



Đề cương bài giảng

Lập trình hướng đối tượng

Ngành : Công nghệ thông tin

Trình độ đào tạo: Đại học

Người biên soạn : Khoa CNTT

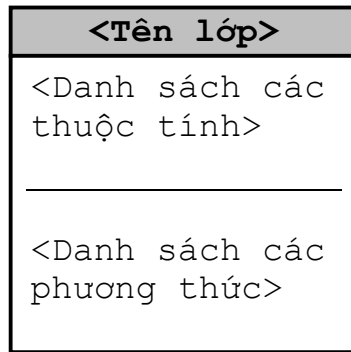
(Lưu hành nội bộ)



BÀI 2: QUAN HỆ KẾT TẬP VÀ PHẠM VI TRUY CẬP

1. Đọc và hiểu sơ đồ lớp

Một lớp được biểu diễn bằng một hình chữ nhật, trong đó ghi tên lớp, danh sách các thuộc tính và danh sách các phương thức của lớp, như hình sau:



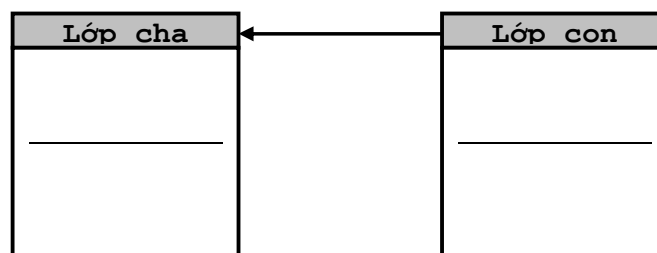
Hình 1. Biểu diễn một lớp

Như vậy, bằng cách nhìn vào sơ đồ lớp, trước hết cần phải xác định: có bao nhiêu lớp cần cài đặt trong bài; với mỗi lớp, cần xác định sơ qua các thuộc tính và phương thức của chúng.

Một sơ đồ lớp không chỉ có duy nhất 1 lớp mà thường chứa nhiều lớp. Các lớp trong sơ đồ không độc lập với nhau mà thường có những mối liên hệ nhất định. Vì vậy, để đọc chính xác một sơ đồ lớp, cần phải nắm vững các mối liên hệ giữa các lớp.

Theo chuẩn UML, hai lớp bất kỳ có thể có nhiều kiểu liên hệ với nhau. Tuy nhiên, để đơn giản, tài liệu này sẽ trình bày 2 kiểu liên hệ chính:

- **Liên hệ cha con:** Nếu hai lớp A và B có liên hệ cha con thì mối liên hệ này được biểu diễn bằng một mũi tên rỗng (sau đây dùng mũi tên đặc !), nối giữa A và B. Chiều của mũi tên chỉ từ lớp con về lớp cha.

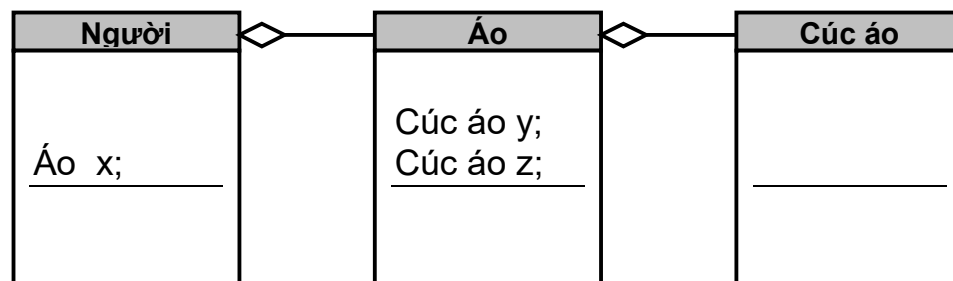


Hình 2. Mũi tên chỉ mối liên hệ cha – con

- **Liên hệ kết tập (hàm chứa):** Bạn là một đối tượng thuộc lớp “Người”. Chiếc áo bạn đang mặc chỉ là một thuộc tính của bạn; tuy nhiên nó lại là một đối tượng thuộc lớp áo. Vậy giữa lớp “Áo” và lớp “Người” có quan hệ với nhau như thế nào?

Trên thực tế có rất nhiều mối quan hệ kiểu như vậy. Chẳng hạn với lớp áo, những chiếc cúc gắn trên đó chỉ là các thuộc tính của áo, nhưng nó lại là các đối tượng thuộc lớp “Cúc áo”.

Khi đó, ta nói: lớp “Cúc áo” kết tập vào lớp “Áo” và lớp “Áo” kết tập vào lớp “Người”. Để biểu diễn mối quan hệ giữa hai lớp A và B trong đó B kết tập vào A, người ta dùng một đường nối giữa A và B, đầu phía A, người ta vẽ một hình thoi rỗng. Vì vậy, có thể biểu diễn mối quan hệ giữa 3 lớp “Người”, “Áo”, “Cúc áo” theo hình sau:

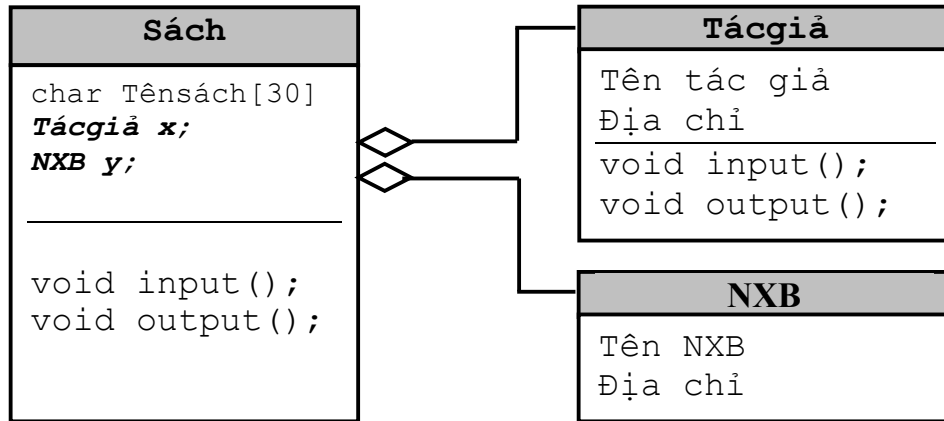


Hình 3. Mối quan hệ kết tập.

Để thấy x là một thuộc tính của lớp “Người”, tuy nhiên nó lại là một đối tượng thuộc lớp “Áo” (hay còn gọi là áo x); và thuộc tính này đã thể hiện mối quan hệ kết tập của lớp “Áo” vào lớp “Người”. Khi đọc sơ đồ lớp, hãy quan tâm tới những thuộc tính kiểu như vậy. Tương tự thuộc tính y, z của lớp “Áo” thể hiện sự kết tập của lớp “Cúc áo” vào lớp “Áo”. Những thuộc tính như vậy sẽ không có kiểu nguyên thủy (int, float, char...) mà có kiểu là một tên lớp (x có kiểu Áo; y, z có kiểu Cúc áo).

Trở lại ví dụ chính, để thấy sơ đồ gồm 3 lớp: Sách, Tác giả, NXB và hai lớp Tác giả, NXB kết tập vào lớp Sách. Tuy nhiên, trong lớp Sách, các bạn hãy chú ý tới 2 thuộc tính thể hiện sự kết tập này: “Tác giả” (1) , “Nhà xuất bản” (2). Tác giả của một cuốn sách phải là một đối tượng thuộc lớp “Tác giả”; vì vậy ta cần hiểu (1) chính là một đối tượng thuộc lớp “Tác giả”; cách hiểu chính xác phải là: Tác giả x; Tương tự, với thuộc tính (2), ta cần hiểu đó chính là NXB y; với x, y là hai đối tượng mà ta tự đặt tên, thuộc lớp “Tác giả” và lớp “NXB”.

Vậy cụ thể, sơ đồ lớp trên được hiểu như sau:



Hình 4. Cách hiểu rõ hơn các thuộc tính kết tập.

