第3章

面向对象编程基础

【知识目标】

(1) 理解面向对象的含义。

(2) 掌握面向对象的类的属性和行为的抽象定义方法。

(3) 理解面向对象的抽象性、封装性、继承性和多态性。

【能力目标】

(1) 能够用面向对象的思想(对象和类)考虑现实中要解决的问题。

(2) 能够正确运用面向对象的抽象性、封装性对现实中的事物进行抽象定义。

【内容导读】

本章涉及的是面向对象编程的基础知识,介绍了面向对象的核心概念和主要特性。通过本章的学习使读者能够对面向对象编程的基本思想有一个总体认识。

3.1 面向对象概念

面向对象编程是一种新的程序设计范型,其基本思想是使用对象、类、继承、封装、消息等基本概念来进行程序设计。它从现实世界中客观存在的事物(即对象)出发来构造软件系统,并在系统构造中尽可能运用人类的自然思维方式,直接以现实世界中的事物为中心来思考问题、认识问题,并根据这些事物的本质特点,把它们抽象地表示为系统中的对象,作为系统的基本构成单位。面向对象编程是当前软件开发技术的主流。

面向对象编程具有以下优点。

(1) 面向对象编程侧重于对象。对象具有特定的行为和属性,行为和属性决定了对象之间的交互作用方式,以及对象本身的行为方式。由于对象反映了现实世界,所以使得程序更易于设计。

(2) 面向对象编程能够隐藏特定的数据。面向对象编程可以只把对象的特定的行为公开给用户,从而隐藏了不需要或者不方便透露给用户的数据。

(3) 面向对象编程允许对象是独立的。在面向对象编程中,对象能够独立存在,并且具有调用其他对象行为的功能。通过使用面向对象的编程方法,开发人员能够创建出反映现实世界对象的应用程序。

(4) 面向对象编程允许代码可重用性。面向对象编程中对象是一个模块单元,是完备的实体,因此可以具有高度的可重用性。

(5) 面向对象编程基于消息或者事件驱动。在这类应用程序中,每个对象都可以向其他对象发送消息。

1. 对象

客观世界中任何一个事物都可以看成一个对象(Object),对象可以是自然物体(如汽车、房屋、狗),也可以是社会生活中的一种逻辑结构(如班级、部门、组织),甚至一篇文章、一个图形、一项计划等都可以视作对象。对象是构成系统的基本单位,在实际社会生活中,人们都在不同的对象中活动。

任何一个对象都具有两个要素,即属性(Attribute)和行为(Behavior),一个对象往往由一组属性和一组行为构成,一辆汽车是一个对象,它的属性是生产厂家、品牌、型号、颜色、价格等,它的行为是它的功能,如发动、停止、加速等,一般来说,凡是具备属性和行为这两个要素的,都可以作为对象。

对象是某些事物的一个抽象,反映事物在系统中需要保存的必要信息和发挥的作用,是包含一些特殊属性(数据)和服务(行为方法)的封装实体。具体来说,它应有唯一的名称,有一系列状态(表示为数据),有表示该对象的一系列行为(方法),简言之,就是

对象＝属性＋行为(方法、操作)

2. 事件与方法

事件(Event)又称为消息(Message),表示向对象发出的服务请求。方法(Method)表示对象能完成的服务或执行的操作功能。

一个系统中的多个对象之间通过一定的渠道相互联系,要使某一个对象实现某一种行为或操作,应当向它传送相应的消息。例如想让汽车行驶,必须由人去踩油门,向汽车发出相应的信号。对象之间就是这样通过发送和接收消息互相联系的。

在面向对象的概念中,一个对象可以有多个方法,提供多种服务,完成多种操作功能。但这些方法只有在另外一个对象向它发出请求之后(发生事件)才会被执行。

3. 类与对象

逻辑意义上的类是现实世界中各种实体的抽象概念,而对象是现实生活中的一个个实体,例如,在现实世界中一辆辆汽车、摩托车、自行车等实体是对象,而交通工具则是这些对象的抽象,交通工具就是一个类。

在面向对象的概念中,类(Class)表示具有相同属性和行为的一组对象的集合,并为该类的所有对象提供统一的抽象描述。

类是对相似对象的抽象,而对象是该类的一个特例,类与对象的关系是抽象与具体的

关系。

4. 类的基本特征

面向对象的最基本的特征是抽象、封装、继承及多态。

(1) 抽象

抽象(Abstraction)是处理事物复杂性的方法,只关注与当前目标有关的方面,而忽略与当前目标无关的那些方面,例如,在学生成绩管理中,张三、李四、王五作为学生,我们只关心和他们成绩管理有关的属性和行为,如学号、姓名、成绩、专业等特性。抽象的过程是将有关事物的共性归纳、集中的过程,例如,凡是有轮子、能滚动并前进的陆地交通工具统称为“车子”,把其中用汽油发动机驱动的抽象为“汽车”,把用马拉的抽象为“马车”。

