



NGUYÊN LÝ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Bài 9: Đa hình và Hàm ảo

Giảng viên: TS. Lý Anh Tuấn

Email: tuarla@tlu.edu.vn

Nội dung

1. Cơ bản về hàm ảo

- Kết gán muợn
- Thi hành hàm ảo
- Khi nào sử dụng hàm ảo
- Lớp trừu tượng và hàm ảo thuần túy

2. Con trỏ và hàm ảo

- Tương thích kiểu mở rộng
- Ép kiểu lên và ép kiểu xuống

Cơ bản về hàm ảo

- Đa hình
 - Liên kết nhiều ngữ nghĩa với một hàm
 - Hàm ảo cung cấp khả năng này
 - Là nguyên tắc cơ bản của lập trình hướng đối tượng
- Ảo
 - Tồn tại về bản chất mặc dù không ở dạng thực
- Hàm ảo
 - Có thể được sử dụng trước khi được định nghĩa

Ví dụ hình vẽ

- Lớp các kiểu hình vẽ
 - Hình chữ nhật (rectangle), hình tròn (circle), hình ovan (oval), vân vân
 - Mỗi hình vẽ là một đối tượng thuộc các lớp khác nhau
 - Dữ liệu hình chữ nhật: độ cao, chiều rộng, tâm điểm
 - Dữ liệu hình tròn: tâm điểm, bán kính
- Tất cả dẫn xuất từ một lớp cha: Figure
- Hàm cần có: draw()
 - Sử dụng chỉ thị khác nhau cho mỗi hình vẽ

Ví dụ hình vẽ: center()

- Mỗi lớp cần một hàm draw khác nhau
- Có thể gọi draw trong mỗi lớp:
Rectangle r;
Circle c;
r.draw(); //Gọi hàm draw của lớp Rectangle
c.draw(); // Gọi hàm draw của lớp Circle
- Lớp cha Figure bao gồm các hàm áp dụng cho tất cả các hình vẽ; chẳng hạn: center(): di chuyển hình vẽ vào tâm của màn hình
 - Xóa hình ban đầu, sau đó vẽ lại
 - Do vậy Figure::center() sẽ gọi hàm draw để vẽ lại
 - Vấn đề: Gọi hàm draw() từ lớp nào?

Ví dụ hình vẽ: Hình mới

- Xét kiểu hình vẽ mới như sau:
lớp Triangle
được dẫn xuất từ lớp Figure
- Hàm `center()` được kế thừa từ Figure
 - Nó có làm việc với hình tam giác không?
 - Nó sử dụng `draw()` khác nhau với mỗi hình
 - Nó sẽ sử dụng `Figure::draw()` → không làm việc với hình tam giác
- Cần hàm `center()` được kế thừa sử dụng hàm `Triangle::draw()` chứ không phải hàm `Figure::draw()`
 - Nhưng lớp Triangle thậm chí còn chưa được viết khi viết `Figure::center()`

Ví dụ hình vẽ: Hình ảo

- Câu trả lời là sử dụng hàm ảo
- Nói cho bộ biên dịch:
 - Không biết hàm được thi hành như thế nào
 - Đợi cho đến khi được sử dụng trong chương trình
 - Khi đó nhận thi hành từ bản thể đối tượng
- Được gọi là kết gán muộn hoặc kết gán động
 - Hàm ảo thi hành kết gán muộn

Một ví dụ khác

- Chương trình ghi sổ của cửa hàng bán phụ tùng ô tô
 - Theo dõi các giao dịch
 - Chưa biết tất cả các giao dịch
 - Ban đầu chỉ có các giao dịch bán lẻ
 - Sau đó: Giao dịch giảm giá, thư đặt hàng, vận vận
 - Phụ thuộc vào các nhân tố khác bên cạnh giá và thuế

Hàm ảo: Phụ tùng ô tô

- Chương trình phải:
 - Tính toán tổng doanh thu hàng ngày
 - Tính toán giao dịch giá trị lớn nhất/nhỏ nhất trong ngày
 - Có thể tính giá trị trung bình của các giao dịch trong ngày
- Tất cả đến từ các hóa đơn lẻ
 - Nhưng nhiều hàm tính hóa đơn sẽ được thêm vào sau
 - Khi các kiểu giao dịch khác được thêm vào
- Do vậy hàm tính hóa đơn sẽ là ảo

