

# VIỆC KHỞI TẠO ĐỐI TƯỢNG, HÀM BẠN VÀ LỚP BẠN

**ThS. Trần Anh Dũng**

C++



Microsoft®

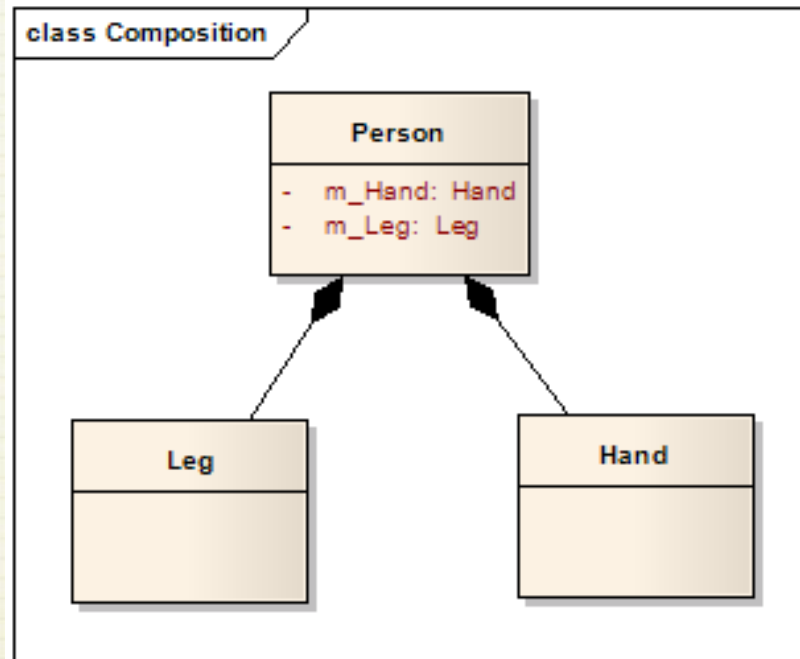
**Visual Studio®**

# Nội dung

- ❖ Đối tượng là thành phần của lớp
- ❖ Đối tượng là thành phần của mảng
- ❖ Đối tượng được cấp phát động
- ❖ Hàm bạn
- ❖ Lớp bạn
- ❖ Các nguyên tắc xây dựng lớp

# 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.





# Đối tượng là thành phần của lớp

- ❖ Phương thức thiết lập (nếu có) sẽ được tự động gọi cho các đối tượng thành phần.
- ❖ Khi đối tượng kết hợp bị hủy → đối tượng thành phần của nó cũng bị hủy, nghĩa là phương thức hủy bỏ sẽ được gọi cho các đối tượng thành phần, sau khi phương thức hủy bỏ của đối tượng kết hợp được gọi.

# Đối tượng là thành phần của lớp

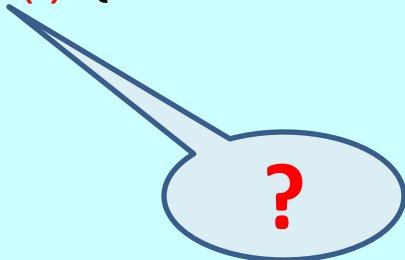
- ❖ Nếu đối tượng thành phần phải cung cấp tham số khi thiết lập thì đối tượng kết hợp (đối tượng lớn) **phải có phương thức thiết lập** để cung cấp tham số thiết lập cho các đối tượng thành phần.
- ❖ Cú pháp để khởi động đối tượng thành phần là **dùng dấu hai chấm (:)** theo sau bởi tên thành phần và tham số khởi động.

