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，或者说它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，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，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、减速以及改变档位等，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，同样也有形态和行为。一个软件对象利用一个或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。例如，事件是一个GUI（图形用户界面）窗口系统的对象，可以代表用户按下鼠标按钮或键盘上的按键所产生的反应。

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

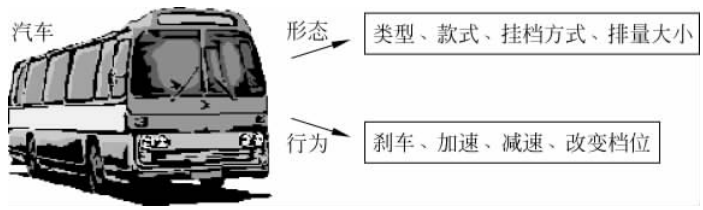

图 3-1　汽车对象的形态和行为

对象的方法，如图 3-2(a)所示。

再如，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。

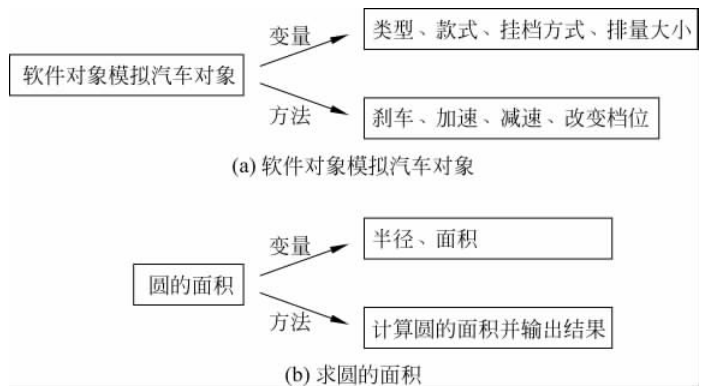

图 3-2　软件对象的变量和方法

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，还需说明究竟是小轿车还是大卡车。因此，汽车是抽象、不具体一个类的概念。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 作为它的关键字。例如，要创建一个汽车类，则可表示为：

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

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

从本节开始就接触到类了。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:

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

图 3-3 所示为一个具体类的形式。

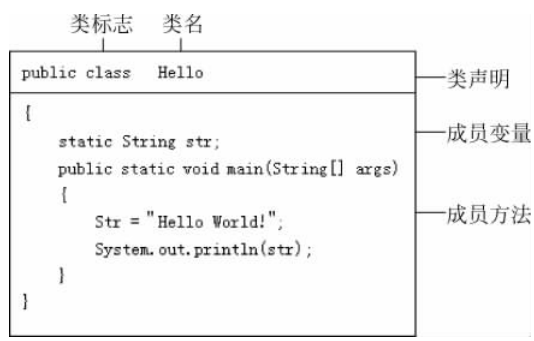

图 3-3 类的形式

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

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如下。

- public：这个 public 关键字声明了类可以在其他类中使用。其缺省时，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，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。通常，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，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。

创建子类格式如下：

```
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。当实现多接口时，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两类：成员变量和局部变量。如果类体中的一个变量在所有方法外部声明，该变量称为成员变量。如果一个变量在方法内部声明，该变量称为局部变量。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。

$$\text{变量}\begin{cases}\text{成员变量（在类体中定义，在整个类中都有效）}\\ \text{局部变量（在方法中定义，只在本方法中有效）}\end{cases}$$

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。该修饰符可分为 public、protected、package 和 private,它们的含义在后面将会详细地介绍。

[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,这些内容在后面章节讲述关键字 static 时再介绍。

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。

3. 屏蔽成员变量

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;
  }
}
```

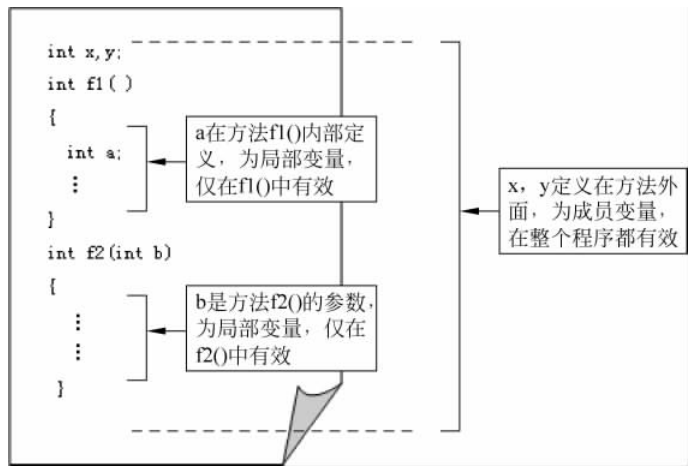

图 3-4　变量的作用域

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

已知,一个类由类声明和类体两部分组成,方法的定义也由方法声明和方法体两个部分组成。方法定义的一般形式为:

```
返回类型  方法名(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);
    return(返回值);
}
```

方法体

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。方法体一般由三部分组成:第一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;第二部分完成方法功能的具体实现;第三部分由 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。

```
public static void main(String[] args)        ← 方法声明,无返回值
{
   String str = "Hello World!";   ← 局部变量
   System.out.println(str);       ← 方法的功能
}                                              ← 方法体
```

图 3-5 一个 main 方法的代码

方法还可以有许多其他的属性,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。

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。

源程序如下：

```
1  int  mysum(int n)
2  {
3    int i, s = 0;
4    for(i = 1; i <= n; i++)
5      s = s + i;
6    return s;
7  }
```

方法声明，声明名为mysum，有int类型返回值，有参数

声明局部变量

实现方法功能

方法体

将计算的结果s返回出去

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～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，求二个数中的较大数。

源程序如下：

```
1  int max(int x, int y)
2  {
3    if(x > y)   return x;
4    else    return y;
5  }
```

定义max方法，该方法有两个形参

若x大于y，返回值为x，否则返回值为y

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 方法又称为主方法，它可以调用任何其他的方

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。

源程序如下:

```
1.  import javax.swing.*;
2.  public class  Example3_3
3.  {
4.    public static void main(String[] args)
5.    {
6.      int sum = mysum(100);
7.      JOptionPane.showMessageDialog(null, "1 + 2 + 3 + … + 100 = " + sum);
8.      System.exit(0);
9.      }
10.
11.   static int  mysum(int n)
12.  {
13.    int i, s = 0;
14.    for(i = 1; i<= n; i++)
15.      s = s + i;
16.    return s;
17. }
```

调用mysum方法,实参100,函数将返回值赋值给sum

定义mysum方法,将实参100传值给形参n

形参n以具体值100进行运算

将计算的结果s返回给被调函数

【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。

源程序如下:

```
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,两个实参

将实参的值传给形参

main( )

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,该方法有两个 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);
```

