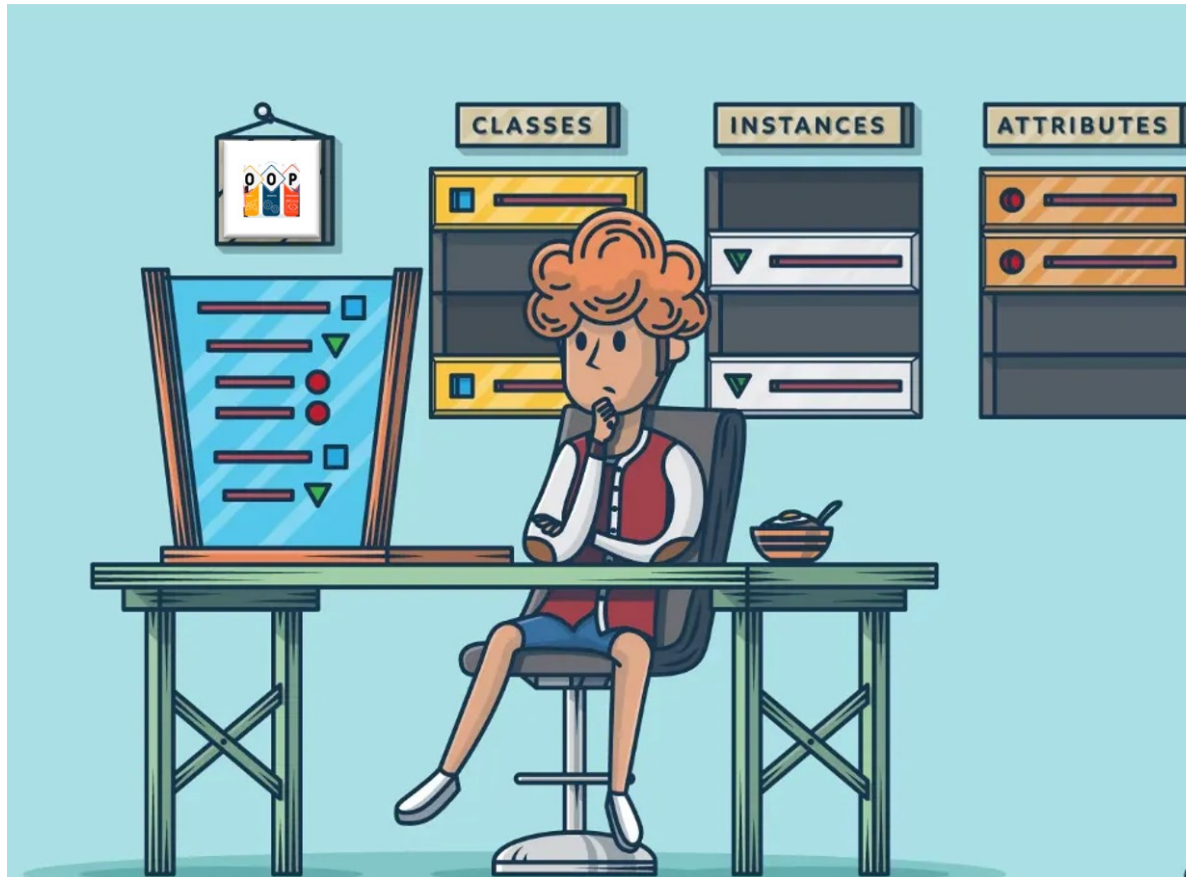




## LỚP VÀ ĐỐI TƯỢNG – MỘT SỐ VẤN ĐỀ LIÊN QUAN



# Nội dung

**1**

**Đối tượng là thành phần của lớp**

**2**

**Đối tượng là thành phần của mảng**

**3**

**Đối tượng được cấp phát động**

**4**

**Giao diện và chi tiết cài đặt**

**5**

**Các nguyên tắc xây dựng lớp**

# 1. 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” (đối tượng kết hợp) được tạo ra, các đối tượng thành phần của nó cũng được tạo ra.
- Khi đối tượng kết hợp bị hủy → các 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.

# 1. Đối tượng là thành phần của lớp (tt)

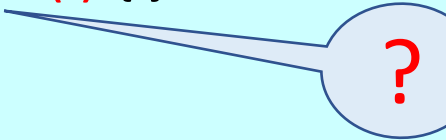
- ❖ Nếu phương thức thiết lập của đối tượng thành phần cần cung cấp tham số thì đối tượng kết hợp phải có phương thức thiết lập để cung cấp tham số cho các phương thức thiết lập này.
- ❖ Để cung cấp tham số cho các phương thức thiết lập của đối tượng thành phần ta dùng dấu hai chấm (:) tiếp theo là tên đối tượng thành phần và các tham số khởi tạo được truyền vào.