# Ví dụ

```
class TamGiac{  
    Diem A, B, C;  
public:  
    TamGiac(double xA, double yA, double xB, double yB,  
double xC, double yC){  
    }  
    void Ve();  
    // ...  
};  
  
TamGiac t(100,100,200,400,300,300);
```

# Ví dụ

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

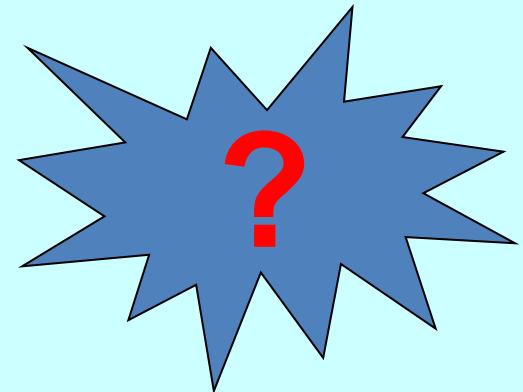


Cú pháp dấu hai chấm  
cũng được dùng cho đối  
tượng thành phần thuộc  
kiểu cơ sở

```
TamGiac t (100, 100, 200, 400, 300, 300, 1);
```

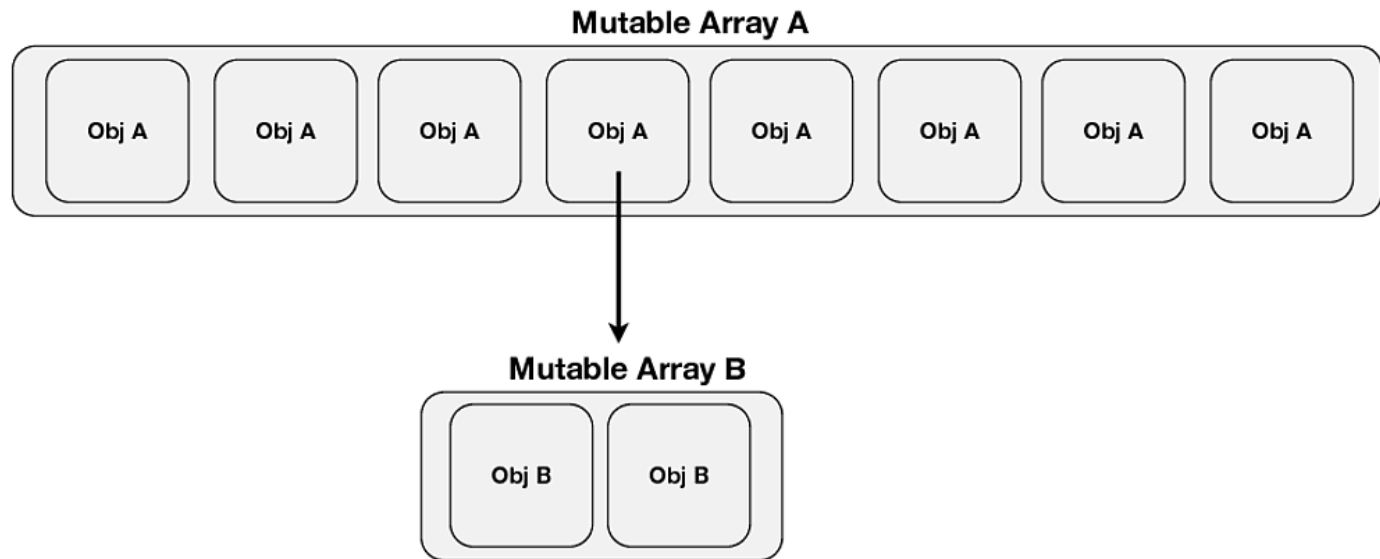
# Ví dụ

```
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;  
    }  
};
```





# Đối tượng là thành phần của mảng



# Đối tượng là thành phần của mảng

- ❖ Khi một mảng được tạo ra → các phần tử của nó cũng được tạo ra → phương thức thiết lập sẽ được gọi cho từng phần tử.
- ❖ Vì không thể cung cấp tham số khởi động cho tất cả các phần tử của mảng → khi khai báo mảng, mỗi đối tượng trong mảng phải có **khả năng tự khởi động**, nghĩa là có thể thiết lập không cần tham số.

