

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C#

Nội dung

1. Tổng quan về Lập trình hướng đối tượng

- Khái niệm
- Đối tượng – object
- Lớp đối tượng – class
- Các đặt trưng cơ bản

2. Xây dựng class trong C#

3. Xây dựng các thành phần trong class

Tổng quan về lập trình hướng đối tượng

❑ Khái niệm

- Lập trình hướng đối tượng (Object-Oriented Programming - OOP) là một phương pháp thiết kế và phát triển phần mềm dựa trên kiến trúc lớp (class) và đối tượng (object).

❑ Đối tượng – Object

- Là **một** thực thể trong thực tế
 - Con người: Nhân viên Trần Anh Tuấn, Sinh viên Lê Bảo Huy
 - Đồ vật: Bàn B01, Phòng học E304
 - Chứng từ: Hóa đơn HD200606-S21, Đơn đặt hàng DH200605
 - ...

Tổng quan về lập trình hướng đối tượng

□ Đối tượng – Object

- Một đối tượng phải có những thông tin cơ bản để nhận diện được đối tượng đó, ví dụ đối tượng xe ô tô có những thông tin sau:
 - Mã số xe: ...
 - Hiệu xe:
 - Màu sơn: ...
 - Hãng sản xuất:
 - Năm sản xuất:
 - ...
- Và những hành vi tương ứng với nó, ví dụ như:
 - Lái, nổ máy, dừng lại, ...
- Nhiệm vụ của người lập trình là biểu diễn một đối tượng thực tế thành một đối tượng trong chương trình



Tổng quan về lập trình hướng đối tượng

□ Lớp đối tượng – class

- 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 được dùng cho những đối tượng của lớp đó
- Kết quả của sự **TRƯỞU TƯỢNG HOÁ (Abstraction)** các đối tượng:
 - Cùng loại
 - Cùng các thông tin mô tả về đối tượng



Tổng quan về lập trình hướng đối tượng

❑ Lớp đối tượng – class

- Ví dụ: Từ những đối tượng xe ô tô cụ thể, ta có thể trừu tượng hóa thành lớp XeOto



Lớp: XeOto

Tổng quan về lập trình hướng đối tượng

□ Lớp đối tượng – class

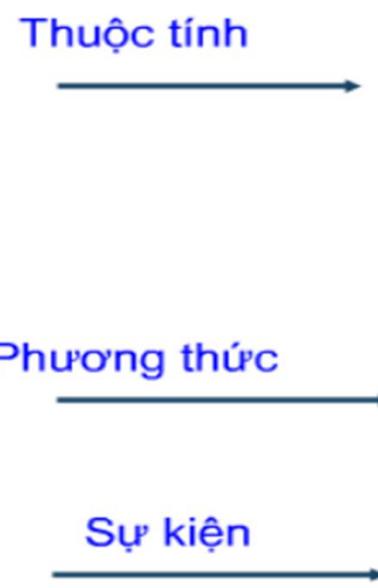
- Các thành phần của class
 - **Biến thành viên (Field)**
 - Lưu trữ các thông tin mô tả về đối tượng, ví dụ: Mã xe, số xe, ...
 - **Thuộc tính (Property) và phương thức (Method):**
 - Dùng để cập nhật, tính toán, cung cấp và xử lý thông tin của đối tượng, ví dụ:
 - **Thuộc tính** MaXe, SoXe, ...
 - **Phương thức** Lai, DungLai, NoMay, ...
 - **Sự kiện (event):**
 - Gởi thông báo của đối tượng ra bên ngoài, ví dụ: sự kiện HetXang



Tổng quan về lập trình hướng đối tượng

□ Lớp đối tượng – class

- Các thành phần của class
 - Minh họa tổng quát



XeOto
ID
HieuXe
HangSanXuat
MauSon
NamSanXuat
NoMay
VaoSo
Thang
TatMay
SapHetXang

Tổng quan về lập trình hướng đối tượng

❑ Các đặc trưng cơ bản

- Tính trừu tượng (Abstraction)
- Tính đóng gói (Encapsulation)
- Tính kế thừa (Inheritance)
- Tính đa hình (Polymorphism)

Nội dung

1. Tổng quan về Lập trình hướng đối tượng
2. Xây dựng class trong C#
 1. Khai báo class
 2. Tạo đối tượng có kiểu class
3. Xây dựng các thành phần trong class

Xây dựng class trong C#

❑ Khai báo class

```
<từ khóa khai báo phạm vi> class <tên_lớp>
{
    // khai báo các biến thành viên (Fields)
    // khai báo các thuộc tính (Properties)
    // khai báo các phương thức (Methods)
    // ...
}
```

