CHƯƠNG 4: NGUYÊN LÝ KẾ THỪA

Giảng viên: Trần Thị Ngân

Email: ngantt@tlu.edu.vn

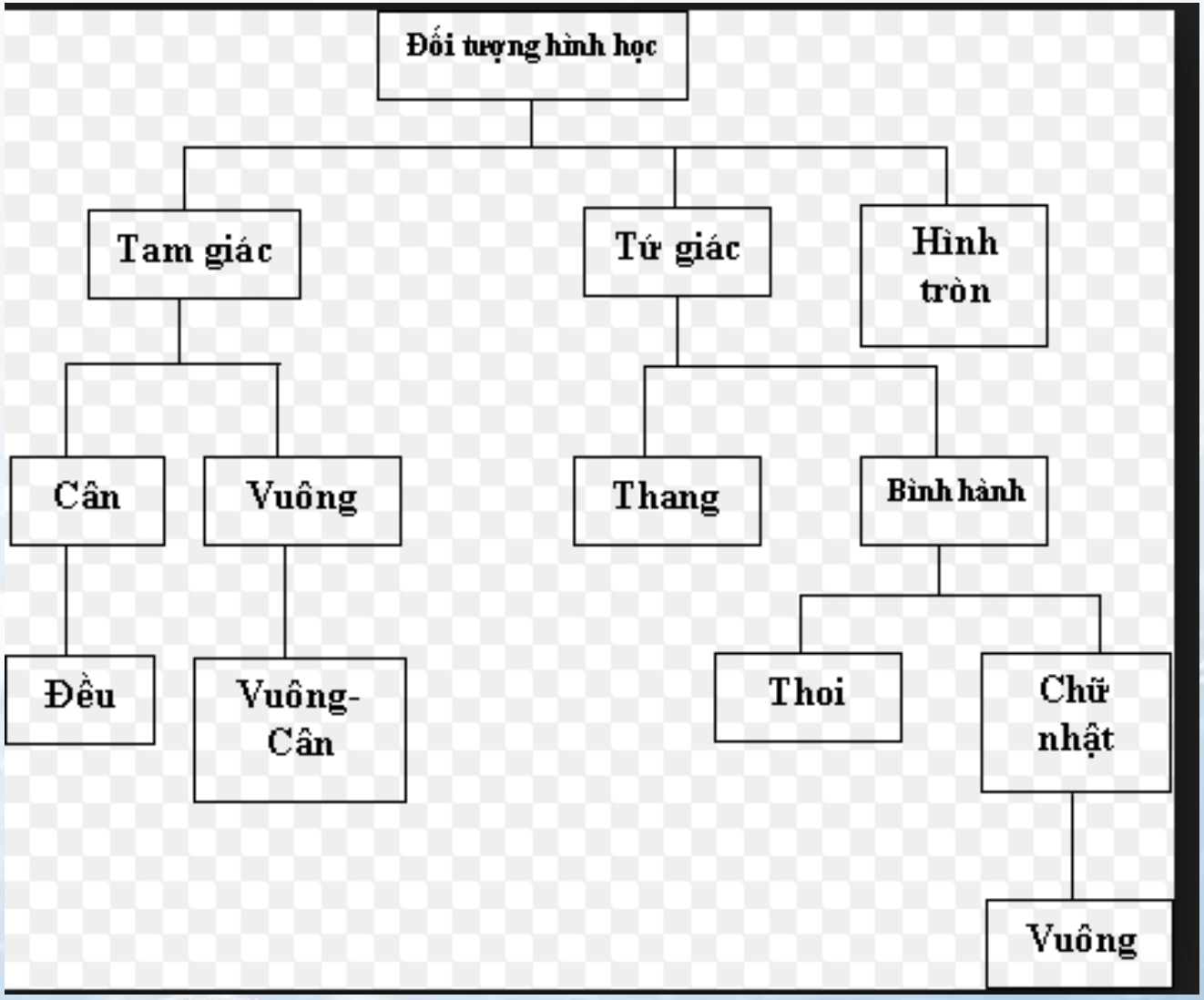
NỘI DUNG

4.1 Các kiểu kế thừa

4.2 Hàm tạo và hàm hủy trong kế thừa

4.3 Định nghĩa lại hàm thành viên

Giới thiệu


Đối tượng hình học
Tam giác
Tứ giác
Hình tròn
Cân
Vuông
Thang
Bình hành
Đều
Vuông-Cân
Thoi
Chữ nhật
Vuông

Giới thiệu

- Kế thừa là một cơ chế sử dụng lại rất mạnh, **cho phép định nghĩa lớp mới từ các lớp đã có bằng cách sử dụng lại**
- Quy ước:
  - Lớp mới ⇔ lớp dẫn xuất
  - Lớp đã có ⇔ lớp cơ sở
- Lớp dẫn xuất sẽ kế thừa các thuộc tính hay các phương thức từ các lớp cơ sở

Định nghĩa lớp dẫn xuất

- Cú pháp:

class tên-lớp-dẫn-xuất **: mode** tên-lớp-cơ-sở

{

….; *//các thành phần mới hoặc cần định nghĩa lại*

};

mode: là chế độ kế thừa (xác định các điều khiển, các hàm thành viên trong lớp cơ sở được xuất hiện như thế nào trong lớp dẫn xuất). Có thể nhận giá trị: public, private, protected.

