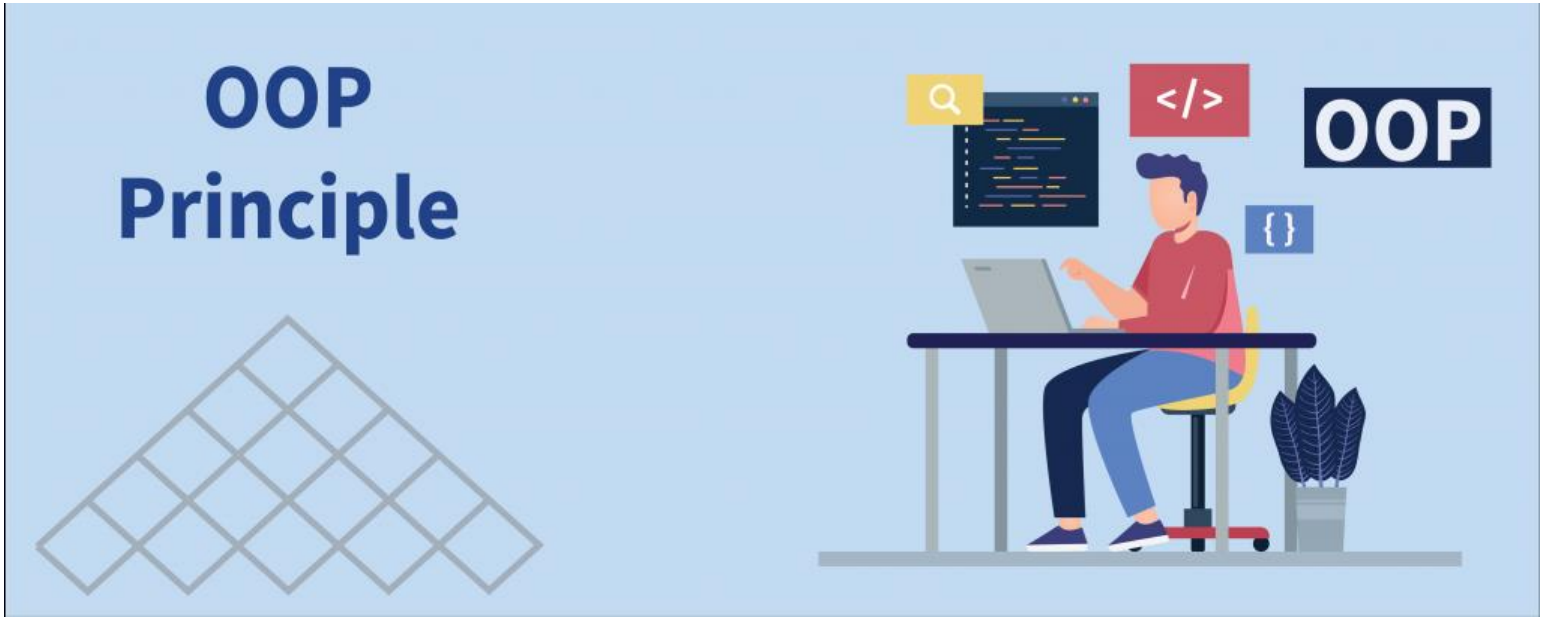


ĐA HÌNH

Ths. Đinh Thị Thúy – AI LAB/TLU

Thuydt@thanglong.edu.vn – PAI006

Xây dựng cây phả hệ

- Case study: GradedActivity và các lớp con
- Cây phả hệ

Case study: GradedActivity

- Giáo viên thường giao nhiều loại bài tập lấy điểm: bài lý thuyết, bài thực hành, bài trắc nghiệm, bài giữa kỳ, bài cuối kỳ,…
- Các hoạt động này đều giống nhau là sinh ra một điểm số và điểm này về sau sẽ chuyển sang dạng chữ A, B, C, D, F.
- Nên viết một lớp cha là GradedActivity chứa những yếu tố chung cho mọi bài tập tính điểm. Sau đó từ lớp cha này sẽ dẫn xuất ra nhiều lớp con ứng với các loại bài tập khác nhau

GradedActivity & FinalExam

- Lớp GradedActivity mô tả một bài tập với điểm số và điểm chữ. Điểm số sẽ được nhập từ ngoài, còn điểm chữ được suy ra từ điểm số.
- Lớp FinalExam mô tả bài thi cuối kỳ cũng có điểm số và điểm chữ, song điểm số không được nhập, mà được tính từ tổng số câu trong đề thi và tổng số câu làm sai. Cả đề tổng 100 điểm.
- 2 lớp này chung nhau:
  - điểm số
  - getter cho điểm số
  - Hàm tính điểm chữ (tại sao ko có biến điểm chữ ??)
- => Để tái sử dụng mã ta sẽ dùng kế thừa, nhưng ai kế thừa ai ??

