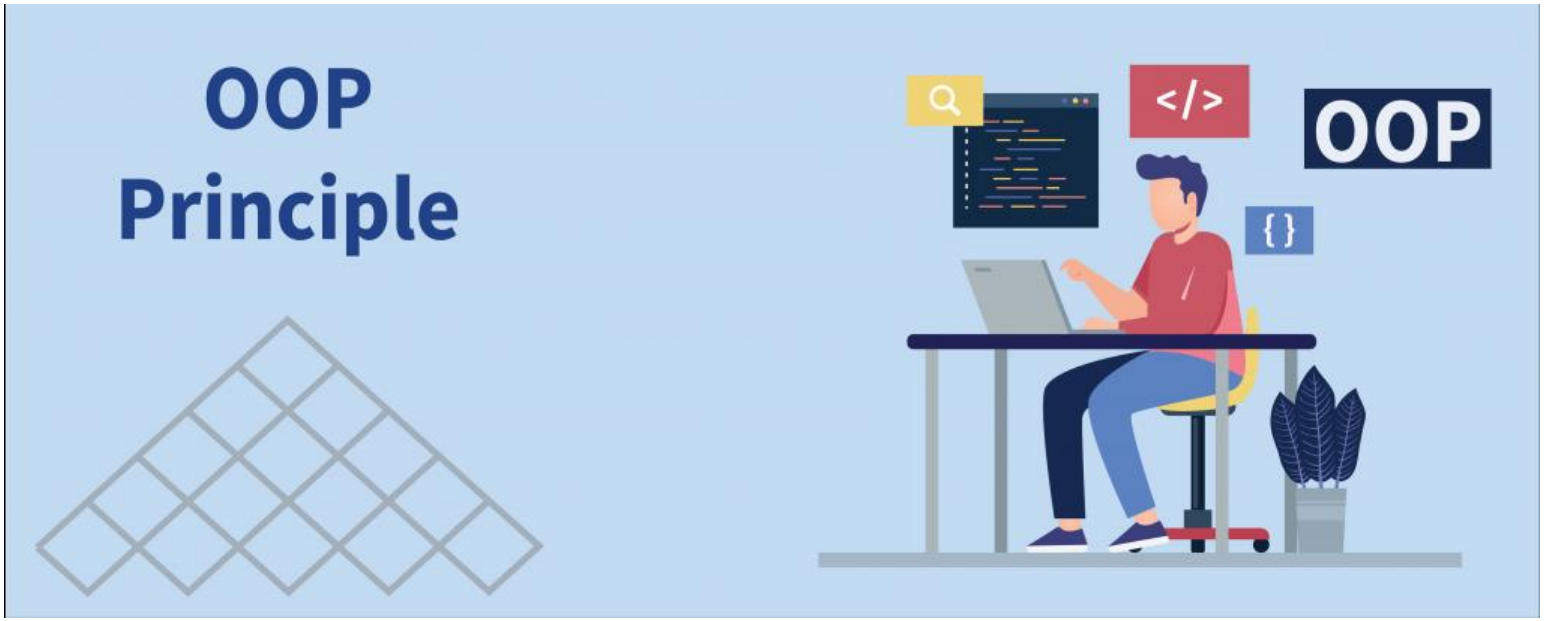



# KẾ THỪA

Ths. Đinh Thị Thúy – AI LAB/TLU

Thuydt@thanglong.edu.vn – PAI006

# NỘI DUNG

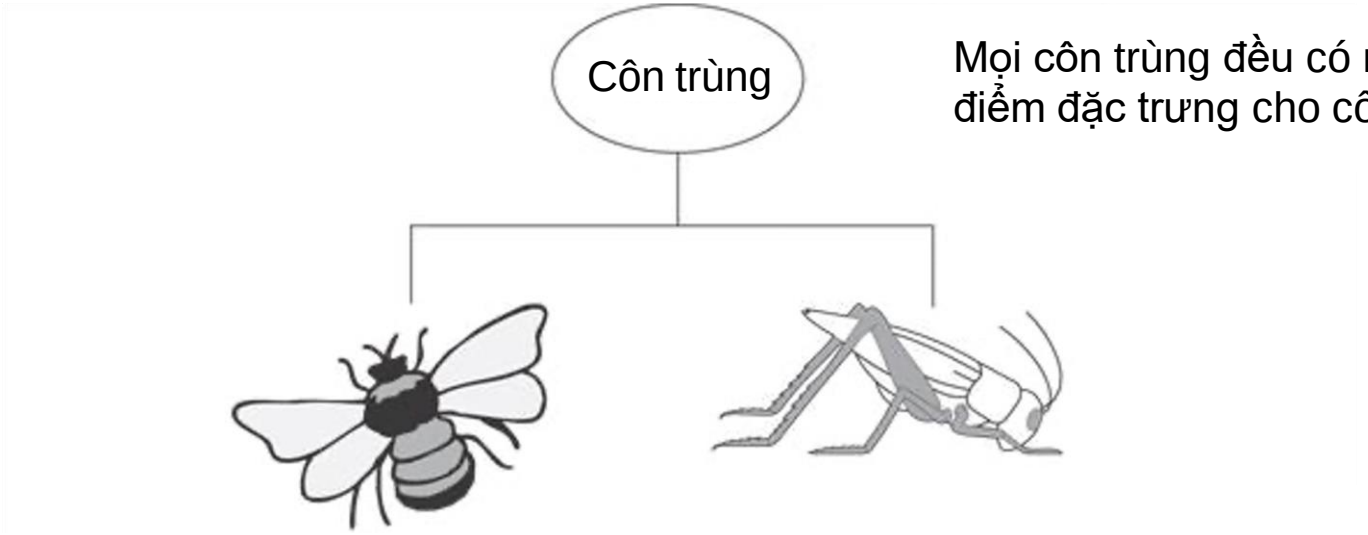
- Sự kế thừa, lớp cha, lớp con
- Đặc tả truy cập thành viên và lớp cơ sở
- Hàm tạo, hàm huỷ
- Tái định hàm lớp cha
- Xây dựng cây phả hệ
- Hàm ảo, ghi đè, đa hình

# SỰ KẾ THỪA, LỚP CHA, LỚP CON

- Sự kế thừa - Quan hệ “is a”
- Lớp con có các thành viên nào?

