CHƯƠNG 3.
LỚP VÀ ĐỐI TƯỢNG

Khoa Công Nghệ Phần Mềm

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN, KHU PHỐ 6, PHƯỜNG LINH TRUNG, QUẬN THỦ ĐỨC, TP. HỒ CHÍ MINH

[T] 08 3725 2002 101 | [F] 08 3725 2148 | [W] www.uit.edu.vn | [E] info@uit.edu.vn

Nội dung

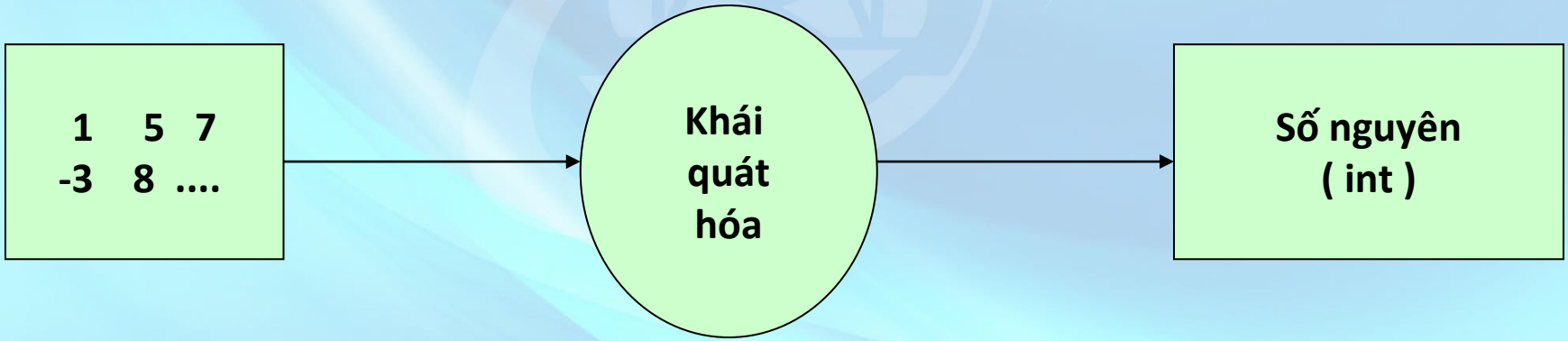
1. Giới thiệu Lớp (class)
   a. Khai báo lớp
   b. Các thành phần của lớp
   c. Cơ chế tạo lập các lớp
   d. Định nghĩa hàm thành phần
   e. Tạo lập đối tượng
   f. Phạm vi truy xuất
2. Các phương thức (Hàm - Function)
   a. Hàm thiết lập – Constructor
   b. Hàm hủy bỏ – Destructor
   c. Hàm bạn – friend function, lớp bạn – class friend
   d. Các phương thức Truy vấn - Queries, Cập nhật - Updates
3. Thành viên tĩnh – static member
4. Các nguyên tắc xây dựng lớp

Giới thiệu

- Lớp trong C++ thực chất là một kiểu dữ liệu do người sử dụng định nghĩa.
- Trong C++, dùng từ khóa class để chỉ điểm bắt đầu của một lớp sẽ được cài đặt.
- Một lớp bao gồm:
  - ✓ Thành phần dữ liệu (thuộc tính)
  - ✓ Thành phần xử lý (Phương thức/ hàm thành phần).

Lớp đối tượng - class

- 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.
- Lớp là cái để thiết kế và lập trình
- Đối tượng là cái ta tạo (từ một lớp) tại thời gian chạy.



Khai báo lớp

```
class <tên_lớp>
{
   //Thành phần dữ liệu

   //Thành phần xử lý
};
```

Khai báo lớp

```
class <tên_lớp>
{
  private: // dùng riêng trong phạm vi lớp đó
        <khai báo các thành phần riêng trong từng đối tượng>
  protected: //bảo vệ phạm vi lớp đó và các lớp con kế thừa
        <khai báo các thành phần riêng trong từng đối tượng, có thể truy cập từ lớp dẫn xuất >
  public: //dùng chung ở mọi nơi nếu đối tượng tồn tại
        <khai báo các thành phần công cộng>
};
```

Khai báo lớp

Header

```
class class_name
{

Access_Control_label:
        members;
        (data & code)

Access_Control_label :
        members;
        (data & code)
};
```

