Chương 2:
LỚP & ĐỐI TƯỢNG

Giáo viên : ThS. Trần Văn Thọ
Đơn vị : Bộ môn KTHT & MMT

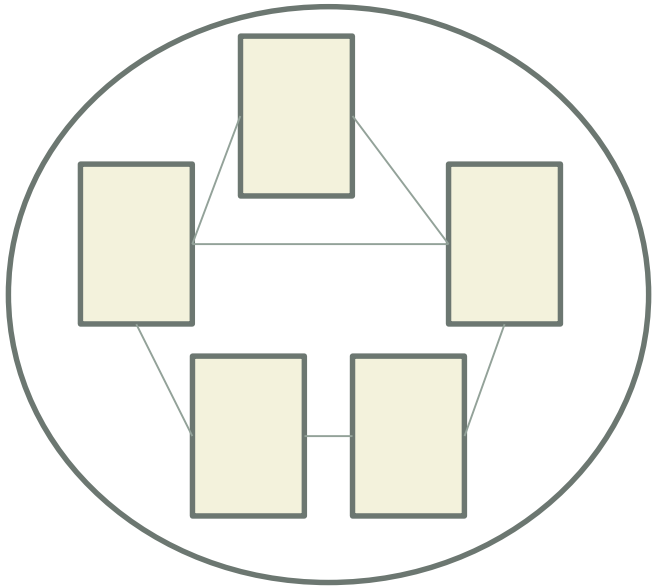
1 Giới thiệu lập trình hướng đối tượng

- Lập trình hướng đối tượng là tính năng quan trọng nhất của C++
- Ngôn ngữ C chỉ hỗ trợ phương pháp lập trình thủ tục
  - Chương trình được tổ chức thành các HÀM
  - HÀM và DỮ LIỆU tách rời nhau
- Lập trình hướng đối tượng (Object-Oriented Programming) giới thiệu một phương thức khác để xây dựng chương trình máy tính
  - Chương trình được tổ chức xoay quanh **các đối tượng (Object)**
  - **Đối tượng** bao hàm bên trong cả DỮ LIỆU và HÀM

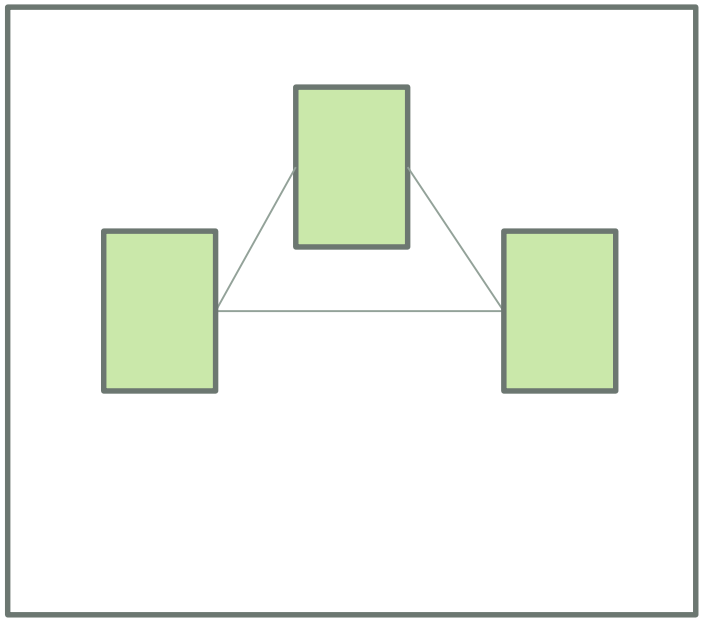
1 Giới thiệu lập trình hướng đối tượng

Lập trình thủ tục và lập trình hướng đối tượng

Điểm chung: **chia để trị**



Tổ chức thành các Hàm



Hướng đối tượng tổ chức thành các Lớp và Đối tượng

Ưu điểm: chương trình gần gũi hơn với các thực thể của bài toán

1 Giới thiệu lập trình hướng đối tượng

- Phương pháp hướng đối tượng được áp dụng rất phổ biến trong việc phát triển các chương trình phần mềm hiện nay: từ phần mềm hệ thống đến ứng dụng, truyền thông liên lạc và giải trí
- Bên cạnh C++, hầu hết các ngôn ngữ lập trình bậc cao đang thịnh hành đều hỗ trợ tính năng hướng đối tượng
  - Java
  - C#
  - Objective C
  - Python

2. Lớp

Khái niệm:

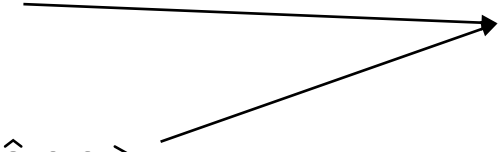
- Là một kiểu dữ liệu trừu tượng
    - Phản ánh một thực thể trong bài toán
- Do người lập trình định nghĩa
- Gồm 2 thành phần:
    - Thuộc tính: các thành phần mang dữ liệu
    - Phương thức: các hàm tương tác với các thuộc tính

2. Lớp

2.1 Định nghĩa:

Cú pháp:

```
class <tên lớp> {
private:
<các thành phần riêng>
public:
<các thành phần công cộng>
};
<định nghĩa các hàm thành phần chưa được định
 nghĩa bên trong lớp>
```



Các thuộc tính: danh sách các biến mang dữ liệu

Các phương thức: định nghĩa hoặc nguyên mẫu các hàm thành phần

2. Lớp

Ví dụ

```
class point {
 private:
    int x,y;
 public:
    void init(int ox, int oy);
    void move(int dx, int dy);
    void display();
};
```