- Từ khóa khai báo phạm vi: xác định phạm vi hoạt động của class.
Nếu bỏ qua thì class sẽ có phạm vi hoạt động là internal
- Các từ khóa khai báo phạm vi thường dùng khi khai báo class:
 - **internal**: class được sử dụng trong cùng một assembly
 - **public**: class được sử dụng trong cùng assembly và các assembly
được tham chiếu đến assembly hiện hành

Xây dựng class trong C#

❑ Khai báo class

- Cách tạo file class:
 - Tạo thư mục chứa các class
 - Click phải vào thư mục → Chọn Add → Class
 - Đặt tên cho tập tin (*.cs)

class NhanVien { //thành phần dữ liệu string maNhanVien; string hoNhanVien; string tenNhanVien; }	//các hành vi của class public void nhapNhanVien(string ma, string ho, string ten) { maNhanVien = ma; hoNhanVien = ho; tenNhanVien = ten; }
---	---

Xây dựng class trong C#

□ Tạo đối tượng có kiểu class

- Các đối tượng có thể được tạo bằng cách sử dụng từ khóa **new**, sau đó là tên của class:

<Tên_class> tên_đối_tượng = new <Tên_class>();

- Ví dụ: khai báo một đối tượng nv có kiểu class NhanVien

NhanVien nv = new NhanVien();

- Khi một instance của một class được tạo, một tham chiếu tới đối tượng sẽ được thiết lập. (Trong ví dụ trên, nv là một tham chiếu tới một đối tượng mới dựa trên class NhanVien), nhưng nếu bạn thực hiện các khai báo sau đây:

```
NhanVien nv1 = new NhanVien();
```

```
NhanVien nv2;
```

