

# CHƯƠNG 3. LỚP VÀ ĐỐI TƯỢNG (tt)

*Khoa Công Nghệ Phần Mềm*





# 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( );
    // ...
};

TamGiac t;           //Error ?
Diem d;              //Error ?
```



# 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.
- ❖ 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) : A(xA,yA), B(xB,yB),C(xC,yC){
        cout<<"Khoi tao tam giac";
    }
    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();  
    // ...  
};  
TamGiac t (100, 100, 200, 400, 300, 300, 1);
```

?

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



# Ví dụ

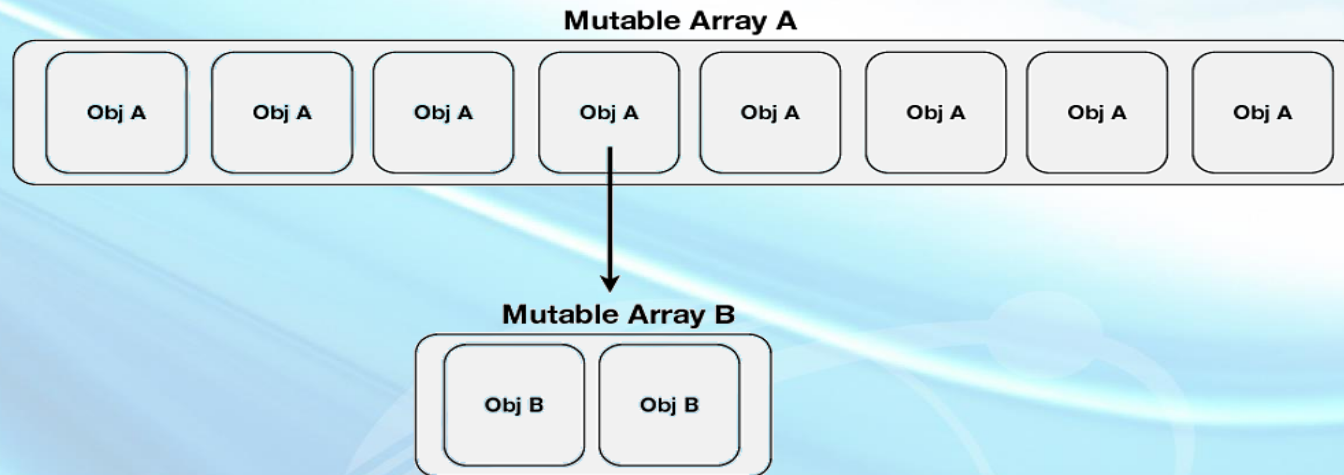
```
class Diem{  
    double x,y;  
public:  
    Diem(double xx = 0, double yy = 0) : x(xx), y(yy){ }  
  
    void Set(double xx, double yy){  
        x = xx;  
        y = yy;  
    }  
};
```







# Đối tượng là thành phần của mảng



- ❖ 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 phương thức thiết lập không cần tham số.





# Đối tượng là thành phần của mạng

❖ 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 arrString[3];    // err  
Diem arrDiem[5];        // err  
SinhVien arrSV[7];      // err
```








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

A red line with two arrows pointing downwards from the default values '0' in the constructor 'Diem(double xx = 0, double yy = 0)' to the parameters 'xx' and 'yy' in the 'Set' method, illustrating how the default values are used as arguments for the 'Set' method.



# 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;  
    }  
};
```

A red arrow points from the top right towards the default parameter 's = ""' in the first constructor of the String class.



