

# Kỹ thuật lập trình

## Phần III: Lập trình hướng đối tượng

---

### Chương 5: Lớp và đối tượng



# Nội dung chương 5



- 5.1 Khái niệm
- 5.2 Từ cấu trúc sang lớp
- 5.3 Biến thành viên
- 5.4 Hàm thành viên
- 5.5 Kiểm soát truy nhập

# 5.1 Khái niệm

## Đối tượng là gì?

- Thực thể phần mềm
- Mô hình/đại diện của một đối tượng vật lý:
  - Tank, Heater, Furnace
  - Motor, Pump, Valve
  - Sensor, Thermometer, Flowmeter
  - Control Loop, Control System
- Hoặc một đối tượng logic ("conceptual object):
  - Trend, Report, Button, Window
  - Matrix, Vector, Polynomial

# Một đối tượng có...

- ➔ Các thuộc tính (attributes)
- ➔ Trạng thái (state)
  - Dữ liệu
  - Quan hệ
- ➔ Hành vi (behavior)
  - Các phép toán
  - Đặc tính phản ứng
- ➔ Căn cước (identity)
- ➔ Ngữ nghĩa/trách nhiệm (semantic/responsibilities)



# Lớp là gì?

- Một lớp là thực thi của các đối tượng có chung
  - Ngữ nghĩa
  - Thuộc tính
  - Quan hệ
  - Hành vi
- Lớp = Đóng gói [Cấu trúc dữ liệu + hàm thao tác]
  - Lớp các vector, lớp các ma trận (dữ liệu phần tử + các phép truy nhập và phép toán cơ bản)
  - Lớp các hình chữ nhật (các dữ liệu tọa độ + phép vẽ, xóa,...)
  - Lớp các mô hình hàm truyền (các hệ số đa thức tử/mẫu, các phép toán xác định tính ổn định, xác định các điểm cực,...)
- Các dữ liệu của một lớp => biến thành viên
- Các hàm của một lớp => hàm thành viên
- Các biến của một lớp => một đối tượng, một thể nghiệm

# Lập trình hướng đối tượng (object-oriented programming, OOP)

- Trừu tượng hóa (*abstraction*): giúp đơn giản hóa vấn đề, dễ sử dụng lại
- Đóng gói dữ liệu/che dấu thông tin (*data encapsulation/information hiding*): nâng cao giá trị sử dụng lại và độ tin cậy của phần mềm
- Dẫn xuất/thừa kế (*subtyping/inheritance*): giúp dễ sử dụng lại mã phần mềm và thiết kế
- Đa hình/đa xạ (*polymorphism*): giúp phản ánh trung thực thế giới thực và nâng cao tính linh hoạt của phần mềm

*Phương pháp luận hướng đối tượng cho phép tư duy ở mức trừu tượng cao nhưng gần với thế giới thực!*

## 5.2 Từ cấu trúc sang lớp

```
struct Time {
    int hour; // gio
    int min;  // phut
    int sec;  // giay
};

void addHour(Time& t, int h) {
    t.hour += h;
}

void addMin(Time& t, int m) {
    t.min += m;
    if (t.min > 59) {
        t.hour += t.min/60;
        t.min %= 60;
    }
    else if (t.min < 0) {
        t.hour += (t.min/60 - 1);
        t.min = (t.min % 60) + 60;
    }
}
```

```
void addSec(Time& t, int s) {
    t.sec += s;
    if (t.sec > 59) {
        addMin(t, t.sec/60);
        t.sec %= 60;
    }
    else if (t.sec < 0) {
        addMin(t, t.sec/60 - 1);
        t.sec = (t.sec % 60) + 60;
    }
}

void main() {
    Time t = {1, 0, 0};
    addMin(t, 60);
    addMin(t, -5);
    addSec(t, 25);
    ...
}
```

# Một số vấn đề của cấu trúc

- Truy nhập dữ liệu trực tiếp, không có kiểm soát có thể dẫn đến không an toàn

```
Time t1 = {1, 61, -3};           // ??!  
Time t2;                          // Uncertain values  
int h = t2.hour;                  // ??!  
int m = 50;  
t2.min = m + 15;                  // ??!
```

- Không phân biệt giữa “chi tiết bên trong” và “giao diện bên ngoài”, một thay đổi nhỏ ở chi tiết bên trong cũng bắt người sử dụng phải thay đổi mã sử dụng theo!

