệt hóa) – Tổng quát hóa

5.1. Kế thừa (Inheritance)

5.2. Đa hình (Polymorphism)

5.3. Lớp trừu tượng (Abstract class)

5.4. Interface

5.2. Đa hình (Polymorphism)

- **Tính đa hình** (Polymorphism) được hiểu là *trong từng trường hợp, hoàn cảnh khác nhau thì đối tượng có hình thái khác nhau tùy thuộc vào từng ngữ cảnh*.

Ví dụ: Khi bạn ở trong trường học là sinh viên thì bạn có nhiệm vụ học, nghe giảng,..., nhưng khi bạn ở nhà thì bạn lại đóng vai trò là thành viên trong gia đình và bạn có nhiệm vụ phải làm việc nhà, ...

5.2. Đa hình (Polymorphism)

Đa hình gồm có 2 loại:

- Đa hình khi biên dịch (Compile-time Polymorphism) – Biên dịch tĩnh (Static Binding or Early Binding).
- Đa hình khi chạy (Run-time Polymorphism) – Biên dịch động (Dynamic Binding or Late Binding).

5.2. Đa hình (Polymorphism)

Đa hình khi biên dịch (Compile-time Polymorphism):

C# cung cấp hai kỹ thuật để triển khai đa hình khi biên dịch:

- Nạp chồng hàm (Function overloading)
- Nạp chồng toán tử (Operator overloading)

5.2. Đa hình (Polymorphism)

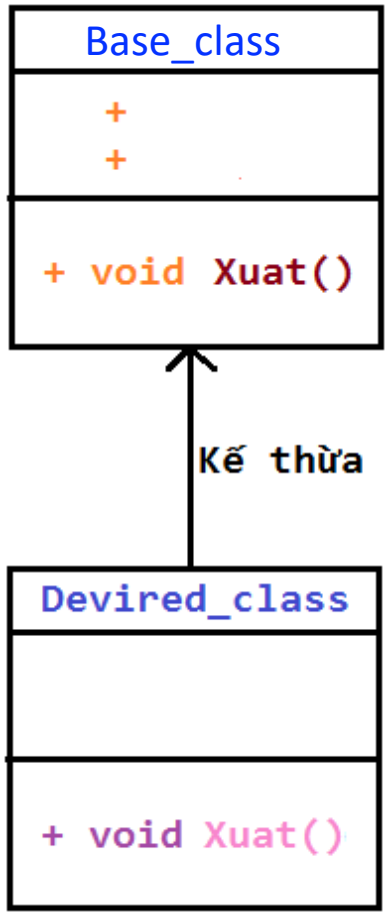
Đa hình khi chạy (Run-time Polymorphism):

Tức là chỉ khi chạy mới biết sử dụng phương thức nào. Để thể hiện được tính đa hình khi chạy thì cần 2 điều kiện sau:

- Các lớp phải có quan hệ kế thừa với cùng 1 lớp cha nào đó.
- Phương thức đa hình phải được ghi đè (override) ở các lớp con.

5.2. Đa hình (Run-time Polymorphism)

Ví dụ 3:

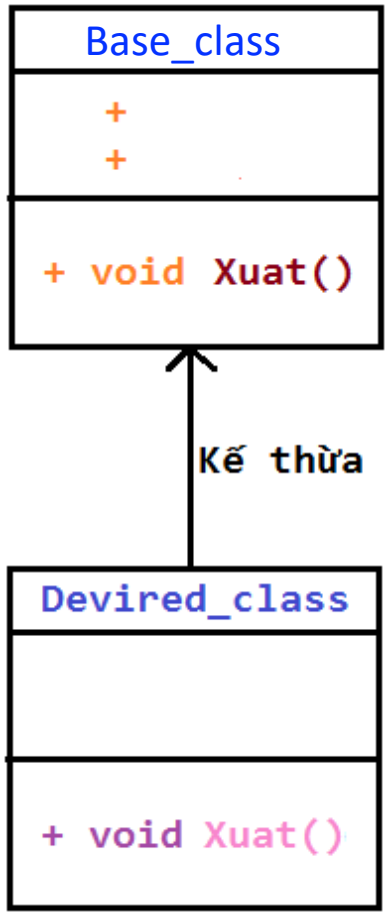



```
Base_class dc_o = new Devired_class();
dc_o.Xuat();
```

Ta mong muốn chương trình sẽ gọi đúng phương thức **Xuat()** của lớp đã được cấp phát vùng nhớ. Nhưng thực tế không phải vậy.

