

# PHẦN 2: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OBJECT-ORIENTED PROGRAMMING)

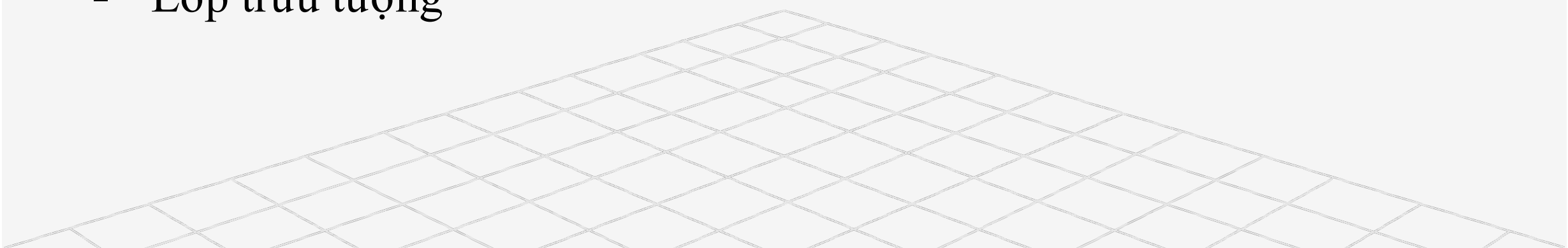
# NỘI DUNG

---

- Lịch sử các phương pháp lập trình
- Khai báo và sử dụng lớp
- Hàm tạo, Hàm hủy
- Quan hệ bạn bè: hàm bạn, lớp bạn
- Định nghĩa toán tử trong lớp
- **Kế thừa và đa hình**
- Khuôn mẫu (template)

# NỘI DUNG

---

- Kế thừa là gì?
  - Đơn kế thừa
  - Đa kế thừa
  - Đa hình
  - Phương thức ảo
  - Lớp trừu tượng
- 
- A decorative grid pattern of thin gray lines forming a perspective view of a floor or ceiling, located at the bottom of the slide.

