

第 3 章

CHAPTER 3

Java 语言面向对象的特征

Java 语言是一种完全的面向对象的程序设计语言,它支持面向对象的程序设计方法。本章简单介绍面向对象的基本概念;详细介绍 Java 语言中类和对象的定义及使用方法,介绍继承和多态两个重要特性,介绍包和接口的概念;并介绍几个常用的工具类。

通过本章的学习,读者可以全面了解 Java 语言的面向对象特征,进一步掌握使用 Java 语言的编程方法和编程技巧。

3.1 面向对象的基本概念

面向对象程序设计是当代计算机技术发展的一个里程碑,它用一种全新的程序设计思路来处理实际问题。与面向过程的程序设计方法不同,面向对象的程序设计方法更符合人们认识客观世界的思维习惯,降低了问题难度和分解问题的复杂性,提高了程序代码的重用性,简化了编程的过程,可以以较小的代价和较高的效率获得满意的效果。实践证明,面向对象程序设计方法是软件开发的新趋势。

面向对象的程序设计方法涉及类、对象、封装、继承和多态等基本概念。搞清这些基本概念,有助于对面向对象程序设计的理解,有助于对 Java 语言的掌握。本节主要介绍这些概念。

3.1.1 对象和类

面向过程的程序设计是以研究和实现解题过程为主体,而面向对象的程序设计是以研究各种对象为主体。

对象是面向对象程序设计的核心。面向对象的程序是由若干对象组成的,通过对象的相互作用来实现整个程序的功能。

类是面向对象程序设计的关键。类是同种类型的对象的集合,它是对所要研究的问题中的客观事物的抽象。

1. 对象

什么是对象？简单地说，对象就是对客观世界中的实体的映射。例如，计算机是客观世界中真实存在的一种实体。计算机有主机、外设等硬件设备，每种设备具有形状、大小、颜色等外部特性；计算机内装有各种软件，具有其内部功能。将计算机的外部特性和内部功能抽象出来便是一种对象。

对象具有 3 个要素或称为 3 大特性。它们分别是状态、行为和标志。

对象的状态是对象的静态属性，通常用变量来表示该对象内部的各种信息。每个对象应有其自己的内部变量，这些变量的值表示了该对象所处的状态。当对象的变量值发生了改变，则表示该对象状态发生了变化。例如，对于一台计算机主机，表示它的状态的变量有型号、大小、颜色等。

对象的行为是对象的动态属性，又称对象的操作。对象通过某种操作来设置或改变对象的状态。例如，计算机的开机或关机便是一种操作。对象的行为在计算机中用方法来表示，方法便是一种函数，对象的行为或操作被定义在函数体内。面向对象的方法中，对象内部包含了描述状态的各种属性和对其属性进行操作的若干方法。另外，还应有一个对象与其他对象进行通信的接口，以便用外部环境来改变对象的状态。

对象的标志是用来区分不同对象的标识符。每一个对象都具有一个仅属于它的唯一的标志。对象的标志是用对象名来表示的。

对象的上述 3 大属性：状态、行为和标志，在计算机实现中分别用对象的变量、对象的方法和对象名来表示。这便是对象的 3 大要素。

2. 类

什么是类？简单地讲，类是一种自定义的数据类型，是同种对象的集合。类是对现实世界中的实体的高度抽象。现实世界中不同的客观实体之间存在有共性，为处理问题方便，将客观实体的公共属性抽象为类，而属于类的某一个对象被称为类的一个实例。例如，一张扑克牌被抽象为点数和花色，定义一个类包含有点数和花色的变量和对它们的操作，而每一张扑克牌就是这个类的一个对象，即为一个实例。

类和对象是一般与个别的关系。类是一个模板，由它可以产生具有某种共同特性的若干个不同的对象。对象又是现实世界中实体的映射，类是对具有共同特性的实体的抽象。使用面向对象的方法解决现实世界的问题时，先将现实世界中的实体抽象成为概念世界的数据类型，抽象的数据类型映射了实体中与需要解决的问题相关的属性。再将这抽象的数据类型通过计算机世界的逻辑表现出来，这便是面向对象方法中的类，最后将类实例化为所需的对象。

