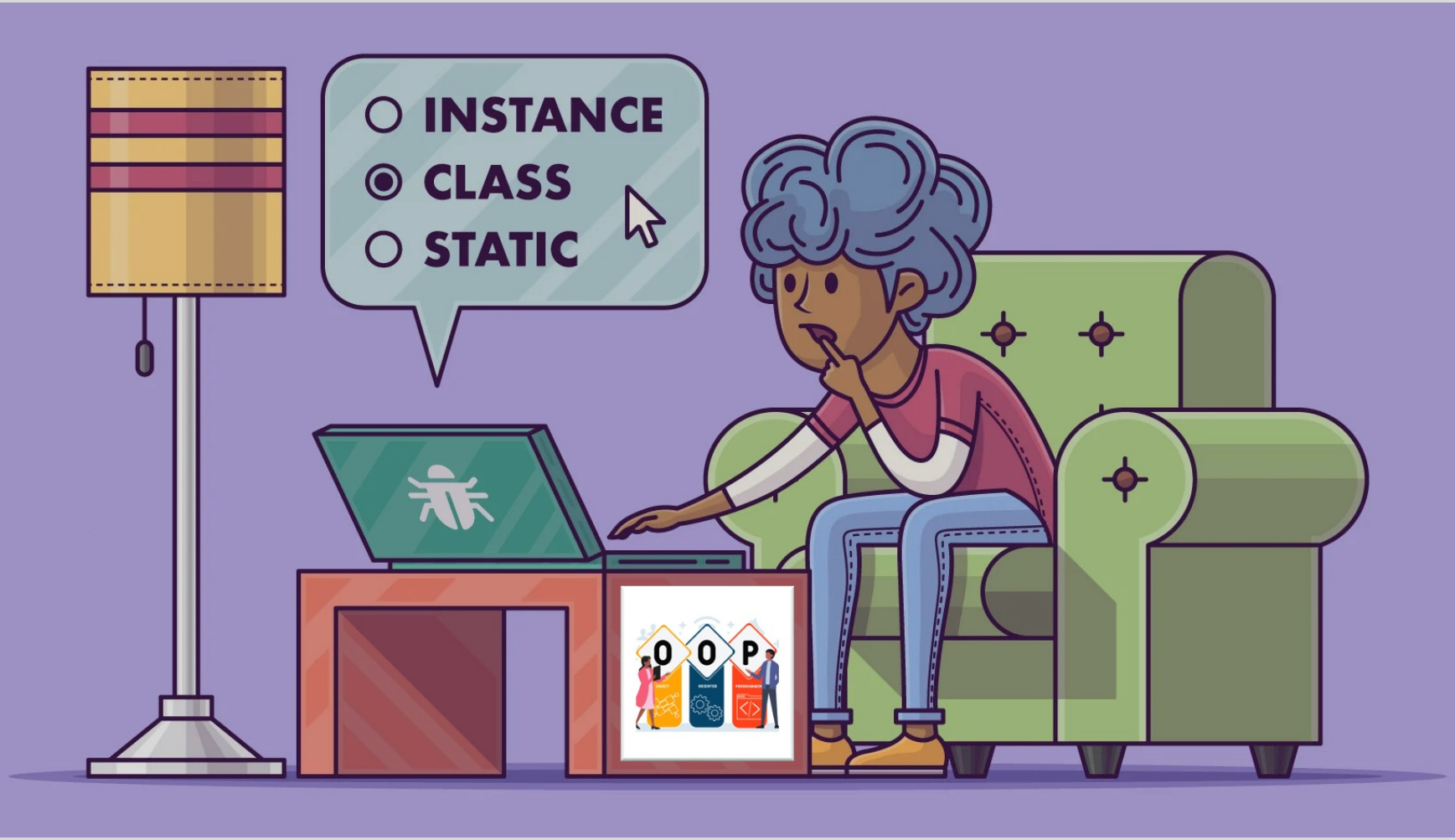


IT002 - Lập trình hướng đối tượng

LỚP VÀ ĐỐI TƯỢNG

Nội dung

1. Lớp
2. Đối tượng
3. Phạm vi truy xuất
4. Hàm thành phần của Lớp
5. Hàm bạn, Lớp bạn

1. Lớp

- Khái niệm Lớp
- Khai báo Lớp
- Các thành phần của Lớp

1.1 Khái niệm Lớp

- Đối tượng **là thực thể** được xác định trong thời gian hệ thống hướng đối tượng hoạt động.
- Đối tượng được chọn sao cho nó thể hiện được **một cách gần nhất** so với những **thực thể trong thế giới thực.**
- Lớp là một mô tả trừu tượng của **nhóm các đối tượng cùng bản chất**, là kết quả của việc khái quát hóa các thực thể.

1.1 Khái niệm Lớp (tt)

- Đối tượng gồm có dữ liệu và các hàm xử lý trên dữ liệu đó.
- Như vậy, một lớp bao gồm **thành phần dữ liệu *(thuộc tính)*** và **thành phần xử lý dữ liệu *(phương thức/hàm thành phần).***

1.2 Khai báo Lớp

```
class <tên_lớp>
{
    //Khai báo các thuộc tính;
    //Khai báo các phương thức;
};
//Định nghĩa các phương thức
```

1.2 Khai báo Lớp – Ví dụ

```
class DIEM {
  private:
        int x,y;
  public:
        void nhap();
};
void DIEM::nhap() {
  cin >>x >>y;
}
```

1.2 Khai báo Lớp (tt)

- Dùng các từ khóa **private** và **public** để qui định phạm vi sử dụng của các thành phần (thuộc tính và phương thức) khi khai báo chúng. Mặc định là **private**.
  - **private:** chỉ được sử dụng bên trong lớp.

    => Các hàm không phải là phương thức của lớp không được phép sử dụng các thành phần này.
  - **public:** được sử dụng bên trong và bên ngoài lớp.
- Các thuộc tính thường được khai báo là **private** và các phương thức thường được khai báo là **public**.

1.3 Các thành phần của lớp

❖ Thuộc tính:

- Thuộc tính của lớp có thể là các **biến, mảng, con trỏ** có kiểu chuẩn/kiểu tự định nghĩa.
- Thuộc tính của lớp **không thể có kiểu của chính lớp đó** nhưng có thể là con trỏ kiểu lớp đó.

Ví dụ: class A{

A x; **//Không cho phép**

A *p; //Được phép

};

1.3 Các thành phần của lớp

❖ Phương thức:

- Các phương thức có thể được cài đặt bên trong hoặc bên ngoài định nghĩa lớp. Khi cài đặt phương thức bên ngoài định nghĩa lớp phải:
  + **Khai báo prototype** của phương thức bên trong định nghĩa lớp.
  + Kiểu_Trả_Về **Tên_Lớp::**Tên_Phương_Thức(ds đối số){…}

1.3 Các thành phần của lớp (tt)

- Xác định các thuộc tính:

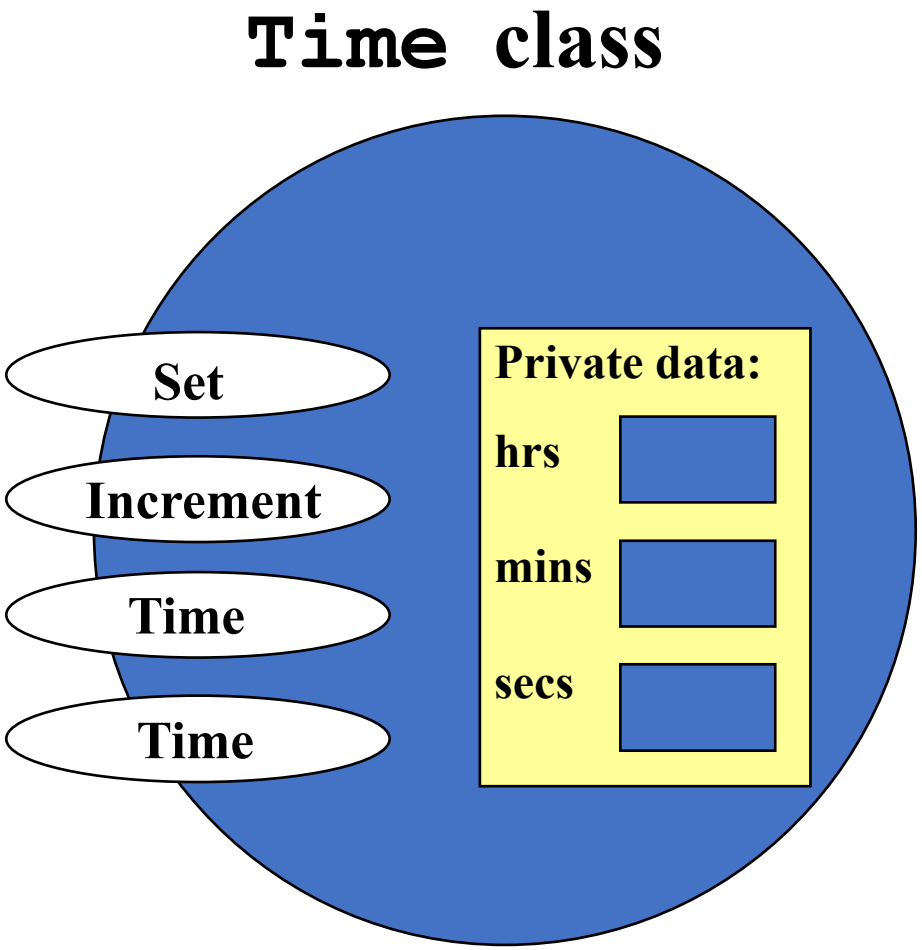
  Việc xác định các thuộc tính của lớp còn tùy thuộc vào việc sử dụng các đối tượng của lớp trong từng bài toán thực tế.

