Lập trình hướng đối tượng và C++

Bài 8: Tương ứng bội

TS. Nguyễn Hiếu Cường
Bộ môn CNPM, Khoa CNTT, Trường Đại học GTVT
Email: cuonggt@gmail.com

Hà Nội, 2018

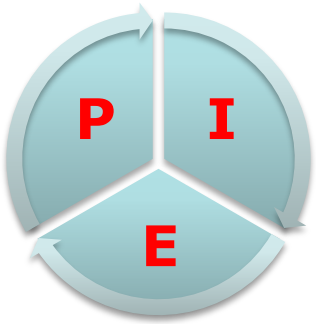
Nội dung chính

1. Giới thiệu môn học
2. Các khái niệm cơ bản
3. Hàm trong C++
4. Lớp và đối tượng
5. Định nghĩa chồng toán tử
6. Hàm tạo và hàm huỷ
7. Dẫn xuất và thừa kế
8. **Tương ứng bội**
9. Khuôn hình

Khái niệm tương ứng bội

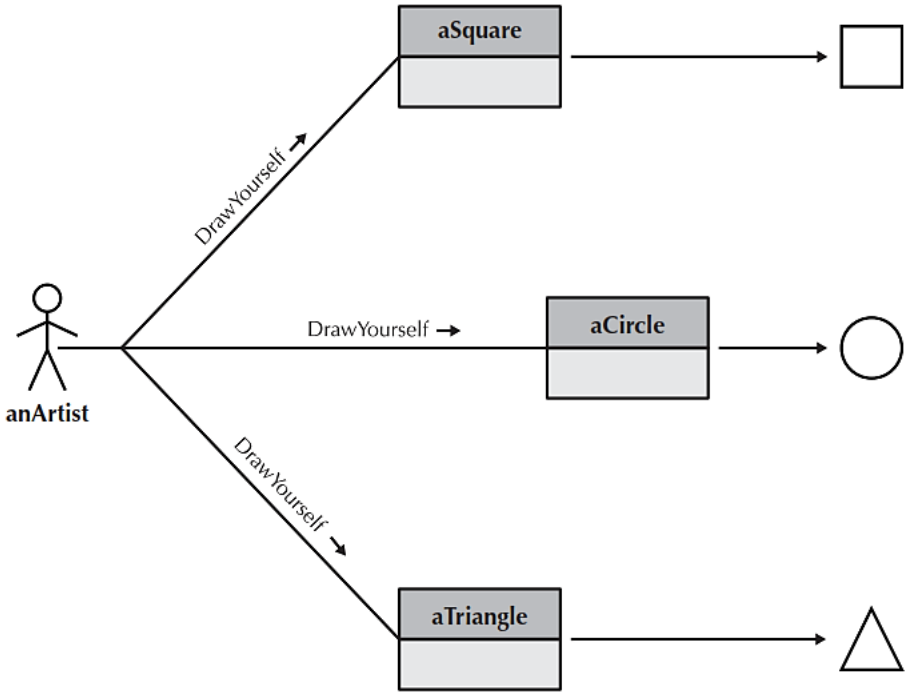
- Tương ứng bội (Polymorphism)
  - Còn gọi là "Tính đa hình"

  - Là một trong các "trụ cột" của OOP
    - Polymorphism
    - Inheritance
    - Encapsulation