Bước 1: Phân tích lớp cha & lớp con

Lớp GradedActivity gồm các thành viên:

- biến score chỉ điểm số của bài tập (điểm trong khoảng [0,100]
- setter getter cho biến score
- (các) hàm tạo để khởi tạo score
- hàm trả về điểm chữ (A,B,C,D,F) tính từ điểm số

```
F thuộc [0; 60]
D thuộc (60;70]
C thuộc (70;80]
B thuộc (80;90]
A thuộc (90;100]
```

Lớp FinalExam kế thừa từ GradedActivity, bổ sung thêm:

- tổng số câu đề thi, tổng số câu làm sai
- setter getter phù hợp cho các biến
- (các) hàm tạo để khởi tạo biến

} ***Nhớ cập nhật score lớp cha***

Main sẽ tạo đối tượng FinalExam để test việc tính điểm số, điểm chữ

Bước 2: Cài đặt lớp cha

```
class GradedActivity {
    double score;   // điểm số
public:
    void setScore(double s)
    { score = s; }

    GradedActivity(double s = 0)
    : score(s){}

    double getScore() const
    { return score; }

    char getLetterGrade() const
    {
        char letterGrade;
        if (score > 90) letterGrade = 'A';
        else if (score > 80) letterGrade = 'B';
        else if (score > 70) letterGrade = 'C';
        else if (score > 60) letterGrade = 'D';
        else letterGrade = 'F';
        return letterGrade;
    }
};
```

Chỉ có biến score lưu điểm số. Còn điểm chữ được dẫn xuất từ điểm số, nên có thể viết hàm tính toán trả về thay vì tạo biến điểm chữ

Bước 3: Cài đặt lớp con

```
class FinalExam : public GradedActivity
{
    int numQuestion;        // tổng số câu đề thi
    int numMissed;          // tổng số câu sai

    void setScore() {
        // tính điểm rồi đặt cho biến score lớp cha
        double tmp = 100 - (numMissed * 100.0/numQuestion);
        GradedActivity::setScore(tmp);
    }
public:
    void setInfor(int question, int missed){
        numQuestion = question;
        numMissed = missed;
        setScore();
    }

    FinalExam(int question = 0, int missed = 0)
    { setInfor(question, missed); }

    double getNumQuestion() const { return numQuestion; }
    int getNumMissed() const { return numMissed; }
};
```

Hàm setScore lớp con trùng tên nên tái định hàm setScore lớp cha.

Để gọi đúng phiên bản hàm cha cần nêu tên đầy đủ

Bước 4: Demo chương trình

Hàm main làm tiếp các việc sau:

- Tạo biến FinalExam với số câu đề thi và số câu sai cho trước
- In ra điểm số, điểm chữ của bài thi

```
int main()
{
    // từ số câu đề thi và số câu sai tạo đối tượng FinalExam
    int question = 50, missed = 18 ;
    FinalExam test(question, missed);

    // hiển thị điểm thi dạng số, dạng chữ
    cout << "Diem thi so: " << test.getScore() << endl;
    cout << "Diem thi chu: " << test.getLetterGrade() << endl;
    return 0;
}
```

```
Diem thi so: 64
Diem thi chu: D
```

GradedActivity & CurvedActivity

- Lớp GradedActivity mô tả một bài tập có điểm.
- Có lúc giáo viên muốn hiệu chỉnh điểm trước khi công bố, ví dụ nhân thêm với một tỷ lệ nào đó.
- Lớp CurvedActivity mô tả một đầu điểm hiệu chỉnh như vậy, nó kế thừa GradedActivity và có thêm các thành viên:
  - điểm gốc rawScore, tỷ lệ rate. Luôn cần có score = rawScore* rate
  - các setter, getter cho từng biến rate, rawScore
  - hàm tạo 2 tham số khởi tạo các biến, và có 2 đối mặc định tự chọn
  - cần làm sao để không gọi được setScore lớp cha từ đối tượng lớp con
- Main tạo ra đối tượng CurvedActivity và test các hàm

VD: GradedActivity & CurveActivity

```
class CurvedActivity : public GradedActivity
{
    double rawScore;    // điểm gốc
    double rate;        // tỷ lệ hiệu chỉnh

    void setScore(){    // tái định setScore lớp cha
        GradedActivity::setScore(rawScore * rate);
    }
public:
    void setRawScore(double s)
    {
        rawScore = s;
        setScore();
    }
    void setRate(double r)
    {
        rate = r;
        setScore();
    }

    CurvedActivity(double raw = 0, double rate = 0)
    : rawScore(raw), rate(rate) { setScore(); }

    double getRate() const { return rate; }
    double getRawScore() const { return rawScore; }
};
```

```
int main()
{
    CurvedActivity test (8, 1.1) ;
    cout << test.getScore() << endl;
    cout << test.getLetterGrade() << endl;
    return 0;
}
```

```
8
8.8
```