# Đối tượng là thành phần của mảng

❖ Đối tượng có khả năng tự khởi động trong những trường hợp nào?

1. Lớp không có phương thức thiết lập
2. Lớp có phương thức thiết lập không tham số
3. 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;
        }
        // ...
};
```



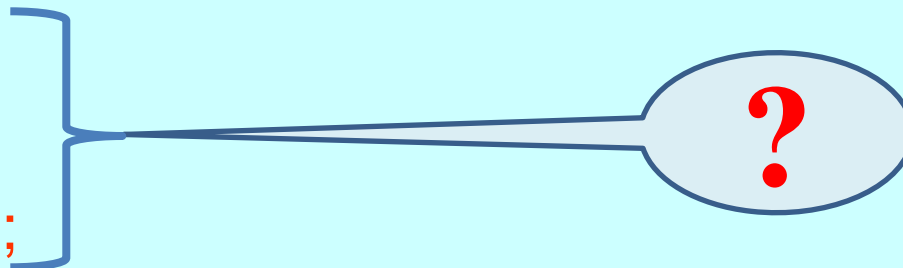
# Đối tượng là thành phần của mảng

```
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 arrs[3];  
Diem arrd[5];  
SinhVien arrsv[7];
```



# 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;
    }
    // ...
};
```

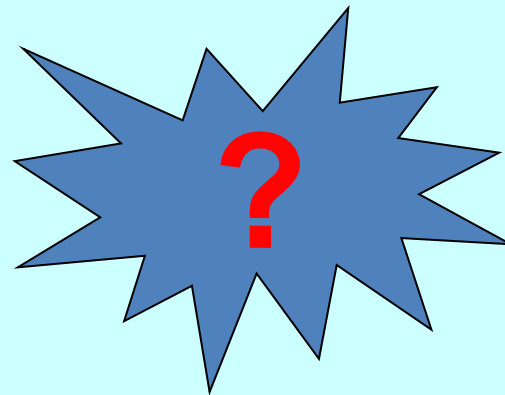
# Dùng phương thức thiết lập với tham số có giá trị mặc nhiên

```
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, 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];  
Diem ad[5];  
SinhVien asv[7];
```



# 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)
    {}
    // ...
};
```

# Dùng phương thức thiết lập không tham số

```
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, 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];  
Diem ad[5];  
SinhVien asv[7];
```





# 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 hủy đi bằng phép toán **delete**
- ❖ Phép toán **new** cấp đối tượng trong vùng heap và gọi phương thức thiết lập cho đối tượng được cấp.

# 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 phát và hủy 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;
```

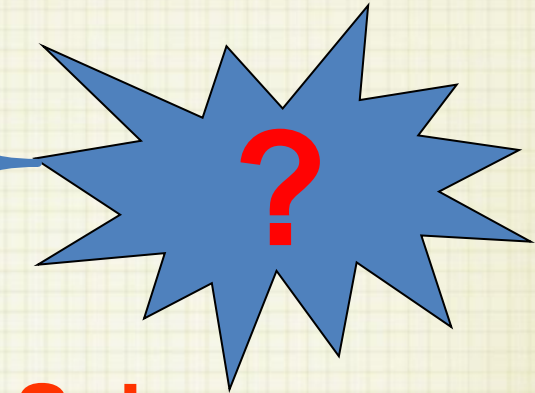
```
delete pi;
```

# Cấp phát và hủy nhiều đối tượng

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

```
Diem *pad = new Diem[5];
```

```
String *pas = new String[5];
```



**Sai**

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



# Cấp và hủy nhiều đối tượng

❖ 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'*

❖ Khắc phục lỗi?

