第5章 面向对象程序设计基础

前4章介绍了C#语法和编程的所有基础知识，现在已经可以编写和调试控制台应用程序了。但是，要了解C#语言和.NET Framework的强大功能，还需要使用面向对象编程技术。

本章主要介绍面向对象技术、类和对象的创建、类的成员、静态类和静态成员的基本知识以及Visual Studio 2010中的面向对象编程工具。

5.1 面向对象的概念

5.1.1 面向对象编程

面向对象程序设计(Object Oriented Programming，OOP)是创建计算机应用程序的一种新的方法，它解决了传统编程技术带来的许多问题。

传统的面向过程程序设计求解问题的基本策略是功能分解，它将整个系统看做一个大的处理过程，然后将其分解为若干个易于处理的子过程。在分析过程中，用数据描述各子过程之间的联系，整理各个子过程的执行顺序。这种方法缺乏对问题的基本组成对象的分析，不够完备，尤其是当功能需求发生变化时，将导致大量修改，不易维护。

面向对象程序设计的基本思想是从要解决的问题本身出发，尽可能运用人类的思维方式，以现实世界中的事物为中心思考问题、认识问题，使得软件开发的方法与过程尽可能接近人类认识世界、解决问题的方法与过程。面向对象技术以对象为基本单位，将数据和操作封装在对象内部，不受外界干扰。它使得一个复杂的软件系统可以通过定义一组相对独立的模块来实现，这些独立模块彼此之间只需交换那些为了完成系统功能所必须交换的信息。当模块内部实现发生变化而导致代码修改时，只要对外接口操作的功能不变，就不会给软件系统带来影响，因此提高了软件的可维护性，也增加了重用代码的机会。

举例来说，假定计算机上的一个高性能应用程序是一辆赛车。如果使用传统的编程方法，这辆赛车就是一个单元。如果要改进该车，就必须把它送回厂商那里，让汽车专家升级它，或者购买一辆新车。如果使用OOP技术，就只需从厂商处购买新的引擎，自己按照说明替换它。

5.1.2 类和对象

在客观世界中，每一个有明确意义和边界的事物都可以看做一个对象，它是一个可以辨识的实体。对象充满着整个世界，任何具体的事物都是一个对象。例如，日常生活中人们要与不同的对象打交道，人们坐的公交车是对象，用的计算机是对象，看的电视是对象。每个对象都有其状态和行为，以区别于其他对象。例如，一台电视有型号、尺寸、生产厂家等状态，也有开机、换台等行为。可以把具有相似特征的事物归为一类，例如，所有的电视机可以归为“电视机类”。

在面向对象的程序设计中，对象的概念就是对现实世界中对象的模型化，它同样有自己的状态和行为，对象的状态用数据来表示，称为属性；对象的行为用代码来表示，称为方法。而类则是对具有相同属性和方法的一组相似对象的描述。从另一个角度来看，对象就是类的一个实例。

5.1.3 面向对象的特点

面向对象的最基本特征是封装性、继承性和多态性。

1. 封装性

封装性来源于黑盒的概念。根据黑盒的概念，人们无需懂得对象的工作原理和内部结构，就可以使用日常生活中的许多对象。例如，电视机的内部结构很复杂，使用它时，只需知道如何操作几个基本按钮即可。

在 OOP 中，把对象的数据和代码组合在同一个结构中，就是对象的封装性。将对象的数据封装在对象内部，外部程序必须而且只能使用正确的方法才能访问要读写的数据。封装的目的在于将对象的使用者与设计者分开，使用者不必了解对象方法的具体实现，只需要用设计者提供的消息接口来访问该对象。

2. 继承性

继承性是指特殊类的对象拥有一般类的属性和方法。其中，一般类称为基类或父类，特殊类称为派生类或子类。继承的好处是共享代码，继承后，父类的所有属性和方法都将存在于子类中。

如图 5-1 所示，汽车类包含轿车类、客车类和卡车类，那么汽车类就是一般类，具有的属性包括重量、车轮、车门等，具有的行为包括启动、行驶和鸣笛等。客车类作为特殊的汽车类，除了继承汽车类的所有属性和行为外，还具有一些特殊的行为，例如载客。