方法中的第一个实参 3 赋值给形参 x,第二个实参 4 赋值给形参 h,经 area 方法运算后,得到返回值 6.0。

实参与形参的传递关系如图 3-6 所示。

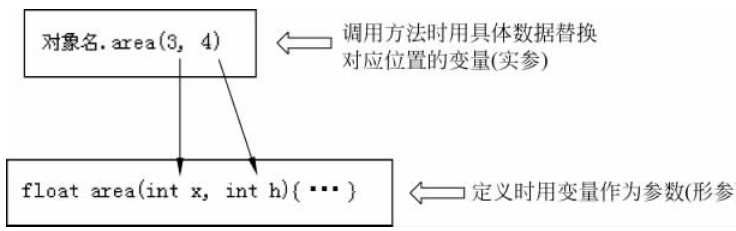

图 3-6　实参与形参的传递关系

程序运行结果如图 3-7 所示。

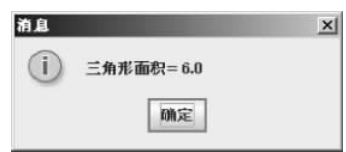

图 3-7　方法声明与调用的运行结果

2. 方法调用的过程

在 Java 语言中,程序运行总是从 main()开始,按方法体中语句的逻辑顺序向后依次执行。如遇到方法调用,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(如图 3-8 所示)。

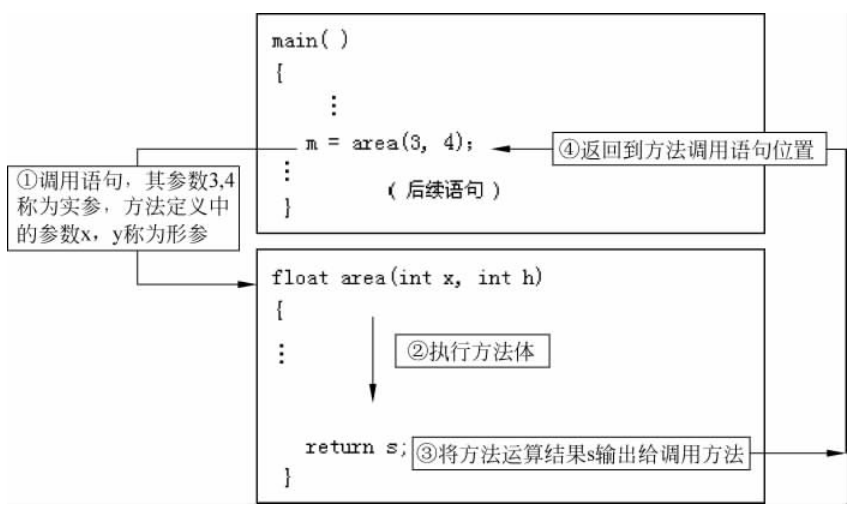

图 3-8　所示函数的调用过程

(1) 方法调用,并把实参的值传递给形参。

(2) 执行被调用方法 area 的方法体,形参用所获得的数值进行运算。

(3) 通过 return 语句将被调用方法的运算结果输出给主调方法。

(4) 返回到主调方法的方法调用表达式位置,继续后续语句的执行。

3. 方法调用的传值过程

在 Java 语言中,调用有参方法时,是通过实参向形参传值的。

形参只能在被调方法中使用,实参则只能在主调方法中使用。形参是没有值的变量,发生方法调用时,主调方法把实参的值传送给被调方法的形参,从而实现主调方法向被调方法的数据传送。方法的调用过程也称为值的单向传递,是实参到形参的传递。因此在传递时,实参必须已经有值,并且实参的个数与类型必须与形参的个数及类型完全一致。

方法调用时实参数值按顺序依次传递给相应的形参,传递过程如图 3-9 所示。

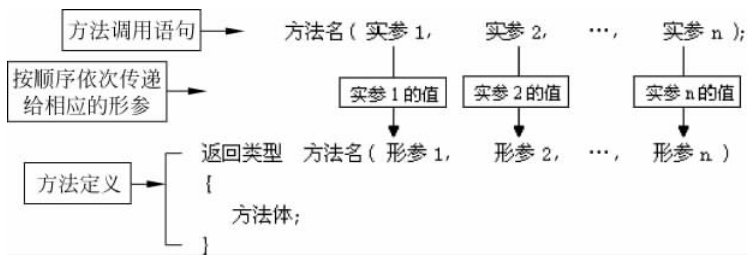

图 3-9 方法参数按值依次传递

下面进一步考察数据值从实参到形参的传递过程,如图 3-10 所示。

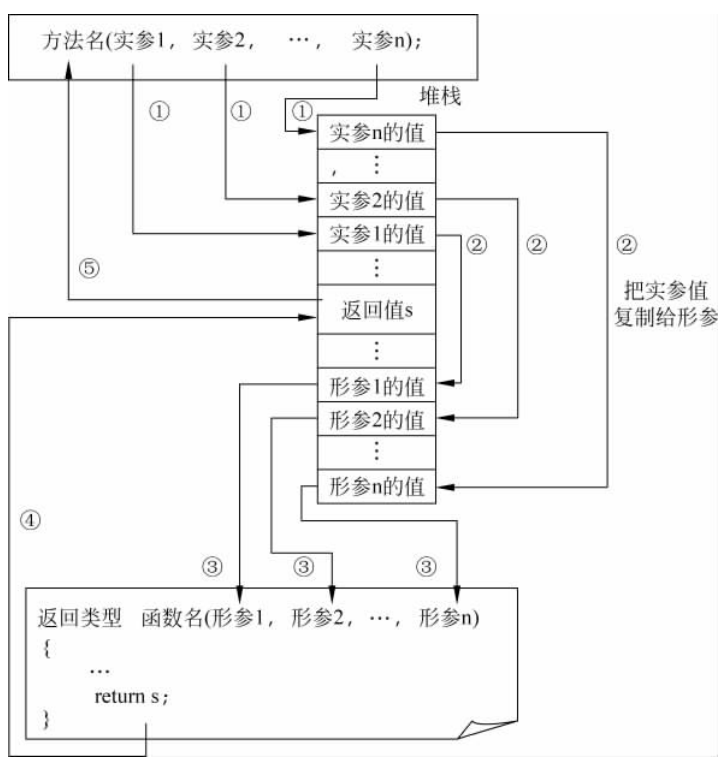

图 3-10 实参到形参的传递过程

(1) 主调方法为实参赋值,将实参值存放到内存中专门存放临时变量(又称动态局部变量)的区域中。这块存储区域称为堆栈。

(2) 当参数传递时,主调方法把堆栈中的实参值复制一个备份给被调方法的形参。

(3) 被调方法使用形参进行功能运算。

(4) 被调方法把运算结果(方法返回值)存放到堆栈中,由主调方法取回。此时,形参所占用的存储空间被系统收回。注意,此时实参值占用的存储单元被继续使用。

3.3.3　方法重载

方法重载是指多个方法享有相同的名字,但是这些方法的参数必须不同,或者是参数的个数不同,或者是参数类型不同。返回类型不能用来区分重载的方法。

注意:在设计重载方法时,参数类型的区分度一定要足够,不能是同一简单类型的参数,如 int 型和 long 型。

【例 3-5】 计算平面空间距离的计算公式分别是 $\sqrt{x^2+y^2}$ 和 $\sqrt{x^2+y^2+z^2}$,使用一个方法名用方法重载实现的程序如下:

```
1  /* 方法重载示例 */
2  import javax.swing.*;
3  public class Example3_5
4  {
5    static double distance(double x , double y)          ← 该方法有两个参数
6      {
7        double d = Math.sqrt(x*x + y*y);
8        return d;
9      }
10    static double distance(double x , double y ,double z )   ← 该方法有三个参数
11    {
12      double d = Math.sqrt(x*x + y*y + z*z);
13      return d;
14    }
15    public static void main(String args[])
16    {
17      double d1 = distance(2,3);                         ← 调用两个参数的方法
18      double d2 = distance(3,4,5);                       ← 调用三个参数的方法
19      JOptionPane.showMessageDialog(null,
20              "接受两个参数: 平面距离 d = " + d1 + "\n" +
21              "接受三个参数: 空间距离 d = " + d2);
22       System.exit(0);
23   }
24 }
```

