



Lớp và Đối tượng

Lớp trong C++



- ❖ Một lớp bao gồm các thành phần dữ liệu (*hay là thuộc tính*) và các phương thức (*hay là hàm thành phần*)
- ❖ Lớp trong C++ thực chất là một kiểu dữ liệu do người sử dụng định nghĩa.

Đối tượng



- ❖ **Lưu giữ trạng thái:** mỗi đối tượng có trạng thái (dữ liệu của nó) và các thao tác đi kèm.
- ❖ **Định danh:** Mỗi đối tượng bất kể đang ở trạng thái nào đều có định danh và được đối xử như một thực thể riêng biệt.
- ❖ **Thông điệp:** là phương tiện để một **đối tượng A** chuyển thông tin tới **đối tượng B**, yêu cầu B thực hiện một trong số các thao tác của B.

Lớp đối tượng - Class



- ❖ **Lớp**: là khuôn mẫu để tạo các đối tượng (tạo các thể hiện) → *Mỗi đối tượng có cấu trúc và hành vi giống như lớp đối tượng mà nó được tạo từ đó.*
- ❖ Lớp là cái ta thiết kế và lập trình.
- ❖ Đối tượng là cái ta tạo (từ một lớp) tại thời gian chạy.

Classes & Objects



```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
}
```

```
Rectangle r1;
Rectangle r2;
Rectangle r3;
```

⋮

```
int a;
```

Khai báo lớp



```
class <tên lớp> {
```

```
  private:
```

```
    <khai báo các thành phần riêng trong từng đối tượng>
```

```
  protected:
```

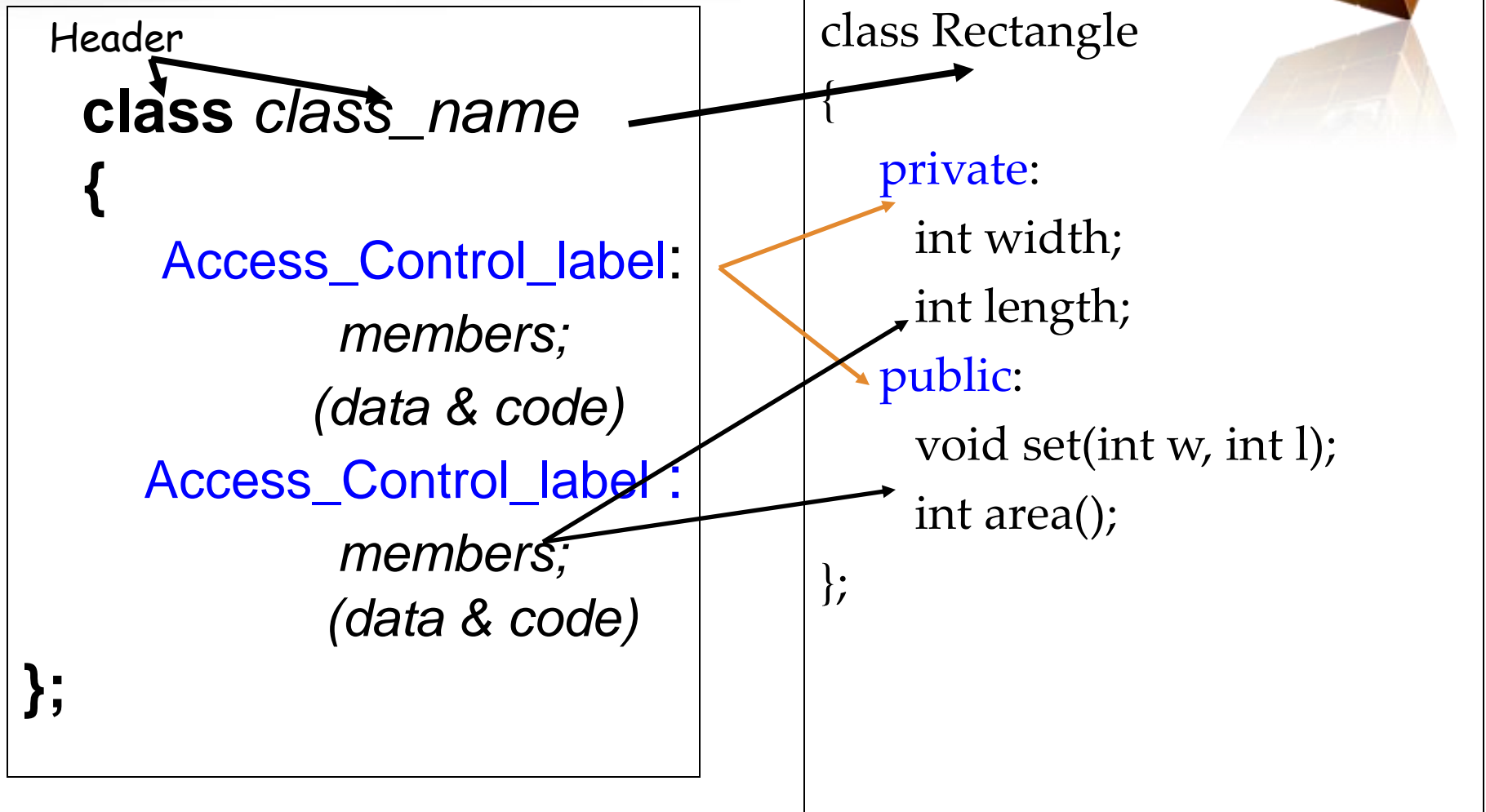
```
    <khai báo các thành phần riêng trong từng đối tượng, có thể truy  
    cập từ lớp dẫn xuất>
```

```
  public:
```

```
    <khai báo các thành phần giao tiếp của từng đối tượng>
```

```
};
```