# VD: CÔN TRÙNG

- Có nhiều loài là tập con chuyên biệt hơn của loài khác




Mọi côn trùng đều có những điểm đặc trưng cho côn trùng

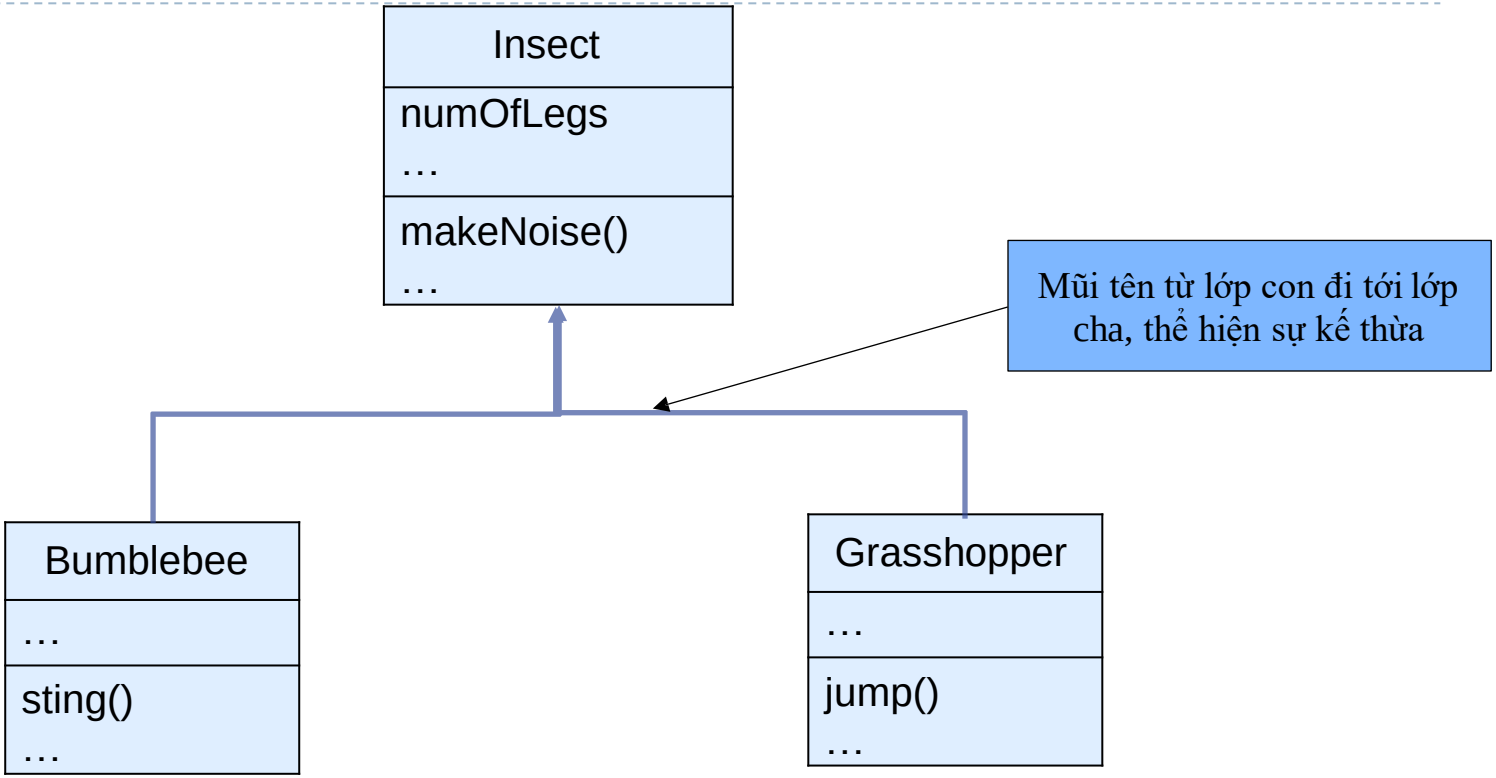
Ngoài các điểm chung của côn trùng, ong nghệ còn có những đặc trưng của riêng nó, vd như biết đốt

Ngoài các điểm chung của côn trùng, châu chấu còn có những đặc trưng của riêng nó, vd như biết nhảy

# SỰ KẾ THỪA – QUAN HỆ "IS A"

- Xảy ra khi lớp này có quan hệ "là một" với lớp khác:
  - Chó xù là một con chó => Poodle kế thừa Dog
  - Xe hơi là một phương tiện => Car kế thừa Vehicle
  - Cầu thủ là một vận động viên => FootballPlayer kế thừa Athlete
- Lớp kế thừa sẽ mở rộng thêm cho lớp bị kế thừa
  - Poodle có mọi đặc điểm hành vi của Dog cộng có thêm lông xù
- Giúp tạo ra lớp mới từ lớp sẵn có nhờ thêm vào/thay thế thành viên
- Lớp sẵn có gọi là lớp cha hay lớp cơ sở, là lớp tổng quát hơn
- Lớp mới gọi là lớp con hay lớp dẫn xuất, là lớp chuyên biệt hơn

# LỚP CON CÓ GÌ?




- Có mọi thành viên của lớp cha, mà không cần viết lại
- Cộng với mọi thành viên được định nghĩa trong lớp con

# VD LỚP CON CÓ GÌ?

```
class Insect{
    string color;
public:
    void setColor(string);
    string getColor() const;
    void makeNoise() const;
};
class Grasshopper: public Insect{
    int highestJump;
public:
    int jump();
    int getHighestJump() const;
};
```

Có mọi thành viên lớp cha

Lẫn mọi thành viên lớp con

Grasshopper khai báo kế thừa từ Insect

- Khai báo kế thừa ở ngay sau tên lớp con:

  class tên_lớp_con : từ_khoá_kế_thừa tên_lớp_cha

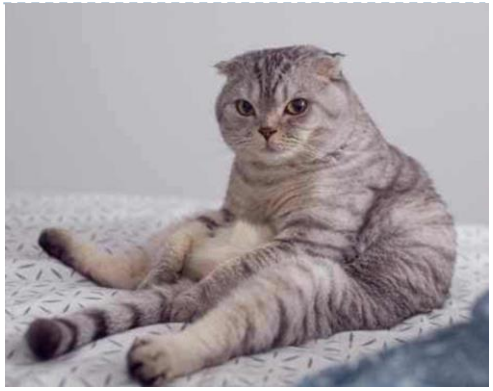
  - Từ khoá kế thừa: public/protected/private (~ nói sau)

```
class Insect{
    string color;
public:
    void setColor(string s)
    { color = s; }
    string getColor() const
    { return color; }
    void makeNoise() const {
        cout << "I'm an insect. I\'m in "
        << color << ". \n";
    }
};
class Grasshopper: public Insect{
    int highestJump;
public:
    int jump(){
        int tmp = rand()%200;
        if (tmp > highestJump)
            highestJump = tmp;
        return tmp;
    }
    int getHighestJump() const
    { return highestJump; };
};
```

```
int main(){
    srand(time(0));
    Insect obj1;
    obj1.setColor("red");
    cout << obj1.getColor() << endl;
    obj1.makeNoise();
    Grasshopper obj2;
    obj2.setColor("brown green");
    cout << obj2.getColor() << endl;
    obj2.makeNoise();
    obj2.jump();
    cout << obj2.getHighestJump()
        << " cm!\n";
    obj2.jump();
    cout << obj2.getHighestJump()
        << " cm!\n";
    return 0;
}
```

```
red
I'm an insect. I'm in red.
brown green
I'm an insect. I'm in brown green.
47 cm!
176 cm!
```

# VD: CAT VÀ SCOTTISHFOLD



- Lớp Cat đang có các thành viên chính sau:
  - tên, tuổi, cân nặng
  - setter getter tương ứng
  - các hàm tạo
  - hàm meow
- Viết lớp ScottishFold mô tả mèo tai cụp kế thừa lớp Cat và bổ sung:
  - biến mô tả kiểu lông bằng một xâu, có setter getter phù hợp
  - hàm doBuddhaPose() in ra “Purrr, I am doing a Buddha pose!”;
- Tạo ra một con mèo tai cụp, lông ngắn, đặt tên Tony Stark, 2 tuổi, 4 cân, bảo Tony kêu meow, ngồi kiểu Buddha. Cuối cùng in ra mọi thông tin của nó.