2. Xác định thứ tự cài đặt các lớp trong sơ đồ

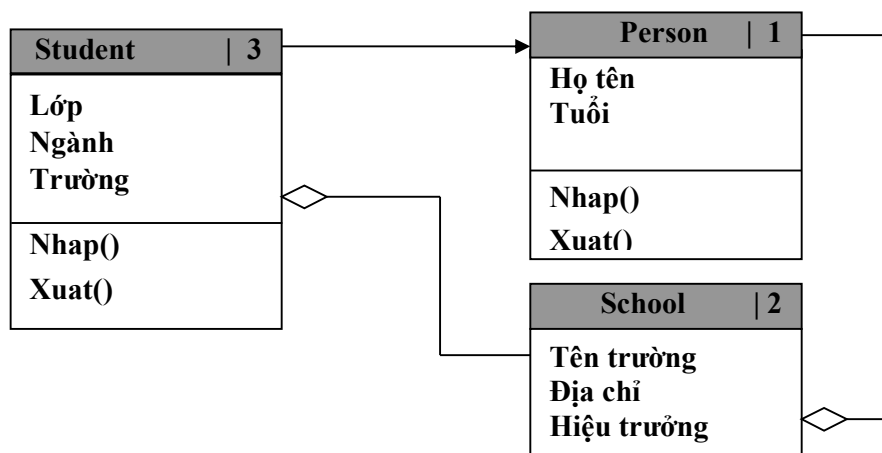
Một việc làm không tốn nhiều thời gian mà lại rất quan trọng khi cài đặt sơ đồ lớp là xác định đúng thứ tự cài đặt các lớp trong một sơ đồ. Việc xác định sai thứ tự cài đặt trong một sơ đồ phức tạp có thể gây ra hậu quả xấu. Để làm được việc này, hãy chú ý tuân thủ quy tắc sau:

[1]. Cha trước, con sau

[2]. Không hình thoi trước, có hình thoi sau.

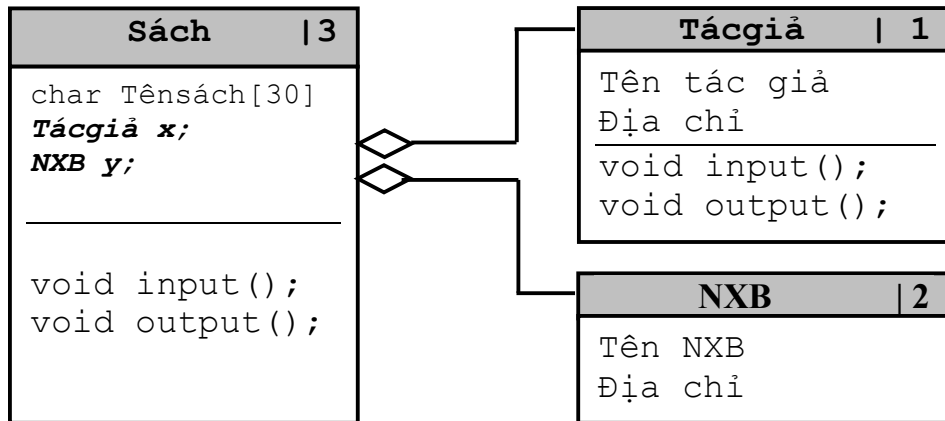
Quy tắc này có nghĩa là: với 2 lớp có quan hệ cha con, phải cài đặt (hoặc khai báo) lớp cha trước, sau đó cài đặt lớp con. Với hai lớp có quan hệ kết tập thể hiện bằng một đường nối với 1 đầu của nó có chứa hình thoi, ta hãy cài đặt lớp ở phía không có hình thoi trước, có hình thoi sau.

Ví dụ với sơ đồ sau, thứ tự cài đặt được xác định như sau (số thứ tự được đánh kèm với mỗi lớp):



Hình 5. Thứ tự cài đặt của các lớp trong sơ đồ

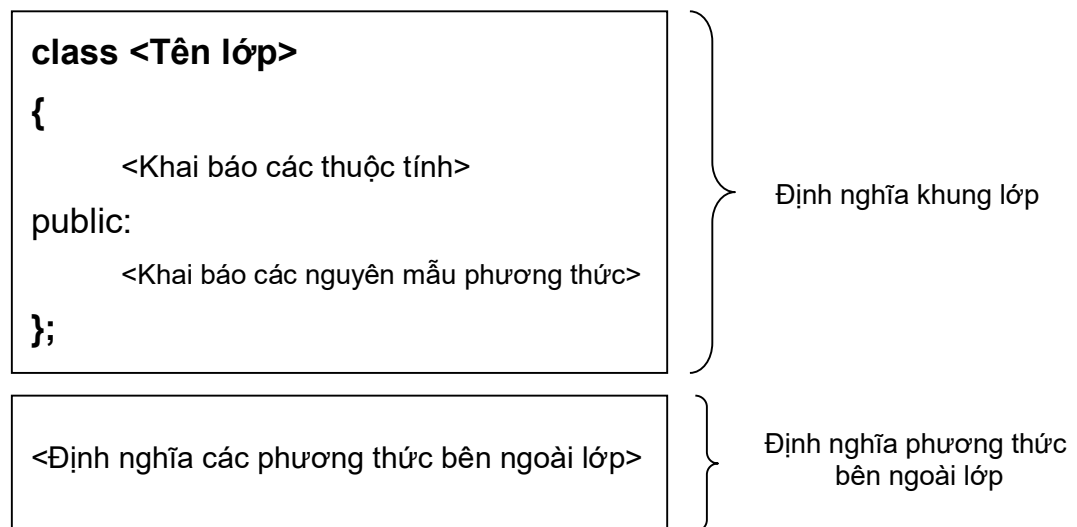
Như vậy, dễ dàng xác định được thứ tự cài đặt cho các lớp trong ví dụ minh họa. Trong sơ đồ này, vì lớp “Tác giả” và lớp “NXB” không có liên hệ trực tiếp gì với nhau nên có thể cài đặt lớp nào trước đều được:



Hình 6. Thứ tự cài đặt các lớp của ví dụ minh họa

3. Cài đặt một lớp đơn giản

Một lớp thường được cài đặt theo cách sau (để không mất tập trung, tác giả chỉ trình bày 1 cách phổ biến).



Hình 7. Định nghĩa một lớp

[1]. Như vậy ta chia công việc này thành 2 phần: định nghĩa khung lớp và định nghĩa các phương thức ngoài lớp. Với khung lớp, mỗi thuộc tính là một

biến. Biến này có thể có kiểu nguyên thủy (int, float, char...) hoặc kiểu lớp (nếu nó thể hiện quan hệ kết tập – xin xem phần trên).

[2]. Trước phần <Khai báo các nguyên mẫu phương thức> ta đặt từ khóa “public:”. Từ khóa này nhằm xác định phạm vi truy cập “công cộng” cho các phương thức của lớp. Mỗi thuộc tính hay phương thức ta đều có thể chỉ định 1 phạm vi truy cập cho chúng với 3 mức độ truy cập là **private** (riêng tư – chỉ được truy cập trong nội bộ lớp), **public** (công cộng – được truy cập ở mọi nơi) và **protected** (được bảo vệ - chỉ được truy cập trong nội bộ lớp và các lớp con kế thừa lớp này).

Cách đặt phạm vi truy cập: ta viết

<Tên phạm vi truy cập> :

Phạm vi này sẽ ảnh hưởng tới tất cả các thuộc tính, phương thức đứng sau nó cho tới khi gặp một phạm vi truy cập khác. Các thuộc tính không được chỉ định phạm vi truy cập thì mặc định có phạm vi truy cập **private** (riêng tư).

Mỗi phương thức là một hàm được viết theo quy tắc viết hàm của C++ như sau:

```
<Kiểu trả về> <Tên phương thức> ([Danh sách các đối])  
{  
    //Thân phương thức  
}
```

[3]. Với sơ đồ lớp trong ví dụ minh họa, tất cả các phương thức của các lớp đều có kiểu “void” và tên phương thức đã cho trước (input, output), vì vậy một phương thức input() có thể được viết như sau:

```
void input( )  
{  
    //Thân phương thức  
}
```

Phương thức này không có giá trị trả về (nếu vậy thì ta viết kiểu trả về là void); phương thức cũng không có đối vào (ta để trống nhưng không thể thiếu các dấu mở đóng ngoặc).