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

Các thành phần của lớp

- Thuộc tính: Các thuộc tính được khai báo giống như khai báo biến trong C
- Phương thức: Các phương thức được khai báo giống như khai báo hàm trong C. Có 2 cách khai báo:
  - Khai báo trong lớp
  - Khai báo ngoài lớp

Cơ chế tạo lập các lớp

- Xác định các thuộc tính (dữ liệu)
  - Những gì mà ta biết về đối tượng – giống như một struct
- Xác định các phương thức (hành vi)
  - Những gì mà đối tượng có thể làm
- Xác định các quyền truy xuất
  - *Sẽ trình bày sau*

Định nghĩa hàm thành phần

❖Định nghĩa các hàm thành phần ở bên ngoài lớp:

<kiểu dữ liệu trả về> <tên lớp>::<tên hàm> (<ds các tham số>)

```
{
    <nội dung >
}
```

Ví dụ:

```
void Point::Xuat()
{
    //.....
}
```

Định nghĩa hàm thành phần

```
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);
```

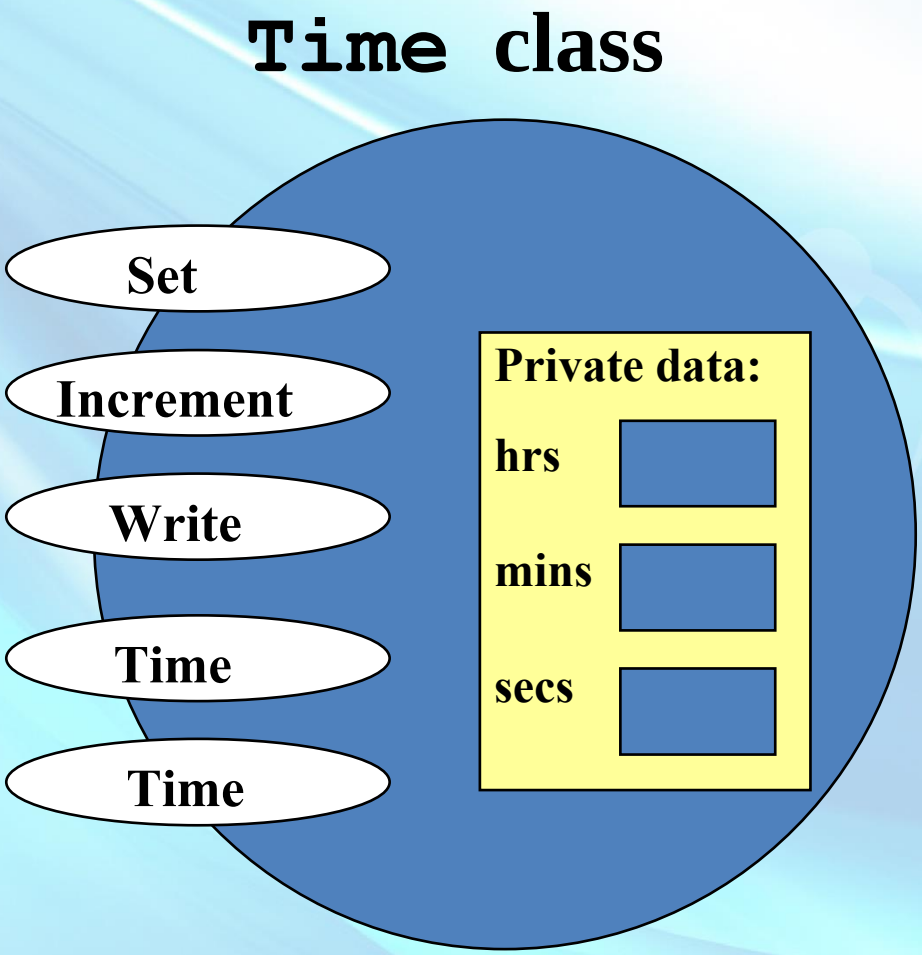
```
void Rectangle :: set (int w, int l)
{
     width = w;
     length = l;
}
```

scope operator

Ví dụ: Class Time

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

Sơ đồ mô tả lớp Time



Tạo lập đối tượng

❖Khai báo và tạo đối tượng:

<tên lớp> <tên đối tượng>;