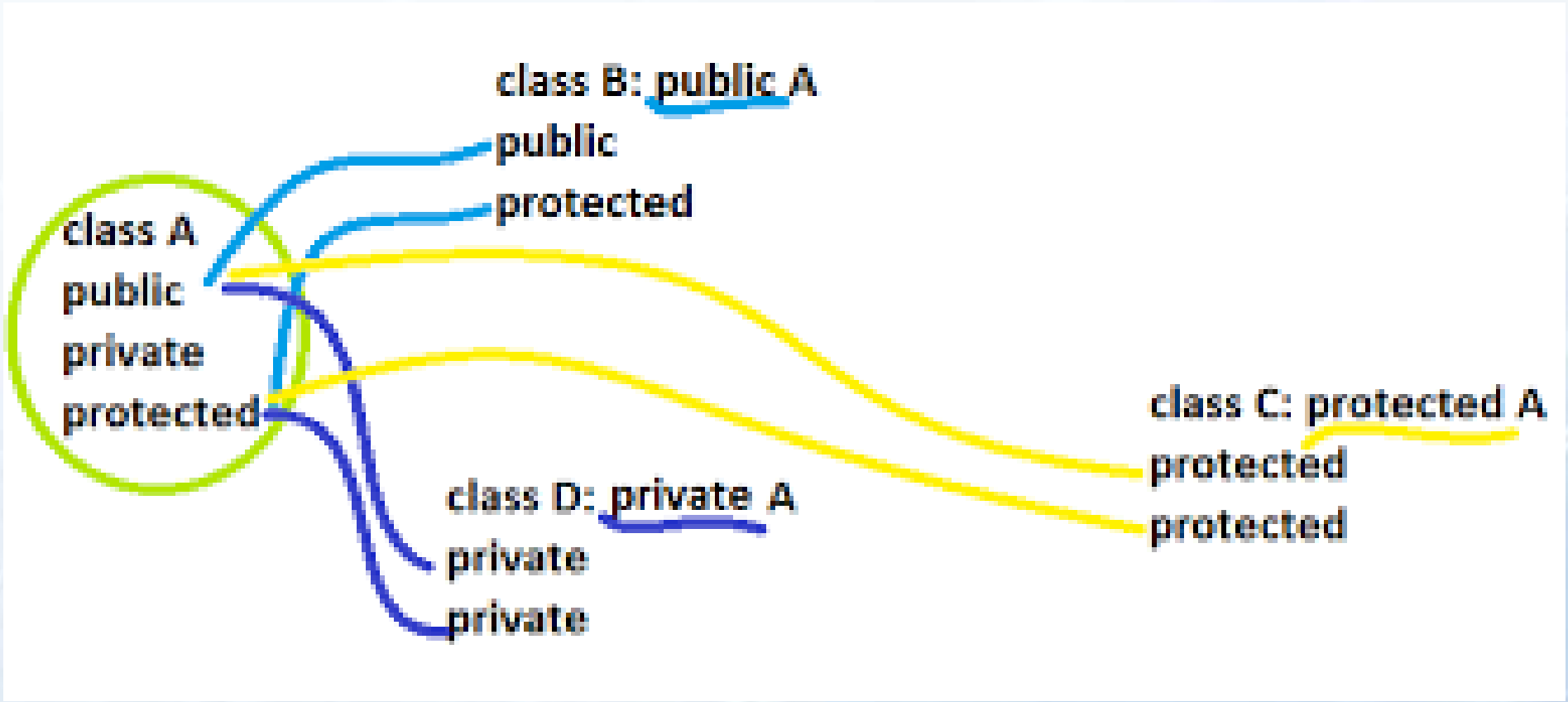
Chế độ kế thừa

- **public**: các thành phần chung (**public, protected**) của lớp cơ sở trở thành phần chung (**public, protected**) của lớp dẫn xuất ⇔ các đối tượng của lớp dẫn xuất có thể truy xuất đến thành phần public của lớp cơ sở
- **private:** tất cả các thành phần **public** hay **protected** của lớp cơ sở trở thành phần riêng **private** của lớp dẫn xuất ⇔ các thành phần public của lớp cơ sở chỉ truy cập được thông qua hàm thành phần của lớp dẫn xuất
- **protected**: tất cả các thành phần **public** và **protected** của lớp cơ sở thành thành phần **protected** trong lớp dẫn xuất

Tổng hợp cơ chế kế thừa

Lớp cơ sở	Thừa kế public	Thừa kế private	Thừa kế protected
	Kết quả trong lớp dẫn xuất		
private	–	–	–
public	public	private	protected
protected	protected	private	protected

Tổng hợp cơ chế kế thừa



Lớp thừa kế (derived classes)

- Xét lớp *Người (Nguoi).* Có thể bao gồm nhiều loại nhỏ:

- Học sinh ***(****Hocsinh****)***

- Công nhân ***(****Congnhan****)…***

- Khái niệm tổng quát về *Người* là hữu ích. Lớp này được định nghĩa trước như một khung chung:

- Tất cả mọi Người đều có những thông tin chung: Tên, năm sinh, sở thích,..

- Các hàm thành viên liên quan đến những dữ liệu này là giống nhau (cơ sở) cho tất cả mọi người.

