第6讲 多态性与虚函数

6.1 多态性的概念
6.2 一个典型的例子
6.3 虚函数
6.4 纯虚函数与抽象类

6.1 多态性的概念

多态性(polymorphism, [pɔli’mɔ:fizəm])是面向对象程序设计的一个重要特征。

多态性: 向不同的对象发送同一个消息，不同的对象在接收时会产生不同的行为(即方法)。也就是说，每个对象可以用自己的方式去响应同一个消息。

多态性分为两类:

静态多态性和动态多态性。

静态多态性：通过函数的重载实现的(运算符重载实质上也是函数重载)多态性。

动态多态性：是在程序运行过程中才动态地确定操作所针对的对象，动态多态性是通过虚函数(virtual function)实现的。

6.2 一个典型的例子

例6.1　先建立一个Point(点)类，包含数据成员x,y(坐标点)。以它为基类，派生出一个Circle(圆)类，增加数据成员r(半径)，再以Circle类为直接基类，派生出一个Cylinder(圆柱体)类，再增加数据成员h(高)。要求编写程序，重载运算符“<<”和“>>”，使之能用于输出以上类对象。

步骤：

先声明基类，再声明派生类，逐级进行，分步调试。

(1) 声明基类Point类

```
#include <iostream.h>
class Point
{public:
  Point(float x=0,float y=0);//有默认参数的构造函数声明
void setPoint(float,float);                  //设置坐标值
  float getX( ) const {return x;}      //读x坐标，const常成员函数，不允许改变x
  float getY( ) const {return y;}     //读y坐标
  friend ostream & operator<<(ostream &,const Point &);//重载运算符“<<”
protected:                                   //受保护成员
  float x,y;
};
Point::Point(float a,float b) //Point的构造函数，对x,y初始化
{x=a;y=b;}
//设置x和y的坐标值
void Point::setPoint(float a,float b)        //为x,y赋新值
{x=a;y=b;}
//重载运算符“<<”，使之能输出点的坐标
ostream & operator<<(ostream &output,const Point &p)//返回ostream的引用
{output<<"["<<p.x<<","<<p.y<<"]"<<endl;
 return output;
}
```

对上面写的基类声明进行调试:

```
int main( )
{Point p(3.5,6.4);//建立Point类对象p
 cout<<"x="<<p.getX( )<<",y="<<p.getY( )<<endl;
//输出p的坐标值
 p.setPoint(8.5,6.8); //重新设置p的坐标值
 cout<<"p(new):"<<p<<endl; //重载运算符输出p点坐标
} //cout<<p相当于operator<<(cout,p)
```

运行结果为

```
x=3.5,y=6.4
p(new):[8.5,6.8]
```

测试程序检查了基类中各函数的功能，以及运算符重载的作用，证明程序是正确的。

(2) 声明派生类Circle

```
class Circle: public Point//circle是Point类的公用派生类
{public:
  Circle(float x=0,float y=0,float r=0); //构造函数
  void setRadius(float);                 //设置半径值
  float getRadius( ) const; //读取半径值，常成员函数声明和定义时都要有const
  float area ( ) const;                  //计算圆面积
  friend ostream & operator<< (ostream &,const Circle &);//重载运算符“<<”
 protected:
  float radius;
};
//定义构造函数，对圆心坐标和半径初始化
Circle::Circle(float a,float b,float r):Point(a,b),radius(r) { }
void Circle::setRadius(float r) {radius=r;}//设置半径值
float Circle::getRadius( ) const {return radius;} //读取半径值
float Circle::area( ) const {return 3.14159*radius*radius;} //计算圆面积
//重载运算符“<<”，使之按规定的形式输出圆的信息
ostream &    operator<<    (ostream &output,const Circle &c)
{output<<"Center=["<<c.x<<","<<c.y<<"],r="<<c.radius<<",
area="<<c.area( )<<endl;
 return output;
}
```