Defining a class



Định nghĩa các hàm thành phần ở bên ngoài khai báo lớp



<tên kiểu giá trị trả về> <tên lớp>::<tên hàm> (<danh sách tham số>)

{

<nội dung >

}

void point::display() { }

Define a Member Function



```
class Rectangle
{
    private:
        int width, length;
    public:
        void set (int w, int l);
        int area() {return width*length; }
}
```

class name

member function name

inline

```
r1.set(5,8);
rp->set(8,10);
```

```
void Rectangle :: set (int w, int l)
{
    width = w;
    length = l;
}
```

scope operator

Khai báo đối tượng



❖ Tạo đối tượng:

<tên lớp> <tên đối tượng> ;

<tên đối tượng> = new <tên lớp> ;

❖ Gọi hàm thành phần của lớp

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

<tên con trỏ đối tượng>→<tên hàm thành phần>(<danh sách các tham số nếu có>);

Declaration of an Object



```
class Rectangle
```

```
{
```

```
    private:
```

```
        int width;
```

```
        int length;
```

```
    public:
```

```
        void set(int w, int l);
```

```
        int area();
```

```
}
```

r1 is statically allocated

```
main()
```

```
{
```

```
    Rectangle r1;
```

```
    → r1.set(5, 8);
```

```
}
```

r1

**width = 5
length = 8**

Declaration of an Object

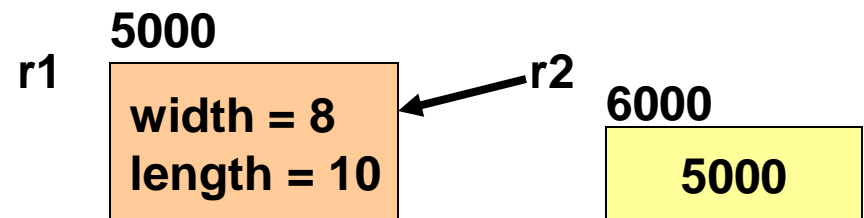


r2 is a pointer to a Rectangle object

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
}
```

```
main()
{
    Rectangle r1;
    r1.set(5, 8);           //dot notation

    Rectangle *r2;
    r2 = &r1;
    r2->set(8,10);         //arrow notation
}
```



Declaration of an Object



```
class Rectangle
```

```
{
```

```
    private:
```

```
        int width;
```

```
        int length;
```

```
    public:
```

```
        void set(int w, int l);
```

```
        int area();
```

```
}
```

r3 is dynamically allocated

```
main()
```

```
{
```

```
    Rectangle *r3;
```

```
    r3 = new Rectangle();
```

```
    r3->set(80,100);    //arrow notation
```

```
    delete r3;
```

```
    ➡ r3 = NULL;
```

```
}
```

r3

6000

NULL

Ví dụ



❖ Xây dựng lớp Điểm (Point) trong hình học 2D

- Thuộc tính (dữ liệu)
 - Tung độ
 - Hoành độ
- Thao tác (phương thức)
 - Khởi tạo
 - Di chuyển
 - In ra màn hình
 -

Ví dụ



```
/*point.cpp*/  
#include <iostream.h>  
#include <conio.h>  
class point {  
    /*khai báo các thành phần dữ liệu riêng*/  
    private:  
        int x,y;  
    /*khai báo các hàm thành phần công cộng*/  
    public:  
        void init(int ox, int oy);  
        void move(int dx, int dy);  
        void display();  
};
```



```

void point::init(int ox, int oy) {
    cout<<"Ham thanh phan init\n";
    x = ox; y = oy;
    /*x,y là các thành phần của đối tượng gọi hàm thành phần*/
}
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";
}

```



```
void main() {  
    point p;  
    p.init(2,4); /*gọi hàm thành phần từ đối tượng*/  
    p.display();  
    p.move(1,2);  
    p.display();  
    getch();  
}
```



Ham thanh phan init

Ham thanh phan display

Toa do: (2,4)

Ham thanh phan move

Ham thanh phan display

Toa do: (3,6)

Từ khoá xác định thuộc tính truy xuất




- ❖ Trong định nghĩa của lớp ta có thể xác định khả năng truy xuất thành phần của một lớp nào đó từ bên ngoài phạm vi lớp. Ta có **private** và **public** là các từ khoá xác định thuộc tính truy xuất
- ❖ Mọi thành phần được liệt kê trong phần public đều có thể **truy xuất trong bất kỳ hàm nào**
- ❖ Những thành phần được liệt kê trong phần private **chỉ được truy xuất bên trong phạm vi lớp**

Từ khoá xác định thuộc tính truy xuất (t.t)



- ❖ Trong lớp có thể có nhiều nhãn private và public
- ❖ Mỗi nhãn này có phạm vi ảnh hưởng cho đến khi gặp một nhãn kế tiếp hoặc hết khai báo lớp
- ❖ Nhãn private đầu tiên có thể bỏ đi vì C++ ngầm hiểu rằng các thành phần trước nhãn public đầu tiên là private