抽象能表示同一类事物的本质,如果一个人会使用自己家里的电视机,在别人家里看到即便是不同牌子的电视机,也能对它进行操作。因它具有所有电视机所共有的特征,而C#语言中的数据类型就是对一系列具体数据的抽象,例如:int 是对所有整数的抽象,double 是对所有双精度浮点型数据的抽象。

(2) 封装

封装(Encapsulation)有两个方面的含义:一是将有关的数据和操作代码封装在一个对象中,形成一个基本单位,各个对象之间相对独立,互不干扰;二是将对象中某些部分对外隐藏,即隐藏其内部细节,只留下少量接口,与外界联系,接收外界的消息。这种对外界隐藏的做法称为信息隐藏(Information Hiding)。信息隐藏有利于数据安全,防止无关的人了解和修改数据。

封装把对象的全部属性和全部行为结合在一起形成一个不可分割的独立单位。而通过信息隐藏技术,用户只能见到对象封装界面上的信息,对象内部对用户是隐蔽的。

例如,一台电视机就是一个封装体。从设计者的角度来讲,不仅需要考虑内部的各种元器件,还要考虑主机板、显像管等元器件的连接与组装;从使用者的角度来讲,只关心其型号、颜色、重量等属性,只关心电源开关按钮、音量开关、调频按钮、视频输入/输出接口等用起来是否方便,根本不用关心其内部构造。

因此,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行为的具体实现,只需要用设计者提供的消息接口来访问该对象。

(3) 继承

继承是指新类可以获得已有类(称为父类或基类)的属性和行为,新类称为已有类的派生类或子类。例如,在软件开发中已建立了一个名为 A 的类,又想建立一个名为 B 的类,而后者与前者内容基本相同,只是在前者基础上增加一些新的属性和行为,显然不必再从头设计一个新类,只须在 A 类的基础上增加一些新的内容即可,而 B 类的对象拥有 A 类的全部属性与方法,称作 B 类对 A 类的继承,在 B 类中不必重新定义已在 A 类中定义过的属性和方法。其中,A 类即为基类或父类,B 类即为派生类或子类。继承机制的优势在于降低了软件开发的复杂性和费用,使软件系统易于扩充,大大缩短了软件开发周期,对于大型软件的开发具有重要的意义。

(4) 多态

多态(Polymorphism)是指在基类中定义的属性或方法被派生类继承后,可以具有不同的数据类型或表现出不同的行为。其对象对同一消息会做出不同的响应,如张三、李四和王五分别是属于 3 个班的 3 个学生,在听到上课铃声后,他们会分别走进 3 个不同的教室。同样,“启动”是所有交通工具都具有的操作,但不同的具体交通工具其“启动”操作的具体实现是不同的,如汽车的启动是“发动机点火,启动引擎”,启动轮船时要“起锚”,气球飞艇启动是“充气,解缆”。为了实现多态性,需要在派生类中更改从基类中自动继承来的数据类型或方法。这种为了替换基类的部分内容而在派生类中重新进行定义的操作,在面向对象的概念中称为覆盖。这样一来,不同类的对象可以响应同名的消息(方法)来完成特定的功能,但其具体的实现方法却可以不同。

3.2　类和对象

3.2.1　类的声明和实例化

1. 类的声明

类的声明一般格式如下:

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

其中,访问修饰符用来限制类的作用范围或访问级别,可省略,默认为 internal(内部),其他具体含义如表 3.1 所示;类名应为一个合法的 C# 标识符;基类表明所定义的类是一个派生类,可省略;类的成员放在花括号中,构成类的主体,用来定义类的属性和行为。类的成员包括常量、字段、属性、索引器、方法、事件、构造函数等。

表 3.1　类的访问修饰符

修饰符	说　明
public	允许外界访问
protected	只允许派生类访问
private	不允许外界访问
abstract	指定类为抽象类,是其他类的基类 ,不能实例化
sealed	指定类不能被继承
internal	访问修饰符,允许同一命名空间的其他类访问