Ví dụ: hàm tạo, hàm hiển thị,…

Ví dụ - Xây dựng lớp người - Person

```
class Nguoi
{
  protected:
    string hoTen;
    int namSinh;
  private:
   string soThich;
  public:
    Person(){
      hoTen ="Nguyen Van Hoa";
      namSinh=1999;
      soThich="Lap trinh";
    }
void HienThi();
};
```

Giao diện lớp Hocsinh

- Khai báo lớp kế thừa

  class Hocsinh : public Nguoi

- Giao diện (interface) của lớp thừa kế chỉ liệt kê những thành viên mới hoặc sẽ được định nghĩa lại (Bởi vì tất cả những thành viên khác kế thừa từ lớp cơ sở đã được định nghĩa trước đó!)
- Lớp Hocsinh thêm các thành viên sau:

  Biến thành viên: *maHS;*

  Hàm thành viên: *void Nhap(); void HienThi();*

Ví dụ

```
class Nguoi
{
  protected:
    string hoTen;
    int namSinh;
  private:
   string soThich;
  public:
    Person(){
      hoTen ="Nguyen Van Hoa";
      namSinh=1999;
      soThich="Lap trinh";
    }
void HienThi();
};
```

```
class HocSinh : public Nguoi
{
  protected:
    int maHS;
  public:
    void Nhap();
    void HienThi();
};
```

Định nghĩa lại hàm thành viên

Hàm *HienThi()* của lớp cơ sở Nguoi

- Được định nghĩa lại trong các lớp thừa kế
- Do các nhóm người khác nhau có thể có các hiển thị khác nhau.

Định nghĩa lại hàm thành viên

- Lớp Hocsinh định nghĩa lại hàm thành viên *Hienthi()* của lớp cơ sở
- Phiên bản mới của hàm *Hienthi()* sẽ "ghi đè" (overrides) phiên bản cũ đã được cài đặt trong lớp cơ sở Person
- Cài đặt của hàm thành viên này phải được thực hiện trong lớp Hocsinh

Định nghĩa lại hàm thành viên

- Định nghĩa lại hàm VS nạp chồng hàm

Rất khác nhau

Định nghĩa lại:

- Định nghĩa lại hàm trong lớp thừa kế
- CÙNG danh sách tham số
- Thực chất là viết lại cùng một hàm

Nạp chồng hàm

- Danh sách tham số khác nhau
- Định nghĩa một hàm mới với tham số khác

Định nghĩa lại hàm thành viên

```
Void Nguoi::HienThi()//Hàm HienThi() của lớp cơ sở
{
 cout << "\n Nam sinh : " << namSinh;
 cout << "\n Ho ten : " << hoTen;
 cout << "\n So thich : " << soThich;
}
void HocSinh::HienThi()
//Hàm HienThi() được định nghĩa lại trong lớp Hocsinh
{
  cout << "\n Nam sinh : " << namSinh;
  cout << "\n Ho ten : " << hoTen;
  cout << "\n Ma hoc sinh : " << maHS;
}
```

Truy xuất hàm định nghĩa lại

- Khi được định nghĩa lại một hàm trong lớp con, định nghĩa của hàm này trong lớp cơ sở không bị mất đi!

```
Nguoi N1;
Hocsinh HS1;
HS1.Nhap();
 N1.Hienthi();
    //gọi hàm Hienthi() của lớp Nguoi
HS1.Hienthi();
    //gọi hàm Hienthi() của lớp Hocsinh
HS1. Nguoi ::Hienthi();
   //gọi hàm Hienthi() của lớp nào?
```

VÍ DỤ

Xây dựng lớp **HoaDon** bao gồm:

Biến thành viên:

soLuong có kiểu **int**: Số lượng sản phẩm

giaCa có kiểu **double**: Giá sản phẩm

Hàm thành viên:

- ✓ Một hàm tạo không tham số để khởi tạo một hóa đơn gồm một sản phẩm, giá cả của sản phẩm là 10.000 vnđ
- ✓ Một hàm tạo hai tham số **HoaDon(int sl, double gc)*:*** Thiết lập giá trị cho số lượng và giá cả tương ứng
- ✓ Hàm **tongTien():** Tổng tiền cho sản phẩm

VÍ DỤ

Lớp **HoaDonKM** kế thừa từ lớp **HoaDon**. Lớp này có thêm các biến và hàm thành viên sau

Biến thành viên:

giamGia có kiểu **double**: Số tiền được khuyến mại

Hàm thành viên:

- ✓ Hàm tạo không đối số **HoaDonKM():** Khởi tạo hóa đơn gồm 1 sản phẩm, giá sản phẩm là 10.000 vnđ, giảm giá 1000 vnđ
- ✓ Một hàm tạo ba tham số **HoaDonKM(int sl, double gc, double gg):** Thiết lập giá trị cho các biến thành viên (số lượng, giá cả và tiền giảm giá tương ứng)
- ✓ Hàm **thanhToan()*:*** Tính số tiền khách hàng cần thanh toán, sử dụng hàm **tongTien** thừa kế từ lớp **HoaDon**)

Ví dụ - Định nghĩa lớp HoaDon

```
#include <iostream>
using namespace std;
class HoaDon{
    protected:
        int soLuong; double giaCa;
    public:
        HoaDon()
        {
            soLuong=1;
            giaCa=10;
        }
        HoaDon(int sl, double gc)
        {
            soLuong=sl;
            giaCa=gc;
        }
        double tongTien();
};
```

Ví dụ - Định nghĩa lớp HoaDonKM

```
class HoaDonKM : HoaDon {
private:
double giamGia;
public:
    HoaDonKM()
    {
        giamGia=1;
    }
    HoaDonKM(int sl, double gc, double gg):HoaDon(sl, gc)
    {
        giamGia=gg;
    }
    double thanhToan();
};
```

Ví dụ - Các hàm thành viên

```
double HoaDon::tongTien()
{
    return soLuong*giaCa;
}

double HoaDonKM::thanhToan(){
    return tongTien()-giamGia;
}
```

