

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

Bộ môn Công nghệ Phần mềm



CHƯƠNG 3: LỚP VÀ ĐỐI TƯỢNG

23/02/2023

ĐẠI HỌC CHÍNH QUY

Nội dung

- ✓ Một số khái niệm cơ bản
- ✓ Các phương thức thông dụng
- ✓ Tham số **this**
- ✓ Nạp chồng phương thức
- ✓ Thành phần tĩnh (static)
- ✓ Generics
- ✓ Danh sách đối tượng



LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP)

- ✓ **Lập trình hướng đối tượng (Object Oriented Programming – OOP)** là một cách nhìn mới trong lập trình.
- ✓ **Lấy đối tượng** làm nền tảng để xây dựng chương trình.
- ✓ **Đối tượng** là sự gắn kết giữa dữ liệu của đối tượng và các hàm (còn gọi là phương thức) thao tác trên các dữ liệu này.

Đối tượng = Dữ liệu + Phương thức

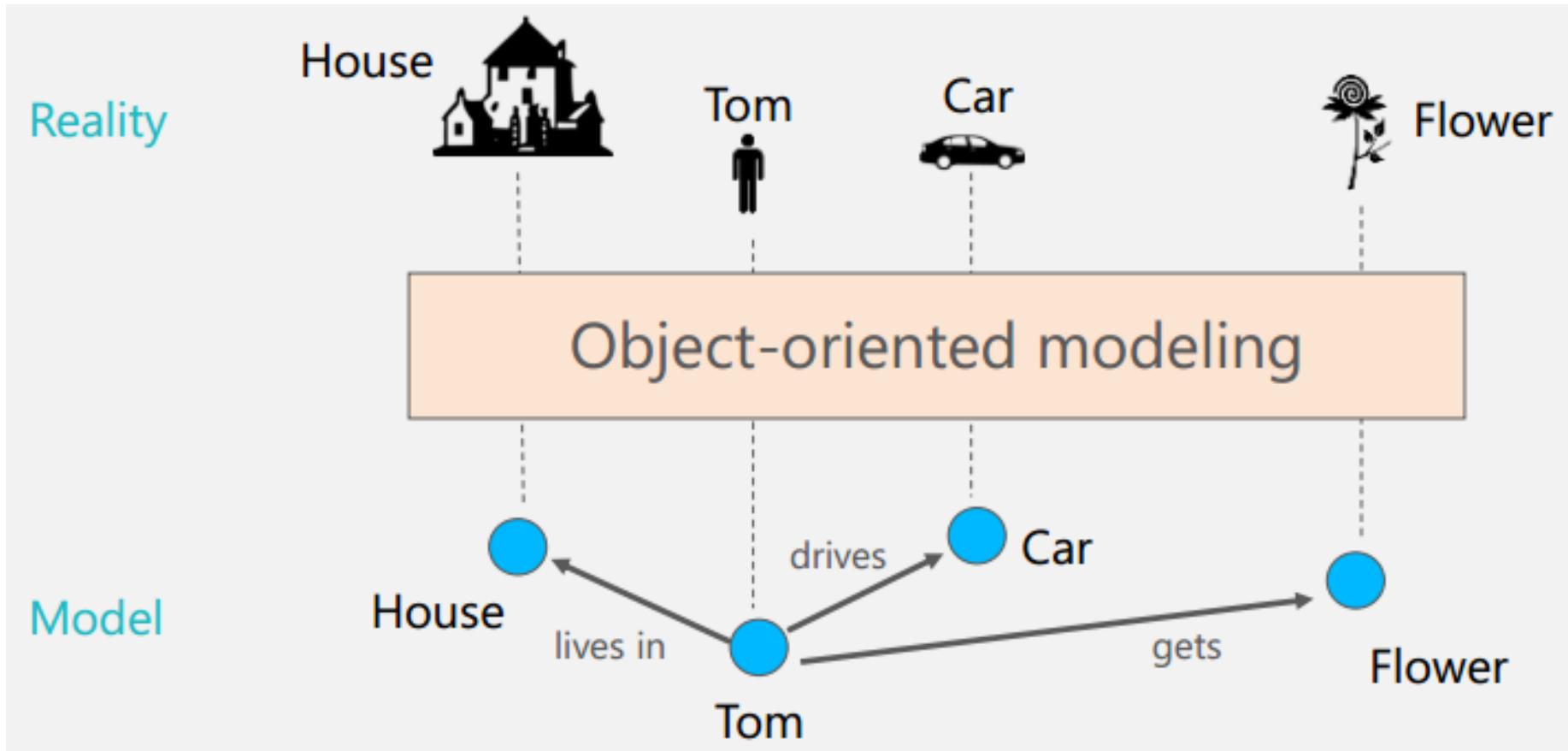
LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP)

✓ Đặc điểm

- Tính mở cao.
- Cơ chế bao đóng, che dấu tạo ra sự an toàn.
- Hỗ trợ mạnh nguyên lý sử dụng lại nhiều nhất có thể và tạo ra mọi khả năng để kế thừa.

✓ Một số ngôn ngữ lập trình hướng đối tượng: C++, **C#**, Common Lisp Object System, Eiffel, Fortran, **Java**, Objective-C, Ocaml, Object Pascal, Perl, **PHP**, **Python**, **Ruby**, Simula, Sleep, Smalltalk, ADA, Visual FoxPro.

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP)



MỘT SỐ KHÁI NIỆM CƠ BẢN

- ✓ Đối tượng (Object)
- ✓ Lớp (Class)
 - Thuộc tính (Property)
 - Phương thức (Method)
- ✓ Phạm vi truy xuất

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Đối tượng (Object)

- Là các **sự vật** trong thế giới thực: Bàn, ghế, xe, ...
- Là các **khái niệm** như: Khoa, phòng ban, ...
- Mỗi đối tượng (Ví dụ: ô tô) đều có:
 - **Các thông tin, trạng thái:**

VD: màu sắc, tốc độ, năm sx, ...

- **Các hoạt động:**

VD: Tăng ga, phanh, giảm tốc, đi đến một địa điểm, ...

MỘT SỐ KHÁI NIỆM CƠ BẢN

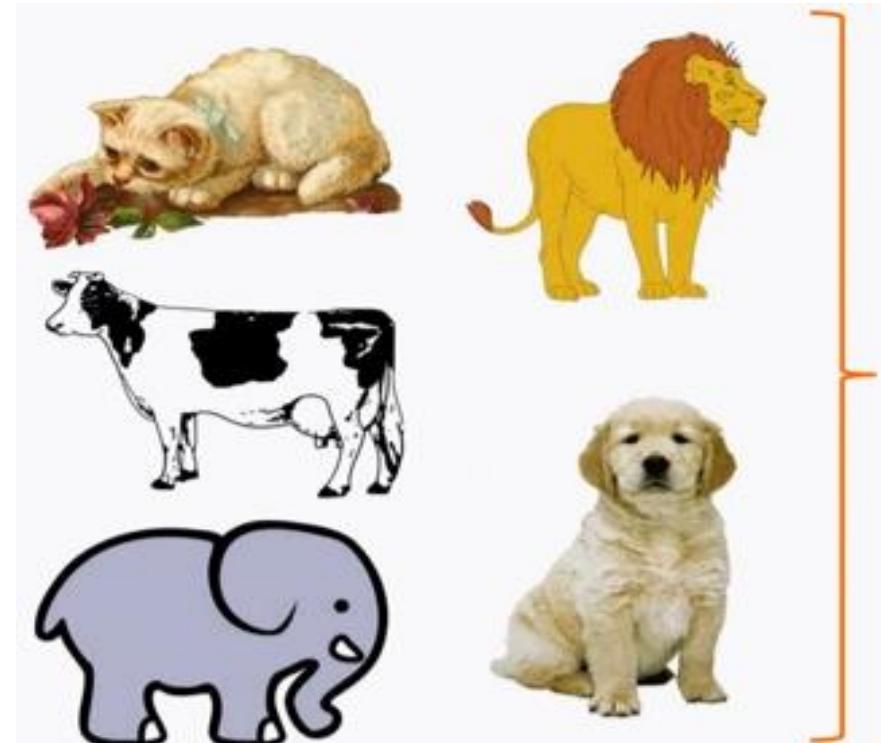
✓ Đối tượng (Object)

- Đối tượng là **duy nhất**. Không có hai đối tượng giống nhau dù cùng chia sẻ các tính chất, trạng thái.
- **Mô hình hóa vào lập trình**: Đóng gói thành trạng thái (state) và hành vi (behavior).
 - **Trạng thái** được biểu diễn bởi các thuộc tính (attributes) và các mối quan hệ (relationships).
 - **Hành vi** được biểu diễn bởi các thao tác (operations) hay phương thức (methods).

MỘT SỐ KHÁI NIỆM CƠ BẢN



Nhóm các **Xe ô-tô**



Nhóm các **Động vật**

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Lớp (Class)

- Các đối tượng có đặc tính tương tự nhau được gom chung lại thành **Lớp đối tượng**.

Ví dụ: Người là một lớp đối tượng

- Lớp định nghĩa các **thuộc tính** và **phương thức chung** cho tất cả các đối tượng của cùng một loại nào đó.
- Một đối tượng là một thể hiện cụ thể của một lớp.

Ví dụ: Mỗi đối tượng xe đạp là một thể hiện của lớp XeDap

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Lớp (Class)

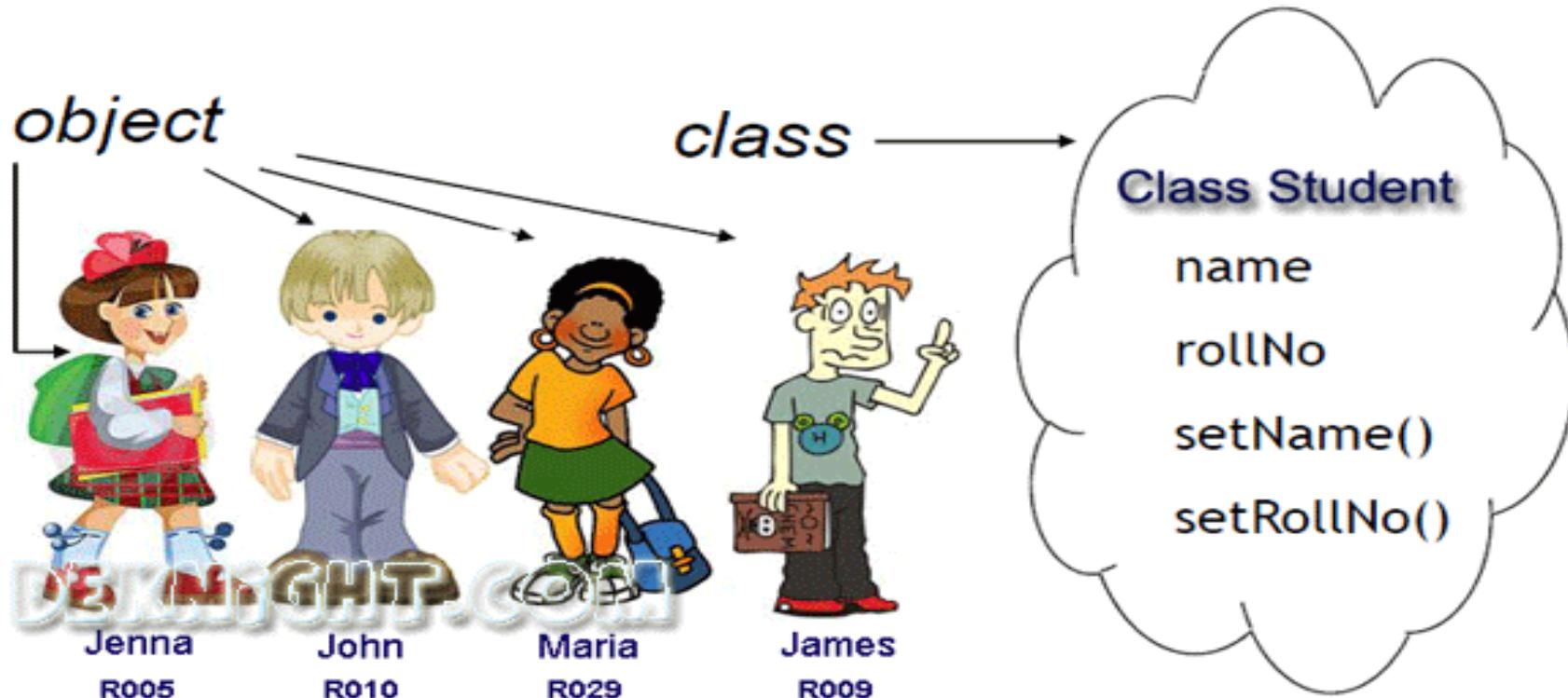
Ví dụ: Lớp SINHVIEN gồm:

- Thuộc tính: Họ tên, giới tính, ngày tháng năm sinh, điểm tb, đối tượng ưu tiên, ...
- Phương thức: Học bài, làm bài thi, làm bài tập, ...