❖Gọi hàm thành phần của lớp:

<tên đối tượng> = new <tên lớp>;

<tên đối tượng> . <tên hàm thành phần> (<ds các tham số>);

<tên con trỏ đối tượng> → <tên hàm thành phần> (<ds các tham số>);

Khai báo đối tượng

```
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

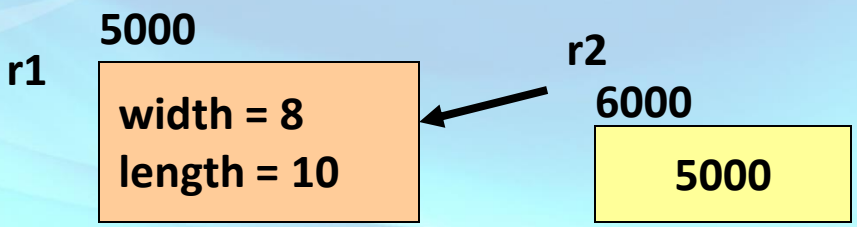
Khai báo đối tượng

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

r2 is a Pointer to a Rectangle object

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

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



Khai báo đối tượng

```
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
    - Tung độ (float)
    - Hoành độ (float)
  - Thao tác (phương thức)
    - Nhập, Xuất
    - Di chuyển theo vector (m,n)
    - Tính khoảng cách với một điểm

Viết chương trình nhập vào 2 điểm, xuất ra 2 điểm vùa nhập, khoảng cách giữa 2 điểm đó và tọa độ mới của 2 điểm khi tịnh tiến theo vector có giá trị do người dùng nhập vào.

Ví dụ

```
/*Point.h*/
#include <iostream.h>
using namespace std;
class Point
{
        private: /*khai báo các thành phần dữ  liệu riêng*/
                int x,y;
        public: /*khai báo các hàm thành phần công cộng*/
                void KhoiTao(int ox, int oy);
                void DiChuyen(int dx, int dy);
                void Xuat();
   };
```

Ví dụ

```
/*Point.cpp*/
void Point::KhoiTao(int ox, int oy)
{   cout<<"Ham thanh phan khoi tao gia tri\n";
    x = ox;
    y = oy;  /*x,y là các thành phần của đối tượng gọi hàm thành phần*/
 }
void Point::DiChuyen(int dx, int dy)
{   cout<<"Ham thanh phan di chuyen\n";
     x += dx;
     y += dy;
 }
void Point::Xuat()
{   cout<<"Ham thanh phan xuat\n";
    cout<<"Toa do: "<<x<<","<<y<<"\n";
}
```

Ví dụ

```
void main()
{
    Point p;
    p.KhoiTao(2,4); /*gọi hàm thành phần từ đối tượng*/
    p.Xuat();
    p.DiChuyen(1,2);
    p.Xuat();
 }
```

```
Ham thanh phan khoi tao gia tri
Ham thanh phan xuat
Toa do: 2,4
Ham thanh phan di chuyen
Ham thanh phan xuat
Toa do: 3,6
```

Phạm vi 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.
- private, protected và public là các từ khoá xác định phạm vi 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 đó* (*và hàm bạn, lớp bạn*).

Phạm vi truy xuất

- Trong một 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 *không cần ghi* vì C++ ngầm hiểu rằng các thành phần trước nhãn public đầu tiên là private.

Phạm vi truy xuất – Ví dụ

```
class TamGiac
{  private:
        float a,b,c;   //độ dài ba cạnh
        int LayLoai();   //cho biết kiểu của tam giác: 1-d,2-vc,3-c,4-v,5-t
        float TinhDienTich();   //tính diện tích của tam giác

  public:
        void Nhap();  //*nhập vào độ dài ba cạnh
        void Xuat();   //in ra các thông tin liên quan đến tam giác
  };
```

Tham số hàm thành phần

```
void Point::KhoiTao (int xx, int yy)
{
        x = xx;
        y = yy;  //x, y la thanh phan cua lop Point
}
```

- Hàm thành phần có quyền truy cập đến các thành phần private của đối tượng gọi nó.

Tham số hàm thành phần

- Hàm thành phần có quyền truy cập đến:
  - Các thành phần private của các đối tượng,
  - tham chiếu đối tượng
  - con trỏ đối tượng

  có cùng kiểu lớp khi được dùng là tham số hình thức của nó.