【程序说明】

(1) 在程序的第 5 和第 10 行定义了两个方法名都是 distance 的方法,第一个 distance 方法有两个参数,第二个 distance 方法有三个参数。

(2) 在程序的第 17 行调用了有两个参数的 distance 方法,第 18 行调用了有三个参数的 receive 方法。编译器会根据参数的个数和类型来决定当前所使用的方法。

(3) 从这个例子可以看出,重载显然表面上没有减少编写程序的工作量,但实际上重载使得程序的实现方式变得很简单,只需要使用一个方法名,就可以根据不同的参数个数选择该方法不同的版本。方法的重载与调用关系如图 3-11 所示。

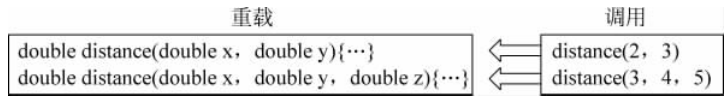

图 3-11 重载与调用关系

程序运行结果如图 3-12 所示。

图 3-12 计算平面距离和空间距离

3.3.4 构造方法

构造方法是一个特殊的方法，主要用于初始化新创建的对象。构造方法的方法名要求与类名相同，而且无返回值。在创建对象时，Java 系统会自动调用构造方法为新对象初始化。另外，构造方法只能通过 new 运算符调用，用户不能直接调用。需要注意的是，在这里说构造方法无返回值，并不是要在构造方法名前加上 void，构造方法名是不能有 void 的，如果在构造方法名前加了 void，系统就不会自动调用该方法了。

一个类可以创建多个构造方法，当类中包含有多个构造方法时，将根据参数的不同的来决定要用哪个构造方法来初始化新创建对象的状态，达到方法重载的目的。

【例 3-6】 计算长方体的体积。

源程序如下：

```
1.  /* 构造长方体 */
2.  class Box
3.  {
4.    double width, height, depth;
5.    Box()
6.     {
7.         width = 10;
8.         height = 10;
9.         depth = 10;
10.    }
11.   double volume()
12.    {
13.        return width * height * depth;
14.     }
15.  }
16. public class Example3_6
17.  {
18.     public static void main(String args[])
19.      {
20.         Box box = new Box();
21.         double v;
22.         v = box.volume();
23.         System.out.println("长方体体积为: " + v);
24.       }
25.  }
```

Box类的构造方法，与类同名

普通方法,计算长方体体积

应用构造方法创建实例对象

调用普通方法

【程序说明】

程序的第 5 行定义了构造方法，在第 20 行使用构造方法创建实例对象。

在一个类的程序中，也可以没有定义构造方法，则 Java 系统会认为是定义了一个缺省构

造方法，缺省构造方法是无任何内容的空方法。当编写类时，只有在需要进行一些特别初始化的场合，才需要定义构造方法。

【例 3-7】 使用缺省构造方法设计一个计算长方体体积的程序。

源程序如下：

```
1.  /* 缺省构造方法构造长方体类 */
2.    class Box                    ← 该类没有定义构造方法
3.    {
4.      double width, height, depth;
5.      double volume()       //计算长方体体积
6.        {
7.           width = 10;
8.           height = 10;
9.           depth = 10;
10.           return width * height * depth;
11.       }
12.    }
13.   public class Example3_7
14.     {
15.       public static void main(String args[])
16.      {
17.           Box box = new Box();    ← 应用缺省构造方法创建实例对象
18.           double v;
19.           v = box.volume();
20.           System.out.println("长方体体积为: " + v);
21.         }
22.   }
```

【程序说明】

本程序与例 3-6 比较，它们都没有定义构造方法。在程序的第 17 行创建实例对象时，使用系统默认的缺省构造方法。

3.4 对象

在本章的一开始就介绍了类与对象的概念，已知，类是一个抽象的概念，而对象是类的具体化。类与对象的关系相当于普通数据类型与其变量的关系。声明一个类只是定义了一种新的数据类型，类通过实例化创建了对象，才真正创建了这种数据类型的物理实体。

一个对象的生命周期包括三个阶段：创建、使用和释放。

1. 对象的创建

创建对象的一般格式为：

```
类名 对象名 = new 类名([参数列表]);
```

该表达式隐含了对象声明、实例化和初始化三个部分。

1）对象声明

声明对象的一般形式为：

```
类名 对象名;
```

对象声明并不为对象分配内存空间,而只是分配一个引用空间;对象的引用类似于指针,是32位的地址空间,它的值指向一个中间的数据结构,它存储有关数据类型的信息以及当前对象所在的堆的地址,而对于对象所在的实际的内存地址是不可操作的,这就保证了安全性。

2) 实例化

实例化是为对象分配内存空间和进行初始化的过程,其一般形式为:

```
对象名 = new 构造方法();
```

运算符new为对象分配内存空间,调用对象的构造方法,返回引用;一个类的不同对象分别占据不同的内存空间。在执行类的构造方法进行初始化时,可以根据参数类型或个数不同调用相应的构造方法,进行不同的初始化,实现方法重构。

2. 对象的使用

对象要通过访问对象变量或调用对象方法来使用。

通过运算符"."可以实现对对象自己的变量访问和方法的调用。变量和方法可以通过设定访问权限来限制其他对象对它的访问。

1) 访问对象的变量

对象创建之后,对象就有了自己的变量。对象通过使用运算符"."实现对自己的变量访问。

访问对象成员变量的格式为:

```
对象名.成员变量;
```

例如,设有一个A类其结构如下:

```
class A
  { int x; }
```

想对其变量x赋值,则先创建并实例化类A的对象a,然后再通过对象给变量x:

```
A a = new A();
a. x = 5;
```

2) 调用对象的方法

对象通过使用运算符"."实现对自己的方法调用。

调用对象成员方法的格式为:

```
对象名.方法名([参数列表]);
```

例如,在例3-7中,定义了Box类。在Box类中定义了三个double类型的成员变量和一个volume方法,将来每个具体对象的内存空间中都保存有自己的三个变量和一个方法的引用,并由它的volume方法来操纵自己的变量,这就是面向对象的封装特性的体现。要访问或调用一个对象的变量或方法需要首先创建这个对象,然后用算符"."调用该对象的某个变量或方法。

【例3-8】 用带参数的成员方法计算长方体的体积。

```
1.  /* 用带参数的成员方法计算长方体体积 */
2.  import javax.swing.*;
```

```
3.  class Box                                          ← 定义Box类
4.  {
5.    double volume(double width, double height, double depth )   ← 定义有三个形参的方法
6.    {
7.        return width * height * depth;
8.      }
9.  }
10. public class Example3_8
11.   {
12.     public static void main(String args[])
13.     {
14.         double v;
15.         Box box = new Box();                       ← 建立Box类对象box
16.         v = box.volume(3, 4, 5);                   ← 调用对象box的方法,将三个实参传值给形参
17.         JOptionPane.showMessageDialog(null, "长方体体积 = " + v);
18.           System.exit(0);
19.     }
20.  }
```