5.2. Đa hình (Run-time Polymorphism)

Ví dụ 3:



Giải quyết:

- Khai báo phương thức Xuat() của lớp cha là phương thức ảo (**virtual**).
- **override** phương thức **Xuat()** của lớp cha (Base_class) tại lớp con (Devired_class) .

5.2. Đa hình (Run-time Polymorphism)

Cú pháp:

- **Lớp cha: khai báo thêm từ khóa virtual**

```
public virtual <Kiểutrảvề> <Tên_PT>(DSTS)
{…}
```

- **Lớp con: sử dụng từ khóa override.**

```
public override <Kiểutrảvề> <Tên_PT>(DSTS)
{…}
```

Bài tập 2

1. Xây dựng lớp **nhân viên** với các thông tin: mã nhân viên, tên nhân viên, hệ số lương, phòng ban, số ngày làm việc; Phương thức: Xếp loại, Tính thu nhập.

Biết rằng: Nếu số ngày làm việc > 25 loại A, nếu số ngày làm việc > 22 loại B, còn lại loại C.

Lương = 1210 * hệ số lương * hệ số thi đua

Trong đó hệ số thi đua là 1.00; 0.75 và 0.50 tương ứng với loại thi đua là A, B, C.

Bài tập 2

2. Xây dựng lớp Cán bộ quản lý. Biết rằng cán bộ quản lý cũng là một nhân viên nhưng có thêm các đặc điểm sau: Mỗi cán bộ quản lý sẽ có thêm thuộc tính chức vụ và hệ số chức vụ.

Lương = hệ số lương *1210 * hệ số thi đua

+ hệ số chức vụ * 1100

Nội dung

Đặc biệt hóa (Chuyên biệt hóa) – Tổng quát hóa

5.1. Kế thừa (Inheritance)

5.2. Đa hình (Polymorphism)

5.3. Lớp trừu tượng (Abstract class)

5.4. Interface

5.3. Lớp trừu tượng (Abstract class)

- C# còn thể hiện Đa hình động (đa hình run-time) thông qua một khái niệm mới, đó là **Lớp trừu tượng (Abstract class)**.

5.3. Lớp trừu tượng (Abstract class)

Khái niệm:

- **Lớp trừu tượng (abstract class)** thực chất là một lớp cơ sở mà các lớp khác có cùng bản chất (kiểu, loại, nhiệm vụ) có thể dẫn xuất từ nó.
- Lớp trừu tượng là một lớp chưa hoàn chỉnh.
- Từ khóa **abstract** được dùng để khai báo lớp trừu tượng.

5.3. Lớp trừu tượng (Abstract class)

- Từ khoá ***abstract*** cũng có thể được dùng cho phương thức – ***phương thức trừu tượng*** (abstract method).
- Phương thức trừu tượng chỉ được khai báo trong các lớp trừu tượng và có ý nghĩa tương tự phương thức ảo (virtual method) ngoại trừ nó không có phần định nghĩa (phần thân).

5.3. Lớp trừu tượng (Abstract class)

Cách khai báo:

#1. Đối với lớp trừu tượng

```
[<quyen_truy_cap>] abstract class <Ten_lop>
{
      ...
}
```

Trong đó: **quyen_truy_cap**: là *public* hay *protected*.

5.3. Lớp trừu tượng (Abstract class)

Cách khai báo:

#2. Đối với phương thức trừu tượng

```
<quyen_truy_cap> abstract <kieu_tra_ve>
                    <Ten_phuong_thuc>([<danh_sach_tham_so>]);
```

Trong đó: **quyen_truy_cap**: là *public* hay *protected*.

5.3. Lớp trừu tượng (Abstract class)

Ví dụ 4:

```
public abstract class Animal
{
    //Field, method
    public abstract void Eat();
}
```

5.3. Lớp trừu tượng - Một số lưu ý

- ***abstract class*** không thể là một sealed class, vì sealed class không cho lớp khác kế thừa nó. Và cũng không được khai báo với từ khóa **static**.

```
public abstract sealed class Animal{ ...}
public abstract static class Animal{ ...}

//Error
```

5.3. Lớp trừu tượng - Một số lưu ý

- Lớp con kế thừa từ *abstract class* bắt buộc **override tất cả *abstract method*** của lớp cha, tức là sẽ phải định nghĩa cho các phương thức mới khai báo prototype của lớp cha.

5.3. Lớp trừu tượng - Một số lưu ý

- ***abstract class*** nên chứa ít nhất một *abstract method*, nếu chúng ta khai báo *abstract class* mà không có *abstract method* nào thì chương trình cũng không có lỗi gì xảy ra, nhưng như vậy thì không đúng với tư tưởng của *abstract class*.