Định nghĩa lớp Sale

- class Sale

```
{
public:
    Sale();
    Sale(double thePrice);
    double getPrice() const;
    virtual double bill() const;
    double savings(const Sale& other) const;
private:
    double price;
};
```

Hàm thành viên: savings và toán tử



- ```
double Sale::savings(const Sale& other) const
{
 return (bill() – other.bill());
}
```
- ```
bool operator < (      const Sale& first,
                    const Sale& second)
{
    return (first.bill() < second.bill());
}
```
- Lưu ý cả hai đều sử dụng hàm thành viên bill()

Lớp Sale

- Biểu diễn các giao dịch của một mặt hàng không tính giảm giá hoặc phụ phí
- Lưu ý giữ nguyên từ virtual trong khai báo của hàm thành viên bill
 - Hiệu quả: Sau này, các lớp dẫn xuất của Sale có thể định nghĩa phiên bản hàm hóa đơn của chúng
 - Các hàm thành viên khác của Sale sẽ sử dụng phiên bản dựa vào đối tượng của lớp dẫn xuất
 - Chúng sẽ không tự động sử dụng phiên bản của Sale

Định nghĩa lớp dẫn xuất DiscountSale

- class DiscountSale : public Sale
{
public:
 DiscountSale();
 DiscountSale(double thePrice,
 double the Discount);
 double getDiscount() const;
 void setDiscount(double newDiscount);
 double bill() const;
private:
 double discount;
};

Thi hành bill() của DiscoutSale

- Hàm ảo trong lớp cơ sở
 - Tự động ảo trong lớp dẫn xuất
- Khai báo lớp dẫn xuất (trong giao diện)
 - Không đòi hỏi có từ khóa “virtual”
 - Nhưng thường được thêm vào cho dễ đọc

Lớp dẫn xuất DiscountSale

- Hàm thành viên bill() của DiscountSale thì khác với của Sale
 - Đặc biệt với việc giảm giá
- Các hàm thành viên savings và “<”
 - Sẽ sử dụng định nghĩa này của bill cho tất cả các đối tượng của lớp DiscountSale
 - Thay vì để mặc định với phiên bản được định nghĩa trong lớp Sale

Hàm ảo

- Lớp Sale được viết trước lớp dẫn xuất DiscountSale
 - Các hàm thành viên savings và “<” được biên dịch thậm chí trước khi có ý tưởng về lớp DiscountSale
- Do vậy trong một lời gọi như:
DiscountSale d1, d2;
d1.savings(d2);
 - Lời gọi tới hàm bill() trong savings() biết sử dụng định nghĩa bill() từ lớp DiscountSale
- Rất hữu ích

Hàm ảo

- Giải thích về kết gán muộn
 - Hàm ảo thi hành kết gán muộn
 - Nói với bộ biên dịch đợi cho đến khi hàm được sử dụng trong chương trình
 - Quyết định định nghĩa nào được sử dụng dựa vào việc gọi đối tượng
- Là quy tắc lập trình hướng đối tượng rất quan trọng

Ghi đè

- Định nghĩa hàm ảo thay đổi trong lớp dẫn xuất
 - Chúng ta nói là nó được ghi đè
- Tương tự như các hàm chuẩn được định nghĩa lại
- Như vậy:
 - Hàm ảo bị thay đổi: ghi đè
 - Hàm không phải hàm ảo bị thay đổi: định nghĩa lại

Hàm ảo: Nhược điểm

- Chúng ta đã biết các ưu điểm của hàm ảo
- Nhược điểm chính: phụ phí
 - Sử dụng nhiều bộ nhớ hơn
 - Kết gán muộn là “trong khi chạy”, do vậy chương trình chạy chậm hơn
- Do vậy không nên sử dụng hàm ảo nếu không thực sự cần thiết