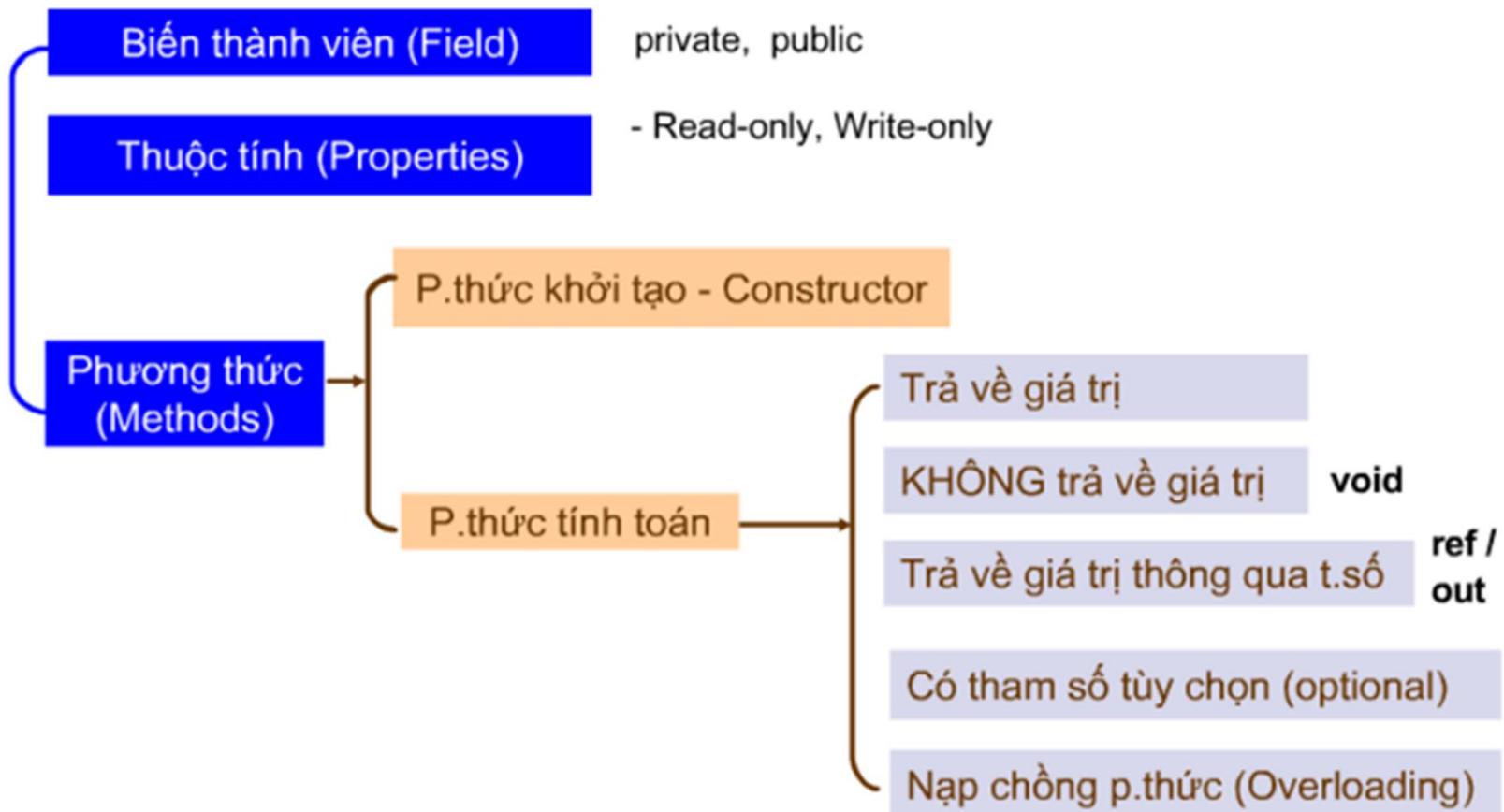
```
nv2 = nv1;
```

=> nv2 sẽ tham chiếu đến cùng đối tượng trên vùng nhớ heap giống như nv1

Nội dung

1. Tổng quan về Lập trình hướng đối tượng
2. Xây dựng class trong C#
3. Xây dựng các thành phần trong class
 1. Biến thành viên (Field)
 2. Thuộc tính (Property)
 3. Các phương thức (Method)

Xây dựng các thành phần trong class



Xây dựng các thành phần trong class

□ Biến thành viên (Field)

- Khái niệm:

- Biến thành viên là các trường (field) lưu trữ dữ liệu của một class, nó cũng chính là các thành phần dữ liệu của class.
- Các trường dữ liệu của class được khai báo bên trong khối class với việc chỉ định cấp độ truy xuất của trường dữ liệu (public/private)
- Nếu không chỉ ra phạm vi thì mặc định là **private**

- Khai báo:

```
class <Tên lớp>
{
    // Khai báo biến thành viên
    [private | public] Kieu_du_lieu Bien_thanh_vien_1;
    .....
    [private | public] Kieu_du_lieu Bien_thanh_vien_N;
    // Tiếp tục cho những khai báo khác
}
```

Xây dựng các thành phần trong class

▫ Biến thành viên (Field)

- Khai báo:

- **private**: không được phép giao tiếp ra bên ngoài
- **public**: được phép giao tiếp ra bên ngoài

- Ví dụ:

```
class NhanVien
{
    // Khai báo biến thành viên
    private string hoNhanVien;
    private string tenNhanVien;
    private DateTime ngaySinh;
    private bool gioiTinh;
    private double heSoLuong;
    // Tiếp tục cho những khai báo khác
}
```

Xây dựng các thành phần trong class

□ Thuộc tính (Property)

- Khái niệm:

- Thuộc tính là thành phần được sử dụng để truy xuất đến các biến thành viên (Field) được khai báo private bên trong class
- Mỗi thuộc tính chỉ truy xuất đến một biến thành viên duy nhất

- Khai báo chuẩn:

```
class <Tên lớp>
{
    // Khai báo thuộc tính
    [private | public] Kieu_du_lieu_X Ten_thuoc_tinh
    {
        get { return Tên_biến_thành_viên; }
        set { Tên_biến_thành_viên = value; }
    }
    // Tiếp tục cho những khai báo khác
}
```

Xây dựng các thành phần trong class

□ Thuộc tính (Property)

- Ví dụ:

```
class NhanVien
{
    // Khai báo biến thành viên
    single heSoLuong;
    // Khai báo các thuộc tính
    public double heSoLuong
    {
        get {return heSoLuong;}
        set {heSoLuong = value;}
    }
    // Tiếp tục cho những khai báo khác
}
```

Xây dựng các thành phần trong class

□ Thuộc tính (Property)

- Thuộc tính Read only (chỉ đọc)
 - Chỉ khai báo thành phần get { ... }
- Thuộc tính Write only (chỉ ghi)
 - Chỉ khai báo thành phần set { ... }
- Ví dụ:

```
class NhanVien
{
    //Thuộc tính chỉ đọc
    public double tienLuongCoBan
    {
        get {return heSoLuong * 1050000;}
    }
}
```

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

• Phương thức khởi tạo (Constructor)

- Constructor là một phương thức đặc biệt có cùng tên với tên của class chứa nó, có vai trò khởi tạo các thành viên dữ liệu của đối tượng mới
- Constructor được triệu gọi ngay sau khi khởi tạo đối tượng bằng lệnh **new**, và tương ứng với mỗi đối tượng nó chỉ được gọi một lần duy nhất. Có 2 loại: khởi tạo không tham số và có tham số
- **Phương thức khởi tạo KHÔNG tham số**
 - Được dùng cho việc khởi tạo các giá trị mặc định cho dữ liệu của class.
 - Khi khởi tạo đối tượng không phải truyền tham số lúc triệu gọi