Các thuộc tính: 2 biến thành phần x,y

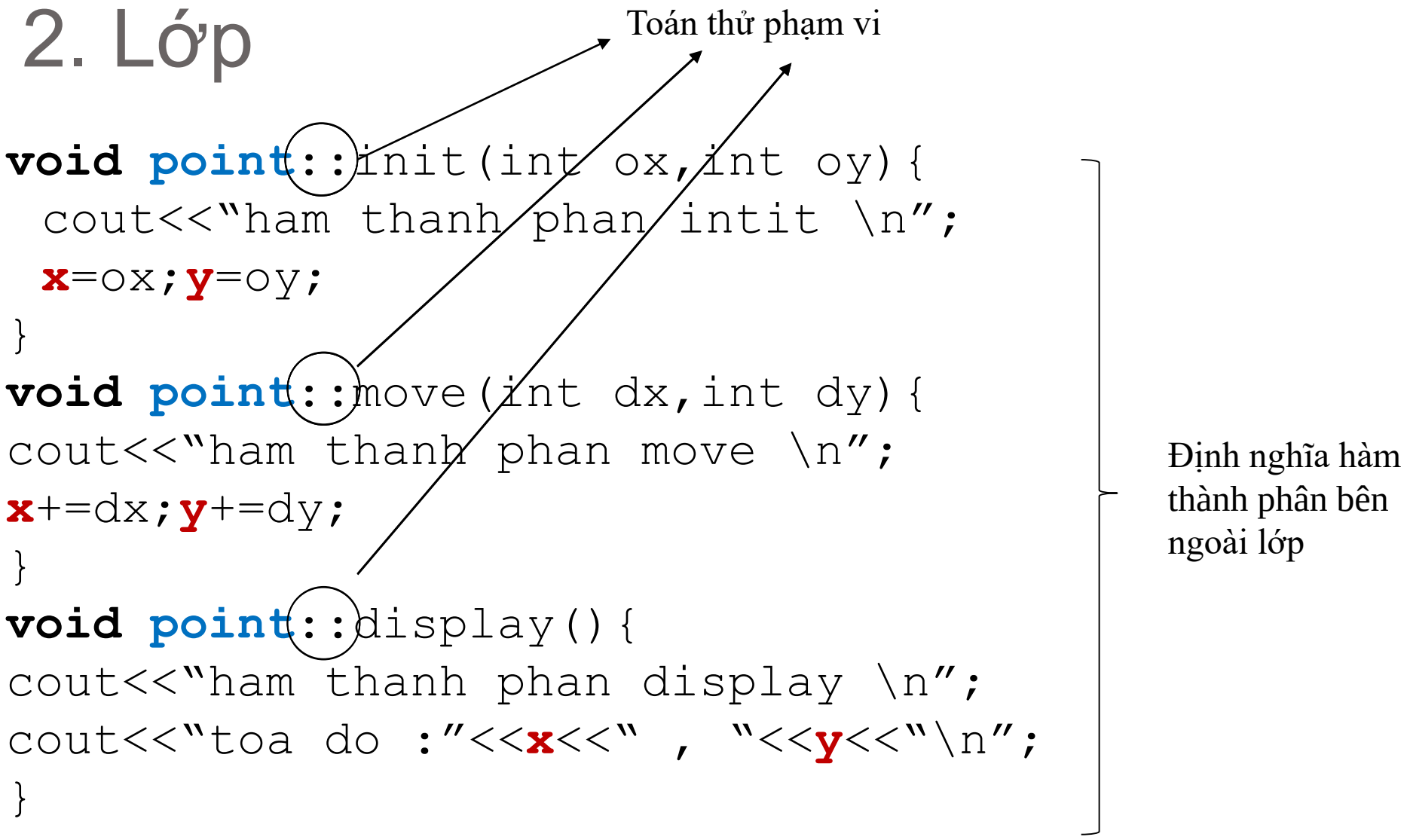
Các phương thức:
3 hàm thành phần
(nguyên mẫu)

2. Lớp

```
void point::init(int ox,int oy){
  cout<<"ham thanh phan intit \n";
  x=ox;y=oy;
}
void point::move(int dx,int dy){
cout<<"ham thanh phan move \n";
x+=dx;y+=dy;
}
void point::display(){
cout<<"ham thanh phan display \n";
cout<<"toa do :"<<x<<" , "<<y<<"\n";
}
```

Định nghĩa hàm thành phần bên ngoài lớp

2. Lớp


Toán thử phạm vi
void point::init(int ox,int oy){
cout<<"ham thanh phan intit \n";
x=ox;y=oy;
}
void point::move(int dx,int dy){
cout<<"ham thanh phan move \n";
x+=dx;y+=dy;
}
void point::display(){
cout<<"ham thanh phan display \n";
cout<<"toa do :"<<x<<" , "<<y<<"\n";
}
Định nghĩa hàm thành phân bên ngoài lớp

2. Lớp

Khai báo biến thành phần:

```
<kiểu> <tên biến>;
```

Khai báo hàm thành phần trong lớp

```
<kiểu trả về> <tên hàm>(<danh sách tham số>){}
```

Khai báo nguyên mẫu hàm thành phần trong lớp và định nghĩa lại ở ngoài lớp:

```
<kiểu trả về> <tên hàm>(<danh sách tham số>);
….
<kiểu trả về> <tên lớp>::<tên hàm>(<danh sách
 tham số>){
….<thân hàm>
}
```

2. Lớp

1.2 Phạm vi lớp

- **Private**:

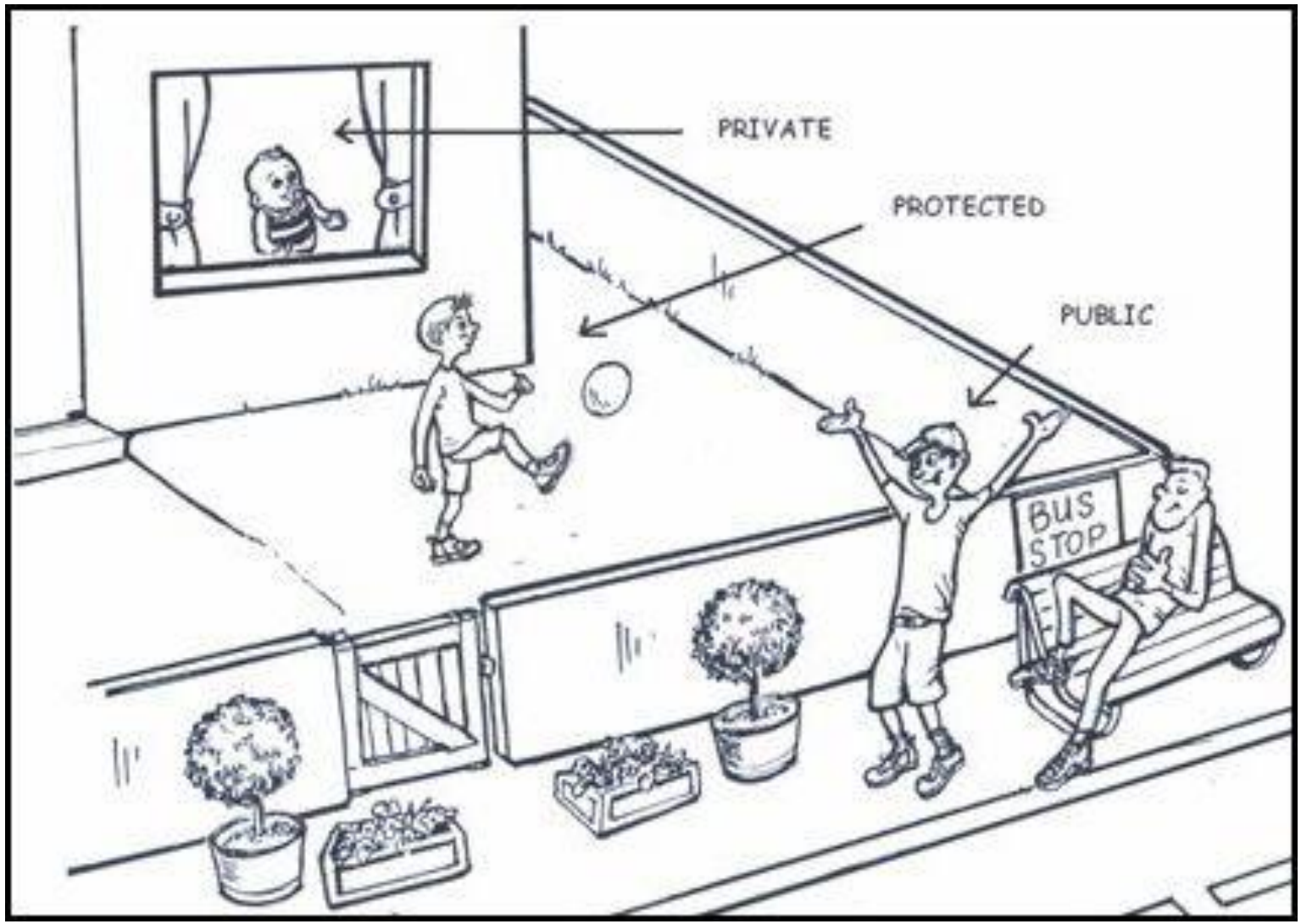
Các thành phần sau từ khóa này được bảo vệ và không thể truy cập từ bên ngoài, chỉ có thể truy cập bởi các hàm thành phần của lớp.

- **Public** :

Các thành phần được phép truy cập từ bất kỳ hàm nào

- **Protected :**

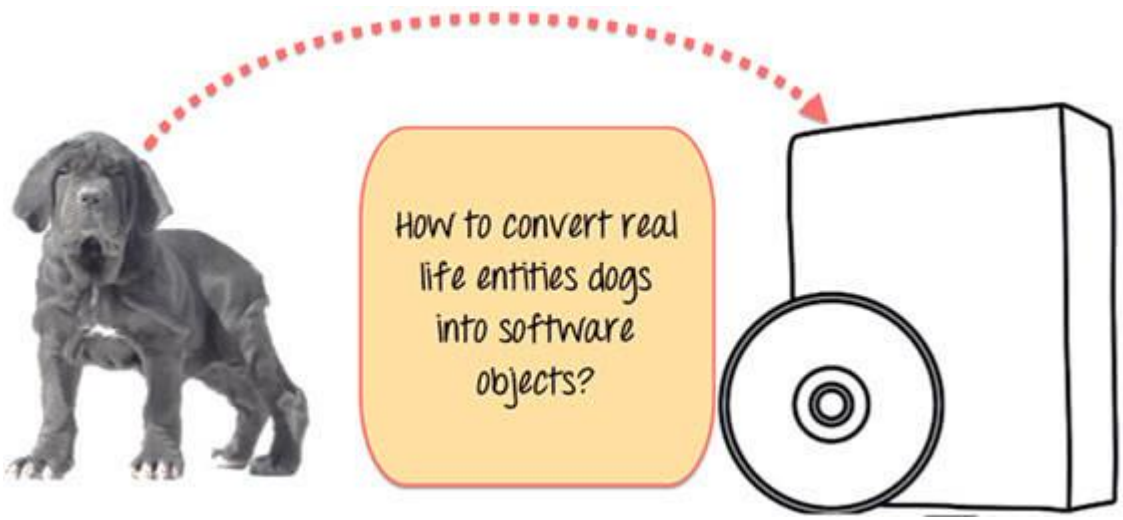
Tương tự private,nhưng cho phép hàm thành phần lớp con có thể truy cập.