Ví dụ - Hàm main()

```
double HoaDonKM::thanhToan(){
    return tongTien()-giamGia;
}
int main()
{
    HoaDon d1(5,3);
    cout<<"Tong tien cua hoa don la: "<<d1.tongTien()<<" nghin dong";
    cout<<endl;
    HoaDonKM km;
    cout<<"Tong tien cua hoa don km la: "<<km.thanhToan()<<" nghin dong";
    HoaDonKM km1(3,5,3);
    cout<<endl;
    cout<<"Tong tien cua hoa don km1 la: "<<km1.thanhToan()<<" nghin dong";
    return 0;
}
```

Ví dụ áp dụng

SINHVIEN
Dữ liệu: - Ma sinh vien - Họ tên - Tuổi Thao tác: -Nhập -Xuất

DIEM TONG KET
Dữ liệu: -Mon Toan -Mon Ly -Mon Hoa Phương thức - Nhập - In -Tinh diem trung binh -Xep loai

Áp dụng:

-Tạo một danh sách điểm tổng kết để quản lý điểm của Sinh viên

-In ra danh sách các sinh viên có điểm trung bình lớn hơn 8

-Đếm số sinh viên xếp loại khá

Bài tập áp dụng

NHÂN SỰ

Dữ liệu:
- Mã nhân viên
- Họ tên
-Tuổi

Thao tác:
-Nhập
-Xuất
-Kiểm tra về hưu

CÔNG NHÂN

Dữ liệu:
- Mức lương
- Ngày công
- Tay nghề

Phương thức
- Nhập
- In
- Tính lương

CÁN BỘ

Dữ liệu:
- Hệ số lương
- Phụ cấp
- Chuyên môn
- Trình độ

Phương thức
- Nhập
- In
- Tính lương

Áp dụng:
- Tạo ra danh sách để quản lý công nhân và quản lý cán bộ
- In ra màn hình danh sách các công nhân đã đủ điều kiện về hưu
- In ra màn hình danh sách các cán bộ chưa đủ điều kiện về hưu
- In ra tiền lương cao nhất trong công nhân và tiền lương thấp nhất trong cán bộ

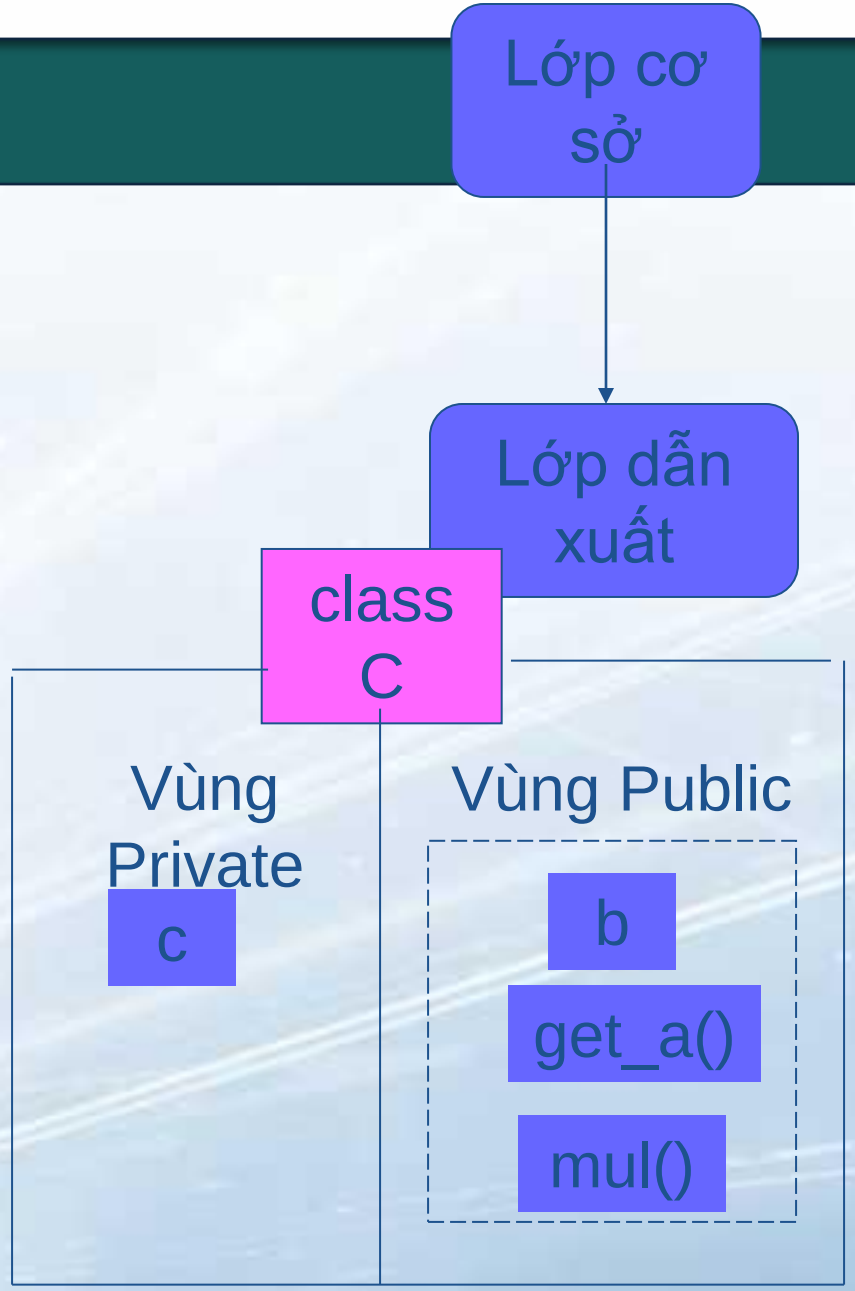
II. Phân loại kế thừa

1. Kế thừa đơn

- Là một lớp chỉ kế thừa từ một lớp đã có

•Ví dụ:

```
class B{
        int a;
public:
        int b;
        int get_a( );
};
class C : public B {
   int c;
public:
   void mul( void ) { c=b*get_a( ); };
};
```



