# Inheritance

Nguyễn Văn Khiết

# Nội dung

- Lớp kế thừa
- Sử dụng phương thức của lớp cha
- Định nghĩa lại phương thức
- Mối quan hệ IS A
- Đa hình
- Hàm ảo

# Lớp kế thừa

- Kế thừa trong lập trình hướng đối tượng là việc định nghĩa một lớp mới dựa trên một lớp có sẵn. ➔ Lớp mới có những thuộc tính và phương thức của lớp có sẵn.
- Kế thừa giúp tái sử dụng mã nguồn → giảm thời gian, chi phí cho việc phát triển sản phẩm.
- Kế thừa giúp giảm thời gian và công sức khi có thay đổi trong mã nguồn

# Lớp kế thừa

- Khi lớp B kế thừa lớp A
  - Lớp A là lớp cơ sở (base class), lớp cha (super class)
  - Lớp B là lớp kế thừa, lớp con (sub class), lớp dẫn xuất (derived class)
  - Lớp B có thể sử dụng lại các thuộc tính và phương thức của lớp A
  - Lớp B có thể được bổ sung thêm một số thuộc tính và phương thức riêng.

# Lớp kế thừa

- Khai báo kế thừa trong C++

  **class <Lớp kế thừa> : <Loại kế thừa> <Lớp cơ sở>**

- Ví dụ:

```
class B : public A
{
    private:
        // Các thành phần bổ sung private của B
    public:
        // Các thành phần bổ sung public của B
};
```

# Lớp kế thừa

- Khi lớp B kế thừa lớp A:
  - Các thành viên (thuộc tính và phương thức) private của A sẽ không được kế thừa
  - Phạm vi của các thành viên được kế thừa không được mở rộng, chỉ có thể được thu hẹp.

# Lớp kế thừa

- Các loại kế thừa: quy định phạm vi tối đa các thành viên trong lớp được kế thừa
  - public (loại mặc định): giữ nguyên phạm vi
  - protected: phạm vi tối đa là protected
  - private: phạm vi tối đa là private

# Lớp kế thừa

- Các loại kế thừa:
  - public: giữ nguyên phạm vi các thành phần **public** và **protected**
  - protected: các thành phần **public**, **protected** ở lớp cha là thành phần **protected** ở lớp con.
  - private: tất cả các thành phần **public**, **protected** ở lớp cha là thành phần **private** ở lớp con.

# Lớp kế thừa

- Ví dụ:

```
class Rectangle {
protected:
    float mWidth;
    float mHeight;
public:
    Rectangle();
    Rectangle(float, float);
    virtual ~Rectangle();
    float Area();
};
class Square : public Rectangle {
public:
    Square();
    Square(float);
    virtual ~Square();
};
```

# Sử dụng phương thức của lớp cha

- Sử dụng constructor
  - Constructor không được kế thừa, phải viết lại constructor của lớp con trong đó có thể gọi constructor của lớp cha.
  - Gọi constructor của lớp cha:
    - Default: constructor của lớp con sẽ ngầm định gọi default constructor của lớp cha
    - Để tường minh gọi constructor của lớp cha, sử dụng cú pháp

```
LopCon::LopCon(kieudulieu x, kieudulieu y) : LopCha(x, y){
    //…
}
```

Ví dụ:

```
Square::Square(float w) : Rectangle (w, w){
}
```

# Sử dụng phương thức của lớp cha

- Trình tự khởi tạo đối tượng
  - Constructor của lớp cha sẽ được gọi trước, sau đó mới tới constructor của lớp con
  - Ví dụ:
    - Lớp Square kế thừa lớp Rectangle
    - Khi tạo một đối tượng của lớp Square thì constructor lớp Rectangle sẽ thực thi trước, sau đó mới tới constructor của lớp Square

# Sử dụng phương thức của lớp cha

- Trình tự hủy đối tượng
  - Destructor của lớp con sẽ được gọi trước, sau đó mới tới destructor của lớp cha`
  - Ví dụ:
    - Lớp Square kế thừa lớp Rectangle
    - Khi hủy một đối tượng của lớp Square thì destructor lớp Square sẽ thực thi trước, sau đó mới tới destructor của lớp Rectangle

# Sử dụng phương thức của lớp cha

- Các phương thức (non-private):
  - Được kế thừa nên có thể sử dụng bình thường.
  - Ví dụ

```
Square square(5);
square.Area();
```

# Định nghĩa lại phương thức

- Đôi khi, lớp con có thể thay đổi một vài phương thức của lớp cha
- ➔ Định nghĩa lại phương thức
- Ví dụ:

```
class MonHoc {
        …
public:
        MonHoc();
        float TinhDiemTongKet();
};

class MonCTDL: public MonHoc {
public:
        MonCTDL();
        float TinhDiemTongKet();
};
```