综上所述，用面向对象方法来解决问题，更接近于人类的自然思维模式；更接近于人类认识世界的客观规律；并且更能很好地反映客观事物的本来面目。因此，面向对象的程序设计语言更接近于自然语言，使用面向对象的方法会使软件的开发过程更快捷、更高效。

3.1.2 封装性

封装性是指将描述某种实体的数据和基于这些数据的操作集合到一起,形成一个封装体。在该封装体内,数据受到保护,通过与外界的接口实现信息交流。因此,封装性具有如下3个特点:

- (1) 将被描述的实体的属性和行为封装在一起,看作是一个不可分隔的整体,构成程序中不可分隔的独立单位。
- (2) 被封装的某些信息在该封装体外是不可见的,只能通过内部方法来改变它,通常称这些信息被隐藏了。
- (3) 在被封装的属性和行为中,规定了有些被隐藏在封装体内,有些外部是可见的不同访问权限。

面向对象的方法中,类是抽象的数据类型,它是对现实世界中实体抽象的结果。由于类中成员授予不同的访问权限,这可以保证只有被授权的成员才可以访问某种变量,进而改变其状态,这样就保证了数据的安全性和系统的严密性。

封装使得抽象的数据类型提高了可重用性。面向对象的方法把提高可重用性作为一个追求的目标。由于封装使其类成为一个完整的整体,对外接口单一,可适应各种环境;因此,封装性提高了可重用性。

3.1.3 继承性

继承性是面向对象方法中的重要特性。继承反映了两个类之间的一种关系。当一个类拥有另一个类的所有数据和方法时,称这个类继承了另一个类,这两个类具有继承的关系。被继承的类称为父类或基类,继承父类所有成员的类称为子类或派生类。

父类生成子类是创建新类的一种方法。一个类可以同时生成多个子类,父类和子类的继承关系,实际上是事物的一般性和特殊性的关系。这种关系符合了人们对客观事物认识过程和思维方法。例如,现实生活中存在有各种汽车,将它们抽象为一个类,即汽车类,该类包含了汽车所共有的属性和行为。在汽车类的基础上生成一个新类,即轿车类。轿车类是汽车类的子类。轿车类包含了汽车类中所有的属性和行为,并且还包含轿车类所特有的属性和行为。轿车类继承了汽车类,在定义轿车类时,不必再重复汽车类中已有的属性和行为,这便大大地提高了程序代码的重用性,同时提高了软件的开发效率和安全性以及可扩展性。

继承可分为两种:单重继承和多重继承。单重继承是指只有一个父类的继承;多重继承是指有一个以上父类的继承。单重继承的程序结构比较简单,采用单纯的树状结构;而多重继承的程序结构相当复杂,其结构是网状的。实际中,有些面向对象的程序设计语言,例如C++语言,支持单重继承和多重继承两种继承;而Java语言只支持单重继承,不支持多重继承,但是,Java语言可通过接口(interface)方式弥补由于不支持多重继承而带来的子类不可使用多个父类成员的不足。

3.1.4 多态性

多态性也是面向对象方法的一个重要特性。多态性是指在程序中出现的“重名”现象。在面向对象的程序设计中,为提高程序的抽象程度和简洁程度,而出现“重名”现象。

在 Java 语言中,多态性主要表现在如下两个方面。

(1) 方法重载。通常指在同一个类中,相同的方法名对应着不同的方法实现,但是方法的参数不同。

(2) 成员覆盖。通常是指在不同类(父类和子类)中,允许有相同的变量名,但是数据类型不同;也允许有相同的方法名,但是对应的方法实现不同。

在重载的情况下,同一类中具有相同名字的方法,如何选择它的方法体呢?不能使用类名来区分,通常采用不同的方法形参表,区分重载要求形参在类型、个数和顺序的不同。在定义重载方法时,应在方法的形参的类型、个数和顺序有所不同,以便在选择时能够区别开来。

在覆盖情况下,同名的方法存在于不同的类中,在调用方法只要指明其方法所归属的类名就可以了。通过对对象选择其方法通常是动态的。

综上所述,多态性的特点将提高程序的抽象程度和简洁性,并且最大程度地降低程序的模块之间的耦合性。

3.2 类的定义

本节介绍类的定义格式、类的成员,以及类中变量和方法应遵循的规则,进而熟悉 Java 语言中类的定义方法和使用规则。

3.2.1 类的定义格式

前面介绍过,类是若干种属性和服务的封装体,即是变量和方法的集合。

定义类的格式如下所示:

```
[<修饰符>] class <类名> [<extends <父类名>] [<implements <接口名>]
{
    <类的成员变量说明>
    <类的方法定义>
}
```