```
class tamgiac{  
    private:  
        float a,b,c; /*độ dài ba cạnh*/  
    public:  
        void nhap(); /*nhập vào độ dài ba cạnh*/  
        void in(); /*in ra các thông tin liên quan đến tam giác*/  
    private:  
        int loaitg(); /*cho biết kiểu của tam giác: 1-d,2-vc,3-c,4-v,5-t*/  
        float dientich(); /*tính diện tích của tam giác*/  
};
```



```
class tamgiac{
    private:
        float a,b,c; /*độ dài ba cạnh*/
        int loaitg(); /*cho biết kiểu của tam giác: 1-d,2-vc,3-c,4-v,5-t*/
        float dientich(); /*tính diện tích của tam giác*/
    public:
        void nhap(); /*nhập vào độ dài ba cạnh*/
        void in(); /*in ra các thông tin liên quan đến tam giác*/
};
```


Đối tượng như tham số của hàm thành phần



- ❖ Hàm thành phần có quyền truy nhập đến các thành phần private của đối tượng gọi nó:

```
void point::init(int xx,int yy)
{
    x=xx;
    y=yy;
}
```

Đối tượng như tham số của hàm thành phần (t.t)



- ❖ Hàm thành phần có quyền truy nhập đến tất cả các thành phần private của các đối tượng, tham chiếu đối tượng hay con trỏ đối tượng có cùng kiểu lớp khi được dùng là tham số hình thức của nó

```
int trung(point pt) {return(x==pt.x && y==pt.y);}
```

```
int trung(point *pt) {return(x==pt->x && y==pt->y);}
```

```
int trung(point &pt) {return(x==pt.x && y==pt.y);}
```

Con trỏ this



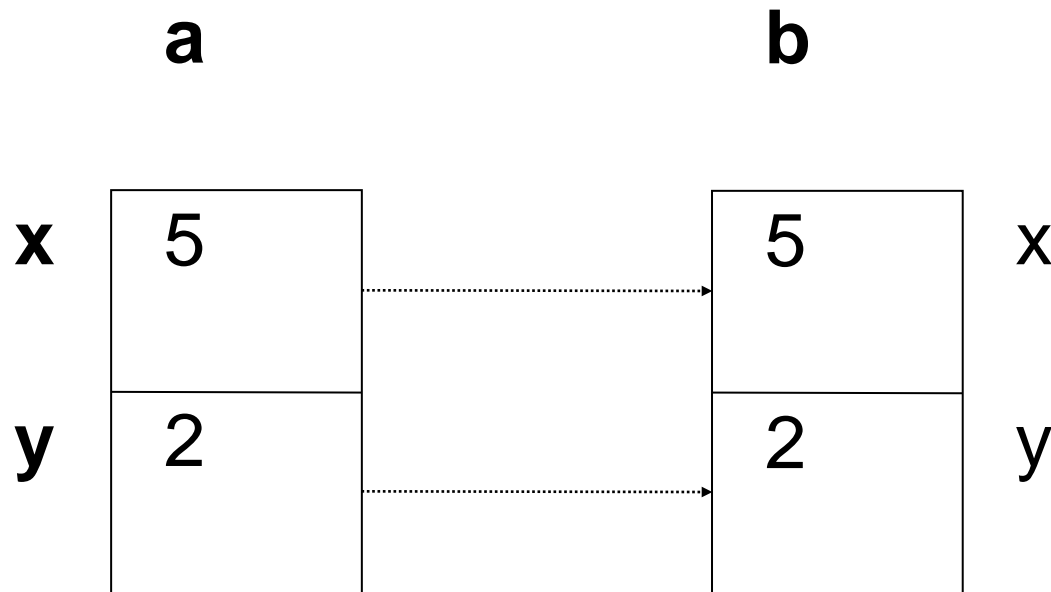
- ❖ Từ khoá this trong định nghĩa của các hàm thành phần lớp dùng để xác định địa chỉ của đối tượng dùng làm **tham số ngầm định** cho hàm thành phần
 - ❖ Con trỏ this tham chiếu đến đối tượng đang gọi hàm thành phần
- ```
int trung(point pt)
{return(this->x==pt.x && this->y==pt.y);}
```

# Phép gán đối tượng



```
point a, b;
a.init(5,2);
b=a;
```

việc sao chép giá trị các thành phần dữ liệu (x, y) từ đối tượng a sang đối tượng b tương ứng từng đôi một



# Object Initialization



## 1. By Assignment

```
#include <iostream.h>

class circle
{
 public:
 double radius;
};
```

- Only work for public data members
- No control over the operations on data members

```
int main()
{
 circle c1; // Declare an instance of the class circle
 c1.radius = 5; // Initialize by assignment

 circle c2 = {-10}; // Initialize by aggregation, not valid for a circle
}
```

# Object Initialization