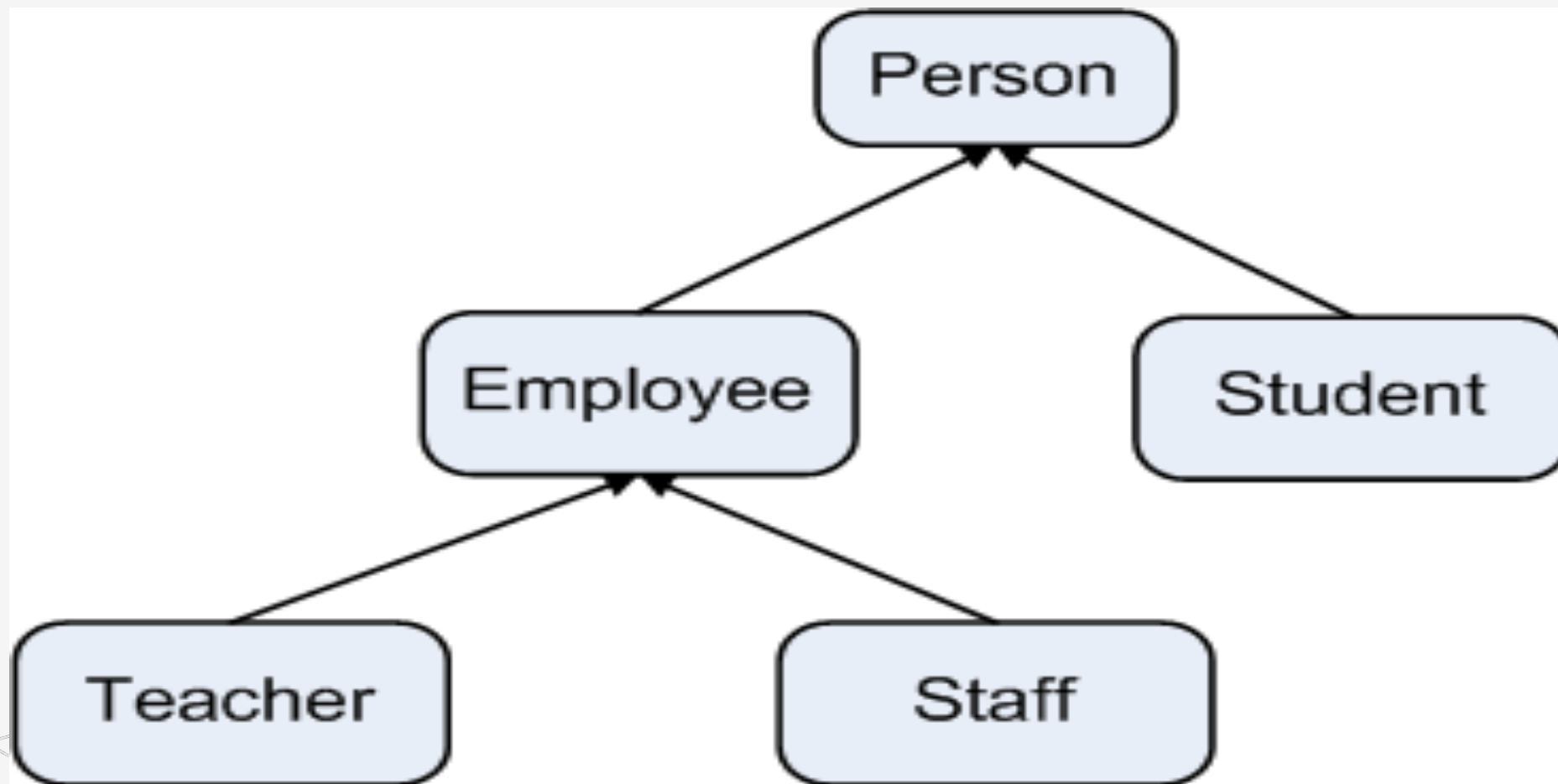
# KẾ THỪA LÀ GÌ

---

- Cho phép xây dựng các lớp mới từ một lớp đã có
  - Lớp đã có gọi là **Lớp cơ sở (base class)**
  - Lớp kế thừa gọi là **Lớp dẫn xuất (derived class)**
- Một lớp dẫn xuất có thể kế thừa từ nhiều lớp cơ sở, một lớp cơ sở có thể được kế thừa cho nhiều lớp
- Một lớp kế thừa từ 1 lớp cơ sở gọi là đơn kế thừa

# VÍ DỤ MINH HỌA

---



# CÁCH KHAI BÁO KẾ THỪA

- Định nghĩa lớp cơ sở
- Định nghĩa lớp dẫn xuất
- Ví dụ:

```
class A {  
    // thành phần dữ liệu của A  
    // hàm thành phần của A  
  
};
```

Đối tượng của lớp B mang đầy đủ các thành phần dữ liệu và hàm thành phần của cả lớp B và lớp A; không có chiều ngược lại

Kiểu kế thừa: public, protected, private

```
class B: public A {  
    // thành phần dữ liệu trong B  
    // hàm thành phần của trong B  
  
};
```

## VÍ DỤ 8-1

```
1 //vd8-1.cpp
2 #include <iostream>
3 using namespace std;
4 // khai bao lop point
5 class point {
6     public:
7     int x, y;
8     public:
9     point() { x = y = 0;}
10    point(int, int);
11    void pointdisplay();
12
13 };
14 point::point(int x, int y) {
15     this->x = x;
16     this->y = y;
17 }
```

```
18 void point::pointdisplay() {
19     cout << "x = " << x << endl;
20     cout << "y = " << y << endl;
21 }
22 // khai bao lop circle ke thua lop point
23 class circle : public point {
24     int r; // lop circle co them 1 thanh phan du lieu
25     public:
26     circle() {x = y = r = 0;}
27     circle(int, int, int);
28     void circledisplay();
29 };
30 circle::circle(int x, int y, int r) {
31     this->x = x;
32     this->y = y;
33     this->r = r;
34 }
```

## VÍ DỤ 8-1

---

```
35 void circle::circledisplay() {  
36     // gọi hàm thành phần của lớp cơ sở  
37     pointdisplay();  
38     cout << "r = " << r << endl;  
39 }  
40  
41 // hàm main  
42 int main() {  
43     circle A(10, 20, 5);  
44     A.pointdisplay(); // gọi hàm pointdisplay của lớp point  
45     A.circledisplay(); // gọi hàm circledisplay của lớp circle  
46     return 0;  
47 }  
48
```