[4]. Nguyên mẫu của phương thức chính là dòng đầu tiên của phương thức, kèm theo dấu chấm phẩy. Ví dụ với phương thức input ở trên thì nguyên mẫu của nó là: **void input();**

[5]. Khi định nghĩa phương thức ngoài lớp thì cần thêm **<Tên lớp>::** vào trước tên phương thức để xác định rõ là phương thức này thuộc lớp nào. Ví dụ với phương thức input của lớp Sách, ta viết:

```
void Sach::input()
{
    //Thân phương thức
}
```

Với phương thức input của lớp “Tác giả”, ta viết:

```
void Tacgia::input()
{
    //Thân phương thức
}
```

Vậy ta có thể cài đặt cho lớp “Tác giả” như sau:

Khung lớp: chú ý kết thúc khung lớp có dấu “;”

```
class Tacgia
{
    char TenTg[30];
    char Diachi[50];
public:
    void input();
    void output();
};
```

Cài đặt phương thức ngoài lớp:

```
void Tacgia::input()
{
    //Thân phương thức input
}

void Tacgia::output()
{
    //Thân phương thức output
}
```

Tương tự lớp NXB, do lớp này không có phương thức nên việc cài đặt chúng hết sức đơn giản (chỉ có khung lớp).

```
class NXB
{
    char TenNXB[30];
    char Diachi[50];
};
```

Và lớp Sách được cài đặt với khung lớp như sau:

```
class Sach
{
    char Tensach[30];
    Tacgia x;
    NXB y;
public:
    void input();
    void output();
};
```

và các phương thức ngoài lớp

```
void Sach::input()
{
    //Thân phương thức input
}

void Sach::output()
{
    //Thân phương thức output
}
```


Về nội dung của các phương thức (thân phương thức) thường rất đơn giản. Với ví dụ trên ta chỉ việc tiến hành nhập, xuất các thuộc tính của lớp tương ứng. Với thuộc tính kiểu xâu, ta dùng lệnh gets(Thuộc tính); để nhập và chú ý làm sạch bộ đệm bàn phím sau khi nhập bằng lệnh fflush(stdin); để xuất, ta dùng lệnh cout<< “nội dung cần xuất”; và dùng endl để xuống dòng. Bây giờ, hãy bắt đầu với lớp “Tác giả”:

```
void Tacgia::input()
{
    cout<< “Tên tác giả:”; gets(TenTg); fflush(stdin);
    cout<< “Địa chỉ:”; gets(Diach); fflush(stdin);
}

void Tacgia::output()
{
    cout<< “Tên tác giả: “<< TenTg<<endl;
    cout<< “Địa chỉ: ”<<Diachi<<endl;
}
```

Nhưng với lớp Sách, do có quan hệ kết tập của 2 lớp “Tác giả” và “NXB” vào nên khi nhập, xuất ta cần xác định rõ lớp Sách này có bao nhiêu thuộc tính. Hãy nhìn vào sơ đồ lớp, dễ thấy lớp Sách có 3 thuộc tính chính gồm: Tên sách, x, y; nhưng nếu phân tích chi tiết thì x là một đối tượng thuộc lớp Tác giả nên nó có hai thuộc tính là x.TenTg và x.Diachi. Tương tự y cũng có 2 thuộc tính là y.TenNXB và y.Diachi. Vì vậy, một cách chi tiết, lớp Sách phải có 5 thuộc tính.

Nhưng vì x thuộc lớp “Tác giả” nên nó có cả hai phương thức input() và output() của lớp “Tác giả”. Phương thức này có nhiệm vụ nhập, xuất cho các thuộc tính x.TenTg và x.Diachi. Vì vậy, để nhập hai thuộc tính này ta chỉ cần gọi phương thức x.input(); . Việc sử dụng phương thức của đối tượng x để nhập các thuộc tính cho nó được gọi là “sử dụng lại code” hay reuse trong quan hệ kết tập. Cách sử dụng lại code này được viết đơn giản:

<Tên đối tượng>. <Phương thức>;

Tuy nhiên, với thuộc tính (đối tượng) y; ta lại không có được sự tiện lợi như vậy. Đơn giản vì y thuộc lớp NXB, mà lớp này lại không có các phương thức input(), output(). Do vậy, trong lớp Sách này, ta không thể gọi phương thức

để nhập, xuất cho hai thuộc tính y.TenNXB và y.Diachi mà cần nhập, xuất trực tiếp chúng.

Vậy hai phương thức input() và output() của lớp Sách được viết như sau:

```
void Sach::input()
{
    cout<< "Tên sách:"; gets(Tensach); fflush(stdin);
    x.input();      //dòng này nhập 2 thuộc tính x.TenTg và x.Diachi
    cout<< "Tên NXB:"; gets(y.TenNXB); fflush(stdin);
    cout<< "Địa chỉ NXB:"; gets(y.Diachi); fflush(stdin);
}

void Sach::output()
{
    cout<< "Tên sách: "<<Tensach<<endl;
    x.output();      //dòng này xuất 2 thuộc tính x.TenTg và x.Diachi
    cout<< "Tên NXB:" <<y.TenNXB<<endl;
    cout<< "Địa chỉ NXB:" <<y.Diachi<<endl;
}
```

Vậy toàn bộ code cho sơ đồ lớp trong ví dụ minh họa được viết lại là:

<pre> class Tacgia { char TenTg[30]; char Diachi[50]; public: void input(); void output(); }; void Tacgia::input() { cout<< "Tên tác giả:"; gets(TenTg); fflush(stdin); cout<< "Địa chỉ:"; gets(Diachi); fflush(stdin); } void Tacgia::output() { cout<< "Tên tác giả: "<< TenTg<<endl; cout<< "Địa chỉ: "<<Diachi<<endl; } </pre>	1
<pre> class NXB { char TenNXB[30]; char Diachi[50]; }; </pre>	2
<pre> class Sach { char Tensach[30]; Tacgia x; NXB y; public: void input(); void output(); }; void Sach::input() { cout<< "Tên sách:"; gets(Tensach); fflush(stdin); x.input(); //dòng này nhập 2 thuộc tính x.TenTg và x.Diachi cout<< "Tên NXB:"; <u>gets(y.TenNXB);</u> fflush(stdin); cout<< "Địa chỉ NXB:"; <u>gets(y.Diachi);</u> fflush(stdin); } void Sach::output() { cout<< "Tên sách: "<<Tensach<<endl; x.output(); //dòng này xuất 2 thuộc tính x.TenTg và x.Diachi cout<< "Tên NXB:" <<<u>y.TenNXB</u><<endl; cout<< "Địa chỉ NXB:" <<<u>y.Diachi</u><<endl; } </pre>	3

Việc cài đặt sơ đồ lớp trong ví dụ minh họa 1, về cơ bản đã hoàn thành. Tuy nhiên đoạn code trên đã mắc phải một lỗi cơ bản. Nếu copy đoạn code trên

vào C++ và soát lỗi ta sẽ gặp 4 thông báo lỗi giống nhau. (phần in đậm và gạch chân trong phương thức Sach::input() và Sach::output()). Lỗi này được hiểu đơn giản như sau:

Do hai phương thức Sach::input() và Sach::output() là thuộc lớp Sach, nhưng trong thân của nó lại truy cập tới hai thuộc tính TenNXB và Diachi của lớp NXB. Hai thuộc tính này có phạm vi truy cập là riêng tư (private) nên không thể được truy cập từ ngoài lớp NXB. Để khắc phục lỗi này, xin xem phần sau: cách truy cập thuộc tính riêng tư.

4. Cách truy cập thuộc tính riêng tư

[1]. Chuyển thuộc tính riêng tư thành công cộng:

Như trên, lỗi của 2 phương thức Sach::input() và Sach::output() dễ dàng khắc phục bằng cách đơn giản là đặt phạm vi truy cập công cộng (public) cho hai thuộc tính TenNXB và Diachi trong lớp NXB. Như vậy lớp NXB được viết lại là:

```
class NXB
{
public:
    char TenNXB[30];
    char Diachi[50];
};
```

Nhưng như vậy, về bản chất không phải là cách để truy cập thuộc tính riêng tư, mặt khác, các thuộc tính cũng hiếm khi được đặt phạm vi truy cập public để đảm bảo an toàn và tính riêng tư về dữ liệu của đối tượng.

[2]. Bổ sung các phương thức Set/ Get cho mỗi thuộc tính:

Chẳng hạn khi muốn truy cập thuộc tính TenNXB của lớp NXB từ lớp Sách, ta bổ sung phương thức GetTenNXB() trả về TenNXB và phương thức SetTenNXB(char * Ten) để gán Ten vào TenNXB. Lớp NXB được viết lại là:

```
class NXB
{
    char TenNXB[30];
    char Diachi[50];
public:
    char * GetTenNXB()
    { return TenNXB;}

    void SetTenNXB(char* Ten)
    {strcpy(TenNXB, Ten);}
};
```

và khi truy cập thuộc tính TenNXB, ta không truy cập trực tiếp thuộc tính này mà truy cập gián tiếp qua hai phương thức Set/ Get ở trên. Ví dụ với phương thức Sach::output(), nếu muốn xuất TenNXB, thay vì viết cout<<y.TenNXB<<endl; ta viết: cout<<y.GetTenNXB()<<endl;

Thực chất của cách này là truy cập gián tiếp vào thuộc tính riêng tư thông qua hai phương thức Set/ Get là hai phương thức công cộng mà ta bổ sung vào. Cách này không được tác giả trình bày chi tiết do không khuyến khích, đề nghị đọc thêm tài liệu nếu cần.