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

• Phương thức khởi tạo (Constructor)

▪ Phương thức khởi tạo KHÔNG tham số

```
public class NhanVien
{
    string maNhanVien; string hoNhanVien; string tenNhanVien;
    //Phương thức khởi tạo không tham số
    public NhanVien()
    {
        maNhanVien = "001";
        hoNhanVien = "Phạm";
        tenNhanVien = "Thiên Thanh";
    }
}
```

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

- Phương thức khởi tạo (Constructor)

- Phương thức khởi tạo CÓ tham số

- Được phép có nhiều constructor có tham số trong 1 class

```
public class NhanVien
{
    string maNhanVien; string hoNhanVien; string tenNhanVien;
    //Phương thức khởi tạo có tham số
    public NhanVien(string ma, string ho, string ten)
    {
        maNhanVien = ma;
        hoNhanVien = ho;
        tenNhanVien = ten;
    }
}
```

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

• Phương thức khởi tạo (Constructor)

- Ví dụ: khai báo khởi tạo đối tượng nv1, nv2 và nv3

//sử dụng constructor KHÔNG tham số

NhanVien nv1 = new NhanVien();

//sử dụng constructor CÓ tham số

NhanVien nv2 = new NhanVien("002", "Phạm", "Nhật Duy");

//Sử dụng cú pháp mới, có từ C#3.0 trở lên

NhanVien nv3 = new NhanVien{ MaNhanVien="003",

 HoNhanVien="Lê", TenNhanVien="Huỳnh Hương" };

//Với MaNhanVien, HoNhanVien, TenNhanVien là thuộc tính khai báo với từ khóa public

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

• Phương thức tính toán, xử lý

▪ Cũng được gọi là hàm, là một khối mã lệnh dùng để xử lý một chức năng cụ thể. Chúng ta tạm phân ra các loại sau:

- Phương thức không trả về giá trị
- Phương thức trả về giá trị
- Phương thức trả về giá trị thông qua tham số
- Phương thức với tham số tùy chọn (Optional)
- Nạp chồng phương thức (Method Overloading)

Xây dựng các thành phần trong class

- Các phương thức (Methods)
 - Phương thức tính toán, xử lý – không trả về giá trị
 - Cú pháp:

```
public void Tên_phương_thức ([Tham số])
{
    // Khối lệnh xử lý
}
```

- Ví dụ:

```
public void xuatThongTin()
{
    string chuoi = string.Format("Mã NV: {0}\nHọ nhân viên: {1}\n Tên nhân viên: {2}",
maNhanVien, hoNhanVien, tenNhanVien);
    System.Windows.Forms.MessageBox.Show(chuoi);
}
```

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

- Phương thức tính toán, xử lý – trả về giá trị

- Cú pháp:

```
public kiểu_dữ_liệu Tên_phương_thức ([Tham số])
{
    // Khối lệnh xử lý
    return giá_trị_trả_về;
}
```

- Ví dụ:

```
public double tinhLuong()
{
    double kq = heSoLuong * 1050000;
    return kq;
}
```

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

• Phương thức tính toán, xử lý – trả về giá trị thông qua tham số

- Truyền tham số có sử dụng từ khóa **ref** hoặc **out**, cách truyền này gọi là truyền tham chiếu. Nếu không sử dụng 2 từ khóa này thì gọi là truyền tham trị
- **Truyền tham trị**: chỉ có bản sao của tham số thực được truyền 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**: mọi thay đổi của tham số hình thức trong phương thức sẽ ảnh hưởng trực tiếp tới tham số thực
- **Đặc điểm của truyền tham chiếu**: trả về giá trị thông qua tham số của phương thức:
 - Sau khi xử lý, phương thức sẽ trả về **một** hoặc **nhiều** giá trị.

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

- Phương thức tính toán, xử lý – trả về giá trị thông qua tham số

- Ví dụ:

```
public int test(int so1)
{
    so1 += 1;
}
//Khi sử dụng
int a = 5;
test (a);
Ketqua.Text = a.ToString(); // trả về 5
```

```
public void test( ref int so1)
{
    so1 += 1;
}
//Khi sử dụng
int a = 5;
test (ref a);
Ketqua.Text = a.ToString(); // trả về 6
```

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

• Phương thức tính toán, xử lý – với tham số tùy chọn (optional)

- Khi khai báo ta phải gán các giá trị mặc định, và khi sử dụng ta được phép bỏ qua