# Kiểu Kế Thừa: public, protected, private

---

| Trong lớp cơ sở | Kế thừa public  | Kế thừa protected   | Kế thừa private   |
|-----------------|---|---|---|
| public          | <b>public</b> trong lớp dẫn xuất; có thể truy xuất bởi các hàm thành viên, hàm bạn, hàm tự do | <b>protected</b> trong lớp dẫn xuất, có thể truy xuất bởi các hàm thành viên và hàm bạn | <b>private</b> trong lớp dẫn xuất, có thể truy xuất bởi các hàm thành viên và hàm bạn |
| protected       | <b>protected</b> trong lớp dẫn xuất; có thể truy xuất bởi các hàm thành viên và hàm bạn       | <b>protected</b> trong lớp dẫn xuất; có thể truy xuất bởi các hàm thành viên và hàm bạn | <b>private</b> trong lớp dẫn xuất; có thể truy xuất bởi các hàm thành viên và hàm bạn |
| private         | Không gọi được từ lớp dẫn xuất  | Không gọi được từ lớp dẫn xuất  | Không gọi được từ lớp dẫn xuất  |

# HÀM TẠO, HÀM HỦY TRONG LỚP KẾ THỪA

---

- Các hàm tạo, hàm hủy không được kế thừa trong lớp dẫn xuất (trừ hàm cấu tử không tham số)
- Tuy nhiên, các hàm tạo của lớp dẫn xuất có thể gọi các hàm tạo của lớp cơ sở
- Hoặc trong hàm tạo của lớp dẫn xuất đặt lời gọi tường minh đến hàm tạo của lớp cơ sở để khởi tạo các thành phần thuộc lớp cơ sở trước
- Các hàm hủy được gọi theo chiều ngược lại, nghĩa là gọi hàm hủy của lớp dẫn xuất, rồi mới đến lớp cơ sở

# QUAN HỆ GIỮA CÁC ĐỐI TƯỢNG CỦA LỚP CƠ SỞ VÀ LỚP DẪN XUẤT

- Một đối tượng của lớp dẫn xuất có thể được gán cho một đối tượng của lớp cơ sở; nhưng không có chiều ngược lại
- Ví dụ: `point A; circle B; thì: A = B; //ok; nhưng B = A; // error`
- Tuy nhiên, có thể sử dụng phép ép kiểu để biến một con trỏ lớp cơ sở thành một con trỏ lớp dẫn xuất

Minh họa  
ép kiểu  
con trỏ đối  
tượng

```
point A(1,2);  
circle B(10, 20, 5);  
point *pA = &A;  
circle *pB = &B;  
pB = (circle*)pA; // ep kieu con tro  
pA->pointdisplay();  
pB->circledisplay();
```

## VÍ DỤ 8-2: XÂY DỰNG LỚP PERSON VÀ LỚP STUDENT

Person

- Dữ liệu: name, age
- Các hàm tạo
- Hàm nhập dữ liệu
- Hàm xuất dữ liệu

Student

- Dữ liệu: major, schoolname
- Các hàm tạo
- Hàm nhập dữ liệu
- Hàm xuất dữ liệu
- Hàm trả về những đối tượng có schoolname = "HNUE"

# ĐỊNH NGHĨA LẠI MỘT HÀM CỦA LỚP CƠ SỞ TRONG LỚP DẪN XUẤT (OVERRIDING FUNCTION)

---

- Có thể định nghĩa lại một hàm của lớp cơ sở trong lớp dẫn xuất (trùng tên, trùng tham số và kiểu dữ liệu của chúng)
- Khi đó đối tượng của lớp dẫn xuất sẽ ưu tiên gọi hàm của lớp dẫn xuất (gọi là overriding – ghi đè)
- Muốn gọi hàm của lớp cơ sở thì sử dụng toán tử phạm vi: `::`;
- Ví dụ:
  - Cả point và circle đều có hàm void display();
  - Khi đó: circle A;
  - `A.display();` // sẽ gọi hàm display() của circle
  - `A.point::display();` //sẽ gọi hàm display() của point

# ĐA THỪA KẾ

---

- Một lớp dẫn xuất kế thừa nhiều lớp cơ sở => đa thừa kế
- Đa thừa kế cho phép sử dụng lại nhiều thành phần của phần mềm
- Nhưng dễ nảy sinh các **nhập nhằng** và phát sinh lỗi
- Ví dụ

```
class A {  
    .....  
    .....  
};
```

```
class B {  
    .....  
    .....  
};
```