Tham số hàm thành phần

```
int KiemTraTrung(Point pt)
{
        return (x == pt.x && y == pt.y);
}
int KiemTraTrung(Point *pt)
{
        return (x == pt→x && y == pt→y);
}
int KiemTraTrung(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 hiện tại dùng làm tham số ngầm định cho hàm thành phần.
- Con trỏ this tham chiếu đến đối tượng (*hiện tại của lớp*) đang gọi hàm thành phần.
- Ví dụ1:

```
int KiemTraTrung(Point pt)
{
        return (this → x == pt.x && this → y == pt.y);
}
```

Ví dụ2:

```
#include <iostream>
using namespace std;
class NhanVien {
    int msnv;
    string ten;
    int tuoi;
    public:
      NhanVien(int msnv, string ten, int tuoi) {
          cout << "Trong ham xay dung: " << endl;
          cout << "    msnv: " << msnv << endl;
          cout << "    ten: " << ten << endl;
          cout << "    Tuoi: " << tuoi << endl;
this-> msnv = msnv;
this-> ten = ten;
this-> tuoi = tuoi; }
```

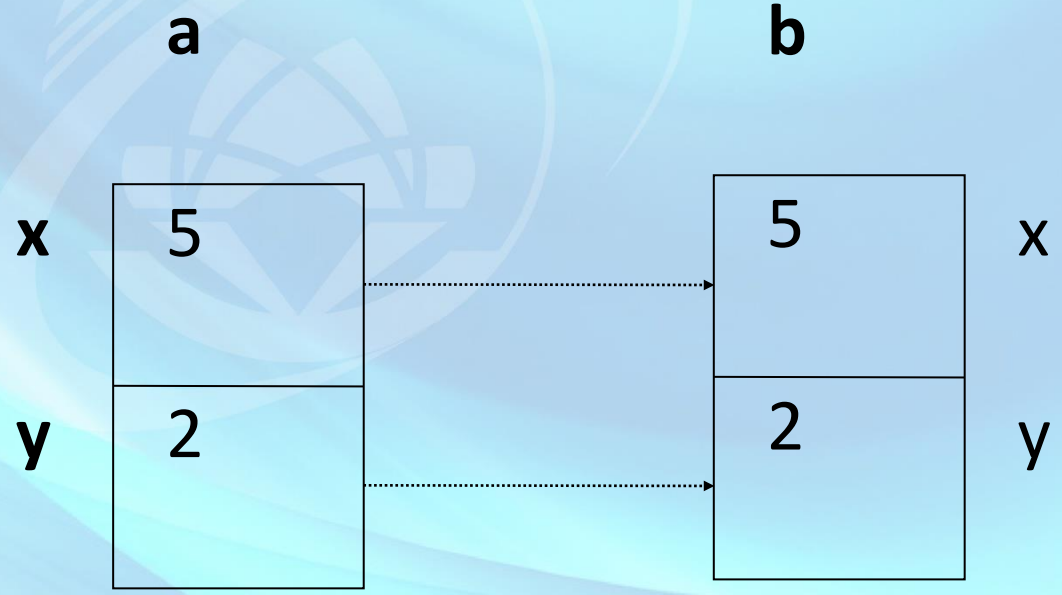
```
  void HienThi() {
      cout << "Ham in thong tin cua doi tuong nhan
vien: \n ";
      cout << ten << endl;
      cout << "Ma so nhan vien:" << msnv << endl;
      cout << " Tuoi: " << tuoi << endl;
      }
};
 int main() {
   NhanVien  n1=NhanVien(111231,  “Nguyen  Van
A", 25);
    n1.Hien
    return 0
}
```

```
Trong ham xay dung:
   msnv: 111231
   ten: Nguyen Van A
   Tuoi: 25
Ham in thong tin cua doi tuong nhan vien:
Nguyen Van A
   Ma so nhan vien: 111231
   Tuoi: 25
```

Phép gán đối tượng

- Là việc sao chép giá trị các thành phần dữ liệu từ đối tượng a sang đối tượng b tương ứng từng đôi một
- Ví dụ:

```
Point a, b;
a.KhoiTao(5,2);
b = a;
```



Hàm thiết lập – Constructor

- Trong hầu hết các thuật giải, để giải quyết một vấn đề →thường phải thực hiện các công việc:
  - Khởi tạo giá trị cho biến, cấp phát vùng bộ nhớ của biến con trỏ, mở tập tin để truy cập,…
  - Hoặc khi kết thúc, chúng ta phải thực hiện quá trình ngược lại như: thu hồi vùng bộ nhớ đã cấp phát, đóng tập tin,…
- Các ngôn ngữ OOP có các phương thức để thực hiện công việc này một cách *"tự động"* gọi là *phương thức thiết lập* và *phương thức hủy bỏ*.

Hàm thiết lập – Constructor

- Constructor là một loại phương thức đặc biệt dùng để khởi tạo thể hiện của lớp.
- Bất kỳ một đối tượng nào được khai báo đều phải sử dụng một hàm thiết lập để khởi tạo các giá trị thành phần của đối tượng.
- Hàm thiết lập được khai báo giống như một phương thức với tên phương thức trùng với tên lớp và không có giá trị trả về *(kể cả void)*.
- Constructor phải có phạm vi là public.

Hàm thiết lập – Constructor

- Constructor có thể được khai báo chồng như các hàm C++ thông thường (*có thể có nhiều hàm thiết lập trong một lớp).*

- Constructor có thể được khai báo với các tham số có giá trị ngầm định *(tham số mặc nhiên)*

Ví dụ

```
class Point{
    int x, y;                                   /*các thành phần dữ liệu*/
    public:
        Point() { x = 0; y = 0; }               /*Hàm thiết lập mặc định*/
        Point(int ox, int oy) { x = ox; y = oy; } /* Hàm thiết lập gán giá trị*/
        void DiChuyen(int dx, int dy);
        void Xuat();
};
Point a(5,2);       //ok
Point b;            //ok
Point c(3);         //ok?
```

← Err do không tìm được hàm tương ứng

- *Để khắc phục, có thể gán trị đầu cho biến oy của hàm* Point(int ox, int oy=1)

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 constructor khác (overload)
  - MyClass x(5);
  - MyClass* p = new MyClass(5);

Constructor mặc định

- Đối với constructor mặc định, nếu ta không cung cấp bất kỳ 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.

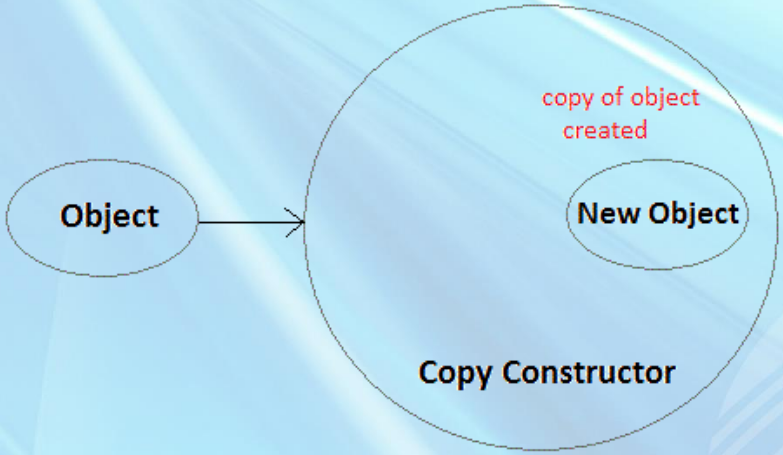
Ví dụ

```
class Point{
    /*Khai báo các thành phần dữ liệu*/
    int x, y;
    public:
        Point(int ox, int oy = 1){ x = ox; y = oy;}
        void DiChuyen(int dx, int dy);
        void Xuat();
};
Point a(5,2);        //ok?
Point b;             //ok?
Point c(3);          //ok?
```

Copy constructor

- Chúng ta có thể tạo đối tượng mới giống đối tượng cũ một số đặc điểm, không phải hoàn toàn như phép gán bình thường, hình thức "giống nhau" được định nghĩa theo quan niệm của người lập trình. Để làm được vấn đề này, trong các ngôn ngữ OOP cho phép ta xây dựng phương thức thiết lập sao chép.
- Đây là phương thức thiết lập có tham số là tham chiếu đến đối tượng thuộc chính lớp này.
- Trong phương thức thiết lập sao chép có thể ta chỉ sử dụng một số thành phần nào đó của đối tượng ta tham chiếu →"gần giống nhau"

Copy constructor




