



# CHƯƠNG 3: HƯỚNG ĐỐI TƯỢNG TRONG C# (tt)

**Thời gian: 120 phút**

**Người trình bày: HỒ LÊ VIỆT NIN**

**Email: [holevietnin@duytan.edu.vn](mailto:holevietnin@duytan.edu.vn)**

**Điện thoại: 0905455246**



# PHẦN 2

## ĐẶT ĐIỂM HƯỚNG ĐỐI TƯỢNG TRONG C#



# Nội Dung

- ❖ Tính đóng gói (Encapsulation).
- ❖ Tính đa hình (Polymorphism)
- ❖ Tính kế thừa (Inheritance)





# ĐẶC ĐIỂM OOP TRONG C#

Để hỗ trợ những nguyên tắc cơ bản của lập trình hướng đối

Tượng (OOP), tất cả các ngôn ngữ lập trình OOP, kể cả Java đều có ba đặc điểm chung:

- Tính đóng gói (Encapsulation).
- Tính đa hình (Polymorphism)
- Tính kế thừa (Inheritance)



# ĐẶC ĐIỂM 1 TÍNH ĐÓNG GÓI

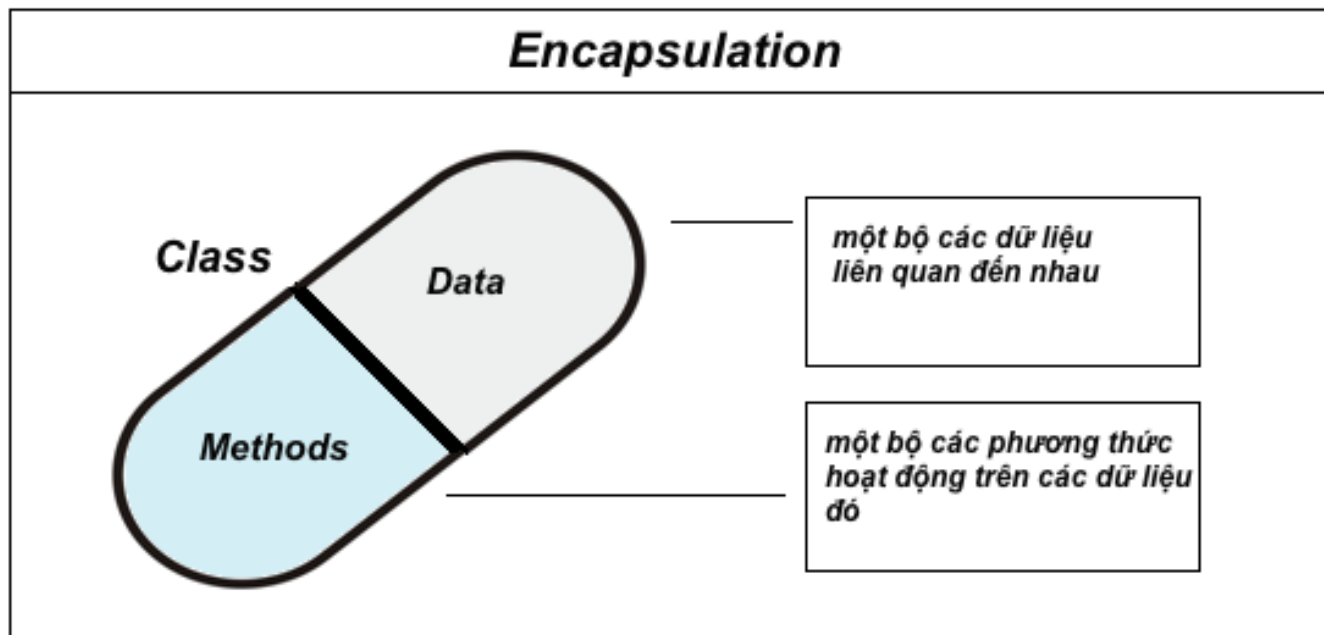


# TÍNH ĐÓNG GÓI (Encapsulation)

- Mục đích: hạn chế sự truy cập tự do vào dữ liệu, không quản lý được.
- Là một cơ chế dùng một vỏ bọc kết hợp thành phần dữ liệu và các thao tác trên dữ liệu đó thành một thể thống nhất, tạo nên sự an toàn, tránh việc sử dụng không đúng thiết kế...
- Việc truy nhập đến dữ liệu phải thông qua các phương thức của đối tượng lớp.