# 1. Đối tượng là thành phần của lớp – Ví dụ

```
class C{  
  
private:  
  
    int m,n;  
  
    A u,v;  
  
    B p,q,r;  
  
public:  
  
    C(int m1, int n1, int a1, int b1, double x1, double y1,  
      double x2, double y2, double z2):  
    u(),v(a1,b1),q(x1,y1),r(x2,y2,z2){ //u() hoặc không cần liệt kê  
  
        m = m1; n = n1;}  
  
}; //end of class
```

# 1. Đối tượng là thành phần của lớp – Ví dụ (tt)

```
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();  
}; //end of class  
TamGiac t(100, 100, 200, 400, 300, 300, 1);
```



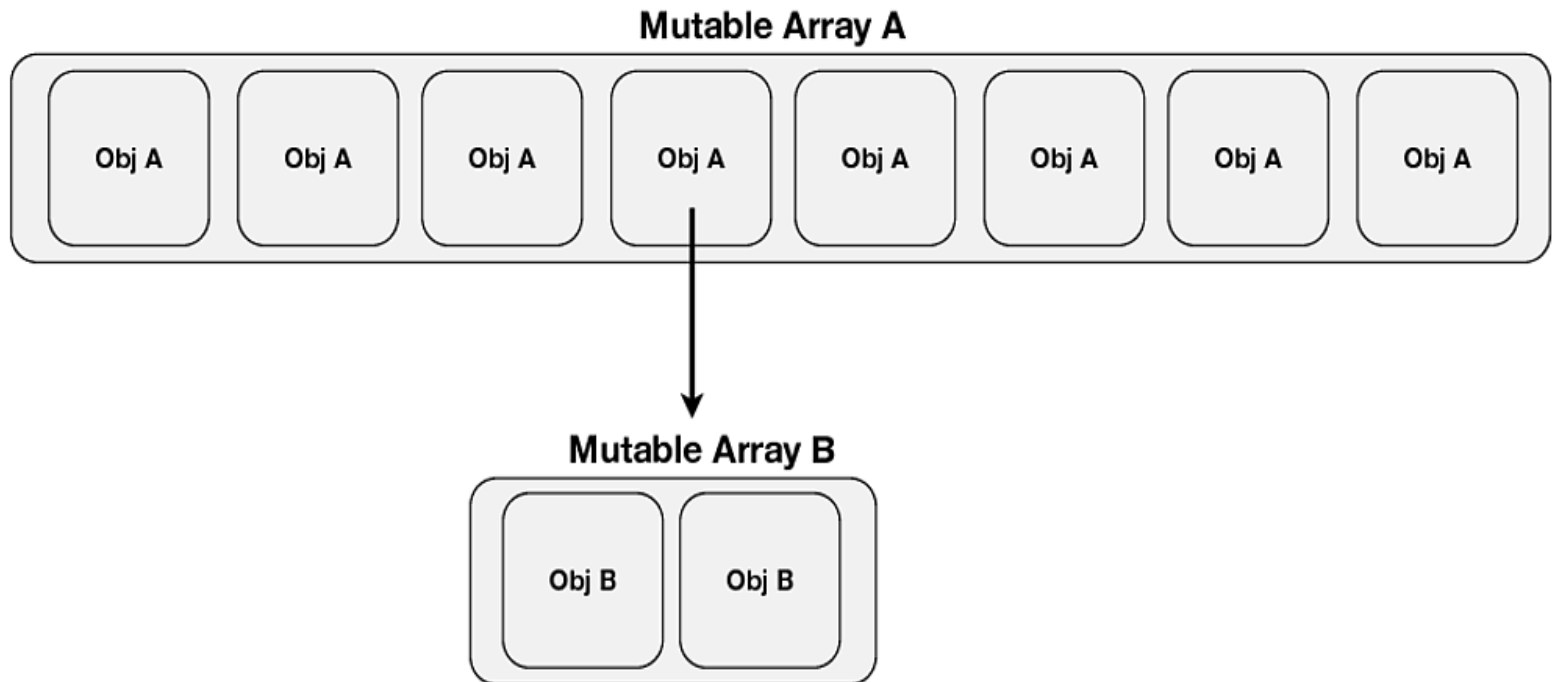
# 1. Đối tượng là thành phần của lớp – Ví dụ (tt)

```
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;  
    }  
}; //end of class
```





## 2. Đối tượng là thành phần của mảng



## 2. Đối tượng là thành phần của mạng (tt)

- ❖ Khi một mạng 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 tạo cho tất cả phần tử của mạng → khi khai báo mạng, mỗi đối tượng trong mạng phải có khả năng tự khởi động.

## 2. Đối tượng là thành phần của mảng (tt)

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 đối số (hàm tạo mặc định).
3. Lớp có phương thức thiết lập mà mọi đối số đều có giá trị mặc nhiên.