```
Foo(const Foo& existingFoo);
```

tham số là đối tượng được sao chép

Kiểu tham số là tham chiếu đến đối tượng kiểu Foo

từ khoá const được dùng để đảm bảo đối tượng được sao chép sẽ không bị sửa đổi

Hàm hủy bỏ – Destructor

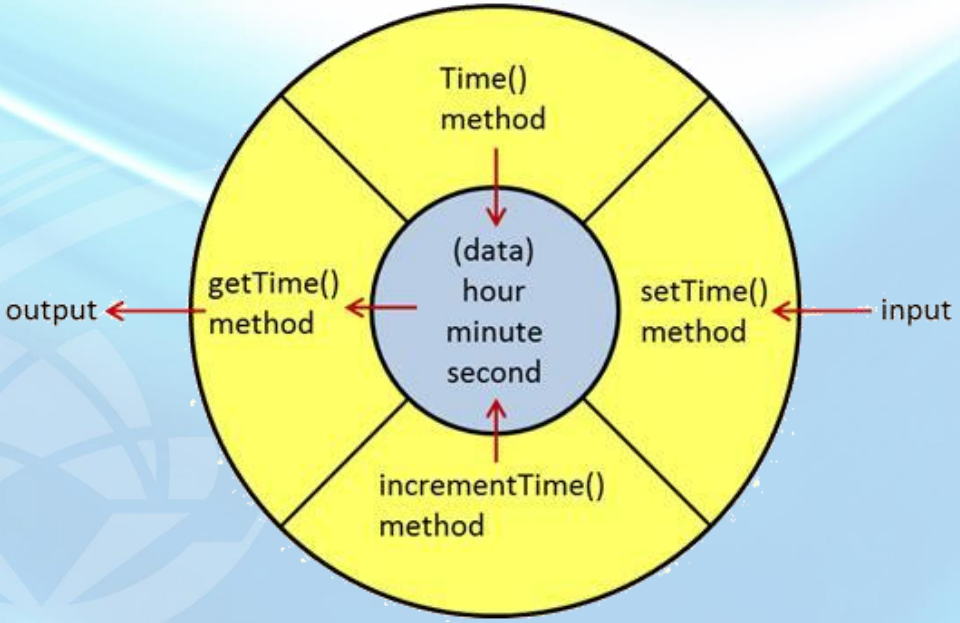
- Destructor, được gọi ngay trước khi một đối tượng bị thu hồi.
- Destructor thường được dùng để thực hiện việc dọn dẹp cần thiết trước khi một đối tượng bị hủy.
- Một lớp chỉ có duy nhất một Destructor
- Phương thức Destructor trùng tên với tên lớp nhưng có dấu ~ đặt trước
- Được tự động gọi thực hiện khi đối tượng hết phạm vi sử dụng.
- Destructor phải có thuộc tính public

Ví dụ

```
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 hủy bỏ
    void Xuat();
};
```

Thao tác với dữ liệu private

- Khi muốn truy xuất dữ liệu private từ các đối tượng thì phải làm thế nào?
- Khi muốn cập nhật dữ liệu private từ các đối tượng thì phải làm thế nào?



Phương thức Truy vấn

- 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 > 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.

Phương thức Truy vấn

- Đối với các truy vấn đơn giản, quy ước đặt tên phương thức như sau: Tiền tố "Get", tiếp theo là tên của thành viên cần truy vấn
  - int GetX(); *// LayX();*
  - int GetSize(); *// LaySize();*
- *Các loại truy vấn khác nên có tên có tính mô tả.*
- Truy vấn điều kiện nên có tiền tố "is" *(la)*

Phương thức Cập nhật

- Thường để 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 là gán một giá trị nào đó cho một thành viên dữ liệu.
- Đối với dạng cập nhật đơn giản, quy ước đặt tên như sau: Dùng tiền tố "Set" kèm theo tên thành viên cần sửa.
  - int SetX(int); *// CapnhatX(int);*

Truy vấn và Cập nhật

- Nếu phương thức Get/Set chỉ có nhiệm vụ cho ta đọc/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 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.
  - Chỉ cho phép các dữ liệu có thể truy vấn hay thay đổi mới được truy cập đến.

Ví dụ