(Sinh viên Nguyễn Văn A, Lý Thị B là **đối tượng thuộc lớp SINHVIEN**)

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Phân biệt giữa đối tượng và lớp



MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Các thành phần của lớp

- **Tên lớp (class name):** duy nhất, dùng để phân biệt lớp với lớp khác trong cùng một phạm vi.
- **Thành phần dữ liệu, thuộc tính (data members, attributes):** là các thành phần cấu tạo nên đối tượng, đặc trưng cho đối tượng.
- **Phương thức (methods/functional members):** các hoạt động của đối tượng thuộc lớp.

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Các thành phần của lớp

Ví dụ: Lớp **PhanSo** gồm các thành phần

- **Thuộc tính**
 - Tỷ số
 - Mẫu số
- **Phương thức**
 - Nhap(): nhập dữ liệu cho phân số
 - Xuat (): xuất dữ liệu cho phân số
 -



MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Sơ đồ lớp

GiaoVien
MaGV
TenGV
ChuyenMon
HocVi
Giangbai()
Chamthi()

Sơ đồ lớp

GiaoVien 1

010678

Nguyen Sinh Long

Cong nghe Thong tin

Thac si

Giangbai()

Chamthi()

Sơ đồ đối tượng
(Sơ đồ thể hiện)

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Các tính chất của lớp

- Tính trừu tượng (abstraction)
- Tính đóng gói (encapsulation)
- Tính kế thừa (inheritance)
- Tính đa hình (polymorphism)

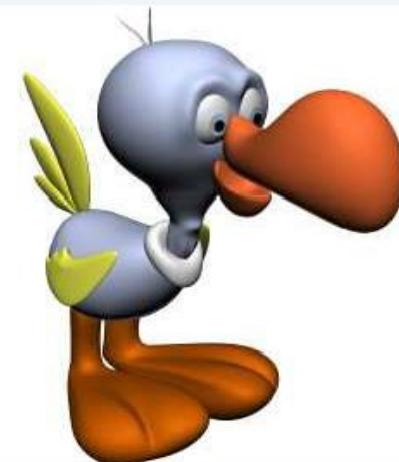
MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Tính trừu tượng

Là cách nhìn đơn giản hóa về một đối tượng mà trong đó chỉ bao gồm những đặc điểm được quan tâm và **bỏ qua** những chi tiết không cần thiết.



Trừu tượng hóa
(Abstraction)



MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Tính trừu tượng

Ví dụ: Xây dựng chương trình Quản lý Nhân sự cho một công ty.

- **Đối tượng:** NhanVien
- **Thành phần dữ liệu:** Họ tên, Tuổi, Giới tính, Chiều cao, Cân nặng, Học vấn, Mức lương, Tình trạng hôn nhân, Màu da, Màu tóc, Sở thích ẩm thực, Sở thích âm nhạc, ...
- **Thành phần hành động:** Tuyển dụng nhân viên mới, Sa thải một nhân viên, Tăng lương, Thưởng, ...

MỘT SỐ KHÁI NIỆM CƠ BẢN

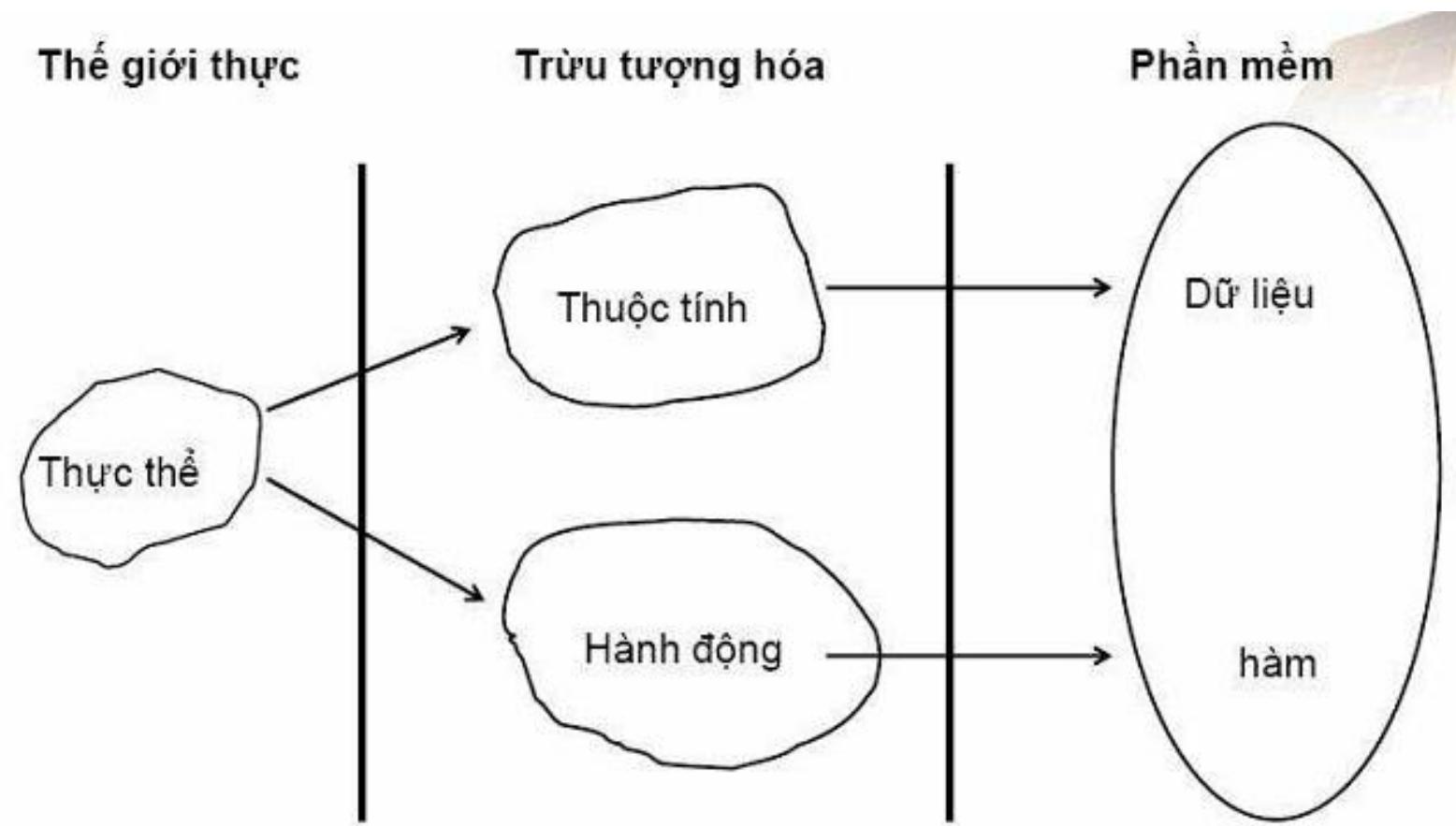
✓ Tính trừu tượng

Ví dụ: Khi lập trình biểu diễn đối tượng **nhân viên**

- **Thành phần dữ liệu:** Không thể mô tả toàn bộ → Sử dụng các thành phần dữ liệu cần thiết.
- **Thành phần hành động:** Không thể mô tả toàn bộ → Sử dụng các hoạt động liên quan tới ứng dụng.

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Tính trừu tượng



MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Tính trừu tượng

Đối tượng trong thế giới thực

- Là một thực thể cụ thể mà thông thường có thể **sờ**, **nhìn thấy** hay **cảm nhận** được.
- Đều có **dữ liệu** và **hành động** (phương thức) riêng.

	Dữ liệu	Hành động	
Con chó	Tên Màu Giống	Sủa Vẫy tai Chạy Ăn	

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ **Tính trừu tượng**

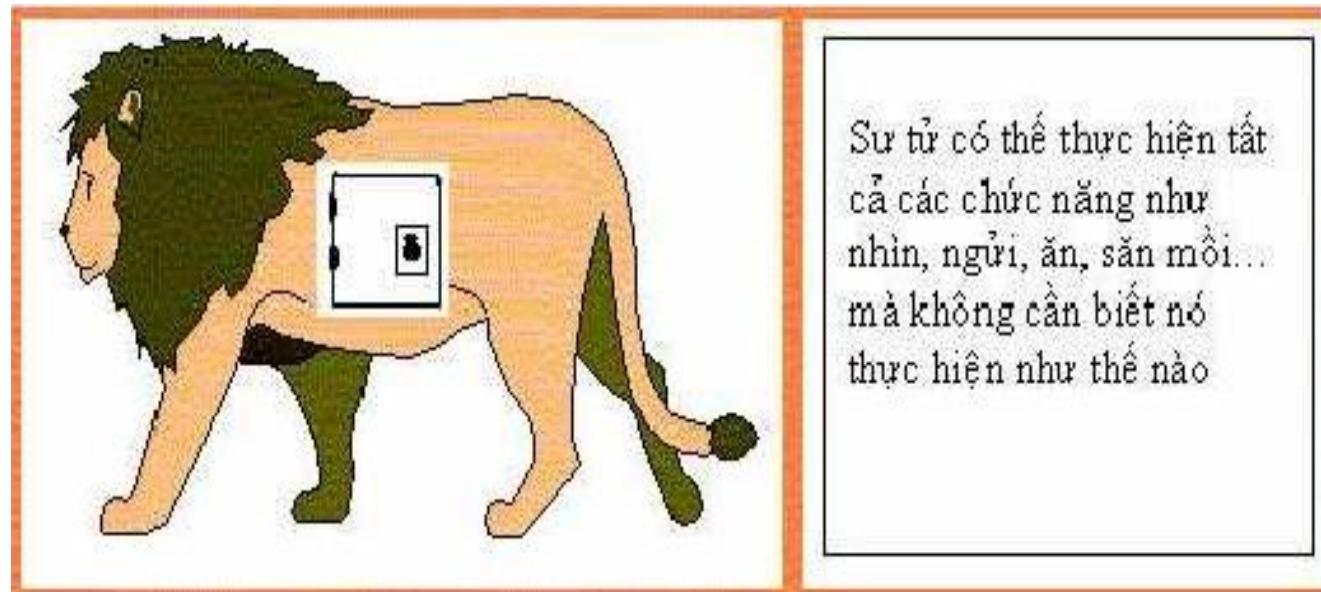
Đối tượng trong lập trình

- Dùng để mô tả, biểu diễn đối tượng trong thế giới thực.
- Có dữ liệu và hành động (phương thức, hàm) tác động trên các dữ liệu đó.
- Tính trừu tượng cho phép loại bỏ những tính chất phức tạp của đối tượng bằng cách chỉ đưa ra các thuộc tính và phương thức cần thiết của đối tượng trong lập trình.