PRIVATE
PROTECTED
PUBLIC
BUS STOP

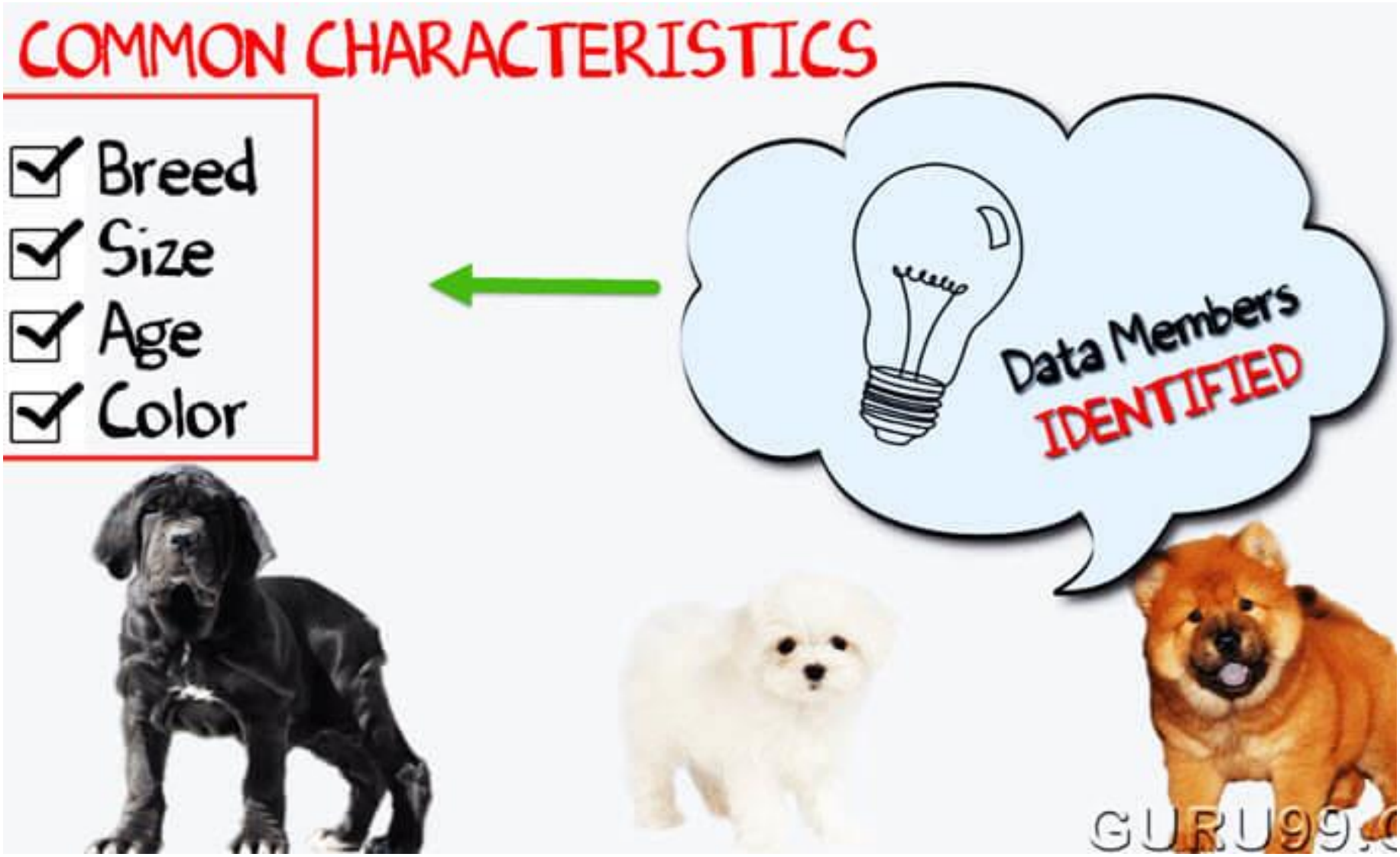
3. Đối tượng (Object)

2.1 Khái niệm:

- Đối tượng là một thể hiện của một lớp khi thực thi chương trình.
- Các thuộc tính trong lớp sẽ là các trường dữ liệu trong đối tượng
- Mỗi đối tượng có một bộ dữ liệu riêng, không chia sẻ với các đối tượng khác.
- Các đối tượng hoàn toàn xác định qua “tên” của nó.