- Xác định các phương thức:
  - Hàm tạo; Hàm hủy
  - Hàm truy vấn; Hàm cập nhật
  - Hàm toán tử
- Chỉ định các quyền truy xuất.

1. Lớp – Ví dụ

```
class  Time {
  private:
        int   hrs;
        int   mins;
        int   secs;
  public:
        void Set (int  hours , int  minutes , int  seconds);
        void Increment ();
        Time (int  initHrs, int  initMins,  int  initSecs );
        Time ( );  //default constructor
} ;
```

1. Lớp – Ví dụ (tt)

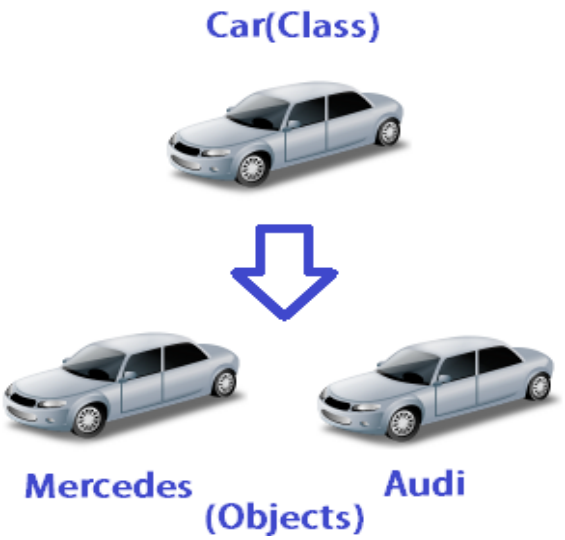

Time class
Set
Increment
Time
Time
Private data:
hrs
mins
secs

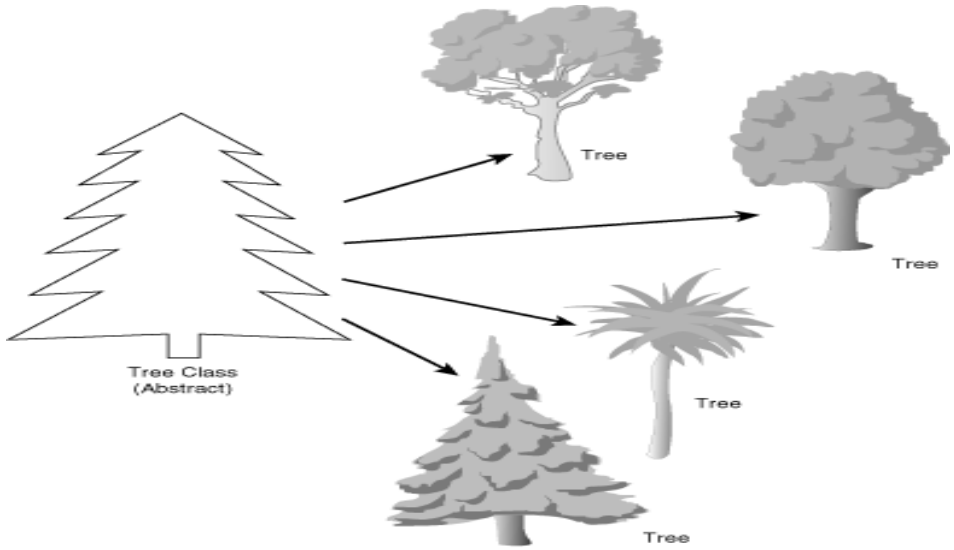
2. Đối tượng

- Khái niệm Đối tượng
- Khai báo Đối tượng, mảng Đối tượng
- Con trỏ Đối tượng
- Cấp phát bộ nhớ cho Đối tượng
- Đối tượng hằng, phương thức hằng

2.1 Khái niệm Đối tượng

- Lớp là một mô tả trừu tượng của nhóm các đối tượng cùng bản chất, ngược lại mỗi một đối tượng là một thể hiện cụ thể cho những mô tả trừu tượng đó.





2.1 Khái niệm Đối tượng (tt)

- Trong lập trình hướng đối tượng:
  - Lớp là kiểu dữ liệu do người sử dụng định nghĩa;
  - Các đối tượng là các biến kiểu class.

2.2 Khai báo Đối tượng

❖ Khai báo đối tượng:

<tên lớp> <tên đối tượng>;

Ví dụ: DIEM d1, d2;

❖ Khai báo mảng đối tượng:

<tên lớp> <tên mảng đối tượng>[số phần tử];

Ví dụ: DIEM d[20];

-> d là mảng kiểu DIEM gồm 20 phần tử.

2.2 Khai báo Đối tượng (tt)

- Mỗi đối tượng sau khi khai báo sẽ được cấp phát một vùng nhớ riêng để chứa các thuộc tính của chúng.
- Không có vùng nhớ riêng để chứa các phương thức cho mỗi đối tượng.

  Các phương thức sẽ được sử dụng chung cho tất cả đối tượng cùng lớp.

⇒ Về bộ nhớ được cấp phát thì đối tượng giống như cấu trúc. Theo VD trên thì:

sizeof(d1) = 2 * sizeof(int) = 4

sizeof(d) = 20 * 4 = 80

2.2 Khai báo Đối tượng (tt)

- **Truy xuất thuộc tính của đối tượng:**

  Tên_Đối_Tượng.Tên_Thuộc_Tính

  Tên_Mảng_Đối_Tượng[chỉ số].Tên_Thuộc_Tính

- **Sử dụng các phương thức của lớp (mà đối tượng thuộc về):**

  Tên_Đối_Tượng.Tên_Phương_Thức(ds đối);

  Tên_Mảng_Đối_Tượng[chỉ số].Tên_Phương_Thức(ds đối);

  -> Để chỉ rõ phương thức thực hiện trên các thuộc tính của đối tượng nào.

2.3 Con trỏ Đối tượng

- Con trỏ đối tượng dùng để chứa địa chỉ của đối tượng, mảng đối tượng.
- **Khai báo:**

  Tên_Lớp * Tên_Con_Trỏ;

- **Ví dụ:** DIEM *p1, *p2; DIEM d1, d2; DIEM d[20];

  p1 = &d2; //p1 trỏ tới d2

  p2 = d; //p2 trỏ tới đầu mảng d

  => sau đó có thể dùng tên con trỏ như tên mảng (p2[i] và d[i] là như nhau).

2.3 Con trỏ Đối tượng (tt)

- Khi đó, để truy xuất thuộc tính của đối tượng thông qua con trỏ:

  Tên_Con_Trỏ ->Tên_Thuộc_Tính

- Để gọi các phương thức của lớp thông qua con trỏ:

  Tên_Con_Trỏ ->Tên_Phương_Thức(ds đối);

2.4 Cấp phát bộ nhớ cho Đối tượng

- Dùng toán tử **new** và **Tên_lớp** để cấp phát một vùng nhớ cho **một hoặc một dãy** các đối tượng.