## 2. Đối tượng là thành phần của mảng (tt)

```
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;
        }
}; //end of class
```

## 2. Đối tượng là thành phần của mảng (tt)

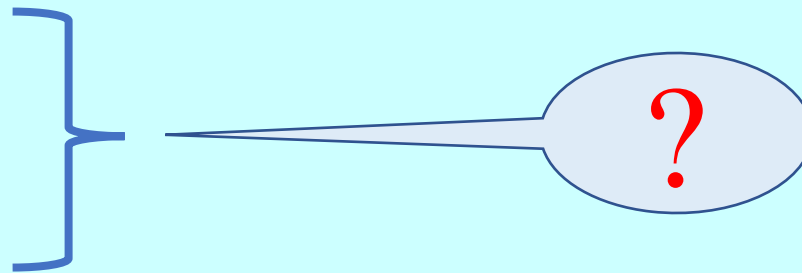
```
#include "iostream"
#include "string.h"
class String {
    char *p;
public:
    String(char *s) { p = _strdup(s); } //StringDuplicate
    String(const String &s) { p = _strdup(s.p); } //Hàm tạo sao chép
    ~String() {
        cout << "delete " << (void *) p << "\n";
        delete [] p;
    }
}; //end of class

delete is used to de-allocate memory allocated for single object.
delete [] is used to de-allocate memory allocated for array of objects.
```

## 2. Đối tượng là thành phần của mảng (tt)

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

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



# Phương thức thiết lập với đối có giá trị mặc nhiên – 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;
    }
}; //end of class
```

# Phương thức thiết lập với đối có giá trị mặc nhiên – Ví dụ (tt)


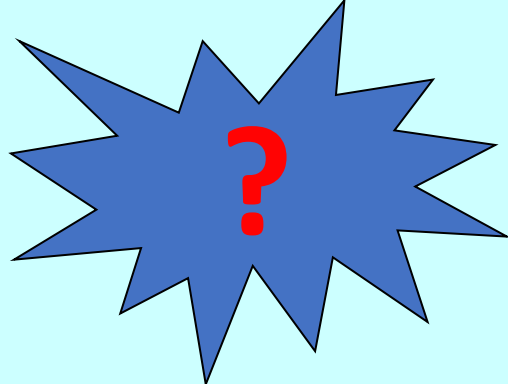
```
#include "iostream"
#include "string.h"

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;
    }
}; //end of class
```



# Phương thức thiết lập với đối có giá trị mặc nhiên – Ví dụ (tt)

```
class SinhVien{  
    String MaSo, HoTen;  
    int NamSinh;  
public:  
    SinhVien(char *ms="19920014", char *ht="Nguyen Van A", int  
        ns = 1982) : MaSo(ms), HoTen(ht), NamSinh(ns) { }  
};  
Diem arrd[5];  
String arrs[3];  
SinhVien arrsv[7];
```



# Phương thức thiết lập không đối số – Ví dụ

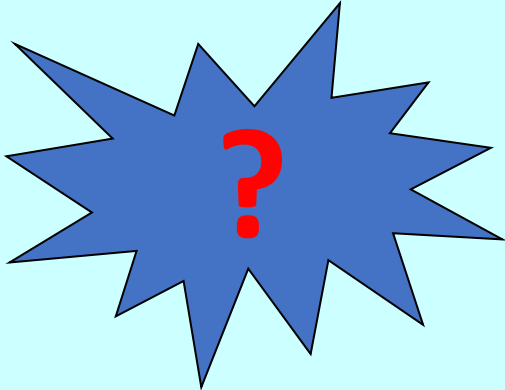
```
class Diem
{
    double x,y;
public:
    Diem(double xx, double yy) : x(xx), y(yy){ }
    Diem() : x(0), y(0){ }
}; //end of class
```

# Phương thức thiết lập không đối số – Ví dụ (tt)

```
#include "iostream"
#include "string.h"
class String{
    char *p;
public:
    String(char *s) { p = _strdup(s); }
    String() { p = _strdup(""); }
    ~String() {
        cout << "delete " << (void *) p << "\n";
        delete [] p;
    }
}; //end of class
```

# Phương thức thiết lập không đối số – Ví dụ (tt)