# ĐẶC TẢ TRUY CẬP

- Thành viên kế thừa được truy cập thế nào?
- Đặc tả truy cập thành viên: public, protected, private
- Đặc tả truy cập lớp cơ sở: public, protected, private

# THÀNH VIÊN KẾ THỪA ĐƯỢC TRUY CẬP THẾ NÀO?

```
class Insect
private:
    string color;
public:
    void setColor(string);
    string getColor() const;
    void makeNoise() const;
```

```
class Grasshopper
private:
    int highestJump;
public:
    int jump();
    int getHighestJump() const;
```

```
class Grasshopper : public Insect
??
    string color;
??
    void setColor(string);
    string getColor() const;
    void makeNoise() const;

private:
    int highestJump;
public:
    int jump();
    int getHighestJump() const;
```

- Khi kế thừa ở lớp con, các thành viên lớp cha sẽ có mức độ truy cập gì?

# THÀNH VIÊN KẾ THỪA ĐƯỢC TRUY CẬP THẾ NÀO?

1) Đặc tả truy cập của thành viên cha khi ở lớp con phụ thuộc vào đặc tả truy cập của chính thành viên đó khi ở lớp cha, lẫn vào đặc tả truy cập của lớp con với lớp cha

2) Gọi là đặc tả truy cập thành viên và đặc tả truy cập lớp cơ sở

```
class Grasshopper: public Insect{
    int highestJump;
public:
    int jump();
    int getHighestJump() const;
    void printColor() const{
        cout << "My Dad is in "
          << color << endl; // is it OK?
    }
};
```

```
int main(){
    Insect obj1;
    obj1.setColor("red");
    cout << obj1.getColor() << endl;
    obj1.makeNoise();
    obj1.color;  // not OK
    Grasshopper obj2;
    obj2.setColor("brown green");
    cout << obj2.getColor() << endl;
    obj2.makeNoise();
    obj2.color; // also not OK ??
```

# 1. ĐẶC TẢ TRUY CẬP THÀNH VIÊN

- Xác định thành viên đó truy cập được bởi mã ở đâu?
- Thành viên private: chỉ truy cập được từ mã trong lớp đó
  - Mã ngoài lớp đó, gồm cả các hàm lớp con cũng không thể truy cập
  - Chỉ truy cập được gián tiếp thông qua các hàm protected hay public
- Thành viên protected: truy cập được từ mã trong lớp đó lẫn lớp con
  - Mã ngoài lớp đó và lớp con không thể truy cập trực tiếp
- Thành viên public: truy cập được từ mọi nơi
- Lớp con kế thừa mọi thành viên, song chỉ truy cập được các thành viên protected hay public

# 2. ĐẶC TẢ TRUY CẬP LỚP CƠ SỞ

- Xác định mức độ lớp dẫn xuất truy cập được lớp cơ sở - độ tin tưởng
- C++ hỗ trợ 3 loại đặc tả truy cập lớp cơ sở – 3 chế độ kế thừa:
  - Kế thừa kiểu public: đối tượng lớp con được tin tưởng như lớp cha

```
class Child : public Parent { };
```

  - Kế thừa kiểu protected: đối tượng lớp con được tin tưởng một phần

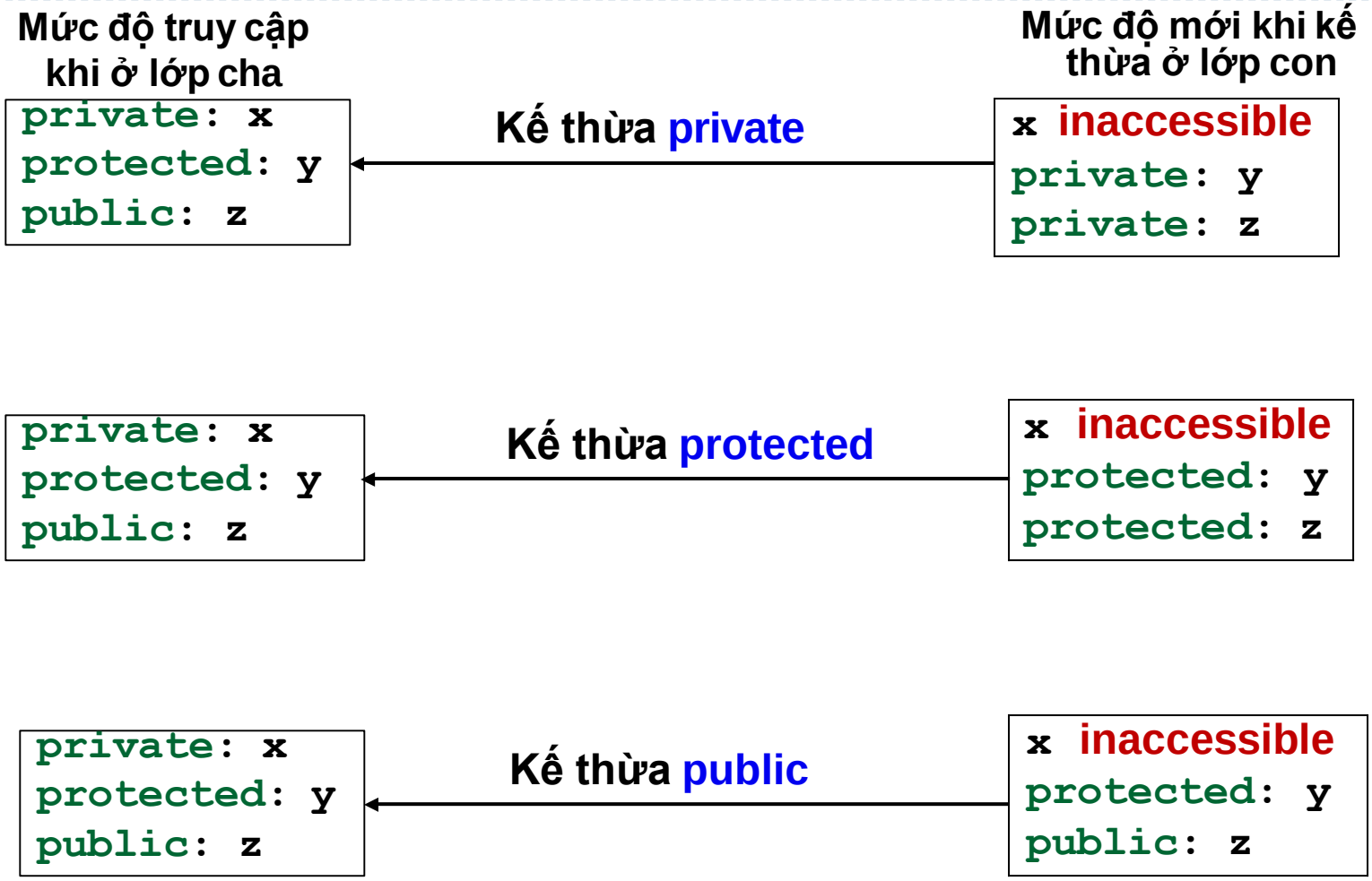
```
class Child : protected Parent{ };
```