# TÍNH ĐÓNG GÓI (Encapsulation)

- ❖ Các dữ liệu và phương thức có liên quan với nhau được đóng gói thành các lớp để tiện cho việc quản lý và sử dụng. Điều này được thể hiện qua cách ta xây dựng 1 **class**.



# TÍNH ĐÓNG GÓI (Encapsulation)

❖ Đóng gói còn để che giấu một số thông tin và chi tiết cài đặt nội bộ để bên ngoài không thể nhìn thấy.  
Cụ thể:

- Các biến thường sẽ có phạm vi là **private**.
- Các phương thức thường sẽ có phạm vi là **public**.
- **Sinh viên hãy giải thích vì sao?**





# TÍNH ĐÓNG GÓI (Encapsulation)

- **Các biến** thường sẽ có phạm vi là **private**. Vì đây chính là các thông tin nội bộ của lớp không thể để truy cập 1 cách tùy tiện được (che giấu thông tin).
- **Các phương thức** thường sẽ có phạm vi là **public**. Vì đây chính là các hành vi (thao tác) mà lớp hỗ trợ cho chúng ta thực hiện những công việc nhất định nên cần phải cho phép mọi người có thể sử dụng được.



# ĐẶC ĐIỂM 2 TÍNH KẾ THỪA



# 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 cũng vậy: Kế thừa cho phép các đối tượng của lớp này được quyền sử dụng một số các dữ liệu cũng như các phương thức thành phần của lớp khác đã được xây dựng trước đó được gọi là lớp cha.



# KỂ THỪA

- Video tham khảo:

<https://www.youtube.com/watch?v=ldeK98pta0>



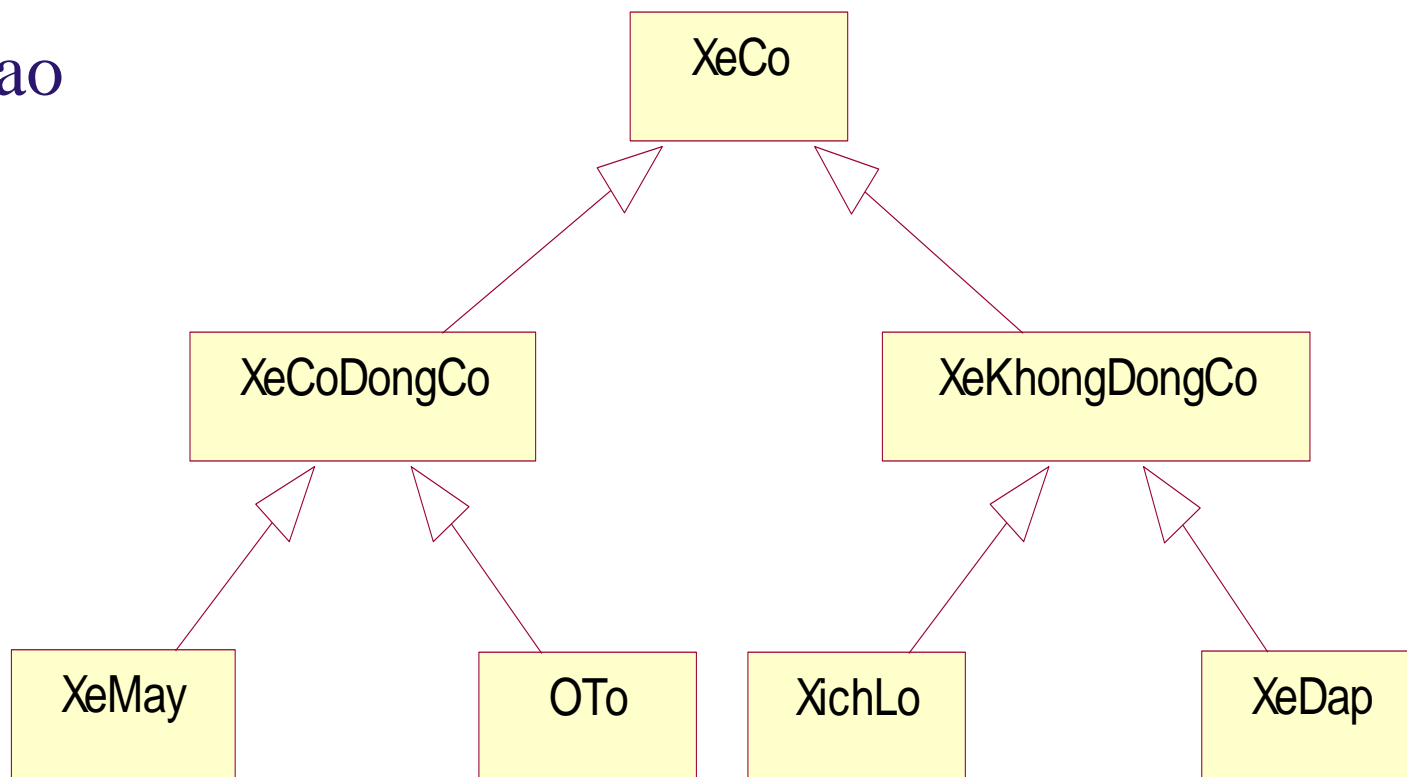
# KẾ THỪA

- Hai nguyên lý kế thừa:
  - Đơn kế thừa: một lớp con kế thừa một lớp cha
  - Đa kế thừa: một lớp con kế thừa từ nhiều lớp cha
- C# chỉ hỗ trợ đơn kế thừa từ một lớp



# KẾ THỪA

- Tạo ra cấu trúc phân cấp các lớp.
- Các lớp ở cấp thấp hơn kế thừa các thuộc tính của lớp ở cấp cao