- Ví dụ:

```
public int tong(int x, int y = 5, int z = 7)
{
    int kq = x+y+z;
    return kq;
}

int kq = tong(1, 2, 3);           // tham số truyền theo thứ tự
int kq = tong(1, 2);             // bỏ qua z → tong(1, 2, 7)
int kq = tong(1);                // bỏ qua y và z → tong(1, 5, 7)

// Truyền tham số theo tên:
int kq = tong(1, z: 3);         // truyền tên z
int kq = tong(x: 1, z: 3);      // truyền tên x và z → tong(1, 5, 3)
int kq = tong(z: 3, x: 1);      // đảo ngược tham số
```

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

• Nạp chồng phương thức

- Tái định nghĩa phương thức: cho phép cài đặt các phương thức có cùng tên với nhau
- Các phương thức được tái định nghĩa phải thỏa: có cùng dạng hàm (method signature), đó là:
 - Có tên giống nhau
 - Kiểu của các tham số khác nhau
 - Số lượng của tham số khác nhau

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

• Nạp chồng phương thức

▪ Ví dụ:

```
public void xuatThongTin()
{
    string chuoit = string.Format("Mã NV: {0}\nHọ nhân viên: {1}\n Tên nhân viên: {2}",
        maNhanVien, hoNhanVien, tenNhanVien);
    System.Windows.Forms.MessageBox.Show(chuoit);
}
public string xuatThongTin(string ho, string ten)
{
    string chuoit = string.Format("Họ nhân viên: {0}\n Tên nhân viên: {1}", ho, ten);
    return chuoit;
}
public string xuatThongTin(string ma, string ho, string ten)
{
    string chuoit = string.Format("Mã NV: {0}\nHọ nhân viên: {1}\n Tên nhân viên: {2}",
        ma, ho, ten);
    return chuoit;
}
```

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

- Các từ khóa giới hạn phạm vi truy xuất của phương thức, thuộc tính:
 - **private**: chỉ được truy xuất trong nội bộ class
 - **public**: được phép giao tiếp ra bên ngoài
 - **private static**: chia sẻ dùng chung trong nội bộ class
 - **public static**: chia sẻ ra bên ngoài, dùng chung cho tất cả các đối tượng
 - Khi sử dụng phương thức này ta không cần khởi tạo đối tượng, mà chỉ cần sử dụng cú pháp:
<đối_tượng>. <tên_phương_thức>
- Lưu ý: nếu một phương thức **static** thì các biến thành viên được dùng trong phương thức đó cũng phải khai báo **static**

Xây dựng các thành phần trong class

❑ Các phương thức (Methods)

- Các từ khóa giới hạn phạm vi truy xuất của phương thức, thuộc tính:

- Ví dụ: trong lớp NhanVien, khai báo mức lương cơ bản. Mức lương này được áp dụng cho tất cả đối tượng nhân viên. Do đó, khi khai báo ta phải dùng public static

```
public class NhanVien
{
    public double heSoLuong;
    public static double mucLuongCoBan;
    public double luongCoBan()
    {
        return mucLuongCoBan * heSoLuong;
    }
}
```

//nv1 và nv2 dùng chung mức lương cơ bản
NhanVien.mucLuongCoBan = 1050000;
NhanVien nv1 = new NhanVien();
nv1.heSoLuong = 3.5;
double luong = nv1.luongCoBan();
NhanVien nv2 = new NhanVien();
nv2.heSoLuong = 3;
Luong = nv2.luongCoBan();

TÍNH THỪA KẾ VÀ ĐA HÌNH CỦA CLASS

Nội dung

1. Tính thừa kế (Inheritance)

1. Khái niệm
2. Xây dựng lớp thừa kế

2. Tính đa hình (Polymorphism) của class

1. Khái niệm
2. Từ khóa this, base
3. Ghi đè (overriding)
4. Nạp chồng hàm (Overloading)
5. Kết nối trễ (Late binding)

Tính thừa kế (Inheritance)

□ Khái niệm

- Tính thừa kế là một khái niệm nền tảng cho phép tái sử dụng mã lệnh đang tồn tại và điều này giúp tiết kiệm được thời gian trong việc lập trình
- Các class có thể thừa kế từ class khác. Class mới được gọi là **class dẫn xuất** (hay còn gọi là class con) sẽ được quyền truy xuất đến tất cả các thành viên dữ liệu và các phương thức không có sử dụng bở từ **private** của **class cơ sở** (hay còn gọi là class cha)
- Ngầm định một lớp được tạo ra sẽ kế thừa lớp **System.Object**
- Một lớp chỉ được phép kế thừa tối đa 1 lớp

- C# cho phép kế thừa phân cấp (inheritance hierarchy)

Tính thừa kế (Inheritance)