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Tính đóng gói

- Là việc che giấu việc thực thi chi tiết của một đối tượng.
- Người dùng không phụ thuộc vào việc sửa đổi sự thực thi bên trong.



MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Tính đóng gói

- **Ẩn thông tin** → Thuộc tính được lưu trữ hay phương thức được cài đặt như thế nào được che giấu đi từ các đối tượng khác.



MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Tính kế thừa

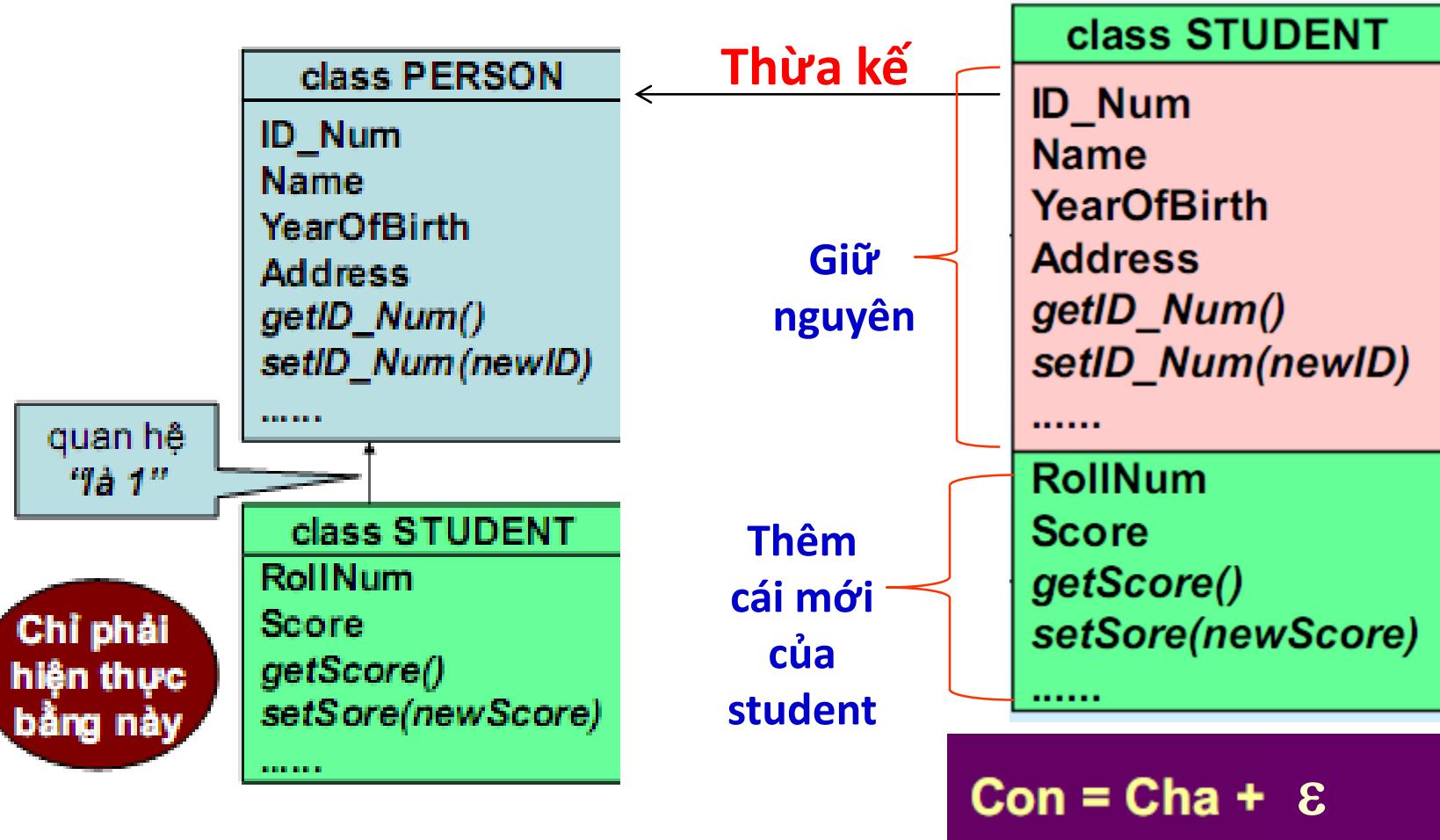
- Các lớp khác nhau nhưng có thể có chung một số đặc điểm.
- Có thể đặc tả lớp đối tượng mới từ những lớp đối tượng đã có. Khi đó, lớp mới mang đặc tính của lớp đã có (di truyền) + **đặc tính riêng của nó**.

Con = Cha + ε (riêng của con mà cha không có)

⇒ Tiết kiệm công sức viết code và test.

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Tính kế thừa



MỘT SỐ KHÁI NIỆM CƠ BẢN

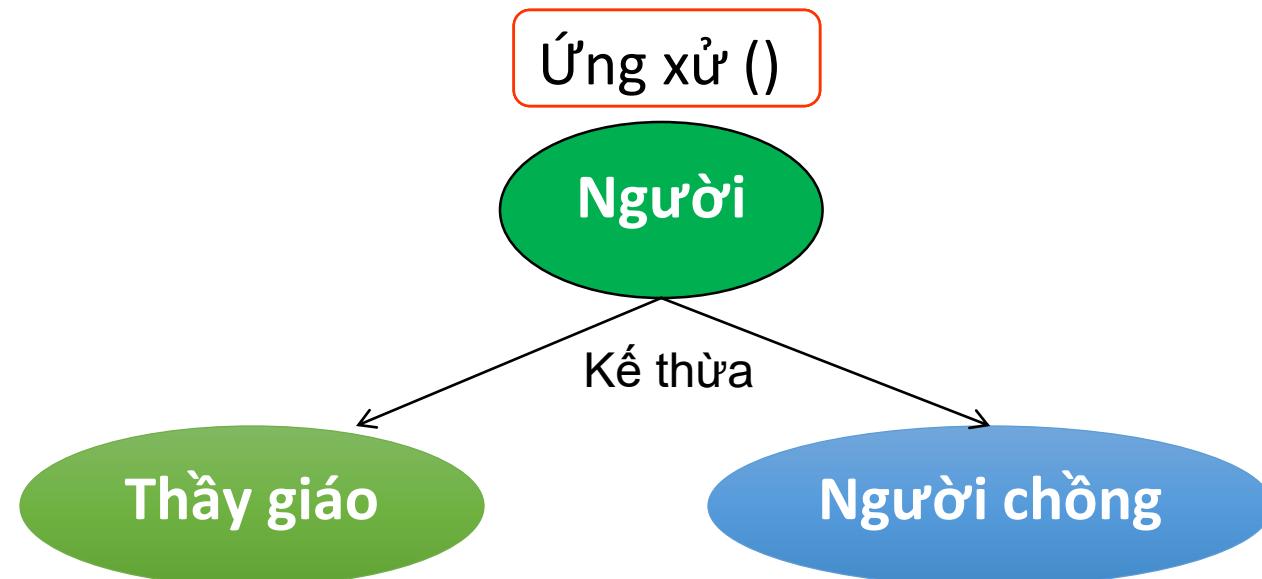
✓ Tính đa hình

- Là khả năng thực hiện khác nhau cho cùng một hành vi.
- Là kỹ thuật cho phép thay đổi nội dung của cùng một hành vi trong 2 lớp cha – con.

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Tính đa hình

- Là khả năng thực hiện khác nhau cho cùng một hành vi



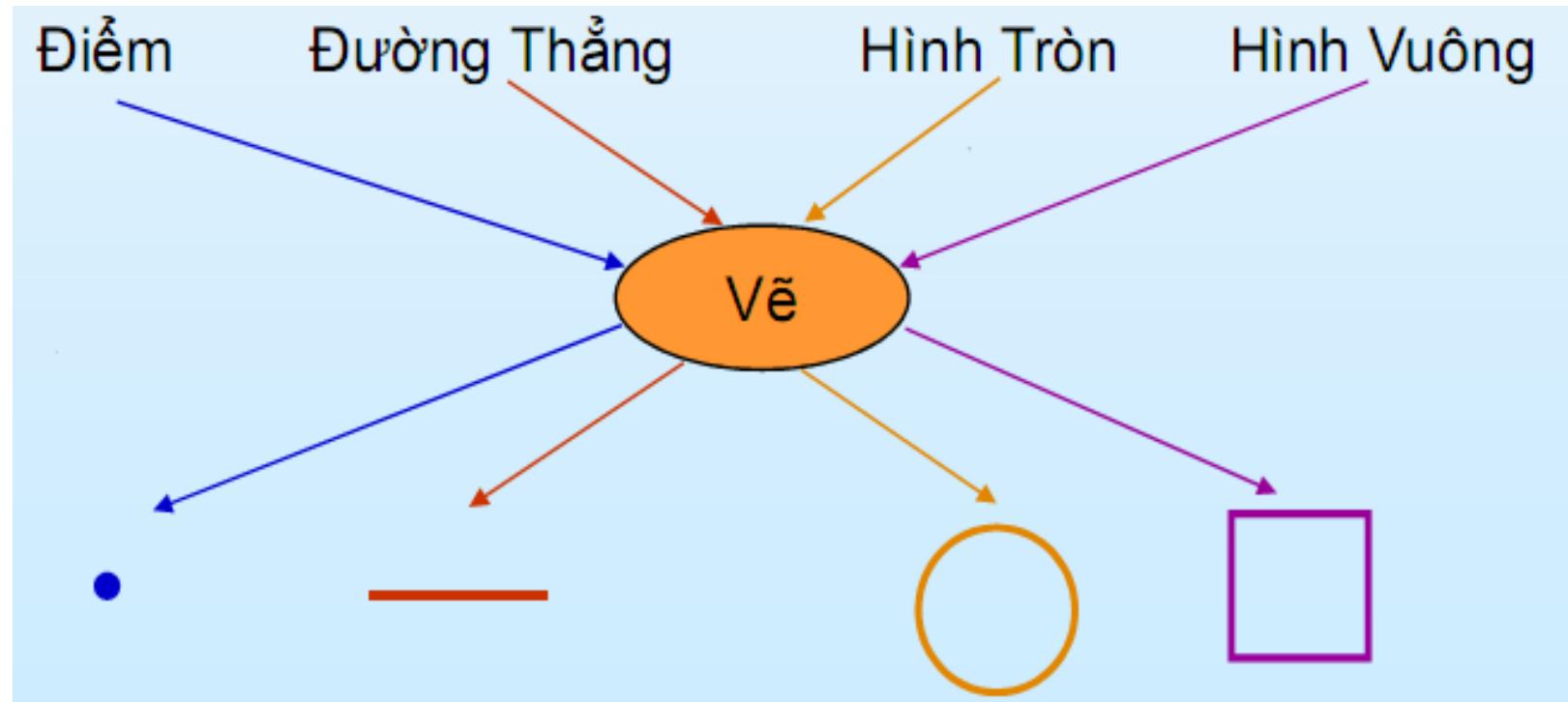
Ứng xử của một thầy giáo

Ứng xử người chồng

MỘT SỐ KHÁI NIỆM CƠ BẢN

✓ Tính đa hình

- Là khả năng thực hiện khác nhau cho cùng một hành vi





ƯU ĐIỂM CỦA OOP

1. Tính đóng gói làm giới hạn phạm vi sử dụng của các biến, nhờ đó việc quản lý giá trị của biến dễ dàng hơn, việc sử dụng mã an toàn hơn.
2. Phương pháp này làm cho tốc độ phát triển các chương trình mới nhanh hơn vì mã được tái sử dụng và cải tiến dễ dàng, uyển chuyển.
3. Phương pháp này tiến hành tiến trình phân tích, thiết kế chương trình thông qua việc xây dựng các đối tượng có sự tương hợp với các đối tượng thực tế. Điều này làm cho việc sửa đổi dễ dàng hơn khi cần thay đổi chương trình.

