Ch-12

多态性与虚函数

主要内容

- 多态性的概念
- 一个典型的例子
- 利用虚函数实现动态多态性
- 纯虚函数与抽象类

12.1 多态性的概念

在面向对象方法中的多态性是指：向不同的对象发送同一个消息，它们接收后会产生不同的行为（即方法）。

- 消息是指对类的成员函数的调用；
- 不同的行为是指不同的实现，也就是调用了不同的函数。

例如函数重载、运算符重载都是多态现象。

□ 从系统实现的观点看，多态性分为两类：静态多态性和动态多态性。

- 静态多态性（又称编译时的多态性）是指在编译时就可以确定调用哪个函数，它是通过函数重载实现的，例如函数重载和运算符重载都属于静态多态性。
- 动态多态性（又称运行时的多态性）是指在程序运行中才能确定操作所针对的对象，它是通过虚函数实现的。

12.2 一个典型的例子

见项目PJ_12.01相关文件

12.3 利用虚函数实现动态多态性

所谓虚函数，就是在基类声明的函数是虚拟的，并不是实际存在的函数，然后在派生类中才真正定义该函数。

虚函数声明的形式：

virtual 类型说明符 函数名(参数表)

12.3.1 虚函数的作用

虚函数的作用是在派生类中定义与基类函数同名的函数，通过基类指针或引用来访问基类或派生类中的同名函数。

□ 虚函数的使用方法（见例PJ_12.02）

（1）在基类用virtual声明成员函数为虚函数。在派生类中重新定义同名函数，让它具有新的功能；

（2）在派生类中重新定义此函数时，要求函数名、函数类型、参数个数和类型与基类的虚函数相同。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自动成为虚函数；

（3）定义一个指向基类对象的指针变量，并让它获得同一类族中某个对象的地址；

（4）用该指针变量调用虚函数，调用的就是该对象所属类的虚函数。

说明：如果一个基类有多个派生类，而派生类又产生新的派生类，这样就形成了同一基类的类族。

12.3.2 静态关联与动态关联

在编译时即可确定其调用的函数（或虚函数）属于哪个类，这个过程称为静态关联。由于是在运行前进行的关联，因此又称为早期关联（**early binding**）。例如函数重载和用对象名调用虚成员函数。

在运行中把虚函数和类对象绑定在一起，此过程称为动态关联，这多态性是动态的多态性，即运行阶段的多态性。在运行阶段，基类指针变量先指向了某个类对象，然后通过此指针变量调用该对象中的函数。此时调用哪个对象的函数是确定的。

在运行阶段，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。

由于动态关联是在编译之后的运行阶段进行的，因此又称为滞后关联（**late binding**）

12.3.3 在什么情况下应该声明虚函数

- 使用虚函数，有两点要注意：

（1）只能用virtual声明类的成员函数把它作为虚函数，而不能讲类外的普通函数声明为虚函数，因为虚函数的作用就是允许在派生类中对基类的虚函数重新定义；

（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的但与该虚函数具有相同的参数（个数和类型）和函数类型的同名函数。

口 什么情况下应该声明虚函数

（1）首先看成员函数所在的类是否会作为基类，然后看成员函数在类的继承后有无可能修改功能，如果需要修改功能，一般应该将它声明为虚函数；

（2）如果成员函数在类被继承后功能不需要修改，或派生类用不到函数，则不要把它声明为虚函数。不要仅仅考虑到要作为基类而把本类中的所有成员函数都声明为虚函数。

（3）应考虑对成员函数的调用是通过对象名还是通过基类指针或引用去访问，如果是通过基类指针或引用去访问的，则应当声明为虚函数；

（4）有时，在定义虚函数时，并不定义其函数体，即函数体是空的。它的作用只是定义了一个函数名，具体功能留给派生类去实现。

◆ 当一个类带有虚函数时，编译系统会为该类创建一个虚函数表（virtual function table，简称vtable），它是一个指针数组，存放每个虚函数的入口地址。同时编译系统会自动在类中添加一个成员数据vptr，它是一个指向虚函数表的指针。

12.3.4 虚析构函数

例12.3 基类中有非虚构函数时的执行情况

```
4   class Point
5   {
6       public:
7           Point() {}
8           ~Point() {
9               cout<<"executing Point destructor"<<endl;
10          }
11  };
12
13  class Circle:public Point
14  {
15      public:
16          Circle() {}
17          ~Circle() {
18              cout<<"executing Circle destructor"<<endl;
19          }
20      private:
21          int radus;
22  };
23  int main()
24  {
25      Point *p=new Circle;
26      delete p;
27      return 0;
28  }
```

□当派生类的对象撤销时一般先调用派生类的析构函数，然后调用基类的析构函数。但是如果用new运算符建立一个动态的派生类对象，并将指向该派生类对象的指针赋给一个基类的指针变量。这样，若基类中有析构函数，则在程序中用delete运算符撤销该派生类对象时，系统只会执行基类的析构函数，而不执行派生类的析构函数。

```
executing Point destructor
```

□ 若将基类的析构函数声明为虚函数，如

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

程序其他部分不变，再运行程序，结果为

```
executing Circle destructor

executing Point destructor
```

当基类的析构函数为虚函数时无论指针指的是同一类族中的哪一个类对象，撤销对象时，编译系统都会采用动态关联，调用相应的析构函数。

若将基类的析构函数声明为虚函数，则所有派生类的析构函数都将自动成为虚函数。

◆ 构造函数不能声明为虚函数。

12.4 纯虚函数与抽象类

12.4.1 纯虚函数

在基类中不能为虚函数给出一个有意义的实现时，可将其声明为纯虚函数，其实现留待派生类完成。

纯虚函数为派生类提供一个一致的接口；

□ 纯虚函数声明的一般形式

```
class 类名 {
    virtual 类型 函数名(参数表) = 0;
    ……
}
```

12.4.2 抽象类

□ 定义

带有纯虚函数的类称为抽象类。

□ 抽象类的主要作用

通过它为一个类族建立一个公共的接口，使它们能够更有效地发挥多态特性。

一般而言，抽象类只描述一组子类共同操作接口，而完整的实现留给子类。

□ 说明

- 抽象类是一个特殊的类，是为了抽象和设计的目的而建立的，它处于继承层次的结构的较上层，即只能用作其他类的基类，抽象类是不能定义对象的；
- 从一个抽象类派生的类必须提供纯虚函数的实现代码或在该派生类中仍将它说明为纯虚函数，否则编译错误；
- 抽象类不能用作参数类型、函数返回类型或显式转换的类型，但可以说明指向抽象类的指针和引用，此指针可以指向它的派生类，实现多态性。

□ 抽象类的说明

- 抽象类是一个特殊的类，是为了抽象和设计的目的而建立的，它处于继承层次的结构的较上层，即只能用作其他类的基类，抽象类是不能定义对象的；
- 从一个抽象类派生的类必须提供纯虚函数的实现代码或在该派生类中仍将它说明为纯虚函数，否则编译错误；
- 抽象类不能用作参数类型、函数返回类型或显式转换的类型，但可以说明指向抽象类的指针和引用，此指针可以指向它的派生类，实现多态性；
- 构造函数不能是虚函数，析构函数可以是虚函数。

CH12 End