```
class SinhVien {  
    String MaSo, HoTen;  
    int NamSinh;  
public:  
    SinhVien(char *ms, char *ht, int ns) : HoTen(ht), MaSo(ms),  
        NamSinh(ns) { }  
    SinhVien() : MaSo("19920014"), HoTen("Nguyen Van A"),  
        NamSinh(1982) { }  
}; //end of class  
Diem arrd[5];  
String arrs[3];  
SinhVien arrsv[7];
```



### 3. Đối tượng được cấp phát động

#### Cấp phát tĩnh và cấp phát động:

- ❖ **Cấp phát tĩnh:** kích thước bộ nhớ cần cấp phát phải được xác định trước khi chương trình được biên dịch.
- ❖ **Cấp phát động:** kích thước bộ nhớ cần cấp phát được xác định khi chương trình thực thi và cấp phát trong quá trình thực thi.

### 3. Đối tượng được cấp phát động (tt)

#### Cấp phát động:

`int *a = new int; -> delete a;`

`int *b = new int[10]; -> delete [ ] b;`

=> Cấp phát ***một vùng nhớ liên tục*** có kích thước là:

***Số lượng phần tử \* sizeof(kiểu dữ liệu) = 10 \* 4 = 40***

=> Lưu ***địa chỉ đầu*** của vùng nhớ này vào biến con trỏ.

### 3. Đối tượng được cấp phát động (tt)

#### Lợi ích của cấp phát động:

- ❖ Có thể cấp phát vùng nhớ ***có kích thước bất kỳ*** bằng cách truyền tham số vào trong cặp dấu **[ ]** khi cấp phát vùng nhớ (cấp phát tĩnh không làm được việc này).
- ❖ Có thể ***sử dụng lại*** vùng nhớ đã được giải phóng bằng câu lệnh **delete**

### 3. Đối tượng được cấp phát động (tt)

#### Lợi ích của cấp phát động (tt):

**Ví dụ:** để tiết kiệm bộ nhớ ta dùng con trỏ và cấp phát bộ nhớ cho các đối tượng thay cho việc dùng mảng đối tượng.

```
TS ts[100]; //dùng mảng đối tượng
```

-> TS \*ts;

```
ts = new TS[số_thí_sinh];
```

*//số\_thí\_sinh được nhập vào*

Sau đó có thể dùng tên con trỏ giống như tên mảng ở trên: ts[i]



### 3. Đối tượng được cấp phát động (tt)

- ❖ Như vậy đối tượng được cấp phát động là đối tượng được tạo ra bằng từ khóa **new** và bị hủy đi bằng từ khóa **delete**
- ❖ Từ khóa **new** cấp phát vùng nhớ cho đối tượng trên **Heap** và gọi phương thức thiết lập cho đối tượng được cấp.

### 3. Đối tượng được cấp phát động – Ví dụ

```
#include "iostream"
#include "string.h"
class String {
    char *p;
public:
    String( char *s ) { p = _strdup(s); } //StringDuplicate
    String( const String &s ) { p = _strdup(s.p); } //Hàm tạo sao chép
    ~String() { delete [] p; }
}; //end of String class

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

# 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;
```

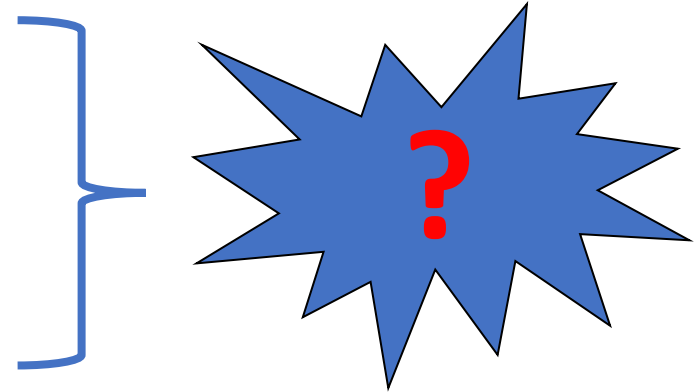
```
delete pd;
```

```
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];
```



-> Sẽ nhận được thông báo lỗi:

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

# Cấp phát và hủy nhiều đối tượng (tt)

- ❖ Trong trường hợp cấp phát nhiều đối tượng, ta không thể cung cấp tham số cho từng phần tử được cấp phát.
- ❖ **Cách giải quyết:** cung cấp phương thức thiết lập để các đối tượng của mảng có khả năng tự khởi động:
  - Phương thức thiết lập không đối số/hàm tạo mặc định;
  - Phương thức thiết lập với đối số có giá trị mặc nhiên (tất cả đối số đều được cung cấp giá trị mặc nhiên).

# Cấp phát và hủy nhiều đối tượng (tt)