# 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 arrString[3];  
Diem arrDiem[5];  
SinhVien arrSV[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 arrString[3];  
Diem arrDiem[5];  
SinhVien arrSV[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];
```



- ❖ 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 chuoai 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, [] pad, [] pai;

nên là:

**delete** [] pas;

**delete** [] pad;

**delete** [] pai;



err?



# Bài tập





# Hàm bạn, lớp bạn

- ❖ Giả sử có lớp Vector, lớp Matrix
- ❖ Cần viết **hàm nhân** Vector với một Matrix
- ❖ **Hàm nhân** này:
  - Không thể thuộc lớp Vector
  - Không thể thuộc lớp Matrix
  - Không thể là hàm tự do (*vì hàm không thuộc lớp sẽ không truy cập được thuộc tính private*)

**→Giải pháp: Xây dựng hàm truy cập dữ liệu?**





# Hàm bạn (Friend function)

- ❖ **Hàm bạn** của một lớp là hàm không phải là hàm thành phần của lớp, nhưng có khả năng truy xuất đến mọi thành phần của đối tượng (kể cả private, protected);
- ❖ **Cú pháp:** **friend** <kiểu trả về> <tên hàm>(tham số);
- ❖ Sau đó định nghĩa hàm ở ngoài lớp như các hàm tự do khác, có thể khai báo một hay nhiều hàm “bạn”
- ❖ **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)

- ❖ Đây là cách cho phép chia sẻ dữ liệu giữa các đối tượng với một hàm tùy ý trong chương trình (**hàm friend**) hoặc chia sẻ các thành phần của đối tượng có thuộc tính **private** hay **protected** với các đối tượng khác (**lớp friend**).
- ❖ Hàm bạn không phải là hàm thành viên nên không bị ảnh hưởng của từ khoá truy xuất
- ❖ Hàm bạn của một lớp có thể là hàm tự do, hàm thành phần của một lớp khác



# Ví dụ 1

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



# Ví dụ 1

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

Thuộc tính của CounterClass





# Hàm bạn (Friend function)

## ❖ 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 thể có giá trị trả về thuộc kiểu lớp đó.



# Ví dụ 2

```
#include <iostream>
#include <conio.h>
class Beta;
class Alpha
{
    private:
        float X;
    public:
        Alpha()
        { X = 5.0; }
    friend float Init(Alpha,Beta);
};
class Beta
{
    private:
        float Y;
    public:
        Beta()
        { Y = 1.0; }
    friend float Init(Alpha,Beta);
};
```



## Ví dụ 2

```
float Init(Alpha A,Beta B) // như một hàm toàn cục
{
    return A.X + B.Y;
}
int main()
{
    Alpha A;
    Beta B;
    cout << Init(A,B) << " là kết quả của hai lớp\n";
    return 0;
}
```

### Kết quả:

6 là kết quả của hai lớp



# Ví dụ 3

```
#include <iostream.h>
#include <conio.h>
class sophuc
{ float a,b;
  public : sophuc() {}
          sophuc(float x, float y)
          {a=x; b=y;}
  friend sophuc tong(sophuc,sophuc);
  friend void hienthi(sophuc);
};
sophuc tong(sophuc c1,sophuc c2)
{sophuc c3;
  c3.a=c1.a + c2.a ;
  c3.b=c1.b + c2.b ;
  return (c3);
}
```





## Ví dụ 3

```
void hienthi(sophuc c)
{cout<<c.a<<" + "<<c.b<<"i"<<endl; }
void main()
{ sophuc d1 (2.1,3.4);
  sophuc d2 (1.2,2.3) ;
  sophuc d3 ;
  d3 = tong(d1,d2);
  cout<<"d1= "; hienthi(d1);
  cout<<"d2= "; hienthi(d2);
  cout<<"d3= "; hienthi(d3);
}
```

### Kết quả::

d1= 2.1 + 3.4i

d2= 1.2 + 2.3i

d3= 3.3 + 5.7i



## Ví dụ 4

```
#include <iostream.h>
#include <conio.h>
class LOP1;
class LOP2
{ int v2;
  public:
    void nhap(int a)
    { v2=a;}
    void hienthi(void)
    { cout<<v2<<"\n"; }
  friend void traodoi(LOP1 &, LOP2 &);
};
```



# Ví dụ 4

```
class LOP1
{ int v1;
  public:
    void nhap(int a)
    { v1=a;}
    void hienthi(void)
    { cout<<v1<<"\n";}
    friend void traodoi(LOP1 &, LOP2 &);
};

void traodoi(LOP1 &x, LOP2 &y) // như một hàm toàn cục
{ int t = x.v1;
  x.v1 = y.v2;
  y.v2 = t;
}
```



# Ví dụ 4