【程序说明】

(1) 程序的第 5 行在 Box 类定义了一个带有三个参数、返回类型为 double 的 volume 方法。

(2) 程序的第 15 行在 Example3_8 类建立并实例化了 Box 类的对象 box,在第 16 行,对象 box 调用自己的 volume 方法,这里必须对方法中的形参赋确定的数据值(实参)。

程序运行结果如图 3-13 所示。

图 3-13　计算长方体体积

【例 3-9】 用对象作为方法的参数计算圆柱体的体积。

```
1.  /* 对象作为方法的参数使用 */
2.  class 圆面积
3.  {
4.    double r;
5.    圆面积(double r)
6.   {
7.       this.r = r;                                   ← 在this.r=r中,右边的变量r为形参(局部变量),左边的变量this.r为成员变量,即在第4行声明的变量。通过本赋值语句,将外部形参r的值传到对象方法area,从而计算圆的面积
8.   }
9.    double area()
10.    {
11.      return 4.14 * r * r;
12.     }
13. }
14. class  圆柱体积
15. {
16.    圆柱体积(圆面积 a, double h)                    ← 形参a为圆面积类的对象,圆面积类为引用类型
17.    {
18.     double v = a.area() * h;                      ← 调用圆面积类的对象a的area方法与形参h相乘
19.     System.out.println("圆柱体的体积 = " + v);
20.    }
21. }
22. class Example3_9
23. {
24.    public static void main(String args[])
```

```
25.     {
26.         圆面积 A = new 圆面积(5);
27.         圆柱体积 c = new 圆柱体积(A, 10);
28.     }
29. }
```

在main方法中,实例化类对象

【程序说明】

通过本例可以看到,类对象和变量一样,可以作为参数使用。

(1) 程序第 2～第 13 行定义了一个圆面积类,类中有一个计算圆面积的 area 方法。

(2) 程序的第 14～第 21 行定义了一个圆柱体积类,其构造方法以圆面积类的对象 a 为参数,在方法内第 18 行调用对象 a 的 area 方法。

用类对象作为参数使用,即用引用型变量作参数,需要说明的是,引用型变量名不是表示变量本身,而是变量的存储地址。理解这一点很重要,因为如果改变了引用型变量的值,那它就会指向不同的内存地址。在 C 语言中,称其为指针。

3. 释放对象

当不存在对一个对象的引用时,该对象成为一个无用对象。Java 的垃圾收集器自动扫描对象的动态内存区,把没有引用的对象作为垃圾收集起来并释放。但由于垃圾收集器自动收集垃圾操作的优先级较低,因此也可以用其他一些办法来释放对象所占用的内存。

例如,使用系统的

```
System.gc();
```

要求垃圾回收,这时垃圾回收线程将优先得到运行。

另外,还可以使用 finalize 方法将对象从内存中清除。finalize 方法可以完成包括关闭已打开的文件、确保在内存不遗留任何信息等功能。finalize 方法是 Object 类的一个成员方法,Object 类处于 Java 类分级结构的顶部,所有类都是它的子类。其他子类可以重载 finalize 方法来完成对象的最后处理工作。

3.5 面向对象特性

Java 语言中有三个典型的面向对象的特性:封装性、继承性和多态性,下面将详细阐述。

3.5.1 封装性

封装性就是把对象的属性和服务结合成一个独立的相同单位,并尽可能隐蔽对象的内部细节。例如,在银行日常业务模拟系统中,账户这个抽象数据类型把账户的金额和交易情况封装在类的内部,系统的其他部分没有办法直接获取或改变这些关键数据,只有通过调用类中的适当方法才能做到这一点。如调用查看余额的方法来了解账户的金额,调用存取款的方法来改变金额等。只要给这些方法设置严格的访问权限,就可以保证只有被授权的其他抽象数据类型才可以执行这些操作和改变当前类的状态。这样,就保证了数据安全和系统的严密。

在面向对象的程序设计中,抽象数据类型是用“类”这种面向对象的结构来实现的,每个类里都封装了相关的数据和操作。在实际的开发过程中,类在很多情况下用来构建系统内部的模块,由于封装性把类的内部数据保护得很严密,模块与模块间仅通过严格控制的界面进行交互,使模块之间耦合和交叉大大减少。从软件工程的角度看,大大降低了开发过程的复杂性,

提高了开发的效率和质量，也减少了出错的可能性，同时还保证了程序中数据的完整性和安全性。

在 Java 语言中，对象就是对一组变量和相关方法的封装，其中变量表明了对象的状态，方法表明了对象具有的行为。通过对象的封装，实现了模块化和信息隐藏。

1. 修饰符

为了对类对象封装，通过对类成员修饰符施以一定的访问权限，从而实现类中成员的信息隐藏。

类体定义中的一般格式为：

```
class 类名
{
  [变量修饰符]  类型  成员变量名;
  [方法修饰符]  返回类型  方法名(参数){…}
}
```

变量修饰符有[public 或 protected 或 private]、[static]、[final]、[transient]和[volatile]。

方法修饰符有[public 或 protected 或 private]、[static]、[final 或 abstract]、[native]和[synchronized]。

2. 访问权限的限定

在类体成员定义的修饰符可选项中，提供了 4 种不同的访问权限。它们是 private、default、protected 和 public。

1) private

在一个类中被限定为 private 的成员，只能被这个类本身访问，其他类无法访问。如果一个类的构造方法声明为 private，则其他类不能生成该类的一个实例。

2) default

在一个类中不加任何访问权限限定的成员属于缺省的(default)访问状态，可以被这个类本身和同一个包中的类所访问。

3) protected

在一个类中被限定为 protected 的成员，可以被这个类本身、它的子类(包括同一个包中以及不同包中的子类)和同一个包中的所有其他的类访问。

4) public

在一个类中限定为 public 的成员，可以被所有的类访问。

表 3-1 列出了这些限定词的作用范围。

表 3-1　Java 中类的限定词的作用范围

限　定　词	同 一 个 类	同 一 个 包	不同包的子类	不同包非子类
private	√			
default	√	√		
protected	√	√	√	
public	√	√	√	√

3.5.2 继承性

继承性是面向对象程序中两个类之间的一种关系,即一个类可以从另一个类即他的父类继承状态和行为。被继承的类(父类)也可以称为超类,继承父类的类称为子类。继承为组织和构造程序提供了一个强大而自然的机理。

一般来说,对象是以类的形式来定义的。面向对象系统允许一个类建立在其他类之上。例如,山地自行车、赛车以及双人自行车都是自行车,那么在面向对象技术中,山地自行车、赛车以及双人自行车就是自行车类的子类,自行车类是山地自行车、赛车以及双人自行车的父类。

Java 不支持多重继承。

1. 子类的定义

子类定义的一般形式为:

```
class SuperClass                                  ← 设已定义父类
  { …  }
class SubClass extends SuperClass                 ← 定义继承父类的子类
  {  …  }
```