Ví dụ: cấu trúc Time được sửa lại tên biến thành viên:

```
struct Time {  
    int h, m, s;  
};
```

Đoạn mã cũ sẽ không biên dịch được:

```
Time t;  
t.hour = 5;
```



# Đóng gói hay "lớp hóa"

```
class Time {  
    int hour; // gio  
    int min;  // phut  
    int sec;  // giay
```

Biến thành viên  
(member variable)

```
public:
```

```
    Time() {hour=min=sec=0;}
```

Hàm tạo (constructor)

```
    void setTime(int h, int m, int s)  
    {  
        hour = h;  
        min = sec = 0;  
        addSec(s);  
        addMin(m);  
    }
```

```
    int getHour() { return hour; }  
    int getMin()  { return min;  }  
    int getSec()  { return sec;  }
```

Hàm thành viên  
(member functions)

```
    void addHour(int h) { hour += h; }  
    ...
```

```

void addMin(int m) {
    min += m;
    if (min > 59) {
        hour += min/60;
        min %= 60;
    }
    else if (min < 0) {
        hour += (min/60 - 1);
        min = (min % 60) + 60;
    }
}

void addSec(int s) {
    sec += s;
    if (sec > 59) {
        addMin(sec/60);
        sec %= 60;
    }
    else if (sec < 0) {
        addMin(sec/60 - 1);
        sec = (sec % 60) + 60;
    }
}
};

```

```

void main()
{
    Time t;
    t.addHour(1);
    t.addMin(60);
    t.addMin(-5);
    t.addSec(25);
    t.hour = 1; // error
    t.min = 65; // error
    t.sec = -3; // error
    t.setTime(1, 65, -3);
    int h = t.getHour();
    int m = t.getMin();
    int s = t.getSec();
}

```

## 5.3 Biến thành viên

- Khai báo biến thành viên của một lớp tương tự như cấu trúc

```
class Time {  
    int hour, min, sec;  
    ...  
};
```

- Mặc định, các biến thành viên của một lớp không truy nhập được từ bên ngoài (biến riêng), đương nhiên cũng không khởi tạo được theo cách cổ điển:

```
Time t = {1, 0, 0};    // error!  
t.hour = 2;            // error!
```

- Có thể làm cho một biến thành viên truy nhập được từ bên ngoài (biến công cộng), tuy nhiên ít khi có lý do cần làm như thế:

```
class Point {  
    public:  
        int x,y;  
};
```

- Kiểm soát việc truy nhập các biến riêng thông qua các hàm thành viên
- Cách duy nhất để khởi tạo giá trị cho các biến thành viên là sử dụng hàm tạo:

```
class Time {  
    ...  
public:  
    Time() {hour=min=sec=0;}  
};  
Time t;    // t.hour = t.min = t.sec = 0;
```

- Một số biến thành viên có vai trò lưu trữ trạng thái bên trong của đối tượng, không nên cho truy nhập từ bên ngoài (ngay cả gián tiếp qua các hàm)

```
class PID {  
    double Kp, Ti, Td;    // controller parameters  
    double I;            // internal state  
    ...  
};
```

## 5.4 Hàm thành viên

### Định nghĩa cấu trúc & hàm

```
struct Time {  
    int hour, min, sec  
};  
void addHour(Time& t, int h) {  
    t.hour += h;  
}  
...
```

### Định nghĩa lớp

```
class Time {  
    int hour, min, sec;  
public:  
    void addHour(int h) {  
        hour += h;  
    }  
    ...  
};
```

### Gọi hàm với biến cấu trúc

```
Time t;  
...  
addHour(t, 5);
```

### Gọi hàm thành viên của DT

```
Time t;  
...  
t.addHour(5);
```

Ở đây có sự khác nhau về cách viết,  
nhưng chưa có sự khác nhau cơ bản

# Khai báo và định nghĩa hàm thành viên

- Thông thường, lớp cùng các hàm thành viên được **khai báo** trong tệp tin đầu (\*.h). Ví dụ trong tệp có tên “mytime.h”:

```
class Time {  
    int hour,min,sec;  
public:  
    void addHour(int h);  
    void addMin(int m);  
    void addSec(int s);  
    ...  
};
```