Biến score lớp cha bị tác động bởi biến rawScore và rate lớp con. Nên mọi hàm cập nhật rawScore và rate đều cần cập nhật score lớp cha

GradedActivity & PassFailActivity

- Lớp GradedActivity mô tả một bài tập mà có điểm chữ dạng A,B,C,D,F.
- Song một số bài tập chỉ tính qua hay trượt, nên điểm chữ chỉ có dạng P (pass) hay F (fail).
- Lớp PassFailActivity mô tả một bài tập có qua hay trượt như vậy, nó kế thừa GradedActivity và có thêm các thành viên:
  - điểm tối thiểu để qua minPassing
  - setMinPassing khởi tạo minPassing lớp con
  - hàm tạo 2 tham số với đối mặc định, sẽ khởi tạo score lớp cha và minPassing lớp con
  - tái định getLetterGrade lớp cha để trả về P, F thay vì A, B, C, D, F.
- Main sẽ tạo đối tượng PassFailActivity để test các hàm.

```
class PassFailActivity : public GradedActivity
{
    double minPassing; // điểm đỗ tối thiểu
public:
    void setMinPassing(double mp)
    {  minPassing = mp; }

    PassFailActivity(double score = 0, double mp = 0)
    : GradedActivity(score), minPassing (mp) {}

    double getMinPassing() const
    { return minPassing; }

    char getLetterGrade() const{
        char letterGrade;
        if (getScore() >= minPassing)
            letterGrade = 'P';
        else letterGrade = 'F';
        return letterGrade;
    }
};
int main()
{
    PassFailActivity test (65, 64);
    cout << test.getScore() << endl;
    cout << test.getLetterGrade() << endl;
    return 0;
}
```

Hàm tạo con khởi tạo phần lõi cha thông qua hàm tạo cha

```
65
P
```

PassFailActivity & PassFailExam

- Lớp PassFailActivity mô tả một bài tập có điểm chữ dạng P hay F. Bài thi cũng là một bài tập trả về điểm chữ dạng P hay F, song phần điểm số thì không được nhập vào mà tính từ số câu đề thi và số câu sai.
- PassFailExam mô tả một bài thi như vậy. Nó kế thừa PassFailActivity và có thêm các thành viên:
  - tổng số câu đề thi và tổng số câu làm sai
  - setter getter phù hợp cho các biến
  - hàm tạo để khởi tạo cho các biến
  - tái định hàm nếu cần thiết
- Main sẽ tạo đối tượng PassFailExam để test các hàm.

PassFailActivity & PassFailExam

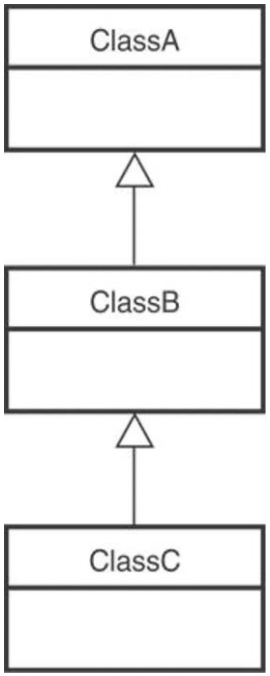
```
class PassFailExam : public PassFailActivity
{
    int numQuestion;    // số câu hỏi
    int numMissed;      // số câu trả lời sai
    void setScore() {   // tái định hàm lớp cha
        double num = 100 - (numMissed * 100/numQuestion);
        GradedActivity::setScore(num);
    }
public:
    void setInfor(int question, int missed){
        numQuestion = question;
        numMissed = missed;
        setScore();
    }
    PassFailExam(int question = 0, int missed = 0, int mp = 70)
    {
        setInfor(question, missed);
        setMinPassing(mp);
    }

    double getNumQuestion() const{ return numQuestion; }
    int getNumMissed() const { return numMissed; }
};
```