测试以上Circle类的定义,包括测试其中的成员函数:

```
int main( )
{Circle c(3.5,6.4,5.2);//建立Circle类对象c，并给定圆心坐标和半径
 cout<<"original circle:x="<<c.getX()<<", y="<<c.getY()<<", r="<<
c.getRadius( )<<", area="<<c.area( )<<endl;//输出圆心坐标、半径和面积
 c.setRadius(7.5);                  //设置半径值
 c.setPoint(5,5);                   //设置圆心坐标值x,y
 cout<<"new circle:\\n"<<c; //用重载运算符“<<”输出Circle对象的信息
 Point & pRef=c;                    //pRef是Point类的引用变量，被c初始化
 cout<<"pRef:"<<pRef; //用重载运算符输出Point类对象的引用pRef的信息
 return 0;
}
```

运行结果为original circle:x=3.5, y=6.4, r=5.2, area=84.9486

new circle:

Center=[5,5], r=7.5, area=176.714

pRef:[5,5]

(3) 声明Circle的派生类Cylinder

```
class Cylinder: public Circle// Cylinder是Circle的公用派生类
{public:
  Cylinder (float x=0,float y=0,float r=0,float h=0);//构造函数声明
  void setHeight(float);                    //设置圆柱高
  float getHeight( ) const;                 //读取圆柱高
  float area( ) const;                      //计算圆表面积
  float volume( ) const;                    //计算圆柱体积
  friend ostream& operator<<(ostream&,const Cylinder&);//重载运算符声明
 protected:
  float height;                             //圆柱高
};
Cylinder::Cylinder(float a,float b,float r,float h) :Circle(a,b,r),height(h) {}
void Cylinder::setHeight(float h){height=h;} //设置圆柱高
float Cylinder::getHeight( ) const {return height;} //读取圆柱高
float Cylinder::area( ) const{ return 2*Circle::area( )+2*3.14159*radius*height;}
float Cylinder::volume() const{return Circle::area()*height;}
ostream & operator<< (ostream &output,const Cylinder& cy)
{output<<"Center=["<<cy.x<<","<<cy.y<<"],r="<<cy.radius<<",h="<<cy.height<<"\\narea="<<cy.area( )<<", volume="<<cy.volume( )<<endl;
 return output;
}
```

cy1.area( )调用Cylinder的area函数。调用cy1基类Circle的area函数，应为: cy1.Circle::area( )。

测试:

```
int main( )
{Cylinder cy1(3.5,6.4,5.2,10);//定义Cylinder类对象cy1
 cout<<"\noriginal cylinder:\nx="<<cy1.getX( )<<", y="<<cy1.getY( )   <<", r="<<cy1.getRadius( )<<", h="<<cy1.getHeight( )<<"\\narea="   <<cy1.area()<<",volume="<<cy1.volume()<<endl;
cy1.setHeight(15);                  //设置圆柱高
cy1.setRadius(7.5);                 //设置圆半径
cy1.setPoint(5,5);                  //设置圆心坐标值x,y
 cout<<"\nnew cylinder:\\n"   <<cy1;   //用重载运算符“<<”输出cy1的数据
 Point &pRef=cy1;                    //pRef是Point类对象的引用变量
cout<<"\npRef as a Point:"<<pRef;     //pRef作为Point类对象的引用输出
 Circle &cRef=cy1;                   //cRef是Circle类对象的引用变量
 cout<<"\ncRef as a Circle:"<<cRef;   //cRef作为Circle类对象的引用输出
 return 0;
}
```

<<运算符函数的形参有一个类型是不同的，属函数重载，系统编译时可以自动判定应调用哪个重载函数。属静态多态性。

6.3 虚函数

6.3.1 虚函数的作用

虚函数的作用：

1、允许在派生类中重新定义与基类同名的函数。

2、通过基类指针或引用来访问基类和派生类中的同名函数。

cy1.area( )调用Cylinder的area函数。调用cy1基类Circle的area函数，应为: cy1.Circle::area( )。为使系统自动调用同一类族中不同类对象的同名函数，引入虚函数。