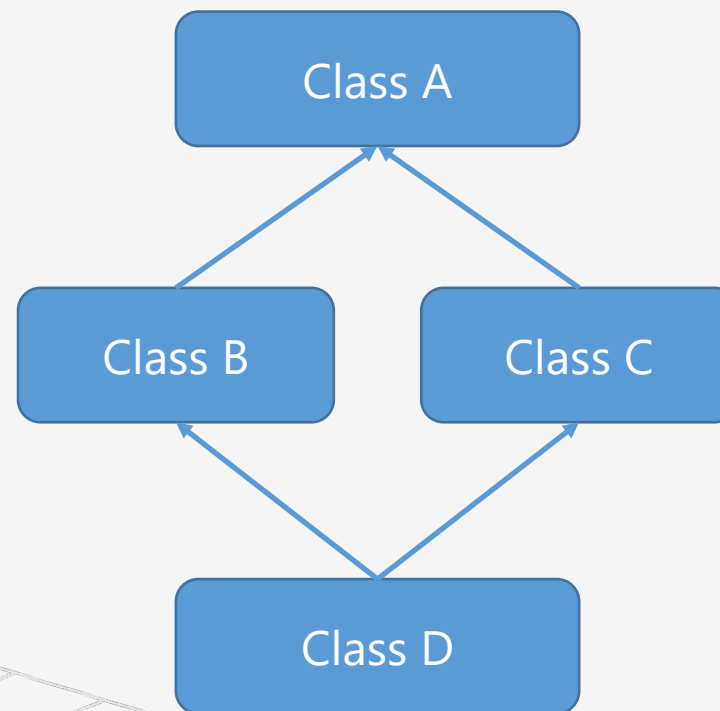
```
class C : public A, public B {  
    .....  
    .....  
};
```

- Lớp C kế thừa cả lớp A và B => đa thừa kế
- C mang đầy đủ đặc trưng của A và B, cộng thêm các đặc trưng mới của nó
- Tuy nhiên: nếu A và B có chung 1 đặc trưng, thì C sẽ lấy của A hay B? => nhập nhằng

# NHẬP NHẲNG TRONG ĐA THỪA KẾ

- Giả sử có sơ đồ thừa kế như hình vẽ

```
class A { public: void F(); };  
class B : public A {...};  
class C : public A {...};  
class D : public B, public C {...};  
D d;  
d.F(); // không dịch
```



# LỚP CƠ SỞ ẢO (VIRTUAL BASE CLASS)

---

- Trong trường hợp ví dụ trên; khi gọi `d.F()`; thì trình biên dịch không biết là đi theo nhánh kế thừa nào
- Có hai cách giải quyết
  - Có thể chỉ thêm đường đi: `d.B::F()`; hoặc `d.C::F()`;
  - Khai báo A là **lớp cơ sở ảo** đối với B, C; thêm từ khóa `virtual` vào trước A trong khai báo kế thừa A từ B và C. Khi đó trình biên dịch chỉ coi như có 1 lớp cơ sở A.



# KHAI BÁO LỚP CƠ SỞ ẢO

---

```
class A {  
    .....  
    void F();  
    .....  
};
```

```
class B : virtual public A {  
    .....  
    .....  
};
```

```
class C : virtual public A {  
    .....  
    .....  
};
```

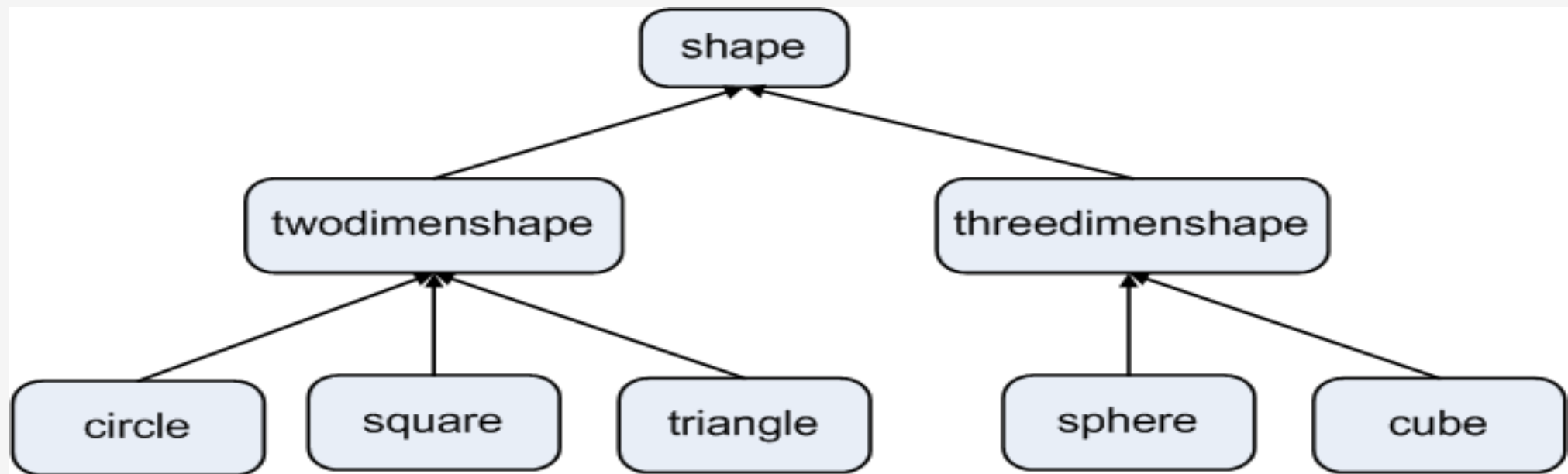
```
class D : public B, public C {  
    .....  
    .....  
};
```