2. 对象

类是抽象的,无法直接使用,所以必须创建该类的实例即对象,然后再通过这个对象来访问其数据成员或调用其方法。

(1) 对象的声明与创建

声明对象的格式与声明基本数据类型的格式相同,其语法格式如下:

```
类名  对象名;
```

例如:

```
Student  stu1;                    //声明一个 Student 对象 stu1
```

但是,对象声明后需要用 new 关键字将其实例化,这样才能为对象在内存中分配空间,实例化对象的语法格式如下:

```
对象名=new 类名();
```

例如:

```
stu1=new Student();               //为 stu1 分配内存空间
```

也可以在声明对象的同时实例化对象。语法格式如下:

```
类名  对象名=new 类名();
```

例如:

```
Student stu2=new Student();     //声明的同时创建对象
```

(2) 类成员的访问

类成员有两种访问方式:一种是在类的内部访问;另一种是在类的外部访问。

在类的内部访问类的成员,表示一个类成员要使用当前类中的其他成员,可以直接使用成员名称,有时为了避免引起混淆,也可采用如下形式:

```
this.类成员
```

说明:this 表示当前对象,是 C# 语言的关键字。

3.2.2 类的数据成员和作用域

类的成员包括类的常量、字段、属性、索引、方法、事件、构造函数等,其中,常量、字段和属性都是与类的数据有关的成员。

1. 字段

字段表示类的成员变量,字段的值代表某个对象的数据状态。不同的对象,数据状态不同,意味着各字段的值也不同。声明字段的方法与定义普通变量的方法相同,其一般格式如下:

```
[访问修饰符] 数据类型 字段名;
```

其中,访问修饰符用来控制字段的访问级别,可省略。

例如:

```
public double radius;
```

2. 属性

由于字段表示状态和数据，通常是私有的，所以必须有一种机制能够向类外提供信息。但是如果把字段的修饰符改成 public，就会违反封装规则，导致可从类外直接操作字段。基于这种需求，引入了属性，属性不表示存储位置，而是一个特殊的接口，用于对外交互类的静态信息。属性具有 get()和 set()两个方法，get()获取字段值，set()使用 value 关键字获取用户的输入并设置字段的值。

定义属性的一般形式如下：

```
[访问修饰符] 数据类型 属性名
{
    get
    {
        //获取属性的代码,用 return 返回值
    }
    set
    {
        //设置属性的代码,用 value 赋值
    }
}
```

3.2.3　类的可访问性

类成员的访问级别默认为 private，但可将其声明为其他的类型。类成员访问修饰符的说明如表 3.2 所示。

表 3.2　类成员访问修饰符的说明

访问修饰符	说　明
public（共有）	访问不受限制
protected（保护）	访问限于所在类和所在类的派生类
private（私有）	访问限于所在类
protected internal（内部受保护）	访问限于所在类和所在类的派生类，或同一命名空间内
internal（内部）	访问限于同一命名空间内

3.3　类的方法

3.3.1　方法的声明与调用

方法的使用分声明与调用两个环节。

1. 方法的声明

声明方法的一般形式如下：

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

其中：

(1) 访问修饰符控制方法的访问级别，可用于方法的修饰符包括 public、protected、private 和 internal 等。访问修饰符是可选的，默认情况下为 private。

(2) 方法的返回类型用于指定由该方法计算和返回的值的类型，可以是任何合法的数据类型，包括值类型和引用类型，如果一个方法不返回一个值，则返回值类型使用 void 关键字来表示。

(3) 方法名必须符合 C# 的命名规范，与变量名的命名规则相同。

(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，可省略参数列表，但不能省略圆括号。当参数不止一个时，需要使用逗号分隔，同时每一个参数都必须声明数据类型，即使这些参数的数据类型相同也不例外。

(5) 花括号中的内容为方法的主体，由若干条语句组成，每一条语句都必须使用分号结尾。当方法结束时如果需要返回操作结果，则使用 return 语句返回，并且返回的值的类型要与返回值的类型相匹配。如果使用 void 将方法标记为无返回值的方法，则可省略 return 语句。

【任务 3.1】 求任意两个整数之间的所有数的平方和。

```
private static int sum(int x, int y)  //方法头
{
    int i, s=0;                        //方法中变量的定义
    for (i=x; i<=y; i++)               //通过循环求得 x,y 之间的所有数的平方之和
    {
        s=s+i*i;
    }
    return (s);                        //返回累加和的值
}
```

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，调用者既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，但静态方法例外，静态方法可以通过类名直接调用。

(1) 在方法声明的类中调用该方法。其语法格式如下：

方法名(参数列表)

(2) 在方法声明的类的外部调用该方法,需要通过类声明的对象调用该方法,其格式如下：

对象名.方法名(参数列表);