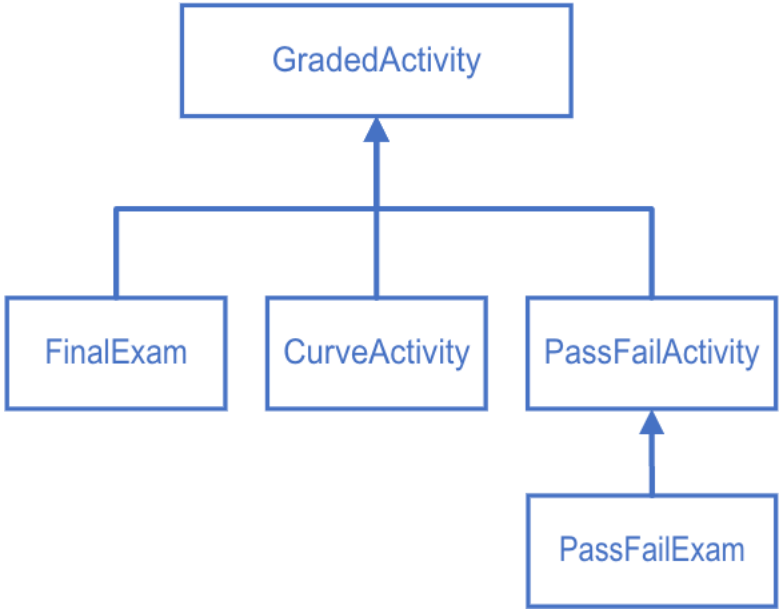
```
int main()
{
    cout << endl;
    PassFailExam test (50, 18, 60);
    cout << test.getScore() << endl;
    cout << test.getLetterGrade() << endl;
    cout << endl;
    return 0;
}
```

64
P

Cây phả hệ

- Một lớp con của lớp này có thể là lớp cha của lớp khác
- Cây phả hệ là một cấu trúc mô tả quan hệ kế thừa phân cấp của nhiều lớp, bắt đầu từ lớp gốc (lớp tổ tiên), toả ra các lớp con, cháu,…





Hàm ảo, ghi đè, tính đa hình

- Liên kết tĩnh và liên kết động
- Hàm ảo và ghi đè
- Tính đa hình

Liên kết lời gọi hàm với hàm

- Xét hàm sau:

```
void displayGrade(const GradedActivity & activity) {
    cout << "Diem so: " << activity.getScore() << endl;
    cout << "Diem chu: " << activity.getLetterGrade() << endl << endl;
}
```

- Tham số ở đây là một biến tham chiếu kiểu GradedActivity, mà bất kỳ đối tượng nào có nguồn gốc từ GradedActivity cũng "là một" GradedActivity, nên ta được phép truyền vào hàm này một đối tượng FinalExam, một đối tượng PassFailExam, một đối tượng bất kỳ thuộc lớp con, cháu của lớp GradedActivity, hoặc truyền vào chính đối tượng GradedActivity.
- Tổng quát, một đối tượng con "là một" đối tượng cha và được phép làm mọi thứ cha làm, ví dụ truyền vào hàm, trả về từ hàm, gán cho cha, …

Liên kết lời gọi hàm với hàm

- Ta lần lượt truyền các đối tượng kiểu khác nhau vào hàm:

```
GradedActivity test1(80);
PassFailActivity test2(65, 60);
PassFailExam test3(60, 50, 18);

displayGrade(test1);
displayGrade(test2);
displayGrade(test3);
```

```
Diem so: 80
Diem chu: C

Diem so: 65
Diem chu: D

Diem so: 64
Diem chu: D
```

- Kết quả ở đây không như ta chờ đợi, hàm getLetterGrade trả về 'D' cả khi ta đang có các đầu điểm chỉ xét qua/trượt.
- Như vậy, hàm getLetterGrade của lớp GradedActivity được thực thi thay vì phiên bản hàm getLetterGrade của lớp PassFailActivity và PassFailExam.
- Đây là kết quả của việc liên kết tĩnh

Liên kết tĩnh

- Sự liên kết (binding): là quá trình khớp lời gọi hàm với mã hàm
- Bình thường C ++ sử dụng liên kết tĩnh (static binding) – liên kết tại thời điểm dịch:
  - Trình dịch nói chung kiểm tra khá máy móc, ví dụ nếu biến khai báo kiểu GradedActivity thì nó quyết định coi mọi lời gọi trên biến đó gắn với hàm thuộc lớp GradedActivity, kể cả khi biến này thực ra là đối tượng PassFailActivity.
- Có thể khắc phục điều này nếu kích hoạt liên kết động, bằng cách chuyển getLetterGrade thành hàm ảo.

Hàm ảo

- Hàm thành viên ảo: là một hàm thành viên lớp cha mà có lúc lại chạy thành phiên bản hàm định nghĩa ở lớp con
- Hàm được xác định bằng từ khóa virtual:

  virtual void Y() {...}
- Hỗ trợ liên kết động (dynamic binding) – liên kết lúc thời gian chạy
  - Trình biên dịch sẽ không liên kết hàm, mà chương trình sẽ liên kết. Cụ thể là, nó sẽ xác định đúng loại đối tượng thực hiện cuộc gọi và liên kết lời gọi đó với phiên bản hàm ở lớp thích hợp.