5.3. Lớp trừu tượng - Một số lưu ý

- ***abstract class*** có thể có 3 loại phương thức: *abstract method, virtual method, normal method*.

```
public abstract class Animal
{
    public abstract void AnimalSound();
    public virtual void Eat()
    {
        //...
    }
    public void Run()
    {
        //...
    }
}
```

5.3. Lớp trừu tượng - Một số lưu ý

- Không thể tạo ra đối tượng (object instance) từ một *abstract class*.

```
Animal a = new Animal(); //Error
```

5.3. Lớp trừu tượng - Một số lưu ý

- ***abstract method*** chỉ được khai báo trong *abstract class*, và chỉ khai báo prototype, không có phần định nghĩa.
- ***abstract method*** phải có phạm vi truy cập là **public** hoặc **protected** để lớp con có thể override, không được là **private**.
- Quyền truy cập của *abstract method* phải giống nhau trong phần khai báo ở lớp cha và phần định nghĩa ở lớp con.

5.3. Lớp trừu tượng - Một số lưu ý

- ***abstract method*** không được khai báo sử dụng từ khóa **virtual**. Bởi vì bản thân *abstract method* đã bao hàm khái niệm **virtual**.

```
public abstract class Animal
{
  protected abstract virtual void AnimalSound();
}
//Error
```

5.3. Lớp trừu tượng - Minh họa

Thiết kế lớp cho chương trình **tính điểm trung bình (DTB) và quy đổi điểm hệ 4 (điểm chữ)**. Biết rằng mỗi môn học đều có các thông tin sau: *Mã môn học, tên môn học, số tín chỉ*. Các môn học được chia làm 3 loại:

- Lý thuyết: có 2 cột điểm là điểm tiểu luận và điểm cuối kỳ.

  DTB = Điểm tiểu luận*0.3 + Điểm cuối kỳ*0.7

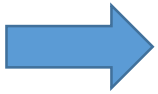
- Thực hành: có 3 cột điểm kiểm tra. **DTB tính bằng trung bình cộng các bài kiểm tra**.

- Đồ án: có điểm của giáo viên hướng dẫn (GVHD) và giáo viên phản biện (GVPB). **DTB = (điểm GVHD + điểm GVPB)/ 2**

5.3. Lớp trừu tượng - Minh họa

Đặc điểm chung của một môn học:

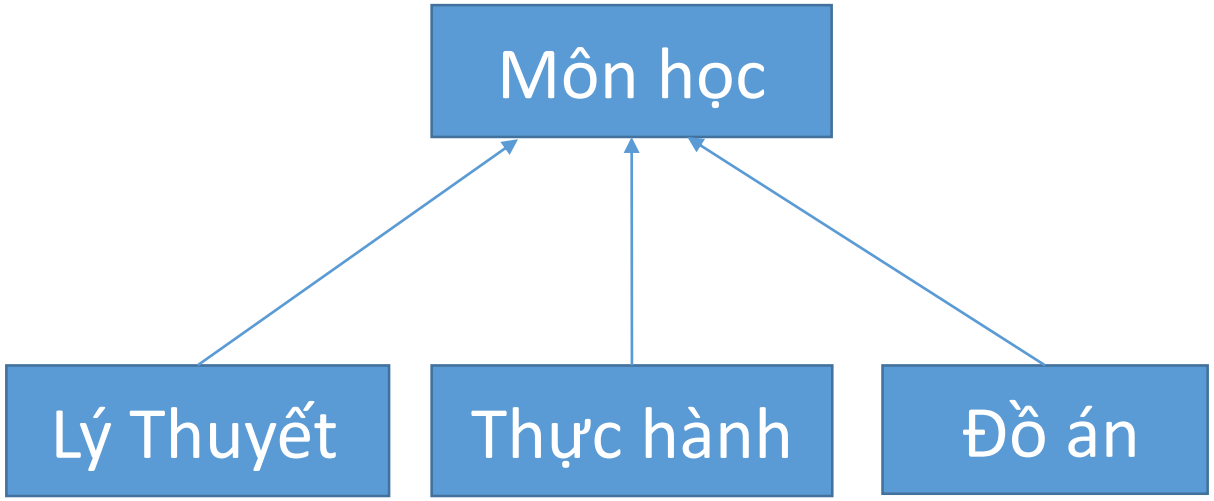
- Mã môn học,
- Tên môn học,
- Số tín chỉ
- Tính điểm Trung bình (DTB())
- Quy đổi điểm chữ (DiemChu())