```
#include "iostream"
#include "string.h"
class String{
    char *p;
public:
    String (char *s = "Alibaba") { p = _strdup(s); }
    String (const String &s) { p = _strdup(s.p); } //Hàm tạo sao chép
    ~String () {delete [] p;}
};

class Diem {
    double x,y;
public:
    Diem (double xx, double yy) : x(xx),y(yy){};
    Diem () : x(0),y(0){}; //Hàm tạo mặc định
};
```

# Cấp phát và hủy nhiều đối tượng (tt)

- ❖ Khi đó tất cả đối tượng của mảng đều được khởi động với cùng giá trị.

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

-> Cả 5 điểm có cùng tọa độ (0,0)

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

-> Cả 5 chuỗi đều có giá trị là “Alibaba”

- ❖ Hủy nhiều đối tượng: **delete [ ]**

**delete [ ]** pad;

**delete [ ]** pas;

## 4. Giao diện và chi tiết cài đặt

### ❖ Lớp có hai phần:

- Phần giao diện được 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 cài đặt các hàm thành phần, vô hình đối với người dùng.



## 4. Giao diện và chi tiết cài đặt (tt)

- ❖ Có thể thay đổi chi tiết cài đặt:
  - thay đổi thành phần dữ liệu của lớp,
  - thay đổi chi tiết cài đặt các hàm thành phần (do sự thay đổi thành phần dữ liệu hoặc để cải tiến giải thuật).
- ❖ Nhưng đảm bảo không thay đổi phần giao diện
  - không ảnh hưởng đến người sử dụng
  - 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();
};
```

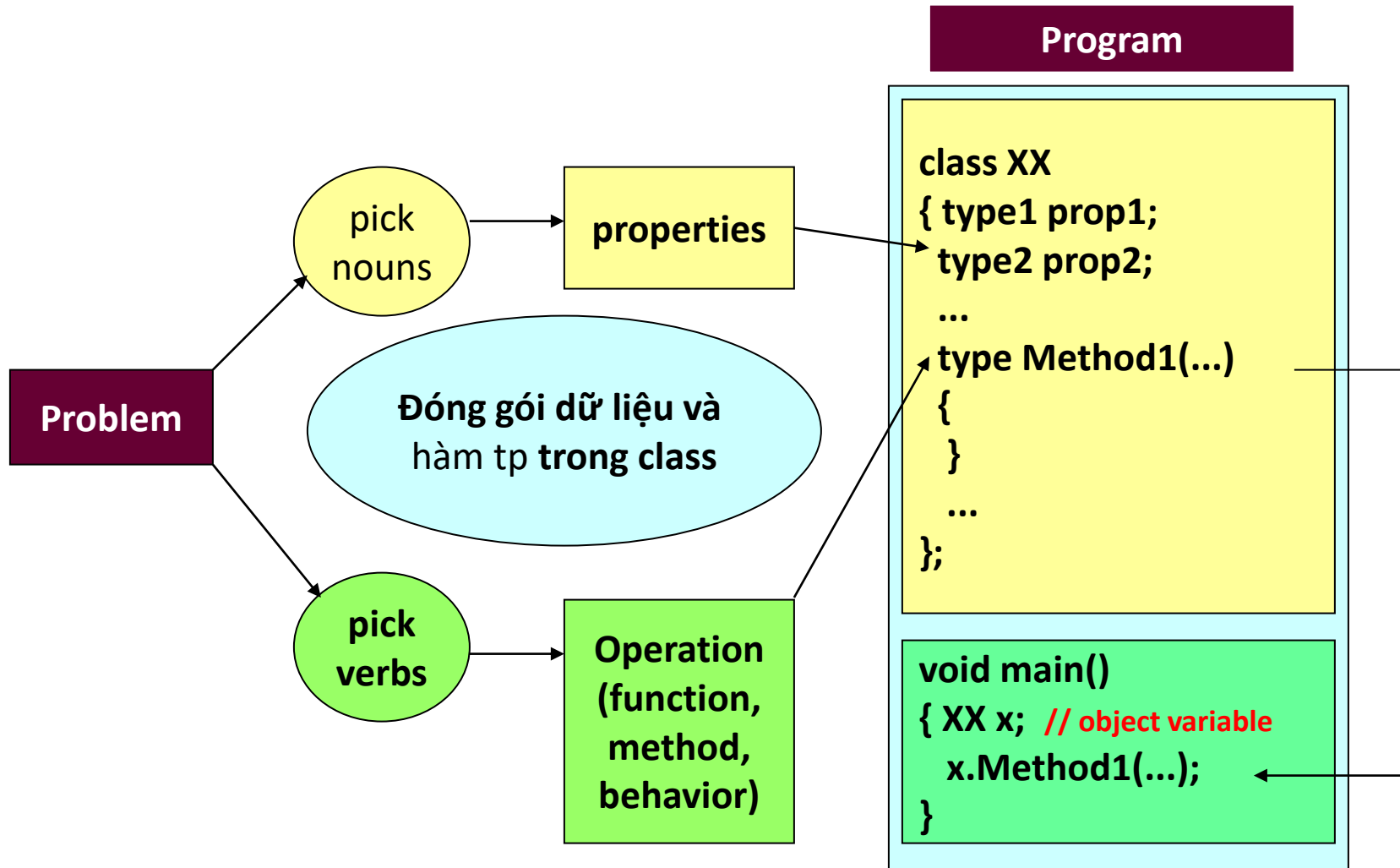
# Lớp ThoiDiem – Cách 2

```
class ThoiDiem{
    long tsgiaiy; //tổng số giây tính từ 0 giờ
    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 tsgiaiy/3600;}
    int LayPhut() const {return (tsgiaiy%3600)/60;}
    int LayGiay() const {return tsgiaiy%60;}
    void Nhap();
    void Xuat() const;
    void Tang();
    void Giam();
};
```