CÁC BƯỚC THIẾT KẾ THEO OOP

1. Xác định các dạng đối tượng (lớp) của bài toán.
2. Tìm kiếm các đặc tính chung (dữ liệu chung) trong các dạng đối tượng này, những gì chúng cùng nhau chia sẻ.
3. Xác định lớp cơ sở dựa trên các đặc tính chung của các dạng đối tượng.
4. Từ lớp cơ sở, xây dựng các lớp dẫn xuất chứa các thành phần, những đặc tính không chung còn lại của các dạng đối tượng.
5. Quan hệ giữa các đối tượng (Sử dụng hay kế thừa).

Lớp và Đối tượng

1. Khái niệm

- **Đối tượng**
 - Thuộc tính (attributes)
 - Hành vi (behavior)
 - Dấu hiệu nhận diện (identity)
- **Lớp**
 - Các thành phần
 - Các tính chất

Lớp và Đối tượng

An
8

Bình
7

Lê
7.5

Nam
6

Tú
4

Lan
5



Đối tượng

Lớp

SinhVien

TenSV

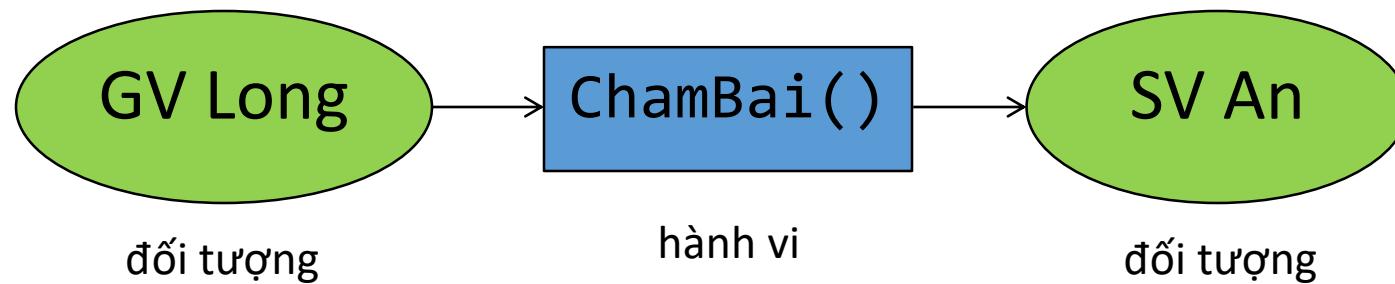
Diem

DiHoc()

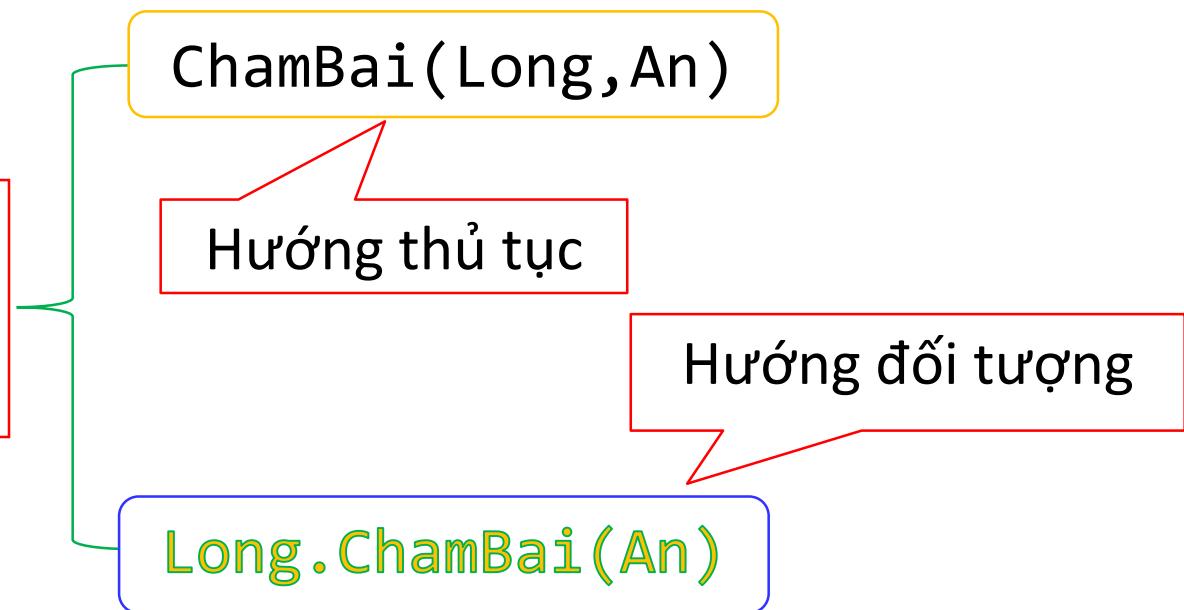
LamBai()

...

Vì sao lại sử dụng hướng đối tượng ?



Khi muốn diễn đạt:
giáo viên Long chấm
bài sinh viên An





Vì sao lại sử dụng hướng đối tượng ?

- ❖ **Diễn đạt theo hướng thủ tục:**

Tên hành vi(**Đối tượng dữ liệu**);

- ❖ **Diễn đạt theo hướng hướng đối tượng:**

Đối tượng dữ liệu.Tên hành vi();



Nhận biết đối tượng

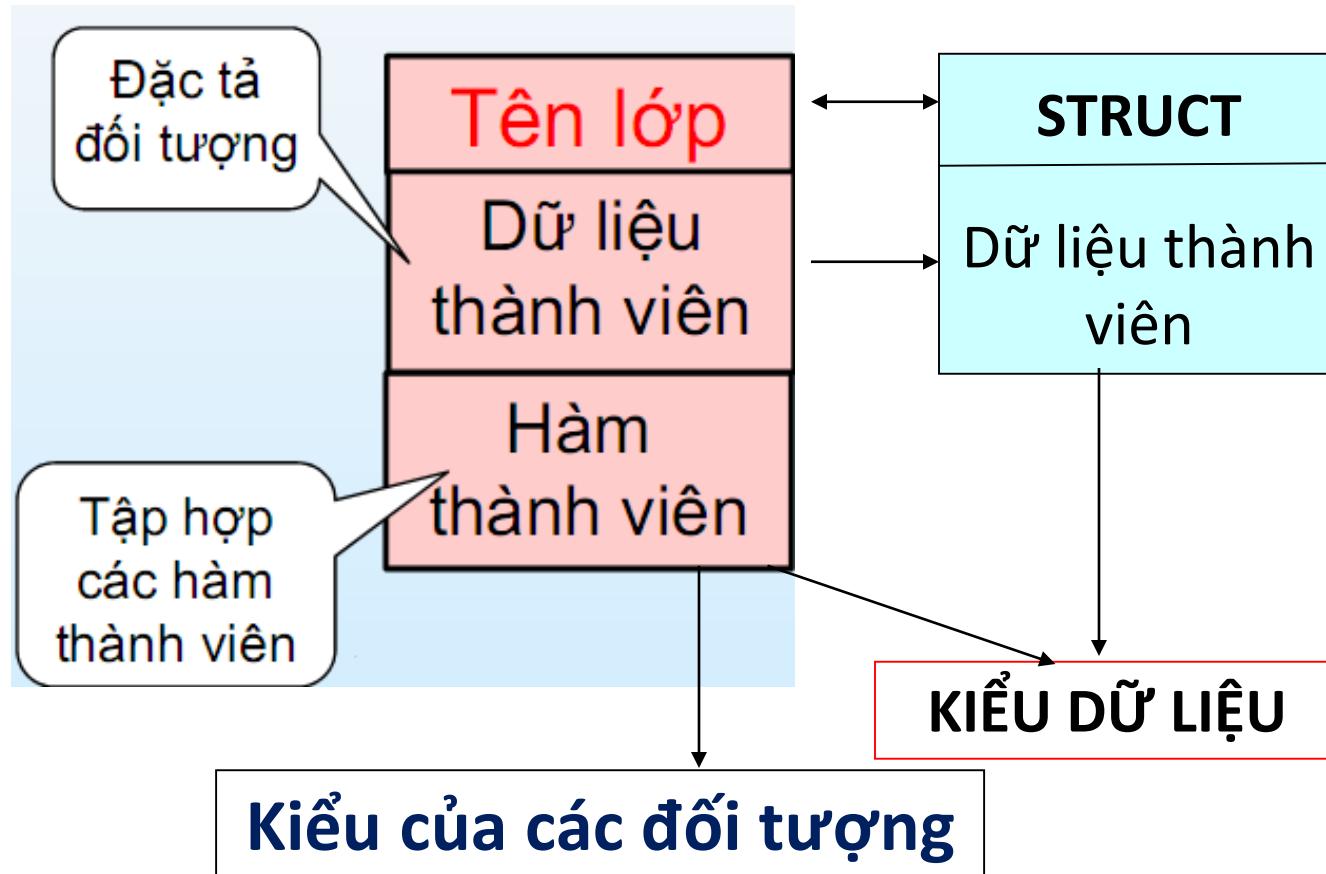
❖ **Danh từ** → *Thuộc tính mô tả đối tượng, ta cần biết thông tin gì về đối tượng*

Ví dụ: Họ tên, Mã số, Email, tử số, mầu số,...

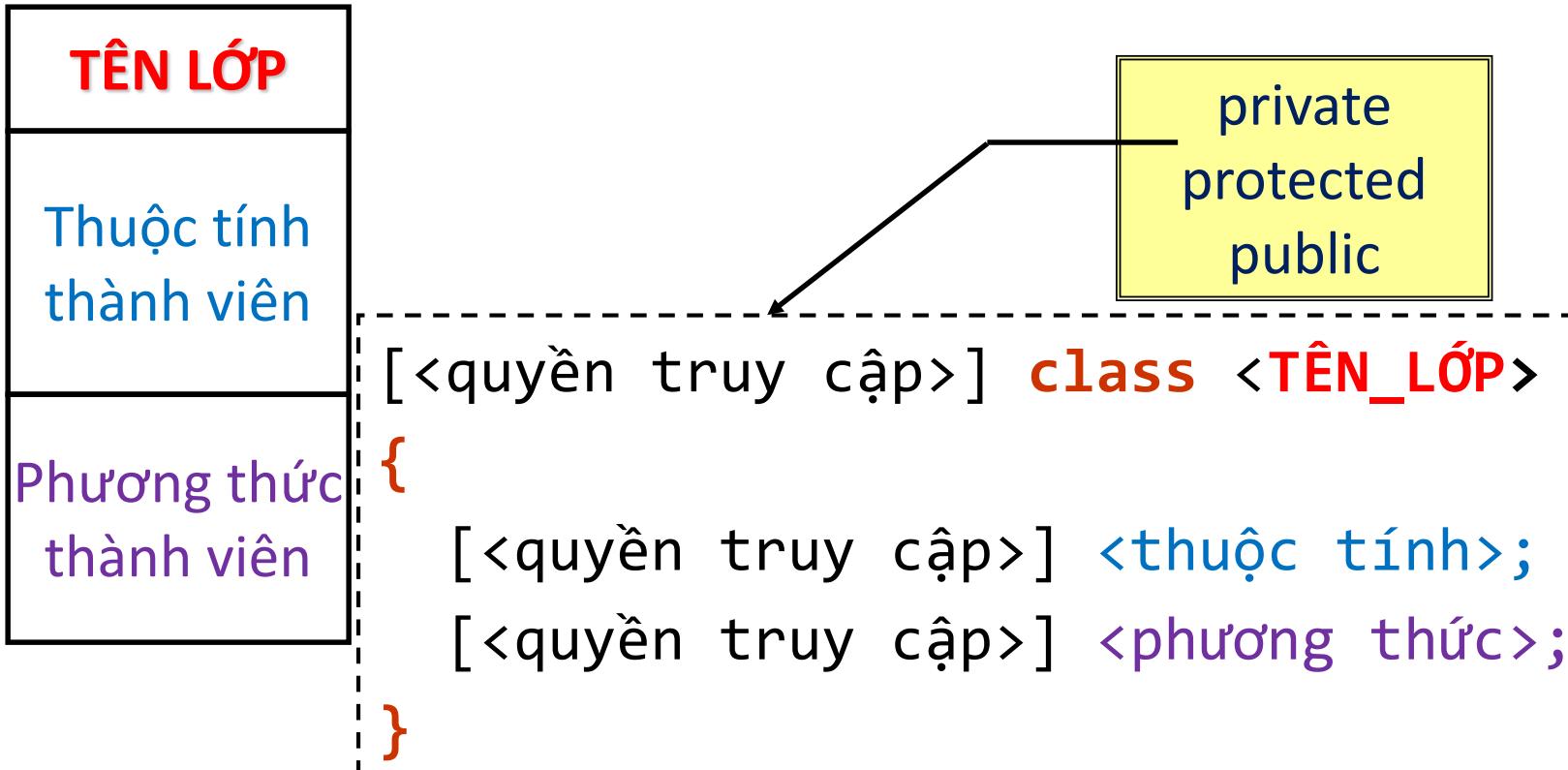
❖ **Động từ** → *Hành vi (phương thức) của đối tượng, xét xem đối tượng cần có xử lý gì*