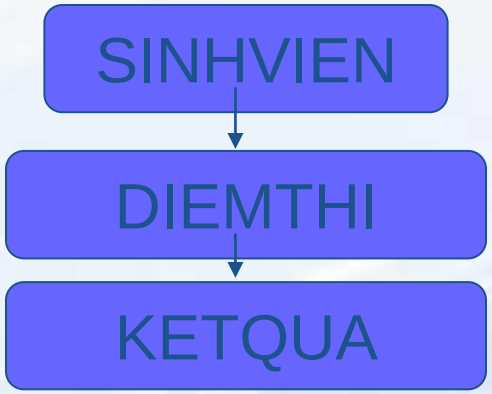
2. Kế thừa đa mức

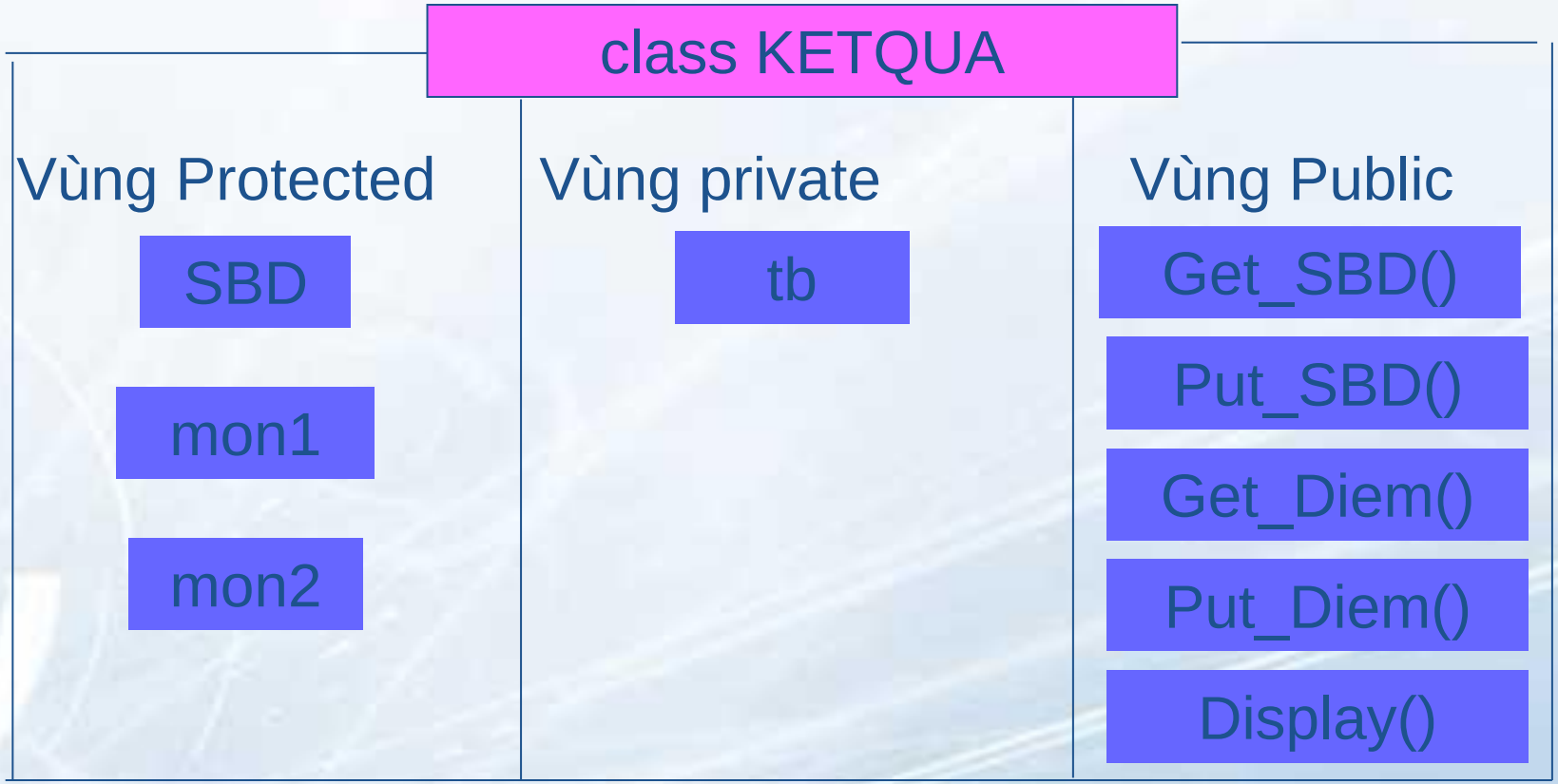
•Kế thừa đa mức là một lớp dẫn xuất kế thừa một lớp cơ sở nhưng thông qua lớp trung gian ở giữa

•Ví dụ:

```
class SINHVIEN {
        char *hoten;
protected:
        int SBD;
public:
        void get_SBD( int );
        void put_SBD( void );
 };
```