Khi đó:  
D d;  
d.F(); // OK

# BÀI TẬP TẠI LỚP

---

- Xây dựng các lớp theo cây kế thừa



Yêu cầu: Thiết kế các thành phần dữ liệu và hàm thành phần của các lớp sao cho thể hiện tối ưu tư tưởng kế thừa.

# TÍNH ĐA HÌNH (POLYMORPHISM)

---

- Khó đưa ra 1 định nghĩa chính xác ☹️
- Thông qua mô tả: Tính đa hình là khả năng cho phép sự thể hiện khác nhau của một phương thức ở các lớp kế thừa khác nhau
- Có thể nhìn thấy 3 hình thức thể hiện của tính đa hình
  - **Overloading (quá tải hoặc nạp chồng)**: khả năng viết lại các hàm với sự khác nhau về tham số
  - **Overriding (Ghi đè)**: là khả năng thay thế các phương thức của lớp cha, ông, bằng chính phương thức đó ở lớp con, cháu.
  - **Late binding (Kết nối muộn)**: tức là quyết định gọi phương thức của lớp nào được đưa ra khi chương trình thực thi chứ không phải khi dịch chương trình

# PHƯƠNG THỨC ẢO (VIRTUAL FUNCTION)

---

- Kết nối tĩnh (static binding): kết nối một lời gọi hàm đến một hàm tại thời điểm dịch chương trình (compile time)
- Kết nối động (dynamic binding; hay còn gọi kết nối muộn – late binding): kết nối một lời gọi hàm đến một hàm tại thời điểm chạy chương trình (run time)
- Phương thức ảo (virtual function) là phương thức được kết nối động khi chạy chương trình

# PHƯƠNG THỨC ẢO (VIRTUAL FUNCTION)

---

- Khi xây dựng các lớp cho 1 cây kế thừa, các hành vi chung của các lớp được khai báo dưới dạng phương thức ở trong lớp cơ sở. Phương thức này có thể được thể hiện khác nhau ở các lớp kế thừa. Khi đó phương thức này được khai báo là phương thức ảo
- Cách khai báo:
  - Thêm từ khóa `virtual` vào trước khai báo hàm, ví dụ: *`virtual void display();`*
  - Một phương thức được khai báo ảo trong lớp cơ sở thì nó cũng ảo trong các lớp dẫn xuất (trong các lớp dẫn xuất có thể thêm từ khóa *`virtual`* hoặc không cần)

## PHƯƠNG THỨC ẢO (VIRTUAL FUNCTION)

---

- Phương thức ảo giống như một hàm quá tải (nạp chồng), nhưng điều đặc biệt ở đây là: **kiểu dữ liệu, số lượng và kiểu dữ liệu của các tham số của hàm ảo ở cả hai lớp cơ sở và dẫn xuất đều như nhau**
- Phương thức ảo tạo nên tính đa hình trong c++, bởi vì có thể nói một "giao diện" cho nhiều phương thức. Có nghĩa là với một thông điệp sẽ cho các kết quả khác nhau khi con trỏ của lớp cơ sở trỏ đến đối tượng của một lớp dẫn xuất nào đó
- Phương thức ảo được gọi theo cơ chế kết nối muộn