Ví dụ: Tính điểm TB, Vẽ, cộng, trừ, nhập,...

Các thành phần của lớp



Cú pháp khai báo lớp





Định nghĩa class

```
class <<ClassName>>
```

```
{
```

```
    <<type>> <<field1>>;
```

```
...
```

```
    <<type>> <<fieldN>>;
```

Khai báo các trường

```
    <<type>> <<method1>>([parameters]) {
```

```
        // body of method
```

```
}
```

```
...
```

```
    <<type>> <<methodN>>([parameters]) {
```

```
        // body of method
```

```
}
```

Khai báo các phương thức

```
}
```

Cú pháp khai báo lớp

■ Khai báo phương thức

[<Từ khóa truy cập>] <kiểu trả về> <tên_phương
thức>([<dsthamső>])

```
{  
    // body of method  
}
```

Ví dụ:

```
float CalculateCharge(float hours)  
{  
    return (hours*billingRate);  
}
```



Cú pháp khai báo lớp

Ví dụ:

```
class PhanSo
{
    //khai báo biến lớp
    int tuSo;
    int mauSo;

    //khai báo phương thức
    ...
}
```

Phạm vi truy cập

- **public (+):** công cộng, có thể truy xuất ở bất kỳ đâu bên trong lớp và ngoài lớp.
- **protected (#):** chỉ được truy cập ở trong lớp và lớp con.
- **private (-):** chỉ được truy cập bên trong lớp

Ngoài ra còn có: internal, protected internal,..

Phạm vi truy cập

Tâm vực truy xuất:

	Trong lớp	Lớp con	Ngoài lớp
public +	+	+	+
protected #	+	+	-
private -	+	-	-

+: được phép truy xuất

- : không được phép truy xuất



Phạm vi truy cập

Lưu ý:

- Thuộc tính không được kiểu **public**
- Nếu không được đề cập, thì Access Specifier (tùy khóa truy cập) **mặc định cho một kiểu lớp là internal**
- Chế độ truy cập mặc định cho các thành viên là **private**

Các bước thiết kế lớp

- ✓ Xác định thành phần thuộc tính
- ✓ Xác định thành phần phương thức

Các bước thiết kế lớp

- ✓ **Lớp** là biểu hiện của khái niệm → là **danh từ**
- ✓ **Thuộc tính** là thành phần dữ liệu tạo nên đối tượng → là **danh từ**
- ✓ **Phương thức** là các hoạt động của lớp → là **động từ.**

Ví dụ

Ví dụ 1: Xây dựng lớp `HinhTron` với phương thức tính chu vi và diện tích khi biết bán kính.

- Attribute: Bán kính hình tròn (double)
- Method: Tính chu vi, Tính diện tích hình tròn



Ví dụ

```
class HinhTron
{
    double bankinh;
    public double Chuvi()
    {
        return bankinh * 2 * Math.PI;
    }
    public double Dientich()
    {
        return Math.Pow(bankinh, 2) * Math.PI;
    }
    public void Nhap()
    {
        Console.Write("Nhap gia tri cho ban kinh:");
        bankinh = double.Parse(Console.ReadLine());
    }
}
```



Ví dụ

Ví dụ 2: Xây dựng lớp hình chữ nhật với phương thức tính chu vi và diện tích

Ví dụ 3: Xây dựng lớp phương trình bậc 2 với phương thức đếm số nghiệm của phương trình.

Ví dụ 4: Xây dựng lớp phân số với phương thức tính tổng hai phân số.

Ví dụ 5: Xây dựng lớp Người với phương thức tính tuổi.

Ví dụ 6: Xây dựng lớp tài khoản ngân hàng với phương thức rút tiền và nạp tiền vào tài khoản.

Khai báo và sử dụng đối tượng

- **Khai báo đối tượng của lớp**

`<Tên_lớp> <tên đối tượng> = new <tên_lớp>();`

Ví dụ: `HinhTron ht = new HinhTron();`

- **Truy xuất thành phần của đối tượng**

`<tên đối tượng>.<tên thành phần>;`

Ví dụ: Truy xuất thành phần bán kính và thực thi phương thức chu vi của đối tượng ht

`ht.bankinh; ht.chuvi();`

Khai báo và sử dụng đối tượng

```
static void Main(string[] args)
{
    HinhTron ht;;
}
```

STACK

ht ?

HEAP

```
static void Main(string[] args)
{
    HinhTron ht=null;
}
```

ht null

```
static void Main(string[] args)
{
    HinhTron ht=new HinhTron();
}
```

ht @

Bankinh
Chuvi()
Dientich()



Ví dụ minh họa

```
static void Main()
{
    HinhTron ht = new HinhTron();
    ht.Nhap();
    Console.WriteLine("Chu vi hinh tron: {0:0.00}",
                      ht.Chuvi());
    Console.WriteLine("Dien tich hinh tron:
                      {0:0.00}", ht.Dientich());
    Console.ReadLine();
}
```

Bài tập

Bài tập 1: Xây dựng điểm trong không gian 2 chiều Oxy với phương thức tính khoảng cách từ điểm đó tới gốc tọa độ.

Bài tập 2: Viết hàm xây dựng lớp phân số trong toán học với các phương thức nhập, xuất, rút gọn.

Bài tập 3: Một thí sinh dự thi chứng chỉ Tin học gồm có các thông tin họ và tên, số báo danh, năm sinh, điểm thi lý thuyết, điểm thi thực hành. Thí sinh sẽ có kết quả đậu nếu tổng điểm lớn hơn 10 và không có môn nào dưới 2. Hãy xây dựng lớp Thí sinh.

Ví dụ

```
public class HinhTron
{
    double bankinh;
    public void Nhap()
    {
        Console.Write("Nhập giá trị cho bán kính:");
        bankinh = double.Parse(Console.ReadLine());
    }
    public double Chuvi()
    {
        return bankinh * 2 * Math.PI;
    }
    public double DienTich()
    {
        return Math.Pow(bankinh, 2) * Math.PI;
    }
}
```

Ví dụ

```
static void Main()
```

```
{
```

```
    HinhTron ht = new HinhTron();
```

```
    ht.bankinh = 5;
```

Lỗi

```
    Console.WriteLine("Ban kinh:{0}", ht.bankinh);
```

```
    Console.WriteLine("Chu vi: {0}", ht.Chuvi());
```

```
    Console.WriteLine("Dien tich: {0}", ht.Dientich());
```

```
}
```

Làm cách nào để có thể
đưa dữ liệu từ bên ngoài
vào cho đối tượng của lớp
HinhTron?



Phương thức get/set

- ✓ Dùng để bảo vệ việc truy cập thành phần dữ liệu của lớp.
- ✓ Cho phép người dùng kiểm tra giá trị trước khi gán cho thành phần dữ liệu của lớp.
- ✓ Giúp tạo ra các field “ảo” với “bên ngoài”, che dấu cài đặt thực sự bên trong lớp.
 - Phương thức get: được sử dụng để trả về giá trị của thuộc tính.
 - Phương thức set: được sử dụng để gán giá trị cho thuộc tính.

Cú pháp

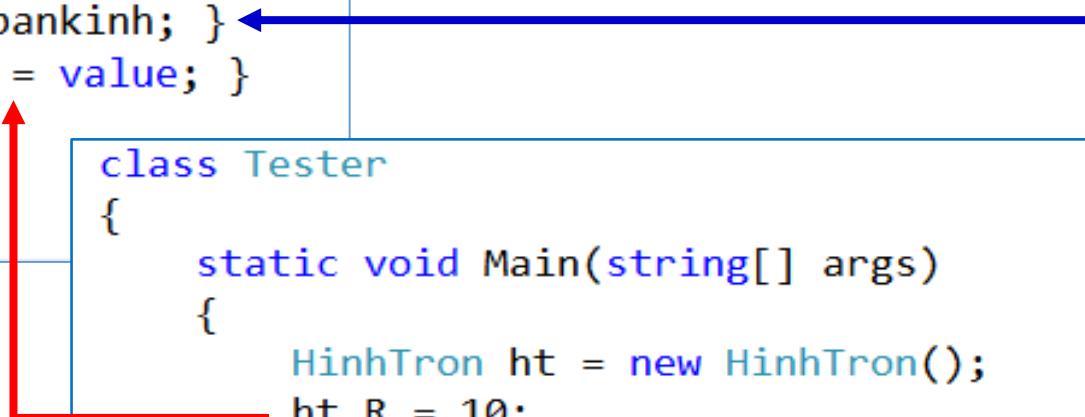
```
public <Kiểu_trả_về> <Tên_Property>
{
    get
    {
        //các câu lệnh ...
        return <biểu thức trả về>;
    }
    Set
    {
        //các câu lệnh ... xử lý giá trị value
        <tên thuộc tính> = value;
    }
}
```

Ví dụ

Ví dụ 1: Xây dựng property cho thuộc tính bankinh trong lớp HinTron

```
public class HinTron
{
    private double bankinh;
    public double R
    {
        get { return bankinh; }
        set { bankinh = value; }
    }
    ...
}
```

```
class Tester
{
    static void Main(string[] args)
    {
        HinTron ht = new HinTron();
        ht.R = 10;
        Console.WriteLine("BK là {0}", ht.R);
    }
}
```





Ví dụ

Ví dụ 2: Xây dựng property cho các thuộc tính trong lớp tài khoản ngân hàng (Mã tài khoản, Tên tài khoản, Số dư)

```
public class TKhoan
{
    string maTK;
    string tenTK;
    int soDu;
    public string MaTKhoan
    {
        get { return maTK; }
        set { maTK = value; }
    }
}
```

```
public string TenTKhoan
{
    get { return tenTK; }
    set { tenTK = value; }
}
public int SoDuTK
{
    get { return soDu; }
}
//...
```

Ví dụ

Trong cài đặt **property**, có thể cài đặt giống như một phương thức thông thường: **có khai báo biến, sử dụng các cấu trúc điều khiển, ...**

```
public String Ketqua
{
    get
    {
        if (diemtoan + diemvan > 10 && diemtoan >= 2 && diemvan >= 2)
            return "Đậu";
        else
            return "Rớt";
    }
}
```