  - Kế thừa kiểu private: đối tượng lớp con không được tin tưởng

```
class Child : private Parent{ };    // chế độ mặc định
```

| | | | | |
|---|---|---|---|---|
| Mức độ truy cập của thành viên lớp cha ***từ đối tượng lớp con*** | = | Mức độ truy cập của thành viên lớp cha ***từ đối tượng lớp cha*** | + | Mức độ lớp con truy cập được lớp cha |

# HIỆU ỨNG TỪ 2 LOẠI ĐẶC TẢ TRUY CẬP




*Từ đây nếu không nói thêm thì các bài luôn để kế thừa public*

# Review: Kết hợp 2 đặc tả

| Nếu lớp cơ sở được kế thừa kiểu **public**, thì các đặc tả truy cập thành viên này ở lớp cơ sở: | Sẽ trở thành đặc tả truy cập sau ở lớp kế thừa: |
|---|---|
| private | Không truy cập được |
| protected | protected |
| public | public |
| Nếu lớp cơ sở được kế thừa kiểu **private**, thì các đặc tả truy cập thành viên này ở lớp cơ sở: | Sẽ trở thành đặc tả truy cập sau ở lớp kế thừa: |
| private | Không truy cập được |
| protected | private |
| public | private |
| Nếu lớp cơ sở được kế thừa kiểu **protected**, thì các đặc tả truy cập thành viên này ở lớp cơ sở: | Sẽ trở thành đặc tả truy cập sau ở lớp kế thừa: |
| private | Không truy cập được |
| protected | protected |
| public | protected |

# VD: CAT VÀ SCOTTISHFOLD

***Muốn viết print trong ScottishFold để in mọi thông tin mèo thì sao?***

- Hàm đó cần truy cập thông tin lớp cha (name, age, weight) lẫn con(hair)
- Cách 0: Cha không cho con thông tin, chỉ cho hàm truy xuất lẻ tẻ
  - Lớp cha: để biến private và các getter
  - Lớp con: hàm print sẽ gọi 3 getter, hệt như hàm main khi in thông tin ☹
- Cách 1: Cha không cho con thông tin, nhưng cho hàm truy xuất hiệu quả
  - Lớp cha: để biến private, có hàm printCat in ra tất cả name, age, weight
  - Lớp con: hàm print sẽ gọi printCat, rồi in thêm hair
- Cách 2: Cha cho con thông tin
  - Lớp cha: đổi các biến thành protected
  - Lớp con: hàm print in trực tiếp ra name, age, weight, hair.

# LỰA CHỌN BIẾN CHA PRIVATE VS. PROTECTED

- Tuỳ vào lớp con trực tiếp thay đổi dữ liệu lớp cha thì có sao không?
  - Nếu dữ liệu có ràng buộc (xác thực đầu vào quan trọng, là biến dẫn xuất, …) ta thường cấm con chọc trực tiếp vào dữ liệu cha (để biến private)
  - Còn nếu không nghiêm trọng, ta có thể để dữ liệu cha là protected, cho phép con quyền truy cập trực tiếp.

# VD: SHAPE VÀ CIRCLE

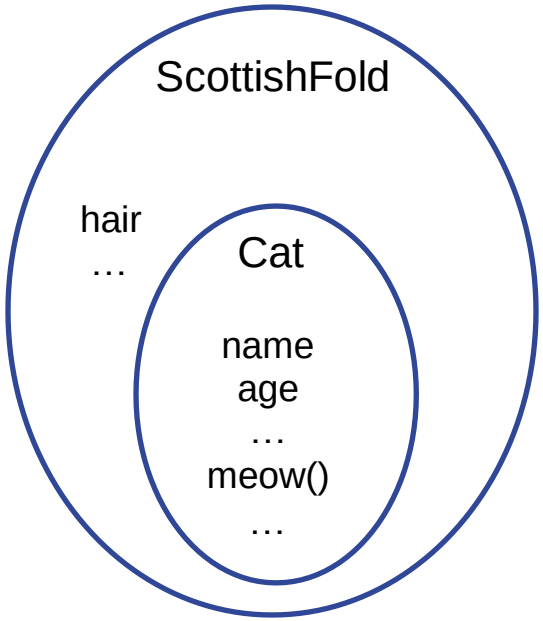
- Viết lớp Shape mô tả một hình, có các thành viên:
  - Biến area chỉ diện tích, với setter và getter cho nó
- Viết lớp Circle mô tả hình tròn, kế thừa Shape và bổ sung thêm:
  - Biến radius chỉ bán kính, với setter và getter cho nó
- Nên để area là private hay protected?
  - Private, vì đây là biến dẫn xuất ở lớp con (~ nói thêm ở hàm ảo)).
- Cần làm sao để ở Circle, area luôn bằng công thức diện tích hình tròn?
- Viết main demo 2 lớp trên:
  - Tạo một biến Shape, đặt diện tích, rồi in ra diện tích của hình
  - Tạo một Circle, đặt bán kính, rồi in ra bán kính, diện tích hình tròn

# HÀM TẠO & HÀM HUỶ

- Hàm tạo, hàm huỷ lớp con và lớp cha
- Thứ tự thực thi hàm tạo, hàm huỷ
- Gọi tường minh hàm tạo lớp cha

# HÀM TẠO & HÀM HUỶ, CON VÀ CHA

- Mỗi đối tượng lớp con đều đang bao chứa một đối tượng lớp cha
- Cần phải tạo lõi dữ liệu của cha rồi mới tạo dữ liệu của con
- => Hàm tạo con chạy hàm tạo cha trước rồi nó mới thực hiện
  - Gọi tự động bởi trình biên dịch
  - Gọi tường minh bởi lập trình viên
- Ngược lại, cần huỷ phần vỏ bên ngoài của con trước rồi mới huỷ đến phần lõi của cha
- Hàm huỷ con chạy xong mới gọi hàm huỷ cha
  - Luôn gọi bởi trình biên dịch (lập trình viên muốn gọi cũng được nhưng không cần)



# VD: THỨ TỰ THỰC THI HÀM TẠO, HÀM HUỶ

```
class BaseDemo
{
public:
    BaseDemo() { cout << "Chay ham tao BaseDemo.\n"; }
    ~BaseDemo() { cout << "Chay ham huy BaseDemo.\n"; }
};

class DeriDemo : public BaseDemo
{
public:
    DeriDemo() { cout << "Chay ham tao DeriDemo.\n"; }
    ~DeriDemo() { cout << "Chay ham huy DeriDemo.\n"; }
};

int main()
{
    cout << "\nTao doi tuong DeriDemo.\n";
    DeriDemo object;
    cout << "Chuong trinh chuan bi ket thuc.\n";
    return 0;
}
```

```
Tao doi tuong DeriDemo.
Chay ham tao BaseDemo.
Chay ham tao DeriDemo.
Chuong trinh chuan bi ket thuc.
Chay ham huy DeriDemo.
Chay ham huy BaseDemo.
```

# VD: HÀM TẠO CHA VÀ HÀM TẠO CON

Nhớ lại Circle kế thừa Shape.

