第 5 章

类 和 对 象

封装性(Encapsulation)是面向对象程序设计的重要特性之一,主要通过类和对象体现。类是 C++ 语言的核心概念,其本质是一种数据类型;而对象是某一种类的实例,因此类和对象密切相关。通过类的封装性,可以实现数据的隐藏性,便于程序的维护和修改。本章主要介绍类的定义以及类和对象的应用。

学习目标:

(1) 掌握类和对象的定义,掌握对象的初始化方法,了解成员函数的特性;

(2) 掌握静态成员的概念和使用方法,了解常类型的概念和使用方法;

(3) 掌握对象指针、对象引用以及对象数组的使用方法,了解子对象和堆对象的基本概念;

(4) 掌握友元函数和友元类的概念和使用方法;

(5) 了解类的作用域和对象的生存期的概念。

5.1 类的定义

类是一种用户自定义的数据类型,是对具有共同属性和行为的一类事物的抽象描述,共同属性被描述为类中的数据成员,共同行为被描述为类中的成员函数。

5.1.1 类的定义格式

类的定义格式一般分为声明和实现两部分。声明部分用来声明类中的成员,包括数据成员和成员函数的声明,其中数据成员又称为属性,成员函数又称为方法或操作,用来对数据进行操作。数据成员的声明包含数据成员的名字和类型;成员函数可以在类体内定义,也可以在类体内只对其进行声明,而在类体外实现部分进行定义。如果一个类中所有的成员函数都是在类体内定义,则该类就没有类外实现部分。在类体内定义的成员函

数为内联函数。具体定义格式如下：

```
class  <类名>                                          //声明部分
{
    public:
          <公有数据成员和成员函数的声明或实现>
    protected:
          <保护数据成员和成员函数的声明或实现>
    private:
          <私有数据成员和成员函数的声明或实现>
};
<函数类型>  <类名>::<成员函数名>  (<参数表>)    //实现部分
{
    <函数体>
}
```

其中，class 为关键字；＜类名＞为标识符；花括号内是类的声明部分，对该类的成员进行声明。类成员包含数据成员和函数成员，数据成员通常是变量或对象的声明语句，而函数成员指函数的定义或函数声明语句。

public：关键字，说明其后的成员为公有成员。通常将类的成员函数全部或部分定义为公有成员。

private：关键字，可默认，说明其后的成员为私有成员。通常将类的数据成员定义为私有成员，是被隐藏的部分。

protected：关键字，说明其后的成员为保护成员。

关键字 public、private 和 protected 称为访问权限控制符或访问权限修饰符，它们在类体中出现的顺序和次数没有限制，用来说明类成员的访问权限。

【例 5.1】　定义一个水果类。

```
class Fruit
{
    public:                                  //声明公有的成员函数
        void SetFruitNumber(int num){fruitNumber=num;}
        void SetFruitName(string name){fruitName=name;}
        void SetPurchasePrice(double price){purchasePrice=price;}
        int GetFruitNumber(){return fruitNumber;}
        string GetFruitName(){return fruitName;}
```

```
        double GetPrice(){return purchasePrice;}
        void DispFruitName(){cout<<"水果名称:"<<fruitName;}
        void DispPurchasePrice(){cout<<"进价为:"<<purchasePrice;}
        void DispFruitNumber(){cout<<"水果编号"<<fruitNumber;}
        void DispFruit();
    private:                                    //声明私有的数据成员
        int fruitNumber;
        string fruitName;
        double purchasePrice;
};
void Fruit::DispFruit()
{
    cout<<"水果编号:"<<fruitNumber<<", 名称:"<<fruitName;
}
```

该类有 13 个成员,包括 3 个私有的数据成员和 10 个公有的成员函数。其中成员函数 DispFruit()在类体内声明,在类体外定义,用于显示数据成员的值;而其他几个成员函数的定义都在类体内给出,用于给数据成员赋值、获取数据成员的值并分别显示数据成员的值。3 个数据成员分别表示水果的编号、名称和进货价格。

【例 5.2】 定义一个类,描述平面上的一个点。

```
class Point
{
    public:                                     //声明公有的成员函数
        void SetValue(int x,int y);
        int GetX(){return X;}
        int GetY(){return Y;}
        void Move(int x,int y);
    private:                                    //声明私有的数据成员 X 和 Y
        int X,Y;
};
void Point::SetValue(int x,int y)       //成员函数 SetValue()在类体外的实现
{
    X=x;Y=y;
}
```

```
void Point::Move(int x,int y)              //成员函数 Move()在类体外的实现
{
    X+=x;Y+=y;
}
```

该类的两个数据成员 X 和 Y 是私有成员，分别表示平面上点的横坐标和纵坐标；该类定义了 4 个公有的成员函数，其中 SetValue()用于给数据成员 X 和 Y 赋值，GetX()和 GetY()用于返回数据成员 X 和 Y 的值，Move()用来改变点的坐标值 X 和 Y。