[3]. Lớp bạn, hàm bạn

Để lớp Sách có thể truy cập được thuộc tính riêng tư của lớp NXB, cách đơn giản nhất là cho lớp Sách làm bạn với lớp NXB. Nếu lớp A là bạn của lớp B thì A có toàn quyền truy cập thuộc tính riêng tư của B. Điều ngược lại không đúng.

Vậy làm thế nào để lớp Sách là bạn của lớp NXB? rất đơn giản, trong lớp NXB, ta khai báo: **friend class Sach;**

Vậy lớp NXB được viết lại là:

```
class NXB
{
    char TenNXB[30];
    char Diachi[50];
    friend class Sach;
};
```

Vậy toàn bộ đoạn mã cài đặt sơ đồ lớp của ví dụ minh họa đã hoàn tất nhờ một thao tác đơn giản là đặt lớp Sách là bạn của lớp NXB.

Vậy còn hàm bạn? chúng ta hãy xét tiếp yêu cầu của ví dụ minh họa. Với ví dụ này, sau khi cài đặt xong sơ đồ lớp, việc tiếp theo là: *“Hãy cài đặt hàm main nhập vào một danh sách gồm n cuốn sách, in ra các sách do nhà xuất bản “Thanh Niên” ấn hành.”*

Toàn bộ công việc này được giải quyết trong hàm main. Sau khi định nghĩa lớp, việc tiếp theo là viết hàm main để sản sinh các đối tượng thuộc các lớp vừa định nghĩa và sử dụng các đối tượng này để hoàn thành các công việc yêu cầu của đề bài.

Với ví dụ này, ta chỉ đơn giản là khai báo 1 mảng các đối tượng thuộc lớp Sách và nhập danh sách này. Sau đó duyệt mảng và in ra các sách có NXB là “Thanh Niên”. Hãy xem code sau:

```

void main()
{
    Sach a[100]; int n;

    //Nhập danh sách – trước tiên nhập n
    cout<< “n=”; cin>>n;
    for(int i=0; i<n; i++)
        a[i].input();

    // Xuất thông tin sách của NXB Thanh Niên
    for( i=0; i<n; i++)
        if (strcmp(a[i].y.TenNXB, “Thanh Niên”) == 0)
            a[i].output();
}

```

Đoạn code trên cũng mắc lỗi truy cập tương tự như trên. Vì hàm main nằm ngoài các lớp Sach và NXB nhưng lại truy cập tới thuộc tính y (của lớp Sach) và TenNXB (của lớp NXB - đoạn in đậm và gạch chân) nên việc báo lỗi truy cập là hiển nhiên.

Để khắc phục lỗi này, tương tự, ta sẽ cho hàm main là bạn của lớp Sách và lớp NXB.

Nhưng hàm main, do đặc quyền của nó, nó không bao giờ chịu làm bạn với bất kỳ lớp thông thường nào. Vậy xử lý vấn đề này thế nào?

Đơn giản là ta sẽ tách đoạn code mà trong đó có truy cập thuộc tính riêng tư thành 1 hàm riêng (nằm ngoài hàm main); với ví dụ trên, đoạn code đó là đoạn xuất thông tin sách của NXB Thanh Niên. Vậy hàm main giờ được tách thành 2 hàm như sau:

```

void In(Sach a[100], int n)
{
    for( int i=0; i<n; i++)
        if (strcmp(a[i].y.TenNXB, “Thanh Niên”) == 0)
            a[i].output();
}

void main()
{
    Sach a[100]; int n;

    //Nhập danh sách – trước tiên nhập n
    cout<< “n=”; cin>>n;
    for(int i=0; i<n; i++)
        a[i].input();

    // Xuất thông tin sách của NXB Thanh Niên – gọi hàm In
    In(a, n);
}

```

Rõ ràng ta đã đẩy lỗi truy cập từ hàm main sang hàm void In(...). Như vậy hàm main không còn là kẻ gây ra lỗi truy cập nữa, mà lỗi này bị đẩy cho hàm void In(...). Và việc tiếp theo, đơn giản ta cho hàm void In(...) làm bạn với lớp Sách và NXB. Vậy lớp Sách và lớp NXB được viết lại là:

<pre>class NXB { char TenNXB[30]; char Diachi[50]; friend class Sach; friend void In(Sach *a, int n); // ← };</pre>
<pre>class Sach { char Tensach[30]; Tacgia x; NXB y; public: void input(); void output(); friend void In(Sach *a, int n); // ← };</pre>

Vậy để 1 hàm (chẳng hạn hàm void In(...) ở trên) là hàm bạn của 1 lớp (chẳng hạn lớp Sách hoặc NXB) thì đơn giản: Trong thân lớp, ta khai báo: **friend <Nguyên mẫu của hàm>;**

Tuy nhiên, hãy lưu ý 2 vấn đề quan trọng mà có thể gặp sai lầm khi thiết lập hàm bạn cho lớp:

[1]. Nguyên mẫu của hàm không được chứa đối vào là mảng: ví dụ với hàm In ở trên, theo cách hiểu thông thường thì nguyên mẫu của hàm phải là dòng đầu tiên của hàm + dấu “;” tức là: void In(Sach a[100], int n);

Nguyên mẫu này có 2 đối vào là Sach a[100] và int n. Với đối a, nó không hợp lệ do nó là mảng. Vậy ta cần chuyển a thành con trỏ (vì con trỏ lúc này cũng tương đương với mảng) và khai báo hàm bạn sẽ là: **friend void In(Sach *a, int n);**

[2]. Nếu viết: **friend void In(Sach *a, int n);** thì trước đó, ta cần khai báo lớp Sách, nếu không chương trình sẽ báo lỗi chỗ đối vào **Sach * a**, do nó không hiểu Sach là gì (vì chương trình dịch sẽ dịch từ trên xuống và tới đó nó chưa gặp lớp Sach).

Để không bị lỗi này, đơn giản là ta khai báo lớp bạn **friend class Sach;** trước khi khai báo hàm bạn (như trên) và đặt dòng khai báo hàm bạn: **friend void In(Sach *a, int n);** sau dòng khai báo lớp bạn này.

Nhưng trong những trường hợp không có khai báo lớp bạn thì sao? Khi đó ta có thể thêm khai báo nguyên mẫu của lớp Sách trước lớp NXB. Tức là:

```
class Sach; //dòng này khai báo nguyên mẫu lớp Sách
class NXB
{
    char TenNXB[30];
    char Diachi[50];
    friend void In(Sach *a, int n); //vậy có thể để khai báo này lên trên
    friend class Sach;
};
```