- Các hàm thường được **định nghĩa** trong tệp tin nguồn (\*.cpp):

```
#include "mytime.h"  
...  
void Time::addHour(int h) {  
    hour += h;  
}
```

- Có thể định nghĩa một hàm thành viên trong tệp tin đầu dưới dạng một hàm inline (chỉ nên áp dụng với hàm đơn giản), ví dụ:

```
inline void Time::addHour(int h) { hour += h; }
```

- Một hàm thành viên cũng có thể được định nghĩa trong phần khai báo lớp => mặc định trở thành hàm inline, ví dụ

```
class Time {  
    int hour,min,sec;  
public:  
    void addHour(int h) { hour += h; }  
};
```

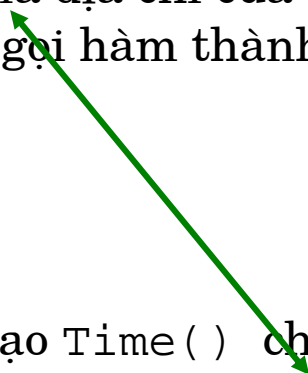
- Khi định nghĩa hàm thành viên, có thể sử dụng các biến thành viên và gọi hàm thành viên khác mà **không cần** (thậm chí không thể được) đưa tên biến đối tượng, ví dụ:

```
void Time::addSec(int s) {  
    ...  
    addMin(sec/60);  
    ...  
}
```

# Bản chất của hàm thành viên?

```
class Time {
    int hour,min,sec;
public:
    Time() { hour=min=sec=0; }
    void addHour(int h) {
        this->hour += h; // con trỏ this chính là địa chỉ của
    }                // đối tượng gọi hàm thành viên
    ...
};

void main() {
    Time t1,t2;           // Tự động gọi hàm tạo Time() cho t1 và t2
    t1.addHour(5);        // Có thể hiểu như là addHour(&t1,5);
    t2 = t1;              // OK
    t2.addHour(5);        // Có thể hiểu như là addHour(&t2,5);
    ...
}
```





## 5.5 Kiểm soát truy nhập

- **public:** Các thành viên công cộng, có thể sử dụng được từ bên ngoài
- **private:** Các thành viên riêng, không thể truy nhập được từ bên ngoài, ngay cả trong lớp dẫn xuất (sẽ đề cập sau)  

```
class Time {  
    private:  
        int hour,min,sec;  
        ...  
};
```
- Mặc định, khi đã khai báo **class** thì các thành viên là **private**.
- **protected:** Các thành viên được bảo vệ, không thể truy nhập được từ bên ngoài, nhưng truy nhập được các lớp dẫn xuất (sẽ đề cập sau)

## 5.6 Con trỏ đối tượng

```
#include "mytime.h"
void main() {
    Time t;                // call constructor Time()
    t.addHour(5);
    Time *pt = &t;         // pt is identical to this pointer
    pt->addSec(70);
    pt = new Time;          // call constructor Time()
    pt->addMin(25);
    ...
    delete pt;
    pt = new Time[5]; // call constructor 5 times
    for (int i=0; i < 5; ++ i)
        pt[i].addSec(10);
    ...
    delete [] pt;
}
```

# Bài tập về nhà

- Dựa trên cấu trúc Vector và các hàm liên quan đã thực hiện trong chương 4, hãy xây dựng lớp đối tượng Vector với các hàm thành viên cần thiết.
- Khai báo một lớp thực hiện lưu trữ thông tin của một lớp sinh viên gồm những thành phần thuộc tính như sau:
  - Số hiệu sinh viên : Kiểu số nguyên
  - Họ và tên: Chuỗi ký tự
  - Năm sinh: Kiểu số nguyên
- Khai báo và định nghĩa mở rộng lớp quản lý sinh viên bằng các hàm thành viên thực hiện các chức năng như sau:
  - Nhập họ tên sinh viên
  - Nhập số hiệu sinh viên
  - Nhập năm sinh
  - Tìm và hiển thị thông tin sinh viên khi biết mã số