例6.2 基类与派生类中有同名函数display 。

```
#include <iostream>
#include <string>
using namespace std;
class Student//声明基类Student
{public:
   Student(int, string,float);//声明构造函数
   void display( ); //声明输出函数
  protected:  //受保护成员，派生类可以访问
   int num;
   string name;
   float score;
 };
//Student类成员函数的实现
Student::Student(int n, string nam,float s)                    //定义构造函数
 {num=n;name=nam;score=s;}
void Student::display( )                                       //定义输出函数
{cout<<"num:"<<num<<"\nname:"<<name<<"\nscore:"<<score<<"\n";}
```

```
class Graduate: public Student//声明公用派生类Graduate
{public:
   Graduate(int, string, float, float);                    //声明构造函数
   void display( );                                        //声明输出函数
private:
  float pay;
};
void Graduate::display( )                                  //定义输出函数
{cout<<"num:"<<num<<"\nname:"<<name<<"\nscore:"<<score<<"\npay="<
<pay<<endl;}
Graduate::Graduate(int n, string nam,float s,float p):Student(n,nam,s),pay(p){ }
int main()
 {Student stud1(1001,"Li",87.5);                  //定义Student类对象stud1
  Graduate grad1(2001,"Wang",98.5,563.5); //定义Graduate类对象grad1
  Student *pt=&stud1;                             //定义指向基类对象的指针变量pt
  pt->display( );
  pt=&grad1;
  pt->display( ); //注意：pt是指向基类对象的指针变量
  return 0;
 }
```

运行结果如下：
num:1001(stud1的数据)
name:Li
score:87.5

num:2001
name:wang
score:98.5
(仅输出了grad1中基类部分的数据，因为pt是指向基类对象的指针变量)

若在Student类中声明display函数时，改为：
virtual void display( );
这样就把Student类的display函数声明为虚函数。此时运行结果:
num:1001(stud1的数据)
name:Li
score:87.5
num:2001　　(grad1中基类部分的数据)
name:wang
score:98.5
pay=563.5　　(这一项改之前是没有的，但是我们希望有的)

由虚函数实现的动态多态性就是: 同一类族中不同类的对象，对同一函数调用作出不同的响应。

函数重载处理的是同一层次上的同名函数问题，而虚函数处理的是不同派生层次上的同名函数问题。 同一类族的虚函数的首部是相同的，而函数重载时函数的首部是不同的(参数个数或类型不同)。

虚函数的使用方法是:

(1) 在基类用virtual声明成员函数为虚函数，这样就可以在派生类中重新定义此函数，为它赋予新的功能，并能方便地被调用。在类外定义虚函数时，不必再加virtual。

(2) 在派生类中重新定义此函数，要求函数名、函数类型、函数参数个数和类型全部与基类的虚函数相同，并根据派生类的需要重新定义函数体。

(3) C++规定，当一个成员函数被声明为虚函数后，其派生类中的同名函数都自动成为虚函数，但习惯上一般在每一层声明该函数时都加virtual，使程序更加清晰。如果在派生类中没有对基类的虚函数重新定义，则派生类简单地继承其直接基类的虚函数。

(4) 定义一个指向基类对象的指针变量，并使它指向同一类族中需要调用该函数的对象。通过该指针变量调用此虚函数，此时调用的就是指针变量指向的对象的同名函数。

(5) 需要说明：有时在基类中定义的非虚函数会在派生类中被重新定义(如例6.1中的area函数)，如果用基类指针调用该成员函数，则系统会调用对象中基类部分的成员函数；如果用派生类指针调用该成员函数，则系统会调用派生类对象中的成员函数，这并不是多态性行为(使用的是不同类型的指针),没有用到虚函数的功能。