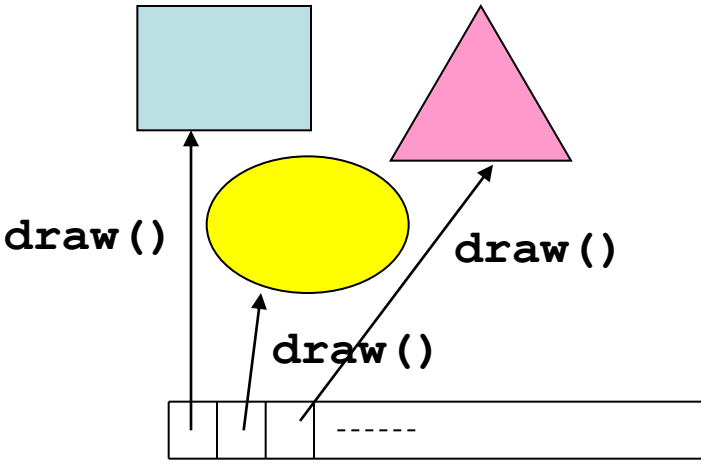
Khái niệm về tương ứng bội

- Xử lý các đối tượng của các lớp có liên quan theo một cách chung



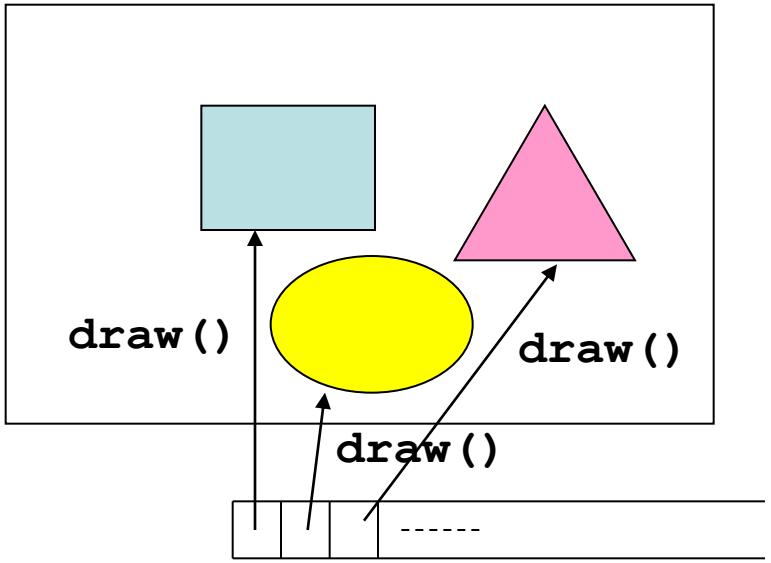
Tác dụng của tương ứng bội

- "Một giao diện, nhiều cài đặt"
- Chương trình đơn giản, rõ ràng và dễ bảo trì hơn
  - Chỉ cần có draw() cho tất cả các hình, thay vì cần các phương thức draw_square(), draw_circle(), draw_triangle()...
  - Dễ dàng phát triển chương trình mà không cần sửa đổi nhiều



Ví dụ

```
Shape* shapes[10];
...
for (i = 0; i < numShapes; i++)
{
    ...
    shapes[i] -> draw();
}
```



```cpp
// pointers to base class
#include <iostream>
using namespace std;

class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
};

class Rectangle: public Polygon {
  public:
    int area()
      { return width*height; }
};

class Triangle: public Polygon {
  public:
    int area()
      { return width*height/2; }
};

int main () {
  Rectangle rect;
  Triangle trgl;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << rect.area() << '\n';
  cout << trgl.area() << '\n';
  return 0;
}
```

```
20
10
```

```cpp
// virtual members
#include <iostream>
using namespace std;

class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area ()
      { return 0; }
};

class Rectangle: public Polygon {
  public:
    int area ()
      { return width * height; }
};

class Triangle: public Polygon {
  public:
    int area ()
      { return (width * height / 2); }
};

int main () {
  Rectangle rect;
  Triangle trgl;
  Polygon poly;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  Polygon * ppoly3 = &poly;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly3->set_values (4,5);
  cout << ppoly1->area() << '\n';
  cout << ppoly2->area() << '\n';
  cout << ppoly3->area() << '\n';
  return 0;
}
```

```
20
10
0
```

```cpp
// abstract base class
#include <iostream>
using namespace std;

class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void) =0;
};

class Rectangle: public Polygon {
  public:
    int area (void)
      { return (width * height); }
};

class Triangle: public Polygon {
  public:
    int area (void)
      { return (width * height / 2); }
};

int main () {
  Rectangle rect;
  Triangle trgl;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << ppoly1->area() << '\n';
  cout << ppoly2->area() << '\n';
  return 0;
}
```

```
20
10
```

```cpp
// pure virtual members can be called
// from the abstract base class
#include <iostream>
using namespace std;

class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area() =0;
    void printarea()
      { cout << this->area() << '\n'; }
};

class Rectangle: public Polygon {
  public:
    int area (void)
      { return (width * height); }
};

class Triangle: public Polygon {
  public:
    int area (void)
      { return (width * height / 2); }
};

int main () {
  Rectangle rect;
  Triangle trgl;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  return 0;
}
```

```
20
10
```