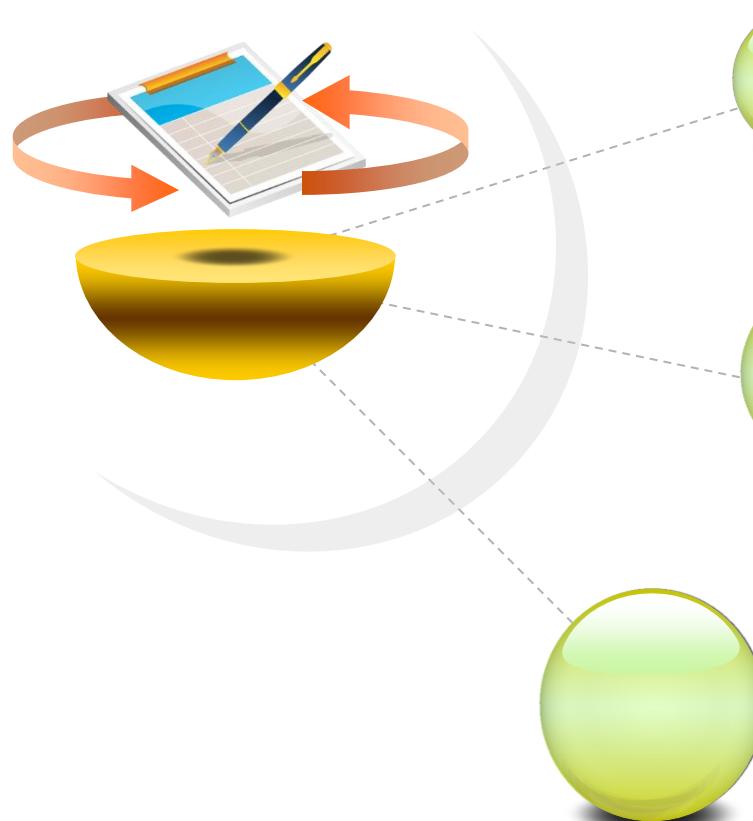
Bài tập

Bài tập 1: Xây dựng phương thức get/set cho thuộc tính bán kính của lớp hình tròn. Xây dựng phương thức trả về chu vi, diện tích hình tròn.

Bài tập 2: Xây dựng lớp hình tam giác có thuộc tính là độ dài 3 cạnh. Xây dựng phương thức trả về chu vi và diện tích của lớp tam giác.

Phương thức khởi tạo (Constructor)

- ✓ Ý nghĩa: dùng để **khởi tạo giá trị ban đầu** cho các **thành phần dữ liệu** của đối tượng.



PTKT mặc định

PTKT có tham số

PTKT sao chép

Phương thức khởi tạo

- ✓ Nếu một lớp không có phương thức khởi tạo nào, chương trình dịch (C#) sẽ tự động sinh ra một phương thức khởi tạo mặc định cho lớp. Các thành phần thuộc tính sẽ được gán các giá trị mặc định.

Kiểu dữ liệu	Giá trị mặc định
int, long, byte,...	0
bool	false
char	'\0', Null
enum	0
Reference	Null

Phương thức khởi tạo

- **Phương thức khởi tạo mặc định (default constructor):** là phương thức **không tham số đầu vào**. Các thông tin ban đầu cho đối tượng của lớp bằng những giá trị mặc định (hoặc do lập trình viên quy định).
- **Phương thức khởi tạo sao chép (copy constructor):** là phương thức **nhận tham số đầu vào là 1 đối tượng thuộc cùng 1 lớp**.
- **Phương thức khởi tạo có tham số:** là phương thức khởi tạo không thuộc 2 loại trên. Các thông tin ban đầu của đối tượng sẽ phụ thuộc vào giá trị các tham số của phương thức khởi tạo.

Phương thức khởi tạo

✓ Đặc điểm của phương thức khởi tạo

- Không có kiểu trả về
- Trùng tên với tên lớp
- Phạm vi truy xuất thường là **public**
- Một lớp có thể có nhiều phương thức khởi tạo
- Được tự động gọi khi đối tượng được tạo ra.

Phương thức khởi tạo

```
public class Diem2D
{
    private int x;
    private int y;

    public Diem2D()
    {
        x = y = 0;
    }

    public Diem2D(int hoanhdo, int tungdo)
    {
        x = hoanhdo;
        y = tungdo;
    }
}
```

Phương thức khởi
tạo không tham số
(PTKT mặc định)

```
static void Main(string[] args)
{
    //tạo điểm có tọa độ (0,0)
    Diem2D diemA=new Diem2D();
    //tạo điểm có tọa độ (3,4)
    Diem2D diemB=new Diem2D(3,4);
}
```

Phương thức khởi
tạo có tham số
(2 tham số)

Phương thức khởi tạo

Phương thức khởi tạo sao chép (Copy Constructor)

- Là một phương thức khởi tạo có tham số
- Tạo ra một đối tượng từ một đối tượng có sẵn
- Sao chép dữ liệu của đối tượng đã tồn tại (cùng kiểu)
- Mỗi lớp chỉ có 1 phương thức khởi tạo sao chép

```
public Diem2D(Diem2D d)
{
    x = d.x;
    y = d.y;
}
```

Phương thức hủy (Destructor)

- Dọn dẹp bộ nhớ cho đối tượng.
- Tự động gọi khi đối tượng bị hủy.
- **Mỗi lớp có duy nhất một hàm hủy.**
- Trong C# đã cung cấp cơ chế dọn rác (Garbage Collection) nên không cần khai báo phương thức hủy
 - ✓ Phương thức `Finalize()` sẽ được gọi bởi cơ chế dọn rác khi đối tượng bị hủy.
 - ✓ Phương thức chỉ giải phóng các vùng nhớ mà đối tượng nắm giữ, không tham chiếu tới đối tượng khác.

Phương thức hủy

Cú pháp phương thức hủy

```
~<Tên_lớp> ()  
{  
    //câu lệnh  
}
```

```
~DuongThang()  
{  
    Console.WriteLine("Đang giải phóng vùng nhớ cho p1,p2");  
}
```

Tham số this

```
public class HinhTron
{
    private double bankinh;
    public double chuvi()
    {
        return this.bankinh * 2 * Math.PI;
    }

    public double dientich()
    {
        return Math.Pow(this.bankinh, 2) * Math.PI;
    }
    public void nhap()
    {
        Console.Write("Nhập bán kính hình tròn: ");
        this.bankinh = double.Parse(Console.ReadLine());
    }
}
```



Tham số this

- ✓ Trong mỗi phương thức **không tĩnh** (non-static) của lớp đều có một tham số ngầm định – **tham số this**.
- ✓ **this** đại diện cho một class hay một đối tượng **đang làm việc trên nó**.
- ✓ Khi có một đối tượng gọi phương thức của lớp. Đối tượng gọi sẽ được truyền giá trị vào tham số this.



Tham số this

Những trường hợp sử dụng tham chiếu this:

- ✓ Tránh xung đột tên khi tham số của phương thức trùng tên với tên biến dữ liệu của đối tượng.
- ✓ Dùng để truyền đối tượng hiện tại làm tham số cho một phương thức.

Tham số this

- ✓ Tránh xung đột tên khi tham số của phương thức trùng tên với tên biến dữ liệu của đối tượng

```
public class Date
{
    int Year;
    int Month;
    int Day;
    public Date(int Day, int Month, int Year)
    {
        Console.WriteLine("Constructor co 3 tham so!");
        this.Year = Year;
        this.Month = Month;
        this.Day = Day;
    }
    //...
}
```

Tham số this

- ✓ Dùng để truyền đối tượng hiện tại làm tham số cho một phương thức

Ví dụ: Trong lớp Diem2D, xây dựng phương thức tính khoảng cách giữa 2 điểm.

```
public double KhoangCach2D(Diem2D d1, Diem2D d2)
{
    return Math.Sqrt(Math.Pow(d1.x - d2.x, 2) +
                    Math.Pow(d1.y - d2.y, 2));
}
```

Ta thấy: Cần có 2 tham số là đối tượng của lớp điểm, nhưng khi cài đặt phương thức này ta chỉ cần 1 tham số.

Tham số this

```
public double KhoangCach2D(Diem2D d)
{
    return Math.Sqrt(Math.Pow(this.x - d.x, 2) +
                    Math.Pow(this.y - d.y, 2));
}
```

Khi gọi:

```
double kcd1_d2 = d1.KhoangCach2D(d2);
```

Số lượng tham số có kiểu lớp = số lượng tham số thực – 1;

Bài tập

Bài tập 3: Xây dựng lớp NgayThang lưu trữ ngày, tháng, năm với các phương thức: Phương thức khởi tạo mặc định, phương thức khởi tạo 3 tham số, kiểm tra năm nhuận, xác định ngày hôm trước, xác định ngày hôm sau của đối tượng hiện hành.

Bài tập 4: Xây dựng lớp để mô tả một định thức cấp 2 trong toán học. Với thành phần dữ liệu là một mảng 2 chiều 2 dòng, 2 cột.

Nạp chồng phương thức (Overload method)

- ✓ Khái niệm: *là hiện tượng hai hay nhiều phương thức trùng tên cùng tồn tại trong một lớp.*
- ✓ **Đặc điểm:**
 - Trùng tên phương thức.
 - Thuộc cùng một lớp.
 - **Khác nhau:** số lượng tham số, thứ tự các tham số, kiểu dữ liệu của tham số.



Nạp chồng phương thức

Ví dụ: Khi cần **lấy tên sinh viên** trong một lớp thì có rất nhiều cách thực hiện: **lấy tên dựa vào mã số sinh viên** hoặc **lấy tên dựa vào email**.

```
class Student
{
    public string GetName(int masv) {...}
    public string GetName(string email) {...}
}
```

Nạp chồng phương thức

✓ Lưu ý:

- Không quan tâm đến kiểu dữ liệu trả về
- Không quan tâm đến tên tham số

Nạp chồng phương thức

```
int max(int a, int b)
{
    int m = a;
    if (m < b)
        m = b;
    return m;
}

double max(double a, double b)
{
    double m = a;
    if (m < b)
        m = b;
    return m;
}
```

```
static void Main()
{
    int a = 8, b = 12;
    Console.WriteLine("{0}", max(a,b));
    double x=4.5, y=2.1;
    Console.WriteLine("{0}", max(x,y));
}
```

Nạp chồng phương thức

✓ Lưu ý:

➤ Không chấp nhận hai phương thức có cùng tên, cùng danh sách tham số, chỉ khác nhau kiểu dữ liệu trả về.

Ví dụ:

```
public int cong(int a, int b) {}
```

```
public float cong(int a, int b) {}
```



Nạp chồng phương thức

Bài tập 1: Khai báo lớp PhanSo chứa các thông tin tử số và mẫu số ($mẫu số \neq 0$) cùng các phương thức theo yêu cầu sau:

- a. Phương thức khởi tạo;
- b. Phương thức cộng: 2 phân số, phân số với một số nguyên (viết dạng nạp chồng phương thức).



Đặc điểm của thuộc tính, phương thức

- Chỉ có thể sử dụng sau khi khởi tạo đối tượng.
- Dữ liệu thuộc về riêng mỗi đối tượng (xét cùng 1 thuộc tính thì các đối tượng khác nhau thì thuộc tính đó sẽ mang các giá trị khác nhau).
- Được gọi thông qua tên của đối tượng.

Người lập trình mong muốn **1** thuộc tính nào đó **được dùng chung** cho mọi đối tượng (chỉ được cấp phát 1 vùng nhớ duy nhất)???



Thành phần tĩnh (static)

- Được khởi tạo **một lần duy nhất** ngay khi biên dịch chương trình.
- **Dùng chung** cho mọi đối tượng.
- **Được gọi thông qua tên lớp.**
- **Được huỷ** khi kết thúc chương trình.