# KÊ THỪA

## ❖ Lớp giáo viên

- ☐ Họ tên.
- ☐ Ngày sinh.
- ☐ Giới tính.
- ☐ Số CMDN

Mã lệnh  
bị trùng

☐ Mã giáo viên

- ☐ Nhập
- ☐ Xuất

Mã lệnh  
bị trùng

☐ Xếp loại thi đua

## ❖ Lớp sinh viên

- ☐ Họ tên.
- ☐ Ngày sinh.
- ☐ Giới tính.
- ☐ Số CMDN

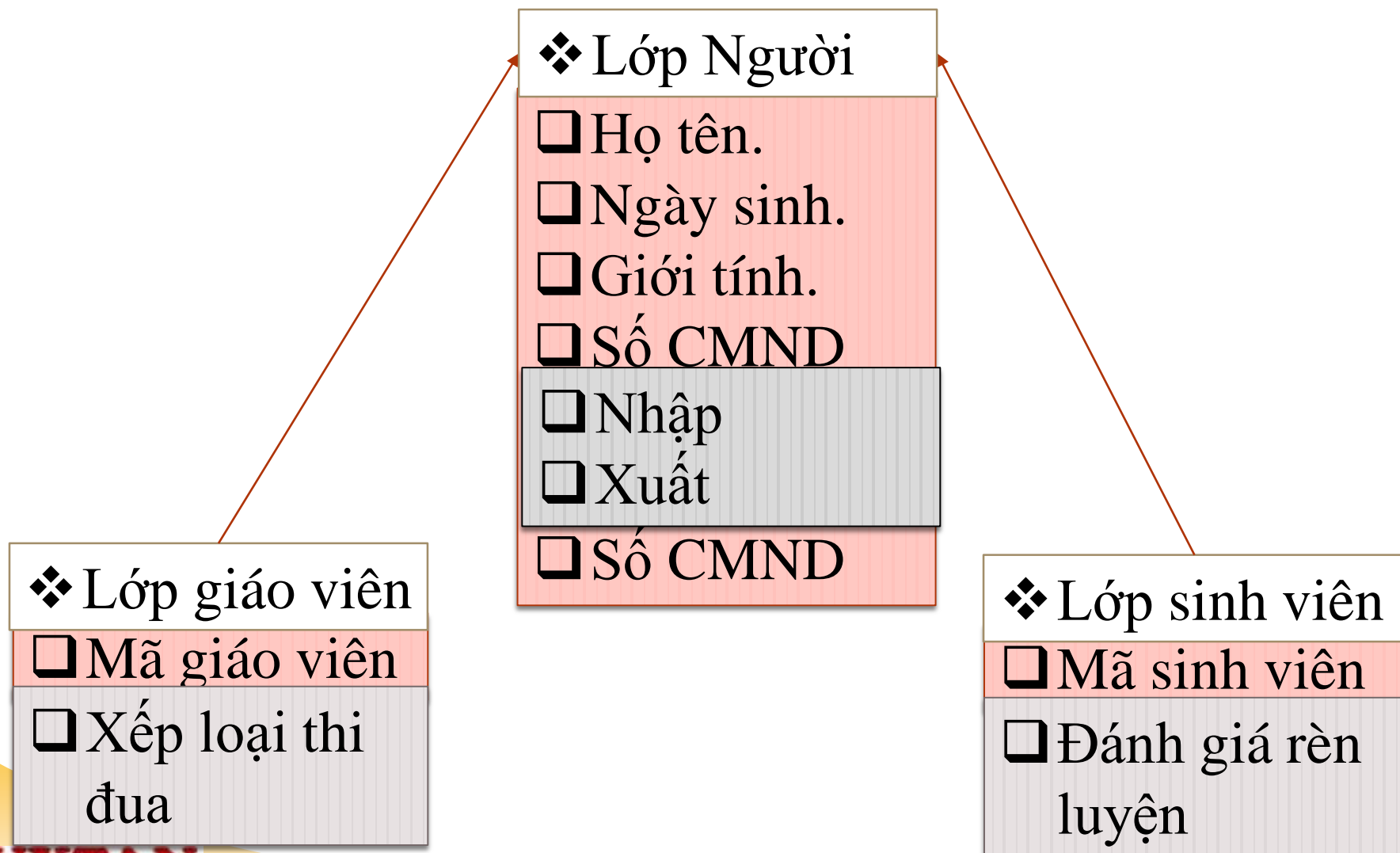
☐ Mã sinh viên

- ☐ Nhập
- ☐ Xuất

☐ Đánh giá rèn luyện



# KẾ THỪA







# KẾ THỪA

## Ưu điểm

- ❖ Cho phép xây dựng 1 lớp mới từ lớp đã có.
  - Lớp mới gọi là **lớp con (subclass)** hay **lớp dẫn xuất (derived class)**.
  - Lớp đã có gọi là **lớp cha (superclass)** hay **lớp cơ sở (base class)**.
- ❖ Cho phép chia sẻ các thông tin chung nhằm tái sử dụng và đồng thời giúp ta dễ dàng nâng cấp, dễ dàng bảo trì.



# KẾ THỪA

- **Lớp con (lớp dẫn xuất) có thể:**
  1. Thêm mới các chức năng
  2. Sử dụng các chức năng thừa kế
  3. Ghi đè các chức năng thừa kế
- **Lớp con có hầu hết các thành phần của lớp cha (lớp cơ sở) trừ:**
  1. Các hàm khởi tạo
  2. Hàm hủy

# CÚ PHÁP ĐỊNH NGHĨA KẾ THỪA

## ➤ Cú pháp:

```
class Lớp Con : Lớp Cha{  
  
    ...  
  
}
```

## ➤ Ví dụ minh họa

```
class GiaoVien: Nguoi{  
  
    ...  
  
}
```



# BA TIỀN TỔ TRONG KẾ THỪA

C# cung cấp 3 tiền tố để hỗ trợ tính kế thừa của lớp:

- **public**: lớp có thể truy cập từ các gói, chương trình khác.
- **sealed**: Lớp hằng, lớp không thể tạo dẫn xuất (không thể có con), hay đôi khi người ta gọi là lớp “vô sinh”.
- **abstract**: Lớp trừu tượng (không có khai báo các thành phần và các phương thức trong lớp trừu tượng). Lớp dẫn xuất sẽ khai báo, cài đặt cụ thể các thuộc tính, phương thức của lớp trừu tượng.