3. 多态性

同一操作作用于同一类型的不同对象时，可以有不同的解释，产生不同的执行结果，这就是多态性。换句话说，同一个类型的实例调用“相同”的方法，产生的结果是不同的。

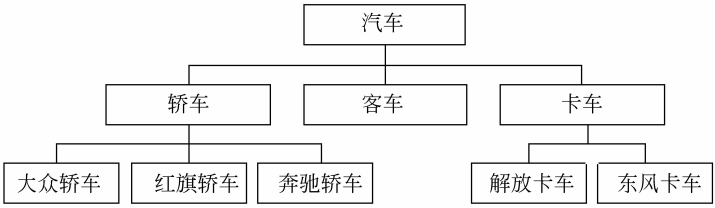

图 5-1　汽车类的继承关系

在 C# 语言中，多态是通过基类引用指向派生类对象，调用其虚方法实现的。

5.2　类 的 声 明

在面向对象程序设计中，所使用的每一个对象都是通过类来创建的，因此创建对象之前必须先创建类。在 C# 语言中，类是一种数据类型，使用前必须先声明，格式如下：

```
[访问修饰符] class 类名[:基类]
{
    类的成员
}
```

说明：

(1) 访问修饰符用来限制类的作用范围或访问级别，可省略。

默认情况下，类声明为内部的，即只能在当前项目中访问，也可以用 internal 访问修饰符显式指定。如果在其他项目中要访问某个类，必须用 public 访问修饰符，将其指定为公共类。

(2) 类名表示所定义的类的名称，必须符合 C# 标识符的命名规则。

(3) ":基类"表示所定义的类是一个派生类，可省略，默认继承于 System.Object。

(4) 类的成员是构成类的主体，用来定义类的数据和行为。

【例 5-1】 声明 Circle 类。

```
class Circle
{
    //类成员
}
```

5.3　类 的 成 员

例 5-1 中的类定义中只给出了一个空的框架，没有定义其成员。

类的成员可以分为两大类：一是类本身所声明的，二是从基类中继承来的。类的成

员包含常量、字段、属性、方法、索引器、事件、构造函数和析构函数。

类的每个成员都需要指定访问修饰符，不同的修饰符会造成对成员访问能力不一样。如果没有显式指定类成员的访问修饰符，则默认为 private。C＃中类成员的访问修饰符及其含义如表 5-1 所示。

表 5-1　类成员的访问修饰符及其含义

访问修饰符	含　义
public	表示公共成员，访问不受限制
private	表示私有成员，访问仅限于该类
internal	表示内部成员，访问仅限于当前项目
protected	表示受保护的成员，访问仅限于该类及其派生类
protected internal	表示受保护的内部成员，访问仅限于该类或当前项目的派生类

【例 5-2】 成员访问修饰符。

如图 5-2 所示，i、j、k、l、m 是类 A 的成员，均能被类 A 访问。除此之外，成员 j 用 public 修饰，是一公共成员，类 B、C、D、E 都可以访问；成员 k 用 internal 修饰，是一内部成员，能被当前项目中的类 B、C 访问；成员 l 用 protected 修饰，是一受保护成员，能被类 A 的派生类 C、D 访问；成员 m 用 protected internal 修饰，是一受保护的内部成员，能被类 A 的派生类 C、D 及当前项目中的类 B 访问。

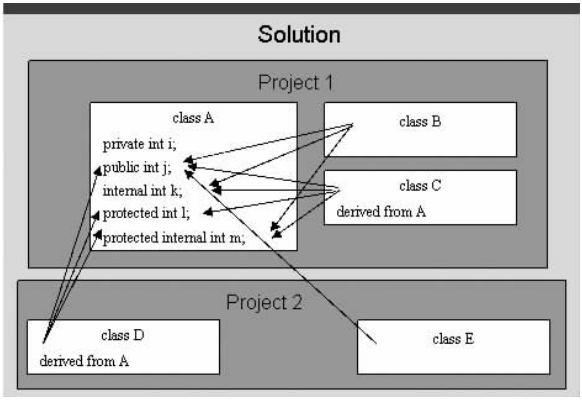

图 5-2　成员访问修饰符

5.3.1　常量

类的常量成员是一种符号常量，这种符号常量在声明时必须进行初始化，而程序运行时不能再对它的值进行更改。其声明方法与第 2 章介绍的符号常量的声明方法基本相同，格式如下：

```
[访问修饰符] const 数据类型 常量名=常量的值
```

【例 5-3】 完善 Circle 类，增加常量成员 pi，表示圆周率 π。

```
class Circle
{
    const float pi=3.14f;
}
```

5.3.2 字段

字段是在类范围声明的变量，用于保存类或对象的状态数据。声明字段的方法与声明普通变量的方法基本相同，格式如下：

[访问修饰符] 数据类型 字段名

【例 5-4】 完善 Circle 类，增加字段成员 radius，表示圆的半径。

```
class Circle
{
    const float pi=3.14f;
    private float radius;
}
```