其中, class, extends 和 implements 都是定义类所用的关键字。

在<类名>前用关键字 class,说明其后是一个新的类名。类名的命名方法同标识符,通常通过类的名字可以看出该类的功能或作用。

在<父类名>前用关键字 extends,用来说明定义的新类是哪个已存在的类的子类。这个已存的父类可以是 Java 语言类库中定义的类;也可以是编程者已定义好的类。通过关键字 extends 和<父类名>可以知道新类已拥有了父类中的哪些变量和方法。

在<接口名>前用关键字 implements,说明当前定义的新类中实现哪个接口所定义的

功能和方法。接口是Java语言实现多重继承的一种特殊机制,将在3.5.1节中讲述。

〈修饰符〉是用来说明类的作用域和其他性质的。类的修饰符有如下4种。

(1) public。说明该类为公共类,它可以被其他类所使用。Java语言规定,程序中主类必须是公共类。

(2) 默认说明。规定该类只能被同一个包中的类所使用,而不能被其他包中的类所访问。

(3) abstract。使用该修饰符的被称为抽象类。抽象类是一种没有具体对象的概念类。通常抽象类是它的所有子类的公共属性的集合。

(4) final。说明该类不可能有子类,故称为最终类。这种类的特点是用来完成某种标准功能的类。final和abstract不能用来同时修饰一个类。

以上是类头,包含〈修饰符〉、class和〈类名〉、extends和〈父名〉,以及implements和〈接口名〉。

下面介绍类体,类的定义是由类头和类体这两部分组成的。

类体中定义了类的具体内容,包括类的属性和服务,即类的变量和方法。

【例3-1】列举一个简单的类。Exam3_1.java程序定义一个描述学生3门课成绩的类,具体定义如下:

```
class Student
{
    String name,stuNumber;
    double score1,score2,score3;
    void set1(String s1,String s2)
    {
        name=s1;
        stuNumber=s2;
        System.out.println(name+" "+stuNumber);
    }
    double setScore(double d1,double d2,double d3)
    {
        double d;
        score1=d1;
        score2=d2;
        score3=d3;
        d=d1+d2+d3;
        return d;
    }
}
```

3.2.2 变量

Java语言中,类的成员分为成员变量和方法两种。成员变量是被说明在类体内,作为类的一种属性;而方法是被说明在类体内的另一种成员。

下面介绍成员变量的相关规定。

成员变量可以是简单变量,也可以是某个类的对象,还可以是数组和其他复杂数据结构。

1. 成员变量的定义格式

简单成员变量的定义格式如下:

[<修饰符>] <变量类型> <变量名>=[<初值>];

其中,<变量名>同标识符,尽量做到“见名知义”; <变量类型>通常指变量的基本数据类型。<变量名>和<变量类型>是不可省略的。

对象成员的定义格式如下:

[<修饰符>] <类名> <对象名>[=new <类名>(<实参表>)];

其中,<类名>是另一个类的名字,Java 语言规定:一个类体内可以包含另一个类的对象。当类中含有其他类的对象时,反映了两个类之间的包含关系。对一个类中包含另一个类的对象可以创建它时进行初始化,也可以用类的相关方法创建它。

2. 变量的修饰符

变量的<修饰符>有访问控制修饰符(5 种)和非访问控制修饰符(3 种),合计共有 8 种。分别介绍如下。

1) 访问控制修饰符(5 种)

(1) 公有访问控制符 public。具有该修饰符的变量称为公共变量。如果它属于一个公共类,则它可以被所有类所访问。

(2) 默认访问控制符 类中变量如无访问控制符,则具有包访问性,可被同一个包的其他类所访问。

(3) 私有访问控制符 private。使用该修饰符说明的变量仅可被该类自身访问,任何其他类都不可访问,包括该类的子类。

(4) 保护访问控制符 protected。使用该修饰符说明的变量可被该类自身、同包的其他类和其他包中该类的子类所访问。该修饰符的特点是允许其他包中该类的子类所访问。