【任务 3.2】 方法的使用。

```
namespace example_3
{
    class Person
    {
        private String name="姚明";              //类的数据成员声明
        private int age=32;
        public Person(string Name, int Age)     //构造函数,函数名和类同名,无返回值
        {
            name=Name;
            age=Age;
        }
        public void Display()                   //显示姓名和年龄,相当于 Get()函数
        {
            Console.WriteLine("姓名:{0},年龄: {1}", name, age);
        }
        public void SetName(string PersonName)  //指定修改姓名的方法(函数)
        {
            name=PersonName;
        }
        public void SetAge(int PersonAge)       //指定修改年龄的方法(函数)
        {
            age=PersonAge;
        }
    }
    class Class1
    {
        static void Main(string[] args)
        {
            Person OnePerson=new Person("刘翔", 31);    //生成类的对象
            OnePerson.Display();
            OnePerson.SetName("张琳");          //通过 SetName()方法访问类成员
            OnePerson.SetAge(28);
            OnePerson.Display();
         }
    }
}
```

3.3.2 方法的参数传递

在声明方法时，所定义的参数是形式参数（简称形参），这些参数的值由调用方负责为其传递，调用方传递的是实际数据，称为实际参数（简称实参），调用方必须严格按照被调用的方法所定义的参数类型和顺序指定实参。在调用方法时，参数传递就是将实参传递给形参的过程。

方法的参数传递按性质可分为按值传递参数与按引用传递参数。

1. 按值传递参数

按值传递参数时，把实参变量的值赋给相对应的形参变量，即被调用的方法所接收到的只是实参数据值的一个副本。当在方法内部更改了形参变量的数据值时，不会影响实参变量的值，即实参变量和形参变量是两个不相同的变量，它们具有各自的内存地址和数据值。因此，实参变量的值传递给形参变量时是一种单向值传递。

值类型的参数在传递时默认为按值传参。string 和 object 虽然是引用型数据，但从表现形式来看，具有按值传参的效果。

【任务 3.3】 按值传递参数示例。

```
class example
{
    static void exchange(int a, int b)
    {
        int temp;
        temp=a; a=b; b=temp;
    }
    public static void Main()
    {
        int x, y;
        Console.WriteLine("请输入 x 和 y 的值: ");
        x=Convert.ToInt32(Console.ReadLine());
        y=Convert.ToInt32(Console.ReadLine());
        exchange(x, y);
        Console.WriteLine("交换后的 x 和 y 的值: {0},{1}", x, y);
    }
}
```

程序运行结果如图 3.1 所示。

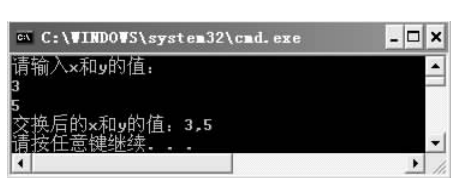

图 3.1 任务 3.3 运行结果

2. 按引用传递参数

一个方法只能返回一个值，但在实际应用中常常需要方法能够返回多个值或修改传入的参数值并返回，如果需要完成以上任务，只用 return 语句是无法做到的，这时可以使用按引用传递参数的方式来实现。

调用方法传递引用型参数时，调用方将把实参变量的引用赋给相对应的形参变量。实参变量的引用代表数据的内存地址，因此，形参变量和实参变量将指向同一个引用。如果在方法内部更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。

当值类型和 string 类型参数要按引用方式传参时，可以通过 ref 关键字来声明引用参数，无论是形参还是实参，只要希望传递数据的引用，就必须添加 ref 关键字。

【任务 3.4】 按引用传递参数示例。

```
class example
{
    static void exchange(ref int a,ref int b)
    {
        int temp;
        temp=a; a=b; b=temp;
    }
    public static void Main()
    {
        int x, y;
        Console.WriteLine("请输入 x 和 y 的值: ");
        x=Convert.ToInt32(Console.ReadLine());
        y=Convert.ToInt32(Console.ReadLine());
        exchange(ref x, ref y);
        Console.WriteLine("交换后的 x 和 y 的值: {0},{1}", x, y);
    }
}
```

程序运行结果如图 3.2 所示。

图 3.2　任务 3.4 运行结果

3. 数组型参数

数组也是引用类型数据，把数组作为参数传递时，也是引用传参。但把数组作为参数，有两种使用形式：一种是在形参数组前不添加 params 修饰符，另一种是在形参数组前添加 params 修饰符。不添加 params 修饰符时，所对应的实参必须是一个数组名；添加