Phương thức tĩnh và động

- Phương thức tĩnh
  - Thông điệp truyền tới một đối tượng thì phương thức của lớp (mà đối tượng đó được khai báo) sẽ được thực hiện
  - Liên kết tĩnh = liên kết sớm (early binding): compile time
- Phương thức động (phương thức ảo)
  - Thông điệp truyền tới một đối tượng thì phương thức của lớp (tương ứng với đối tượng đó) sẽ được thực hiện
  - Liên kết động = liên kết muộn (late binding): run time

Phương thức ảo

- Phương thức động còn gọi là phương thức ảo (virtual function)
- Tên phương thức ảo phải hoàn toàn giống nhau ở tất cả các lớp (trong cùng hệ thống phân cấp lớp)
- Phương thức ảo được định nghĩa như phương thức thông thường nhưng thêm từ khóa **virtual** ở phía trước

```
virtual void draw();
```

Ví dụ (phương thức ảo)

```
class Shape {
public:
  virtual void draw()
  {
    cout<<"draw shape \n";
  }
  void paint()
    {
    cout<<"paint shape \n";
  }
};
class Square: public Shape {
public:
  void draw(){cout<<"draw square \n";}
  void paint(){cout<<"paint square"; }
};
class Circle: public Shape {
public:
  void draw() {cout<<"draw circle";}
  void paint() {cout<<"paint circle";}
};
```

```
int main()
{
  Shape *ptr;

  Circle c;
  Square s;
  ptr= &c;        // ptr chứa đc &c ?
  ptr->draw();
  ptr->paint();

  ptr= &s;
  ptr->draw();
  ptr->paint();
}
```

Có gì khác nhau giữa **draw()** và **paint()** ?

Lớp cơ sở trừu tượng

- Phương thức ảo thuần túy
  - Dùng trong trường hợp "chung chung"

```
virtual void draw() = 0;
```

- Lớp cơ sở trừu tượng
  - Là lớp trong đó có ít nhất một phương thức ảo thuần túy

```
class Shape
{
public:
  virtual void draw()=0;
  void paint();
};
```

  - Lớp cơ sở trừu tượng không có đối tượng!

Hảm hủy ảo

- Nếu hàm hủy của lớp không phải phương thức ảo thì sao?
- Ví dụ:

```
class DIEM {
public:
  DIEM();
  ~DIEM();
};
class DIEM_MAU: public DIEM {
public:
  DIEM_MAU();
  ~DIEM_MAU();
};
int main() {
  DIEM *p1, *p2;
  p1= new DIEM;
  p2= new DIEM_MAU;
  . . .
  delete p1;
  delete p2;
}
```

Ví dụ (hàm hủy - sai)

```
class DIEM
{
public:
  DIEM();
  ~DIEM();
};

class DIEM_MAU: public DIEM
{
public:
  DIEM_MAU();
  ~DIEM_MAU();
};

int main()
{
  DIEM *p1, *p2;
  p1= new DIEM;
  p2= new DIEM_MAU;
  . . .
  delete p1;
  delete p2;
}
```

Kết quả của delete p2 thế nào?

+ Do p2 là con trỏ kiểu DIEM nên khi hủy đối tượng (trỏ bởi p2) nó sẽ gọi hàm hủy của lớp DIEM. Như vậy khi hủy một đối tượng DIEM_MAU nhưng hàm hủy của DIEM_MAU lại không được gọi → chưa đạt yêu cầu.

+ Nếu trong thành phần của DIEM_MAU có con trỏ và được cấp bộ nhớ thì sau khi xóa đối tượng DIEM_MAU như thế sẽ tạo thành "rác" trong bộ nhớ.

+ Nếu hàm hủy là hàm ảo thì khi delete đối tượng của lớp nào thì hàm hủy của lớp đó được thực hiện.

Ví dụ (hàm hủy - đúng)

```
class DIEM {
public:
  DIEM();
  virtual ~DIEM();  // Hàm hủy ảo
};

class DIEM_MAU: public DIEM {
public:
  DIEM_MAU();
  ~DIEM_MAU();
};

int main()
{
  DIEM *p1, *p2;
  p1= new DIEM;
  p2= new DIEM_MAU;
  . . .
  delete p1;
  delete p2;
}
```

Kết quả của delete p2 thế nào?

+ p2 là con trỏ kiểu DIEM, nhưng p2 chứa địa chỉ của đối tượng DIEM_MAU, và vì hàm hủy là hàm ảo nên hàm hủy của DIEM_MAU được thực hiện.