通过继承可以实现代码复用。Java 中所有的类都是通过直接或间接地继承 java.lang.Object 类得到的。继承而得到的类称为子类,被继承的类称为父类(又称超类)。子类不能继承父类中访问权限为 private 的成员变量和方法。子类可以重写父类的方法,及命名与父类同名的成员变量。

【例 3-10】 类的继承性示例,创建一个 A 类和它的子类 B 类,通过子类 B 的实例对象调用从父类 A 继承的方法。

```
1  /* 类的继承 */
2  class  A                    //A 类为父类
3  {
4    void  prnt(int x,int y)
5     {
6       int z = x + y;
7       String  s = "x + y = ";
8       System.out.println(s + z);
9     }
10 }
11 class  B  extends  A        ← B类是A的子类,B类拥有自己定义的
12 {                              bb方法,还拥有继承于父类A的prnt方法
13    String  str;
14    void  bb()
15     {
16        System.out.println(str);
17     }
18  }
19 public  class  Example3_10
20  {
21    public  static  void  main(String args[])
22    {
23       B b = new B();         ← 创建B类的实例对象b
```

```
24        b.str = "Java学习";
25        b.bb();          ← bb()是实例对象b的方法
26        b.prnt(3, 5);    ← prnt()是实例对象b父类A的方法,b继承了该方法
27    }
28 }
```

【程序说明】

(1) 本程序创建了三个类：A、B、Example3_10。其中，A 是超类；B 是 A 的子类；而 Example3_10 是运行程序的主类。

(2) 在程序的第 26 行，类 B 的对象 b 调用了 prnt 方法，但在类 B 中并没有定义这个方法，由于 B 的父类有这个方法，所以 B 的对象 b 可以调用 prnt 方法。

【例 3-11】 创建子类对象时，Java 虚拟机首先执行父类的构造方法，然后再执行子类的构造方法。

```
1  class A
2  {
3    A()
4    { System.out.println("上层父类 A 的构造方法");  }        ← C的上层父类
5  }
6  class B  extends A
7   {
8     B()
9     { System.out.println("父类 B 的构造方法");  }          ← C的父类
10 }
11 class C  extends B
12 {
13   C()
14    { System.out.println("子类 C 的构造方法"); }           ← 子类
15 }
16 public  class  Example3_11
17 {
18   public static void main(String[ ] args)
19   {
20       C  c = new C();     ← 实例化子类对象时,父类的构造方法也会执行
21   }
22 }
```

程序运行结果为：

```
上层父类 A 的构造方法
父类 B 的构造方法
子类 C 的构造方法
```

【程序说明】

从本程序的运行结果可以看出：

(1) 父类的构造方法也会被子类继承，且 Java 虚拟机首先执行父类的构造方法。

(2) 在多继承的情况下，创建子对象时，将从继承的最上层父类开始，依次执行各个类的构造方法。

2. 成员变量的隐藏和方法的重写

子类通过隐藏父类的成员变量和重写父类的方法，可以把父类的状态和行为改变为自身

的状态和行为。

例如：

```
class SuperClass
{
    int x; …
    void setX(){ x = 0; } …
}
class SubClass extends SuperClass
{
    int x;                    //隐藏了父类的变量 x
    ⋮
    void setX()               //重写了父类的方法 setX()
     {
        x = 5;
     }
    ⋮
  }
```

注意：子类中重写的方法和父类中被重写的方法要具有相同的名字，相同的参数表和相同的返回类型，只是方法体不同。

如果子类重写了父类的方法，则运行时系统调用子类的方法；如果子类继承了父类的方法(未重写)，则运行时系统调用父类的方法。

【例 3-12】 子类重写了父类的方法，则在运行时，系统调用子类的方法。

```
1  /* 子类重写了父类的方法 */
2    import java.io.*;
3    class A
4    {
5      void callme()
6       {
7         System.out.println("调用的是 A 类中的 callme 方法");
8       }
9     }
10     class B extends A
11    {
12       void callme()
13        {
14              System.out.println("调用的是 B 类中的 callme 方法");
15        }
16    }
17    public class Example3_12
18    {
19       public static void main(String args[])
20        {
21           A a = new B();
22           a.callme();
23        }
24   }
```

【程序说明】

在程序的第 22 行，父类对象 a 引用的是子类的实例，所以 Java 运行时调用子类 B 的 callme 方法。

程序运行结果为：

```
调用的是 B 类中的 callme 方法
```

3. super 关键字

Java 中通过关键字 super 来实现对父类成员的访问，super 用来引用当前对象的父类。Super 的使用有三种情况：

(1) 访问父类被隐藏的成员变量或方法。

例如：

```
super.variable;
```

(2) 调用父类中被重写的方法。

例如：

```
super.Method([paramlist]);
```

(3) 调用父类的构造方法。由于子类不继承父类的构造方法，当要在子类中使用父类的构造方法时，则可以使用 super 来表示，并且 super 必须是子类构造方法中的第一条语句。

例如：

```
super([paramlist]);
```

再如，设有一个类 A

```
1  class A{
2       A(){ System.out.println("I am A class!");  }
3       A(String str){ System.out.println(str);  }
4     }
```

设 B 是 A 的子类，且 B 调用 A 的构造方法，则有

```
1  class   B extends A  {
2     B()
3       {   super("Hello!");  }      ← 调用父类的构造方法
4     public static void main(String[ ] args)
5     {   new B();  }
6  }
```

运行上述程序，看到 B 调用了 A 的构造方法，运行结果为：

```
Hello!
```

4. this 关键字

this 是 Java 的一个关键字，表示某个对象。this 可以用于构造方法和实例方法，但不能用于类方法。

(1) this 用于构造方法时，代表调用该构造方法所创建的对象。

(2) this 用于实例方法时，代表调用该方法的当前对象。

this 的使用格式为：

```
this.当前类的成员方法();
```

或

```
this.当前类的成员变量;
```

下面是一个使用关键字 this 的例子。由于成员变量与函数的局部变量同名,在函数中为了分辨成员变量,则在成员变量前面用关键字 this 加以说明。

```
1  class B
2  {
3    int s, i;
4    int  mysum(int s)
5    {
6      for(i = 1; i<=s; i++)
7          this.s = this.s + i;
8      return this.s;
9    }
10 }
```

定义成员变量s

函数中的形参s(局部变量)与成员变量s同名

为了分辨成员变量s,使用this.s,表示当前类的成员变量

3.5.3 多态性

在面向对象程序中,多态是一个非常重要的概念。多态是指一个程序中同名的方法共存的情况,有时需要利用这种“重名”现象来提供程序的抽象性和简洁性。如前所述,一个实例由类生成,将实例以某种方式连接起来,就为模块提供了所需要的动态行为。模块的动态行为是由对象间相互通信而达成的,多态的含义是一个消息可以与不同的对象结合,而且这些对象属于不同类。同一消息可以用不同方法解释,方法的解释依赖于接收消息的类,而不依赖发送消息的实例。多态通常是一个消息在不同的类中用不同方法实现的。