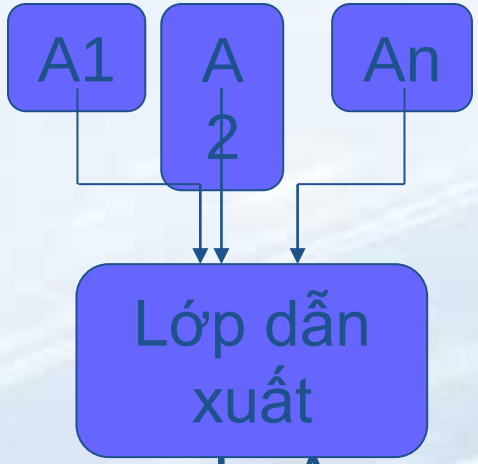
```
class DIEMTHI : public SINHVIEN {
protected:
        float mon1, mon2;
public:
        void get_Diem( float , float );
        void put_Diem( void );
};
class KETQUA : public DIEMTHI {
        float tb;
public:
        void Display( void );
};
```




class KETQUA
Vùng Protected
SBD
mon1
mon2
Vùng private
tb
Vùng Public
Get_SBD()
Put_SBD()
Get_Diem()
Put_Diem()
Display()

3. Kế thừa bội

- Là một lớp dẫn xuất kế thừa của nhiều lớp cơ sở cùng lúc ⇔ kết hợp đặc trưng của các lớp để tạo lớp mới

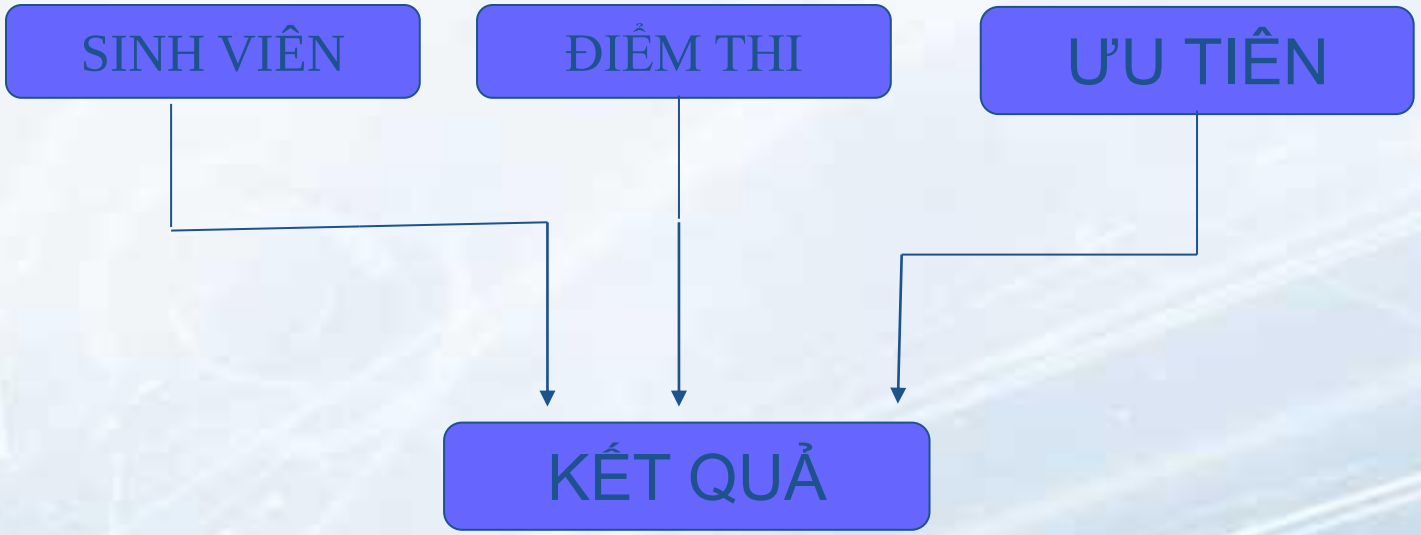


•Cú pháp:

```
class lớp-dẫn-xuất : mode A1, mode A2, …, mode An
{
        …..;   //định nghĩa lớp mới
};
```

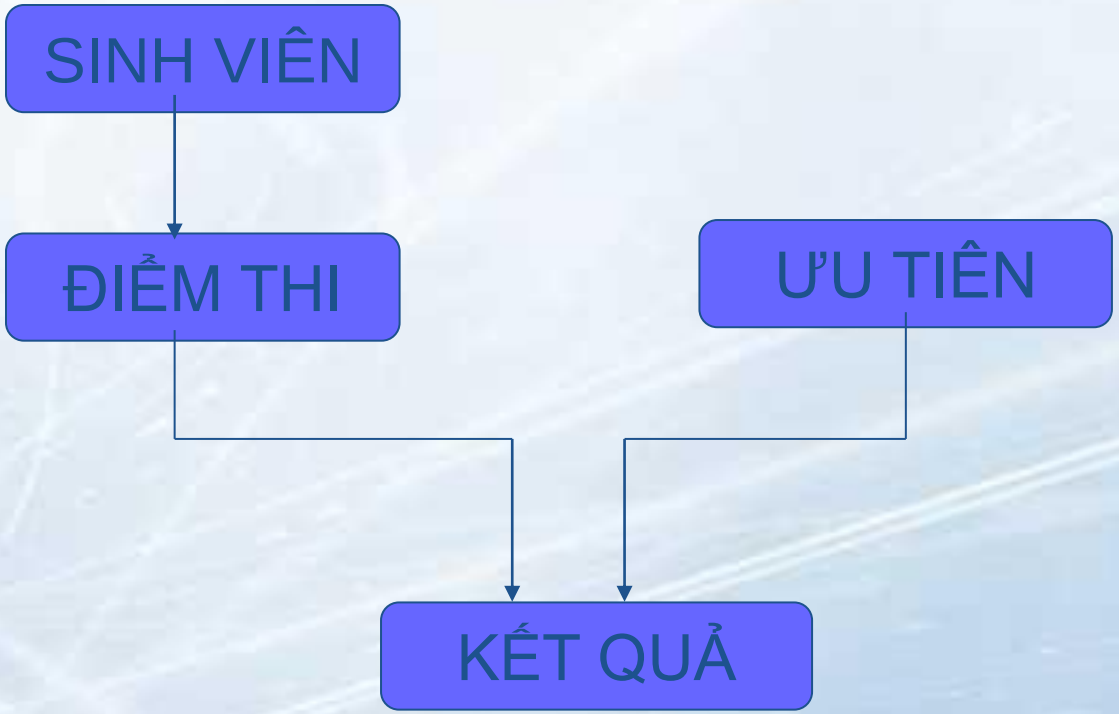
Chú ý: *mỗi lớp sẽ có một chế độ kế thừa khác nhau*