## VÍ DỤ 8-3

```
1 //vd8-3.cpp
2 #include <iostream>
3 using namespace std;
4 class A{
5     public:
6     virtual void display();
7 };
8 void A::display() {
9     cout << "Day la ham display cua lop A !";
10    cout << endl;
11 }
12 //lop B ke thua lop A
13 class B: public A{
14     public:
15     void display();
16 };
```

```
17 void B::display() {
18     cout << "Day la ham display cua lop B !";
19     cout << endl;
20 }
21 // ham show co 1 doi so la doi tuong lop A
22 void show(A *a) {
23     a->display(); // goi ham display
24 }
25 main() {
26     A a, *pA;
27     B b, *pB;
28     pA = &a; pB = &b;
29     pA->display();
30     pB->display();
31     show(pA); // goi A::display()
32     show(pB); // goi B::display()
33     return 0;
34 }
```

Select D:\DATA\OOP2020\codes\vd8-3.exe

```
Day la ham display cua lop A !
Day la ham display cua lop B !
Day la ham display cua lop A !
Day la ham display cua lop B !
```

## VÍ DỤ 8-3

---

- Hàm void display(); là một hàm ảo trong lớp cơ sở A, nên nó cũng làm hàm ảo trong lớp dẫn suất B
  - Hàm show(A \*a); có đối số là một con trỏ kiểu lớp cơ sở A
  - Lời gọi hàm show(pA) thì sẽ gọi hàm **A::display()**
  - Lời gọi hàm show(pB) thì nếu display() không phải là hàm ảo, thì sẽ gọi hàm **A::display()** ; nhưng vì display() là hàm ảo, nên sẽ gọi **B::display()**
- => Lời gọi hàm này được kết nối muộn, khi thực thi chương trình mới quyết định là gọi hàm của đối tượng thuộc lớp nào



# CƠ CHẾ KẾT NỐI MUỘN

---

## Cơ chế của việc kết nối muộn:

- Khi có khai báo hàm virtual trong lớp cơ sở thì trình biên dịch sẽ thêm vào mỗi đối tượng của lớp cơ sở và các lớp dẫn xuất của nó một con trỏ trỏ đến **bảng phương thức ảo** (con trỏ này có tên là vptr).
- Mỗi lớp có 1 bảng phương thức ảo.
- Bảng này chứa các con trỏ trỏ đến các đoạn chương trình đã biên dịch của các phương thức ảo. Trình biên dịch chỉ tạo các bảng phương thức ảo khi có khai báo đối tượng, đến khi thực thi chương trình thì phương thức ảo của lớp mới được kết nối và thực thi thông qua con trỏ vptr.

# CÁC ĐẶC TRƯNG CỦA PHƯƠNG THỨC ẢO

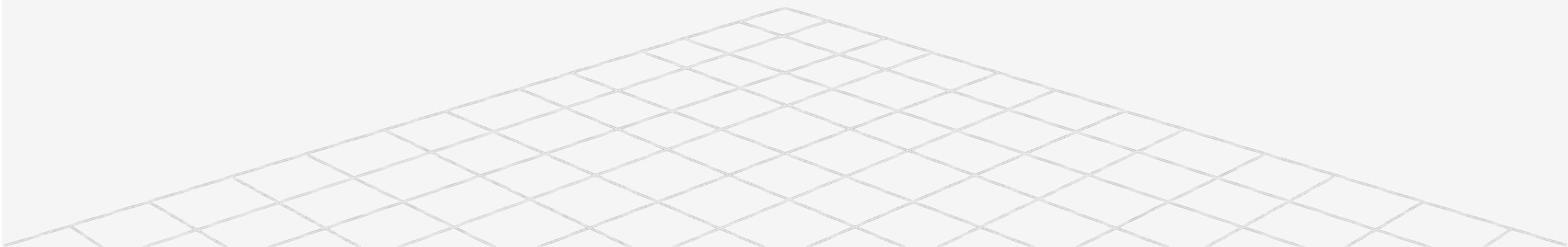
---

- Phương thức ảo không thể là phương thức tĩnh
- Không cần thiết phải thêm từ khóa virtual vào trước định nghĩa hàm ảo trong lớp dẫn xuất
- Một phương thức ảo trong lớp cơ sở thì phải được khai báo giữ nguyên kiểu dữ liệu và danh sách tham số trong lớp dẫn xuất thì mới đảm bảo là phương thức ảo trong lớp dẫn xuất. Nếu không trình biên dịch sẽ coi là một hàm thành viên khác