  Nếu thành công sẽ trả về giá trị khác **NULL**

  Ví dụ: DIEM *p = new DIEM;

  DIEM *q = new DIEM[n];

- Dùng toán tử **delete** để giải phóng vùng nhớ đã cấp phát và gán lại con trỏ bằng **NULL**

  delete p; delete [ ] q;

  p = NULL; q = NULL;

2.4 Cấp phát bộ nhớ cho Đối tượng (tt)

- Khi đó, **để truy xuất đến thành phần của đối tượng** (thuộc tính/phương thức):

  p->tên_thành_phần

  (q+i)->tên_thành_phần (hoặc q[i].tên_thành_phần)

- **Để biểu thị đối tượng:**

  ***p**

  ***(q+i)** (hoặc q[i])

2.4 Cấp phát bộ nhớ cho Đối tượng (tt)

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

Cách 1: Dùng mảng đối tượng.

```
TS ts[100];
```

Cách 2: Dùng con trỏ.

```
TS *ts;
ts = new TS[số_thí_sinh]; //số_thí_sinh được nhập vào
```

Và dùng tên con trỏ giống như tên mảng ở trên: ts[i]

2.5 Đối tượng hằng, phương thức hằng

❖ **Đối tượng hằng:**

- Khi khai báo cần sử dụng hàm tạo để khởi gán giá trị cho đối tượng hằng.
- Giá trị truyền vào cho hàm tạo có thể là hằng, biến, biểu thức, hàm.
- Đối tượng hằng không thể bị thay đổi mà chỉ được tạo ra hoặc hủy bỏ.

Ví dụ: const DIEM d = DIEM(100,100);

//DIEM(100,100) biểu thị một đối tượng kiểu DIEM có các thuộc tính x = 100 và y = 100, đối tượng này được gọi là đối tượng hằng.

2.5 Đối tượng hằng, phương thức hằng (tt)

❖ Phương thức hằng:

- Thêm từ khóa const vào sau tên hàm trong phần khai báo prototype của hàm và định nghĩa hàm.
- Trong thân của phương thức hằng không cho phép thay đổi các thuộc tính của lớp.

Ví dụ: void in() const; //prototype của hàm

void PS::in() const {…}

2.6 Đối tượng – Ví dụ

Xây dựng **class point** trong hệ trục tọa độ Oxy.

- **Thuộc tính:**
  - Tung độ
  - Hoành độ
- **Phương thức:**
  - Khởi tạo
  - Di chuyển
  - In ra màn hình

2.6 Đối tượng – Ví dụ (tt)

```
/*point.cpp*/
#include <iostream>
using namespace std;
class point {
    /*khai báo các thuộc tính*/
    private:
        int x,y;
    /*khai báo các phương thức/hàm thành phần*/
    public:
        void init(int ox, int oy);
        void move(int dx, int dy);
        void display();
}; //kết thúc khai báo lớp
```

2.6 Đối tượng – Ví dụ (tt)

```
/*Định nghĩa các hàm thành phần*/
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";
}
```

2.6 Đối tượng – Ví dụ (tt)

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

```
Ham thanh phan init
Ham thanh phan display
Toa do: 2 4
Ham thanh phan move
Ham thanh phan display
Toa do: 3 6
```

3. Phạm vi truy xuất

- Từ khóa **private**
- Từ khóa **public**
- Từ khóa **static**

3.1 Từ khóa private và public

Các thành phần của lớp (thuộc tính và phương thức) có thể được khai báo là private hoặc public.

- Các thành phần riêng của lớp sẽ được liệt kê trong phần private.

  => Các thành phần này chỉ được sử dụng trong phạm vi của lớp (trong thân của các phương thức cùng lớp).

- Các thành phần công cộng của lớp sẽ được liệt kê trong phần public.

  => Các thành phần này có phạm vi sử dụng trong toàn chương trình (trong thân của bất kỳ hàm nào trong chương trình).

3.1 Từ khóa private và public – Ví dụ

```
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 loại tam giác: 1-d,2-vc,3-c,4-v,5-t
    float DienTich(); //tính diện tích của tam giác
};
```

3.1 Từ khóa private và public – Ví dụ (tt)

```
class TamGiac{
  private:
    float a,b,c; //độ dài ba cạnh
    int Loaitg(); //cho biết loại 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
};
```

3.2 Từ khóa static

1) **Dữ liệu tĩnh:** các thuộc tính của lớp được khai báo bằng từ khóa static gọi là dữ liệu tĩnh.

Dữ liệu tĩnh là dữ liệu dùng chung cho mọi thể hiện của lớp.

Ví dụ: class A{

```
        private:
            static int st;
            int x;
};
int A::st = 1;
```

Dữ liệu tĩnh phải được khởi gán giá trị ban đầu bằng một **câu lệnh khai báo** đặt ngay sau định nghĩa của lớp mà nó thuộc về (bên ngoài tất cả các hàm).

3.2 Từ khóa static (tt)

Lưu ý:

Dữ liệu tĩnh được cấp phát một vùng nhớ cố định, nó tồn tại ngay cả khi lớp chưa có một đối tượng nào.

=> Dữ liệu tĩnh là dữ liệu chung cho cả lớp, nó không phải của riêng mỗi đối tượng.

Ví dụ: Khai báo 2 đối tượng **u, v** thuộc lớp **A**: **A u,v;**
Khi đó: + **u.x và v.x** -> có hai vùng nhớ khác nhau

+ **u.st và v.st** -> cùng chỉ đến 1 vùng nhớ được cấp phát cho dữ liệu tĩnh **st** (ngay cả khi **u và v** chưa được khai báo).

3.2 Từ khóa static

2) Phương thức tĩnh:

- Nếu phương thức được cài đặt bên trong định nghĩa lớp thì thêm từ khóa static trước định nghĩa phương thức.
- Nếu phương thức được cài đặt bên ngoài định nghĩa lớp thì chỉ thêm từ khóa static trước prototype của phương thức (trong định nghĩa lớp).

3.2 Từ khóa static (tt)

Lưu ý:

- Phương thức tĩnh không có đối ngầm định xác định bởi con trỏ this, nó không gắn với một đối tượng cụ thể nào của lớp, khi đó để gọi tới phương thức tĩnh có thể dùng cách gọi thông qua tên lớp mà không cần dùng bất cứ đối tượng nào.

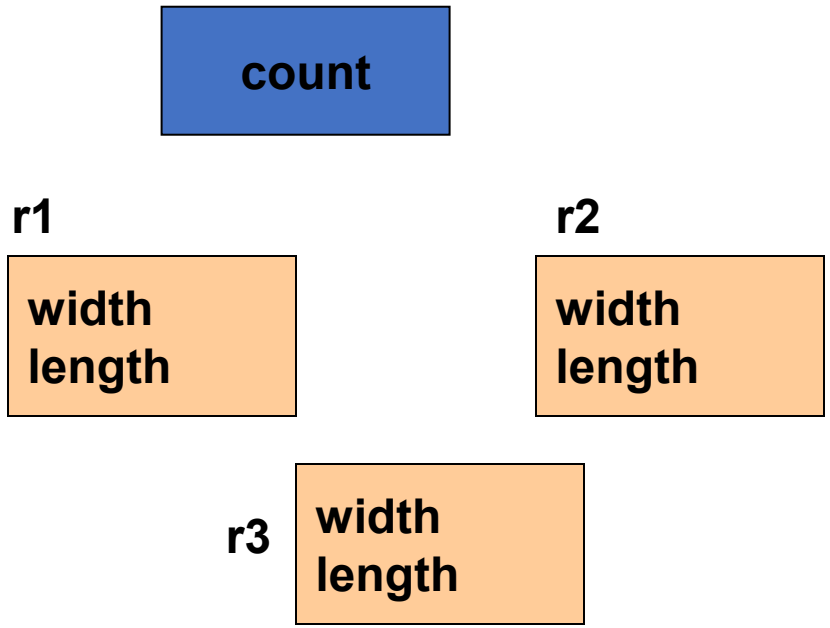
  Tên_Lớp::Tên_Phương_Thức_Tĩnh(ds đối);

- Vì phương thức tĩnh là hàm thành phần của lớp nên nó có thể truy nhập tới các thành phần của lớp.