❑ Xây dựng lớp thừa kế

- Cú pháp:

Tên_class_con : Tên_class_cơ_sở

- Ví dụ: xét class cơ sở khai báo như sau:

```
class LopCha
{
    public int giatri;
    public static string mThongtin = "";
    public LopCha()
    {
        mThongtin += "Phương thức khởi tạo của lớp cha \n";
        giatri=10;
    }
    public string Xuat()
    {
        return "Phương thức xuất của lớp cha: giá trị=" + giatri;
    }
}
```

Tính thừa kế (Inheritance)

□ Xây dựng lớp thừa kế

- Ví dụ: tạo class dẫn xuất từ class cơ sở với khai báo như sau:

```
class LopCon: LopCha
{
    public LopCon()
    {
        mThongtin += "Phương thức khởi tạo của lớp con";
    }
}
//Đoạn code sử dụng
private void button1_Click(object sender, EventArgs e)
{
    LopCon lc = new LopCon();
    lblKetqua.Text = LopCha.mThongtin;
    lblKetqua2.Text = lc.Xuat();
    lc.giatri = 20;
    lblKetqua3.Text = lc.Xuat();
}
```

// Kết quả xuất ra màn hình khi thực thi ví dụ trên:

Phương thức khởi tạo của lớp cha

Phương thức khởi tạo của lớp con

Phương thức xuất của lớp cha: giá trị=10

Phương thức xuất của lớp cha: giá trị=20

Tính thừa kế (Inheritance)

□ Xây dựng lớp thừa kế

- Các từ khóa quy định phạm vi sử dụng các thành phần khai báo trong lớp cha:
 - Từ khóa **protected**: cho phép lớp con sử dụng
 - Từ khóa **private**: không cho phép lớp con sử dụng
 - Từ khóa **virtual**: cho phép lớp con ghi đè với từ khóa **override**
- Do đó:
 - Lớp dẫn xuất chỉ được phép kế thừa các thành phần được khai báo với bối từ truy xuất là public hoặc protected ở lớp cơ sở

Tính thừa kế (Inheritance)

❑ Xây dựng lớp thừa kế

- Ví dụ: bổ sung phương thức sau vào LopCha, cài đặt lại phương thức này vào lớp con

```
//Cài đặt phương thức vào LopCha
public virtual string phuongThucVirtual()
{
    return "Đây là phương thức Virtual của lớp cha";
}

//Cài đặt phương thức trên vào LopCon
public override string phuongThucVirtual()
{
    return base.phuongThucVirtual() + "\n Được lớp con sử dụng lại và ghi
đè lên";
}

//từ khóa base dùng để tham chiếu đến thành phần của lớp cha
```

Tính thừa kế (Inheritance)

□ Xây dựng lớp thừa kế

- Ví dụ: bổ sung phương thức sau vào LopCha, cài đặt lại phương thức này vào lớp con → cài đặt code sử dụng

```
LopCon lc = new LopCon();
lblKetqua.Text = lc.phuongThucVirtual();
//kết quả xuất ra màn hình:
```

Đây là phương thức Virtual của lớp cha

Được lớp con sử dụng lại và ghi đè lên

Tính thừa kế (Inheritance)

❑ Xây dựng lớp thừa kế

- Tóm tắt:

```
LopCha
{
    public phuongThuc_01
    protected phuongThuc_02
    public virtual phuongThuc_03
    private phuongThuc_04
}
```



```
LopCon: LopCha
{
    phuongThuc_01
    phuongThuc_02
    public override phuongThuc_03
    ——————phuongThuc_04—————
}
```

Được phép sử dụng phương thức 01 và 02

Được phép tái định nghĩa lại phương thức 03

Nội dung

1. Tính thừa kế (Inheritance)

1. Khái niệm
2. Xây dựng lớp thừa kế

2. Tính đa hình (Polymorphism) của class

1. Khái niệm
2. Từ khóa this, base
3. Ghi đè (overriding)
4. Nạp chồng hàm (Overloading)
5. Kết nối trễ (Late binding)

Tính đa hình (Polymorphism)

❑ Khái niệm

- Là khả năng cho phép xử lý các đối tượng khác nhau trên cùng một phương thức
 - Hay nói một cách khác, là nó có khả năng gởi cùng một thông điệp đến những đối tượng khác nhau
 - Ví dụ: thông điệp “**Tính chu vi**” được gởi đến cả hai đối tượng hình chữ nhật và hình tròn. Trong hai đối tượng này đều có chung phương thức “**Chuvi**”, tuy nhiên tùy theo thời điểm mà đối tượng nhận thông điệp, lúc đó hình tương ứng sẽ được tính chu vi
- Tính đa hình được thể hiện dưới các hình thức:
 - Ghi đè (overriding)
 - Nạp chồng (Overloading)
 - Kết nối trễ (Late binding)