6.3.2 静态关联与动态关联

确定调用的具体对象的过程称为关联(binding，['baɪndɪŋ])。即指把一个函数名与一个类对象捆绑在一起，建立关联。

静态关联：在程序运行前进行关联的，如函数重载。

动态关联：在运行阶段把虚函数和类对象“绑定”在一起的。如在运行阶段，基类指针变量先指向了某一个类对象，然后通过此指针变量调用该对象中的函数。且指针可以先后指向不同的类对象，从而调用同一类族中不同类的虚函数。

6.3.3 在什么情况下应当声明虚函数

使用虚函数时，有两点要注意:

(1)只能用virtual声明类的成员函数，使它成为虚函数，而不能将类外的普通函数声明为虚函数。

(2) 一个成员函数被声明为虚函数后，在同一类族中的类就不能再定义一个非virtual的同名函数（参数个数、类型、函数返回值类型完全相同）。

根据什么考虑是否把一个成员函数声明为虚函数呢？

主要考虑以下几点：

(1) 成员函数所在的类如果作为基类，且在类的继承后有可能被更改功能，应该将它声明为虚函数。

(2) 成员函数的调用如果是通过基类指针或引用去访问的，应当声明为虚函数。

6.3.4 虚析构函数

例6.3 基类中有非虚析构函数时的执行情况。

```
#include <iostream>
using namespace std;
class Point//定义基类Point类
{public:
  Point( ){ }                                          //Point类构造函数
  ~Point(){cout<<"executing Point destructor"<<endl;}
//Point类析构函数
};
class Circle: public Point                             //定义派生类Circle类
{public:
  Circle( ){ }                                         //Circle类构造函数
  ~Circle( ){cout<<"executing Circle destructor"<<endl;}
//Circle类析构函数
 private:
  int radius;
};
```

```
int main( )
{ Point *p=new Circle; //用new建立临时对象， p是指向基类的指针变量
delete p; //用delete释放动态存储空间
return 0;
}
```

输出结果？？

运行结果：

```
executing Point destructor
```

表示只执行了基类Point的析构函数，而没有执行派生类Circle的析构函数。

如果希望能执行派生类Circle的析构函数，可以将基类的析构函数声明为虚析构函数，如

```
virtual ~Point(){cout<<"executing Point destructor"<<endl;}
```

程序其他部分不改动，再运行程序，结果为

```
executing Circle destructor
executing Point destructor
```

希望看到的结果：先调用派生类的析构函数，再调用基类的析构函数。

注意：

1、最好把基类的析构函数声明为虚函数。这将使所有派生类的析构函数自动成为虚函数。这样，如果程序中显式地用了delete运算符准备删除一个对象，而delete运算符的操作对象用了指向派生类对象的基类指针，则系统会调用相应类的析构函数。

2、专业人员一般都习惯声明虚析构函数，即使基类并不需要析构函数，也显式地定义一个函数体为空的虚析构函数，以保证在撤销动态分配空间时能得到正确的处理。

3、构造函数不能声明为虚函数。这是因为在执行构造函数时类对象还未完成建立过程，当然谈不上函数与类对象的绑定。

6.4 纯虚函数与抽象类

6.4.1 纯虚函数

在例6.1程序中的Point类中没有定义area函数。但是，在其直接派生类Circle和间接派生类Cylinder中都需要有area函数，而且这两个area函数的功能不同，一个是求圆面积，一个是求圆柱体表面积。

实现用同一种方式去调用同一类族中不同类的所有area函数。

可在基类Point中加一个area函数，并声明为虚函数：

```
virtual float area( ) const {return 0;}
```

其返回值为0，表示“点”是没有面积的。

为简化，可以不写出这种无意义的函数体，只给出函数的原型，并在后面加上“=0”，如

virtual float area( ) const =0;//纯虚函数

这就将area声明为一个纯虚函数(pure virtual function)。

声明纯虚函数的一般形式是

virtual 函数类型 函数名 (参数表列) =0;