多态的实现是由消息的接收者确定一个消息应如何解释,而不是由消息的发送者确定,消息的发送者只需要指导另外的实例可以执行一种特定操作即可,这一特性对于扩充系统的开发是特别有效的。按这种方法可开发易于维护、可塑性好的系统。例如,如果希望加一个对象到类中,这种维护只涉及新对象,而不涉及给它发送消息的对象。

在 Java 语言中,多态性体现在两个方面:由方法重载实现的静态多态性(编译时多态)和方法重写实现的动态多态性(运行时多态)。

1. 编译时多态

在编译阶段,具体调用哪个被重载的方法,编译器会根据参数的不同来静态确定调用相应的方法。

2. 运行时多态

由于子类继承了父类所有的属性(私有的除外),所以子类对象可以作为父类对象使用。程序中凡是使用父类对象的地方,都可以用子类对象代替。一个对象可以通过引用子类的实例调用子类的方法。

3.5.4 其他修饰符的用法

1. final 关键字

final 关键字可以修饰类、类的成员变量和成员方法,但对其作用是各不相同的。

1）final 修饰成员变量

可以用 final 修饰变量，目的是为了防止变量的内容被修改，经 final 修饰后，变量就成为了常量。其形式为：

```
final 类型 变量名;
```

当用 final 修饰成员变量时，在定义变量的同时就要给出初始值。

2）final 修饰成员方法

当一个方法被 final 修饰后，则该方法不能被子类重写。其形式为：

```
final 返回类型 方法名(参数)
 {
   ⋮
}
```

3）final 修饰类

一个类用 final 修饰后，则该类不能被继承，即不能成为超类。其形式为：

```
final class 类名
{
    ⋮
}
```

2. static 关键字

在 Java 语言中，成员变量和成员方法可以进一步分为两种：类成员和实例成员。用关键字 static 修饰的变量或方法就称为类变量和类方法。没有用关键字 static 修饰的变量或方法就称为实例变量或实例方法。

用 static 关键字声明类变量和类方法的格式如下：

```
static 类型 变量名;
static 返回类型 方法名(参数)
{
    ⋮
}
```

如果在声明时用 static 关键字修饰，则声明为静态变量和静态方法。在调用静态方法时，不要进行实例化而直接调用。

【例 3-13】 说明静态方法的调用。

```
1  /* 静态方法的调用   */
2  class B
3  {
4   public static void p()
5     {
6      System.out.println("I am B!");
7     }
8  }
9  class  Example3_13
10  {
11   public static void main(String[] args)
12     {
13       B.p();
```

```
14      }
15   }
16 /* 如果类 B 中的 p()没有声明为 static,则必须在 Example3_13 中对 B 进行实例化,否则编译不能
17 通过
18  class B{
19      public  void p(){
20         System.out.println("I am B!");
21        }
22    }
23  class  Example3_13  {
24      public static void main(String[] args) {
25         B b = new B();
26         b.p();
27       }
28     }
29   */
```

【程序说明】

(1) 在程序的第 2～第 8 行定义了 B 类,其方法 p()是用 static 修饰的静态方法,所以类 Example3_12 在第 13 行可以直接调用 p 方法,而没有对 B 进行实例化。

(2) 在程序的第 19～第 28 行,说明了由于类 B 中的 p 方法没有声明为 static,所以它是实例方法,因此在 Example3_12 调用时,必须先对 B 进行实例化(第 25 和第 26 行)。

3. 类成员与实例成员的区别

类成员与实例成员的区别如下:

1) 实例变量和类变量

每个对象的实例变量都分配内存,通过该对象来访问这些实例变量,不同的实例变量是不同的。

类变量仅在生成第一个对象时分配内存,所有实例对象共享同一个类变量,每个实例对象对类变量的改变都会影响其他的实例对象。类变量可通过类名直接访问,无须先生成一个实例对象,也可以通过实例对象访问类变量。

2) 实例方法和类方法

实例方法可以对当前对象的实例变量进行操作,也可以对类变量进行操作,实例方法由实例对象调用。

但类方法不能访问实例变量,只能访问类变量。类方法可以由类名直接调用,也可由实例对象进行调用。类方法中不能使用 this 或 super 关键字。

【例 3-14】 实例成员和类成员的区别。

```
1  class Member
2     {
3        static int classVar;
4        int instanceVar;
5        static void setClassVar(int i)
6          {
7            classVar = i;
8            //instanceVar = i;          //该语句错误,因为类方法不能访问实例变量
9          }
10        static int getClassVar()
11          {
```

```
12              return classVar;
13            }
14          void setInstanceVar(int i)
15            {
16              classVar = i;              //实例方法不但可以访问类变量,也可以访问实例变量
17             instanceVar = i;
18            }
19          int getInstanceVar()
20            {
21              return instanceVar;
22       }
23    }
24    public class Example3_14
25    {
26       public static void main(String args[])
27        {
28          Member m1 = new Member();
29          Member m2 = new Member();
30          m1.setClassVar(1);
31          m2.setClassVar(2);
32          System.out.println("m1.classVar = " + m1.getClassVar()
33                                    + "m2.ClassVar = " + m2.getClassVar());
34          m1.setInstanceVar(11);
35          m2.setInstanceVar(22);
36          System.out.println("m1.InstanceVar = " + m1.getInstanceVar()
37                           + "m2.InstanceVar = " + m2.getInstanceVar());
38       }
39    }
```

4. abstract 关键字(抽象类)

Java 语言中,用关键字 abstract 来修饰一个类时,这个类叫做抽象类。

抽象类往往用来描述对问题领域进行分析、设计中得出的抽象概念,是对一系列看上去不同,但是本质上相同的具体概念的抽象。例如,如果进行一个图形编辑软件的开发,就会发现问题领域存在着圆、三角形这样一些具体概念,从外形上看它们是不同的,但从概念方面来看,但是它们又都属于形状这样一个概念。形状这个概念在问题领域是不存在的,是一个抽象概念。正是因为抽象的概念在问题领域没有对应的具体概念,所以用以表征抽象概念的抽象类是不能实例化的。

抽象类的式如下:

```
abstract class 类名            //定义抽象类
{
   成员变量;
   方法();                     //定义普通方法
   abstract 方法();            //定义抽象方法
}
```

抽象类是专门设计用来让子类继承的类。用 abstract 关键字来修饰一个方法时,这个方法叫做抽象方法。抽象类有以下特点:

(1) 抽象类中可以包含普通方法,也可以包含抽象方法;可以只有普通方法,没有抽象方法。

(2) 抽象类中的抽象方法是只有方法声明,没有代码实现的空方法。

(3) 抽象类不能被实例化。

(4) 若某个类包含了抽象方法,则该类必须被定义为抽象类。

(5) 由于抽象方法都是没有完成代码实现的空方法,因此抽象类的子类必须重写父类定义的每一个抽象方法。

【例 3-15】 抽象类应用示例。

```
1  /* 抽象类 */
2  abstract  class  生物
3  {
4     public  abstract String 特征();
5   }
6
7  class  植物  extends  生物          //植物是生物的子类
8   {
9        String leaf;
10     植物(String _leaf)
11         {this.leaf = _leaf;}
12     public String 特征()
13          {  return leaf;  }
14  }
15
16 class  动物  extends  生物          //动物是生物的子类
17 {
18    String mouth;
19    动物(String _mouth)
20     {  this.mouth = _mouth;  }
21    public String 特征()
22     {    return mouth;       }
23 }
24
25 public  class  Example3_15
26  {
27     public  static  void  main(String args[])
28      {
29          植物 A = new 植物("叶");
30         System.out.println("植物的特征: " + A.特征());
31          动物 B = new 动物("嘴巴");
32         System.out.println("动物的特征: " + B.特征());
33      }
34   }
```