```
int main()
{ LOP1 ob1;
  LOP2 ob2;
  ob1.nhap(150);
  ob2.nhap(200);
  cout << "Gia tri ban dau :" << "\n";
  ob1.hienthi();
  ob2.hienthi();
  traodoi(ob1, ob2); //Thuc hien hoan doi
  cout << "Gia tri sau khi thay doi:" << "\n";
  ob1.hienthi();
  ob2.hienthi();
  return 0;
}
```

## Kết quả:

Gia tri ban dau :

150

200

Gia tri sau khi thay doi:

200

150





# 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 (**lớp friend**) cho lớp khác.



# Lớp bạn (Friend class)

- ❖ 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 có tính đối xứng,
  - Không có tính bắc cầu.
- Quan hệ friend có vẻ như vi phạm khái niệm đóng gói (*encapsulation*) của OOP nhưng có khi lại cần đến nó để cài đặt các mối quan hệ giữa các lớp và khả năng **đa năng hóa toán tử** trên lớp (*sẽ đề cập ở chương sau*)



# Ví dụ 1

```
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.SecretTom++; //Jerry là bạn nên có thể truy cập  
    }  
};
```



## Ví dụ 2

```
#include <iostream>
class CSquare;
class CRectangle
{   int width, height;
    public:   int area (void)
              {return (width * height);}
              void convert (CSquare a);
};
class Csquare
{   private:
        int side;
    public:
        void set_side (int a)
        {side=a;}
    friend class CRectangle;
};
```





```
void CRectangle::convert (CSquare a)
{
    width = a.side;
    height = a.side;
}
int main ()
{
    CSquare sqr;
    CRectangle rect;
    sqr.set_side(4);
    rect.convert(sqr);
    cout << rect.area();
    return 0;
}
```

**Kết quả: 16**



# 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 KiemTraHopLe(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); //cập nhật  
    int GetGio() const {return gio; } //lấy giờ  
    int GetPhut() const {return phut; } //lấy phút  
    int GetGiay() const {return giay; } //lấy giây  
    void Nhap();  
    void Xuat() const;  
    void Tang();  
    void Giam();  
};
```





# Lớp ThoiDiem – Cách 2

```
class ThoiDiem{
    long tsgiai;
    static bool KiemTraHopLe(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 GetGio() const {return tsgiai/3600;} //lấy tổng số tính bằng giây
    int GetPhut() const {return (tsgiai%3600)/60;}
    int GetGiay() const {return tsgiai%60;}
    void Nhap();
    void Xuat() const;
    void Tang();
    void Giam();
};
```