Hàm ảo thuần túy

- Lớp cơ sở có thể không định nghĩa rõ nghĩa một vài thành viên
 - Mục đích chỉ để các lớp khác kế thừa
- Nhắc lại lớp Figure
 - Tất cả các hình vẽ là các đối tượng của các lớp dẫn xuất:
 - Hình chữ nhật, hình tròn, hình tam giác, vân vân.
 - Lớp Figure không có ý tưởng về việc vẽ như thế nào
- Đặt nó là một hàm ảo thuần túy:
`virtual void draw() = 0;`

Lớp cơ sở trừu tượng

- Hàm ảo thuần túy không đòi hỏi phải định nghĩa
 - Bắt tất cả các lớp dẫn xuất định nghĩa phiên bản của riêng chúng
- Lớp có một hoặc nhiều hàm ảo thuần túy là: lớp cơ sở trừu tượng
 - Có thể được sử dụng như lớp cơ sở
 - Không cho phép tạo đối tượng từ nó
 - Vì nó không có các định nghĩa hoàn chỉnh cho tất cả các thành viên
- Nếu lớp dẫn xuất thất bại trong việc định nghĩa tất cả các hàm ảo thuần túy
 - Nó cũng là lớp cơ sở trừu tượng

Tương thích kiểu mở rộng

- Biết rằng:
Derived là lớp dẫn xuất của Base
 - Các đối tượng Derived có thể được gán cho các đối tượng kiểu Base
 - Nhưng ngược lại thì không
- Xét ví dụ trước:
 - Một DiscountSale là một Sale nhưng ngược lại thì không đúng

Ví dụ tương thích kiểu mở rộng

- ```
class Pet
{
public:
 string name;
 virtual void print() const;
};
class Dog : public Pet
{
public:
 string breed;
 virtual void print() const;
};
```

# Lớp Pet và Dog

- Bây giờ cung cấp khai báo:  
Dog vdog;  
Pet vpet;
- Lưu ý các biến thành viên name và breed là public
  - Chỉ nhằm mục đích ví dụ, không điển hình

# Sử dụng lớp Pet và Dog

- Một chú chó cũng là một thú cưng:
  - `vdog.name = "Tiny";`  
`vdog.breed = "Great Dane";`  
`vpet = vdog;`
  - Những lệnh này được phép
- Có thể gán giá trị cho kiểu cha, nhưng ngược lại thì không
  - Một thú cưng không phải là một chú chó

# Vấn đề tách lớp

- Lưu ý rằng giá trị được gán cho vpet mất trường breed của nó
  - `cout<<vp.pet.breed;`
    - Tạo ra thông điệp lỗi
  - Được gọi là vấn đề tách lớp
- Dường như là phù hợp
  - Dog được chuyển thành biến Pet, nên nó sẽ được đối xử như một Pet
    - Và do vậy không có các tính chất của dog
  - Tạo ra tranh cãi triết học thú vị

# Khắc phục vấn đề tách lớp

- Trong C++, vấn đề tách lớp gây trở ngại
  - Nó vẫn là một Great Dane tên là Tiny
  - Chúng ta muốn tham chiếu tới giống của nó thậm chí khi nó được đối xử như một Pet
- Có thể làm điều này bằng con trỏ trỏ tới biến động

# Ví dụ vấn đề tách lớp

- ```
Pet *ppet;  
Dog *pdog;  
pdog = new Dog;  
pdog->name = "Tiny";  
pdog->breed = "Great Dane";  
ppet = pdog;
```
- Không thể truy cập trường breed của đối tượng được trả bởi ppet:

```
cout << ppet->breed;           //Khong hop le
```

Ví dụ vấn đề tách lớp

- Phải sử dụng hàm thành viên ảo:
`ppet->print();`
 - Gọi hàm thành viên `print` trong lớp `Dog`
 - Bởi vì nó là ảo
- C++ đợi để xem con trỏ `ppet` đang trỏ tới đối tượng nào trước khi gọi kết gán.