Xây dựng lớp kế thừa (Lớp môn học)

5.3. Lớp trừu tượng - Minh họa

Sơ đồ thiết kế lớp



5.3. Lớp trừu tượng - Minh họa

Phương thức DTB()

- Không biết là môn nào
- Không biết điểm thành phần
- Không xác định được công thức tính điểm

Xây dựng phương thức DTB là Phương thức trừu tượng

Phương thức DiemChu()

+ Nếu Điểm TB < 4: F
+ Điểm TB <= 4.9: D
+ Điểm TB <= 5.4: D+
+ Điểm TB <= 6.4: C
+ Điểm TB <= 6.9: C+
+ Điểm TB <= 7.9: B
+ Điểm TB <= 8.4: B+
+ Điểm TB <= 9.4: A

Ngược lại là A+

Sự khác nhau giữa abstract và virtual method

abstract method	virtual method
Lớp con bắt buộc phải override abstract method ở lớp cha.	Lớp con không bắt buộc phải override virtual method ở lớp cha. Nếu định nghĩa của virtual method ở lớp cha đã phù hợp thì có thể sử dùng. Nếu chưa phù hợp thì lớp con có quyền override virtual method ở lớp cha.
Tuyệt đối chỉ được khai báo là prototype ở lớp cha không có thân hàm, tức là chưa được định nghĩa.	Cần phải được định nghĩa ở lớp cha.
Chỉ được sử dụng trong abstract class.	Có thể sử dụng được trong cả abstract class và normal class.

Bài tập 3

Thiết kế và xây dựng lớp cho chương trình tính tiền phòng hàng tháng cho một cơ sở cho thuê phòng trọ. Biết rằng mỗi phòng đều có thông tin sau: *Mã số phòng, số người ở, số điện và số nước*. Cơ sở chia phòng ra làm 2 loại khác nhau:

- Phòng loại A: Có thêm thông tin về số lần người thân thăm và ở lại qua đêm (SoNguoithan) và tiền phòng được tính như sau:

Tiền phòng =1400 + 2*Số điện + 8*Số nước + 50*SoNguoithan

- Phòng loại B: Có thêm thông tin khối lượng giặt ủi (giatui), và số máy sử dụng internet (somay).

Tiền phòng =2000 + 2*Số điện + 8*Số nước + Giatui*5 + Somay*100

Nội dung

Đặc biệt hóa (Chuyên biệt hóa) – Tổng quát hóa

5.1. Kế thừa (Inheritance)

5.2. Đa hình (Polymorphism)

5.3. Lớp trừu tượng (Abstract class)

5.4. Interface

5.4. Interface

- Một lớp chỉ có thể kế thừa từ một superclass trực tiếp (đơn kế thừa). Để hỗ trợ **đa thừa kế** (multi-inheritance) trong lập trình hướng đối tượng, C# thay thế bằng ***Interface***.
- Interface như là một *"mặt nạ"*, *một* ***bản thiết kế*** *của một lớp*, cùng cách thức hoạt động, **nhưng khác bản chất bên trong**.
- Interface cũng biểu diễn mối quan hệ IS-A (thừa kế), và **không thể khởi tạo đối tượng**.

Đặc điểm Interface

- Interface **không có constructor**, cũng không có destructor.
- **Không thể** khai báo phạm vi truy cập cho các thành phần bên trong interface. Các thành phần này sẽ mặc định là **public**.
- Interface **không chứa field** dù là biến tĩnh vẫn không được.

Đặc điểm Interface

- Phương thức: **chỉ chứa khai báo không chứa phần định nghĩa** (phương thức trừu tượng). Khi khai báo **không khai báo từ khoá abstract**.
- Các lớp có thể thực thi nhiều interface cùng lúc.
- Một **interface có thể kế thừa nhiều interface khác** nhưng **không thể kế thừa bất kỳ lớp nào**.

Mục đích

- Vì C# không hỗ trợ đa kế thừa nên interface ra đời như là 1 giải pháp cho việc đa kế thừa này.
- Trong 1 hệ thống việc trao đổi thông tin giữa các thành phần **cần được đồng bộ và có những thống nhất chung**. Vì thế dùng interface sẽ giúp đưa ra **những quy tắc chung** mà bắt buộc các thành phần trong hệ thống này phải làm theo mới có thể trao đổi với nhau được.

Khai báo và Sử dụng

Cú pháp:

```
interface <tên interface>
{
    //Khai báo các thành phần interface
}
```

Lưu ý: Để tránh nhầm lẫn với lớp kế thừa thì khi đặt tên interface người ta thường thêm tiền tố "**I**" để nhận dạng.