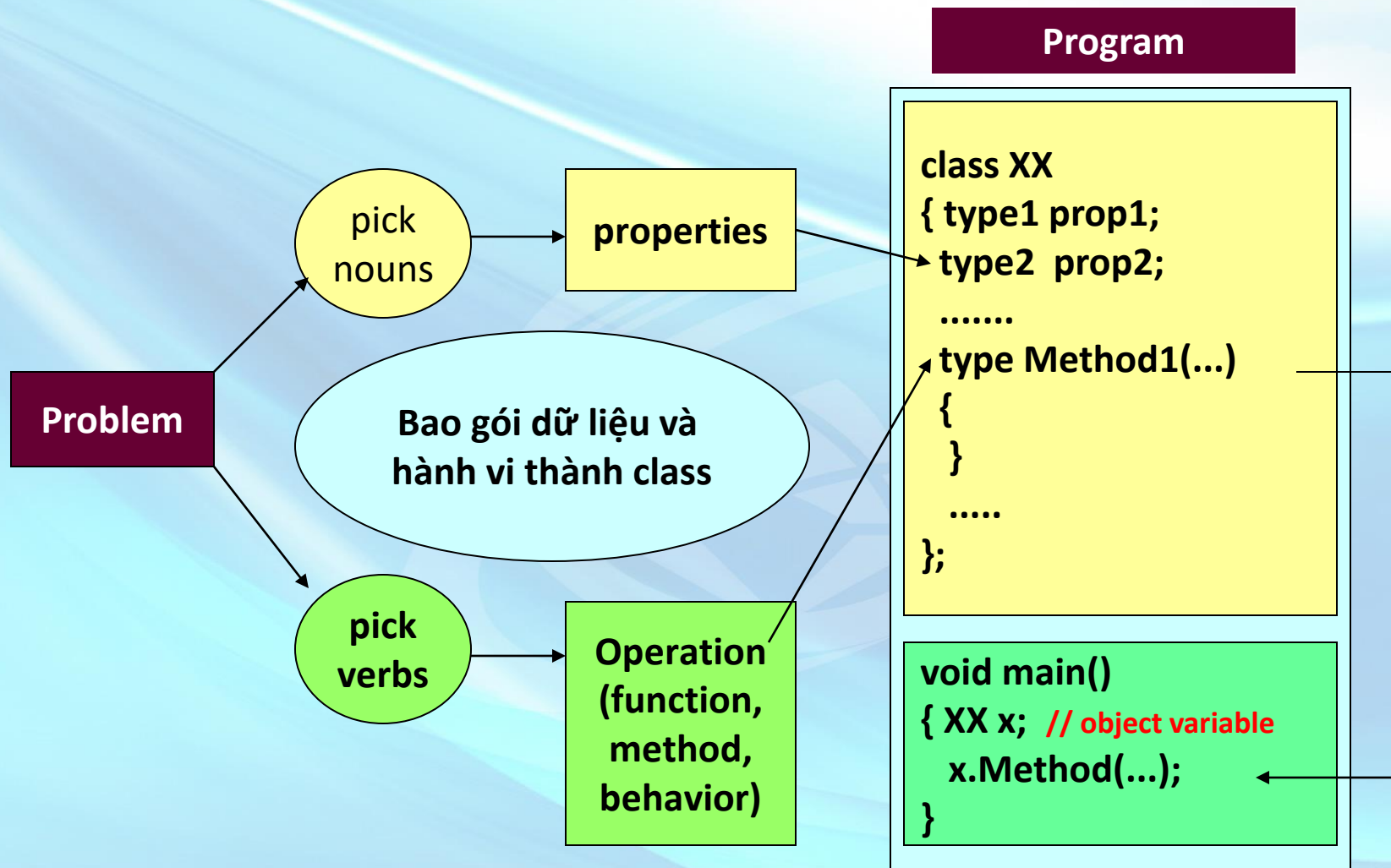


# Các nguyên tắc xây dựng lớp

- ❖ **Xây dựng 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 dùng hàm thành phần để thực hiện tính toán. Ví dụ: chu vi, diện tích của một tam giác

```
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 có thể 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. (**default constructor**)
- ❖ Nếu đối tượng có nhu cầu cấp phát tài nguyên thì phải có **phương thức thiết lập**, **phương thức sao chép** để khởi động đối tượng bằng đối tượng cùng kiểu và có **phương thức hủy** để dọn dẹp; Ngoài ra, còn có **phép gán** (chương 4).
- ❖ Nếu đối tượng đơn giản không cần tài nguyên riêng → Không cần copy constructor và destructor



# Bài tập

## 1. Xây dựng các lớp sau

- Ma trận các số nguyên
- Đơn thức
- Đa thức
- Yêu cầu:
  - Các thuộc tính, phương thức cần thiết (nhập, xuất, cộng, trừ, nhân,...)
  - Các phương thức khởi tạo (cả khởi tạo sao chép), hủy cần thiết





# BÀI TẬP

2. Xây dựng lớp Vector (N số thực) và các phương khởi tạo, hủy (nếu có) thức nhập, xuất, cộng, trừ, tịnh tiến, tính **norm** (*chuẩn*) của vector. Viết chương trình quản lý một danh sách các vector và thực hiện các yêu cầu:

- Nhập N vector và xuất ra thông tin các Vector vừa nhập
- Xuất các vector có norm tăng dần
- Tịnh tiến các vector theo vector là chính nó
- Tính tổng và hiệu các vector với chính nó
- **Ghi chú:**

- $V(x_1, x_2, \dots, x_n), \text{norm}(V) = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$



# BÀI TẬP

3. Xây dựng lớp TamGiac có thành phần là 3 Diem với các phương khởi tạo, hủy (nếu có) thức nhập, xuất, tính chu vi, tính diện tích, tịnh tiến. Viết chương trình quản lý một danh sách các TamGiac và thực hiện các yêu cầu:

- Nhập vào N (do người dùng nhập) tam giác và xuất ra thông tin (3 điểm, chu vi, diện tích) các tam giác vừa nhập
- Xuất ra tam giác có chu vi lớn nhất
- Xuất ra tam giác có diện tích nhỏ nhất
- Xuất ra thông tin (tọa độ các điểm) các tam giác sau khi tịnh tiến theo Vector  $V(a, b)$  (do người dùng nhập vào)



# Q & A