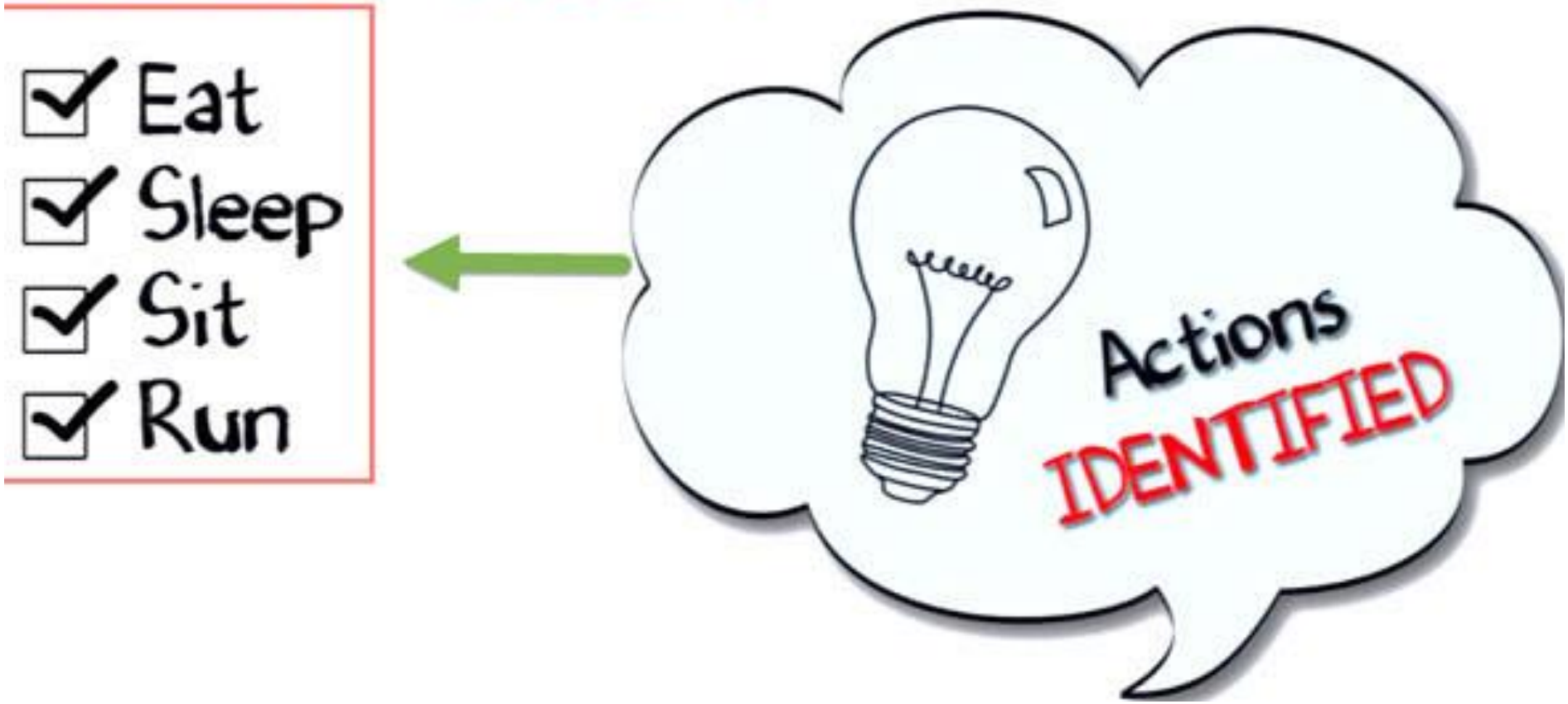


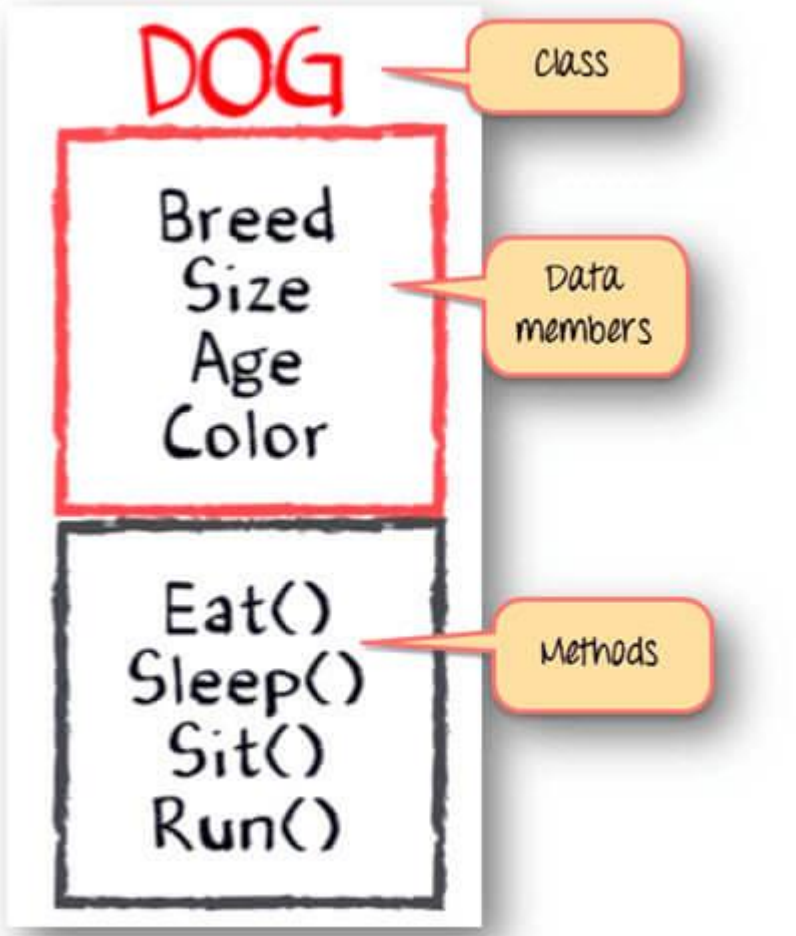
http://www.guru99.com/



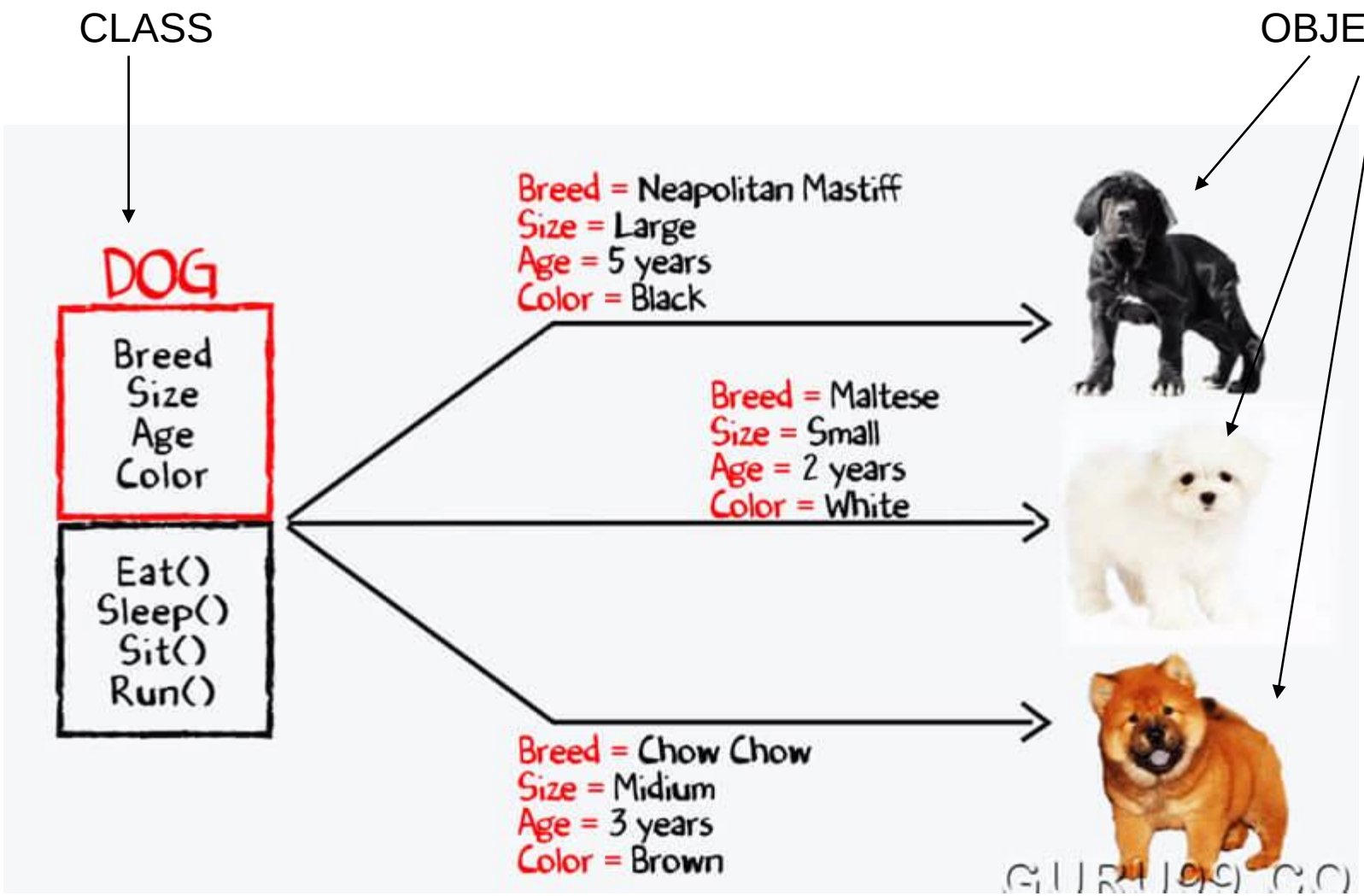
http://www.guru99.com/

COMMON ACTIONS





http://www.guru99.com/



http://www.guru99.com/

3. Đối tượng

2.2 Khai báo:

Cú pháp:

```