# Cú pháp

## ❖ Cú pháp định nghĩa lớp dẫn xuất(lớp con)

```
Class Tênlớpcon : Tênlớpcha  
{ //Thân lớp con  
}
```

**Hoặc**

```
Class Tênlớpdẫnxuất : Tênlớpcha  
{ //Thân lớp con  
}
```

**Hoặc**

```
Class Derived : SuperClass  
{ //Thân lớp con  
}
```



# BÀI TẬP KẾ THỪA

Xây dựng ứng dụng C# Console gồm các lớp sau:

1. Lớp Xe: (đã tạo ở bài tập trước)
2. Lớp Xe Du Lịch: Thừa kế từ lớp Xe
  - a. Bổ sung thêm trường Số chỗ ngồi (int)
  - b. Bổ sung vào phương thức Nhập() và Xuất() số chỗ ngồi
3. Lớp Xe Chở Hàng: Thừa kế từ lớp Xe
  - a. Bổ sung thêm trường Số tấn(double)
  - b. Bổ sung vào phương thức Nhập() và Xuất() số chỗ ngồi





# ĐẶC ĐIỂM 3 TÍNH ĐA HÌNH



# Đa hình là gì?

- ❖ **Polymorphism: nhiều hình thức, nhiều kiểu tồn tại.**
- ❖ **Đa hình trong lập trình.**
  - Đa hình phương thức
    - Có cùng tên, các bộ tham số khác nhau được xác định bởi: Số tham số, kiểu tham số, thứ tự tham số
  - Đa hình đối tượng.
    - Nhìn nhận đối tượng theo nhiều kiểu khác nhau. Các đối tượng khác nhau giải nghĩa thông điệp theo cách thức khác nhau