- Ví dụ:



4. Kế thừa kép:

- Là kế thừa bằng kết hợp nhiều loại kế thừa để tạo ra lớp dẫn xuất mới



IV. Hàm tạo và hàm hủy trong lớp dẫn xuất

- Lớp dẫn xuất kế thừa các thành viên của lớp cơ sở **ngoại trừ hàm tạo và hàm hủy**
- Các hàm tạo và hàm hủy của lớp dẫn xuất vẫn có thể gọi các **hàm tạo và hàm hủy** của lớp cơ sở để khởi gán

- **Khi xây dựng hàm tạo của lớp dẫn xuất thì phải thực hiện khởi gán các thành phần của lớp cơ sở trước (gọi hàm tạo của lớp cơ sở trước), sau đó mới thực hiện khởi gán các thành phần trong lớp dẫn xuất**
- Các hàm hủy được gọi theo thứ tự ngược lại => các hàm hủy của lớp dẫn xuất thực hiện trước, sau đó đến các hàm hủy của lớp cơ sở

Hàm tạo trong lớp thừa kế

- Hàm tạo của lớp cơ sở không được kế thừa trong lớp con!
- Chúng được gọi bên trong hàm tạo của lớp con!
- Hàm tạo của lớp cơ sở nên khởi tạo tất cả các biến thành viên

Hàm tạo trong lớp thừa kế

- Gọi hàm tạo của lớp cha trong hàm tạo của lớp con
- Không thể gọi hàm tạo của lớp cha trong hàm tạo của lớp con như hàm, mà phải gọi ở danh sách khởi tạo.
- Nếu lớp con không gọi hàm tạo nào của lớp cơ sở: Hàm tạo mặc định của lớp cơ sở tự động được gọi.

Ví dụ: hàm tạo trong lớp kế thừa

Đoạn chương trình sau sẽ in kết quả nào ra màn hình

```
xVal = 6
xVal = 8
--------------------------------
Process exited after 0.1103 seconds with return value 0
Press any key to continue . . .
```

```
#include<iostream>
using namespace std;
    class Base{
    public:
    int xVal;
    Base(int x=6) : xVal(x){
    cout<<"xVal = "<<xVal<<endl;
        }
};
class Derived:Base
{
public:  Derived(){xVal = 8;}
                void Print(){
    cout<<"xVal = "<<xVal;  }
};
int main(){
    Derived d;
    d.Print();
}
```

* Một số nguyên tắc

- Nếu trong lớp cơ sở không có hàm tạo có tham số thì trong lớp dẫn xuất không bắt buộc xây dựng hàm tạo trong lớp dẫn xuất
- Nếu trong lớp cơ sở chỉ có hàm tạo có tham số thì trong lớp dẫn xuất bắt buộc phải xây dựng hàm tạo trong lớp dẫn xuất
- Cả lớp cơ sở và lớp dẫn xuất đều có tham biến => hàm tạo lớp cơ sở được thực hiện sau đó đến hàm tạo lớp dẫn xuất
- Kế thừa bội thì các hàm tạo được thực hiện lần lượt theo thứ tự khai báo

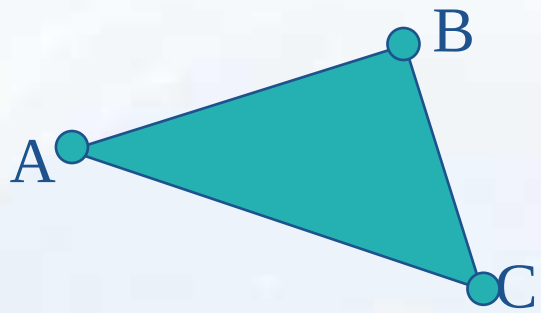
* Ví dụ

- Xây dựng lớp POINT với hai thành phần dữ liệu x,y. *Yêu cầu xây dựng hàm hàm tạo và hàm hủy*
- Xây dựng lớp TAMGIAC kế thừa từ lớp POINT với thành phần thể hiện tọa độ của tam giác. *Yêu cầu xây dựng hàm tạo và hàm hủy tương ứng cho hình tròn*

```
class DIEM
{
    protected:
            int x;
            int y;
    public:
            DIEM( int xx, int yy);
            void nhap( );
            void in( );
};

DIEM::DIEM( int xx, int yy)
{
     x = xx;
     y = yy;
}
```

```
class HINHTG : public DIEM
{
private:
    DIEM B;                         //Tọa độ điểm B
    int Cx, Cy;                     // Tọa độ điểm C
public:
    TAMGIAC( int Axx, int Ayy, int Bxx, int Byy, int Cxx, int Cyy );
    void nhap( );
    void in( );
};
```




```
TAMGIAC::TAMGIAC( int Axx, int Ayy, int Bxx, int Byy, int Cxx,
   int Cyy ) : DIEM(Axx, Ayy) , B(Bxx, Byy)
{
    Cx = Cxx;
    Cy = Cyy;
}
```