+ Sau khi hàm hủy của DIEM_MAU thực hiện, nó sẽ tự động gọi hàm hủy của lớp cơ sở là DIEM

+ Kết quả: hủy thành công

Ví dụ (hàm hủy)

```
#include <iostream>
using namespace std;
class Base {
public:
 ~Base() { cout<<"~Base"<<endl; }
};
class Derived:public Base {
public:
  ~Derived() {cout<<"~Derived"<<endl; }
};
int main() {
  Base *B;
  B = new Derived;
  delete B;
}
```

```
~Base
```

Ví dụ (hàm hủy ảo)

```
#include <iostream>
using namespace std;
class Base {
public:
 virtual ~Base() { cout<<"~Base"<<endl; }
};
class Derived:public Base {
public:
  ~Derived() {cout<<"~Derived"<<endl; }
};
int main() {
  Base *B;
  B = new Derived;
  delete B;
}
```

```
~Derived
~Base
```

Tóm tắt

- Tương ứng bội (tính đa hình)
  - Ý nghĩa của tương ứng bội?
  - Phương thức ảo?
  - Liên kết sớm
  - Liên kết muộn?
  - Hàm hủy ảo

Bài tập (xác định kết quả)

```
class A  {
  int a;
public:
  A() {a = 5;}
  void xuat()  {cout<<a;}
};
class B: public A  {
  int a;
public:
  B() { a = 1;}
  void xuat() {cout<<a; }
};
void main()  {
  A *ob,x;
  B b;
  ob=&x; ob->xuat();
  ob=&b; ob->xuat();
}
```

5

5

```
class A  {
  int a;
public:
  A() {a=5;}
  virtual void xuat()  {cout<<a;}
};
class B: public A  {
  int a;
public:
  B() {a=1; }
  void xuat() {cout<<a; }
};
void main()  {
  A *ob,x;
  B b;
  ob=&x; ob->xuat();
  ob=&b; ob->xuat();
}
```

5

1

Bài tập

- Xác định kết quả chương trình sau:

```
class A     {
public:
    A() { cout << "A constructor\n"; }
    void m1() { cout << "A.m1\n"; m2();  }
    virtual void m2() { cout << "A.m2\n"; }
};
class B : public A     {
public:
    B() { cout << "B constructor\n";}
    void m1() { cout << "B.m1\n"; }
    void m2() { cout << "B.m2\n"; }
};
void func(A &a) { a.m1(); }
int main()   {
    B b;
    func(b);
}
```

A constructor
B constructor
A.m1
B.m2

Bài tập

```
// Instrument2.cpp
#include <iostream>
using namespace std;
enum note { middleC, Csharp, Eflat };
class Instrument {
public:
  void play(note) {
    cout << "Instrument::play" << endl;
  }
};
class Wind : public Instrument {
public:
  void play(note) {
    cout << "Wind::play" << endl;
  }
};
void tune(Instrument& i) {
  i.play(middleC);
}
int main() {
  Wind flute;
  tune(flute);
}
```

Instrument::play

```
// Instrument3.cpp
#include <iostream>
using namespace std;
enum note { middleC, Csharp, Eflat };
class Instrument {
public:
  virtual void play(note) {
    cout << "Instrument::play" << endl;
  }
};
class Wind : public Instrument {
public:
  void play(note) {
    cout << "Wind::play" << endl;
  }
};
void tune(Instrument& i) {
  i.play(middleC);
}
int main() {
  Wind flute;
  tune(flute);
}
```