- Thêm vào Shape một hàm tạo 1 tham số khởi tạo cho area
- Biên dịch lại chương trình, giải thích kết quả
- Thêm vào Circle hàm tạo mặc định không làm gì cả, biên dịch và giải thích kết quả
- Sửa hàm tạo mặc định Circle khởi tạo area, biên dịch và giải thích

# VD: GỌI TỰ ĐỘNG HÀM TẠO CHA

```
class Shape{
    float area;
public:
    Shape(float area)
    : area(area){}
    ...
};
```

```
class Shape{
    float area;
public:
    Shape(float area)
    : area(area){}
    ...
};

class Circle: public Shape{
    float radius;
public:
    Circle(){}
    ...
};
```

```
class Shape{
    float area;
public:
    Shape(float area)
    : area(area){}
    ...
};

class Circle: public Shape{
    float radius;
public:
    Circle(){ area = 5; }
    ...
};
```

```
error: implicit default constructor for 'Circle' must explicitly initialize
       the base class 'Shape' which does not have a default constructor
error: constructor for 'Circle' must explicitly initialize
       the base class 'Shape' which does not have a default constructor
```



***Nếu để trình biên dịch gọi, nó luôn gọi hàm tạo cha mặc định !!***

# HÀM TẠO CON GỌI TƯỜNG MINH HÀM TẠO CHA

- Nếu để trình biên dịch tự gọi hàm tạo cha, nó sẽ luôn gọi hàm tạo mặc định và nếu không tìm thấy thì báo lỗi
- Lập trình viên nên gọi tường minh hàm tạo cha khi:
  - Không có hàm tạo cha mặc định
  - Hoặc có hàm đó, nhưng ta đang muốn dùng hàm tạo cha có đối số
  - Hoặc khi cần cập nhật biến private cha mà cha ko có setter
- Cách gọi hàm tạo cha:
  - ***Đặt lời gọi ở khu vực danh sách khởi tạo***

**Hàm_tạo_con**(các_tham_số) **:** **Hàm_tạo_cha** (các_tham_số) … **{…}**

  - Hàm tạo cha đặt đầu tiên, rồi đến khởi tạo cho các biến lớp con

# VD: GỌI TƯỜNG MINH HÀM TẠO CHA

- Sửa hàm tạo mặc định Circle khởi tạo area bằng 0 qua hàm tạo Shape, rồi khởi tạo tiếpradius bằng 0

```
class Shape{
    float area;
public:
    Shape(float area): area(area){}
    void setArea(float a){ area = a; }
    float getArea()const{ return area; }
};
class Circle: public Shape{
    float radius;
public:
    Circle(): Shape(0), radius(0){}
    ...
}
int main(){
    Circle c;
    cout << c.getRadius() << " " << c.getArea() << endl;
    return 0;
}
```

```
0 0
```

Không còn báo lỗi nhờ chủ động gọi đúng hàm tạo cha đã viết

# VD: THỨ TỰ DANH SÁCH KHỞI TẠO

```
class Parent {
    int x, y;
public:
    Parent(int x, int y): x(x), y(y){}
    void print() const { cout << x << " " << y << endl; }
};
class Child : public Parent {
    int z;
public:
    Child(int a): Parent(a, a * a), z(a) {}
    int getZ() const { return z; }
};
int main(){
    Child c(5);
    c.print();
    cout << c.getZ() << endl;
    return 0;
}
```

Thứ tự khai báo biến là thứ tự các biến sẽ khởi tạo bởi danh sách. Danh sách do đó nên viết theo thứ tự khai báo

Khởi tạo biến lớp cha trước mọi biến lớp con. Danh sách vì thế để hàm tạo cha ở đầu

```
5 25
5
```

# VD: THỨ TỰ DANH SÁCH KHỞI TẠO

```
class Cat{
    string name;
    int age;
    float weight;
public:
    Cat (float f):  weight(f), age(int(weight)) {} // thứ tự gây hiểu lầm, cần đảo lại
    ...
};

int main(){
    // dự định tạo mèo 5.5 kg và 5 tuổi, song không đúng
    Cat c(5.5);
    cout << c.getWeight() << " kg va "
         << c.getAge() << " tuoi\n";
    return 0;
}
```

Danh sách gây hiểu lầm weight khởi tạo trước và sau sẽ dùng khởi tạo age. C++ thực tế khởi tạo age rồi mới đến weight.

```
warning: field 'weight' is uninitialized when used here
```

```
5.5 kg va 0 tuoi
```

# VD: GỌI TƯỜNG MINH HÀM TẠO CHA

Nhớ lại, lớp Cat đã sẵn các hàm tạo có 0,1,2,3 tham số. Hãy thêm vào lớp ScottishFold các hàm tạo sau, gọi đến hàm tạo Cat phù hợp

- Hàm tạo mặc định gán kiểu lông là ngắn, mèo tên Bibi, 1 tuổi, 2 kg
- Hàm tạo mặc định gán kiểu lông là vừa, mèo tên Tom, 1 tuổi, 2 kg
- Hàm tạo mặc định gán kiểu lông là dài, mèo tên Tony, 2 tuổi, 2 kg
- Hàm tạo 1 tham số khởi tạo kiểu lông, mèo tên Clark Kent, 2 tuổi, 2 kg
- Hàm tạo 2 tham số khởi tạo kiểu lông, tên mèo, 1 tuổi, 2 kg
- Hàm tạo 4 tham số khởi tạo kiểu lông, tên mèo, tuổi, cân.