Cập nhật GradedActivity với hàm ảo

```
class GradedActivity {
    double score;   // điểm số
public:
    void setScore(double s)
    { score = s; }

    GradedActivity(double s = 0)
    : score(s){}

    double getScore() const
    { return score; }

    virtual char getLetterGrade() const
    {
        char letterGrade;
        if (score > 90) letterGrade = 'A';
        else if (score > 80) letterGrade = 'B';
        else if (score > 70) letterGrade = 'C';
        else if (score > 60) letterGrade = 'D';
        else letterGrade = 'F';
        return letterGrade;
    }
};
```

Hàm bây giờ là ảo.

Hàm cũng tự động trở thành hàm ảo trong tất cả các lớp dẫn xuất!

Hàm ảo và tính đa hình

- Nếu biên dịch lại đoạn mã khi nãy, sẽ nhận được đầu ra phù hợp:

```
GradedActivity test1(80);
PassFailActivity test2(65, 60);
PassFailExam test3(60, 50, 18);

displayGrade(test1);
displayGrade(test2);
displayGrade(test3);
```

```
Diem so: 80
Diem chu: C

Diem so: 65
Diem chu: P

Diem so: 64
Diem chu: P
```

- Cùng một lời gọi hàm getLetterGrade, cùng gọi trên tham số GradedActivity, nhưng khi tham số này có kiểu cụ thể khác nhau lại cho ra các loại kết quả khác nhau
  - Test1 thì điểm kiểu A,B, C; còn test2, test3 thì kiểu P/F
- Loại hành vi này được gọi là đa hình (polymorphism) - khả năng có nhiều hình dạng – một thông điệp có nhiều cách phản ứng

Tính đa hình: Biến tham chiếu hoặc con trỏ

- Hành vi đa hình chỉ thể hiện khi đối tượng được truy cập qua biến tham chiếu hoặc con trỏ
- Quy trình biểu diễn tính đa hình:
  1. Lớp cha khai báo hàm X là ảo
  2. Lớp con tái định hàm X với tham số y hệt (lớp con ghi đè X)
  3. Có biến lớp cha nhưng tham chiếu/trỏ tới đối tượng lớp con
  4. Khi ấy gọi X trên biến đó sẽ chạy ra phiên bản hàm lớp con
- Việc lớp con có một hàm cùng tên, cùng tham số với một hàm ảo lớp cha còn gọi là sự ghi đè (overriding)

Con trỏ lớp cha và địa chỉ lớp con

- Con trỏ kiểu cha được phép gán bằng địa chỉ đối tượng kiểu con

```
GradedActivity * ptr = new PassFailExam(100, 25, 70);
cout << ptr->getScore() << endl;
cout << ptr->getLetterGrade() << endl << endl;
```

```
75
P
```

- Lưu ý: Song con trỏ lớp cha chỉ biết thành viên lớp cha
  - không thể dùng con trỏ cha để gọi hàm không có ở lớp cha
- Hãy coi con trỏ như cái điều khiển, còn đối tượng là cái TV.
  - Kiểu con trỏ thì xác định danh sách các nút mà điều khiển bấm được – chính là danh sách hàm của lớp GradedActivity
  - Kiểu của đối tượng được trỏ tới sẽ cho biết bấm nút đó sẽ hiện ra gì trên TV. Nhiều loại TV cho nhiều phản ứng khác nhau

Ghi đè vs. Tái định

- Hàm con để ghi đè hàm cha phải có 5 điều kiện:
  - giống tên
  - giống danh sách tham số
  - kiểu trả về hoặc giống hoặc dẫn xuất từ kiểu trả về của hàm cha
  - cùng khai báo const hoặc cùng không khai const
  - hàm lớp cha phải khai báo là ảo
- Còn hàm con muốn tái định hàm cha thì chỉ cần tiêu chí đầu
- Hệ quả của tái định là trên đối tượng con luôn chỉ gọi được hàm con, không thể gọi được hàm cha (trừ khi nêu tên đầy đủ)
- Hệ quả của ghi đè là gọi hàm trên đối tượng cha mà có lúc lại chạy thành hàm lớp con

VD: Animal, Cat, Dog, Duck – Đa hình

Lớp Animal có các thành viên sau

- Biến tên, được để private
- Hàm tạo khởi tạo tên, có đối mặc định
- hàm ảo speak in ra : I am <tên>

