第3章 面向对象程序设计基础

3.1 面向对象的基本概念

面向对象是一种程序设计方法，或者说是一种程序设计规范，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。所谓“面向对象”就是以对象及其行为为中心，来考虑处理问题的思想体系和方法。采用面向对象的方法设计的软件，不仅易于理解，而且易于维护和修改，从而提高了软件的可靠性和可维护性，同时也提高了软件的模块化和可重用化的程度。

Java 是一种纯粹的面向对象的程序设计语言。用 Java 进行程序设计必须将自己的思想转入到面向对象的世界，以面向对象世界的思维方式来思考问题，因为 Java 程序乃至 Java 程序内的一切都是对象。

1. 对象的基本概念

对象是系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位。一个对象由一组属性和对这组属性进行操作的一组服务组成。从更抽象的角度来说，对象是问题域或实现域中某些事物的一个抽象，它反映该事物在系统中需要保存的信息和发挥的作用；它是一组属性和有权对这些属性进行操作的一组服务的封装体。客观世界是由对象和对象之间的联系组成的。

在面向对象的程序设计方法中，对象是一些相关的变量和方法的软件集，是可以保存状态（信息）和一组操作（行为）的整体。软件对象经常用于模仿现实世界中的一些对象，比如桌子、电视、自行车等。

现实世界中的对象有两个共同特征：形态和行为。

例如，把汽车作为对象，汽车的形态有车的类型、款式、挂挡方式、排量大小等，其行为有刹车、加速、减速以及改变挡位等。如图 3.1 所示。

软件对象实际上是现实世界对象的模拟和抽象，它同样也有形态和行为。一个软件对象利用一个或者多个变量来体现它的形态。变量是由用户标识符来命名的数据项。软件对象用方法来实现它的行为，它是跟对象有关联的函数。在面向对象设计的过程中，可以利用软件对象来代表现实世界中的对象，也可以用软件对象来模拟抽象的概念，比如，事件是一个 GUI（图形用户界面）窗口系统的对象，它可以代表用户按下鼠标按钮或者键盘上的按键所产生的反应。

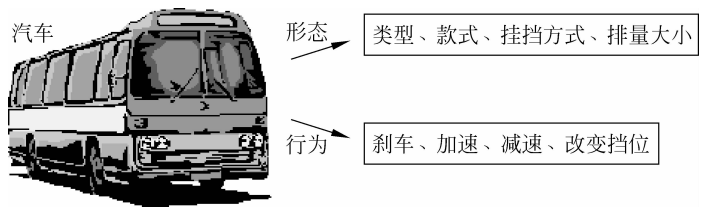
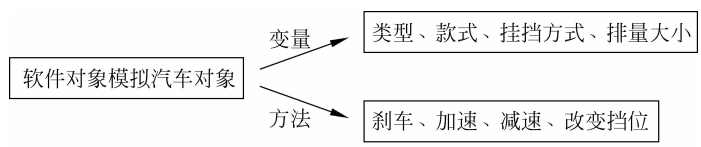


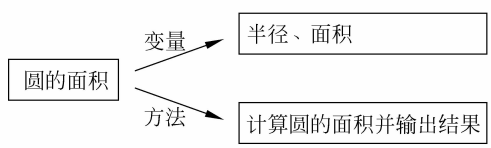
图 3.1　汽车对象的形态和行为

例如,用软件对象模拟汽车对象,汽车的形态就是汽车对象的变量,汽车的行为就是汽车对象的方法,如图 3.2(a)所示。

再例如,用软件计算圆的面积,描述圆面积的形态是圆的半径和圆的面积,计算圆面积的行为是圆的面积公式,因此,圆面积对象的变量是圆的半径和圆的面积,圆面积对象的方法是计算圆面积的公式及输出结果,如图 3.2(b)所示。



(a) 软件对象模拟汽车对象



(b) 求圆的面积

图 3.2　软件对象的变量和方法

2. 类的基本概念

对象是指具体的事物,而类是指一类事物。

把众多的事物进行归纳、分类是人类在认识客观世界时经常采用的思维方法。分类的原则是按某种共性进行划分。例如,客车、卡车、小轿车等具体机车都有相同的属性:有内燃发动机、有车身、有橡胶车轮、有方向盘等,把它们的共性抽象出来,就形成了“汽车”的概念。但当说到某辆车时,光有汽车这个概念是不够的,还须说明究竟是小轿车还是大卡车。因此,汽车是抽象的、不具体的一个类的概念。而具体的某辆汽车则是“汽车”这个类的对象,也称它是汽车类的一个实例。