```
#include <iostream.h>

class circle
{
private:
 double radius;

public:
 void set (double r)
 {radius = r;}
 double get_r ()
 {return radius;}
};
```

## 2. By Public Member Functions

- Accessor
- Implementor

```
int main(void) {
 circle c; // an object of circle class
 c.set(5.0); // initialize an object with a public member function
 cout << "The radius of circle c is " << c.get_r() << endl;
 // access a private data member with an accessor
}
```

# Hàm thiết lập - Constructor



- ❖ **Constructor** là một loại phương thức đặc biệt dùng để khởi tạo thể hiện của lớp.
- ❖ Bất kể loại cấp phát bộ nhớ nào được sử dụng (tĩnh, động), mỗi khi một thể hiện của lớp được tạo, một hàm constructor nào đó của lớp sẽ được gọi.



```
class point
```

```
{
```

```
private:
```

```
 /*khai báo các thành phần dữ liệu*/
```

```
 int x;
```

```
 int y;
```

```
public:
```

```
 /*khai báo các thành phần hàm*/
```

```
 point(int ox,int oy) {x=ox;y=oy;} /*hàm thiết lập*/
```

```
 void move(int dx,int dy) ;
```

```
 void display();
```

```
};
```

```
point a(5,2); //int i=5
```

# Constructor – Lưu ý:



- ❖ Constructor có cùng tên với tên của lớp
- ❖ Constructor **không** có giá trị trả về (kể cả void)
- ❖ Constructor phải có thuộc tính public
- ❖ Constructor có thể được khai báo chồng như các hàm C++ thông thường khác
- ❖ Constructor có thể được khai báo với các tham số có giá trị ngầm định

```
class point
```

```
{
```

```
private:
```

```
/*khai báo các thành phần dữ liệu*/
```

```
int x;
```

```
int y;
```

```
public:
```

```
/*khai báo các thành phần hàm*/
```

```
point() {x=0;y=0};
```

```
point(int ox,int oy) {x=ox;y=oy;} /*hàm thiết lập*/
```

```
void move(int dx,int dy) ;
```

```
void display();
```

```
};
```

```
point a(5,2);
```

```
point b;
```

```
class point
```

```
{
```

```
private:
```

```
 /*khai báo các thành phần dữ liệu*/
```

```
 int x;
```

```
 int y;
```

```
public:
```

```
 /*khai báo các thành phần hàm*/
```

```
 point() {x=0;y=0;}
```

```
 point(int ox,int oy=1) {x=ox;y=oy;} /*hàm thiết lập*/
```

```
 void move(int dx,int dy) ;
```

```
 void display();
```

```
};
```

```
point a(5,2);
```

```
point b;
```

```
point c(3);
```

# Constructor mặc định



❖ **Constructor mặc định (default constructor)** là constructor được gọi khi thể hiện được khai báo mà không có đối số nào được cung cấp

- `MyClass x;`
- `MyClass* p = new MyClass;`

❖ Ngược lại, nếu tham số được cung cấp tại khai báo thể hiện, trình biên dịch sẽ gọi phương thức constructor khác (overload)

- `MyClass x(5);`
- `MyClass* p = new MyClass(5);`

# Constructor mặc định – lưu ý:



- ❖ Đối với constructor mặc định, nếu ta không cung cấp một phương thức constructor nào, C++ sẽ tự sinh constructor mặc định là một phương thức rỗng
- ❖ Tuy nhiên, nếu ta không định nghĩa constructor mặc định nhưng lại có các constructor khác, trình biên dịch sẽ báo lỗi không tìm thấy constructor mặc định nếu ta không cung cấp tham số khi tạo thể hiện.

```
class point
```

```
{
```

```
private:
```

```
 /*khai báo các thành phần dữ liệu*/
```

```
 int x;
```

```
 int y;
```

```
public:
```

```
 /*khai báo các thành phần hàm*/
```

```
 point(int ox,int oy=1) {x=ox;y=oy;}/*hàm thiết lập*/
```

```
 void move(int dx,int dy) ;
```

```
 void display();
```

```
};
```

```
point a(5,2);
```

```
point b;
```

```
point c(3);
```



```
class point
```

```
{
```

```
private:
```

```
 /*khai báo các thành phần dữ liệu*/
```

```
 int x;
```

```
 int y;
```

```
public:
```

```
 /*khai báo các thành phần hàm*/
```

```
 point(int ox=0,int oy=0) {x=ox;y=oy;}/*hàm thiết lập*/
```

```
 void move(int dx,int dy) ;
```

```
 void display();
```

```
};
```

```
point a(5,2);
```

```
point b;
```

```
point c(3);
```

# Destructor



- ❖ Cũng như một phương thức constructor được gọi khi một đối tượng được tạo, loại phương thức thứ hai, destructor, được gọi ngay trước khi **thu hồi một đối tượng**
- ❖ Destructor thường được dùng để thực hiện mọi việc dọn dẹp cần thiết trước khi một đối tượng bị huỷ
- ❖ Destructor không có giá trị trả về, và không thể định nghĩa lại (nó không bao giờ có tham số)
- ❖ Phương thức destructor trùng tên với tên lớp nhưng có dấu ~ đặt trước
- ❖ Destructor phải có thuộc tính public



