LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Chương 3: Lớp và đối tượng (p3)

Trình bày: ThS. Lê Thanh Trọng

1. Hàm hủy bỏ - Destructor
2. Thao tác với dữ liệu
3. Thành phần tĩnh (static)

1. **Hàm hủy bỏ - Destructor**
2. Thao tác với dữ liệu
3. Thành phần tĩnh (static)

Hàm hủy bỏ – Destructor

- Destructor, được gọi ngay trước khi một đối tượng bị thu hồi
- Destructor thường được dùng để thực hiện việc dọn dẹp cần thiết trước khi một đối tượng bị hủy
- Một lớp chỉ có duy nhất một Destructor
- Phương thức Destructor trùng tên với tên lớp nhưng có dấu ~ đặt trước

Hàm hủy bỏ – Destructor

- Nên xây dựng khi lớp có thành phần dữ liệu là con trỏ
- Được tự động gọi thực hiện khi đối tượng hết phạm vi sử dụng
  - Hết phạm vi khai báo đối tượng/ mảng đối tượng
  - Lệnh hủy (delete) đối với con trỏ được gọi
- Destructor phải có thuộc tính public

Ví dụ destructor

```
class DaThuc
{
    private:
    int Bac;
    float* HeSo;
    public:
    DaThuc(){}
    DaThuc(int, float);
    DaThuc(const DaThuc&);
    void NhapDaThuc();
    void XuatDaThuc();
    DaThuc CongDaThuc(const
    DaThuc&);
    DaThuc TruDaThuc(const
    DaThuc&);
    DaThuc NhanDaThuc(const
    DaThuc&);
    ~DaThuc();
};

DaThuc::~DaThuc()
{
    if (HeSo != NULL)
    delete[]HeSo;
    HeSo = NULL;
}
```

1. Hàm hủy bỏ - Destructor
2. **Thao tác với dữ liệu**
3. Thành phần tĩnh (static)

Phương thức truy vấn

- Hay còn được gọi là các **getter**
- Có nhiều loại câu hỏi truy vấn có thể:
  - Truy vấn đơn giản (“giá trị của x là bao nhiêu?”)
  - Truy vấn điều kiện (“thành viên x có > 10 không?”)
  - Truy vấn dẫn xuất (“tổng giá trị của các thành viên x và y là bao nhiêu?”)
- Đặc điểm quan trọng của phương thức truy vấn là nó không nên thay đổi trạng thái hiện tại của đối tượng

Phương thức truy vấn

- Đối với các truy vấn đơn giản, quy ước đặt tên phương thức như sau: Tiền tố "**Get**", tiếp theo là tên của thành viên cần truy vấn
  - int **GetX**();
  - int **GetSize**();
- Các loại truy vấn khác nên có tên có tính mô tả
- Truy vấn điều kiện nên có tiền tố "is"

Phương thức cập nhật

- Hay còn được gọi là các **setter**
- Thường để thay đổi trạng thái của đối tượng bằng cách sửa đổi một hoặc nhiều thành viên dữ liệu của đối tượng đó.
- Dạng đơn giản nhất là gán một giá trị nào đó cho một thành viên dữ liệu.
- Đối với dạng cập nhật đơn giản, quy ước đặt tên như sau: Dùng tiền tố "**Set**" kèm theo tên thành viên cần sửa
  - int S**etX**(int);

Truy vấn và cập nhật

- Nếu phương thức Get/Set chỉ có nhiệm vụ cho ta đọc/ghi giá trị cho các thành viên dữ liệu →Quy định các thành viên private để được ích lợi gì?
  - Ngoài việc bảo vệ các nguyên tắc đóng gói, ta cần kiểm tra xem giá trị mới cho thành viên dữ liệu có hợp lệ hay không
  - Sử dụng phương thức truy vấn cho phép ta thực hiện việc kiểm tra trước khi thực sự thay đổi giá trị của thành viên
  - Chỉ cho phép các dữ liệu có thể truy vấn hay thay đổi mới được truy cập đến

Ví dụ phương thức cập nhật dữ liệu

```
int Student::SetGPA (double NewGPA){
    if ((NewGPA >= 0.0) && (NewGPA <= 10.0)){
        this->GPA= NewGPA;
        return 0;        // Cập nhật thành công
    }
    else
    {
        return -1;       // Cập nhật không thành công
    }
}
```

1. Hàm hủy bỏ - Destructor
2. Thao tác với dữ liệu
3. **Thành phần tĩnh (static)**

Thành phần tĩnh – static member

- Trong C, static xuất hiện trước dữ liệu được khai báo trong một hàm nào đó thì giá trị của dữ liệu đó vẫn được lưu lại như một biến toàn cục
- Trong C++, nếu static xuất hiện trước một dữ liệu hoặc một phương thức của lớp thì giá trị của nó vẫn được lưu lại và có ý nghĩa cho tất cả các đối tượng khác của cùng lớp này (dùng chung)
- Các thành viên static có thể là public, private hoặc protected

Thành phần tĩnh – static member

- Đối với class, static dùng để khai báo thành viên dữ liệu dùng chung cho mọi thể hiện của lớp:
  - Một bản duy nhất tồn tại trong suốt quá trình chạy của chương trình
  - Dùng chung cho tất cả các thể hiện của lớp
  - Bất kể lớp đó có bao nhiêu thể hiện