3.2 Từ khóa static – 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;
```



3.2 Từ khóa static – Ví dụ (tt)

```
class MyClass{
    private:
     static int count; //đếm số đối tượng MyClass
    public:
     MyClass();
     ~MyClass();
     void printCount();
};
int MyClass::count = 0; //phải khởi gán giá trị cho dữ liệu tĩnh
```

3.2 Từ khóa static – Ví dụ (tt)

```
MyClass::MyClass(){
    this → count++;
}
MyClass::~MyClass(){
    this → count--;
}
void MyClass::printCount(){
    cout << “There are currently ” << this → count << “ instance(s)
    of MyClass.\n”;
}
```

3.2 Từ khóa static – Ví dụ (tt)

```
void main()
{
    MyClass *x = new MyClass;
    x → printCount();
    MyClass *y = new MyClass;
    x → printCount();
    y → printCount();
    delete x;
    y → printCount();
}
```

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 kết xuất:

```
Entering a C++ program saying...
Hello, world.
And then exitting…
```

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

Ví dụ về đối tượng toàn cục (tt)

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

4. Hàm thành phần của Lớp

- Khai báo và cài đặt hàm thành phần
- Gọi hàm thành phần
- Con trỏ THIS
- Đối của hàm thành phần
- Các loại hàm thành phần

4.1 Khai báo và cài đặt hàm thành phần

- **Khai báo hàm thành phần** bên trong định nghĩa lớp.

  (bắt buộc phải có khai báo này khi hàm thành phần không được cài đặt bên trong định nghĩa lớp).

  Kiểu_Trả_Về Tên_Phương_Thức(ds đối);

- **Kiểu trả về của hàm thành phần:**

  + void (không có giá trị trả về)

  + Kiểu dữ liệu chuẩn/Kiểu dữ liệu tự định nghĩa

  + **Con trỏ** hoặc **tham chiếu** đến kiểu dữ liệu chuẩn/ kiểu dữ liệu tự định nghĩa.

4.1 Khai báo và cài đặt hàm thành phần (tt)

- **Cài đặt hàm thành phần:** hàm thành phần có thể được cài đặt bên trong hoặc bên ngoài định nghĩa lớp.
  - Nếu hàm thành phần được cài đặt bên ngoài định nghĩa lớp thì phải chỉ định hàm thành phần của lớp nào:
    Kiểu_Trả_Về **Tên_Lớp::**Tên_Phương_Thức(ds đối){…}
  - Hàm thành phần cài đặt bên trong định nghĩa lớp thì được biên dịch theo kiểu **inline**.
  - Vì vậy chỉ những hàm thành phần có cài đặt đơn giản, không chứa các câu lệnh phức tạp thì mới nên đặt trong định nghĩa lớp.

4.1 Khai báo và cài đặt hàm thành phần (tt)

❖ **Cài đặt hàm thành phần (tt):**

- Trong thân của một hàm thành phần có thể:
  + Truy xuất đến các thuộc tính của lớp
  + Gọi các hàm thành phần khác của lớp
- **Kiểu trả về** của hàm thành phần; **đối** của hàm thành phần; **biến cục bộ** của hàm thành phần có thể có kiểu là lớp đang được định nghĩa.

4.1 Khai báo và cài đặt hàm thành phần – Ví dụ

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

Inline function

class name

member function name

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

scope operator

4.2 Gọi hàm thành phần

Để gọi hàm thành phần (htp) của lớp cần tạo đối tượng thuộc lớp trước khi gọi htp (trừ hàm tĩnh có thể được gọi thông qua tên lớp).

<tên đối tượng>.<tên htp>(ds đối);

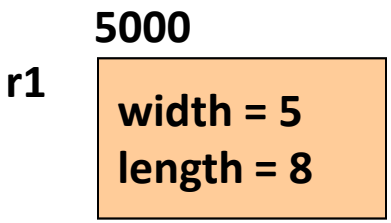
<tên mảng đối tượng>[chỉ số].<tên htp>(ds đối);

<tên con trỏ đối tượng>→<tên htp>(ds đối);

4.2 Gọi hàm thành phần (tt)

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

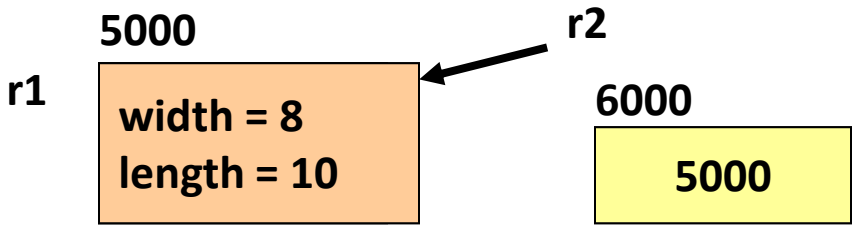
```
void main()
{
     Rectangle r1;
     r1.set(5, 8);
}
```



4.2 Gọi hàm thành phần (tt)

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

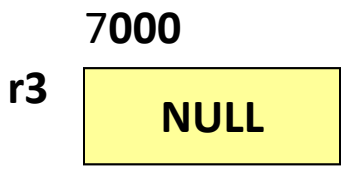
```
main()
{
    Rectangle r1;
    r1.set(5, 8);

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



4.2 Gọi hàm thành phần (tt)

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

```
main()
{
    Rectangle *r3;
    r3 = new Rectangle();
    r3->set(80,100);
    delete r3;
    r3 = NULL;
}
```



4.3 Con trỏ this

Con trỏ **this** được dùng như đối thứ I của hàm, còn gọi là đối ẩn vì không xuất hiện một cách tường minh.

Hàm bao giờ cũng có ít nhất 1 đối là đối ẩn:

- **Với hàm tạo:** đối ẩn được gán giá trị là địa chỉ của đối tượng mới được hình thành.
- **Với hàm hủy:** đối ẩn có giá trị là địa chỉ của đối tượng sẽ bị hủy bỏ.
- **Với hàm toán tử:** đối ẩn có giá trị là địa chỉ của toán hạng thứ nhất.
- **Với hàm thông thường:** đối ẩn có giá trị là địa chỉ của đối tượng chủ thể trong lời gọi hàm (vì vậy mà không cho phép dùng con trỏ **this** trong phương thức tĩnh).

Để truy xuất đối tượng có địa chỉ chứa trong con trỏ **this** ta dùng: ***this**

4.3 Con trỏ this – Ví dụ

```
class TS {
private:
        char *ht;
        double td;
public:
        TS() {
             ht = new char[30];
             td = 0;
        }
        ~TS() {
                 delete ht;
        }
```

```
int main() {
    TS ts[2]; …
    ts[0].hoanvi(ts[1]);
    system("pause");
    return 0;
}
```

```
//Định nghĩa chồng toán tử '=' (còn gọi là toán tử gán).
//Lưu ý là nếu dùng phép gán mặc định sẽ cho kết quả sai.
const TS &operator = (const TS &ts2) {
        this->td = ts2.td;
        strcpy_s(this->ht,30,ts2.ht);
        return ts2;
}
//Hàm sau đây cho thấy vai trò của con trỏ this:
void hoanvi(TS &ts2) {
        TS tmp;
        tmp = *this;
        *this = ts2;
        ts2 = tmp;
}
}//end of class
```

4.4 Đối của hàm thành phần

- Đối của hàm thành phần có thể có kiểu dữ liệu chuẩn hoặc kiểu dữ liệu tự định nghĩa.
- Đối của hàm thành phần cũng có thể là **con trỏ** hoặc **tham chiếu** đến kiểu dữ liệu chuẩn/kiểu dữ liệu tự định nghĩa.
- Thông qua **đối ẩn** (con trỏ **this**), hàm thành phần có thể truy xuất đến các thuộc tính của lớp và gọi hàm thành phần khác của lớp:
  - **this->Tên_thuộc_tính** hoặc chỉ ghi **Tên_thuộc_tính**
  - **this->Tên_phương_thức();** hoặc **Tên_phương_thức();**

4.4 Đối của hàm thành phần (tt)

```
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);
}
```

4.5 Các loại hàm thành phần

- Hàm tạo
- Hàm hủy
- Hàm truy vấn
- Hàm cập nhật
- Hàm toán tử

4.5.1 Hàm tạo

- Hàm tạo: khái niệm, cách viết và việc sử dụng
- Hàm tạo mặc định
- Hàm tạo sao chép
- Toán tử gán
- Hàm tạo và đối tượng thành phần

4.5.1.1 Hàm tạo

❖ **Khái niệm:**

- Hàm tạo (constructor) là một phương thức của lớp dùng để tạo dựng một đối tượng mới.
- Chương trình dịch sẽ cấp phát bộ nhớ cho đối tượng và gọi đến hàm tạo.
- Hàm tạo sẽ khởi gán giá trị cho các thuộc tính của đối tượng.