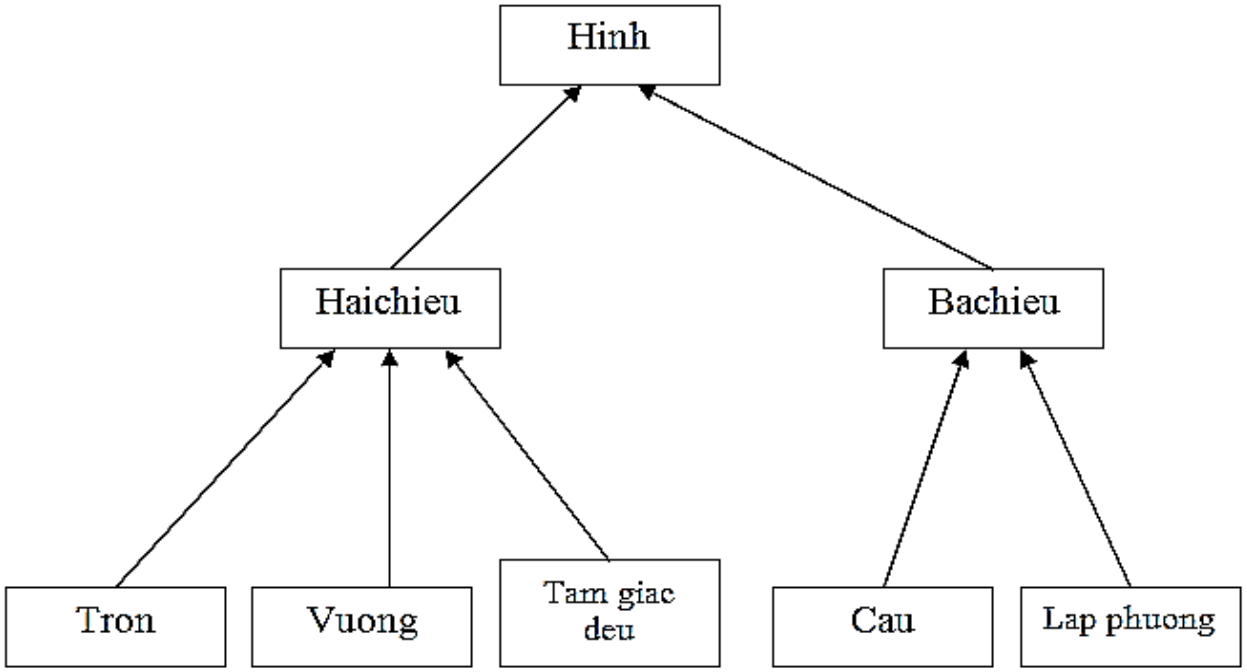
Wind::play

```
// Xét kết quả của chương trình sau:
#include <iostream>
using namespace std;
class Format {
     public:
          void display_form();
          virtual void header() { cout << "This is a header\n"; }
virtual void body() = 0; // hàm thuần ảo
virtual void footer() {     cout << "This is a footer\n\n"; }
};
void Format :: display_form() {
        header();
        for( int index = 0; index <3 ; index ++) { body(); }
footer();
}
class MyForm : public Format {
public:
void body() { cout << "This is the new body of text\n"; }
void footer() { cout << This is the new footer\n"; }
};
int main() {
Format* form = new MyForm;
form->display_form();
}
```

```
This is a header
This is the new body of text
This is the new body of text
This is the new body of text
This is the new footer
```

Bài tập

Xây dựng các lớp theo cây kế thừa sau:



Yêu cầu:

- Tất cả các loại hình đều có chung phương thức `ten()` để xác định tên hình là gì và phương thức `in()` để thể hiện tên hình
- Các hình hai chiều đều có phương thức `dt()` để tính diện tích
- Các hình ba chiều đều có phương thức `tt()` để tính thể tích

Nội dung chính

1. Giới thiệu môn học
2. Các khái niệm cơ bản
3. Hàm trong C++
4. Lớp và đối tượng
5. Định nghĩa chồng toán tử
6. Hàm tạo và hàm huỷ
7. Dẫn xuất và thừa kế
8. Tương ứng bội
9. Khuôn hình

ĐÃ XONG!

#include<bits/stdc++.h>

- Bao gồm tất cả các thư viện chuẩn
- **Ưu điểm**
  - Tiết kiệm thời gian viết code
  - Không phải nhớ tất cả các thư viện, nhất là STL (vector, list…)
- **Nhược điểm**
  - Không phải file tiêu đề C/C++ chuẩn, nên non-portable và có thể không dùng được với các compiler khác GCC (ví dụ: MSVC)
  - Sẽ include cả những thứ không cần thiết làm tăng thời gian compile

Một số kinh nghiệm lập trình C++

1. First make it work, then make it fast
2. Remember "divide and conquer" principle
3. When you create a class, make your names as clear as possible
4. Make classes as atomic as possible
5. Don't use C-style types and functions, even it is backward compatible
6. Don't write your own class templates unless you must
7. Don't repeat yourself
   - If a piece of code is recurring in many functions in derived classes, put that code into a single function in the base class and call it from the derived-class