由类来确定具体对象的过程称为实例化,即类的实例化结果就是对象,而对一类对象的抽象就是类。

类用 class 作为关键字,例如要创建一个汽车类,则可表示为:

$$\text{class 汽车}\begin{cases}\text{变量:类型、款式、挂挡方式、排量大小}\\ \text{方法:刹车、加速、减速、改变挡位}\end{cases}$$

当要通过汽车类来创建一个轿车对象,并使用它的刹车行为方法时,则要用下面的格式

进行实例化：

```
汽车　轿车 = new　汽车( ) ;　　//实例化汽车类,即创建轿车对象
轿车.刹车( ) ;　　//引用汽车对象的刹车方法
```

这里，只是粗略地介绍了类和对象的概念，在后面的内容中将详细介绍类和对象的设计方法。

3.2　类

类和对象是 Java 的核心和本质。它们是 Java 语言的基础，编写一个 Java 程序，在某种程度上来说就是定义类和创建对象。定义类和建立对象是 Java 编程的主要任务。

3.2.1　类的定义

从本书的开始就接触到类了。当然，这都是一些非常简单的类。类是组成 Java 程序的基本要素，本节将介绍如何创建一个类。

1. 类的一般形式

类由类声明和类体组成，而类体又由成员变量和成员方法组成，即：

```
public　class　类名　←—类声明
{
    成员变量;
    成员方法;　←—类体
}
```

图 3.3 说明了一个具体的类的形式。

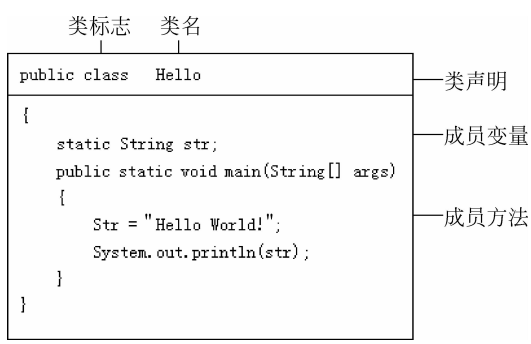


图 3.3　类的形式

2. 类声明

类声明由 4 部分组成：类修饰符、类关键字 class、声明父类、实现接口，其一般形式如下。

```
[public][abstract|final] class 类名 [extends 父类名] [implements 接口列表]
```

```
{
   ⋮
}
```

各组成部分的具体说明如下。

1) 类修饰符

其中,可选项 public,abstract,final 是类修饰符。类修饰符说明这个类是一个什么样的类。如果没有声明这些可选项的类修饰符,Java 编译器将给出缺省值,即指定该类为非 public、非 abstract、非 final 类。类修饰符的含义分别为:

public:这个 public 关键字声明了类可以在其他类中使用。缺省时,该类只能被同一个包中的其他类使用。

abstract:声明这个类为抽象类,即这个类不能被实例化。一个抽象类可以包含抽象方法,而抽象方法是没有实现的空的方法,所以抽象类不具备实际功能,只用于衍生子类。

final:声明该类不能被继承,即不能有子类。也就是说,不能用它通过扩展的办法来创建新类。

2) 类的关键字 class

在类声明中,class 是声明类的关键字,表示类声明的开始,类声明后面跟着类名,按习惯类名要用大写字母开头,并且类名不能用阿拉伯数字开头。给类名命名时,最好取一个容易识别且有意义的名字,避免 A,B,C 之类的类名。

3) 声明父类

extends 为声明该类的父类,这表明该类是其父类的子类。一个子类可以从它的父类继承变量和方法。值得注意的是 Java 和 C++不一样,在 extends 之后只能有一个父类,即 extends 只能实现单继承。

创建子类格式:

```
class SubClass extends 父类名
 {
    ⋮
}
```

4) 实现接口

为了在类声明中实现接口,要使用关键字 implements,并且在其后面给出接口名,当要实现有多个接口时,各接口名以逗号分隔,其形式为:

```
implements 接口 1,接口 2 …
```

接口是一种特殊的抽象类,这种抽象类中只包含常量和方法的定义,而没有变量和方法的实现。一个类可以实现多个接口,以某种程度实现“多继承”。

3.2.2 成员变量和局部变量

在 Java 语言中,变量按在程序中所处不同位置分为 2 类:成员变量和局部变量。如果一个变量在方法内部声明,该变量称之为局部变量。如果类体中的一个变量在所有方法外