注意：

①纯虚函数没有函数体；

②最后面的“=0”并不表示函数返回值为0，它只起形式上的作用，告诉编译系统“这是纯虚函数”；

③这是一个声明语句，最后应有分号。

纯虚函数的作用是在基类中为其派生类保留一个函数的名字，以便派生类根据需要对它进行定义。如果在基类中没有保留函数名字，则无法实现多态性。

6.4.2 抽象类

抽象类(abstract class)：不用来定义对象而只作为一种基本类型的类，用它作为基类建立派生类。由于它常用作基类，通常称为抽象基类(abstract base class)。

抽象类是相对于具体类(concrete，[‘kɔnkri:t] class)而言的。

注意：

1、凡是包含纯虚函数的类都是抽象类。因为纯虚函数是不能被调用的，包含纯虚函数的类是无法建立对象的。

2、如果在抽象类所派生出的新类中对基类的所有纯虚函数进行了定义，那么这些函数就被赋予了功能，可以被调用。这个派生类就不是抽象类，而是可以用来定义对象的具体类(concrete class)。

3、如果在派生类中没有对所有纯虚函数进行定义，则此派生类仍然是抽象类，不能用来定义对象。

6.4.3 应用实例

例6.4 虚函数和抽象基类的应用。

在本章例6.1介绍了以Point为基类的点—圆—圆柱体类的层次结构。现在要对它进行改写，在程序中使用虚函数和抽象基类。

类的层次结构的顶层是抽象基类Shape(形状)。

Point(点), Circle(圆), Cylinder(圆柱体)都是Shape类的直接派生类和间接派生类。

第(1)部分

```
#include <iostream.h>

class Shape//声明抽象基类Shape
{public:
 virtual float area( ) const {return 0.0;}//虚函数
 virtual float volume() const {return 0.0;} //虚函数
 virtual void shapeName() const =0; //纯虚函数
};
```

第(2)部分//声明Point类

```
class Point:public Shape//Point是Shape的公用派生类
{public:
  Point(float=0,float=0);
  void setPoint(float,float);
  float getX( ) const {return x;}
  float getY( ) const {return y;}
  virtual void shapeName( ) const {cout<<"Point:";}//对虚函数进行再定义
  friend ostream & operator<<(ostream &,const Point &);
protected:
  float x,y;
};
Point::Point(float a,float b) {x=a;y=b;}
void Point::setPoint(float a,float b){x=a;y=b;}
ostream & operator<<(ostream &output,const Point &p)
{output<<"["<<p.x<<","<<p.y<<"]";
return output;
}
```