(5) 私有保护访问控制符 private protected。使用该修饰符说明的变量可被该类自身和该类的所有子类所访问,不在同一包中的子类也可访问。

2) 非访问控制修饰符(3 种)

(1) 静态变量 static。静态变量的特点是属于类的,而不是属于某个对象的。静态变量被系统存放在内存的一个公共存储单元中,任何对象都可以访问它,也可以修改它。一旦被修改后,将保持被修改后的内容直到下次被修改为止。静态变量的引用可以使用类名,也可以使用对象名。

(2) 最终变量 final。最终变量就是 Java 语言中的符号常量。使用 final 说明的成员变量是一种符号常量。最终变量的值在程序的整个执行过程中是不会改变的。

最终变量的说明格式如下：

[<修饰符>] final <类型> <变量名>=<初值>;

最终常量通常被说明为 static 的。

使用符号常量可使程序更加易读,便于修改和维护。

(3) 易失变量 volatile。被 volatile 修饰的变量可能同时被多个线程所控制和修改。这种变量在运行过程中可能被其他未知因素来改变,在使用中要特别留心这些影响因素。

【例 3-2】列举一个关于不同变量的例子。通过分析 Exam3_2.java 程序,回答所提出的问题。

(1) 程序 Exam3_2.java:

```
class var
{
    static int a;
    int b;
    public void intprint()
    {
        int c=0;
        ++a;
        ++b;
        ++c;
        System.out.print("a=" + a);
        System.out.print("—b=" + b);
        System.out.println("—c=" + c);
    }
    public void allprint()
    {
        intprint();
        intprint();
    }
}
public class Exam3_2
{
    public static void main(String args[])
    {
        var v1=new var();
        var v2=new var();
        v1.allprint();
        v2.allprint();
    }
}
```

该程序是由两个类组成的,其中有一个含有 main()方法的主类。

通过分析该程序,回答如下 4 个问题:

- ① 类 var 中定义了两个变量 a 和 b, 这两个变量有什么不同?
- ② 类 var 中变量 b 和方法 intprint() 中变量 c 有何不同?
- ③ 类 var 的访问控制权限如何? 如果加上访问控制符 private, 会出现什么结果?
- ④ 分析输出结果。

(2) 程序说明:

该程序的主方法中, 定义了 var 类的两个对象 v1 和 v2, 关于对象定义将在 3.3.1 节中介绍。

3.2.3 方法

Java 语言中, 方法只能作为类的成员, 也就是说, 方法只能定义在类体内。

1. 方法的定义格式

方法的定义格式通常包含方法说明和方法体两部分, 具体格式如下:

```
[<修饰符>] <类型> <方法名> (<参数表>) [<throws 异常类名列表>]  
{  
    <方法体>  
}
```

其中, 方法说明中必须包含<类型>、<方法名>和一对圆括号, 可省略的有<修饰符>、<参数表>和 throws 及后面的<异常类名列表>。方法体是由一对花括号括起来的若干条语句, 其中包含说明语句和执行语句, 也可以为空。

方法的作用通常有如下两个:

- (1) 对该类体内的变量进行各种操作;
- (2) 与其他类的对象进行信息交流, 作为类与外部进行交互通信的接口。

2. 方法的修饰符

方法的修饰符中有访问控制符和非访问控制符, 分别介绍如下。

1) 访问控制符

(1) 公有访问控制符 public。使用该修饰符的方法可作为该类对外的接口, 程序可以通过它与类体内的成员进行信息交换。

(2) 默认访问控制符一个默认访问控制符的方法具有包的访问性, 可被同一包内的其他类所访问。

(3) 私有访问控制符 private。使用该访问控制符修饰的方法只能被该类自身访问, 不能被其他类, 包含自身类的子类所访问。

(4) 保护访问控制符 protected。使用该访问控制符修饰的方法的访问权限: 包含该类自身; 与它在同一个包里的其他类; 在其他包中的该类的子类。

(5) 私有保护访问控制符 private protected。使用该访问控制符修饰的方法可被该类自身和该类的所有子类访问。

2) 非访问控制符

(1) 抽象方法 `abstract`。抽象方法是只有方法说明,而没有具体实现的一种方法。该方法的具体实现会出现在该类的子类中。使用抽象方法的目的是使所有该类的子类都有一个同名的方法作为统一的接口。