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

# Cấp và hủy 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 hủy 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 chuai cung duoc khai dong la "Alibaba"
```



# Cấp và hủy nhiều đối tượng

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

```
delete [] pas;
```

```
delete [] pad;
```

```
delete [] pai;
```

- ❖ Có thể thay ba phát biểu trên bằng một phát biểu duy nhất sau hay không?

```
delete pas,pad,pai;
```

# Hàm bạn, lớp bạn

- ❖ Giả sử có lớp Vector, lớp Matrix
  - ❖ Cần viết hàm nhân Vector với một 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. Tuy nhiên, 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 lớp nếu điều đó không được dự trù trước trong khai báo của lớp.



# Hàm bạn (Friend function)

❖ Các tính chất của quan hệ friend:

- Phải được cho, không được nhận
  - Lớp B là bạn của lớp A, lớp A phải khai báo rõ ràng B là bạn của nó
- Không đối xứng
- Không bắc cầu

# Ví dụ

```
class COUNTERCLASS{  
    int Counter;  
public:  
    char CounterChar;  
    void Init( char );  
    void AddOne( ){  
        Counter++;  
    }  
    friend int Total (int);  
};
```

# Ví dụ

```
COUNTERCLASS MyCounter[26];  
int Total(int NumberObjects)  
{  
    for (int i=0, sum=0; i<NumberObjects; i++)  
        sum += MyCounter[i].Counter  
        //Tính tổng số ký tự trong số các Objects ký tự  
    return sum;  
}
```

# Lớp bạn (Friend class)

- ❖ Một lớp **có thể** truy cập đến các thành phần có thuộc tính **private** của một lớp khác.
- ❖ Để thực hiện được điều này, chúng ta có thể lấy toàn bộ một lớp làm bạn (hàm friend) cho lớp khác.



# Ví dụ

```
class TOM{
public:
    friend class JERRY;    //Có lớp bạn là JERRY
private:
    int SecretTom;        //Bí mật của TOM
};