```
class vector {
 int n; //số chiều
 float *v; //vùng nhớ tọa độ
public:
 vector(); //Hàm thiết lập không tham số
 vector(int size); //Hàm thiết lập một tham số
 vector(int size, float *a);
 ~vector(); //Hàm huỷ bỏ, luôn luôn không có tham số
 void display();
};
```

# Hàm bạn, lớp bạn



- ❖ Giả sử có lớp Vector, lớp Matrix
- ❖ Cần viết hàm **nhân** 1 Vector và 1 Matrix
- ❖ Hàm **nhân**:
  - Không thể thuộc lớp Vector
  - Không thể thuộc lớp Matrix
  - Không thể tự do
- ❖ Giải pháp: xây dựng hàm truy cập dữ liệu.

# Hàm bạn - friend function



- ❖ Hàm bạn không thuộc lớp, nhưng có quyền truy cập các thành viên private.
- ❖ Khi *định nghĩa một lớp*, có thể khai báo một hay nhiều hàm “bạn” (bên ngoài lớp).
- ❖ Ưu điểm: kiểm soát các truy nhập ở cấp độ lớp - không thể áp đặt hàm bạn cho một lớp nếu điều đó không được dự trù trước trong khai báo của lớp.

# Hàm bạn - Lưu ý



- ❖ Vị trí của khai báo “bạn bè” trong lớp hoàn toàn tùy ý
- ❖ Trong hàm bạn, không còn tham số ngầm định this như trong hàm thành phần
- ❖ Hàm bạn của một lớp có thể có một hay nhiều tham số, hoặc có giá trị trả về thuộc kiểu lớp đó

# Ví dụ: Hàm tự do bạn của một lớp



```
class point {
 int x, y;
public:
 point(int abs =0, int ord =0) {
 x = abs;y = ord;
 }
 friend int coincide (point,point);
};
```

# Các phương thức Truy vấn



- ❖ Các phương thức truy vấn (query method) là các phương thức dùng để hỏi về giá trị của các thành viên dữ liệu của một đối tượng
- ❖ Có nhiều loại câu hỏi truy vấn có thể:
  - truy vấn đơn giản (*“giá trị của x là bao nhiêu?”*)
  - truy vấn điều kiện (*“thành viên x có lớn hơn 10 không?”*)
  - truy vấn dẫn xuất (*“tổng giá trị của các thành viên x và y là bao nhiêu?”*)
- ❖ Đặc điểm quan trọng của phương thức truy vấn là nó không nên thay đổi trạng thái hiện tại của đối tượng



# Các phương thức Truy vấn (t.t)



- ❖ Đối với các truy vấn đơn giản, quy ước đặt tên phương thức: tiền tố “get”, tiếp theo là tên của thành viên
  - `int getX();`
  - `int getSize();`
- ❖ Các loại truy vấn khác nên có tên có tính mô tả
- ❖ Truy vấn điều kiện nên có tiền tố “is”

# Các phương thức Cập nhật



- ❖ Ngược lại với truy vấn, các phương thức cập nhật thường thay đổi trạng thái của đối tượng bằng cách sửa đổi một hoặc nhiều thành viên dữ liệu của đối tượng đó.
- ❖ Dạng đơn giản nhất của các phương thức cập nhật là gán một giá trị nào đó cho một thành viên dữ liệu.
- ❖ Đối với dạng cập nhật đơn giản, quy ước đặt tên: dùng tiền tố “set” kèm theo tên thành viên cần sửa.
  - `int setX(int);`

# Truy vấn và Cập nhật – ưu điểm



- ❖ Nếu các phương thức get/set chỉ có nhiệm vụ cho ta đọc và ghi giá trị cho các thành viên dữ liệu, quy định các thành viên private để được ích lợi gì?
  - Ngoài việc bảo vệ các nguyên tắc đóng gói, ta còn cần kiểm tra xem giá trị mới cho thành viên dữ liệu có hợp lệ hay không.
  - Sử dụng phương thức truy vấn cho phép ta thực hiện việc kiểm tra trước khi thực sự thay đổi giá trị của thành viên.
  - Cho phép chỉ các dữ liệu có thể truy vấn hay thay đổi được truy cập đến.

## Ví dụ: Cập nhật



```
int Student::setGPA(double newGPA)
{
 if ((newGPA >= 0.0) && (newGPA <= 4.0))
 {
 this->gpa = newGPA;
 return 0; // Return 0 to indicate success
 }
 else
 {
 return -1; // Return -1 to indicate failure
 }
}
```

# Thành viên dữ liệu tĩnh – Static Data Member



- ❖ Đối với class, **static** dùng để khai báo thành viên dữ liệu dùng chung cho mọi thể hiện của lớp.
  - một bản duy nhất tồn tại trong suốt quá trình chạy của chương trình
  - dùng chung cho tất cả các thể hiện của lớp
  - bất kể lớp đó có bao nhiêu thể hiện

# Static Data Member – Ví dụ



```
class Rectangle
{
 private:
 int width;
 int length;
 static int count;
 public:
 → void set(int w, int l);
 int area();
}
```