4.5.1.1 Hàm tạo (tt)

❖ **Đặc điểm của hàm tạo:**

- Không khai báo kiểu cho hàm tạo, hàm tạo không có kết quả trả về.
- Tên của hàm tạo phải trùng với tên của lớp.
- Hàm tạo có thể có đối hoặc không có đối.
- Các đối số (nếu có) của hàm tạo có thể nhận giá trị mặc định.
- Trong một lớp có thể có nhiều hàm tạo (các hàm này cùng tên nhưng khác danh sách đối).
- Hàm tạo có thể được cài đặt bên trong hoặc bên ngoài định nghĩa lớp.

4.5.1.1 Hàm tạo (tt)

❖ **Dùng hàm tạo trong khai báo:**

- Tên_lớp **Tên_đối_tượng;** -> Gọi tới hàm tạo không đối
- Tên_lớp **Tên_đối_tượng(ds đối);** -> Gọi tới hàm tạo có đối
- Tên_lớp **Tên_mảng_đối_tượng[n];** -> Gọi tới hàm tạo không đối **n lần** và chỉ gọi tới hàm tạo này khi khai báo mảng đối tượng.

=> Trình biên dịch sẽ dựa vào số tham số trong khai báo để gọi hàm tạo tương ứng.

***Lưu ý:** khi khai báo đối của một hàm có kiểu lớp thì sẽ không tạo ra đối tượng mới cho đối (vì đối của hàm chỉ được xem là các tham số hình thức) và do đó không gọi tới các hàm tạo.

4.5.1.1 Hàm tạo (tt)

❖ **Dùng hàm tạo trong cấp phát bộ nhớ:**

- Tên_lớp ***Tên_con_trỏ_đối_tượng** = **new** Tên_lớp**;**

  -> Gọi tới hàm tạo không đối

- Tên_lớp ***Tên_con_trỏ_đối_tượng** = **new** Tên_lớp(ds đối)**;**

  -> Gọi tới hàm tạo có đối

- Tên_lớp ***Tên_con_trỏ_mảng_đối_tượng** = **new** Tên_lớp[n]**;**

  -> Gọi tới hàm tạo không đối **n lần** và chỉ gọi tới hàm tạo này khi khai báo con trỏ mảng đối tượng.

4.5.1.1 Hàm tạo (tt)

- **Dùng hàm tạo để biểu diễn đối tượng hằng:**

  Tên_lớp(ds đối)

  Ví dụ: DIEM(100,200)

  ***<u>**Lưu ý:**</u> vì hằng đối tượng cũng là một đối tượng nên có thể dùng hằng đối tượng để gọi một phương thức của lớp.

  Ví dụ: DIEM(100,200).in();

4.5.1.1 Hàm tạo (tt)

❖ **Lớp không có hàm tạo:**

- Nếu không xây dựng bất kỳ hàm tạo nào cho lớp thì trình biên dịch sẽ cung cấp một hàm tạo mặc định (hàm tạo không đối).
- Khi đó, một đối tượng tạo ra chỉ được cấp phát bộ nhớ, còn giá trị các thuộc tính của nó thì chưa được xác định.
- Nếu trong lớp đã có ít nhất một hàm tạo (có đối hoặc không đối) thì hàm tạo mặc định sẽ không được phát sinh tự động nữa.

4.5.1.1 Hàm tạo – Ví dụ

```
class point{
    int x, y;
    public:
     point() { x = 0; y = 0; } //Hàm tạo không đối
     point(int ox, int oy) { x = ox; y = oy; }      //Hàm tạo có đối
     void move (int dx, int dy);
     void display();
};
//Dùng hàm tạo trong khai báo
point a(5,2); //Gọi hàm tạo có đối, khi đó x = 5 và y =2
point b; //Gọi hàm tạo không đối, x = 0 và y = 0
point c(3); //Báo lỗi vì không có hàm tạo tương ứng
```

4.5.1.1 Hàm tạo – Ví dụ (tt)

```
class point{
      int x, y;
      public:
        point() { x = 0; y = 0; } //Hàm tạo không đối
        //Hàm tạo có đối nhận giá trị mặc định
        point(int ox, int oy = 1){ x = ox; y = oy;}
        void move (int dx, int dy);
        void display();
};
//Dùng hàm tạo trong khai báo
point a(5,2); //Gọi hàm tạo có đối, khi đó x = 5 và y =2
point b; //Gọi hàm tạo không đối, x = 0 và y = 0
point c(3); //Gọi hàm tạo có đối, khi đó x = 3 và y = 1
```

4.5.1.2 Hàm tạo mặc định

- Hàm tạo mặc định là hàm tạo không đối.
- Trong hàm tạo mặc định sẽ dùng các giá trị cố định để khởi gán cho các thuộc tính của đối tượng.
- Vì hàm tạo mặc định sẽ không được phát sinh tự động khi đã có ít nhất một hàm tạo khác nên phải cài đặt hàm tạo mặc định cho các lớp được xây dựng.

4.5.1.2 Hàm tạo mặc định – Ví dụ

```
class point{
    int x, y;
    public:
     point(int ox, int oy = 1){ x = ox; y = oy;}
     void move (int dx, int dy);
     void display();
};
point a(5,2);
point b; //Báo lỗi vì không có hàm tạo không đối được cài đặt
point c(3);
```

4.5.1.3 Hàm tạo sao chép

- Có thể dùng lệnh khai báo để tạo một đối tượng mới từ một đối tượng đã tồn tại.
- **Ví dụ:** PS u;

  PS v(u);

  => Sẽ gọi tới hàm tạo sao chép mặc định của C++

  Hàm này sẽ sao chép nội dung từng bit của **u** vào các bit tương ứng của **v**.

  Như vậy, các vùng nhớ của **u** và **v** sẽ có nội dung như nhau.

4.5.1.3 Hàm tạo sao chép (tt)

- Tuy nhiên, khi lớp có các thuộc tính kiểu con trỏ hay tham chiếu thì phải xây dựng một hàm tạo sao chép cho lớp chứ không sử dụng hàm tạo sao chép mặc định được.
- Bởi vì khi đó 2 con trỏ (thuộc tính của 2 đối tượng) cùng trỏ đến một vùng nhớ nên sự thay đổi của đối tượng này sẽ ảnh hưởng đến đối tượng kia.

4.5.1.3 Hàm tạo sao chép (tt)

- Mục đích của hàm tạo sao chép là tạo ra đối tượng **v** có nội dung ban đầu giống như đối tượng **u** nhưng độc lập với **u**.
- Hàm tạo sao chép có một đối kiểu tham chiếu để khởi gán cho đối tượng mới.

```
Tên_lớp(const Tên_lớp &u){
    this->Tên_thuộc_tính = u.Tên_thuộc_tính;
    this->Tên_con_trỏ = new Kiểu_dữ_liệu[n];
}
Tên_lớp u; //Gọi tới hàm tạo mặc định
Tên_lớp v(u); //Gọi tới hàm tạo sao chép đã xây dựng
```

4.5.1.4 Toán tử gán

- Có thể dùng toán tử gán (=) để gán một đối tượng này cho một đối tượng khác.
- **Ví dụ:** HT h1,h2(100,6); //bán kính và màu

  h1 = h2; //gán h2 cho h1

  => Sẽ gọi tới toán tử gán mặc định của C++

  Toán tử gán mặc định sẽ sao chép đối tượng nguồn **h2** vào đối tượng đích **h1** theo từng bit một.