部声明，该变量称之为成员变量。成员变量从定义位置起至该类体结束均有效，而局部变量只在定义它的方法内有效。

变量{成员变量（在类体中定义，在整个类中都有效）；
局部变量（在方法中定义，只在本方法中有效）。

1. 成员变量

最简单的变量声明的形式为：

```
数据类型　变量名;
```

这里声明的变量类型可以是基本数据类型，也可以是引用数据类型。

数据类型{基本类型（整型、浮点型、逻辑型、字符型）；
引用数据型（数组、类对象）。

声明成员变量的更一般的形式：

```
[可访问性修饰符][static][final][transient][volatile]类型　变量名
```

上述属性用方括号括起来，表示它们都是可选项，其含义分别为：

[可访问性修饰符]：说明该变量的可访问属性，即定义哪些类可以访问该变量。该修饰符可为 public，protected，package 和 private，它们的含义在后面将会更详细地介绍。

[static]：说明该成员变量是一个静态变量（类变量），以区别一般的实例变量。类变量的所有实例使用的是同一个副本。

[final]：说明一个常量。

[transient]：声明瞬态变量，瞬态变量不是对象的持久部分。

[volatile]：声明一个可能同时被并存运行中的几个线程所控制和修改的变量，即这个变量不仅仅被当前程序所控制，而且在运行过程中可能存在其他未知程序的操作来影响和改变该变量的值，volatile 关键字把这个信息传送给 Java 的运行系统。

成员变量还可以进一步分为实例变量和类变量，这些内容在后面讲关键字 static 时再做介绍。

2. 局部变量

在方法中声明的变量以及方法中的参数称为局部变量。局部变量除了作用范围仅适用于本方法之外，其余均与上面讨论的成员变量是一致。

```
class Data
{
  int x = 12,y = 5;
  public void sum()
  {
      int s;
      s = x + y;                    //使用成员变量 x = 12,y = 5
  }
}
```

在类 Data 中,x,y 是成员变量,s 是局部变量。成员变量 x,y 在整个类中有效,类中所有方法都可以使用它们,但局部变量 s 仅限于在 sum()方法内部使用。

局部变量和成员变量的作用范围如图 3.4 所示。其中 x,y 是成员变量,a,b 是局部变量。

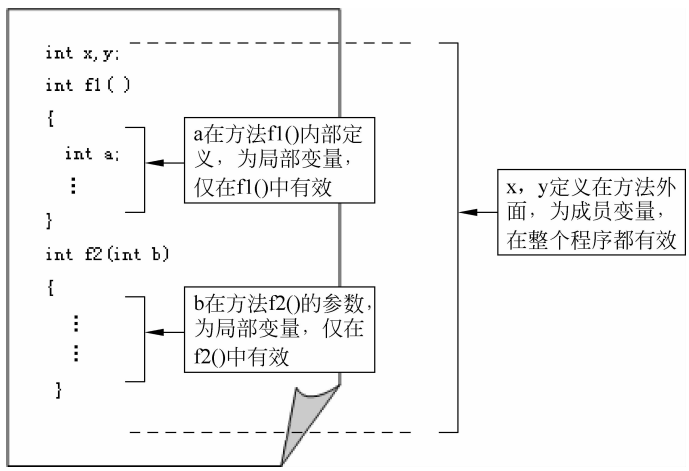


图 3.4 变量的作用域

如果局部变量名与成员变量名相同,则成员变量被屏蔽。

例如:

```
class Data
{
  int x = 12,y = 5;
  public void sum()
  {
      int x = 3;                    //局部变量 x 屏蔽了成员变量
      int s;
      s = x + y;
  }
}
```

由于在 sum()方法内部也定义了变量 x=3,这时成员变量 x=12 被屏蔽,变量 x 的值为 3。y 的值仍是 5,因此,s=3+5=8。

如果在 sum()方法内部还需要使用成员变量 x,则要用关键字 this 来引用当前对象,它的值是调用该方法的对象。

```
class Data
{
  int x = 12,y = 5;
  public void sum()
  {
     int x = 3;                     //局部变量 x
     int s;
     s = this.x + y;                //在 sum()方法使用成员变量,则用 this 来说明
```

```
    }
}
```