成员函数的定义既可以在类体内，也可以在类体外的实现部分。在类体外定义时，必须使用“::”符号说明该函数属于哪个类，该符号称为作用域运算符。如上例中的 SetValue()和 Move()两个成员函数在类体外的定义。

定义类时需要注意以下几点：

(1) 在类体内对数据成员定义时，不允许初始化。

(2) 类中数据成员可以是任意类型的数据：整型、浮点型、字符型、数组、指针和引用等，也可以是对象，但是只能是其他类的对象，而不允许是自身类的对象。

(3) 当在类体外定义成员函数时，必须在类体内有函数原型声明语句。

(4) 一般情况下，在类体内先声明公有成员，后声明私有成员；在声明数据成员时，一般按数据成员的类型大小，由小到大进行声明，这样可以提高时空利用率。

(5) 通常将类的定义部分存放在一个用户自定义的头文件中，方便程序使用，例如可将上例中点类的定义存放在“point.h”文件中。

5.1.2　类成员的访问控制

类成员访问权限的控制，体现了类的隐藏性和封装性，是通过设置类成员的访问控制属性来实现的。访问控制属性有 3 种：public(公有)、private(私有)和 protected(保护)。

公有属性(public)定义了类的外部接口，对类而言，任何外部的访问都要通过外部接口来进行。公有成员既可以在类体内引用，也可以在类体外引用，是类与外界的接口，即公有的成员函数或数据成员可以被程序中的任何函数或语句访问(静态成员除外)。

私有属性(private)的成员不允许类外部的程序代码对其进行直接访问，只能被本类的成员函数或特殊说明的函数(友元)引用，保证了私有成员被隐藏在类中，对外界是不可见的。外部程序代码要访问私有成员，只能通过类的公有成员函数或友元函数，这样就实现了访问控制。

保护属性(protected)与私有属性相似，对于本类相当于私有成员，对于该类的派生类相当于公有成员，即保护的数据成员和成员函数不能被类外程序代码访问，只有类的成员

函数或友元和该类的派生类才可以访问(类的继承和派生在第 6 章中会详细讲述)。

5.1.3 成员函数的特性

成员函数具有一般函数的一些特性,可以重载,可以定义为内联函数,还可设置函数形参的默认值。

1. 成员函数可以重载

一般成员函数可以重载,构造函数也可以重载,但是析构函数不可以重载。

【例 5.3】 重载成员函数声明。

```
class Point
{
    public:                                  //声明公有的成员函数
        void SetValue(int x,int y);
        int GetX(){return X;}
        int GetY(){return Y;}
        void Move(int x,int y);              //重载的成员函数 Move(两个参数)
        void Move(int x);                    //重载的成员函数 Move(一个参数)
    private:                                 //声明私有的数据成员 X 和 Y
        int X,Y;
};
void Point::SetValue(int x,int y)            //成员函数 SetValue()在类体外的实现
{
    X=x;Y=y;
}
void Point::Move(int x,int y)                //成员函数 Move(int,int)在类体外的实现
{
    X+=x;Y+=y;
}
void Point::Move(int x)                      //成员函数 Move(int)在类体外的实现
{
    X+=x;Y+=10;
}
```

该类定义了两个同名的成员函数 Move(int)和 Move(int,int),二者互为重载。

2. 成员函数可以声明为内联函数

成员函数如果在类体内定义，默认为内联函数，而在类体外定义则不是内联函数。如果要使在类体外定义的函数也成为内联函数，则需定义时在函数头前加上关键字 inline。

【例 5.4】 内联函数的定义。

```
class Point
{
    public:
        void SetValue(int x,int y);
        int GetX(){return X;}                    //成员函数 GetX()为内联函数
        int GetY(){return Y;}                    //成员函数 GetY()为内联函数
        void Move(int x,int y);
    private:
        int X,Y;
};
void Point::SetValue(int x,int y)
{
    X=x;Y=y;
}
inline void Point::Move(int x,int y)             //成员函数 Move()定义为内联函数
{
    X+=x;Y+=y;
}
```

该类中声明了 4 个成员函数，其中 GetX()、GetY()和 Move()为内联函数。

3. 成员函数的参数可以设置默认值

成员函数的参数可以设置为默认值，使用方法与一般函数的参数默认值相同。

【例 5.5】 成员函数的参数默认值。

```
class Point
{
    public:
        void SetValue(int x,int y);
        int GetX(){return X;}
        int GetY(){return Y;}
```

```
        void Move(int x,int y=10);          //设置成员函数 Move 的参数 y 的默认值
    private:
        int X,Y;
};
void Point::SetValue(int x,int y)
{
    X=x;Y=y;
}
void Point::Move(int x,int y)               //成员函数 Move()在类体外的实现
{
    X+=x;Y+=y;
}
```