# VD: GỌI TƯỜNG MINH HÀM TẠO CHA

```
// mọi hàm tạo con đều viết được bằng cách gọi hàm đầy đủ tham số của lớp cha
// một số hàm tạo con có thể viết được bằng vài cách khác
 // 5 cách viết hàm tạo mặc định gán kiểu lông là ngắn, mèo tên Bibi, 1 tuổi, 2 kg
 ScottishFold(): Cat("Bibi", 1, 2), hair("ngan"){}   // hàm tạo 3 tham số
 ScottishFold(): Cat("Bibi", 1), hair("ngan"){}      // hàm tạo 2 tham số, 1 mặc định
 ScottishFold(): Cat("Bibi"), hair("ngan"){}         // hàm tạo 1 tham số, 2 mặc định
 ScottishFold(): Cat(), hair("ngan"){}               // hàm tạo 0 tham số, 3 mặc định
 ScottishFold(): hair("ngan"){}         // C++ tự gọi hàm tạo 0 tham số khi ta ko gọi

 // 3 cách viết hàm tạo mặc định gán kiểu lông là vừa, mèo tên Tom, 1 tuổi, 2 kg
 ScottishFold(): Cat("Tom", 1, 2), hair("vua"){}     // hàm tạo 3 tham số
 ScottishFold(): Cat("Tom", 1), hair("vua"){}        // hàm tạo 2 tham số, 1 mặc định
 ScottishFold(): Cat("Tom"), hair("vua"){}           // hàm tạo 1 tham số, 2 mặc định

'/ 2 cách viết hàm tạo mặc định gán kiểu lông là dài, mèo tên Tony, 2 tuổi, 2 kg
 ScottishFold(): Cat("Tony", 2, 2), hair("dai"){}    // hàm tạo 3 tham số
 ScottishFold(): Cat("Tony", 2), hair("dai") {}      // hàm tạo 2 tham số, 1 mặc định

 // 1 cách viết hàm tạo mặc định gán kiểu lông là ngắn, mèo tên Tony, 3 tuổi, 5 kg
 ScottishFold(): Cat("Tony", 3, 5), hair("dai"){}    // hàm tạo 3 tham số
```

# VD: GỌI TƯỜNG MINH HÀM TẠO CHA

```
// mọi hàm tạo con đều viết được bằng cách gọi hàm đầy đủ tham số của lớp cha
// một số hàm tạo con có thể viết được bằng vài cách khác


// 2 cách viết hàm tạo 1 tham số khởi tạo kiểu lông, mèo tên Clark Kent, 2 tuổi, 2 kg
ScottishFold(string hair): Cat("Clark Kent", 2, 2), hair(hair){}        // hàm tạo 3 tham số
ScottishFold(string hair): Cat("Clark Kent", 2), hair(hair){}           // hàm tạo 2 tham số, 1 mặc định

// 3 cách viết hàm tạo 2 tham số khởi tạo kiểu lông, tên mèo, 1 tuổi, 2 kg
ScottishFold(string hair, string name): Cat(name, 1, 2), hair(hair){}   // hàm tạo 3 tham số
ScottishFold(string hair, string name): Cat(name, 1), hair(hair){}      // hàm tạo 2 tham số, 1 mặc định
ScottishFold(string hair, string name): Cat(name), hair(hair){}         // hàm tạo 1 tham số, 2 mặc định

// 1 cách viết hàm tạo 4 tham số khởi tạo kiểu lông, tên mèo, tuổi, cân.
ScottishFold(string hair, string name, int age, float weight)
: Cat(name, age, weight), hair(hair){}                                  // hàm tạo 3 tham số
```

# VD: GỌI TƯỜNG MINH HÀM TẠO CHA

Nhớ lại, lớp Rectangle có các thành viên sau:

- width, length và setter, getter tương ứng
- hàm tạo 2 tham số khởi tạo cho 2 biến, có 2 đối số mặc định là 0
- getArea trả về diện tích hình chữ nhật

Viết lớp Cube mô tả khối hộp kế thừa Rectangle, có thêm:

- Chiều cao height và thể tích volume
- Danh sách setter, getter hợp lý
- Hàm tạo mặc định gán rộng, dài, cao bằng 0
- Hàm tạo 3 tham số khởi tạo cho cả rộng, dài, cao

Demo tạo ra một Cube với rộng, dài, cao nào đó, gọi mọi hàm có thể

# VD: GỌI TƯỜNG MINH HÀM TẠO CHA

```
// Khai báo lớp Rectangle
class Rectangle
{
    double width;
    double length;
public:
    Rectangle()
    : width(0), length(0){}
    Rectangle(double w, double l)
    : width(w), length(l){}

    double getWidth() const
    { return width; }
    double getLength() const
    { return length; }
    double getArea() const
    { return width * length; }
};
```

```
2
5
3
10
30
```

```
// Khai báo lớp Cube
class Cube : public Rectangle
{
protected:
    double height;
    double volume;
public:
    Cube()
    : Rectangle(), height(0), volume(0){}
    Cube(double w, double l, double h)
    : Rectangle(w, l), height(h)
    { volume = getArea() * h; }
    double getHeight() const { return height; }
    double getVolume() const { return volume; }
};
int main()
{
    //tạo đối tượng Cube
    Cube myCube(2, 5, 3);
    cout << myCube.getWidth() << endl
         << myCube.getLength() << endl
         << myCube.getHeight() << endl
         << myCube.getArea() << endl
         << myCube.getVolume() << endl;
    return 0;
}
```

# TÁI ĐỊNH HÀM LỚP CHA

- Tái định
- Hàm tái định che khuất hàm lớp cha

# LỚP CON CÓ GÌ

```
class Insect{
    string color;
public:
    void setColor(string);
    string getColor() const;
    void makeNoise() const;
};
 class Grasshopper: public Insect{
    int highestJump;
public:
    int jump();
    int getHighestJump() const;
 };
```

Có mọi thành viên lớp cha
Lẫn mọi thành viên lớp con

- Thành viên cùng lớp phải khác tên nhau, hoặc là hàm chồng
- Còn thành viên lớp và thành viên kế thừa có cần khác tên không?
  - Không, lớp con có thể có biến, hàm trùng tên biến, hàm lớp cha
  - Nhưng tại sao cần giống để làm gì?

# VD: SHAPE VÀ CIRCLE

- Nhớ lại, Circle kế thừa Shape và có biến bán kính radius, khi cập nhật radius sẽ cập nhật cả biến diện tích area ở lớp Shape theo công thức diện tích tròn.
- Nhưng Circle kế thừa cả hàm setArea 1 tham số từ lớp Shape. Qua hàm này, diện tích hình tròn vẫn có rủi ro bị gán giá trị không liên quan bán kính

```
Circle c;
c.setRadius(5);
cout << c.getRadius() << " " << c.getArea() << endl;
c.setArea (100);
cout << c.getRadius() << " " << c.getArea() << endl;
```

```
5 78.5
5 100
```

- Làm cách nào chặn việc gọi setArea() trên đối tượng Circle? ☹
  - Đặt setArea là private => chặn gọi setArea ngay cả trên Shape
  - Tốt hơn, Circle tự tạo một hàm setArea để thay thế setArea lớp cha

# TÁI ĐỊNH HÀM LỚP CHA

- Là tạo một hàm lớp con trùng tên hàm lớp cha
- ***Hàm tái định sẽ luôn che khuất hàm cha***
  - Đối tượng con luôn chạy bản tái định thay vì bản gốc
- Cho phép ta thay thế hàm cha về thân hàm/ tham số / sự truy cập (??)
- Muốn gọi lại hàm cha phải gọi tên đầy đủ: lớp_cha::tên_hàm (tham_số)
- Phân biệt hàm tái định với hàm chồng:
  - Hàm chồng phải khác nhau tham số và phải cùng một lớp
  - Hàm tái định và hàm gốc được giống tham số, nhưng nằm ở 2 lớp
  - Nạp chồng được hàm toàn cục; song chỉ tái định hàm thành viên.