(2) 静态方法 `static`。静态方法是属于整个类的方法。调用这种方法应该用类名,也可以用对象名。静态方法在内存中的代码不被任何对象所专有。通常静态方法只能处理静态变量。

(3) 最终方法 `final`。使用 `final` 修饰符的方法是最终方法,这是一种不能被当前类的子类重新定义的方法。这种做法可以防止子类对父类方法的重定义,可保证系统的安全性和正确性。

(4) 本地方法 `native`。本地方法使用 `native` 关键字进行修饰,该方法通常用来说明其他语言(如 C、C++、汇编语言等)书写方法体,并实现方法功能的特殊方法。这种方法只在类体内给出说明,而方法体在类体外。使用本地方法可充分利用已有程序,避免重复性劳动。

(5) 同步方法 `synchronized`。该方法主要用于多线程共存的程序中的协调和同步。详细内容在“7.4 线程的同步”一节中介绍。

3. 方法的参数和返回值

定义方法时,可以有参数,也可以没有参数。在有多个参数时应用逗号分隔,每个参数应由参数类型和参数名组成。例如,

```
int intAdd(int i,int j)
{
    return i+j;
}
```

该方法有两个 `int` 型参数 `i` 和 `j`。

Java 语言中,有的方法有返回值,有的方法无返回值。无返回值的方法定义时加 `void` 类型。有返回值的方法定义时加返回值的类型。

方法的返回值是通过 `return` 来实现的,具体格式如下:

```
return <表达式>;
```

要求`<表达式>`的类型与方法定义中返回值的类型相一致。对基本数据类型,可通过自动转换或强制使类型一致;对类类型,完全一致或者为子类;如果返回类型为接口,则返回的数据类型必须实现该接口。例如,

```
class Myclass
{
    ...
    int method1()
    {
        boolean b;
```

```

short s;
int i;
double d;
...
return i;           //对
return s;           //对,可自动转换
return b;           //错
return (int)d;      //对,经强制转换
}
}

```

4. 方法调用

Java 语言中,方法的参数可以是基本类型,也可以是复合类型。

基本类型作为方法参数时,参数传递的是变量值,而不是变量地址值。因此,不能改变调用方法中的参数值。

复合类型作为方法参数(例如对象)时,参数传递的是对象的地址值,因此,对参数的改变会影响到原来的参数值。

在 Java 语言中,复合类型变量实际上是引用,并采用动态联编。

【例 3-3】 分析 Exam3_3.java 程序输出结果,并说明类中两个方法参数的传递方式。

(1) 程序 Exam3_3.java:

```

public class Exam3_3
{
    public static void main(String args[])
    {
        int array[]={5,4,3,2,1}
        int a=0,b=0;
        conveyValue(a,b);
        conveyValue(array)
        for(int i=0; i<array.length; i++)
            System.out.println("array["+i+"]="+array[i]);
    }
    static void conveyValue(int a,int b)
    {
        a+=2;
        b+=3;
        System.out.println("a="+a+",b="+b);
    }
    static void conveyValue(int array[])
    {
        for(int i=0; i<array.length; i++)
            array[i]+=1;
    }
}

```

执行该程序后,输出结果如下:

```
a=2,b=3  
array[0]=6  
array[1]=5  
array[2]=4  
array[3]=3  
array[4]=2
```

(2) 程序说明:

类 Myclass 中,除主方法 main() 外,还有两个同名方法。前一个方法有两个 int 型参数,后一个方法有一个复合类型变量数组参数。因此,前一个方法调用是传值方式的,后一个方法调用是传址方式的。在后一种方法中,对数组 array 元素值的改变,影响到实参数组元素值的改变。

【例 3-4】 分析 Exam3_4.java 程序的输出结果,并指出类中两个方法调用方式的不同。

(1) 程序 Exam3_4.java:

```
public class Exam3_4  
{  
    static double d;  
    public static void main(String args[])  
    {  
        int i;  
        Exam3_4 pt=new Exam3_4();  
        i=5;  
        pt.changeInt(i);  
        System.out.println("int type value is: "+i);  
        pt.d=6.25;  
        pt.changeDouble(pt);  
        System.out.println("double type value is: "+d);  
    }  
    public void changeInt(int v)  
    { v=10; }  
    public void changeDouble(Exam3_4 ref)  
    { ref.d=10.5; }  
}
```