Các lớp con Cat, Dog, Duck lần lượt ghi đè hàm speak để in ra:

- Meow, I am <tên>
- Woff, woff, I am <tên>
- Quack, quack, I am <tên>

Trong main:

- tạo chó, mèo, vịt, bảo chúng kêu
- tạo 3 biến tham chiếu Lớp Animal gắn với chó, mèo, vịt, bảo chúng kêu
- tạo 3 con trỏ Animal, cấp phát động chó, mèo, vịt, bảo chúng kêu
- tạo mảng 6 con trỏ Animal, khởi tạo 3 con mèo, 2 vịt, 1 chó, vài con cấp phát động, vài con tĩnh, sau đó bảo tất cả lần lượt kêu
- tạo hàm tên là introduce nhập vào một con Animal, trong hàm này in ra “What’s your name?” rồi bảo con animal đó kêu. Hàm main sẽ gọi hàm introduce này, truyền vào chó, mèo, vịt

VD: Animal, Cat, Dog, Duck – Đa hình

```
class Animal{
    string name;
public:
    Animal(string name = "")
    : name(name){}
    virtual void speak() const
    { cout << " Toi la " << name << endl; }
};
class Duck
: public Animal{
public:
    Duck(string name = ""): Animal(name){}
    void speak() const {
        cout << "Quack, ";
        Animal::speak ();
    }
};
    void introduce( const Animal &a){
        cout <<"What's your name: " <<endl;
        a.speak();
    }
```

```
class Cat: public Animal{
public:
    Cat(string name = "")
    : Animal(name){}
    void speak() const {
        cout << "Meow, ";
        Animal::speak ();
    }
};
```

```
class Dog: public Animal{
public:
    Dog(string name = "")
    : Animal(name){}
    void speak() const {
        cout << "Woff, ";
        Animal::speak ();
    }
};
```

```
int main(){
    // câu a
    Cat cat ("Bibi");
    Dog dog ("Toto");
    Duck duck ("Chichi");
    cat.speak();
    dog.speak();
    duck.speak();
```

```
    // câu b
    Animal  & o1 = cat,
            & o2 = dog,
            & o3 = duck;
    o1.speak();
    o2.speak();
    o3.speak();
```

```
    // câu c
    Animal  * p1 = new Cat ("Mi"),
            * p2 = new Dog ("Vang"),
            * p3 = new Duck("Ken");
    p1->speak();
    p2->speak();
    p3->speak();
```

```
    // câu d
    Animal * a[6] = {new Cat ("Bibi"), &cat,
                     new Cat ("Mimi"), new Dog ("Vang"),
                     new Dog ("Xam"), new Duck("Kiki")};
    for (int i = 0; i<6; i++)
        a[i]->speak();

    // câu e
    introduce(o1);
    introduce(o2);
    introduce(o3);

    introduce(*p1);
    introduce(*p2);
    introduce(*p3);

    for (int i = 0; i<6; i++)
        introduce(*a[i]);
```

VD: Ship & CruiseShip & CargoShip

Viết lớp Ship (tàu thủy) có:

- Tên con tàu
- Năm xuất xưởng
- Một hàm khởi tạo và các hàm set/get
- Một hàm ảo tên là print để in ra tên con tàu

Viết class CruiseShip (tàu đu lịch) kế thừa từ class Ship có:

- Số hành khách tối đa
- Một hàm khởi tạo và các hàm set/get
- Hàm print ghi đè (override) hàm print của lớp cha in ra tên con tàu và số hành khách

Viết lớp CargoShip (tàu hàng) kế thừa class Ship có:

- Số tấn hàng chở được
- Một hàm khởi tạo và các hàm set/get
- Hàm print ghi đè (override) hàm print của lớp cha in ra tên con tàu và số hàng chở được

Viết chương trình minh họa đơn giản 3 lớp trên

VD: Ship & CruiseShip & CargoShip

```
class Ship{
    string name;
    int year;
public:
    void setInfor(string n, int y){
        name = n;
        year = y;
    }
    Ship(string name = "", int year = 2000)
    { setInfor(name, year);}

    string getName() const{ return name; }
    int getYear() const {return year; }
    virtual void print() const
    { cout << "Ten tau: " << name << endl; }
};
```

```
class CruiseShip: public Ship{
    int limit;
public:
    int setLimit (int lim)
    { limit = lim; }

    CruiseShip(string name = "",
        int year = 2000, int lim = 2000)
    : Ship(name, year), limit(lim){}

    int getLimit() const
    { return limit; }
    void print() const{
        Ship::print();
        cout << limit << endl;
    }
};
```

```
class CargoShip: public Ship{
    int weightload;
public:
    int setWeight(int w)
    { weightload = w; }

    CargoShip(string name = "",
        int year = 2000, int load = 2000)
    : Ship(name, year), weightload(load){}

    int getWeight() const
    { return weightload; }
    void print() const{
        Ship::print();
        cout << weightload << endl;
    }
};
```

```
int main(){
    Ship * a[] = {new CruiseShip("Hai Duong 2", 1998, 200),
                  new CargoShip("Phuong Dong", 2005, 500)};
    for (int i = 0; i<2; i++)
        a[i]->print();

    return 0;
}
```

```
Ten tau: Hai Duong 2
200
Ten tau: Phuong Dong
500
```