## VÍ DỤ 8-4

---

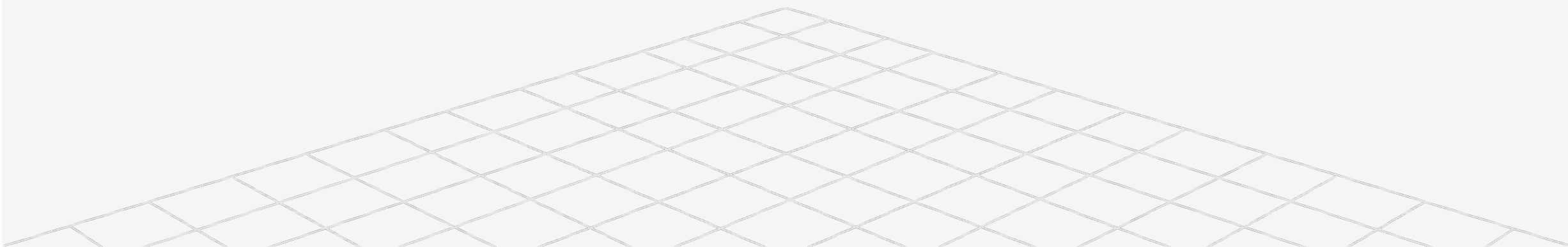
Tự đề xuất và xây dựng một cây thừa kế gồm 1 lớp cơ sở và 1 lớp dẫn xuất, trong đó có phương thức ảo và giải thích ý nghĩa của phương thức ảo trong lớp (tương tự như Ví dụ 8-3).