执行该程序后,输出结果如下:

```
int type value is: 5  
double type value is: 10.5
```

(2) 程序分析:

该程序类中定义了一个主方法,又定义了一个方法 changeInt(),以及另外一个方法 changeDouble()。前一个方法的形参为一个 int 型数,另一个方法的形参为对象。这两

个方法实际上采用了不同的调用方式,前一个方法传递变量值;后一个方法传递地址值。因此,前一种调用在被调用方法中改变参数值,对调用方法的参数没有影响;后一种调用在被调用方法中改变参数值,对调用方法的参数有影响,所以获得上述结果。

思考:请读者在上述程序中增添一个字符串变量,并验证,以字符串变量作方法参数时,被调用方法内对参数的改变影响到调用方法的参数值。

5. 方法的重载

Java 语言支持方法重载这一重要特性,通过方法重载进而实现对象的多态性。

方法重载是指功能相同的多个方法使用同一个方法名。同名的多个方法的参数要有所不同,即在参数类型、参数个数和参数顺序上要有所区别,以便作为选择某个方法的根据。通常只有功能相同的方法进行重载才有意义。例如,计算两个数之和的重载方法定义如下:

```
int add(int i,int j)
{
    return i+j;
}
float add(float i,float j)
{
    return i+j;
}
double add(double i,double j)
{
    return i+j;
}
```

这 3 个同名方法具有相同的功能,即为求两个数之和。3 个重载的方法中,虽然参数个数相同,但是参数的类型不同。

重载方法的选择通常是在编译时进行。系统根据不同的参数类型、个数或顺序,寻找最佳匹配方法。方法类型不参与匹配。匹配的原则如下:

(1) 完全匹配为最佳方案。例如,

```
int a=5,b=3;
add(a,b);
```

系统将选择两个参数为 int 型的方法 add(),因为这是完全匹配的方案。

(2) 如果不能完全匹配时,则尽量选择类型转换代价最小的一种方案进行匹配。转换代价小的转换通常是指由高类型向低类型的转换。

转换匹配时,应遵循下述规则。

- ① 参数不能转换的重载方法不能参加匹配。例如,布尔型不能与其他类型转换。
- ② 算术类型通常可以相互转换,选择转换代价最小的参加匹配。例如,

```
char c1='a',c2='b';
add(c1,c2);
```

编译可将 char 转换为 int, float, double 型,其中转换为 int 的转换代价最小,根据最小匹配原则,调用方法 add(int,int)。

- ③ 复合类型作参数时,子类对象可转换为父类对象,子类层次最小的方法才会匹配。
- ④ 转换后出现两个以上的方法同时具有最小转换值时,编译出错。

【例3-5】 分析 Exam3_5.java 程序的输出结果,并说明重载方法的选择规则。

(1) 程序 Exam3_5.java:

```
public class Exam3_5
{
    public static void main(String args[])
    {
        int i1=3,i2=5;
        char c1='a',c2='b';
        float f1=3.1f,f2=4.1f;
        double d1=2.5,d2=3.8;
        System.out.println(add(i1,i2));
        System.out.println(add(c1,i1));
        System.out.println(add(i1,f1));
        System.out.println(add(f1,d2));
        System.out.println(add(d1,i2));
    }

    int add(int i,int j)
    static
    { return i+j; }

    double add(double i,double j)
    static
    { return i+j; }
}
```

执行该程序后,输出结果如下:

```
8
100
6.1
6.9
7.5
```

(2) 程序分析:

该程序中有两个重载方法,其名为 add(),它们的参数类型不同。在调用重载方法时,根据方法匹配原则选择匹配后的方法。

思考:请读者根据执行结果,说明具体匹配选择情况。

6. 构造方法

构造方法是一种特殊的方法,其功能是对创建的对象初始化。

(1) 构造方法的特点:

- ① 方法名同类名;
- ② 无返回类型;