生物类为抽象类,定义抽象方法

子类重写父类定义的抽象方法

子类重写父类定义的抽象方法

程序运行结果为:

```
植物的特征: 叶
动物的特征: 嘴巴
```

3.6 接口

3.6.1 接口的定义

接口是抽象类的一种,只包含常量和方法的定义,而没有变量和具体方法的实现,且其方

法都是抽象方法。它的用处体现在下面几个方面：

(1) 通过接口实现不相关类的相同行为，而无须考虑这些类之间的关系。

(2) 通过接口指明多个类需要实现的方法。

(3) 通过接口了解对象的交互界面，而无须了解对象所对应的类。

1. 接口定义的一般形式

接口的定义包括接口声明和接口体。

接口定义的一般形式如下：

```
[public] interface 接口名[extends 父接口名]
{
  ⋮     //接口体
}
```

extends 子句与类声明的 extends 子句基本相同；不同的是一个接口可有多个父接口，用逗号隔开，而一个类只能有一个父类。

2. 接口的实现

在类的声明中用 implements 子句来表示一个类使用某个接口，在类体中可以使用接口中定义的常量，而且必须实现接口中定义的所有方法。一个类可以实现多个接口，在 implements 子句中用逗号分开。

【例 3-16】 设原来拟编写一个超类 Prnting，其中定义了一个 prnt 方法供子类继承，现将其改成接口，由子类实现该接口。

```
/* 接口的使用
//设拟编写一个超类 Prnting,其中定义了一个 prnt 方法
class  Prnting
{
    void  prnt()
      {
        System.out.println("蔬菜和水果都重要");
      }
}  */
  //下面要将其改成接口
  //将一个类改成接口,只要把 class 换成 interface,再把其所有方法的内容都抽掉
  interface Prnting
  {
     void  prnt();
  }
  public class Example3_16 implements Prnting          //实现接口,重写 prnt 方法
  {
    public void prnt()                                 //注意,public 不能缺少
     {
         System.out.println("蔬菜重要。");
     }
  }
```

3.6.2　接口的应用

接口的定义及语法规则很简单，但真正要理解接口就不那么容易。从例 3-15 可以看到，

可以用一个类来完成同样的事情。那么,为什么要用接口呢?

例如,项目开发部需要帮助一个用户单位编写管理程序,项目主管根据用户需求,把应用程序需要实现的各个功能都做成接口形式,然后分配给项目组其他成员编写出具体的功能。

【例 3-17】 管理程序具有查询数据、添加数据、删除数据等功能接口的应用示例。

```
1  interface DataOption                                    定义接口
2  {
3    public void dataSelect();                             //查询数据
4    public void dataAdd();                                //添加数据
5    public void dataDel();                                //删除数据
6  }
7  class DataManagement implements DataOption              实现接口的类
8  {
9     public void dataSelect()                             编写接口dataSelect()方法的具体代码
10    {
11      System.out.println("查询数据");
12    }
13    public void dataAdd()                                 编写接口dataAdd()方法的具体代码
14    {
15      System.out.println("添加数据");
16    }
17    public void dataDel()                                 编写接口dataDel()方法的具体代码
18    {
19      System.out.println("删除数据");
20    }
21 }
22 public class Example3_17
23 {
24   public static void main(String args[])
25   {
26     DataManagement data = new DataManagement();
27     data.dataSelect();
28     data.dataAdd();
29     data.dataDel();
30   }
31 }
```

【例 3-18】 编写一个接口程序,其中定义一个计算面积的方法。然后,再设计应用程序实现这个接口,分别计算矩形面积和圆的面积。

```
1   interface Area                                          定义接口
2  {
3    public double area();
4  }
5  class A implements Area                                 定义实现接口的A类
6  {
7     int x, y;
8     void set_xy(int x, int y)
9     {
10      this.x = x;
11      this.y = y;
12    }
13    public double area()                                  编写接口area()方法的具体代码
14    {
```

```
15        double s;
16        s = x * y;
17        return s;
18      }
19 }
20 class B implements Area          ← 定义实现接口的B类
21 {
22   int r;
23   void set_r(int r)
24   {
25        this.r = r;
26   }
27   public double area()           ← 编写接口area()方法的具体代码
28   {
29     double s;
30     s = 3.14 * r * r;
31     return s;
32   }
33 }
34
35 class ex3_18
36 {
37   public static void main(String args[])
38   {
39     int x = 10, y = 5;
40     double ss1, ss2;
41     A aa = new A();
42     aa.set_xy(x,y);
43     ss1 = aa.area();
44     System.out.println("矩形面积 = " + ss1);
45     int r = 5;
46     B bb =  new B();
47     bb.set_r(r);
48     ss2 = bb.area();
49     System.out.println("圆面积 = " + ss2);
50   }
51 }
```

3.7　包

在 Java 语言中，每个类都会生成一个字节码文件，该字节码文件名与类名相同。这样，可能会发生同名类的冲突。为了解决这个问题，Java 采用包来管理类名空间。包不仅提供了一种类名管理机制，也提供了一种面向对象方法的封装机制。包将类和接口封装在一起，方便了类和接口的管理与调用。例如，Java 的基础类都封装在 java.lang 包中，所有与网络相关的类都封装在 java.net 包中。程序设计人员也可以将自己编写的类和接口封装到一个包中。

3.7.1　创建自己的包

1. 包的定义

为了更好地组织类，Java 提供了包机制。包是类的容器，用于分隔类名空间。把一个源

程序归入到某个包的方法用 package 来实现。

package 语句的一般形式为：

```
package 包名;
```

包名有层次关系，包名的层次必须与 Java 开发系统的文件系统目录结构相同。简言之，包名就是类所在的目录路径，各层目录之间以“.”符号分隔。通常包名用小写字母表示，这与类名以大写字母开头的命名约定有所不同。

在源文件中，package 是源程序的第一条语句。

例如，要编写一个 MyTest.java 源文件，并且文件存放在当前运行目录的子目录 abc\test 下，则

```
package abc.test;
public class MyTest
{
  ⋮
}
```

在源文件中，package 是源程序的第一条语句。包名一定是当前运行目录的子目录。一个包内的 Java 代码可以访问该包的所有类及类中的非私有变量和方法。

2. 包的引用

要使用包中的类，必须用关键字 import 导入这些类所在的包。

import 语句的一般形式为：

```
import 包名.类名;
```

当要引用包中所有的类或接口时，类名可以用通配符“*”代替。

【例 3-19】 创建一个自己的包。

本例所创建包的文件目录结构如图 3-14 所示。

图 3-14 包的文件目录结构

(1) 在当前运行目录下创建一个子目录结构 abc\test，在子目录下存放已经编译成字节码文件的 MyTest.class 类，其 MyTest.class 类的源程序为：

```
1  package abc.test;   ← 定义包
2  public class MyTest
3  {
4    public void prn()
5    {
6      System.out.println("包的功能测试");
7    }
8  }
```