Thành phần tĩnh

Có 4 loại thành phần tĩnh chính:

- Biến tĩnh (static variable).
- Phương thức tĩnh (static method).
- Lớp tĩnh (static class).
- Phương thức khởi tạo tĩnh (static constructor).

Biến tĩnh (static)

- Biến tĩnh được sử dụng để tham chiếu thuộc tính chung của tất cả các đối tượng (không phải là duy nhất cho mỗi đối tượng)
- Biến tĩnh chỉ sử dụng bộ nhớ một lần vào thời điểm tải lớp.
- Lợi ích của biến tĩnh là nó **làm cho bộ nhớ chương trình của bạn hiệu quả hơn.**

Biến tĩnh

- Cú pháp:

[quyền truy cập] static <Kiểu dữ liệu> <Tên biến> [= giá trị];

- Truy cập:

Tên_lớp.Tên_biến_static



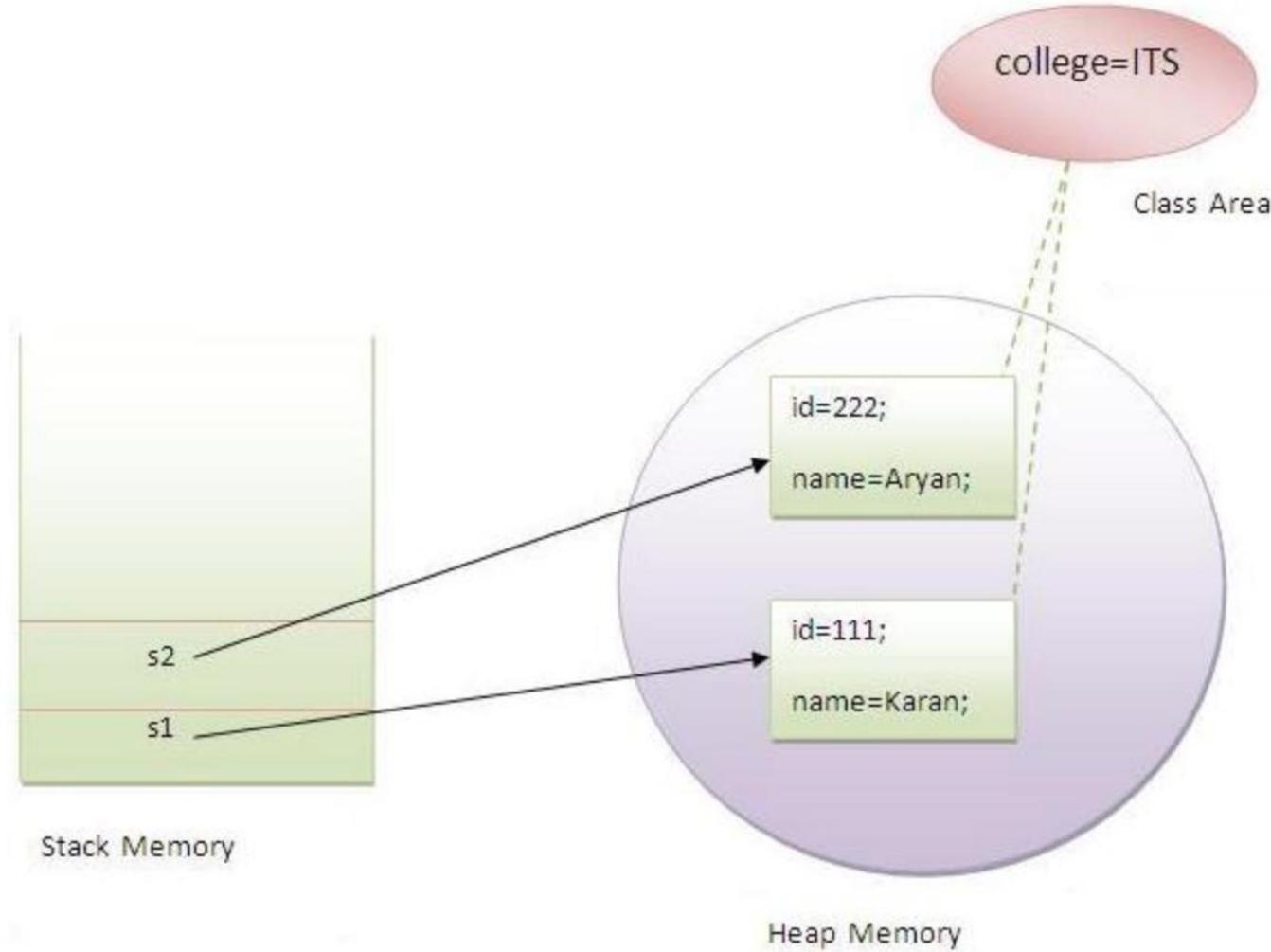
Biến tĩnh

Ví dụ:

```
class Student
{
    int rollno;
    String name;
    static String college = "ITS";
}
```

Biến tĩnh

Ví dụ:



Phương thức tĩnh (static)

- ✓ *C# cung cấp cho bạn một cách định nghĩa method để khi cần có thể gọi sử dụng những method đó một cách thuận tiện mà không cần phải thông qua đối tượng; đó chính là **Static Method**.*
- ✓ **Không sử dụng** tới các thành phần dữ liệu của lớp.

Phương thức tĩnh

- ✓ Không có con trỏ ngầm định this.
- ✓ Chỉ được phép truy xuất vào **thành phần static** của lớp.

Cú pháp:

[AS] **static** <Kiểu trả về> <Tên_PT>([<DSTS>])

{
}

Cú pháp truy xuất:

<Tên_lớp>.<Tên_PT>([<DSTS>]);



Phương thức tĩnh

Ví dụ: Viết một hàm tiện ích đó là tính lũy thừa của 1 số nguyên để hỗ trợ tính toán.

```
class TienIch
{
    public static long LuyThua(int coso, int somu)
    {
        long ketqua = 1;
        for(int i = 0; i<somu; i++)
            ketqua *= coso;
        return ketqua;
    }
}
```



Phương thức tĩnh

Ví dụ: Sử dụng trong các class khác

```
class hinhTron {  
    float banKinh;  
    public hinhTron(float bk) {  
        banKinh = gt;  
    }  
    public float dienTich() {  
        return TienIch.LuyThua(banKinh, 2) * Math.PI;  
    }  
}
```



Phương thức tĩnh

Ví dụ: Sử dụng trong các class khác

```
class hinhTron {  
    float banKinh;  
    public hinhTron(float bk) {  
        banKinh = gt;  
    }  
    public float dienTich() {  
        return TienIch.LuyThua(banKinh, 2) * Math.PI;  
    }  
}
```

Phương thức tĩnh

Hạn chế:

- Phương thức tĩnh **không thể** sử dụng trực tiếp các biến và phương thức không được thiết lập là **static**.

```
class A {  
    int a = 40; //non-static  
    public static void main String args[]){  
        Console.WriteLine(a);  
    }  
}
```

Phương thức khởi tạo tĩnh (static)

- ✓ Phương thức này dùng để **khởi tạo giá trị cho biến tĩnh** của lớp và **sẽ thực thi trước** khi đối tượng đầu tiên của lớp được tạo.
- ✓ **Chỉ được gọi một lần duy nhất.** Không có tham số, không có phạm vi truy xuất.

Cú pháp:

```
static <Ten_lop>(){  
}
```



Phương thức khởi tạo tĩnh

```
class Student {  
    int rollno;  
    String name;  
    static String college;  
  
    static Student() { college = "BBDIT"; }  
  
    Student (int r, String n){  
        rollno = r; name = n;  
    }  
    void display () {  
        Console.WriteLine(rolln + " " + name + " " college);  
    }  
    public static void main(String args[]){  
        Student.change();  
        Student s1 = new Student(111 "Karan");  
        Student s2 = new Student(222 "Aryan");  
        s1.display();  
        s2.display();  
    }  
}
```

Output:111 Karan BBDIT 222 Aryan BBDIT



Phương thức khởi tạo tĩnh

Ví dụ:

```
private static int _GiaGioThuong;
private static int _GiaGioCaoDiem;
static PhongKaraoke()
{
    _GiaGioCaoDiem = 80;
    _GiaGioThuong = 50;
}
```

Lớp tĩnh (static)

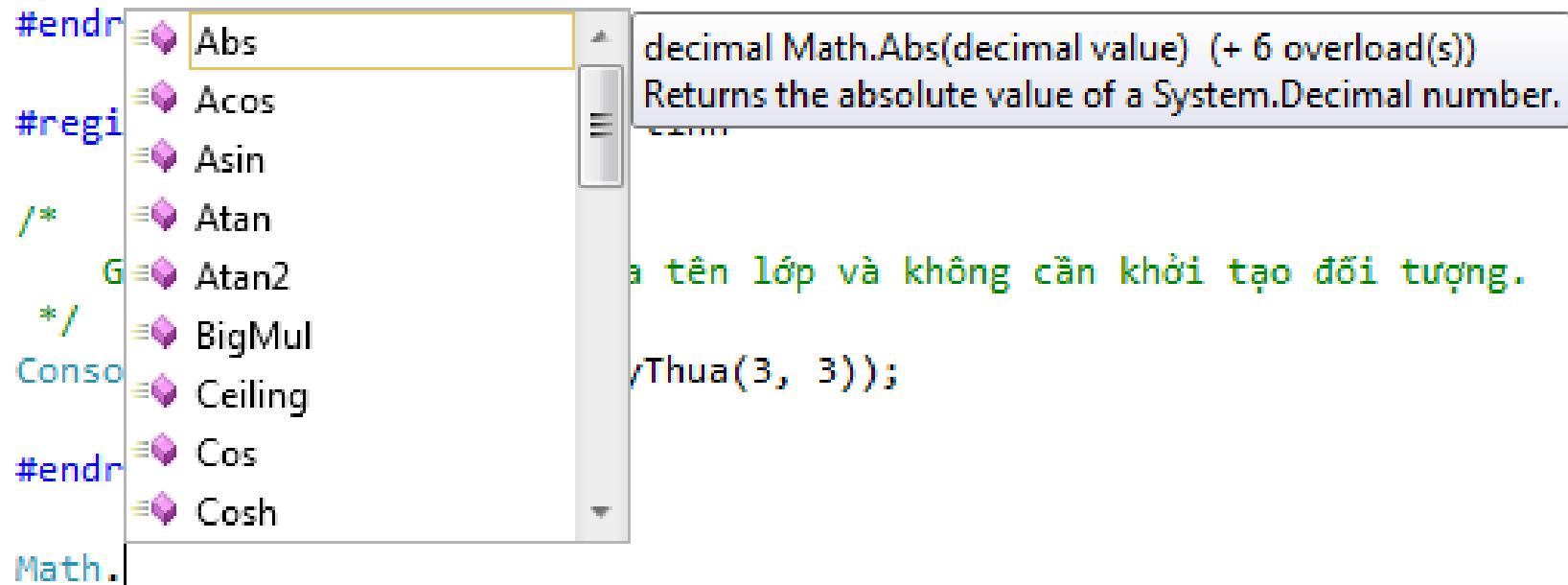
- Chỉ chứa các thành phần tĩnh (biến tĩnh, phương thức tĩnh).
- **Không** thể khai báo, khởi tạo 1 đối tượng thuộc lớp tĩnh.

Với 2 đặc điểm trên có thể thấy lớp tĩnh thường được dùng với mục đích khai báo 1 lớp tiện ích chứa các hàm tiện ích hoặc hằng số vì:

- Ràng buộc các thành phần bên trong lớp phải là **static**.
- Không cho phép tạo ra các đối tượng dư thừa làm lãng phí bộ nhớ.
- Mọi thứ đều được truy cập thông qua tên lớp.