Tính đa hình (Polymorphism)

❑ Sử dụng từ khóa this, base

- Từ khóa **base**:

- Được sử dụng để tham chiếu đến lớp cơ sở từ lớp dẫn xuất
- Ví dụ: Tạo lớp cha NhanVien

```
public class NhanVien
{
    string hoTen;
    int tuoi;
    public NhanVien()           // phương thức khởi tạo không tham số
    {
        hoTen="";
        tuoi=0;
    }
    public NhanVien(string pHoten, int pTuoi)   // phương thức khởi tạo có tham số
    {
        hoTen=pHoten;
        tuoi=pTuoi;
    }
}
```

Tính đa hình (Polymorphism)

❑ Sử dụng từ khóa this, base

• Từ khóa base:

- Được sử dụng để tham chiếu đến lớp cơ sở từ lớp dẫn xuất
- Ví dụ: Tạo lớp con NVVanPhong

```
public class NVVanPhong : NhanVien
{
    int soNgayVang;
    public NVVanPhong(string pHoten, int pTuo, int pNgayVang):base(pHoten, pTuo)
    {
        soNgayVang = pNgayVang;
    }
}
```

*// để tham chiếu đến phương thức khởi tạo có tham số trong lớp cơ sở NhanVien phải sử dụng từ khóa **base**. Nếu không khai báo thì xem như tham chiếu đến p.thức khởi tạo không tham số*

Tính đa hình (Polymorphism)

□ Sử dụng từ khóa this, base

- Từ khóa this:

- Được sử dụng để tham chiếu đến lớp hiện hành (lớp chứa đoạn lệnh đang cài đặt)
- Ví dụ: Sửa lại Constructor của lớp con NVVanPhong

```
public class NVVanPhong : NhanVien
{
    int soNgayVang;
    public NVVanPhong(string pHoten, int pTuo, int pNgayVang):base(pHoten, pTuo)
    {
        this.soNgayVang = pNgayVang;
    }
}
```

Tính đa hình (Polymorphism)

❑ Tính ghi đè (overriding)

- Khái niệm ghi đè được dùng để định nghĩa lại phương thức của lớp cơ sở (lớp cha) trong lớp dẫn xuất (lớp con kế thừa)
- Các điểm cần lưu ý khi thực hiện ghi đè:
 - Phương thức ở lớp cơ sở và lớp dẫn xuất phải có cùng dạng hàm (signature) và kiểu dữ liệu trả về
 - Phương thức lớp cơ sở phải được khai báo với từ khóa **virtual**
 - Phương thức lớp dẫn xuất phải được khai báo với từ khóa **override**

Tính đa hình (Polymorphism)

❑ Tính ghi đè (overriding)

- Ví dụ: Bổ sung phương thức tinhLuong() vào LopCha, sau đó tạo phương thức ghi đè trong LopCon

```
//Bổ sung trong LopCha
public virtual float tinhLuong()
{
    return heSo * 1050000;
}

//Ghi đè trong LopCon
public override float tinhLuong()
{
    return base.tinhLuong() - soNgayVang * 50000;
}
```

Tính đa hình (Polymorphism)

❑ Nạp chồng hàm (overloading)

- Còn được gọi là nạp chồng phương thức
- Cho phép khai báo các phương thức trùng tên nhau nhưng có tham số khác nhau
- Các điểm cần lưu ý khi thực hiện nạp chồng hàm:
 - Tên của các phương thức phải trùng nhau
 - Số lượng tham số phải khác nhau
 - Kiểu dữ liệu của các tham số và thứ tự các tham số phải khác nhau

Tính đa hình (Polymorphism)

❑ Kết nối trễ (Late binding)

- Tính đa hình dựa trên sự kết (Binding), đó là quá trình gắn một phương thức với một hàm thực sự
 - Trình biên dịch chưa thể xác định hàm nào tương ứng với phương thức nào sẽ được gọi mà chỉ xác định được vào lúc chương trình chạy → Điều này gọi là sự kết nối muộn (Late binding)

Tính đa hình (Polymorphism)

❑ Kết nối trễ (Late binding)

- Ví dụ: hãy quan sát bài toán đơn giản minh họa tính đa hình, giao diện màn hình như sau:

Tính lương nhân viên

Họ tên nhân viên:	Phạm Nhật Duy
Tuổi:	22
Hệ số lương:	3.5
Là nhân viên văn phòng	<input checked="" type="checkbox"/>
Số ngày vắng:	2
Số lượng sản phẩm:	