Vậy đoạn code hoàn thiện bài ví dụ trên sẽ phải là:

```
class Tacgia
{
    char TenTg[30];
    char Diachi[50];
public:
    void input();
    void output();
};
void Tacgia::input()
{
    cout<< "Tên tác giả:"; gets(TenTg); fflush(stdin);
    cout<< "Địa chỉ:"; gets(Diachi); fflush(stdin);
}
void Tacgia::output()
{
    cout<< "Tên tác giả: "<< TenTg<<endl;
    cout<< "Địa chỉ: "<<Diachi<<endl;
}

class NXB
{
    char TenNXB[30];
    char Diachi[50];
    friend class Sach;
    friend void In(Sach *a, int n);
};

class Sach
{
    char Tensach[30];
    Tacgia x;
    NXB y;
public:
    void input(); void output();
    friend void In(Sach *a, int n);
};
```



```

void Sach::input()
{
    cout<< "Tên sách:"; gets(Tensach); fflush(stdin);
    x.input();           //dòng này nhập 2 thuộc tính x.TenTg và x.Diachi
    cout<< "Tên NXB:"; gets(y.TenNXB); fflush(stdin);
    cout<< "Địa chỉ NXB:"; gets(y.Diachi); fflush(stdin);
}
void Sach::output()
{
    cout<< "Tên sách: "<<Tensach<<endl;
    x.output();          //dòng này xuất 2 thuộc tính x.TenTg và x.Diachi
    cout<< "Tên NXB:" <<y.TenNXB<<endl;
    cout<< "Địa chỉ NXB:" <<y.Diachi<<endl;
}

void In(Sach a[100], int n)
{
    for( int i=0; i<n; i++)
        if (strcmp(a[i].y.TenNXB, "Thanh Niên") == 0)
            a[i].output();
}

void main()
{
    Sach a[100]; int n;

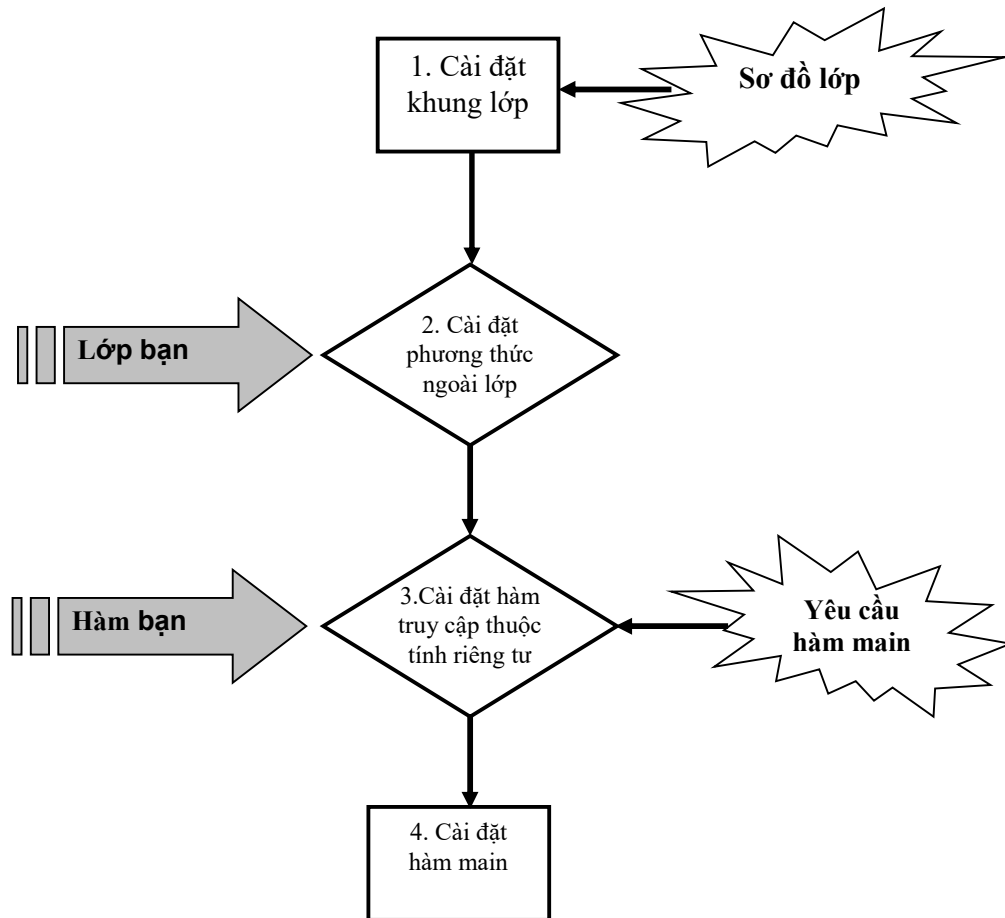
    //Nhập danh sách – trước tiên nhập n
    cout<< "n="; cin>>n;
    for(int i=0; i<n; i++)
        a[i].input();

    // Xuất thông tin sách của NXB Thanh Niên – gọi hàm In
    In(a, n);
}

```

Do trình bày quá chi tiết nên có thể gây khó khăn cho người đọc về tính hệ thống. Rất may là ta không phải lo lắng quá mức về vấn đề đó. Sơ đồ sau đây (tuy chưa hoàn thiện) nhưng nó giúp các bạn hệ thống lại những gì đã được giới thiệu ở phần trên. (xem trang sau)

5. Sơ đồ 4 bước để hoàn thành cài đặt một bài tập dạng sơ đồ lớp

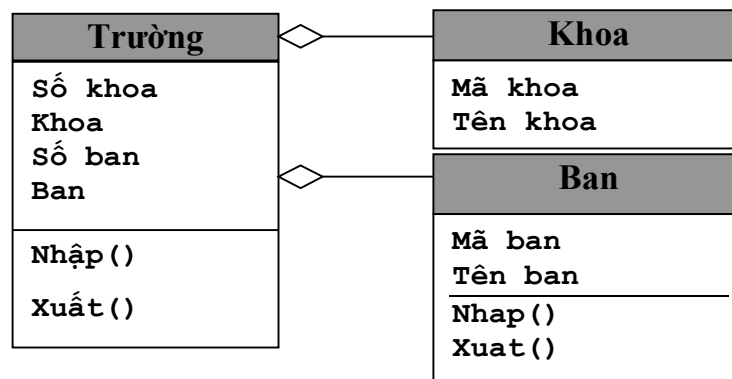


Hình 7. Bốn bước để cài đặt một bài tập dạng sơ đồ lớp.

✎ Kết tập 1- nhiều:

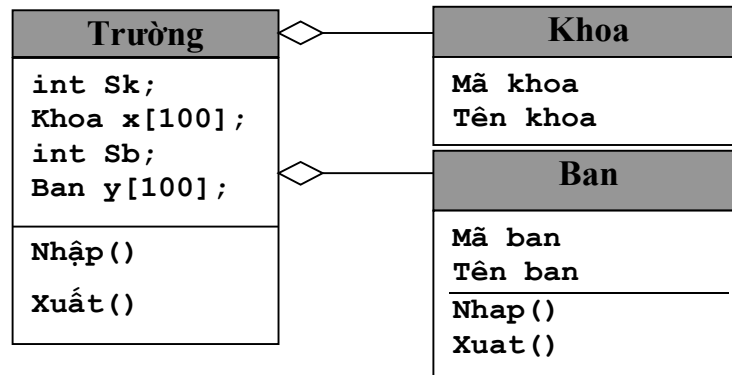
Một số thao tác xử lý tương đối phức tạp có thể cần thiết khi cài đặt các bài tập có quan hệ kết tập 1-nhiều như bài tập sau:

Ví dụ minh họa: Cài đặt lớp theo sơ đồ sau:



Viết chương trình chính nhập vào thông tin của n trường, in ra thông tin của các trường có ít nhất 3 ban hoặc có khoa CNTT.

Theo đề bài, cần hiểu rõ sự kết tập của hai lớp Khoa và Ban vào lớp Trường. Mỗi trường thông thường có nhiều khoa và nhiều ban. Như vậy, ta cần hiểu thuộc tính thể hiện sự kết tập trong lớp Trường (Khoa, Ban) là các mảng đối tượng thuộc lớp Khoa, Ban mà không phải là các đối tượng đơn. Hai thuộc tính Số khoa, Số ban là hai số nguyên thể hiện số khoa, ban thực tế mà mỗi trường có, hay chính là kích thước của 2 mảng đối tượng trên. Vậy sơ đồ được hiểu là:



Hai lớp Khoa, Ban được cài đặt như thông thường mà không gặp phải trở ngại gì:

```

class Khoa
{
    char MaK[30];
    char TenK[30];
};

class Ban
{
    char MaB[30];
    char TenB[30];
public:
    void Nhap();
    void Xuat();
};

void Ban::Nhap()
{
    cout<<"Ma ban ="; gets(MaB); fflush(stdin);
    cout<<"Ten ban ="; gets(TenB); fflush(stdin);
}

void Ban::Xuat()
{
    cout<<"Ma ban:"<<MaB<<endl;
    cout<<"Ten ban:"<<TenB<<endl;
}
    
```

Nhưng lớp Truong, do có 2 mối kết tập 1-nhiều nên khi cài đặt cần chú ý xử lý cho chính xác, đặc biệt là các xử lý hàm bạn, lớp bạn.

```
class Truong
{
    Khoa x[100];
    int Sk;
    Ban y[100];
    int Sb;
public:
    void Nhap();
    void Xuat();
};

void Truong::Nhap()
{
    cout<<"Nhập số khoa:"; cin>>Sk;
    for(int i=0; i<Sk; i++)
    {
        cout<<"Ma khoa:"; gets(x[i].MaK); fflush(stdin);
        cout<<"Ten khoa:"; gets(x[i].TenK); fflush(stdin);
    }

    cout<<"Nhập số ban:"; cin>>Sb;
    for( i=0; i<Sb; i++)  y[i].Nhap();
}

void Truong::Xuat()
{
    for(int i=0; i<Sk; i++)
    {
        cout<<"Ma khoa:"<<x[i].MaK<<endl;
        cout<<"Ten khoa:"<<x[i].TenK<<endl;
    }

    for( i=0; i<Sb; i++)
        y[i].Xuat();
}
```

Nói chung, khi xử lý các thuộc tính kết tập 1-nhiều, vì nó là mảng, nên cần sử dụng các vòng lặp.