Hàm huỷ ảo

- Rất nên để hàm hủy là ảo nếu lớp đó có thể trở thành một lớp cha
- Nếu không, trình biên dịch sẽ kết tĩnh trên hàm huỷ, gọi hàm huỷ cha nghĩa là sẽ chỉ gọi đúng hàm huỷ cha. Nếu đối tượng lại thực sự thuộc lớp con, thì khi ấy chỉ huỷ được phần lõi (??)
- Còn nếu để hàm huỷ ảo, khi gọi huỷ cha sẽ có lúc chạy được huỷ con, và huỷ con rồi cũng sẽ gọi huỷ cha, cho nên đối tượng được huỷ một cách đầy đủ từ ngoài và trong (??)

```
Cat * c = new ScottishFold(“Clark Kent”, 3, 5);
return 0;
```

Đây là lúc hàm huỷ đối tượng được gọi. Nếu hàm huỷ cha là ảo, thì hảm huỷ con sẽ được thực thi, và chạy hàm huỷ con nghĩa là chạy cả hàm cha lẫn hàm con

Hàm ảo thuần tuý và lớp trừu tượng

- Hàm ảo thuần tuý (pure virtual function):

  virtual void Y() = 0; // hoàn toàn chưa có định nghĩa
- Lớp chứa hàm ảo thuần túy sẽ trở thành lớp trừu tượng. Lớp trừu tượng chỉ là một ý niệm mơ hồ, nên không thể sinh đối tượng.
- Các lớp con nếu không ghi đè hàm ảo thuần tuý ở lớp cha thì cũng bị thành lớp trừu tượng (??)
- Nếu muốn được phép sinh đối tượng thì lớp con bắt buộc phải ghi đè MỌI hàm ảo thuần tuý nó kế thừa

VD: BasicShape & Circle & Rectangle

Viết lớp thuần ảo (lớp trừu tượng) tên là BasicShape mô tả một hình cơ bản trong hình học, gồm các thành viên:

- 1 thuộc tính private là area chưa diện tích của hình.
- Hàm setArea cập nhật cho biến area
- hàm getArea trả lại giá trị của area
- hàm calcArea là hàm thuần ảo tính diện tích

Viết lớp Circle (hình tròn) kế thừa từ BasicShape có:

- centerX, centerY là tọa độ tâm hình tròn và radius là bán kính hình tròn
- hàm khởi tạo 3 tham số ứng với 3 thuộc tính trên
- getter cho 3 thuộc tính
- cài đặt hàm calcArea thực hiện tính diện tích và trả lại giá trị

VD: BasicShape & Circle & Rectangle

Viết lớp Rectangle (hình chữ nhật) kế thừa từ BasicShape có:

- thuộc tính dài và rộng
- hàm khởi tạo 2 tham số ứng với 2 thuộc tính dài và rộng
- 2 getter và 2 setter cho cả 2 thuộc tính
- cài đặt hàm calcArea thực hiện tính diện tích và trả lại giá trị

Viết một chương trình đơn giản minh họa 3 lớp trên

VD: BasicShape & Circle & Rectangle

```
class BasicShape{
    float area;
    virtual float calcArea() const = 0;
protected:
    void setArea (){area = calcArea(); }
public:
    float getArea() const {return area;}
    virtual ~BasicShape(){}
};
```

```
class Circle: public BasicShape{
    float centerX, centerY, radius;
    float calcArea() const {
        return 3.14 * radius * radius;
    }
public:
    void setAll(float x, float y, float r){
        centerX = x;
        centerY = y;
        radius = r;
        setArea();
    }

    Circle (float x = 0, float y = 0, float r = 0)
    { setAll(x,y,r); }

    float getX() const { return centerX; }
    float getY() const { return centerY; }
    float getRadius() const { return radius; }
};
```