<tên lớp> <tên đối tượng>;
```

Gọi hàm thành phần:

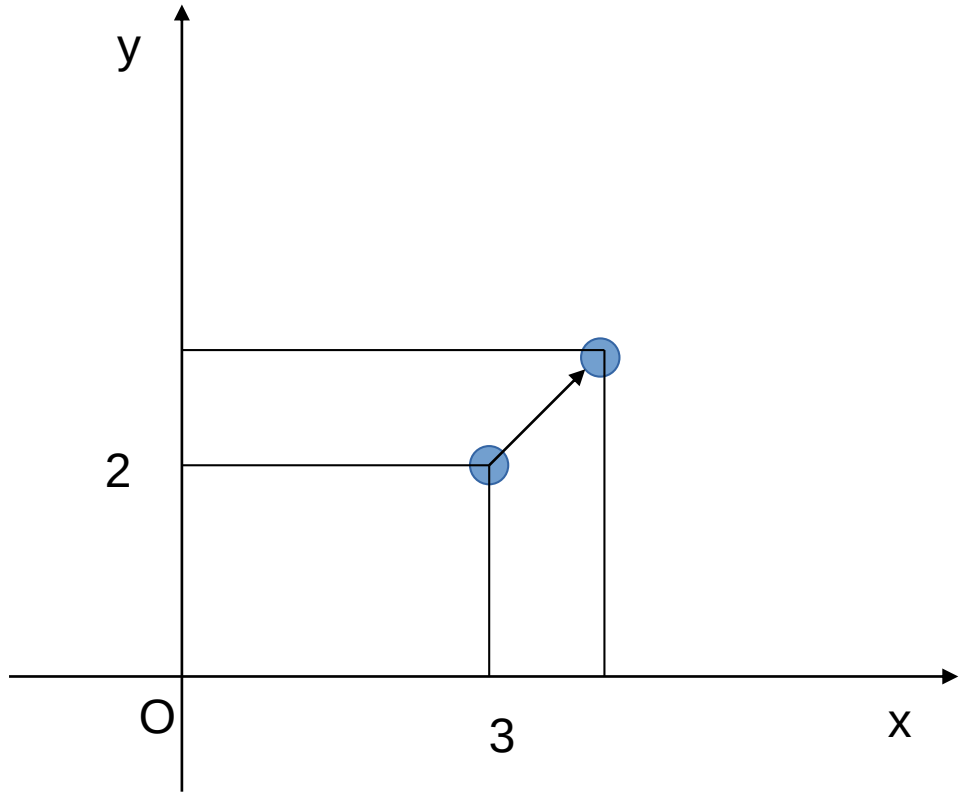
Cú pháp:

```
<tên đối tượng>.<tên hàm thành phần>(<danh sách tham số>);
```

3. Đối tượng

VD:

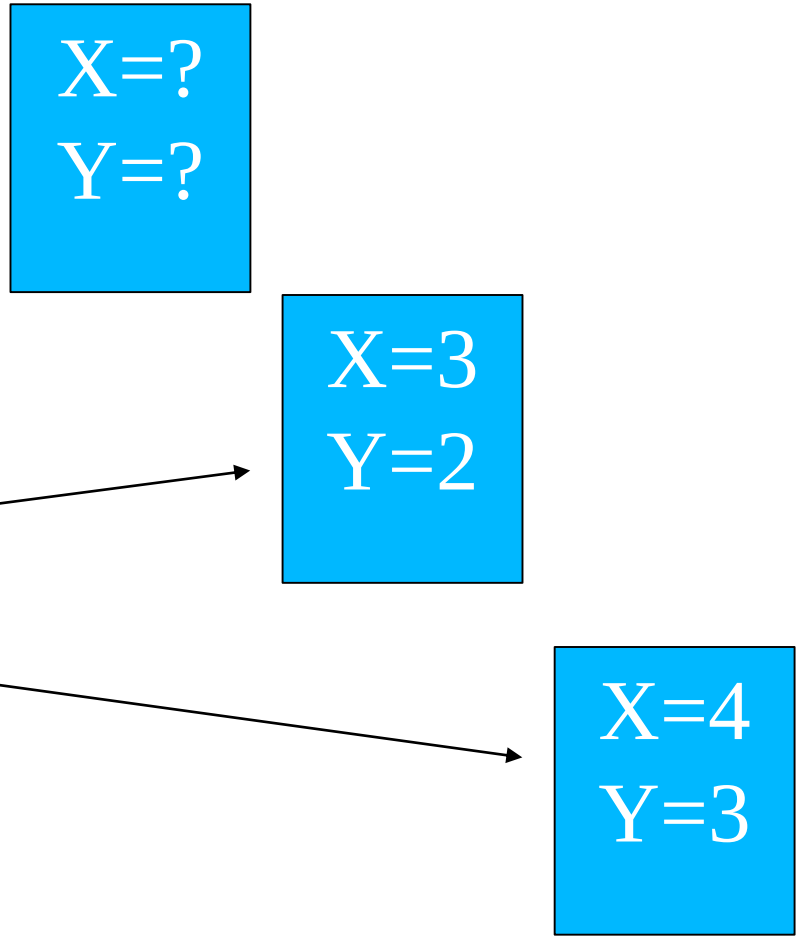
….
point p1;
p1.init(3,2);
p1.move(1,1);
p1.display();
…..