Khai báo và Sử dụng

Cú pháp:

```
interface <tên interface>
{
    //Khai báo các thành phần interface
}
```

Lưu ý: Việc thực thi 1 Interface hoàn toàn giống kế thừa từ 1 lớp. Một lớp kế thừa từ Interface phải định nghĩa tất cả các phương thức trong Interface, nếu không lớp đó phải trở thành lớp trừu tượng và phương thức chưa định nghĩa phải được khai báo thành phương thức trừu tượng.

Khai báo và Sử dụng

Ví dụ:

```
interface I
{
    void meThod();
}
public class ThucThi : I {
    //khai báo các thành phần
    public void meThod()
    {
        Console.WriteLine("...");
    }
}
```

Ví dụ:

```
interface I {...}
```

- Một interface có thể mở rộng một interface khác.

```
interface J : I { }
```

- Một class có thể thực thi một hoặc nhiều interface, cách nhau bởi dấu ,

```
class A : I, J { }
```

- Một class có thể vừa kế thừa một class vừa thực thi nhiều interface.

```
class B : A , I, J { }
```

Bài tập

Một lập trình viên A cần xây dựng ứng dụng quản lý mua bán cà phê cho một công ty. Biết rằng thông tin của các mặt hàng cà phê khi bán ra gồm có: *mã loại cà phê, tên loại cà phê, số lượng và đơn giá*. Biết rằng công ty có 4 loại cà phê với cách tính thành tiền của các loại cà phê như sau:

- Cà phê hạt: Thành tiền = số lượng * đơn giá.
- Cà phê xay nguyên chất: Thành tiền = số lượng * đơn giá + công xay. Trong đó công xay = 100 * số lượng nếu số lượng < 100 ngược lại là 95 * số lượng.

Bài tập

- Cà phê xay có hương liệu: sẽ có thêm thông tin về số lượng hương liệu và giá hương liệu. Thành tiền = số lương * đơn giá + số lượng hương liệu* giá hương liệu + công pha chế. Cứ mỗi đơn vị số lượng thì công pha chế là 2.
- Cà phê đã đóng gói: sẽ có thêm thông tin về loại bao bì. Chỉ có 2 loại bao bì là "giấy" và "hộp nhựa". Nếu loại bao bì là "giấy" thì thành tiền = số lượng * đơn giá *110% ngược lại thì thành tiền = số lượng * đơn giá *120%.

Bài tập

Tuy nhiên trong 4 loại cà phê đang được công ty bán thì **có 2 loại cà phê hạt và cà phê xay nguyên chất** nằm trong danh mục mặt hàng được khuyến khích phát triển để hỗ trợ nông dân nên các mặt hàng này sẽ được nhà nước hỗ trợ một phần chi phí cho công ty (*KinhPhiHoTro*) như sau:

- Cà phê hạt: số lượng *10
- Cà phê xay nguyên chất: số lượng *12

Giả sử bạn là lập trình viên A, hãy phân tích thiết kế và xây dựng lớp cho chương trình ứng dụng này.

Sự giống nhau của Abstract class và Interface

- Không thể khởi tạo đối tượng từ chính nó được.
- Đều có chứa *abstract method*. Trong interface không khai báo với từ khóa *abstract*, nhưng bản chất của các phương thức trong interface là *abstract method*.
- Abstract class và interface đều được kế thừa hoặc thực thi phương thức từ các class dẫn xuất nó.
- Đều có thể thực thi từ một hoặc nhiều interface.
- Abstract class và interface đều giúp cho code trở nên sáng sủa và gọn gàng, dễ bảo trì và nâng cấp.

Sự khác nhau của Abstract class và Interface

Abstract class	Interface
– Vừa có thể kế thừa class, vừa có thể hiện thực interface.	– Không thể kế thừa class, chỉ có thể hiện thực interface.
– Có thể khai báo field, const, constructor, destructor.	– Không được khai báo field, const, constructor, destructor.
– Có thể chứa phương thức đã định nghĩa hoàn chỉnh.	– Không được chứa phương thức định nghĩa, chỉ khai báo prototype của phương thức.
– Ở lớp dẫn xuất khi kế thừa phải dùng từ khóa *override* lúc định nghĩa.	– Không dùng từ khóa *override* khi implement (vì định nghĩa mới hoàn toàn).

Sự khác nhau của Abstract class và Interface

Abstract class	Interface
– Có thể xác định modifier.	– Mọi phương thức, property đều mặc định là public.
– Không hỗ trợ đa kế thừa.	– Hỗ trợ đa kế thừa.