Hàm calcArea lớp cha phải có từ khoá **virtual**. Còn hàm calcArea lớp con ghi đè thì phải có cùng tên, kiểu trả về, danh sách tham số và cả từ khoá **const**

```
class Rectangle: public BasicShape{
    float length, width;
    float calcArea() const {
        return length*width;
    }
public:
    Rectangle (float x = 0, float y = 0)
    { setAll(x,y); }

    void setAll(float l, float w){
        length = l;
        width = w;
        setArea();
    }

    float getLength() const
    { return length; }
    float getWidth() const
    { return width; }
};
```

```
int main(){
    // BasicShape b; // error
    // BasicShape * p = new BasicShape; // error

    BasicShape * p1 = new Circle (3,4,5);
    BasicShape * p2 = new Rectangle (3,4);
    cout << p1->getArea() << endl;
    cout << p2->getArea() << endl;

    return 0;
}
```

```
78.5
12
```

VD: NhanVien & NhanVienSanXuat & NhanVienVanPhong

NhanVien & NhanVienSanXuat & NhanVienVanPhong

- Cho lớp trừu tượng NhanVien, gồm các thành viên: Thuộc tính private: mã nhân viên, lương Hàm ảo thuần tuý tinhLuong nhằm tính ra lương, sẽ được cài ở lớp con Hàm setInfor thiết lập giá trị cho mọi thuộc tính cần thiết Hàm ảo print in ra mã nhân viên, lương

- Lớp NhanVienSanXuat kế thừa NhanVien, có: Thuộc tính private: số sản phẩm, tiền công một sản phẩm override tinhLuong trả về tiền lương = số sản phẩm * tiền công một sản phẩm Hàm setInfor thiết lập giá trị cho mọi thuộc tính cần thiết override hàm print in ra mã nhân viên, lương, số sản phẩm, tiền công một sản phẩm
- Lớp NhanVienVanPhong kế thừa NhanVien, có: Thuộc tính private: số ngày công, lương cơ bản override tinhLuong trả về trả về tiền lương bằng số ngày công * lương cơ bản / 30. Hàm setInfor thiết lập giá trị cho mọi thuộc tính cần thiết override hàm print in ra mã nhân viên, lương, số ngày công, lương cơ bản
- Viết hàm main tạo một mảng 5 phần tử NhanVien, có cả NhanVienSanXuat và NhanVienVanPhong, sau cùng in ra thông tin cả mảng

VD: NhanVien & NhanVienSanXuat & NhanVienVanPhong

```
class NhanVien{
    string ma;
    double luong;
    virtual double tinhLuong() const = 0;
protected:
    void setInfor(string ma1){
        ma = ma1;
        luong = tinhLuong();
     }

public:
    virtual void print() const{
        cout << "Ma nhan vien: "
             << ma << endl
        << "Luong: " << luong << endl;
    }
    virtual~NhanVien(){}
};
```

```
class NhanVienSanXuat: public NhanVien{
    int sosp;
    double tiencong;
    double tinhLuong() const {return sosp *
tiencong; }
public:
    void setInfor(string ma1, int sosp1, double
tiencong1){
        sosp = sosp1;
        tiencong = tiencong1;
        NhanVien::setInfor(ma1);
    }

    NhanVienSanXuat(string ma1, int sosp1, double
tiencong1)
    { setInfor(ma1,  sosp1,  tiencong1); }

    void print() const{
        NhanVien::print();
        cout << "So san pham: " << sosp << endl
             << "Tien cong mot san pham: " <<
tiencong << endl;
    }

};
```

VD: NhanVien & NhanVienSanXuat & NhanVienVanPhong

```
class NhanVienVanPhong: public NhanVien{
    int songaycong;
    double luongcb;
    double tinhLuong() const {return
songaycong*luongcb/30; }
public:
    void setInfor(string ma, int ngaycong,
double luong){
        songaycong = ngaycong;
        luongcb = luong;
        NhanVien::setInfor(ma);
    }

    NhanVienVanPhong(string ma, int ngaycong,
double luong)
    { setInfor(ma, ngaycong, luong); }

    void print() const{
        NhanVien::print();
        cout << "So ngay cong: " << songaycong
<< endl
        << "Luong co ban: " << luongcb <<
endl;
    }
};
```

```
int main(){
    int n = 5;
    NhanVien * a[] = {  new
NhanVienVanPhong("1", 25,  30),
new NhanVienSanXuat("6", 20, 30),
                        new
NhanVienVanPhong("2", 20,  30),
new NhanVienSanXuat("7", 21, 30),
                        new
NhanVienVanPhong("3", 15,  30};

    for (int i = 0; i < n; i++)
        a[i]->print();

    return 0;
}
```

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
- Liên kết động vào thời gian chạy và hàm ảo
- Ghi đè hàm cha và quy trình biểu diễn tính đa hình

Keep calm and carry on