## 5. Các nguyên tắc xây dựng lớp

- ❖ Lớp dùng để biểu diễn khái niệm
  - Tên lớp luôn là **DANH TỪ**
- ❖ Thành phần dữ liệu của lớp là các thuộc tính của 1 khái niệm
  - Tên thuộc tính cũng là **DANH TỪ**
  - \*\*\*Các thuộc tính phải “vừa đủ” để mô tả khái niệm, không dư, không thiếu.
- ❖ Các hàm thành phần của lớp là các hành vi của các đối tượng thuộc lớp
  - Tên các hàm thành phần bắt đầu bằng **ĐỘNG TỪ**

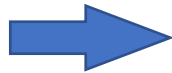
# 5. Các nguyên tắc xây dựng lớp (tt)



## 5. Các nguyên tắc xây dựng lớp (tt)

- ❖ **Loại bỏ các thuộc tính không cần thiết:** loại bỏ các thuộc tính mà *giá trị của nó có thể tính toán được từ giá trị của những thuộc tính khác*. Và khi đó hãy dùng hàm thành phần để thực hiện việc tính toán các giá trị đó.

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



```
class TamGiac{  
    Diem A,B,C;  
public:  
    double TinhChuVi() const;  
    double TinhDienTich() const;  
};
```

## 5. Các nguyên tắc xây dựng lớp (tt)

- ❖ Tuy nhiên, nếu các thuộc tính này cần nhiều tài nguyên và thời gian để thực hiện việc tính toán giá trị cho nó thì ta nên khai báo nó là dữ liệu của lớp.

**Ví dụ:**

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

# 5. Các nguyên tắc xây dựng lớp (tt)

- ❖ Đừng ngại phải xây dựng nhiều lớp nếu điều đó là cần thiết.

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

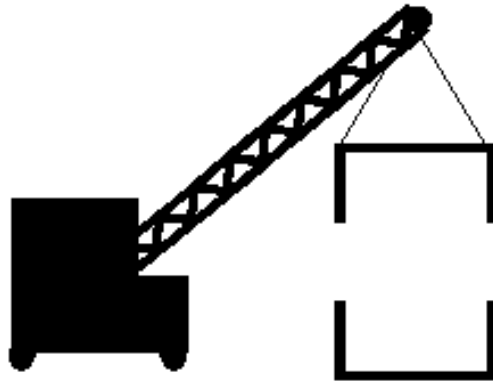


```
class TamGiac{
    Diem A,B,C;
public:
};
class HìnhTron{
    Diem Tam;
    double BanKinh;
public:
};
```



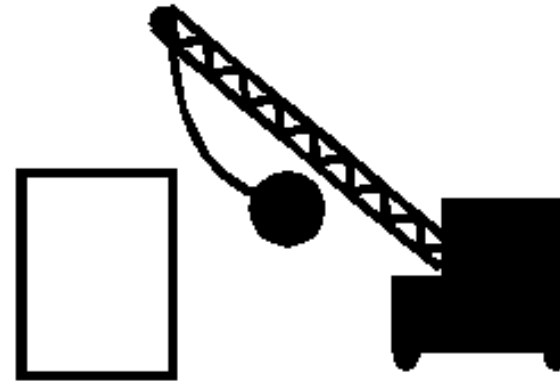
# 5. Các nguyên tắc xây dựng lớp (tt)

- ❖ Trong mọi trường hợp, nên có constructor khởi tạo mặc định.



Constructor

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



Destructor

```
delete MyObjPtr;
```

## 5. Các nguyên tắc xây dựng lớp (tt)

❖ Nếu thành phần dữ liệu của lớp có con trỏ thì lớp cần có các hàm thành phần sau:

+ Hàm tạo

+ Hàm tạo sao chép (copy constructor) để khởi động đối tượng bằng đối tượng cùng kiểu:

**Tên\_lớp**(**const** Tên\_lớp &u) → Tên\_lớp v(u);

+ Hàm hủy để dọn dẹp: **delete** Tên\_con\_trỏ

+ Hàm toán tử gán:

**Kiểu\_trả\_về operator=**(**const** Tên\_lớp &u) → v = u;



Hãy viết một lớp Cipher (Mật mã) trong lập trình hướng đối tượng (OOP) để mã hóa và giải mã một chuỗi ký tự sử dụng phương pháp mã hóa Caesar.