由于 this.x 是成员变量，因此，s＝12＋5＝17。

3.3　成员方法

在 Java 中，必须通过方法才能完成对类和对象的属性操作。成员方法只能在类的内部声明并加以实现。一般在类体中声明成员变量之后再声明方法。

3.3.1　方法的定义

1. 方法的一般形式

我们已经知道一个类由类声明和类体两部分组成，方法的定义也由两个部分组成：方法声明和方法体。方法定义的一般形式为：

```
返回类型 函数名(数据类型 1  参数 1,数据类型 2  参数 2……)   ←方法声明
{
    …      (局部变量定义);
    …    (方法功能实现);          ←方法体
    return  (返回值);
}
```

在方法声明中，返回类型可以是基本数据类型或引用类型，它是方法体中通过 return 语句返回值的数据类型，也称为该方法的类型。当该方法为无返回值时，需要用 void 作方法的类型。

方法名是由用户定义的标识符。方法名后面有一对小括号，如果括号里面是空的，这样的方法就称为无参方法；如果括号里面至少有一个参数(称为形式参数，简称形参)，则称该方法为有参方法。方法的形参是方法与外界关联的接口，形参在定义时是没有值的，外界在调用一个方法时会将相应的实际参数值传递给形参。

用一对大括号括起来的语句构成方法体，完成方法功能的具体实现。方法体一般由 3 部分组成：第 1 部分为定义方法所需的变量，方法内部定义的变量称为局部变量；第 2 部分完成方法功能的具体实现；第 3 部分由 return 语句返回方法的结果。

方法不允许嵌套定义，即不允许一个方法的定义放在另一个方法的定义中。

图 3.5 给出了一个简单 main()方法的代码。在本方法中实现在命令窗口中显示字符串 str 的内容。

方法还可以有许多其他属性，比如参数、访问控制等。

2. 方法的返回值

在方法定义中，方法的类型是该方法返回值的数据类型。方法返回值是方法向外界输出的信息。根据方法功能的要求，一个方法可以有返回值，也可以无返回值(此时方法的类型为 void 型)。方法的返回值一般在方法体中通过 return 语句返回。

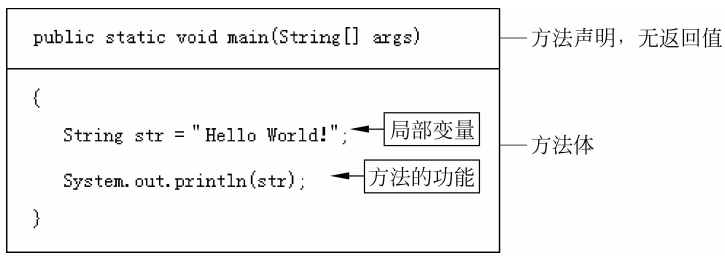


图 3.5 一个 main()方法的代码

return 语句的一般形式为：

```
return 表达式;
```

该语句的功能是将方法要输出的信息反馈给主调方法。

【例 3.1】 有参方法实例。编写一个方法模块，实现计算 1＋2＋3＋…＋n 的 n 项和的功能。

```
1  int  mysum(int n)   ←方法声明,声明名为 mysum,有 int 类型返回值,有参数
2  {
3    int i, s = 0;     ←声明局部变量
4    for(i = 1; i <= n; i++)   ←实现方法功能
5      s = s + i;              ←方法体
6    return s;         ←将计算的结果 s 返回出去
7  }
```

方法说明：

(1) 第 1 行“int mysum(int n)”是方法声明，其中 mysum 是方法名，方法类型为 int 类型，表明该方法计算的结果为整型；括号中的“int n”表示 n 是形式参数，简称形参，其类型为 int。形参 n 此时并没有值。

(2) 第 2 行至第 7 行是方法体部分，用以实现求和的功能。

(3) 第 6 行是通过“return s；”将求得的和值 s 返回作为 mysum 方法的值。

在一个方法中允许有多个 return 语句，但每次调用只能有一个 return 语句被执行，即只能返回一个方法值。

【例 3.2】 方法中有多个 return 的示例，求两个数中的较大数。

```
1  int max(int x, int y)   ←定义方法 max(),该方法有两个形参
2  {
3    if(x > y)   return x;   ←若 x 大于 y,返回值为 x,否则返回值为 y
4    else   return y;
5  }
```