# LỚP TRỪU TƯỢNG (ABSTRACT CLASS)

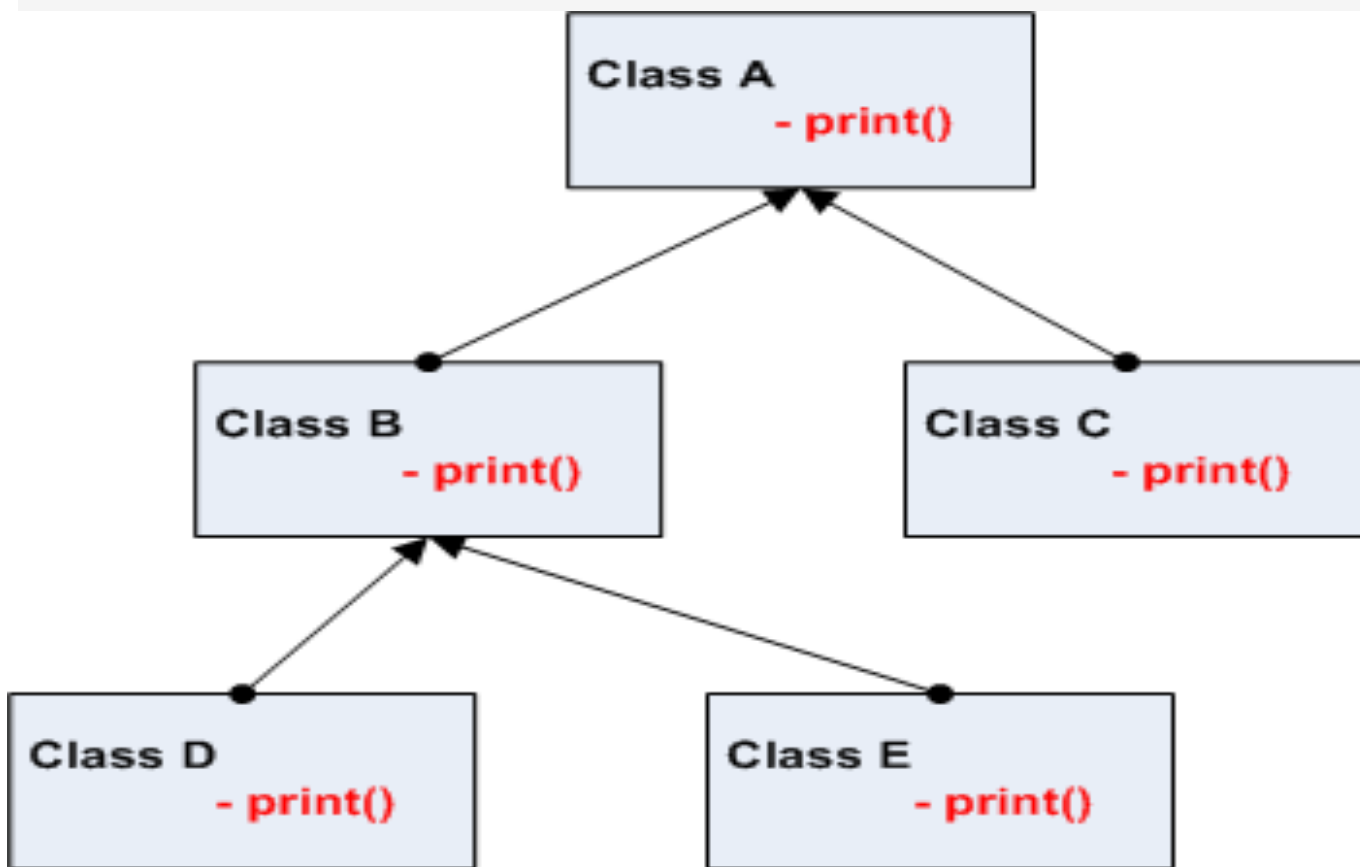
---

- Khi thiết kế các lớp kế thừa thường phải tìm các phương thức được sử dụng chung cho các lớp đưa lên lớp cơ sở
- Để tránh dư thừa, C++ cho phép thiết kế các lớp có các phương thức ảo không làm gì cả và cũng không thể tạo đối tượng thuộc lớp này. Những lớp này được gọi là **lớp trừu tượng**



# LỚP TRỪU TƯỢNG (ABSTRACT CLASS)

- Giả sử cần xây dựng cây kế thừa



- Giả sử không phải tất cả các lớp đều cần đến phương thức `print()`; nhưng nó có mặt ở khắp các lớp để tạo bộ mặt chung của các lớp kế thừa.
- Khi đó phương thức `print()` trong lớp cơ sở thường là không làm gì cả. Chỉ nhằm mục đích phục vụ cho các lớp dẫn xuất

# PHƯƠNG THỨC ẢO THUẦN TÚY (PURE VIRTUAL FUNCTION)

---

- Thông thường đối với các phương thức không thực hiện gì, chúng ta có thể để rỗng
  - Ví dụ: `virtual void print() { }`
  - Tuy nhiên có thể khai báo các đối tượng thuộc kiểu lớp có chứa hàm rỗng => dư thừa
- C++ cho phép định nghĩa hàm ảo thuần túy thay cho hàm rỗng
  - Ví dụ: `virtual void print() = 0;`
  - Khi đó không thể khai báo các đối tượng thuộc kiểu lớp có chứa hàm ảo thuần túy;
  - Lớp này chỉ nhằm mục đích phục vụ cho các lớp dẫn xuất phía sau

## VÍ DỤ 8-5

```
1 //vd8-5.cpp
2 #include <iostream>
3 using namespace std;
4 // lop truu tuong
5 class shape{
6     public:
7         float l;
8         void getData();
9         // ham ao thuan tuy
10        virtual float calculateArea() = 0;
11 };
12 void shape::getData() {
13     cin >> l;
14 }
```

```
12 void shape::getData() {
13     cin >> l;
14 }
15 // lop square ket thua lop shape
16 class square : public shape {
17     public:
18         float calculateArea();
19 };
20 float square::calculateArea() {
21     return l * l; // dien tich hinh vuong
22 }
23 // lop circle ke thua lop shape
24 class circle : public shape {
25     public:
26         float calculateArea();
27 };
```



## VÍ DỤ 8-5

```
28 float circle::calculateArea() {  
29     return 3.14 * l; // dien tich hinh tron  
30 }  
31 // thu nghiem  
32 int main() {  
33     square s;  
34     circle c;  
35     cout << "\nCho biet chieu dai canh hinh vuong: ";  
36     s.getData();  
37     cout<<"Dien tich hinh vuong la: " << s.calculateArea();  
38     cout<<"\nCho biet ban kinh hinh tron: ";  
39     c.getData();  
40     cout << "Dien tich hinh tron la: " << c.calculateArea();  
41     return 0;  
42 }
```

Gọi phương thức  
calculateArea() của  
lớp square

Gọi phương thức  
calculateArea() của  
lớp circle

D:\DATA\OOP2020\codes\vd8-5.exe

```
Cho biet chieu dai canh hinh vuong: 5  
Dien tich hinh vuong la: 25  
Cho biet ban kinh hinh tron: 8  
Dien tich hinh tron la: 25.12
```



# BÀI TẬP TẠI LỚP

---

- Tự thiết kế và xây dựng 1 cây kế thừa đơn giản: gồm cả dữ liệu và các hàm thành phần; trong đó có xây dựng các lớp trừu tượng. Lý giải về sự phù hợp và ưu việt của việc kế thừa, của việc sử dụng lớp trừu tượng trong mô hình bài toán bạn đặt ra
- Lập trình và thử nghiệm chương trình với cây kế thừa bạn vừa nghĩ ra bên trên