# VD:CIRCLE

## tái định Shape

setArea() con tái định setArea cha về danh sách tham số lẫn nội dung, main không thể gọi setArea cha, area không sợ bị sai so với bán kính.

```
5 78.5
5 78.5
```

```
class Shape{
    float area;
public:
    Shape(float area): area(area){}
    void setArea(float a){area = a;}
    float getArea()const{return area;}
};

class Circle: public Shape{
    float radius;
public:
    Circle(): Shape(0), radius(0){}
    void setArea() {        // hàm tái định
        Shape::setArea(3.14 * radius * radius);
    }
    void setRadius(float r){
        radius = r;
        setArea();
    }
    float getRadius() const { return radius; }
};
int main(){
    Circle c;
    c.setRadius(5);
    cout << c.getRadius() << " " << c.getArea() << endl;
    // c.setArea(100);  // error
    c.setArea();
    cout << c.getRadius() << " " << c.getArea() << endl;
    return 0;
}
```

# VD: TIẾP TỤC TÁI ĐỊNH MẠNH HƠN

Hàm setArea() con có thể tái định setArea cha về cả đặc tả truy cập, vd từ public trở thành private. Không thể gọi hàm setArea ở con, trừ phi gọi tên đầy đủ của hàm cha

```
5 78.5
5 78.5
```

```
class Shape{
    float area;
public:
    Shape(float area): area(area){}
    void setArea(float a){ area = a; }
    float getArea()const{ return area; }
};

class Circle: public Shape{
    float radius;
    void setArea() {            // hàm tái định
        Shape::setArea(3.14 * radius * radius);
    }
public:
    void setRadius(float r){
        radius = r;
        setArea();
    }
    Circle(): Shape(0), radius(0){}
    float getRadius() const
    { return radius; }
};
int main(){
    Circle c;
    c.setRadius(5);
    cout << c.getRadius() << " " << c.getArea() << endl;
    // c.setArea(100);  // error
    // c.setArea();     // error
    c.Shape::setArea(10);
    return 0;
}
```

# VD: NẠP CHỒNG, TÁI ĐỊNH, CHE KHUẤT

```
// minh hoạ khác biệt nạp chồng vs. tái định và
// hiệu ứng che lấp hàm lớp cơ sở của việc tái định
#include <iostream>
using namespace std;

class BaseClass
{
public:
    void showMessage(int a, int b)
    { cout << "Hàm lớp cơ sở 0 tham số.\n";}

    void showMessage() // nạp chồng
    { cout << "Hàm lớp cơ sở 1 tham số.\n";}
};

class DerivedClass : public BaseClass
{
public:
    void showMessage() const // tái định
    {
        cout << "Hàm lớp dẫn xuất 0 tham số.\n";
    }
};
```

2 hàm showMessage () lớp cha đều bị che, kể cả hàm 2 tham số. Muốn gọi phải nêu tên đầy đủ

```
int main()
{
    BaseClass b;
    DerivedClass d;
    b.showMessage();
    d.showMessage();
    // d.showMessage(3, 5);
    d.BaseClass::showMessage(3, 5);
    return 0;
}
```

```
Hàm lớp cơ sở 1 tham số.
Hàm lớp dẫn xuất 0 tham số.
Hàm lớp cơ sở 0 tham số.
```

# KHI NÀO CẦN TÁI ĐỊNH?

- Nhớ lại, khi muốn viết hàm print trong lớp ScottishFold, có một cách như sau:
  - Lớp cha: Giữ các biến là private nhưng viết hàm printCat in ra thông tin
  - Lớp con: viết hàm print gọi printCat lớp cha, rồi in thêm chiều dài lông
- Giờ ta biết có thể để hàm in lớp Cat cũng tên print và print ở ScottishFold do đó tái định print ở Cat.
  - Cũng có ích nhất định khi tiện dùng chỉ một tên cho việc in ở nhiều nơi
  - Nhưng không thật cần, vì nếu đặt tên khác nhau cũng vẫn ổn
- Khi nào tái định là thực sự bắt buộc?
  - Khi cần che khuất hàm lớp cha: do không phù hợp đối tượng lớp con nữa.
    - Hàm setArea của lớp Shape sẽ cho phép nhập diện tích bất kỳ và không liên quan biến bán kính của lớp Circle nữa ;
    - hàm likeEatingDogs có thể không phù hợp với đối tượng GenZ (~ xem sau)

# VD: SHAPE3D & CUBE

- Viết lớp Shape3D mô tả hình 3D, có các thành viên:
  - Biến volume chỉ thể tích, với setter và getter cho nó
- Viết lớp Cube mô tả hình hộp vuông, kế thừa Shape3D và bổ sung:
  - Biến size chỉ độ dài cạnh hình hộp, với setter và getter cho nó
- Viết main demo 2 lớp trên:
  - Tạo biến Shape3D, đặt thể tích, rồi in ra thể tích
  - Tạo biến Cube, đặt độ dài cạnh, in ra độ dài cạnh và thể tích hình

# VD: KHOẢNG CÁCH THẾ HỆ

- GenX kế thừa Boomer, bổ sung:
  - Hằng tên like, có 1 hàm tạo sẽ khởi tạo nó bằng một số ngẫu nhiên 1/0
  - Tái định likeEatingDog sẽ tuỳ like là 0/1 mà in ra “I like eating dogs”, hay “I dont like eating dogs”.
- GenY kế thừa genX:
  - Hằng tên eat, có 1 hàm tạo khởi tạo like là 0, còn eat bằng ngẫu nhiên 0/1
  - tái định eatDog sẽ tuỳ hằng eat mà in ra “I eat dogs” hay “I dont eat dogs”.
  - tái định likeEatingDog để không đối tượng genY nào gọi được hàm này
- GenZ kế thừa genY
  - Tái định eatDog để không genZ nào gọi được hàm này.
- Tự bổ sung setter getter và các hàm tạo khác nếu thấy cần thiết, rồi demo các lớp trên

# VD: KHOẢNG CÁCH THẾ HỆ

```
class Boomer{
public:
    Boomer()
    { cout << "I belong to Boomer\n"; }
    void eatDogs() const
    { cout << "I eat dogs\n";}
    void likeEatingDogs() const
    { cout << "I like eating dogs\n"; }
};

class GenX: public Boomer{
    const bool like;
public:

    GenX(): like(rand()%2)
    { cout << "I belong to genX\n"; }
    void likeEatingDogs() const {
        if (like) Boomer::likeEatingDogs();
        else
            cout << "I dont like eating dogs\n";
    }
};
```

```
class GenY: public GenX{
    const bool eat;
    void likeEatingDogs() const {}
public:
    GenY(): eat(rand()%2)
    { cout << "I belong to genY\n"; }
    void eatDogs() const{
        if (eat) Boomer::eatDogs();
        else cout << "I dont eat dogs\n";
    }
};

class GenZ: public GenY{
    void eatDogs() const{}
public:
    GenZ()
    { cout << "I belong to genZ\n"; }
};
```

```
int main(){
    srand(time(0));
    Boomer b;
    b.eatDogs();
    b.likeEatingDogs();

    GenX x;
    x.eatDogs();
    x.likeEatingDogs();

    GenY y;
    y.eatDogs();
    // y.likeEatingDogs();

    GenZ z;
    // z.eatDogs();
    // z.likeEatingDogs();
    return 0;
}
```

```
I belong to Boomer
I eat dogs
I like eating dogs

I belong to Boomer
I belong to genX
I eat dogs
I dont like eating dogs

I belong to Boomer
I belong to genX
I belong to genY
I dont eat dogs

I belong to Boomer
I belong to genX
I belong to genY
I belong to genZ
```