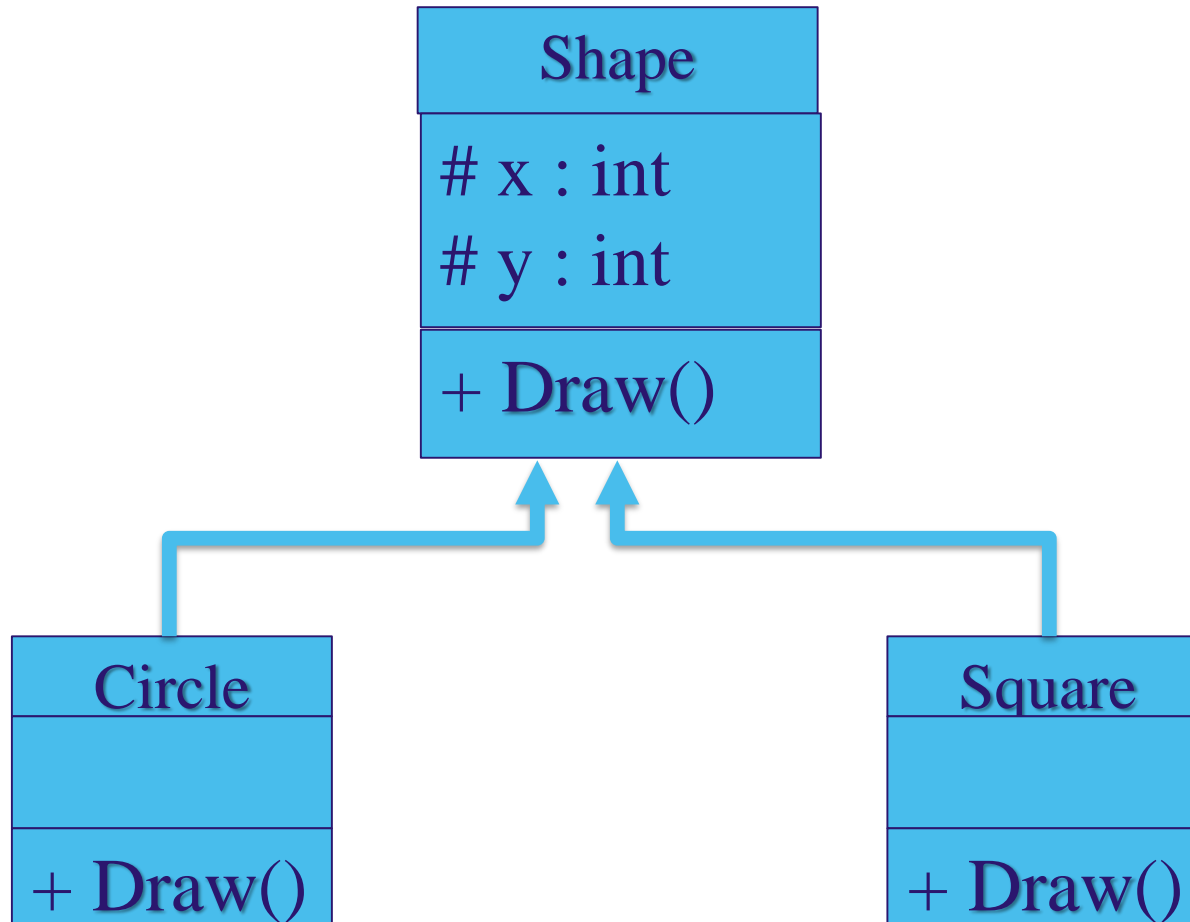


# ĐA HÌNH

- Có hai loại đa hình đó là đa hình tĩnh và động. Sinh viên có thể tìm hiểu thêm thông qua ví dụ sau:
- <https://www.youtube.com/watch?v=zrQVGL4oPro>
- <https://www.youtube.com/watch?v=AUxYWKLHSgM>



# ĐA HÌNH



# ĐA HÌNH VÀ CÁC LIÊN KẾT ĐỘNG

❖ Khả năng giải nghĩa các thông điệp theo các cách thức khác nhau

- `Shape shape = new Shape();`  
`shape.Draw();`
- `Shape circle = new Circle(100,100);`  
`circle.Draw();`
- `Shape square = new Square(30, 30);`  
`Square.Draw();`  
`Shape[] shapes = new Shape[3];`  
`Shapes[0] = new Shape(10, 10);`  
`Shapes[1] = new Circle(20, 20);`  
`Shapes[2] = new Square(30, 30);`



# PHƯƠNG THỨC Virtual



*How to Implement  
CalculatePay method?*





# PHƯƠNG THỨC Virtual

```
class Employee {  
    public void  
    CalculatePay()  
    {  
  
    Console.WriteLine("Employee.Calc  
    ulatePay");  
    }  
}
```



# PHƯƠNG THỨC Virtual

```
class SaleEmployee: Employee
{
    public void
    CalculatePay()
    {
        Console.WriteLine("SaleEmployee.
        CalculatePay");
    }
}
```



Why not polymorphism?



```
Employee e = new  
Employee();  
SaleEmployee s = new  
SaleEmployee();  
  
e.CalculatePay();  
s.CalculatePay();  
  
e = new SaleEmployee();  
e.CalculatePay();
```



# PHƯƠNG THỨC Virtual

```
class Employee {  
    public virtual void  
    CalculatePay()  
    {  
        Console.WriteLine("Employee  
        .CalculatePay");  
    }  
}
```





# PHƯƠNG THỨC Virtual

```
class Employee {  
    public virtual void  
    CalculatePay()  
    {  
    }
```

```
Console.WriteLine("Employee.CalculatePay");  
}  
  
class SaleEmployee: Employee {  
    public override void  
    CalculatePay()  
    {  
  
        Console.WriteLine("SaleEmployee.CalculatePay");  
    }  
}
```



# PHƯƠNG THỨC Virtual

```
class Employee {  
    public virtual void  
    CalculatePay()  
    {
```

```
Console.WriteLine("Employee  
CalculatePay");  
}  
  
class SaleEmployee: Employee {  
    public override void  
    CalculatePay()  
    {
```

```
        Console.WriteLine("Sale  
CalculatePay");  
    }  
}
```

```
Employee e = new  
Employee();  
SaleEmployee s = new  
SaleEmployee();  
e.CalculatePay();  
s.CalculatePay();  
e = new SaleEmployee();  
e.CalculatePay();
```

**Polymorphism!**





# Liên kết tĩnh và liên kết động

## Static and dynamic binding


- Liên kết tĩnh: lời gọi hàm (phương thức) được quyết định khi biên dịch, do đó chỉ có một phiên bản của chương trình con được thực hiện
  - ưu điểm về tốc độ
- Liên kết động: lời gọi phương thức được quyết định khi thực hiện, phiên bản của phương thức phù hợp với đối tượng được gọi
  - C# mặc định sử dụng liên kết động




# Up cast – Down cast

```
static void Main(string[] args)
{
```


```
    Shape sh = new Shape(10,10);
    sh.Draw();
```

 Drawing a **shape** at 10,10

```
    Circle ci = new Circle(20,20);
    ci.Draw();
```