③ 构造方法是在创建对象时,系统自动调用为所创建的对象进行初始化的方法。

构造方法可以没有参数,也可以有多个参数。构造方法可以重载。

(2) 构造方法的调用。构造方法不能被编程人员显式调用,而是用在运算符 new 创建类的对象时,由系统自动调用。构造方法的参数传递和形实结合也在调用时同时完成。使用 new 运算符创建对象的格式如下:

```
<类名><对象名> = new <类名>(<参数表>);
```

该语句完成如下两个任务:

① 用 new 通知系统为所创建的对象在内存开辟一个单元;

② 自动调用相应的构造方法使用给定的<参数表>为创建的对象进行初始化,使其新对象的各个变量获取初值。

(3) 构造方法的作用。构造方法的主要作用是用来给创建的对象进行初始化。尽管 Java 语言中,对基本数据类型都指定了固定的默认值,但是还需要使所创建的对象处于正常合理的状态,因此调用构造方法进行初始化是十分必要的。

构造方法的另一个作用是在构造方法体内引入一些操作,除初始化功能外,还具有更多的其他功能。

(4) 默认构造方法。在定义类时,如果不定义任何的构造方法时,系统会自动为该类生成一个默认的构造方法。该方法名同类名,没有任何形参,也不实现任何操作。使用它创建的对象使用的是默认值。

7. 析构方法

Java 语言提供了自动内存管理能力,可以自动释放掉不再被使用的对象。但是,Java 语言还支持析构方法,其目的在于让程序有机会主动地释放掉不同的对象。

典型的析构方法的格式如下:

```
protected void finalize();
```

当一个有析构方法的对象不再被引用时,系统会自动调用析构方法,释放它所占用的资源。这对打开的文件和窗口系统等主动释放是有益处的。

下面是一个用类析构窗口和文件的析构方法的定义。

```
protected void finalize()
{
    dispose();      //释放窗口系统
    if(fd!=null)
        fd.close(); //关闭文件
}
```

3.2.4 实例

下面以例 3-6 作为本节的小结。通过这一实例,熟悉类的定义、类中变量和方法的说

明及定义、构造方法的定义和重载等内容。

【例3-6】 编程实现输入今天的日期，输出明天的日期。

程序 Exam3_6.java：

```
public class Exam3_6
{
    private int year,month,day;
    Exam3_6
    {
        year=2000;
        month=1;
        day=1;
    }
    Exam3_6(int a,int b,int c)
    {
        year=a;
        month=b;
        day=c;
    }
    Exam3_6(Exam3_6 d)
    {
        year=d.year;
        month=d.month;
        day=d.day;
    }
    public void outDate()
    {
        System.out.print(year+"/"+month+"/"+day);
    }
    public Exam3_6 tomorrow()
    {
        Exam3_6 d=new Exam3_6(this);
        d.day++;
        if(d.day>=d.daysInMonth())
        {
            d.day=1;
            d.month++;
            if(d.month>12)
            {
                d.month=1;
                d.year++;
            }
        }
        return d;
    }
    public int daysInMonth()
```

```
{\n\n    switch(month)\n    {\n        case 1: case 3: case 5: case 7: case 8: case 10: case 12:\n            return 31;\n        case 4: case 6: case 9: case 11:\n            return 30;\n        default:\n            if(year%4==0&&year%100!=0||year%400==0)\n                return 29;\n            else\n                return 28;\n    }\n}\n\npublic static void main(String args[])\n{\n    Exam3_6 d1=new Exam3_6();\n    System.out.print("The current date is (year/month/day):");\n    d1.outDate();\n    System.out.println();\n    System.out.print("Its tomorrow is (year/month/day):");\n    d1.tomorrow().outDate();\n    System.out.println();\n    Exam3_6 dd=new Exam3_6(2004,1,8);\n    System.out.print("The current date is (year/month/day):");\n    dd.outDate();\n    System.out.println();\n    System.out.print("Its tomorrow is (year/month/day):");\n    dd.tomorrow().outDate();\n    System.out.println();\n}\n}
```