Tính lương

Tiền lương: 3,575,000

Tính lương cho nhân viên văn phòng

Tính lương nhân viên

Họ tên nhân viên:	Phạm Nhật Duy
Tuổi:	22
Hệ số lương:	3.5
Là nhân viên văn phòng	<input type="checkbox"/>
Số ngày vắng:	
Số lượng sản phẩm:	100

Tính lương

Tiền lương: 5,675,000

Tính lương cho nhân viên sản xuất

Tính đa hình (Polymorphism)

❑ Kết nối trễ (Late binding)

- Tạo các class cơ sở NhanVien:

```
public class NhanVien
{
    string hoTen;
    int tuoi;
    float heSo;
    public NhanVien() { }
    public NhanVien(string pHoten, int pTuoi, float pHeso)
    {
        hoTen=pHoten;
        tuoi=pTuoi;
        heSo = pHeso;
    }
    public virtual float tinhLuong()
    {
        return heSo * 1050000;
    }
}
```

Tính đa hình (Polymorphism)

□ Kết nối trễ (Late binding)

- Tạo các 2 class dẫn xuất NVVanPhong và NVSanXuat

```
public class NVVanPhong : NhanVien
{
    int soNgayVang;
    public NVVanPhong(string pHoten, int pTuoI,
float pHeso, int pNgayVang):base(pHoten, pTuoI,
pHeso)
    {
        this.soNgayVang = pNgayVang;
    }
    public override float tinhLuong()
    {
        return base.tinhLuong() - soNgayVang *
50000;
    }
}
```

```
public class NVSanXuat : NhanVien
{
    int soSanPham;
    public NVSanXuat(string pHoten, int pTuoI,
float pHeso, int pSoLuong): base(pHoten, pTuoI,
pHeso)
    {
        this.soSanPham = pSoLuong;
    }
    public override float tinhLuong()
    {
        return base.tinhLuong() + soSanPham *
20000;
    }
}
```

Tính đa hình (Polymorphism)

❑ Kết nối trễ (Late binding)

- Code xử lý tính tiền lương:

```
private void btnTinhLuong_Click(object sender, EventArgs e)
{
    string hoTen = txtHoten.Text;
    int tuoi = Convert.ToInt32(txtTuoi.Text);
    float heso = Convert.ToSingle(txtHeso.Text);
    float tienluong; NhanVien nv;
    if (chkLaNVP.Checked == true) {
        int ngayVang = Convert.ToInt32(txtSoNgayVang.Text);
        nv = new NVVanPhong(hoTen,tuoi,heso,ngayVang); //Late Binding
        tienluong = nv.tinhLuong(); //Công thức tính lương cho NV văn phòng
    }
    else {
        int soLuong = Convert.ToInt32(txtSoLuongSP.Text);
        nv = new NVSanXuat(hoTen,tuoi,heso,soLuong);
        tienluong = nv.tinhLuong(); //Công thức tính lương cho NV sản xuất
    }
    lblTienLuong.Text = string.Format("Tiền lương: {0:#,##0}", tienluong);
}
```

Bài tập 1

❖ Xây dựng lớp Tam giác gồm:

- ❑ Thuộc tính độ dài cạnh thứ nhất, độ dài cạnh thứ hai, độ dài cạnh thứ ba
- ❑ Phương thức nhập độ dài 3 cạnh
- ❑ Phương thức tính chu vi tam giác
- ❑ Phương thức tính diện tích tam giác
- ❑ Phương thức kiểm tra ba cạnh đúng của tam giác không
- ❑ Phương thức xác định loại tam giác

Bài tập 2

- ❖ Tạo lớp số phức bao gồm:
 - ❑ Thuộc tính: phần thực, phần ảo
 - ❑ Phương thức khởi tạo số phức
 - ❑ Phương thức in số phức
 - ❑ Phương thức in số phức
 - ❑ Phương thức nạp chõng phép trừ 2 số phức
 - ❑ Phương thức nạp chõng phép nhân 2 số phức
 - ❑ Phương thức nạp chõng phép chia 2 số phức

Bài tập 3

❖ Tạo lớp phân số bao gồm:

- Tạo lớp phân số bao gồm:
- Phương thức khởi tạo phân số
- Phương thức in phân số
- Phương thức tính ước số chung lớn nhất
- Phương thức rút gọn phân số
- Phương thức nạp chồng phép cộng 2 phân số
- Phương thức nạp chồng phép trừ 2 phân số
- Phương thức nạp chồng phép nhân 2 phân số
- Phương thức nạp chồng phép chia 2 phân số
- Phương thức nạp chồng phép so sánh hai phân số
- Phương thức nạp chồng phép chuyển kiểu phân số thành số thực