4.5.1.4 Toán tử gán (tt)

- Tuy nhiên, khi lớp có các thuộc tính kiểu con trỏ hay tham chiếu thì không thể sử dụng toán tử gán mặc định mà phải xây dựng toán tử gán cho lớp.
- Trong đó, **đối ẩn (con trỏ this) biểu thị đối tượng đích** và **1 đối tường minh biểu thị đối tượng nguồn.**

```
Kiểu_trả_về operator=(const Tên_lớp &u){
    this->Tên_thuộc_tính = u.Tên_thuộc_tính;
    this->Tên_con_trỏ = new Kiểu_dữ_liệu[n]; //cấp phát trước
    memcpy(this->Tên_con_trỏ,u.Tên_con_trỏ,n);}
```

Ví dụ: HT v; //gọi tới hàm tạo mặc định của lớp HT

```
v = u;
```

4.5.1.4 Toán tử gán (tt)

- Nếu không có giá trị trả về (kiểu **void**) thì chỉ cho phép gán đối tượng nguồn cho 1 đối tượng đích.
- Nếu toán tử gán trả về kiểu tham chiếu (đến đối tượng nguồn) thì có thể viết câu lệnh gán đối tượng nguồn cho nhiều đối tượng đích.

```
const Tên_lớp& operator=(const Tên_lớp &u){
    …
    return u;}
```

Ví dụ: HT x, y, z; //gọi tới hàm tạo mặc định 3 lần

```
x = y = z = u;
```

Toán tử gán và Hàm tạo sao chép

- Toán tử gán không tạo ra đối tượng mới, chỉ thực hiện phép gán giữa hai đối tượng đã tồn tại.
- Hàm tạo sao chép được dùng để tạo một đối tượng mới và gán nội dung của một đối tượng đã tồn tại cho đối tượng mới vừa tạo.
- **Câu lệnh new và câu lệnh khai báo -> gọi hàm tạo:**

  HT *h = new HT(50,6); //gọi hàm tạo có đối

  HT k = *h; //không gọi toán tử gán mà gọi hàm tạo sao chép
- **Câu lệnh gán -> gọi toán tử gán:**

  HT u; u = *h; //gọi hàm toán tử gán

4.5.1.5 Hàm tạo và đối tượng thành phần

- **Lớp bao:** lớp có thuộc tính là đối tượng của lớp khác.
- Trong các hàm thành phần của lớp bao không cho phép truy nhập trực tiếp đến các thuộc tính của các đối tượng là thành phần của lớp bao.
- Vì vậy khi xây dựng **hàm tạo của lớp bao** phải sử dụng các hàm tạo của các lớp thành phần tương ứng để khởi gán cho các thuộc tính là đối tượng của lớp bao.

Lớp bao – 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
```

4.5.2 Hàm hủy

❖ **Khái niệm:**

- Hàm hủy (destructor) là một phương thức của lớp dùng để thực hiện một số công việc có tính "dọn dẹp" trước khi đối tượng được hủy bỏ.
- Nếu trong lớp không định nghĩa hàm hủy thì hàm hủy mặc định được phát sinh tự động.
- Tuy nhiên, chỉ vùng nhớ cho các thuộc tính khác của đối tượng được giải phóng, còn vùng nhớ được cấp phát động thì không hề bị giải phóng mà sẽ bị chiếm dụng vô ích.

4.5.2 Hàm hủy (tt)

❖ **Đặc điểm của hàm hủy:**

- Không khai báo kiểu cho hàm hủy, hàm hủy không có kết quả trả về.
- Tên của hàm hủy trùng với tên của lớp và có thêm dấu '~' đặt phía trước.
- Hàm hủy không có đối số.
- Trong một lớp chỉ có duy nhất một hàm hủy.
- Hàm hủy có thể được cài đặt bên trong hoặc bên ngoài định nghĩa lớp.

4.5.2 Hàm hủy – Ví dụ

```
class DT{
    int n; //bậc của đa thức
    double *a; //trỏ tới vùng nhớ chứa các hệ số của đa thức
public:
    DT(); //Hàm tạo mặc định
    ~DT(){
     this->n = 0;
     delete this->a;
    }
}; //end of class
```

4.5.3 Hàm truy vấn

- Hàm truy vấn là hàm dùng để truy xuất giá trị của các thuộc tính của đối tượng.
- **Đặt tên hàm truy vấn:**
  - Truy vấn đơn giản: "get" + tên thuộc tính cần truy xuất.

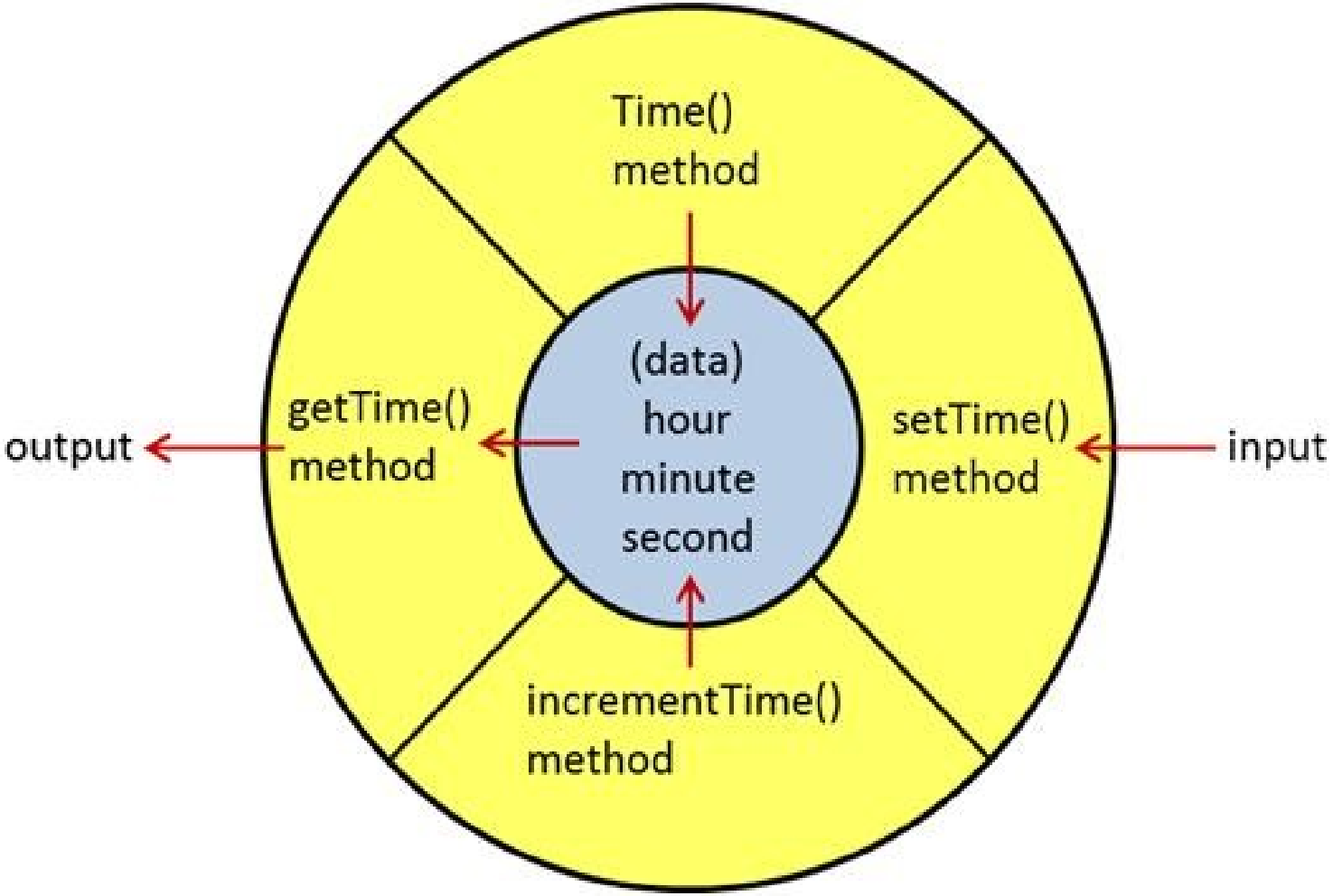
    Ví dụ: int getSize(){…}
  - Truy vấn điều kiện: "is" + điều kiện

4.5.4 Hàm cập nhật

- Hàm cập nhật là hàm dùng để cập nhật giá trị cho các thuộc tính của đối tượng.
- **Đặt tên hàm cập nhật:**
  - Cập nhật đơn giản: "set" + tên thuộc tính cần cập nhật.

    Ví dụ: void setX(int x1){…}

Truy vấn và Cập nhật



Truy vấn và Cập nhật (tt)

Các hàm truy vấn và cập nhật cho phép ta truy xuất và cập nhật giá trị cho **các thuộc tính private** của đối tượng, trong đó:

- Đảm bảo nguyên tắc đóng gói.
- Kiểm tra tính hợp lệ của dữ liệu trong hàm cập nhật.
- Chỉ được truy xuất/cập nhật các thuộc tính của đối tượng thông qua các hàm truy xuất và cập nhật được cung cấp.

Truy vấn và Cập nhật – Ví dụ