3. Đối tượng

VD:

….
point p1;
p1.init(3,2);
p1.move(1,1);
p1.display();
…..



4. Khả năng của hàm thành phần

a/ Chồng hàm (Overloading):

- Cho phép nhiều hàm trong một lớp có thể trùng tên nhau nhưng phải khác danh sách tham số (số lượng, kiểu, thứ tự tham số).
- Khi gọi hàm thành phần từ đối tượng, dựa vào danh sách tham số mà trình biên dịch sẽ chọn hàm thành phần tương ứng để thực hiện

4. Khả năng của hàm thành phần

a/ Chồng hàm:

- VD:

```
Class point{
int x,y;
public:
void init(){
x=0;y=0;
}
void init(int a,int b){
x=a;y=b;
}
}
```

```
int main{
point p1;
p1.init();//x=0,y=0
….
p1.init(3,2);//x=3,y=2
….
Return 0;
}
```

4. Khả năng của hàm thành phần

b/ Tham số với giá trị ngầm định:

- Cho phép thiết lập giá trị ngầm định cho tham số của hàm thành phần: nếu lời gọi hàm không chứa tham số, giá trị của tham số sẽ là giá trị ngầm định.
- Giá trị ngầm định được gán cho tham số ở khai báo nguyên mẫu hàm hoặc ở định nghĩa hàm.
- Các tham số nhận giá trị ngầm định phải đứng cuối trong danh sách tham số.

4. Khả năng của hàm thành phần

b/ Tham số với giá trị ngầm định:

VD:

```
class point{
int x,y;
public:
void init(int a=0,int b=0){
x=a;y=b;
}
}
```

```
int main{
point p1;
p1.init();//x=0,y=0
….
p1.init(3,2);//x=3,y=2
….
Return 0;
}
```

BÀI TẬP

1. Xây dựng lớp Point3D bao gồm các thành phần sau:

- Thuộc tính:
  - Tọa độ x:integer
  - Tọa độ y: integer;
  - Tọa độ z: integer;
- Phương thức:
  - Hàm khởi tạo vơi 3 tham số: x,y,z
  - Hàm khởi tạo không tham số, thiết lập x=0,y=0,z=0
  - Hàm move với 3 tham số x,y,z
  - Hàm display hiển thị x,y,z ra màn hình

BÀI TẬP

1. Xây dựng lớp Point2D bao gồm các thành phần sau:

- Thuộc tính:
  - Tọa độ x:integer
  - Tọa độ y: integer;
- Phương thức:
  - Tự xây dựng các phương thức.
- Yêu cầu: Viết hàm main để tính khoảng các giữa 2 tọa độ A và B.

5. Hàm tạo và hàm hủy

a/ Hàm tạo (Constructor)

- Là một hàm thành phần đặc biệt dùng để khởi tạo giá trị trong lớp và thực hiện các công việc chuẩn bị khác.
- Là hàm không thể thiếu, ngay cả khi người lập trình không định nghĩa thì chương trình cũng sẽ tự tạo một tạo ngầm định cho lớp.
- Tự động được gọi khi khai báo đối tượng.
- Có thể có tham số hoặc không có tham số
- Có thể định nghĩa chồng hàm với hàm tạo
- Tên hàm tạo phải trùng tên lớp và không có kiểu trả về.
- Phải có thuộc tính truy cập là public

5. Hàm tạo và hàm hủy

a/ Hàm tạo (Constructor)

- Cú pháp khai báo: Tên_lớp(Danh sách tham số)

{

Thân hàm tạo.

}

5. Hàm tạo và hàm hủy

a/ Hàm tạo
VD:

```
class point{
int x,y;
public:
point(int ox,int oy){
x=ox;y=oy;
}
point(){
x=0;y=0;
}
};
```

```
int main{
point p1(2,3);//x=2,y=3
….
point p2;//x=0,y=0
….
return 0;
}
```

5. Hàm tạo và hàm hủy

Hàm tạo ngầm định:
- Là hàm tạo được tự động sinh ra nếu người dùng không định nghĩa hàm tạo nào.
- Là hàm tạo không có tham số hoặc tham số nhận giá trị ngầm định
- Bắt buộc phải có trong trường hợp khai báo mảng đối tượng

VD: point array_point[10];

5. Hàm tạo và hàm hủy

b/ Hàm hủy:

- Là một hàm đặc biệt của lớp, có chức năng ngược với hàm tạo

- Thực hiện các công việc trước khi hủy một đối tượng: giải phóng các vùng nhớ,xóa đối tượng khỏi màn hình, in thông báo,…
- Không có kiểu trả về
- Tên bắt đầu bằng dấu ~ và tên lớp
- Không có tham số

5. Hàm tạo và hàm hủy

b/ Hàm hủy:

- Cú pháp: ~Tên_lớp()

{

Thân hàm hủy.