该类中的成员函数 Move()设置了参数的默认值，具体应用时需根据对象的调用情况来确定默认值的使用。

5.2 对象的定义和使用

在 C++ 程序运行时，类是以其实例——对象之间的操作和相互访问来完成程序功能的。

5.2.1 对象的定义方法

类是一种类型，不占用内存空间；对象是类的实例，是实体，定义对象时，系统会分配相应的存储空间。任何一个对象都属于某个类，对象包含其所属类的所有成员，因此在定义对象前，必须先定义类。

对象中只存放数据成员，成员函数不存放在每个对象中，而是存放在一个可被对象访问的公共区域中。

对象的定义格式如下：

```
<类名>  <对象名表>;
```

其中，<类名>是对象所属的类的名字；<对象名表>可以包含多个对象，对象名之间用逗号间隔。对象名可以是一般对象名，也可以是指向对象的指针名或引用名，还可以是对象数组名。

例如：

```
Point p1,p2,*pp,&rp=p1,p[10];
```

其中，p1，p2 是一般对象名；pp 是指向 Point 类对象的指针；rp 是对象 p1 的引用；p 是一维数组，有 10 个元素，每个元素是类 Point 的对象。

5.2.2　对象成员的表示方法

一个对象的成员就是该对象所属类的成员：数据成员和成员函数。

1. 一般对象的成员表示

```
<对象名>.<数据成员名>
<对象名>.<成员函数名>(<参数表>)
```

这里，“.”是成员访问运算符，其功能是表示对象的成员，例如：

```
Point p1;
```

则 p1.X，p1.Y，p1.SetValue(10,5)等即为对象的成员。

2. 指针所指向的对象的成员表示

```
<对象指针名>->数据成员名>
<对象名>->成员函数名>(<参数表>)
```

其中，“－＞”也是一个表示成员的运算符，用于表示指针所指向的对象的成员，例如：

```
Point p1,*pp=&p1;
```

则 pp－＞X，pp－＞Y，pp－＞SetValue(10,5)，pp－＞Move(2,2)等即为 pp 所指向的对象 p1 的成员。

3. 对象引用的成员表示

```
<对象引用名>.<数据成员名>
<对象引用名>.<成员函数名>(<参数表>)
```

例如：

```
Point p1,&rp=p1;
```

则 rp. X,rp. Y,rp. SetValue (10,5)等是 rp 所引用的对象 p1 的成员。

4. 对象数组元素的成员表示

```
<数组名>[<下标>].<数据成员名>
<数组名>[<下标>].<成员函数名>(<参数表>)
```

【例 5.6】 分析下面程序的运行结果，熟悉对象的定义和对象成员的表示方法。

```
#include <iostream>
using namespace std;
class Point                                    //定义平面上的点类型 Point
{
    public:                                    //声明公有成员函数
        void SetValue (int x,int y);           //为数据成员(坐标)赋值
        int GetX(){return X;}                  //取 x 坐标
        int GetY(){return Y;}                  //取 y 坐标
        void Move(int x,int y);                //移动点(改变坐标值)
    private:
        int X,Y;                               //私有数据成员:点的坐标
};
void Point:: SetValue(int x,int y)
{
    X=x;Y=y;
}
void Point::Move(int x,int y)
{
    X+=x;Y+=y;
}
int main()
{
    Point p1,p2;
    p1.SetValue(10,20);
```

```
    p2.SetValue(5,6);
    p1.Move(5,10);
    p2.Move(3,4);
    cout<<"x1="<<p1.GetX()<<",y1="<<p1.GetY()<<endl;
    cout<<"x2="<<p2.GetX()<<",y2="<<p2.GetY()<<endl;
    return 0;
}
```

程序的运行结果：

```
x1=15,y1=30
x2=8,y2=10
```

该程序的主函数中，定义了两个点对象 p1 和 p2，并通过成员函数 SetValue()对其赋值，设置点的坐标，通过成员函数 Move()对其进行移动，改变点的坐标，最后输出两个点的坐标。

5.3　构造函数和析构函数

C++ 语言在创建对象时，系统会自动调用相应的构造函数对所创建的对象进行初始化，即完成对象的数据成员初始化的过程。当一个对象的生存期结束时，系统又会自动调用析构函数释放这个对象。

5.3.1　构造函数

构造函数是一种特殊的成员函数，其功能是对对象进行初始化。

1. 构造函数的特点

构造函数是一种成员函数，它既具有一般成员函数的特性，又具有自身的特点。

(1) 构造函数的名字与类名相同。

(2) 构造函数没有返回值，不允许定义构造函数的返回值类型，包括 void 类型。

(3) 构造函数可以有一个或多个参数，也可以无参数，分别称为有参构造函数和默认构造函数。

(4) 构造函数可以重载，重载构造函数由其参数个数、参数类型来区分。

(5) 程序中一般不直接调用构造函数，而是由系统在创建对象时自动调用执行。