```
Rectangle r1;
Rectangle r2;
Rectangle r3;
```

count

r1

width  
length

r2

width  
length

r3

width  
length

# Thành viên dữ liệu tĩnh - Ví dụ



## ❖ Đếm số đối tượng MyClass

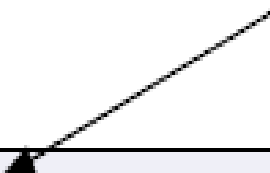
```
class MyClass {
 public:
 MyClass(); // Constructor
 ~MyClass(); // Destructor

 void printCount(); // Output current value of count

 private:
 static int count; // static member to store
 // number of instances of MyClass
};
```

thành viên tĩnh count

Khởi tạo biến đếm bằng 0, vì ban đầu không có đối tượng nào



```
int MyClass::count = 0;

MyClass::MyClass() {
 this->count++; // Increment the static count
}

MyClass::~MyClass() {
 this->count--; // Decrement the static count
}

void MyClass::printCount() {
 cout << "There are currently " << this->count
 << " instance(s) of MyClass.\n";
}
```



```
int main()
{
 MyClass* x = new MyClass;
 x->PrintCount();

 MyClass* y = new MyClass;
 x->PrintCount();
 y->PrintCount();

 delete x;
 y->PrintCount();
}
```

There are currently 1 instance(s) of MyClass.  
There are currently 2 instance(s) of MyClass.  
There are currently 2 instance(s) of MyClass.  
There are currently 1 instance(s) of MyClass.

# Phương thức hằng - const method



- ❖ Từ khoá **const** được dùng cho các tham số của hàm để đảm bảo các tham số được truyền cho hàm sẽ không bị hàm sửa đổi.
  - `int myFunction(const int& x);`
- ❖ Nếu có yêu cầu đặt ra là hàm không được làm thay đổi giá trị của các thuộc tính của class. Khi đó ta sẽ sử dụng một phương thức hằng bằng cách đặt từ **const** vào cuối hàm

```
class MyClass {
 ...
 void printCount() const;
 ...
};
```

```
...
void MyClass::PrintCount() const
{
 // ...
}
```

phải có từ khóa **const** ở cả khai báo  
và định nghĩa phương thức

# Ví dụ về đối tượng toàn cục



❖ Xét đoạn chương trình sau :

```
#include <iostream.h>
void main()
{
 cout << "Hello, world.\n";
}
```

Hãy sửa lại đoạn chương trình trên để có xuất liệu:

Entering a C++ program saying...

Hello, world.

And then exiting...

Yêu cầu không thay đổi hàm main() dưới bất kỳ hình thức nào.

# Ví dụ về đối tượng toàn cục



```
#include <iostream.h>
class Dummy
{
public:
 Dummy() {cout << "Entering a C++ program saying...\n";}
 ~Dummy() {cout << "And then exiting...\n";}
};
Dummy A;
void main()
{
 cout << "Hello, world.\n";
}
```

# Đối tượng là thành phần của lớp



- ❖ Đối tượng có thể là thành phần của đối tượng khác khi một đối tượng thuộc lớp “lớn” được tạo ra, các thành phần của nó cũng được tạo ra. Phương thức thiết lập (nếu có) sẽ được tự động gọi cho các đối tượng thành phần.
- ❖ Đối tượng thành phần phải được cung cấp tham số khi thiết lập thì đối tượng kết hợp (đối tượng lớn) phải có phương thức thiết lập để cung cấp tham số thiết lập cho các đối tượng thành phần.

# Đối tượng là thành phần của lớp



- ❖ Cú pháp để khởi động đối tượng thành phần là dùng dấu hai chấm (:) theo sau bởi tên thành phần và tham số khởi động.
- ❖ Khi đối tượng kết hợp bị huỷ đi thì các đối tượng thành phần của nó cũng bị huỷ đi, nghĩa là phương thức huỷ bỏ sẽ được gọi cho các đối tượng thành phần, sau khi phương thức huỷ bỏ của đối tượng kết hợp được gọi.

# Đối tượng là thành phần của lớp



```
class Diem
{
 double x,y;
public:
 Diem(double xx, double yy) {x = xx; y = yy;}
};
```

```
class TamGiac
{
 Diem A,B,C;
public:
 void Ve() const;
};
```

```
TamGiac t; // Bao sai
Diem D;
```

# Đối tượng là thành phần của lớp



```
class String
{
 char *p;
public:
 String(char *s) {p = strdup(s);}
 String(const String &s) {p = strdup(s.p);}
 ~String() {cout << "delete " << (void *)p << "\n";
 delete [] p;

 //...
 };
class SinhVien
{
 String MaSo;
 String HoTen;
 int NamSinh;
public:
 };
SinhVien s; // Bao sai
```



# Đối tượng là thành phần của lớp - thiết lập



```
class Diem
{
 double x,y;
public:
 Diem(double xx, double yy) {x = xx; y = yy;}
 // ...
};
```

# Khởi động đối tượng thành phần

## Dùng dấu hai chấm (:)



```
class TamGiac
{
 Diem A,B,C;
public:
 TamGiac(double xA, double yA, double xB, double
 yB, double xC, double yC):A(xA,yA),
 B(xB,yB),C(xC,yC){}
 void Ve() const;
 // ...
};