- Lớp Cipher cần có hai thuộc tính là **key (khóa)** và **alphabet (bảng chữ cái)**, cùng hai phương thức là **encrypt (mã hóa)** và **decrypt (giải mã)**. Thuộc tính key sẽ lưu trữ giá trị dịch chuyển cho phương pháp mã hóa Caesar. Thuộc tính alphabet sẽ lưu trữ bảng chữ cái để mã hóa và giải mã các ký tự.
- **Phương thức encrypt** sẽ có một tham số là plaintext (văn bản gốc) và sử dụng thuật toán Caesar để mã hóa chuỗi ký tự này. Kết quả của phương thức là một chuỗi ký tự được mã hóa.
- **Phương thức decrypt** sẽ có một tham số là ciphertext (văn bản đã được mã hóa) và sử dụng thuật toán Caesar để giải mã chuỗi ký tự này. Kết quả của phương thức là một chuỗi ký tự được giải mã.



Đây là bài tập lập trình hướng đối tượng (OOP) lấy ngữ cảnh ở một **phòng thí nghiệm phân tích mã độc (malware)**, có lưu trữ, xử lý thông tin liên quan tới các loại mã độc được phân tích và phương pháp đột biến của mã độc và thông tin họ hàng của chúng.

- **Yêu cầu thực hiện:**

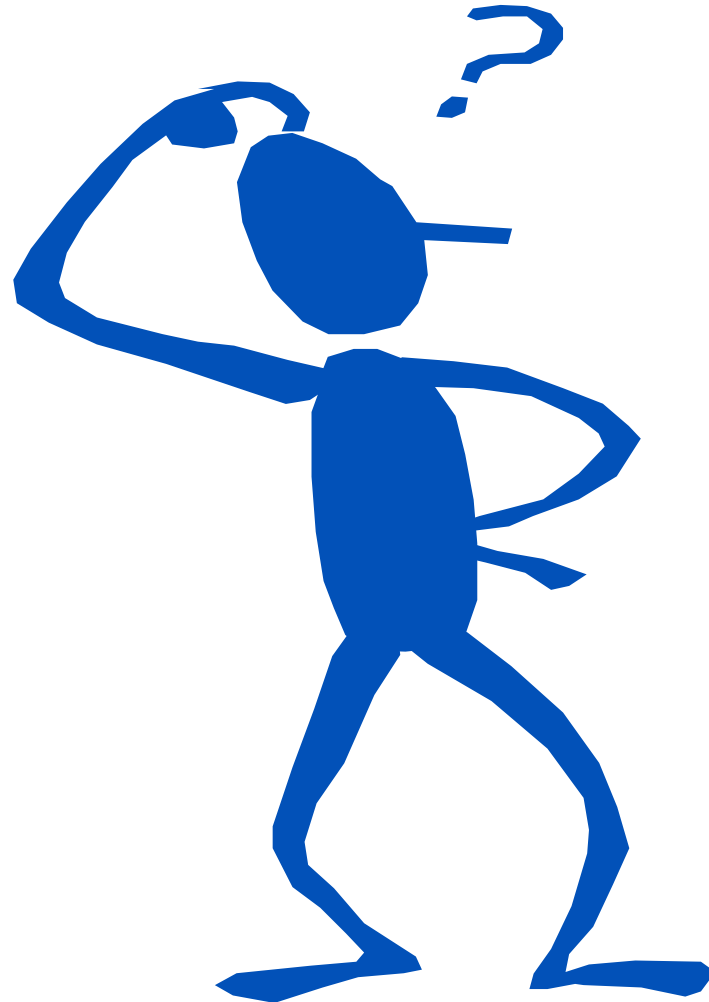
- Khai báo một lớp Malware để đại diện cho một loại mã độc. Lớp này sẽ có các thuộc tính như tên mã độc, họ hàng của nó và các thông tin liên quan khác.
- Tạo một lớp Mutation để mô tả phương pháp đột biến của mã độc. Lớp này sẽ có các thuộc tính như tên phương pháp đột biến, thông tin về cách thực hiện phương pháp và các thông tin liên quan khác.