class JERRY{
public:
    void Change(TOM T){
        T.SecterTom++;    //Bạn nên có thể thao thế
    }
};
```



# Giao diện và chi tiết cài đặt

- ❖ Lớp có hai phần tách rời
  - ❖ Phần giao diện khai báo trong phần **public** để người sử dụng “thấy” và sử dụng.
  - ❖ Chi tiết cài đặt bao gồm dữ liệu khai báo trong phần **private** của lớp và chi tiết mã hóa các hàm thành phần, vô hình đối với người dùng.
- ❖ 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.



# Giao diện và chi tiết cài đặt

- ❖ 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 – 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

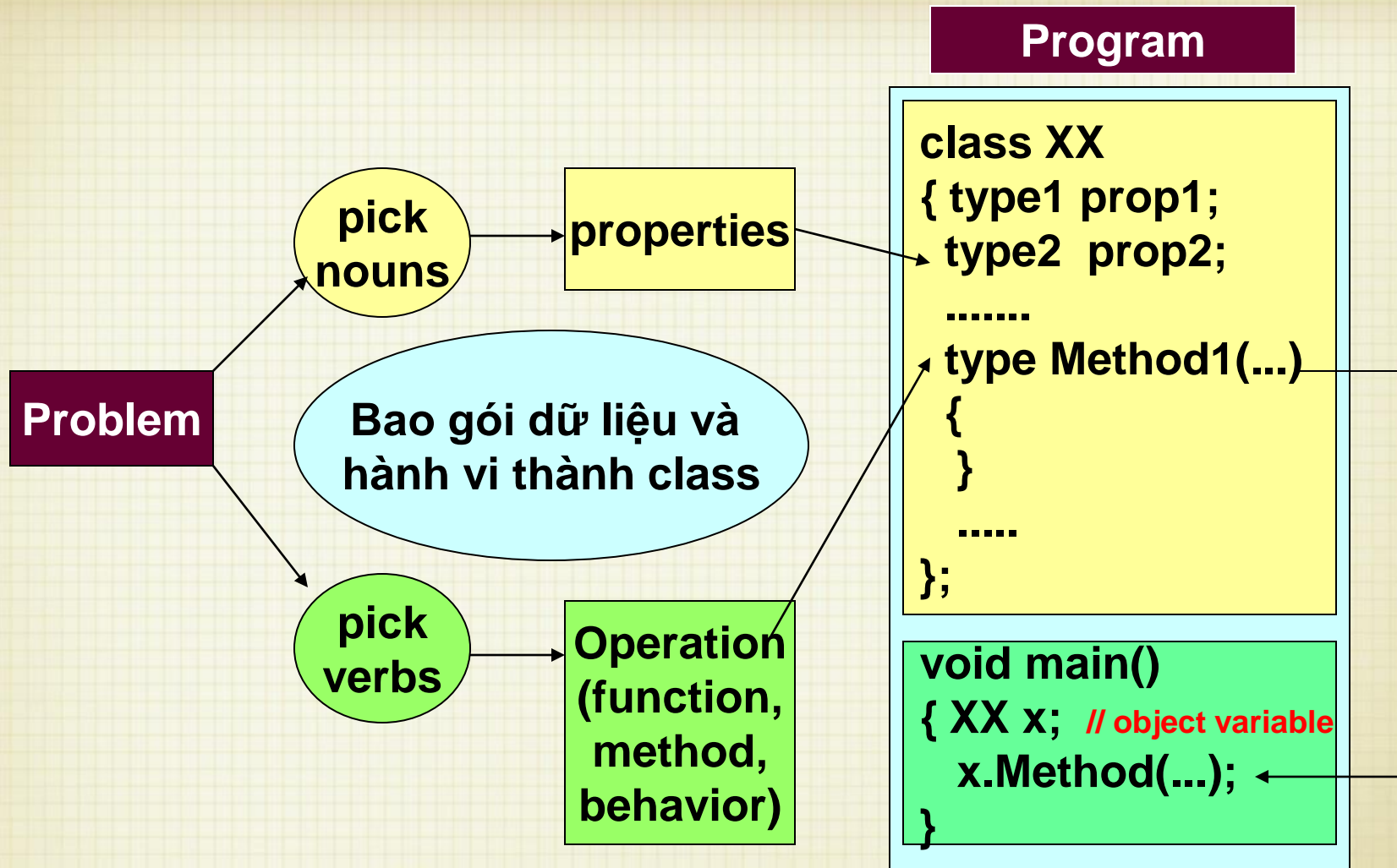
```
class ThoiDiem{
    long tsgiai;
    static bool HopLe(int g, int p, int gy);
public:
    ThoiDiem(int g = 0, int p = 0, int gy = 0) {Set(g,p,gy);}
    void Set(int g, int p, int gy);
    int LayGio() const {return tsgiai/3600;}
    int LayPhut() const {return (tsgiai%3600)/60;}
    int LayGiay()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

# Các nguyên tắc xây dựng lớp

- ❖ **Hình thành lớp:** Khi ta nghĩ đến “nó” như một khái niệm riêng lẻ → Xây dựng lớp biểu diễn khái niệm đó.
- ❖ **Lớp** là biểu diễn cụ thể của một khái niệm vì vậy tên lớp luôn là danh từ.
- ❖ **Các thuộc tính** của lớp là các thành phần dữ liệu nên chúng **luôn là danh từ**.
- ❖ **Các hàm thành phần** (các hành vi) là các thao tác chỉ rõ hoạt động của lớp nên **các hàm là động từ**.