}

Chú ý:

Hàm hủy không có tham số, sẽ được tự động gọi trong các trường hợp sau:

- Kết thúc hàm
- Kết thúc chương trình
- Kết thúc một block
- Toán tử delete được gọi

5. Hàm tạo và hàm hủy

b/ Hàm hủy:

```
class so{
int x;
public:
so(int ox){
x=ox;
}
~so(){
    cout<<"huy doi tuong:"<<x;
    }
};
```

```
void dem(int a){
 so s(a);
}
int main(){
        for(int i=1;i<=2;i++)
                dem(i);
        return 0;
}
```

5. Hàm tạo và hàm hủy

c/ Hàm tạo sao chép:

- Dùng để sao chép dữ liệu từ đối tượng này sang đối tượng mới khi khởi tạo đối tượng.
- Nếu người dùng không định nghĩa, chương trình tự tạo ra hàm tạo sao chép ngầm định.
- Thường được định nghĩa lại trong trường hợp thuộc tính của lớp là biến dạng con trỏ.
- Cú pháp khai báo:

  Tên_Lớp (Tên_lớp &Tên_đối_tượng)
  { Thân hàm tạo sao chép}
- Tự động được gọi khi gặp 1 trong hai câu lệnh sau:
  - Point a2=a1;
  - Point a2(a1);

5. Hàm tạo và hàm hủy

c/ Hàm tạo sao chép:

```
class point{
int x,y;
public:
point(int ox,int oy){
x=ox;y=oy;
}
point(point &p){
cout<<"sao chep tu p \n";
x=p.x;
y=p.y;
}
};
```

```
int main(){
point p1(2,3);//x=2,y=3
….
point p2(p1);//x=2,y=3
….
return 0;
}
```

BÀI TẬP

6. Hàm toán tử

Khái niệm:

- Là các hàm định nghĩa các **phép toán** giữa các đối tượng của lớp.
  - Cũng là một loại hàm: có tham số và giá trị trả về
  - Gần như không có hàm toán tử nào được định nghĩa sẵn (trừ = và []), người dùng phải tự định nghĩa
- Có thể chồng các hàm toán tử
- 2 cách để định nghĩa:
  - Hàm thành phần của lớp
  - Hàm bạn (trừ = và [])
- Cú pháp:
  - **Kiểu_trả_về operator Toán_tử(Danh_sách_tham_số) {…}**

6. Hàm toán tử

Toán tử 1 ngôi:

- Là các phép toán có 1 số hạng tham gia, chính là đối tượng gọi hàm.

VD: ~,!,++,--, -, sizeof()

- Không có tham số
- Đối tượng gọi hàm là toán hạng duy nhất

Chú ý: Hai toán tử ++ và -- có thể dùng ở trước (tiền tố) hoặc sau (hậu tố). Để phân biệt, toán tử tiền tố được định nghĩa như bình thường, còn toán tử hậu tố có thêm tham số thứ hai với kiểu int (dù không dùng).

6. Hàm toán tử

Ví dụ:

- Xây dựng lớp point2D.
- Xây dựng hàm toán tử ++ dạng tiền tố và hậu tố cho lớp point2D.

6. Hàm toán tử

Toán tử 2 ngôi:

- Là các phép toán có 2 số hạng tham gia.

VD: +,-,*,/,%,<<,>>,<,>,<=,>=,==,!=,&,|,^,&&,||

- Nguyên tắc:
  + Không định nghĩa toán tử mới
  + Một trong số các toán hạng phải là đối tượng
  + Khi định nghĩa hàm toán tử là một hàm thành phần thì:
    - Hàm chỉ có 1 tham số
    - Đối tượng thực sự gọi hàm toán tử là số hạng trái
    - Tham số của hàm toán tử là số hạng phải

6. Hàm toán tử

VD:

```
class Point{
int x,y;
public:
Point(int ox,int oy){
        x=ox;y=oy;
}
Point operator+(Point p){
        Point kq(0,0);
        kq.x=x+p.x;
        kq.y=y+p.y;
        return kq;
}
};
```

```
int main(){
Point p1(2,3);
….
Point p2(3,4);
….
Point c(0,0);
c=p1+p2; //c.x=5, c.y=7
//c= p1.operator+(p2)
return 0;
}
```

Bài tập

7. Hàm bạn

- Hàm bạn trong C++ của một lớp được định nghĩa bên ngoài phạm vi lớp đó, nhưng nó có quyền truy cập tất cả thành viên private và protected của lớp đó. Hàm bạn không là các hàm thành viên của lớp.
- Cách khai báo:
  - Hàm bạn được khai báo nguyên mẫu hàm bên trong lớp và được định nghĩa ở bên ngoài lớp.

friend Kiểu_trả_về Tên_hàm(Danh_sách_tham_số);

7. Hàm bạn

VD:

```
class point{
private:
       int x,y;
public:
    point(int ox,int oy){
        x=ox;y=oy;
    }
   friend void hienthi(point p);
};

void hienthi(point p)
{
    cout<<"x= "<<p.x<<" y=
"<<p.y;
}
```

```
int main()
{
     point p1(3,4);
     hienthi(p1);
     return 0;
}
```

7. Hàm bạn

Chú ý:

- Phải sử dụng hàm bạn để định nghĩa toán tử nếu số hạng không phải là đối tượng
- Cú pháp:
  - friend Kiểu_trả_về operatorX (danh sách tham số)
    - X: toán tử
  - Tham số: một trong 2 tham số phải là đối tượng của lớp chứa hàm bạn, thứ tự của tham số chính là thứ tự của số hạng trong phép toán

7. Hàm bạn

VD:

```
class point{
       int x,y;
public:
    point(int ox,int oy){
        x=ox;y=oy;
    }
    void hienthi()
    {
         cout<<"x= "<<x<<" y= "<<y<<endl;
    }
friend point operator+(int a, point p);
};
point operator+(int a, point p){
       p.x+=a;p.y+=a;
       return p;
}
```

```
int main(){
    point p1(2,3);
    p1.hienthi();
    p1=3+p1;
    p1.hienthi();
    return 0;
}
```

BÀI TẬP