*Chú ý:

- Khi thực hiện kế thừa các thành phần dữ liệu của lớp dẫn xuất gồm các loại:
  - thành phần mới khai báo (*Cx, Cy)*
  - thành phần kế thừa từ lớp cơ sở *(Tọa độ đỉnh A – Ax, Ay)*
  - thành phần có kiểu là đối tượng của lớp đã định nghĩa *(đỉnh B – Bx, By)*

*Chú ý:

- Tùy theo loại thành phần dữ liệu khác nhau, cách xây dựng hàm tạo là khác nhau:
  - Nếu thành phần mới: dùng câu lệnh gán trực tiếp trong thân hàm hàm tạo
  - Nếu thành phần kế thừa từ lớp cơ sở: dùng hàm hàm tạo của lớp cơ sở để khởi gán và viết trên dòng tên hàm hàm tạo của lớp dẫn xuất ⇔ **DIEM(Axx, Ayy)**
  - Nếu thành phần có kiểu là đối tượng của lớp đã có: dùng trực tiếp tên của đối tượng được khai báo để khởi gán và cùng viết trên dòng tên hàm hàm tạo ⇔ **B(Bxx, Byy);**

V. Toán tử gán cho lớp dẫn xuất

- Khi trong lớp dẫn xuất có tồn tại thuộc tính dưới dạng con trỏ (kể cả thuộc tính kế thừa từ lớp cơ sở) thì bắt buộc phải xây dựng toán tử gán không được sử dụng toán tử gán mặc định.

*Ví dụ định nghĩa toán tử gán của lớp dẫn xuất:

```
class A
{ ….
    A operator =(A & h)
    {
            //thực hiện định nghĩa toán tử gán của lớp A
    }
    A* get_A() { return this; }
};
class B:public A
{ …..
   B& operator=(B&h)
   {
      A *u1, *u2;
       u1= this->get_A();
      u2 = h.get_A();
     *u1= *u2;
    }
};
```

Cách xây dựng

Bước 1: xây dựng toán tử gán cho lớp cơ sở.

Bước 2: Xây dựng phương thức (trong các lớp cơ sở) để nhận địa chỉ của đối tượng ẩn của lớp:

```
Tên_lớp_cơ_sở* get_DT()
{
        return this;
}
```

Bước 3: xây dựng toán tử gán cho lớp dẫn xuất ⇔ dùng phương thức trên để nhận địa chỉ của đối tượng lớp cơ sở mà lớp dẫn xuất kế thừa, sau đó thực hiện lệnh gán trên 2 đối tượng này

* Ví dụ:

```
class DIEM
{
    private:
            int x;
            int y;
    public:
            DIEM operator = ( DIEM &h )
            {
                    x = h.x;
                    y = h.y;
            }
          DIEM *get_DIEM( )
          {
             return this;
          }
};
```

```
class HCN : public DIEM
{
    private:
       int Cx, Cy;
    public:
    HCN operator = ( HCN &h)
    {
        DIEM *u1, *u2;
        u1 = this -> get_DIEM( );
        u2 = h.get_DIEM( );
        *u1 = *u2;
        Cx = h.Cx;
        Cy = h.Cy;
    }
};
```

VI. Hàm tạo sao chép của lớp dẫn xuất

- Tương tự như toán tử gán, khi các thuộc tính (kể cả các thuộc tính kế thừa) là con trỏ cần xây dựng hàm tạo sao chép để tạo ra đối tượng mới giống và độc lập với đối tượng đã cho.
- Cách xây dựng:

B1: Xây dựng toán tử gán cho lớp dẫn xuất

B2: Xây dựng hàm tạo sao chép cho lớp dẫn xuất theo mẫu:

```
Tên_lớp_dẫn_xuất(Tên_lớp_dẫn_xuất & h)
{
        *this = h;
}
```

VII. Sự nhập nhằng trong đa kế thừa

- Trong đa kế thừa xảy ra trường hợp hai lớp cơ sở của một lớp dẫn xuất có cùng tên thành phần (dữ liệu và phương thức) => Trình biên dịch sẽ không có khả năng hiểu thành phần được sử dụng khi lớp dẫn xuất thực hiện lời gọi ⇔ **dẫn đến sự nhập nhằng**
- Để tránh nhập nhằng và để cho máy hiểu được thì phải chỉ rõ thành phần được truy cập (sử dung) thuộc lớp cơ sở nào?

VII. Sự nhập nhằng trong đa kế thừa

```
class OptionList {
    public:
        // ……
        void Highlight (int part);
};
```

```
class Window {
    public:
        // ……
        void Highlight (int part);
};
```

```
class Menu : public OptionList,
             public Window
    { ……. };
```

Hàm cùng tên

Chỉ rõ hàm của lớp nào

Gọi hàm của lớp nào ?

```
void main() {
    Menu  m1(….);
    m1.Highlight(10);
    ….
}
```

xử lý

```
void main() {
    Menu  m1(….);
    m1.OptionList::Highlight(10);
    m1.Window::Highlight(20);
    ….
}
```