 Drawing a **circle** at 20,20

```
    Square sq = new Square(30,30);
    sq.Draw();
```

 Drawing a **square** at 30,30

```
    Console.ReadLine();
}
```



# Up cast – Down cast

```
static void Main(string[] args)
```

```
{
```

```
    Shape[] shapes = new Shape[3];
```

```
    shapes[0] = new Shape(10, 10);
```

```
    shapes[1] = new Circle(20, 20);
```

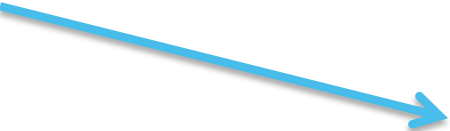
```
    shapes[2] = new Square(30, 30);
```

```
    foreach (Shape s in shapes)
```

```
        s.Draw();
```

```
    Console.ReadLine();
```

```
}
```



Drawing a **shape** at 10,10  
Drawing a **circle** at 20,20  
Drawing a **square** at 30,30



# Up cast

- Up casting là khả năng nhìn nhận đối tượng thuộc lớp dẫn xuất như là một đối tượng thuộc lớp cơ sở dùng đối tượng của lớp dẫn xuất để truyền tham số dùng đối tượng của lớp dẫn xuất làm thuộc tính

```
Shape s = new Circle(100, 100);
```

**Ép kiểu Circle sang kiểu Shape**

Tất cả các phương thức và thuộc tính của Shape tồn tại trong Circle

Up-casting: di chuyển phân cấp đối tượng sang kiểu Shape, ép đối tượng lên kiểu cha



# Down cast

```
public class Circle : Shape
{
    public Circle() { }
    public Circle(int x, int y) : base(x, y) { }
    public override void Draw()
    {
        Console.WriteLine("Drawing a circle at {0},{1}", x, y);
    }
    public void FillCircle()
    {
        Console.WriteLine("Filling circle at {0},{1}", x, y);
    }
}
```



# Down cast

```
Shape sh = new Circle(100,100);
```



```
Sh.FillCircle();
```

Vì chúng ta ép kiểu Circle sang kiểu Shape, chúng ta chỉ có thể sử dụng các phương thức có trong Shape!





# Down cast

```
Circle c;  
c = (Circle) sh;
```



c.FillCircle();



# Up cast – Down cast

- ❖ Sử dụng từ khóa “is” để kiểm tra kiểu trước khi thực hiện down cast

```
foreach (Shape shape in shapes)
{
    shape.Draw();
    if (shape is Circle)
        ((Circle)shape).FillCircle();

    if (shape is Square)
        ((Square)shape).FillSquare();
}
```



# Up cast – Down cast

## ❖ Cách khác sử dụng từ khóa “as”

```
foreach (Shape shape in shapes)
{
```

```
    shape.Draw();
```

```
    Circle c = shape as Circle;
```

```
        if(c != null)
```

```
            c.FillCircle();
```

```
    Square s = shape as Square;
```

```
        if ( s != null)
```

```
            s.FillSquare();
```

```
}
```



# BÀI TẬP ĐA HÌNH

## Tạo Lớp Quản Lý Xe với các yêu cầu sau

- Bao gồm thư viện: List (Dictionary<string, Xe>)
- Phương thức Nhập() void: với yêu cầu. Nhập vào (H) xe chở hàng, (D) xe du lịch
- Phương thức Tìm (Biển số): Xuất ra tìm thấy xe theo biển số hoặc không tìm thấy. Xuất ra thông tin xe nếu tìm thấy
- Phương thức Xóa (Biển số): Xóa Xe theo biển số trong List
- Phương thức Xuất() void



# Tài liệu tham khảo

- [1] Giáo trình lập trình Winform với C#.NET Lê Trung Hiếu, Nguyễn Thị Minh Thi
- [2] Giáo trình lập trình C#.net Phạm Hữu Khang
- [3] C# Language Reference, Anders Hejlsberg and Scott Wiltamuth, Microsoft Corp.
- [4] Professional C#, 2nd Edition, Wrox Press Ltd.
- [5] Web site [www.Codeproject.com](http://www.Codeproject.com)
- [6] Web site [www.CodeGuru.com](http://www.CodeGuru.com)