声明字段时，可以使用赋值运算符为字段指定一个初始值，字段的初始化恰好在调用对象实例的构造函数(详见 5.3.5 节)之前。如果构造函数为字段分配了值，则该值将覆盖字段声明期间给出的任何值。

另外，可以使用 readonly 关键字声明只读字段，其值可以在定义时设定，也可以在构造函数中设定，但不能在其他地方给只读字段赋值。只读字段类似于常量，但比常量灵活得多，二者的区别如表 5-2 所示。

表 5-2 常量与只读字段的区别

	常 量	只 读 字 段
类型限制	必须为基元类型(object 除外)，如 int、string	没有限制，可以是任意类型
赋值	声明时必须为常量赋值	可以在声明时赋值，也可以在类的构造函数中赋值，但不允许在其他地方赋值
内存消耗	直接被编译到目标代码的元数据中，不占内存	占用内存
对象的值	对于所有对象，常量的值是一样的	对于不同的对象，只读字段的值可以不同
调用方式	类.常量	对象.只读字段(如果没有显式声明为静态字段)

5.3.3 属性

字段一般定义为私有或受保护的，不允许外界访问。如果外界需要访问，可以使用类的另一个成员——属性。属性是对类的字段提供特定访问的类成员。由于属性是类的成

员，因此可以访问私有变量。同时，因为属性在类中是按一种与方法类似的方式执行的，因此它不会危害类中私有变量保护和隐藏的数据。

在类中定义属性的格式如下：

```
[访问修饰符] 数据类型 属性名
{
    get
    {
        return 字段名;
    }
    set
    {
        字段名=value;
    }
}
```

其中，get 语句与 set 语句称为属性的访问器。get 访问器用于获取属性值，而 set 访问器用于修改属性值，它们可以有不同的访问级别。set 访问器有一个隐式参数 value，系统通过它将外界的数据传递进来，然后通过赋值运算更新字段值。

在声明属性之前，通常先声明一个私有变量。微软推荐属性和私有变量使用相同的名称，只是该变量名称和属性名称的第一个字母大小写不同。

【例 5-5】 完善 Circle 类，增加属性成员 Radius。

```
class Circle
{
    const float pi=3.14f;
    private float radius;
    public float Radius          //声明公共属性 Radius 以访问私有字段 radius
    {
        get
        {
            return radius;
        }
        set
        {
            if(value<0)
                radius=0;
            else
                radius=value;
        }
    }
}
```

此例中的 Radius 属性同时包含 get 和 set 访问器，是读/写属性。而只包含 get 访问

器的是只读属性，只包含 set 访问器的是只写属性。

【例 5-6】 银行软件系统为一般柜台操作员声明了一个类 Account。在该类中，操作员可以读取银行规定的利率，而对于客户的每笔新存款，操作员只有写入的权限。

```
class Account
{
    private float rate;              //利率
    private float deposit;           //存款
    public float Rate                //只读属性
    {
        get {return rate;}
    }
    public float Deposit             //只写属性
    {
        set {deposit=deposit+value;}
    }
}
```

说明：get 访问器返回某个字段的值时，可以先对这个字段进行计算，再返回结果，但注意不要修改字段的值。

```
public string Sex
{
    //如果不对 sex 属性赋值则返回默认值"Male"
    get {return sex!=null? sex:"Male";}

}
public int Number
{
    //改变了字段的值，错误的编程方式
    get {return number++;}
}
```

5.3.4 方法

方法是对象对外提供的服务，是类中执行数据计算或进行其他操作的重要成员。

1. 方法的定义

在类中定义方法的格式如下：

```
[访问修饰符] 返回值类型 方法名 ([参数列表])
{
    方法体
}
```

说明：

(1) 访问修饰符控制方法的访问级别，常见的成员访问修饰符如表 5-1 所示。

(2) 返回值类型可以是任何合法的数据类型，当无返回值时，返回值类型使用 void 关键字表示。

(3) 方法名必须符合 C#的命名规范。

(4) 参数列表是方法可以接受的输入数据，当方法不需要参数时，可省略，但不能省略圆括号；当参数不止 1 个时，需要使用逗号分隔，同时每一个参数都必须声明参数类型，即使这些参数的数据类型相同也不例外。

(5) 方法体由若干条语句组成，每条语句都必须以“;”结束。如果方法需要返回值，则使用 return 语句返回，并且返回值的类型必须与方法头中的返回值类型相同；使用 void 标记为无返回值的方法，可省略 return 语句。