Hàm hủy ảo

- Nhắc lại: hàm hủy cần để hủy cấp phát dữ liệu được cấp phát động
- Xét:
Base *pBase = new Derived;
...
delete pBase;
 - Sẽ gọi hàm hủy lớp cơ sở mặc dù đang trở tới đối tượng lớp Derived
 - Khai báo hàm hủy là virtual để khắc phục vấn đề này
- Có thể để tất cả các hàm hủy là ảo

Ép kiểu

- Xét:
Pet vpet;
Dog vdog;
...
vdog = static_cast<Dog>(vpет); //Không hợp lệ!
- Không thể ép một thú cưng thành một chú chó, nhưng:
vpет = vdog; // Hợp lệ!
vpет = static_cast<Pet>(vdog); //Cũng hợp lệ!
- Ép kiểu lên thực hiện được
 - Từ kiểu hậu duệ đến kiểu tổ tiên

Ép kiểu xuống

- Ép kiểu xuống khá nguy hiểm
 - Ép từ kiểu tổ tiên xuống kiểu hậu duệ
 - Giả sử thông tin được thêm vào
 - Có thể thực hiện với `dynamic_cast`:
Pet *ppet;
ppet = new Dog;
Dog *pdog = dynamic_cast<Dog*>(ppet);
 - Hợp lệ, nhưng nguy hiểm
- Ép kiểu xuống hiếm khi được sử dụng do:
 - Phải theo dõi tất cả thông tin được thêm vào
 - Tất cả các hàm thành viên phải là ảo

Tóm tắt

- Kết gán muộn hoãn việc quyết định hàm thành viên nào được gọi cho đến khi chương trình chạy
 - Trong C++, hàm ảo sử dụng kết gán muộn
- Các hàm ảo thuần túy không có định nghĩa
 - Các lớp có ít nhất một hàm ảo thuần túy là trừu tượng
 - Không có đối tượng nào có thể tạo ra từ lớp trừu tượng
 - Được sử dụng làm cơ sở cho các lớp khác dẫn xuất

Tóm tắt

- Đối tượng lớp dẫn xuất có thể được gán cho đối tượng lớp cơ sở
 - Các thành viên lớp cơ sở bị mất: vấn đề tách lớp
- Phép gán con trỏ và các đối tượng động
 - Cho phép khắc phục vấn đề tách lớp
- Khai báo tất cả các hàm hủy là ảo
 - Thói quen lập trình tốt
 - Đảm bảo bộ nhớ được hủy cấp phát đúng

Bài tập

Xây dựng lớp `Hinh2D` có một dữ liệu thành viên là màu sắc và hai hàm ảo là `nhap()` và `tinhdientich()`, trong đó hàm `tinhdientich()` là hàm ảo thuần túy. Hàm truy cập và hàm biến đổi dữ liệu màu sắc.

Xây dựng các lớp `HinhTron`, `HinhChuNhat`, `HinhTamGiac` kế thừa lớp `Hinh2D`, trong đó:

- Lớp `HinhTron` có một dữ liệu thành viên mô tả bán kính hình tròn, lớp `HinhChuNhat` có hai dữ liệu thành viên mô tả chiều dài, chiều rộng hình chữ nhật, lớp `HinhTamGiac` có ba dữ liệu thành viên mô tả độ dài ba cạnh hình tam giác.
- Các lớp này cũng bao gồm các hàm tạo, các hàm truy cập, hàm biến đổi và ghi đè hai hàm ảo thuần túy của lớp cha.

a) Viết chương trình nhập vào số n và một danh sách n phần tử gồm các hình tròn, hình chữ nhật hoặc hình tam giác (do người dùng quyết định lúc nhập).

b) In ra tổng diện tích các hình có trong mảng.

Gợi ý

Khai báo một mảng các con trỏ kiểu Hình2D, sử dụng toán tử new để tạo ra các đối tượng động thuộc các kiểu dẫn xuất khác nhau:

```
Hinh2D *ds[100];
int k=0, chon, i;
while(1)
{
    cout<<"\n*Hinh tron/Hinh chu nhat/Hinh tam giac/Thoat (1,2,3,4):";
    cin>>chon; cin.ignore();
    if (chon==4) break;
    if (chon==1) ds[k]=new HinhTron();
    if (chon==2) ds[k]=new HinhChuNhat();
    if (chon==3) ds[k]=new HinhTamGiac();
    ds[k]->nhap();
    k++;
}
```