```
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
    }
}
```

4.5.5 Hàm toán tử

- **Hàm toán tử:** giống như hàm bình thường, chỉ khác cách đặt tên.
- **Đặt tên hàm toán tử: operator**"phép toán"

  Ví dụ: operator<<
- Hàm toán tử có **đối đầu tiên là con trỏ this** (đối ẩn):
  - **Toán tử một toán hạng:** hàm sẽ không có đối tường minh (ví dụ: operator-()).
  - **Toán tử hai toán hạng:** con trỏ this ứng với toán hạng thứ nhất (ví dụ: operator+(SP u2)).

4.5.5 Hàm toán tử – Ví dụ

```
class SoPhuc
{
private:
        double a;
        double b;
public:
        SoPhuc();
        SoPhuc(double a, double b);
        ~SoPhuc();
        SoPhuc operator-() {
                SoPhuc u;
                u.a = -this->a;
                u.b = -this->b;
                return u;
        }
```

```
SoPhuc operator+(SoPhuc u2) {
        SoPhuc u;
        u.a = this->a + u2.a;
        u.b = this->b + u2.b;
        return u;}
};//end of class
int main() {
        SoPhuc u(1,2), v(3,4), s;
        s = -u;
        s.insophuc();
        s = u + v;
        s.insophuc();
        system("pause");
        return 0;
}
```

5. Hàm bạn, Lớp bạn

Bên ngoài lớp có thể truy xuất đến các thành phần private của lớp bằng cách thêm vào từ khóa friend trước tên hàm hoặc tên lớp như sau:

- **Hàm bạn của lớp:**

  friend Kiểu_trả_về **Tên_hàm**(ds đối)

- **Lớp bạn:**

  friend class **Tên_lớp**;

5.1 Hàm bạn

Có 2 cách để trở thành **hàm bạn** của một lớp:

Cách 1:

+ Dùng từ khóa friend để khai báo hàm trong định nghĩa của lớp;

+ Cài đặt hàm bên ngoài định nghĩa lớp.

Lưu ý: hàm bạn không phải là hàm thành phần của lớp.

Ví dụ:

```
class A{
        public:
                friend void f();
};  //end of class
void f(){…}
```

5.1 Hàm bạn (tt)

Cách 2:

+ Dùng từ khóa friend để xây dựng hàm trong định nghĩa của lớp.

Lưu ý: trong trường hợp này hàm bạn vẫn không phải là hàm thành phần của lớp.

Ví dụ: class A{

```
        public:
            friend void f(){…}
};  //end of class
```

5.1 Hàm bạn (tt)

Tính chất của hàm bạn:

- Hàm bạn của một lớp có thể truy nhập tới các thuộc tính **private** của **đối tượng thuộc lớp**. Lưu ý là hàm không thuộc lớp và không phải là hàm bạn của lớp thì không làm được việc này.
- **Một hàm f có thể là bạn của nhiều lớp.** Khi đó:

  + Khai báo **f** trong thân tất cả các class mà nó là bạn:

    friend Kiểu_trả_về **f**(ds đối);

  + Cài đặt **f** bên ngoài tất cả các class này:

    Kiểu_trả_về **f**(ds đối){…}

  Ví dụ: class A{friend void **f**();…};

    class B{friend void **f**();…};

    class C{friend void **f**();…};

    void **f**(){…}

5.1 Hàm bạn (tt)

Lời gọi hàm bạn:

- Hàm thành phần của lớp có một đối ẩn -> lời gọi hàm thành phần của lớp phải gắn với một 1 tượng nào đó, khi đó địa chỉ của đối tượng này được truyền cho con trỏ this.

  Ví dụ: SP SP::cong(SP u2){…} //hàm thành phần của lớp

  ```
  SP u,u1,u2;
  u = u1.cong(u2);
  ```

- Lời gọi hàm bạn giống như lời gọi hàm thông thường, không cần gắn với 1 đối tượng thuộc lớp (vì hàm bạn không phải là hàm thành phần của lớp).

  Ví dụ: **friend** SP cong(SP u1, SP u2){…} //hàm bạn

  ```
  u = cong(u1,u2);
  ```

5.1 Hàm bạn – Ví dụ

```
class COUNTERCLASS{
    int Counter;
public:
    char CounterChar;
    void Init(char);
    void AddOne( ){
     Counter++;
    }
    friend int Total (int); //khai báo hàm bạn của lớp
}; //end of class
```

5.1 Hàm bạn – Ví dụ (tt)

```
COUNTERCLASS MyCounter[26]; //mảng đt toàn cục
int Total(int NumberObjects)//cài đặt hàm bạn bên ngoài lớp
{
    for (int i=0, sum=0; i<NumberObjects; i++)
     sum += MyCounter[i].Counter
    /*hàm bạn có thể truy xuất tới thuộc tính private của đối
    tượng thuộc lớp mà nó là bạn*/
    return sum;
}
```

5.2 Lớp bạn

- Giả sử có lớp **Vector** và lớp **Matrix**.
- Hàm nhân 1 **Vector** với 1 **Matrix** -> 1 **Vector**

  Hàm nhân 1 **Matrix** với 1 **Vector** -> 1 **Vector**
- Trong hàm này chắc chắn sẽ phải truy nhập đến các thuộc tính private của cả 2 lớp (**Vector** và **Matrix**).
- Một hàm chỉ có thể được cài đặt trong một lớp.

Vậy làm thế nào để truy nhập đến các thuộc tính private của cả 2 lớp trong hàm nhân?

=> Khai báo lớp **Vector** là bạn của lớp **Matrix** và ngược lại.

5.2 Lớp bạn (tt)

Giả sử có 2 lớp A, B. Để khai báo lớp này là bạn của lớp kia ta viết như sau:

//Khai báo trước các lớp:

```
class A;
class B;
```

//Định nghĩa các lớp:

```
class A{…
        friend class B; //lớp B là bạn của lớp A};

class B{…
        friend class A; //lớp A là bạn của lớp B};
```

5.2 Lớp bạn (tt)

- Nếu lớp **A** được khai báo là bạn của lớp **B** (trong định nghĩa của lớp **B** chứa câu lệnh "friend class **A**;") thì tất cả hàm thành phần của lớp **A** đều có thể truy nhập đến các thành phần private của lớp **B**.
- Một lớp có thể là bạn của nhiều lớp khác.
- Trong lớp **B** có thể khai báo lớp **A** là bạn và trong lớp **A** có thể khai báo lớp **B** là bạn.
- **Các tính chất của quan hệ friend:**
  - Không đối xứng
  - Không bắc cầu

5.2 Lớp bạn – Ví dụ