【例 5-7】 完善 Circle 类，增加两个方法成员，分别计算圆的面积和周长。

```
class Circle
{
    const float pi=3.14f;
    private float radius;
    public float Radius
    {
        get
        {
            return radius;
        }
        set
        {
            if(value<0)
                radius=0;
            else
                radius=value;
        }
    }
    public float Area()                          //返回圆的面积
    {
        return pi * radius * radius;
    }
    public void Circum()                         //输出圆的周长
    {
        Console.WriteLine("圆的周长: {0}",2 * pi * radius);
    }
}
```

2. 方法的调用

一个方法一旦在某个类中声明，就可由其他方法调用。调用者可以是同一个类中的

方法，也可以是其他类中的方法。如果调用者是同一个类中的方法，则可以直接调用；如果调用者是其他类中的方法，则需要通过类的实例来引用（静态方法除外）。

定义类的方法后，有以下几种调用方法。

（1）作为一条独立的语句，例如：

```
Circle c1=new Circle();              //创建 Circle 类的实例
c1.Circum();                         //调用 c1 对象的 Circum 方法
```

（2）作为表达式的一部分，参与算术运算、赋值运算等，例如：

```
float s=c1.Area();                   //调用 c1 对象的 Area 方法，将返回值赋给变量 s
```

（3）作为另一个方法的参数来使用，例如：

```
//将 c1 对象的 Area 方法作为 Console 类的 WriteLine 方法的参数
Console.WriteLine("圆的面积:{0}",c1.Area());
```

【例 5-8】 利用例 5-7 定义的 Circle 类计算圆的面积和周长，并输出，半径由用户输入。

```
class Program
{
    static void Main(string[] args)
    {
        Circle c1=new Circle();                          //创建 Circle 类的实例
        Console.WriteLine("请输入圆的半径: ");
        c1.Radius=Convert.ToSingle(Console.ReadLine());   //通过 Radius 属性设置半径
        Console.WriteLine("圆的面积为: {0}",c1.Area()); //调用 Area 方法计算面积并输出
        c1.Circum();                                     //调用 Circle 方法输出周长
        Console.ReadLine();
    }
}
```

3. 方法的参数传递

参数是方法的调用者和方法之间传递信息的一种机制。声明方法时，所定义的参数是形式参数（简称形参），这些参数的值由调用者为其传递，调用者传递的是实际数据，称为实际参数（简称实参），调用者必须严格按照被调用的方法定义的参数类型和顺序指定实参。

方法的参数有四种类型：一是值参数，不带任何修饰符；二是引用参数，用 ref 修饰符声明；三是输出参数，用 out 修饰符声明；四是参量参数，用 params 修饰符声明。

1）值参数

调用者向方法传递值参数时，将实参的值赋给相应的形参，即被调用的方法接收到的只是实参数据值的一个副本。若在方法内部修改了形参的值，不会影响实参，即实参和形参是两个不同的变量，它们具有各自的内存地址和数据值。

【例 5-9】 值参数传递。

```
namespace Ch05Ex09
{
    class Program
    {
        static void Main(string[] args)
        {
            Swaper s1=new Swaper();
            Console.Write("请输入第一个整数: ");
            int a=Convert.ToInt32(Console.ReadLine());
            Console.Write("请输入第二个整数: ");
            int b=Convert.ToInt32(Console.ReadLine());
            s1.Swap(a,b);
            Console.WriteLine("交换后,实参的值: {0},{1}",a,b);
            Console.ReadLine();
        }
    }
    class Swaper
    {
        public void Swap(int x,int y)
        {
            int temp;
            temp=x;
            x=y;
            y=temp;
            Console.WriteLine("交换后,形参的值: {0},{1}",x,y);
        }
    }
}
```

上述代码编译后,运行结果如图 5-3 所示。由图中可以看出,虽然在方法中改变了形参的值,但是没有影响到实参。

图 5-3 例 5-9 的运行结果

2) 引用参数

调用者向方法传递引用参数时,将实参的引用赋给相应的形参。实参的引用代表数据值的内存地址,因此形参和实参将指向同一个引用。若在方法内部修改了形参变量所引用的数据值,则同时也修改了实参变量所引用的数据值。C# 通过 ref 关键字声明引用参数,如果希望传递数据的引用,必须在方法声明和方法调用中都明确地指定 ref 关键字。

【例 5-10】 引用参数传递。

对例 5-9 中的代码作如下修改:

```
namespace Ch05Ex09
```