Tuy nhiên lớp Truong đã truy cập thuộc tính riêng tư của lớp Khoa nên nó cần là bạn của lớp Khoa. Vậy lớp Khoa được viết lại là:

```
class Khoa
{
    char MaK[30];
    char TenK[30];
    friend class Truong;
};
```

Bây giờ hãy cài đặt chương trình chính. Vì cần in ra thông tin các trường có nhiều hơn 3 ban hoặc có khoa CNTT nên chắc chắn sẽ truy cập tới thuộc tính riêng tư của lớp Truong (thuộc tính Sb) và lớp Khoa (thuộc tính TenK). Vì vậy một hàm In() để thực hiện công việc này là cần thiết:

```
void In(Truong a[100], int n)
{
    for(int i=0; i<n; i++)
    {
        int Check = 0;
        for(int j = 0; j<a[i].Sk; j++)
            if(strcmp(a[i].x[j].TenK, "CNTT")==0)
                Check=1;

        if (a[i].Sb >3 || Check==1)
            a[i].Xuat();
    }
}

void main()
{
    Truong a[100]; int n;

    cout<<"n="; cin>>n;
    for(int i=0; i<n; i++)
        a[i].Nhap();

    //-----
    In(a, n);
}
```

Rõ ràng là việc xử lý trên các thuộc tính kết tập 1-nhiều luôn phức tạp hơn với nhiều vòng lặp cùng các giải thuật được đưa vào. Điều này có thể gây thêm khó khăn cho những thí sinh chưa nắm vững kiến thức.

Mặt khác hàm In cần là bạn của lớp Truong và lớp Khoa. Vậy đoạn mã sau với các bổ sung hàm bạn, lớp bạn sẽ là lời giải cho bài tập trên:

```
class Khoa
{
    char MaK[30];
    char TenK[30];
    friend class Truong;
    friend void In(Truong *a, int n);
};

class Ban
{
    char MaB[30];
    char TenB[30];
public:
    void Nhap();
    void Xuat();
};
```

```

void Ban::Nhap()
{
    cout<<"Ma ban ="; gets(MaB); fflush(stdin);
    cout<<"Ten ban ="; gets(TenB); fflush(stdin);
}

void Ban::Xuat()
{
    cout<<"Ma ban:"<<MaB<<endl;
    cout<<"Ten ban:"<<TenB<<endl;
}

class Truong
{
    Khoa x[100];
    int Sk;
    Ban y[100];
    int Sb;
public:
    void Nhap();
    void Xuat();
    friend void In(Truong *a, int n);
};

void Truong::Nhap()
{
    cout<<"Nhập số khoa:"; cin>>Sk;
    for(int i=0; i<Sk; i++)
    {
        cout<<"Ma khoa:"; gets(x[i].MaK); fflush(stdin);
        cout<<"Ten khoa:"; gets(x[i].TenK); fflush(stdin);
    }

    cout<<"Nhập số ban:"; cin>>Sb;
    for( i=0; i<Sb; i++)
        y[i].Nhap();
}

void Truong::Xuat()
{

```

```

void In(Truong a[100], int n)
{
    for(int i=0; i<n; i++)
    {
        int Check = 0;
        for(int j = 0; j<a[i]. Sk; j++)
            if(strcmp(a[i].x[j].TenK, "CNTT")==0)
                Check=1;

        if (a[i]. Sb >3 || Check==1)
            a[i].Xuat();
    }
}

void main()
{
    Truong a[100]; int n;
    cout<<"n="; cin>>n;
    for(int i=0; i<n; i++)
        a[i].Nhap();

    In(a, n);
}

```

Phần 2: Hướng dẫn làm bài tập dạng phiếu

Việc cài đặt bài tập dạng phiếu về cơ bản giống với việc cài đặt bài tập dạng sơ đồ lớp. Hãy đảm bảo bạn nắm vững cách cài đặt dạng sơ đồ lớp trước khi đọc sang phần này.

1 Ví dụ minh họa

Viết chương trình đơn giản quản lý việc tạo phiếu kiểm kê tài sản trong các phòng ban của một công ty theo phương pháp hướng đối tượng. Yêu cầu các thuộc tính đều đặt phạm vi truy cập riêng tư và chương trình phải có các chức năng sau:

- **Tạo một phiếu kiểm kê:** cho phép nhập các thông tin chung của phiếu (mã phiếu, ngày kiểm kê); thông tin về nhân viên kiểm kê; thông tin về phòng ban đang kiểm kê; thông tin về các tài sản kiểm kê.
- **In ra phiếu kiểm kê theo mẫu sau:**

PHIẾU KIỂM KÊ TÀI SẢN		
Mã phiếu:	<i>PH01.</i>	Ngày kiểm kê: <i>1/1/2007</i>
Nhân viên kiểm kê:	<i>Kiều Thị Thanh</i>	Chức vụ: <i>Kế toán viên</i>
Kiểm kê tại phòng:	<i>Tổ chức hành chính</i>	Mã phòng: <i>PTC</i>
Trưởng phòng:	<i>Hoàng Bích Hảo</i>	
Tên tài sản	Số lượng	Tình trạng
Máy vi tính	5	Tốt
Máy vi tính	3	Hết khấu hao - hỏng
Bàn làm việc	6	Tốt
Số tài sản đã kiểm kê: 3. <i>Tổng số lượng: 14</i>		

Với dạng này, việc đầu tiên là tiến hành chuyển bài toán về dạng sơ đồ lớp. Khi có sơ đồ lớp, việc cài đặt được tiến hành như PHẦN 1.

Để chuyển bài toán về sơ đồ lớp, ta tiến hành thông qua 4 bước sau:

[1]. Liệt kê các thuộc tính trong phiếu:

Các thuộc tính trong phiếu là những thành phần mà nội dung của chúng có thể khác nhau giữa 2 phiếu. Ví dụ, với 2 phiếu bất kỳ, thì dòng chữ “PHIẾU KIỂM KÊ TÀI SẢN” là giống nhau

Stt	Tên thuộc tính	Ký hiệu
1	Mã phiếu	MaPh
2	Ngày kiểm kê	Ngay
3	Nhân viên kk	TenNV
4	Chức vụ	CVNV
5	Phòng kiểm kê	TenP
6	Mã phòng kk	MaP
7	Trưởng phòng	TruongP
8	Tên tài sản	TenTS
9	SoLuong	SLTS
10	Tình trạng	TTTS
11	Tổng số TS	TongTS
12	Tổng số lượng	TongSL

nên nó không phải là thuộc tính mà ta quan tâm, nhưng dữ liệu về Mã phiếu, Ngày kiểm kê, có thể khác nhau và chúng là các thuộc tính của phiếu.

Vậy ta được danh sách các thuộc tính như danh sách bên:

[2]. Loại bỏ thuộc tính suy diễn:

Thuộc tính suy diễn là thuộc tính mà giá trị của nó có thể tính toán được từ giá trị của các thuộc tính khác (xem tài liệu Hướng dẫn ôn tập môn CSDL – cùng tác giả). Trong danh sách này, thuộc tính bị loại là hai thuộc tính số 11 và 12. Vậy phiếu trên chỉ còn 10 thuộc tính.

[3]. Phân nhóm các thuộc tính:

Ta tiến hành phân nhóm các thuộc tính trong danh sách theo nguyên tắc: **các thuộc tính mô tả chung một loại đối tượng sẽ thành một nhóm**. Ví dụ: các thuộc tính TenNV và CVNV đều mô tả thuộc tính của một loại đối tượng là Nhân Viên; các thuộc tính MaP, TenP, TruongP đều mô tả các thuộc tính của một loại đối tượng là Phòng ban....Do vậy, ta được một danh sách đã phân nhóm như danh sách kế bên:





1	Mã phiếu	MaPh
2	Ngày kiểm kê	Ngay
3	Nhân viên kk	TenNV
4	Chức vụ	CVNV
5	Phòng kiểm kê	TenP
6	Mã phòng kk	MaP
7	Trưởng phòng	TruongP
8	Tên tài sản	TenTS
9	SoLuong	SLTS
10	Tình trạng	TTTS

[4]. Hình thành các lớp và vẽ sơ đồ lớp:

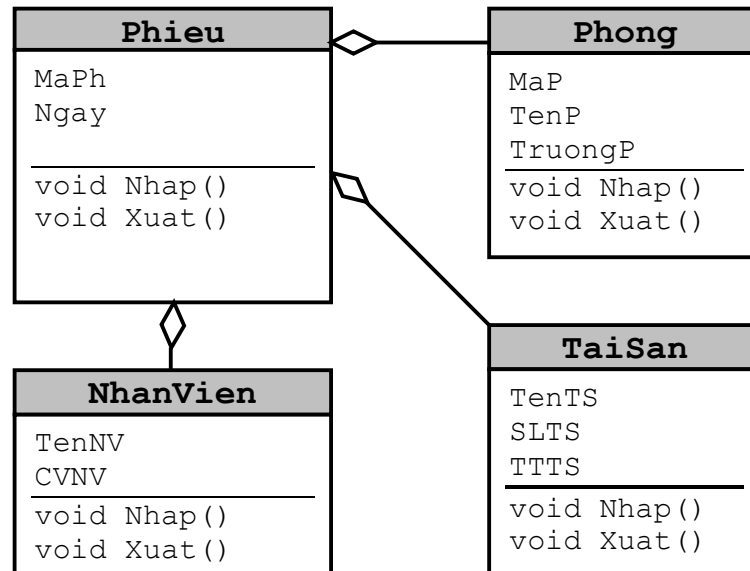
Mỗi nhóm thuộc tính, hãy đưa ra tên loại đối tượng mà nhóm thuộc tính đó mô tả. Ví dụ, với 4 nhóm trên, ta có thể đưa ra 4 danh từ: Phiếu, Nhân Viên, Phòng, Tài sản.