(2) 在当前目录的 PackageTest.java 中，要使用子目录 abc\test 下有 MyTest.class 类中的 prn 方法，则其源程序为：

```
1  import abc.test.MyTest;   ← 引用包
```

```
2  public class PackageTest
3  {
4    public static void main(String args[])
5   {
6        MyTest mt = new Mytest();   ← 调用包中的类
7        mt.prn();
8      }
9  }
```

3.7.2　压缩文件 jar

1. 将类压缩为 jar 文件

在 Java 提供的工具集 bin 目录下有一个 jar.exe 文件，它可以把多个类的字节码文件打包压缩成一个 jar 文件，然后将这个 jar 文件存放到 Java 运行环境的扩展框架中，即将该 jar 文件存放在 JDK 安装目录的 jre\lib\ext 下，这样，其他的程序就可以使用这个 jar 文件中的类来创建对象了。

设有两个字节码文件 Test1.class 和 Test2.class，要将它们压缩成一个 jar 文件 Test.jar。

1）编写 Manifest.mf 清单文件

```
Manifest-Version: 1.0
Main-Class:  Test1  Test2
```

注意：Main-Class 与后面的类名之间要有一个空格，且最后一行要回车换行。将其保存为 Manifest.mf。

2）生成 jar 文件

```
Jar  cfm  Test.jar  Manifest.mf  Test1.class  Test2.class
```

其中，参数 c 表示要生成一个新的 jar 文件；f 表示要生成 jar 文件的文件名；m 表示清单文件的文件名。

2. 将应用程序压缩为 jar 文件

可以用 jar.exe 将应用程序生成可执行文件。在 Windows 环境下，双击该文件，就可以运行该应用程序。

其生成 jar 文件的步骤与前面生成类的 jar 文件相同。当要压缩多个类时，在清单文件中只写出主类的类名，设有主类 A.class，则：

```
Manifest-Version: 1.0
Main-Class: A
```

生成 jar 文件时，也可以使用通配“*.class”。

```
jar  cfm  Test.jar  Manifest.mf  *.class
```

需要注意的是，如果计算机上安装了 WinRAR 解压软件，则无法通过双击该文件的办法执行程序，可以编写一个批处理文件：

```
javaw  -jar Test.jar
```

将其保存为 test.bat，双击这个批处理文件就可以运行应用程序。

实验 3

【实验目的】

(1) 掌握类的声明,对象的创建以及方法的定义和调用。

(2) 掌握打包机制。

(3) 掌握类的继承。

(4) 掌握类接口的使用。

【实验内容】

(1) 运行下列程序,并写出其输出结果。

```
//Father.java:
package tom.jiafei;
public class  Father
{   int  height;
   protected int money;
   public   int weight;
   public Father(int m)
   {  money = m;
   }
   protected int getMoney()
   {  return money;
   }
    void setMoney(int newMoney)
   {  money = newMoney;
   }
}

//Jerry.java:
import tom.jiafei.Father;
public class Jerry extends Father                          //Jerry 和 Father 在不同的包中
{  public Jerry()
   {   super(20);
   }
   public static void main(String args[])
   {  Jerry  jerry = new Jerry();
      jerry.height = 12;                                   //非法的
      jerry.weight = 200;
      jerry.money = 800;
      int m = jerry.getMoney();
      jerry.setMoney(300);                                 //非法的
      System.out.println("m = " + m);
   }
}
```

(2) 运行下列程序,并写出其输出结果。

```
interface  ShowMessage
{   void 显示商标(String s);
}
class TV implements ShowMessage
{  public void 显示商标(String s)
```

```
    {  System.out.println(s);
    }
}
class PC implements ShowMessage
{  public void 显示商标(String s)
    {  System.out.println(s);
    }
}
public class Ex3_2
{  public static void main(String args[])
    {  ShowMessage sm;                                    //声明接口变量
       sm = new TV();                                     //接口变量中存放对象的引用
       sm.显示商标("长城牌电视机");                        //接口回调
       sm = new PC();                                     //接口变量中存放对象的引用
       sm.显示商标("联想奔月 5008PC 机");                  //接口回调
    }
}
```

(3) 编写一个应用程序,求 50 以内的素数,并将其打包,另写一个应用程序调用该包,在以下三种情况下,实现类之间的调用

① 在同一目录下。

② 在不同目录下。

③ 在不同盘符下。

习题 3

(1) 什么是 Java 程序使用的类? 什么是类库?

(2) 如何定义方法? 在面向对象程序设计中方法有什么作用?

(3) 简述构造方法的功能和特点。下面的程序片断是某学生为 student 类编写的构造方法,请指出其中的错误。

```
void Student(int no, String name)
  {
    studentNo = no;
    studentName = name;
    return no;
  }
```

(4) 定义一个表示学生的类 student,包括的成员变量有学号、姓名、性别、年龄,包括的成员方法有获得学号、姓名、性别、年龄和修改年龄。书写 Java 程序创建 student 类的对象及测试其方法的功能。

(5) 扩充、修改程序。为第(4)题的 student 类定义构造方法初始化所有的成员,增加一个方法 public String printInfo(),该方法将 student 类对象的所有成员信息组合形成一个字符串,并在主类中创建学生对象及各方法的功能。

(6) 什么是修饰符? 修饰符的种类有哪些? 它们各有什么作用?

(7) 什么是抽象类,为什么要引入抽象类的概念?

(8) 什么是抽象方法? 如何定义、使用抽象方法?

(9) 包的作用是什么? 如何在程序中引入已定义的类? 使用已定义的用户类、系统类有

哪些主要方式?

(10) 什么是继承? 如何定义继承关系?

(11) 什么是多态? 如何实现多态?

(12) 解释 this 和 super 的意义和作用。

(13) 什么是接口? 为什么要定义接口? 接口和类有什么异同?

(14) 将一个抽象类改写成接口,并实现这个接口。

(15) 编写一个程序实现包的功能。

(16) 填空:

① 如果类 A 继承了类 B,则类 A 被称为_____类,类 B 被称为_____类。

② 继承使_____成为可能,它节省了开发时间。

③ 如果一个类包含一个或多个 abstract 方法,它就是一个_____类。

④ 一个子类一般比其超类封装的功能要_____。

⑤ 标记成_____类的成员不能由该类的方法访问。

⑥ Java 用_____关键字指明继承关系。

⑦ this 代表了_____的引用。

⑧ super 表示的是当前对象的_____对象。

⑨ 抽象类的修饰符是_____。

⑩ 接口中定义的数据成员是_____。

⑪ 接口中没有什么_____方法,所有的成员方法都是_____方法。

(17) 编写一个接口程序,其中定义一个计算体积的方法。然后,再设计应用程序实现这个接口,分别计算矩形柱面体积和圆形柱面体积。

(18) 编写一个 Plus 类,计算 2+4+6+8+…+100 的和,并将其打包,另写一个应用程序调用该包,在以下两种情况下,实现类之间的调用。

① Plus 类与应用程序在同一目录下;

② Plus 类在应用程序所在目录下的子目录 com/examp/plus/ 下。

(19) 编写一个应用程序,用 jar.exe 将类压缩为 jar 可执行文件。