# Định nghĩa lại phương thức

- Gọi phương thức cùng tên của lớp cha:
  - Cú pháp

```
LopCha::TenPhuongThuc(x, y, z);
```

- Ví dụ:

```
//Lớp MonCTDL kế thừa lớp MonHoc
float MonCTDL::TinhDiemTongKet()
{
   float diem = MonHoc::TinhDiemTongKet();
   return ++diem;
}
```

# Quan hệ tổng quát hóa, đặc biệt hóa

- Kế thừa là sự thể hiện của quan hệ tổng quát hóa và đặc biệt hóa
- Ví dụ: MonCTDL, MonLTHDT kế thừa MonHoc
  - MonHoc là tổng quát hóa của các MonCTDL, MonLTHDT.
  - MonCTDL là đặc biệt hóa của MonHoc, nói cách khác, MonCTDL là một MonHoc (mối quan hệ IS A)

# Mối quan hệ IS A

- Lớp B kế thừa lớp A, ta có thể sử dụng như sau

  A* objA = new B();

- hoặc

  B objB;

  A* p = &objB;

- Ví dụ:

  Rectangle* pRec = new Square();

  (hình vuông là một hình chữ nhật)

# Tính đa hình

- Khi ta khai báo một con trỏ lớp cha, con trỏ đó có thể được cấp phát (hoặc trỏ tới vùng nhớ của) một đối tượng của lớp con của nó
- Con trỏ lớp cha tùy tình huống có thể mang hình thái là đối tượng của lớp con.
- ➔ tính đa hình

# Tính đa hình

- Ta có thể định nghĩa cách hành xử tổng quát của lớp cơ sở → các lớp con sẽ được bổ sung vào và có ứng xử riêng
- Tại thời điểm thực thi chương trình (run-time), hình thái cụ thể của đối tượng sẽ được tạo ra và đối tượng sẽ thực thi đúng với xử lý riêng của nó.
- → Tính đa hình giúp ta thiết kế hệ thống dễ dàng mở rộng

# Hàm ảo (virtual function)

- Hàm ảo là hàm thể hiện ứng xử đa hình của lớp đối tượng.
- Hàm ảo là hàm nên được viết lại tại các lớp con.
- Khai báo hàm ảo: thêm từ khóa virtual phía trước.

```
class HeightFigure {
protected:
        float mWidth;
        float mHeight;
public:
        HeightFigure();
        HeightFigure(float, float);
        virtual ~HeightFigure();
        virtual float Area();
};
```

```
class Triangle : public HeightFigure{
 public:
        Triangle();
        Triangle(float, float);
        virtual ~Triangle();
        virtual float Area();
 };
```

# Hàm ảo (virtual function)

- Khi gọi một hàm ảo từ con trỏ đối tượng lớp cha, tùy vào hình thái của đối tượng đó, hàm tương ứng của lớp con sẽ được gọi.
- Ví dụ:

```
HeightFigure *t = new Triangle(3,4);
cout<<t->Area();
```

# Hàm thuần ảo

- Hàm thuần ảo là hàm chỉ có khai báo mà không có phần cài đặt.
- Lớp có hàm thuần ảo là lớp trừu tượng (abstract class)
  - Ta không thể tạo được đối tượng của lớp trừu tượng
- Ví dụ khai báo hàm thuần ảo

```
virtual float Area() = 0;
```

# Hàm thuần ảo

```
class HeightFigure {
protected:
    float mWidth;
    float mHeight;
public:
    HeightFigure();
    HeightFigure(float, float);
    virtual ~HeightFigure();
    virtual float Area() = 0;
};
```

- Với khai báo trên, ta không thể khai báo đối tượng của lớp HeightFigure.

# Hàm hủy ảo

- Hàm hủy của lớp con sẽ tự động gọi hàm hủy của lớp cha **sau khi nó thực hiện xong**
- Hàm hủy nên luôn luôn là hàm ảo, để hàm hủy của lớp con được gọi. (sau đó hàm hủy lớp cha sẽ tự động được gọi)
- Ví dụ:

```
HeightFigure *t = new Triangle(3,4);
cout<<t->Area();
delete t;
```

# Bài tập

- Viết chương trình C++ thực hiện
  - Nhập vào từ bàn phím n, sau đó cho người dùng nhập n hình. Tại thời điểm nhập, với mỗi hình người dùng quyết định hình đó là 1 trong các hình sau: hình chử nhật, hình vuông, hình tam giác.
  - Lưu thông tin danh sách hình xuống file
  - Đọc thông tin danh sách hình từ file
  - Tính tổng diện tích của các hình trong danh sách
  - Loại/Bổ sung thêm hình vào danh sách