Cần chú ý là bao giờ cũng có 1 nhóm mô tả đối tượng Phiếu, vì vậy luôn luôn có danh từ Phiếu trong mọi bài tập dạng này.

Mỗi danh từ ta vừa xác định sẽ thành 1 lớp. Trong ví dụ minh họa này, ta có 4 lớp: Phiếu, NhanVien, Phong, TaiSan (xem hình dưới đây).

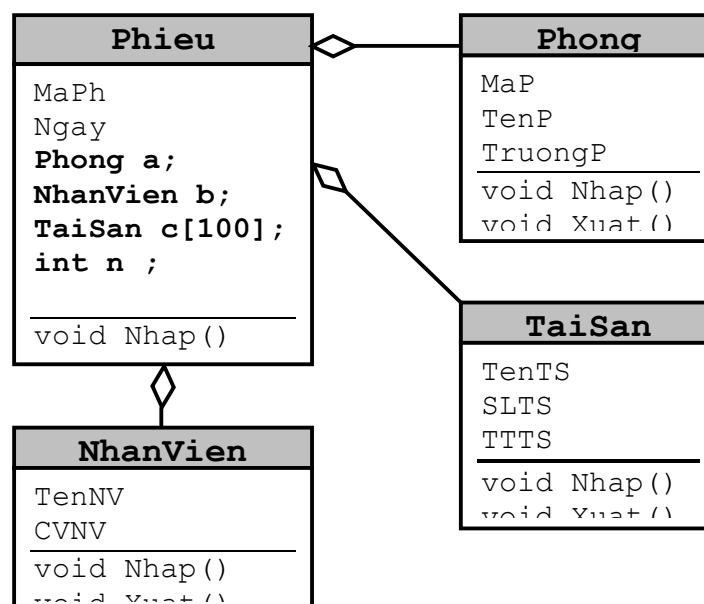
1	Mã phiếu	MaPh	 Phiếu
2	Ngày kiểm kê	Ngay	
3	Nhân viên kk	TenNV	 Nhân Viên
4	Chức vụ	CVNV	
5	Phòng kiểm kê	TenP	 Phòng
6	Mã phòng kk	MaP	
7	Trưởng phòng	TruongP	
8	Tên tài sản	TenTS	 Tài sản
9	SoLuong	SLTS	
10	Tình trạng	TTTS	

Ta tiến hành vẽ sơ đồ lớp với 4 lớp vừa xác định được. Trong mỗi lớp, hãy chuyển các thuộc tính tương ứng thành thuộc tính của lớp và thêm cho mỗi lớp 2 phương thức Nhập(), Xuất().



Riêng lớp Phiếu (có trong mọi bài dạng này) luôn là lớp chính và các lớp còn lại kết tập vào nó (xem sơ đồ trên). Vì vậy lớp này chúng ta cần phải **bổ sung thêm thuộc tính** cho nó để thể hiện sự kết tập này.

Một phiếu, theo mẫu, ngoài thông tin về Mã phiếu, ngày kiểm kê còn có chứa thông tin của một nhân viên, một phòng ban (kiểm kê) và nhiều tài sản (kiểm kê). Đó chính là 3 sự kết tập của 3 lớp **NhanVien**, **Phong**, **TaiSan** vào lớp **Phiếu**. Dễ thấy 3 sự kết tập này được chia làm 2 loại: kết tập 1-1 (**Phòng-Phiếu**; **NhanVien-Phiếu**) và kết tập 1-nhiều (**Phiếu-Tài Sản**). Hãy chú ý vào lớp **Phiếu** với các thuộc tính ta sẽ bổ sung để thể hiện 2 loại kết tập này:



Vì Tài sản kết tập vào phiếu theo kiểu 1 – nhiều, nên ta hiểu trong 1 phiếu có chứa nhiều tài sản. Việc này được thể hiện bằng cách khai báo một mảng c nhiều đối tượng thuộc lớp TaiSan (100 đối tượng – TaiSan c[100];) và thêm biến n là “số tài sản thực tế có trong mỗi phiếu”.

Việc cài đặt, sau đó, được tiến hành theo sơ đồ 4 bước ở PHẦN 1 (hình 7).

Khi cài đặt các lớp này, trong mỗi phương thức Xuat() của mỗi lớp, ta cần chú ý định dạng sao cho khi xuất dữ liệu được giống như mẫu phiếu mà đầu bài cho. Việc này được thực hiện đơn giản bằng cách chú ý đặt dấu xuống dòng (endl) phù hợp, đúng vị trí trong mỗi lệnh cout.

Sau đây là 4 bước để cài đặt theo sơ đồ hình 7:

Bước 0. Thứ tự cài đặt: Phong, NhanVien, TaiSan, Phieu. (chú ý bao giờ cũng cài đặt lớp Phieu sau cùng).

Bước 1. Cài đặt các khung lớp:

```
class Phong
{
    char MaP[30];
    char TenP[30];
    char TruongP[30];
public:
    void Nhap();
    void Xuat();
};

class NhanVien
{
    char TenNV[30];
    char CVNV[30];
public:
    void Nhap();
    void Xuat();
};

class TaiSan
{
    char TenTS[30];
    int SLTS;
    char TTTS[30];
public:
    void Nhap();
    void Xuat();
    friend class Phieu;
};

class Phieu
{
    char MaPh[30];
    char Ngay[12];
    Phong a;
    NhanVien b;
    TaiSan c[100];
    int n;
public:
```

B2. Cài đặt các phương thức ngoài lớp:

Chú ý phương thức `Phieu::Xuat()` do có thống kê tổng số lượng tài sản nên nó truy cập vào thuộc tính `SLTS` của lớp `TaiSan`, vì vậy mà lớp `Phieu` phải là bạn của lớp `TaiSan` (xem B1).

```
void Phong::Nhap()
{
    cout<<"Ma phong:"; gets(MaP); fflush(stdin);
    cout<<"Ten phong:"; gets(TenP); fflush(stdin);
    cout<<"Truong phong:"; gets(TruongP); fflush(stdin);
}
void Phong::Xuat()
{
    cout<<"Kiem ke tai phong: "<<TenP;
    cout<<"Ma phong: "<<MaP<<endl;
    cout<<"Truong phong: "<<TruongP<<endl;
}

void NhanVien::Nhap()
{
    cout<<"Ten nhan vien:"; gets(TenNV); fflush(stdin);
    cout<<"Chuc vu:"; gets(CVNV); fflush(stdin);
}
void NhanVien::Xuat()
{
    cout<<"Nhan vien kiem ke: "<<TenNV;
    cout<<"Chuc vu: "<<CVNV<<endl;
}

void TaiSan::Nhap()
{
    cout<<"Ten tai san:"; gets(TenTS); fflush(stdin);
    cout<<"So luong:"; cin>>SLTS;
    cout<<"Tinh trang:"; gets(TTTS); fflush(stdin);
}
void TaiSan::Xuat()
{
    cout<<TenTS<<"      "<<SLTS<<"      "<<TTTS<<endl;
}

void Phieu::Nhap()
{
    cout<<"Ma phieu:"; gets(MaPh); fflush(stdin);
    cout<<"Ngay lap:"; gets(Ngay); fflush(stdin);
    a. Nhap();
    b. Nhap();
    cout<<"Nhap so tai san cua phieu:"; cin>>n;
    for(int i=0; i<n; i++)
        c[i].Nhap();
}
void Phieu::Xuat()
{
    cout<<"          PHHIEU KIEM KE TAI SAN"<<endl;
    cout<<"Ma phieu: "<<MaPh<<" Ngay kiem ke:"<<Ngay<<endl;
    a. Xuat();
    b. Xuat();
    cout<<"Ten ts|   So luong | Tinh trang"<<endl;
    for(int i=0; i<n; i++)
        c[i].Xuat();
    cout<<"Tong so tai san da kiem ke:"<<n;
    int TongSL=0;
    for(i=0; i<n; i++)
        TongSL+=c[i].SLTS;
    cout<<"      Tong so luong:"<<TongSL;
}
```

B3. Cài đặt hàm truy cập thuộc tính riêng tư: không có.