```
class TOM{
private:
    int SecretTom;          //Bí mật của TOM
public:
    friend class JERRY;     //lớp JERRY là bạn của lớp TOM
};
class JERRY{
public:
    void Change(TOM T){
            T.SecterTom++; //hàm thành phần của lớp JERRY có
            //thể truy nhập đến thành phần private của lớp TOM
    }
}; //end of class
```

Bài tập

Bài toán quản lý thông tin Xe hơi (Car):

- Định nghĩa lớp "Car" (Xe hơi) với các thuộc tính như: mã xe, hãng sản xuất, năm sản xuất, giá bán, màu sắc, v.v.
- Tạo một phương thức để tính toán tuổi của chiếc xe dựa trên năm sản xuất.
- Tạo một phương thức khác để kiểm tra xem một chiếc xe có đang được bảo dưỡng đúng đắn hay không dựa trên số km đã đi và thời gian kể từ lần bảo dưỡng gần nhất.
- Tạo một phương thức để hiển thị tất cả thông tin về chiếc xe.
- Tạo một đối tượng Car mới và gọi các phương thức để hiển thị thông tin chi tiết về xe này, tính toán tuổi và kiểm tra tình trạng bảo dưỡng.

Bài tập 1

Viết chương trình cho phép nhập, xuất, khởi tạo một học sinh. Thông tin cần quan tâm về một học sinh gồm có: Mã học sinh (8 ký tự); Họ tên học sinh (30 ký tự); Điểm toán (int); Điểm văn (int).

- **Danh từ:** Học sinh
- **Động từ:**
  - Nhập một học sinh → Hàm Nhap()
  - Xuất một học sinh → Hàm Xuat()

Bài tập 2

- Xây dựng lớp biểu diễn khái niệm số phức với thành phần dữ liệu gồm có phần thực và phần ảo; Các hàm thành phần gồm có xuất, nhập, định giá trị cho số phức, cộng, trừ, nhân, chia hai số phức.
- Viết chương trình cho phép nhập vào hai số phức, in ra kết quả các phép toán cộng, trừ, nhân, chia hai số phức nhập vào.

Bài tập 3

- Thiết lập lớp biểu diễn khái niệm điểm trong mặt phẳng với hai thành phần dữ liệu là hoành độ và tung độ.
- Viết các phương thức thiết lập; các hàm thành phần cho phép thay đổi nội dung của điểm; lấy hoành độ, tung độ; tịnh tiến; nhập, xuất một điểm; hàm vẽ điểm trong chế độ đồ họa.

Bài tập 4

- Viết định nghĩa lớp TamGiac để biểu diễn khái niệm tam giác trong mặt phẳng với các phương thức thiết lập, huỷ bỏ (nếu có).

  Các hàm thành phần gồm có: nhập, xuất, tịnh tiến, quay, phóng to, thu nhỏ và vẽ tam giác.
- Viết định nghĩa lớp DaGiac để biểu diễn khái niệm đa giác trong mặt phẳng với các hàm thành phần tương tự như lớp TamGiac.
- Xây dựng lớp thời gian với thành phần dữ liệu gồm có giờ, phút, giây và các phép toán >>, << (nhập, xuất) và ++, -- (tăng/giảm 1 giây).

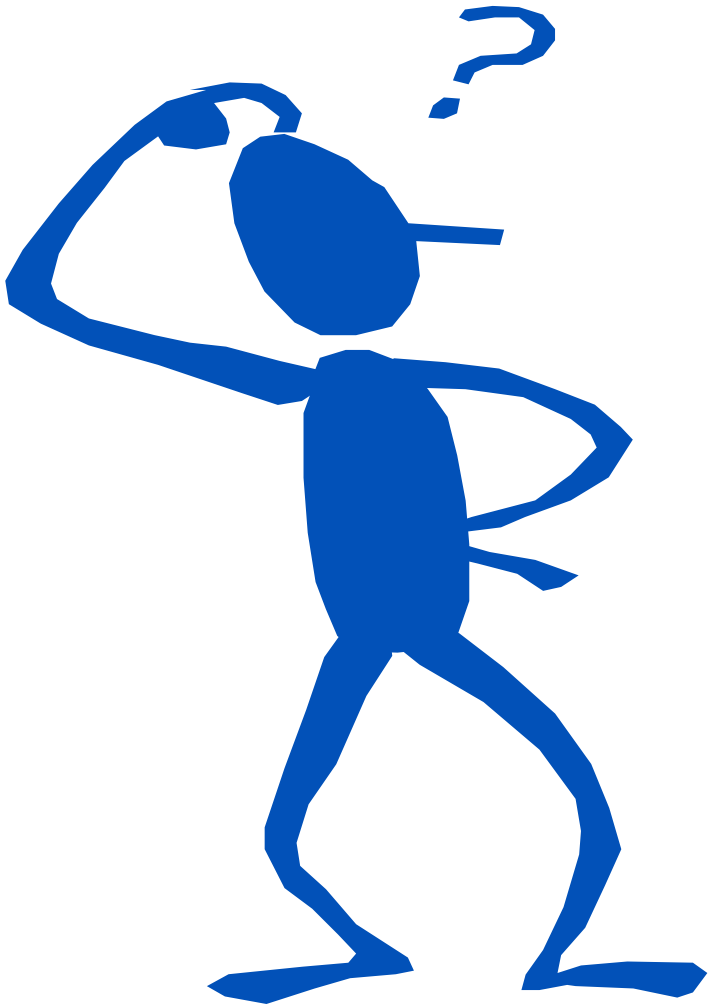
Bài tập kiểm tra trên lớp

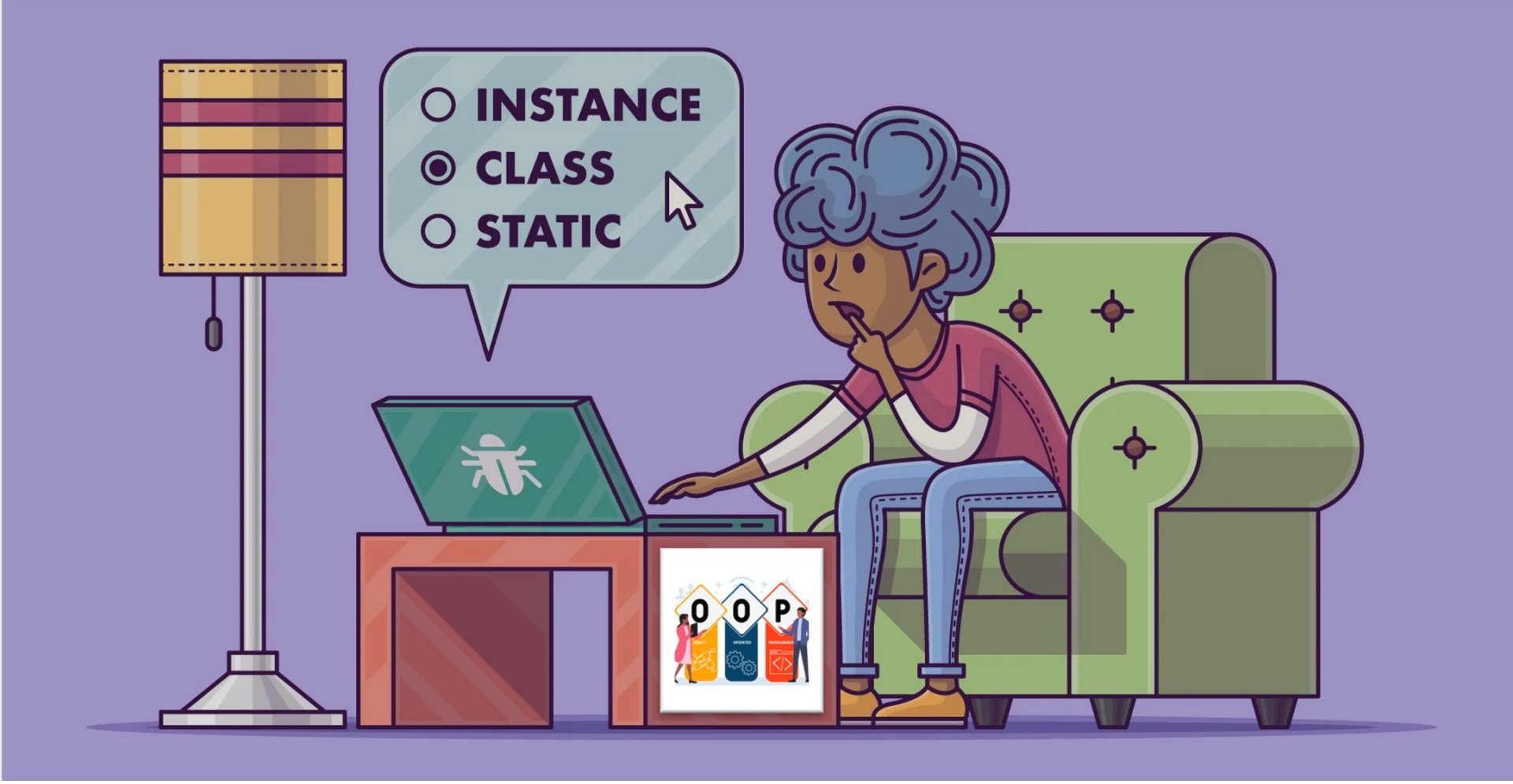
Xây dựng lớp PhanSo với hai thành phần dữ liệu là tử số và mẫu số.

Hãy cài đặt các phương thức và phép toán cần thiết để thực hiện các câu lệnh sau:

```
PhanSo a(7,2);
PhanSo b, c, kq;
cin >> b >> c;
kq = a + b + 5 + c;
cout << kq;
If (a == b)
        cout “Phan so a bang phan so b” << endl;
```

Q & A





IT002 - Lập trình hướng đối tượng