第(3)部分//声明Circle类

```
class Circle: public Point
{public:
  Circle(float x=0,float y=0,float r=0);
  void setRadius(float);
  float getRadius( ) const;
  virtual float area( ) const;
  virtual void shapeName( ) const {cout<<"Circle:";}//对虚函数进行再定义
  friend ostream &operator<<(ostream &,const Circle &);
protected:
 float radius;
};
Circle::Circle(float a,float b,float r):Point(a,b),radius(r){ }
void Circle::setRadius(float r){radius=r; }
float Circle::getRadius( ) const {return radius;}
float Circle::area( ) const {return 3.14159*radius*radius;}
ostream &operator<<(ostream &output,const Circle &c)
{output<<"["<<c.x<<","<<c.y<<"], r="<<c.radius;
 return output;
}
```

```
第(4)部分//声明Cylinder类['silində]
class Cylinder: public Circle
{public:
  Cylinder (float x=0,float y=0,float r=0,float h=0);
  void setHeight(float);
  virtual float area( ) const;
  virtual float volume( ) const;
 virtual void shapeName( ) const {cout<<"Cylinder:";}//对虚函数进行再定义
  friend ostream& operator<<(ostream&,const Cylinder&);
 protected:
  float height;
};
Cylinder::Cylinder(float a,float b,float r,float h):Circle(a,b,r),height(h){ }
void Cylinder::setHeight(float h){height=h;}
float Cylinder::area( ) const{ return 2*Circle::area( )+2*3.14159*radius*height;}
float Cylinder::volume( ) const {return Circle::area( )*height;}
ostream &operator<<(ostream &output,const Cylinder& cy)
{output<<"["<<cy.x<<","<<cy.y<<"], r="<<cy.radius<<", h="<< cy.height;
return output;
}
```

```
第(5)部分//main函数
int main( )
{Point point(3.2,4.5);//建立Point类对象point
 Circle circle(2.4,1.2,5.6);                    //建立Circle类对象circle
 Cylinder cylinder(3.5,6.4,5.2,10.5);           //建立Cylinder类对象cylinder
 point.shapeName(); //通过对象名调用虚函数，静态关联
 cout<<point<<endl;
 circle.shapeName(); //通过对象名调用虚函数，静态关联
 cout<<circle<<endl;
 cylinder.shapeName(); //通过对象名调用虚函数，静态关联
 cout<<cylinder<<endl<<endl;
 Shape *pt;                                     //定义基类指针
 pt=&point;                                     //指针指向Point类对象
 pt->shapeName( );                              //动态关联
 cout<<"x="<<point.getX( )<<",y="<<point.getY( )<<"\\narea="<<pt-
>area( )<<"\\nvolume="<<pt->volume()<<"\\n\\n";
 pt=&circle;                                    //指针指向Circle类对象
 pt->shapeName( );                              //动态关联
 cout<<"x="<<circle.getX( )<<",y="<<circle.getY( )<<"\\narea="<<pt-
>area( )<<"\\nvolume="<<pt->volume( )<<"\\n\\n";
```

```
 pt=&cylinder;                          //指针指向Cylinder类对象
 pt->shapeName( );                      //动态关联
cout<<"x="<<cylinder.getX( )<<",y="<<cylinder.getY( )<<"\\narea="<<pt
->area( )<<"\\nvolume="<<pt->volume( )<<"\\n\\n";
 return 0;
}
```

程序运行结果如下:

```
Point:[3.2,4.5](Point类对象point的数据: 点的坐标)
Circle:[2.4,1.2], r=5.6          (Circle类对象circle的数据: 圆心和半径)
Cylinder:[3.5,6.4], r=5.5, h=10.5  (Cylinder类对象cylinder的数据:  圆心、
半径和高)
Point:x=3.2,y=4.5                  (输出Point类对象point的数据: 点的坐标)
area=0                          (点的面积)
volume=0                        (点的体积)
Circle:x=2.4,y=1.2                 (输出Circle类对象circle的数据: 圆心坐标)
area=98.5203                    (圆的面积)
volume=0                        (圆的体积)
Cylinder:x=3.5,y=6.4               (输出Cylinder类对象cylinder的数据: 圆心
坐标)
area=56.595                     (圆的面积)
volume=891.96                   (圆柱的体积)
```

小结:

(1) 一个基类如果包含一个或一个以上纯虚函数，就是抽象基类。抽象基类不能也不必要定义对象。

(2) 抽象基类可以没有任何物理上的或其他实际意义方面的含义。

(3) 在类的层次结构中，顶层或最上面的几层可以是抽象基类。

(4) 静态关联通过对象名来调用虚函数，动态关联通过基类指针来调用虚函数。

(5) 如果在基类声明了虚函数，则在派生类中凡是与该函数有相同的函数名、函数类型、参数个数和类型的函数，均为虚函数(不论在派生类中是否用virtual声明)。

(6) 利用虚函数和多态性，不论对象在继承层次中处于哪一层，都可以用基类指针来指向它，调用其中的函数。

例如：抽象基类Shape 的间接派生类圆Circle, 圆柱Cylinder，如何计算面积由类的设计者（专业人员）考虑，类的使用者（编程人员）可以用Shape类的指针来指向圆Circle和圆柱Cylinder的对象，调用其中的面积计算函数，不必考虑具体如何计算的问题。

- **作业：**
- **P225第5题**