Lớp tĩnh

- Trong C# có rất nhiều lớp tiện ích sử dụng lớp tĩnh, phương thức tĩnh để khai báo. Ví dụ điển hình đó là lớp **Math**.



The screenshot shows a portion of a C# code editor. A tooltip is displayed over the word "Math.", listing several static methods from the `Math` class:

- Abs
- Acos
- Asin
- Atan
- Atan2
- BigMul
- Ceiling
- Cos
- Cosh

The `Abs` method is highlighted with a yellow selection bar. The tooltip provides the following information:
`decimal Math.Abs(decimal value) (+ 6 overload(s))`
Returns the absolute value of a `System.Decimal` number.

Below the tooltip, the code editor shows a snippet of C# code:
`#endr`
`#regi`
`/*`
 `G`
 `*/`
`Conso`
`#endr`
`Math.`
 `Thua(3, 3));`

A green annotation on the right side of the code editor states: "Không cần khai báo tên lớp và không cần khởi tạo đối tượng."



Bài tập

Bài tập 2: Viết chương trình xây dựng lớp nhân viên với các thông tin như mã nhân viên, họ tên, hệ số lương, năm vào làm (NVL). Biết rằng lương của mỗi nhân viên được tính bằng lương cơ bản * hệ số phụ cấp thâm niên (HSPCTN) trong đó

- ✓ Lương cơ bản = hệ số lương * mức lương tối thiểu (MLTT)
- ✓ $HSPCTN = (\text{năm hiện tại} - NVL)/100$
- ✓ MLTT là giống nhau cho tất cả các nhân viên và do nhà nước qui định. MLTT có thể thay đổi theo thời gian.

Bài tập

Bài tập 3: Viết chương trình quản lý và tính tiền điện hàng tháng cho các hộ gia đình trong chung cư biết rằng thông tin của mỗi hộ gồm có: Mã hộ, Tên chủ hộ, Số điện đầu kỳ, Số điện cuối kỳ, Loại hộ gia đình và có các phương thức như sau:

- ✓ Tính số điện tiêu thụ thực tế: Số điện đầu kỳ - số điện cuối kỳ
- ✓ Tính số điện ưu tiên: Nếu là loại hộ GĐ là A thì được ưu tiên 10, loại B được ưu tiên 5, còn lại là 0.
- ✓ Tính tiền điện: (số điện thực tế - ưu tiên)*Giá điện qui định.

Biết rằng giá điện qui định do chung cư thống nhất và áp dụng cho tất cả các hộ gia đình của chung cư, hiện tại là 3000.

Generic

- Định nghĩa một phương thức, một lớp mà không cần chỉ ra cụ thể kiểu dữ liệu là gì.
- Tuỳ vào kiểu dữ liệu mà người dùng truyền vào, lớp sẽ hoạt động theo kiểu dữ liệu đó.



Sử dụng Generic cho phương thức

Ví dụ: Phương thức hoán vị giá trị của 2 biến như sau:

```
public static void Swap(ref int a, ref int b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

- Thực hiện hoán vị giá trị của 2 biến nguyên
- Yêu cầu muốn hoán vị giá trị cho 2 số thực ???



Sử dụng Generic cho phương thức

Ví dụ: Phương thức hoán vị giá trị của 2 biến thực:

```
public static void Swap(ref double a, ref double b)
{
    double temp = a;
    a = b;
    b = temp;
}
```

- Các phương thức có cùng một logic, chỉ khác biệt duy nhất về kiểu dữ liệu.
- **Generic là một lựa chọn hợp lý → Tránh lặp code**

Sử dụng Generic cho phương thức

Ví dụ: Sử dụng Generic cho phương thức Swap:

```
public static void Swap<T> (ref T a, ref T b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

Gọi hàm:

```
int a = 5, b = 7; double c = 1.2, d = 5.8;
Swap<int>(ref a, ref b);
Swap<double>(ref c, ref d);
```



Sử dụng Generic cho Lớp

Khi xây dựng Lớp nhưng chưa xác định được kiểu dữ liệu của các biến thành viên, thuộc tính hoặc tham số (của các phương thức)

→ **Sử dụng Generic cho Lớp.**

Sử dụng Generic cho Lớp

```
public class MyGeneric<T>
{
    private T[] items;

    public T[] Items
    {
        get { return items; }
    }

    public MyGeneric(int Size)
    {
        items = new T[Size];
    }

    public T GetByIndex(int Index)
    {
        return items[Index];
    }

    public void SetItemValue(int Index, T Value)
    {
        items[Index] = Value;
    }
}
```

```
MyGeneric<int> Generic1 = new MyGeneric<int>(5);
Generic1.SetItemValue(0, 10);
Generic1.SetItemValue(1, 6);
int value1 = Generic1.GetByIndex(1);
```

```
MyGeneric<double> Generic2 = new MyGeneric<double>(5);
Generic2.SetItemValue(0, 2.14);
Generic2.SetItemValue(1, 4.0);
double value2 = Generic2.GetByIndex(1);
```

Generic

Một số loại **Generic Collections** thông dụng:

- **List <T>**: là một Collections giúp lưu trữ các phần tử liên tiếp nhưng có khả năng mở rộng kích thước.
- **Dictionary <Tkey, TValue>**: lớp lưu trữ dữ liệu dưới dạng cặp Key – Value, phần tử trong danh sách được truy xuất thông qua Key.
- **Stack<T>**: Lớp cho phép lưu trữ và thao tác dữ liệu theo cấu trúc LIFO (Last In First Out).
- **Queue<T>**: Lớp cho phép lưu trữ và thao tác dữ liệu theo cấu trúc FIFO (First In First Out).

Danh sách đối tượng

- **List<T>** là một Generic Collections giúp lưu trữ và quản lý một danh sách các đối tượng theo kiểu mảng (truy cập các phần tử bên trong thông qua chỉ số **index**), cho phép **tăng/ giảm số lượng phần tử linh động** khi có nhu cầu.
- Khai báo thư viện:

```
using System.Collections.Generic;
```

Danh sách đối tượng

■ Ví dụ:

//Khai báo và khởi tạo một list các số nguyên rỗng

```
List<int> myList = new List<int>();
```

//Khai báo và khởi tạo một list các số nguyên và
chỉ định sức chứa

```
List<int> myList = new List<int>(5);
```

//Khai báo và khởi tạo một list chứa đối tượng
SinhVien

```
List<SinhVien> myList = new List<SinhVien>();
```

Danh sách đối tượng

Một số thuộc tính của List<T>

- **Count**: trả về một số nguyên cho biết số lượng phần tử hiện có trong danh sách.
- **Capacity**: trả về một số nguyên cho biết sức chứa của danh sách.
- Ví dụ:

```
List<int> a = new List<int>(10);  
a.Add(9);  
a.Add(5);
```

Khi đó:

```
a.Capacity => 10  
a.Count => 2
```

Danh sách đối tượng

Một số phương thức của List<T>

int Add(object value);

Thêm phần tử value vào cuối danh sách, trả về vị trí phần tử thêm vào.

void Insert(int index, object value);

Chèn một phần tử value tại vị trí index.

void RemoveAt(int index);

Xóa phần tử tại vị trí index trong List.

void Remove(object value);

Xóa phần tử value xuất hiện đầu tiên trong List.

void RemoveRange(int index, int count);

Xóa một dãy gồm count phần tử kể từ vị trí index.

void Clear();

Xóa tất cả các phần tử trong List.



Danh sách đối tượng

Một số phương thức của List<T>

bool Contains(object value);

Kiểm tra phần tử có trong List hay không, nếu có trả về true.

void Sort();

Sắp xếp các phần tử trong List theo thứ tự tăng dần.

int IndexOf(object value);

int IndexOf(object value, int start);

int IndexOf(object value, int start, int count);

Trả về chỉ số của phần tử value trong count phần tử của List, kể từ vị trí start, nếu không tìm thấy trả về -1.

int BinarySearch(object value);

Tìm kiếm theo thuật toán nhị phân trong danh sách đã sắp xếp.

Ví dụ List

```
/*
 * Tạo 1 List các kiểu string và thêm 2 phần tử vào List.
 */
List<string> MyList4 = new List<string>();
MyList4.Add("Free");
MyList4.Add("Education");

// In giá trị các phần tử trong List
Console.WriteLine(" List ban dau: ");
Console.WriteLine(" Số lượng phần tử trong List là: {0}", MyList4.Count);
foreach (string item in MyList4)
{
    Console.Write(" " + item);
}

// Kiểm tra 1 phần tử có tồn tại trong List hay không.
bool exists = MyList4.Contains("Kteam");

if (exists == false)
{
    Console.WriteLine(" Không tìm thấy chuỗi Kteam trong List");
}
```



Ví dụ Quản lý sinh viên

Thông tin một sinh viên của một lớp bao gồm: mã sinh viên, họ tên, và điểm. Xây dựng chương trình quản lý sinh viên trong một lớp học với các yêu cầu:

- Nhập thông tin n sinh viên của lớp học;
- Xuất thông tin tất cả những sinh viên trong lớp;
- Xuất thông tin của sinh viên có điểm lớn hơn 8.
- Tìm thông tin của sinh viên dựa vào mã sinh viên hoặc họ tên và xuất thông tin học sinh tìm được.

Ví dụ Quản lý sinh viên

```
class SinhVien
{
    string sMSSV;
    public string MaSo{...}

    string sTen;
    public string TenSV{...}

    double dDiem;
    public double Diem{...}

    public void Nhap()
    {
        Console.WriteLine("\n Nhập mã số sinh viên:");
        MaSo = Console.ReadLine();
        Console.WriteLine("\n Nhập họ tên sinh viên:");
        TenSV = Console.ReadLine();
        Console.WriteLine("\n Nhập điểm sinh viên:");
        Diem = Double.Parse(Console.ReadLine());
    }
    public void Xuat()
    {
        Console.WriteLine("\n {0} \t {1} \t {2}", MaSo, TenSV, Diem);
    }
}
```



Ví dụ Quản lý sinh viên

```
class DSSinhVien
{
    List<SinhVien> lstSV = new List<SinhVien>();

    public List<SinhVien> ListSV
    {
        get { return lstSV; }
        set { lstSV = value; }
    }
    public void NhapDS()
    {
        Console.Write("\n nhap so luong Sinh vien");
        int n = int.Parse(Console.ReadLine());
        for (int i = 0; i < n; i++)
        {
            Console.Write("\n nhap thong tin SV thu {0}",i);
            SinhVien sv = new SinhVien();
            sv.Nhap();
            ListSV.Add(sv);
        }
    }
    public void XuatDS()
    {
        Console.Write("\n Danh sach sinh vien la:");
        foreach (SinhVien sv in ListSV)
        {
            sv.Xuat();
        }
    }
}
```

Bài tập

1. Viết chương trình xây dựng lớp nhân viên với các thông tin như mã nhân viên, họ tên, hệ số lương, năm vào làm (NVL). Biết rằng:

- ✓ Lương = lương cơ bản * hệ số phụ cấp thâm niên (HSPCTN).
- ✓ Lương cơ bản = hệ số lương * mức lương tối thiểu (MLTT).
- ✓ HSPCTN = (năm hiện tại – NVL)/100.
- ✓ MLTT là giống nhau cho tất cả các nhân viên và do nhà nước qui định. MLTT có thể thay đổi theo thời gian.

2. Xây dựng lớp danh sách nhân viên với các phương thức:

- ✓ Nhập danh sách nhân viên.
- ✓ Xuất danh sách nhân viên.
- ✓ Tính tổng lương nhân viên
- ✓ Tìm thông tin nhân viên có lương cao nhất
- ✓ Sắp xếp ds nhân viên theo thứ tự tăng dần của năm vào làm.