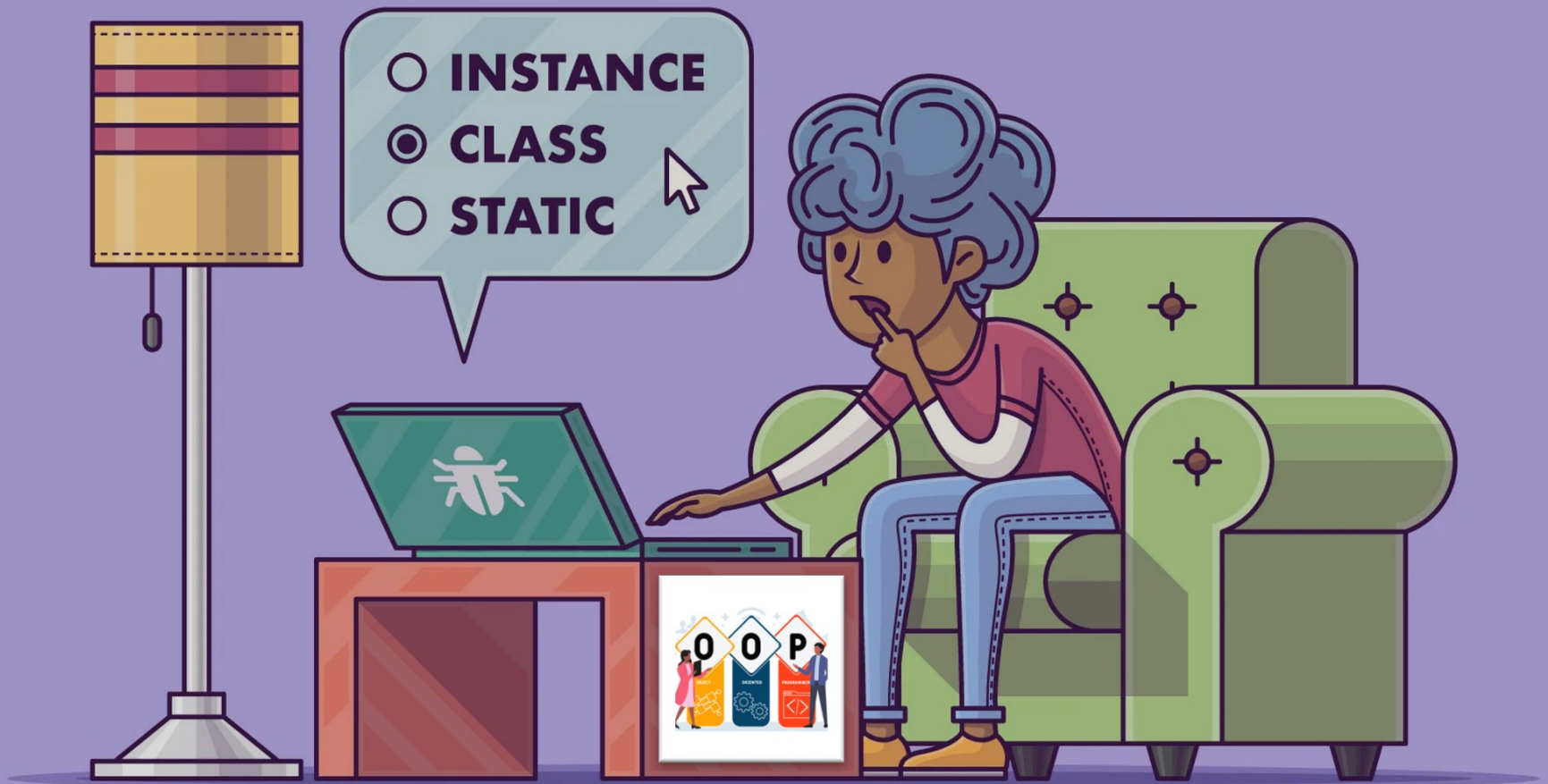
- Tạo một lớp Lab để đại diện cho phòng thí nghiệm. Lớp này sẽ có các thuộc tính như danh sách các loại mã độc đã được phân tích, danh sách các phương pháp đột biến và các thông tin liên quan khác. Lớp Lab sẽ có các phương thức để thêm, xóa và cập nhật thông tin về các loại mã độc và các phương pháp đột biến.
- Tạo một lớp File để đại diện cho một file chứa mã độc. Lớp này sẽ có các thuộc tính như tên file, đường dẫn file và nội dung file. Lớp File sẽ có các phương thức để đọc, ghi và xóa file.

- Tạo một lớp Analysis để thực hiện phân tích các loại mã độc. Lớp này sẽ có các thuộc tính như thông tin về phòng thí nghiệm, thông tin về mã độc cần phân tích và các thông tin liên quan khác. Lớp Analysis sẽ có các phương thức để phân tích mã độc, lưu trữ thông tin về các loại mã độc đã phân tích và cập nhật thông tin liên quan đến các loại mã độc đó.
- Tạo một lớp Main để thực hiện các tác vụ chính của chương trình, bao gồm thêm các loại mã độc mới vào phòng thí nghiệm, phân tích các loại mã độc, thêm các phương pháp đột biến và hiển thị thông tin về các loại mã độc và phương pháp đột biến.

*Lưu ý: Trong bài tập này, các sinh viên sẽ được yêu cầu xây dựng các hàm khởi tạo (constructor) để khởi tạo giá trị cho các đối tượng Malware, Mutation, Lab, File và Analysis.*

# Q & A





## IT002 - Lập trình hướng đối tượng