Thành viên tĩnh – static member

- Các thành viên lớp tĩnh public có thể được truy cập thông qua bất kỳ đối tượng nào của lớp đó (.), hoặc chúng có thể được truy cập thông qua tên lớp sử dụng toán tử định phạm vi (::)
- Các thành viên lớp tĩnh private và protected phải được truy cập thông qua các hàm thành viên public của lớp hoặc thông qua các friend của lớp
- Các thành viên lớp tĩnh tồn tại ngay cả khi đối tượng của lớp đó không tồn tại

Ví dụ thành phần static (1)

```
class Rectangle
{
    private:
       int width;
       int length;
       static int count;
    public:
       Rectangle(){count++;}
       void Set(int w, int l);
       int GetArea();
}
```

```
Rectangle  r1;
Rectangle  r2;
Rectangle  r3;
```

count = 3

Ví dụ thành phần static (2)

❖Đếm số đối tượng MyClass:

```
class MyClass{
    public:
        MyClass();
        ~MyClass();
        void PrintCount();
    private:
        static int count;
};
```

Ví dụ thành phần static (2)

```
int MyClass::count = 0;
MyClass::MyClass(){
    this → count++;
}
MyClass::~MyClass(){
    this → count--;
}
void MyClass::PrintCount(){
    cout << “There are currently ” << this → count << “ instance(s) of
    MyClass.\n”;
}
```

Ví dụ thành phần static (2)

```
void main()
{
    MyClass* x = new MyClass();
    x → PrintCount();
    MyClass* y = new MyClass();
     x → PrintCount();
     y → PrintCount();
    delete x;
     y → PrintCount();
}
```

Ví dụ thành phần static (3)

Xét đoạn chương trình sau:

```
#include <iostream.h>
void main(){
    cout << "Hello, world.\n";
}
```

- Hãy sửa lại đoạn chương trình trên để có kết xuất:
  - Entering a C++ program saying...
  - Hello, world.
  - And then exitting…
- Yêu cầu không thay đổi hàm main() dưới bất kỳ hình thức nào.

Ví dụ thành phần static (3)

```
#include <iostream.h>
class Dummy{
public:
   Dummy(){cout << "Entering a C++ program saying...\n";}
   ~Dummy(){cout << "And then exitting...";}
};
Dummy A;
void main(){
    cout << "Hello, world.\n";
}
```

Tóm tắt về lớp và đối tượng (p3)

❖Hàm hủy bỏ – Destructor

- ❑Được dùng để thực hiện việc dọn dẹp cần thiết trước khi một đối tượng bị hủy nhằm thực hiện các thao tác “dọn dẹp”: giải phóng các tài nguyên, đóng các file hoặc các kết nối mạng, nhằm tránh rò rỉ tài nguyên
- ❑Được gọi ngay trước khi một đối tượng bị thu hồi
- ❑Một lớp chỉ có duy nhất một destructor (không tham số)
- ❑Phương thức Destructor trùng tên với tên lớp nhưng có dấu ~ đặt trước, không có kiểu trả về
- ❑Được tự động gọi thực hiện khi đối tượng hết phạm vi sử dụng
  - Hết phạm vi khai báo đối tượng/ mảng đối tượng ({ })
  - Lệnh hủy (delete) đối với con trỏ được gọi
- ❑Destructor phải có thuộc tính public
- ❑Nên xây dựng khi lớp có thành phần dữ liệu là con trỏ

Tóm tắt về lớp và đối tượng (p3)

❖Thao tác với dữ liệu (getter và setter)

- ❑Là các hàm thành viên được sử dụng để truy cập và thay đổi giá trị của các thuộc tính (biến thành viên) của một lớp thường được khai báo với phạm vi truy cập là private hoặc protected
- ❑Giúp kiểm soát cách các thuộc tính của đối tượng được truy cập và thay đổi, bảo vệ tính toàn vẹn của dữ liệu và thực thi các ràng buộc nếu cần thiết
- ❑**Getter**:
  - Được sử dụng để lấy (truy xuất) giá trị của một thuộc tính của đối tượng
  - Ví dụ: `public int GetValue() { return Value; }`
- ❑**Setter**:
  - Được sử dụng để thiết lập (cập nhật) giá trị của một thuộc tính của đối tượng
  - Ví dụ:

```
void setValue(int v) {
        if (v >= 0)
                value = v;
        else
                cout << "Gia tri khong hop le!" << endl;
}
```

Tóm tắt về lớp và đối tượng (p3)

❖Thành phần tĩnh (static)

❑Được sử dụng để khai báo các thành phần của lớp (biến hoặc hàm) mà **thuộc về lớp** chứ **không phải thuộc về các đối tượng cụ thể**

❑**Biến tĩnh (static variable):**

- Không thuộc về bất kỳ đối tượng nào của lớp mà thuộc về chính lớp đó
- Ví dụ: static int Count;

❑**Hàm tĩnh (static method):**

- Có thể được gọi mà không cần đối tượng. Nó **chỉ có thể truy cập các biến tĩnh và các hàm tĩnh khác trong lớp**, không thể truy cập các biến và hàm thành viên không tĩnh.
- Ví dụ: static int GetCount() { return Count; }


VS
Nạp Năng Lượng
SUPER STAR
SUPER STAR
Trượt