# Các nguyên tắc xây dựng lớp



# Các nguyên tắc xây dựng lớp

- ❖ Các thuộc tính **có thể suy diễn từ những thuộc tính khác** thì nên dùng hàm thành phần để thực hiện tính toán.

```
class TamGiac{  
    Diem A,B,C;  
    double ChuVi;  
    double DienTich;  
public:  
    //...  
};
```

```
class TamGiac{  
    Diem A,B,C;  
public:  
    //...  
    double ChuVi() const;  
    double DienTich() const;  
};
```



# Các nguyên tắc xây dựng lớp

- ❖ Tuy nhiên, nếu các thuộc tính suy diễn đòi hỏi nhiều tài nguyên hoặc thời gian để thực hiện tính toán, ta nên khai báo là dữ liệu thành phần.

```
class QuocGia{  
    long DanSo;  
    double DienTich;  
    double TuổiTrungBinh;  
public:  
    double TinhTuoiTB() const;  
    //...  
};
```

# Các nguyên tắc xây dựng lớp

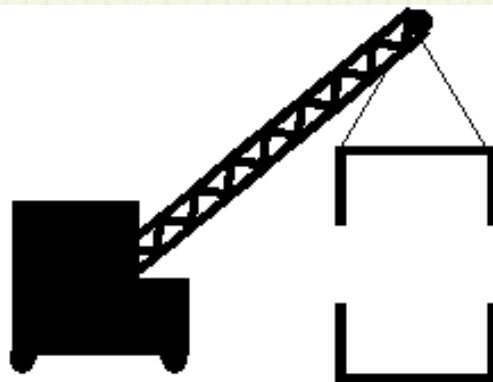
❖ Dữ liệu thành phần nên được kết hợp:

```
class TamGiac{  
    Diem A,B,C;  
public:  
    //...  
};  
class HìnhTron{  
    Diem Tam;  
    double BanKinh;  
public:  
    //...  
};
```

```
class TamGiac{  
    double xA, yA;  
    double xB, yB, xC, yC;  
public:  
    //...  
};  
class HìnhTron{  
    double tx, ty, BanKinh;  
public:  
    //...  
};
```

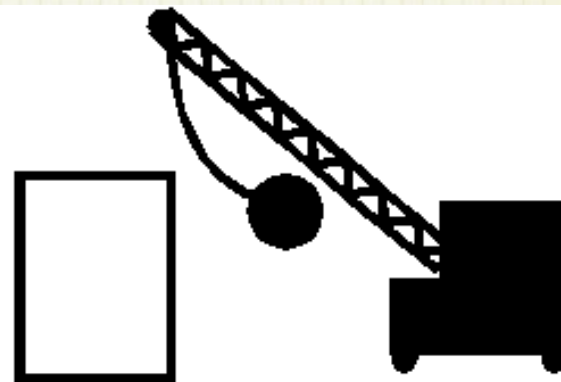
# Các nguyên tắc xây dựng lớp

- ❖ Trong mọi trường hợp, nên có phương thức thiết lập (Constructor) để khởi động đối tượng.
- ❖ Nên có phương thức thiết lập có khả năng tự khởi động không cần tham số



Constructor

```
MyClass *MyObjPtr = new MyClass();
```



Destructor

```
delete MyObjPtr;
```

# Các nguyên tắc xây dựng lớp

- ❖ Nếu đối tượng có nhu cầu cấp phát tài nguyên thì phải có phương thức thiết lập, copy constructor để khởi động đối tượng bằng đối tượng cùng kiểu và có destructor để dọn dẹp. Ngoài ra còn có phép gán (chương 5).
- ❖ Nếu đối tượng đơn giản không cần tài nguyên riêng → Không cần copy constructor và destructor



# Bài tập

- ❖ Viết chương trình cho phép **nhập, xuất, khởi tạo** 1 học sinh. Thông tin cần quan tâm về 1 học sinh:  
Mã học sinh (8 ký tự), họ tên học sinh (30 ký tự), điểm toán (int), điểm văn (int).
- ❖ Danh từ: Học sinh → cấu trúc HS
- ❖ Động từ:
  - Nhập một hs → Hàm Nhap()
  - Xuất một hs → Hàm Xuat();



# Q & A