TamGiac t(100,100,200,400,300,300);
```

# Đối tượng là thành phần của lớp - thiết lập



```
class String
{
 char *p;
public:
 String(char *s) {p = strdup(s);}
 String(const String &s) {p = strdup(s.p);}
 ~String() {cout << "delete "<< (void *)p << "\n";
 delete [] p;
 }
 //...
};
```

# Khởi động đối tượng thành phần

## Dùng dấu hai chấm (:)



```
class SinhVien
{
 String MaSo;
 String HoTen;
 int NamSinh;
public:
 SinhVien(char *ht, char *ms, int ns):HoTen(ht),
 MaSo(ms){NamSinh = ns;}
 //...
};
SinhVien s("Nguyen Van Beo", "19920005", 1985); //Ok
```

# Khởi động đối tượng thành phần

## Dùng dấu hai chấm (:)



- ❖ Cú pháp dùng dấu hai chấm cũng được dùng cho đối tượng thành phần thuộc kiểu cơ bản.

```
class Diem {
 double x,y;
public:
 Diem(double xx, double yy):x(xx), y(yy){}
 // ...
};

class SinhVien {
 String MaSo, HoTen;
 int NamSinh;
public:
 SinhVien(char *ht, char *ms, int ns):HoTen(ht),
 MaSo(ms), NamSinh(ns){}
 //...
};
```

# Đối tượng là thành phần của mảng



- ❖ Khi một mảng được tạo ra, các phần tử của nó cũng được tạo ra, do đó phương thức thiết lập sẽ được gọi cho từng phần tử một.
- ❖ Vì không thể cung cấp tham số khởi động cho tất cả các phần tử của mảng nên khi khai báo mảng, mỗi đối tượng trong mảng phải có khả năng tự khởi động, nghĩa là có thể được thiết lập không cần tham số.
- ❖ Đối tượng có khả năng tự khởi động trong các trường hợp sau:
  - Lớp không có phương thức thiết lập.
  - Lớp có phương thức thiết lập không tham số.
  - Lớp có phương thức thiết lập mà mọi tham số đều có giá trị mặc nhiên.

# Đối tượng là thành phần của mảng



```
class Diem {
 double x,y;
public:
 Diem(double xx, double yy):x(xx), y(yy) {}
 void Set(double xx, double yy) {x = xx, y = yy;}
 // ...
};

class String {
 char *p;
public:
 String(char *s) {p = strdup(s);}
 String(const String &s) {p = strdup(s.p);}
 ~String() {
 cout << "delete " << (void *)p << "\n";
 delete [] p;
 }
 // ...
};
```

# Đối tượng là thành phần của mảng



```
class SinhVien
{
 String MaSo;
 String HoTen;
 int NamSinh;
public:
 SinhVien(char *ht, char *ms, int
 ns):HoTen(ht), MaSo(ms),
 NamSinh(ns){}

 //...
};

String as[3]; // Bao sai
Diem ad[5]; // Bao sai
SinhVien asv[7]; // Bao sai
```



# Dùng phương thức thiết lập với tham số có giá trị mặc nhiên



```
class Diem
{
 double x,y;
public:
 Diem(double xx = 0, double yy = 0):x(xx), y(yy){}
 void Set(double xx, double yy) {x = xx, y = yy;}
 // ...
};
class String
{
 char *p;
public:
 String(char *s = "") {p = strdup(s);}
 String(const String &s) {p = strdup(s.p);}
 ~String() {cout << "delete " << (void *)p << "\n";
 delete [] p;}
 // ...
```

# Dùng phương thức thiết lập với tham số có giá trị mặc nhiên



```
class SinhVien
{
 String MaSo;
 String HoTen;
 int NamSinh;
public:
 SinhVien(char *ht = "Nguyen Van A", char *ms =
 "19920014", int ns =
 1982):HoTen(ht), MaSo(ms),
 NamSinh(ns){}

 //...
};
```

```
String as[3]; // Ok: Ca ba phan tu deu la chuoi rong
Diem ad[5]; // Ok: ca 5 diem deu la (0,0)
SinhVien asv[7]; // Ok: Het sai ca 7 sinh vien deu
 co cung hoTen, maso, namsinh
```

# Dùng phương thức thiết lập không tham số



```
class Diem
{
 double x,y;
public:
 Diem(double xx, double yy):x(xx), y(yy){}
 Diem():x(0), y(0){}
 // ...
};
class String
{
 char *p;
public:
 String(char *s) {p = strdup(s);}
 String() {p = strdup("");}
 ~String() {cout << "delete " << (void *)p << "\n";
 delete [] p;}
 // ...
}
```

# Dùng phương thức thiết lập không tham số



```
class SinhVien {
 String MaSo;
 String HoTen;
 int NamSinh;
public:
 SinhVien(char *ht, char *ms, int ns):HoTen(ht),
 MaSo(ms), NamSinh(ns){}
 SinhVien():HoTen("Nguyen Van A"), MaSo("19920014"),
 NamSinh(1982){}

 //...
};