B4. Cài đặt hàm main:

```
void main()
{
    Phieu x;
    x.Nhap();
    x.Xuat();
}
```

2 Phương pháp chung cài đặt bài tập hđt

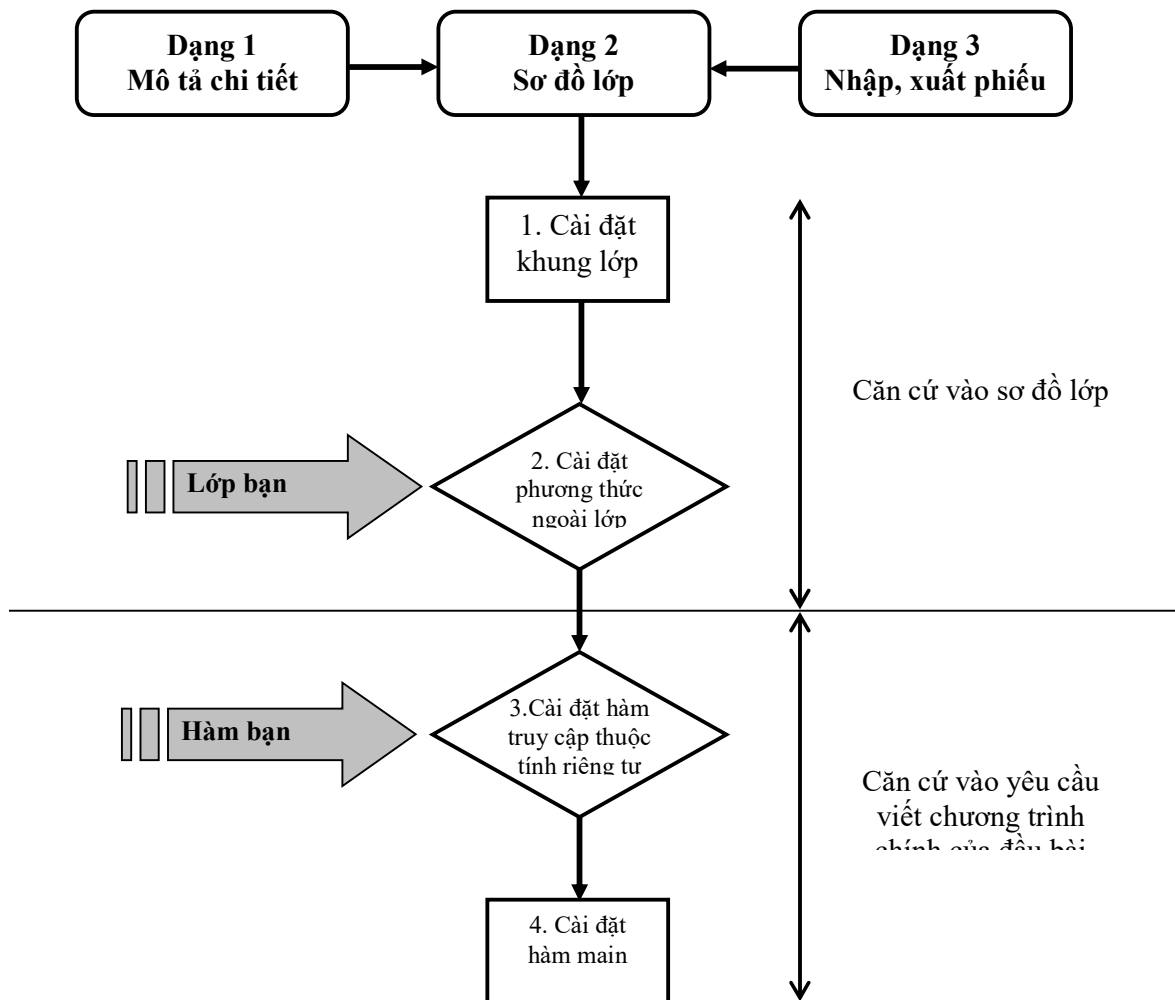
Một đề bài lập trình hướng đối tượng, thông thường được cho ở 3 dạng:

[1]. Dạng mô tả bằng lời: Toàn bộ các lớp, các mối quan hệ giữa các lớp được đề bài mô tả bằng lời một cách chi tiết. Dạng này, các bạn dễ dàng xác định được các lớp của bài và mối quan hệ giữa chúng; các thuộc tính và phương thức trong mỗi lớp. Do vậy, ta dễ dàng vẽ một sơ đồ lớp cho mỗi bài (nếu cần).

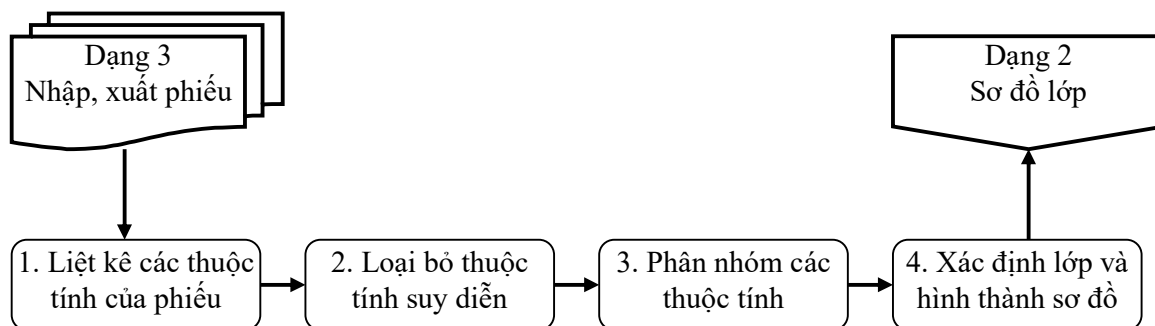
[2]. Dạng sơ đồ lớp: Đề bài yêu cầu cài đặt 1 sơ đồ lớp, trong đó đã thể hiện các lớp, mối quan hệ giữa các lớp, các thuộc tính và phương thức của từng lớp. Như vậy, cần có kiến thức tối thiểu để đọc, hiểu sơ đồ và giải nó theo phương pháp 4 bước ở hình 7 (xem phần 1). Với dạng này, luôn có phần yêu cầu ngay sau sơ đồ lớp. Phần đó, thông thường là yêu cầu các chức năng chính của bài được thực hiện sau khi đã cài đặt sơ đồ lớp.

[3]. Dạng phiếu: Với việc cài đặt các chức năng Nhập, Xuất cho một phiếu bất kỳ, ta dễ dàng chuyển chúng thành sơ đồ lớp và dạng này biến thành dạng [2]. Với dạng này, cần chú ý tới các thuộc tính suy diễn của phiếu. Thông thường, đây là các thuộc tính mang tính thống kê và ta cần tính giá trị cho các thuộc tính này bằng cách thống kê các giá trị của các thuộc tính khác (ví dụ tính tổng số lượng tài sản bằng cách duyệt qua các tài sản và cộng dồn số lượng). Điều này rất dễ bị thí sinh bỏ qua do nó thường không được quan tâm đúng mức.

Vậy một sơ đồ quan trọng trong việc làm bài tập lập trình HDT được cho dưới đây. Người đọc nên dành thời gian thích đáng để nghiên cứu và ghi nhớ sơ đồ này:



Hình 8. Sơ đồ chung cho việc cài đặt



Hình 9. Chuyển một phiếu thành sơ đồ lớp.

Một số lưu ý:

[1]. Sử dụng lại code do kết tập:

<Đối tượng>.<Phương thức>;

Trong đó, <Đối tượng> là 1 thuộc tính thể hiện sự kết tập của một lớp vào một lớp khác.

[2]. Sử dụng lại code do kế thừa:

<Tên lớp>::<Phương thức>;

Trong đó, <Tên lớp> là tên lớp cha. <Phương thức> chính là phương thức mà lớp con kế thừa được từ lớp cha đó.

[3]. Khai báo hàm bạn: đối vào của hàm bạn không được là mảng thông thường mà phải chuyển thành con trỏ. Kiểu của đối, nếu là 1 lớp thì lớp đó cần phải khai báo trước đó.

[4]. Hãy quan sát:

<Phạm vi truy cập> hoặc <Tên lớp> : [Phạm vi kế thừa]

<Tên lớp> :: <Phương thức>

<Đối tượng> . <Phương thức> hoặc <Thuộc tính>