```
int Student::SetGPA (double NewGPA) // hàm cập nhật điểm GPA
{
    if ((NewGPA >= 0.0) && (NewGPA <= 10.0)){
        this->GPA= NewGPA;
        return 0; // Return 0 to indicate success
    }
    else
    {
        return -1; // Return -1 to indicate failure
    }
}
```

Thành viên tĩnh – static member

- Trong C, static xuất hiện trước dữ liệu được khai báo trong một hàm nào đó thì giá trị của dữ liệu đó vẫn được lưu lại như một biến toàn cục.
- Trong C++, nếu static xuất hiện trước một dữ liệu hoặc một phương thức của lớp thì giá trị của nó vẫn được lưu lại và có ý nghĩa cho đối tượng khác của cùng lớp này.
- Các thành viên static có thể là public, private hoặc protected.

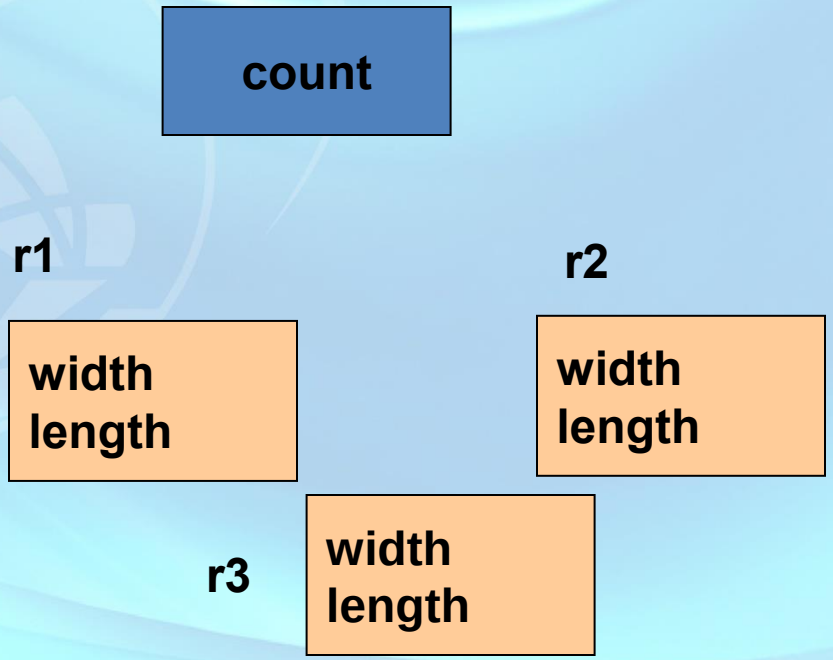
Thành viên tĩnh – static member

- 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.

Ví dụ

```
class Rectangle
{
    private:
        int width;
        int length;
        static int count;
    public:
        Rectangle(){count++;}
        void Set(int w, int l);
        int GetArea();
}
```

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



Ví dụ

- Đếm số đối tượng MyClass:

```
class MyClass{
    public:
        MyClass();
        ~MyClass();
        void PrintCount();
    private:
        static int count;
};
```

Ví dụ

```
int MyClass::count = 0;
MyClass::MyClass(){
    this → count++;
}
MyClass::~MyClass() {
    this → count--;
}
void MyClass::PrintCount(){
    cout << “There are currently ” << this → count << “ instance(s)
    of MyClass.\n”;
}
```

Ví dụ

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

Thành viên tĩnh – static member

- Phương thức static?
  - Đối với các phương thức static, ngoài ý nghĩa tương tự với dữ liệu, còn có sự khác biệt cơ bản đó là việc cho phép truy cập đến các phương thức static khi chưa khai báo đối tượng *(thông qua tên lớp)*

Thành viên tĩnh – static member

- Các thành viên lớp tĩnh public có thể được truy cập thông qua bất kỳ đối tượng nào của lớp đó, hoặc chúng có thể được truy cập thông qua tên lớp sử dụng toán tử định phạm vi.
- Các thành viên lớp tĩnh private và protected phải được truy cập thông qua các hàm thành viên public của lớp hoặc thông qua các friend của lớp.
- Các thành viên lớp tĩnh tồn tại ngay cả khi đối tượng của lớp đó không tồn tại.

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

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

Bài tập