3.3.2 方法的调用

1. 方法调用的语法形式

为实现操作功能而编写的方法必须被其他方法调用才能运行。通常把调用其他方法的

方法称为主调方法，被其他方法调用的方法称之为被调方法。

方法调用的语句形式如下：

```
函数名(实际参数 1, 实际参数 2, …, 实际参数 n);
```

也就是说，一个方法在被调用语句中，其参数称为实际参数。实际参数简称为实参，方法调用中的实参不需要加数据类型，实参的个数、类型、顺序要和方法定义时的形参一一对应。

对有参方法的调用，实际参数可以是常数，变量或其他构造类型数据及表达式，各实参之间用逗号分隔。对无参方法调用时则无实际参数。

定义有参方法时，形式参数并没有具体数据值，在被主调方法调用时，主调方法必须给出具体数据(实参)，将实参值依次传递给相应的形参。

Java 程序的运行总是从 main()开始，main()方法又称之为主方法，它可以调用任何其他方法，但不允许被其他方法调用。除了 main()方法以外，其他任何方法的关系都是平等的，可以相互调用。

【例 3.3】 方法调用示例，计算 1＋2＋3＋…＋100 的和。

算法设计：

在主函数中调用例 3.1 中计算前 n 项和的方法模块，将调用函数时，将函数的参数(实参)设置为 100。这时，函数 mysum 的形参 n 得到具体值 100。从而计算 1＋2＋3＋…＋100 的和。

```
1   import javax.swing.*;
2   public class  Example3_3
3   {
4     public static void main(String[] args)
5     {
6       float sum = mysum(100);   ←调用 mysum( )方法,实参 100,函数将返回值赋值给 sum
7       JOptionPane.showMessageDialog(null, "1 + 2 + 3 + … + 100 = " + s);
8       System.exit(0);
9     }
10
11    int  mysum(int n)   ←定义 mysum 方法,将实参 100 传值给形参 n
12    {
13      int i, s = 0;
14      for(i = 1; i <= n; i++)   ←形参 n 以具体值 100 进行运算
15        s = s + i;
16      return s;   ←将计算的结果 s 返回给被调函数
17  }
```

【例 3.4】 具有多个参数的方法示例。已知三角形的底和高，计算三角形面积。

```
1   import javax.swing.*;
2   public class  Example3_4
3   {
```

```
4     public static void main(String[] args)
5     {
6       float s = area(3, 4);
7       JOptionPane.showMessageDialog(null, "三角形面积 = " + s);
8       System.exit(0);
9     }
10
11    static float area(int x, int h)
12    {
13      float s;
14      s = (x * h) / 2;
15      return s;
16    }
17  }
```

调用 area( )方法,2 个实参

main( )方法

将实参的值传给形参

定义 area( )方法,被调用时形参 x,h 分别以 3 和 4 参加运算

返回值 s

【程序说明】

(1) 在程序的第 11 行定义了一个 area()方法,该方法有 2 个 int 类型的参数,分别代表三角形的底和高。在程序的第 15 行,返回 float 类型数值 s,故方法 area 的返回类型为 float 类型。

(2) 在程序的 6 行,调用 area 方法,原方法中的形参就用具体整型数值替换(称为实参):

```
area(3, 4);
```

方法中的第 1 个实参 3 赋值给形参 x,第 2 个实参 4 赋值给形参 h,经方法 area()运算后,得到返回值 6.0。

实参与形参的传递关系如图 3.6 所示。

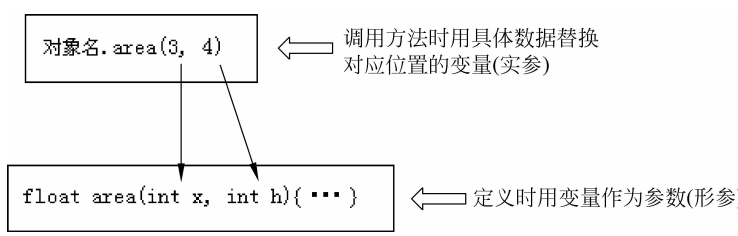


图 3.6 实参与形参的传递关系

程序运行结果如图 3.7 所示。

图 3.7 方法声明与调用的运行结果

2. 方法调用的过程

在 Java 语言中,程序运行总是从 main()方法开始,按方法体中语句的逻辑顺序向后依次执行。如若遇到方法调用,此时就转去执行被调用的方法。当被调用的方法执行完毕后,又返回到主调方法中继续向下执行。

以例 3.4 为例,当调用一个方法时,整个调用过程分为 4 步进行(图 3.8):