# ALGORITHM WORKBENCH

**Employee & ProductionWorker**

- Viết lớp Employee có các thuộc tính: tên, mã số, ngày thuê, tiền lương hàng tháng. Viết một/nhiều hàm tạo, setter, getter phù hợp.
- Viết lớp ProductionWorker kế thừa Employee có thêm các thuộc tính: ca làm việc, mức lương theo giờ, số giờ làm việc trong tháng. Ca làm việc có giá trị bằng 1 nếu hiểu là ca ngày, bằng 2 nếu là ca đêm. Nếu ca đêm thì lương thực lĩnh sẽ bằng lương theo giờ tăng thêm 50%. Viết một/ nhiều hàm tạo, setter, getter phù hợp.
- Cả 2 lớp tự chọn biến private, protected phù hợp

# ALGORITHM WORKBENCH

**Time & Time12**

- Có 2 dạng lưu trữ thời gian là chuẩn 24h và chuẩn 12 h với quy tắc sau:
  - 4h 5 phút 20 giây chiều trong dạng chuẩn 24 sẽ lưu thành 16, 5, 20 ứng với giờ phút giây, còn trong dạng 12 h sẽ lưu thành 4, 5, 20 kèm một biến số nguyên đại diện cho buổi - với 1 là buổi sáng và 2 là buổi chiều.
- Viết lớp **Time12** kế thừa **Time**, cho phép nhập thời gian dạng chuẩn 12 và chuyển thành thời gian dạng chuẩn 24h.

```
class Time
{
protected:
    int hour;
    int min;
    int sec;
public:
    Time(): hour(0), min(0), sec(0){}
    Time(int h, int m, int s)
    : hour(h), min(m), sec(s){}
    int getHour() const{ return hour; }
    int getMin() const { return min; }
    int getSec() const { return sec; }
};
```

# ALGORITHM WORKBENCH

Lớp Time12 có thêm các thành viên:

- Biến hour12, cho phép lưu giờ dạng 12
- Biến period kiểu nguyên, lưu sáng/ chiều với sáng là 1 và chiều là 2
- 13 h 30 phút 4 giây sẽ lưu hour12 = 1, min = 30, sec = 4, period = 2
- Hàm tạo: nhận đối số cho giờ, phút, giây, sáng/chiều như mô tả ở dạng chuẩn 12 và tự cập nhật các biến chuẩn 24 của lớp Time.
- setTime: chấp nhận đối số cho giờ, phút, giây, sáng/chiều như mô tả ở dạng chuẩn 12 và tự cập nhật các biến chuẩn 24 của lớp Time.
- getHour: trả về giờ dạng chuẩn 12.
- getPeriod: trả về sáng hay chiều
- getStandHr: trả về giờ dạng chuẩn 24

# ALGORITHM WORKBENCH

**Time & MilTime**

- Có 2 dạng lưu trữ thời gian là dạng chuẩn và dạng quân đội với quy tắc sau:
  - 4h 5 phút 20 giây trong dạng chuẩn sẽ lưu thành 4, 5, 20 ứng với giờ phút giây, còn trong dạng quân đội sẽ lưu thành bộ chỉ có hai số 405, 20 ứng với "giờ quân đội" và giây.
- Viết lớp **MilTime** kế thừa **Time**, cho phép nhập thời gian dạng chuẩn quân đội và chuyển thành thời gian dạng chuẩn.

```
class Time
{
protected:
    int hour;
    int min;
    int sec;
public:
    Time(): hour(0), min(0), sec(0){}
    Time(int h, int m, int s)
    : hour(h), min(m), sec(s){}
    int getHour() const{ return hour; }
    int getMin() const { return min; }
    int getSec() const { return sec; }
};
```

# ALGORITHM WORKBENCH

Lớp MilTime có thêm các thành viên:

- Biến milHour: chứa giờ dạng quân đội, ví dụ giờ 1630 dạng quân đội nghĩa là 16h30 phút trong thực tế.
- Biến milSec: giây dạng quân đội thì chứa giây giống hệt dạng chuẩn
- Hàm tạo: cần chấp nhận đối số cho giờ và giây ở dạng quân đội. Các thông tin này sẽ chuyển về dạng chuẩn lưu trong các biến của lớp Time.
- setTime: chấp nhận đối số ở dạng quân đội lưu vào milHour và milSec. Các thông tin này sẽ chuyển về dạng chuẩn lưu trong các biến của lớp Time.
- getHour: trả về giờ dạng quân đội.
- getStandHr: trả về giờ dạng chuẩn

**Xác thực đầu vào:** Với lớp Time, sinh viên cần tự xác định tiêu chuẩn đầu vào phù hợp. Với lớp MilTime, không chấp nhận giờ lớn hơn 2359 hay nhỏ hơn 0, cũng không chấp nhận giây lớn hơn

# ALGORITHM WORKBENCH

**Propo &AndPropo & OrPropo & …**

- Viết lớp Propo mô tả một mệnh đề logic, với các thành viên sau:
  - Một biến sequence lưu biểu diễn của mệnh đề. VD “P^QvR”
  - Một biến value lưu giá trị chân lý của mệnh đề. VD true hay false
  - Hàm tạo 2 tham số khởi tạo cho 2 biến
- Viết lớp AndPropo mô tả mệnh đề hội, kế thừa lớp Propo, với các thành viên:
  - Mệnh đề trái và mệnh đề phải của phép hội
  - Hàm tạo 2 tham số khởi tạo 2 mệnh đề lẫn khởi tạo biến sequence và value đúng theo quy tắc phép hội của mệnh đề trái và phải
- Làm tương tự cho OrPropo, NotPropo, ParentPropo, InductPropo, EquiPropo tương ứng với các phép tuyển, phủ định, bao ngoặc, suy ra và tương đương
- Tự thêm setter, getter phù hợp cho các lớp
- Main tạo mệnh đề hằng rồi tạo các mệnh đề phức, in ra xâu và giá trị value

# TỔNG KẾT

- Cú pháp kế thừa & sử dụng các thành viên kế thừa
- Lựa chọn protected vs. private cho thành viên lớp cha
- Hàm tạo con gọi tường minh hàm tạo cha
- Tái định hàm cha

# Keep calm and carry on