String as[3]; // Ca ba phan tu deu la chuoai rong
Diem ad[5]; // ca 5 diem deu la (0,0)
SinhVien asv[7]; // Ca 7 sinh vien deu co cung hoten,
 maso, namsinh
```

# Đối tượng được cấp phát động



- ❖ Đối tượng được cấp phát động là các đối tượng được tạo ra bằng phép toán new và bị huỷ đi bằng phép toán delete
- ❖ Phép toán new cấp đối tượng trong vùng heap (hay vùng free store) và gọi phương thức thiết lập cho đối tượng được cấp.
- ❖ Dùng new có thể cấp một đối tượng và dùng delete để huỷ một đối tượng.
- ❖ Dùng new và delete cũng có thể cấp nhiều đối tượng và huỷ nhiều đối tượng.

# Đối tượng được cấp phát động



```
class String {
 char *p;
public:
 String(char *s) {p = strdup(s);}
 String(const String &s) {p = strdup(s.p);}
 ~String() {delete [] p;}
 //...
};
class Diem {
 double x,y;
public:
 Diem(double xx, double yy):x(xx),y(yy){};
 //...
};
//...
```

# Cấp và huỷ một đối tượng



```
int *pi = new int;
int *pj = new int(15);
Diem *pd = new Diem(20,40);
String *pa = new String("Nguyen Van A");
//...
delete pa;
delete pd;
delete pj;
delete pi;
```

# Cấp và huỷ nhiều đối tượng



- ❖ Trong trường hợp cấp nhiều đối tượng, ta không thể cung cấp tham số cho từng phần tử được cấp

```
int *pai = new int[10];
```

```
Diem *pad = new Diem[5]; // Bao sai
```

```
String *pas = new String[5]; // Bao sai
```

- ❖ Thông báo lỗi cho đoạn chương trình trên như sau:

*Cannot find default constructor to initialize array element of type 'Diem'*

*Cannot find default constructor to initialize array element of type String'*

- ❖ Lỗi trên được khắc phục bằng cách cung cấp phương thức thiết lập để đối tượng có khả năng tự khởi động.



# Cấp và huỷ nhiều đối tượng



```
class String
{
 char *p;
public:
 String(char *s = "Alibaba") {p = strdup(s);}
 String(const String &s) {p = strdup(s.p);}
 ~String() {delete [] p;}
 //...
};

class Diem
{
 double x,y;
public:
 Diem(double xx, double yy):x(xx),y(yy){};
 Diem():x(0),y(0){};
 //...
};
```

# Cấp và huỷ nhiều đối tượng



- ❖ Khi đó mọi phần tử được cấp đều được khởi động với cùng giá trị.

```
int *pai = new int[10];
Diem *pad = new Diem[5];
 // ca 5 diem co cung toa do (0,0)
String *pas = new String[5];
 // Ca 5 chuoai cung duoc khoi dong bang "Alibaba"
```

- ❖ Việc huỷ nhiều đối tượng được thực hiện bằng cách dùng delete và có thêm dấu [] ở trước.

```
delete [] pas;
delete [] pad;
delete [] pai;
```

# Giao diện và chi tiết cài đặt



- ❖ Lớp có hai phần tách rời, một là phần giao diện khai báo trong phần public để người sử dụng “thấy” và sử dụng, và hai là chi tiết cài đặt bao gồm dữ liệu khai báo trong phần private của lớp và chi tiết mã hoá các hàm thành phần, vô hình đối với người dùng.
- ❖ Ta có thể thay đổi uyển chuyển chi tiết cài đặt, nghĩa là có thể thay đổi tổ chức dữ liệu của lớp, cũng như có thể thay đổi chi tiết thực hiện các hàm thành phần (do sự thay đổi tổ chức dữ liệu hoặc để cải tiến giải thuật). Nhưng nếu bảo đảm không thay đổi phần giao diện thì không ảnh hưởng đến người sử dụng, và do đó không làm đổ vỡ kiến trúc của hệ thống.
- ❖ Lớp ThoiDiem có thể được cài đặt với các thành phần dữ liệu là giờ, phút, giây hoặc tổng số giây tính từ 0 giờ.

# Lớp ThoiDiem – Cách 1



```
class ThoiDiem
{
 int gio, phut, giay;
 static bool HopLe(int g, int p, int gy);
public:
 ThoiDiem(int g = 0, int p = 0, int gy = 0) {Set(g,p,gy);}
 void Set(int g, int p, int gy);
 int LayGio() const {return gio;}
 int LayPhut() const {return phut;}
 int LayGiay() const {return giay;}
 void Nhap();
 void Xuat() const;
 void Tang();
 void Giam();
};
```

# Lớp ThoiDiem – Cách 2



```
class ThoiDiem
{
 long tsgiai;
 static bool HopLe(int g, int p, int gy);
public:
 ThoiDiem(int g = 0, int p = 0, int gy = 0) {Set(g,p,gy);}
 void Set(int g, int p, int gy);
 int LayGio() const {return tsgiai / 3600;}
 int LayPhut() const {return (tsgiai%3600)/60;}
 int LayGiay() const {return tsgiai % 60;}
 void Nhap();
 void Xuat() const;
 void Tang();
 void Giam();
};
```