提醒：请读者自己分析该程序执行后的输出结果。

3.2.5 静态变量和静态方法

在类体内使用关键字 static 修饰的变量和方法分别称为类变量和类方法，而不使用该关键字修饰的成员变量和成员方法分别称为对象变量和对象方法。

1. 静态变量

静态变量又称类变量，在类体内使用关键字 static 修饰的成员变量便是该种变量。

静态变量的特点是它不属于某个对象的，而是属于整个类的。静态变量不保存在某

一个对象的存储单元中，而是保存在类的公共内存单元中，任何一个类的对象都可以访问它，包括修改它。静态变量一旦被某个对象修改后，则保存被修改后的值，直到下次被修改为止。因此，静态变量对于类的所有对象来讲，是一个公用变量。

静态变量的定义格式如下：

```
static <类型> <变量名>;
```

例如，

```
class Student
{
    String Name;
    int age;
    double score;
    static long totalNum;
    :
}
```

在该类中，定义一些成员变量，其中 totalNum 是静态变量，而其余 3 个成员变量为非静态变量，即为对象变量。

假定 s 是 Student 类的一个对象。在存储对象 s 的存储单元中，不存放静态变量 totalNum，这个静态变量存放在一个公共存储单元中。

静态变量可以通过类名直接访问，也可以通过对对象访问，两种访问方法是等同的。例如，在上例中，访问静态变量 totalNum 可用如下两种方法：

Student.totalNum 和 s.totalNum

前一种方法是用类名访问静态变量，后一种方法是用对象访问静态变量，两种方法的结果是相同的。

2. 静态方法

静态方法又称类方法，在类体内使用关键字 static 修饰的成员方法便是该种方法。

静态方法是属于整个类的，不是属于某个对象的。因此，应该这样认识静态方法。

(1) 在创建对象时，由于非静态方法是属于对象的，因此在对象占用的内存中有该方法的代码。而静态方法是属于整个类的，因此在对象占有的内存中没有该方法。静态方法在内存中的代码是随着类的定义而进行分配的，它不被某个对象所专有。

(2) 静态方法只能处理静态变量或调用静态方法。它不能处理对象变量或对象方法。

(3) 调用静态方法通常使用类名，也可以使用某个对象名。

以上给出了关于静态方法的特点。

【例 3-7】 分析 Exam3_7.java 程序的输出结果。熟悉程序关于静态变量和静态方法的定义和使用。

(1) 程序 Exam3_7.java：

```
public class A
```

```
{  
    int a;  
    static int b;  
    void setab(int i)  
    {  
        a=i;  
        b=i+1;  
    }  
    int geta()  
    {  
        return a;  
    }  
    static void setb(int i)  
    {  
        b+=i;  
    }  
    static int getb()  
    {  
        return b;  
    }  
}  
public class Exam3_7  
{  
    public static void main(String args[])  
    {  
        A x=new A();  
        A y=new A();  
        x.seta(5);  
        y.seta(10);  
        System.out.println("x. a=" + x.geta() + ", y. a=" + y.geta());  
        x.setb(-5);  
        y.setb(-10);  
        System.out.println("x. b=" + x.getb() + ", y. b=" + y.getb());  
    }  
}
```

执行该程序后,输出结果如下:

```
x. a=5,y. a=10  
x. b=-4,y. b=-4
```

(2) 程序分析:

该程序的类 A 中有一个静态变量,即类变量 b 和一个非静态变量,即对象变量 a。该类中还有两个静态方法 setb()和 getb()和两个对象方法 setab()和 geta()。

请读者回答下述问题。

(1) 在静态方法 setb() 的方法体中, 增添下述语句:

```
a=i-1;
```

是否可以? 上机试一下。

(2) 在对象方法 geta() 的方法体中, 将下列语句:

```
return a;
```

修改为

```
return a+b;
```

是否可以? 上机试一下。

(3) 通过例 3-7 请读者回答下述问题:

① 如何在程序中定义静态变量和静态方法?

② 在静态方法中是否可以引用非静态变量? 在非静态方法中是否可以引用静态变量?

③ 如何将该程序中使用对象引用静态方法修改可使用类名引用静态方法? 并上机调试。

3. 静态初始化器

静态初始化器与构造方法的功能相似, 它们都是用来完成初始化的。所不同的是静态初始化器用来对每个类进行初始化, 即对属于该类的成员变量进行初始化; 而构造方法是用来对每个新创建的对象进行初始化, 即对于对象变量进行初始化。

静态初始化器的定义格式如下:

```
static
{
    <被初始化的成员变量>
}
```

可见静态初始化器是由关键字 static 和一对大括号括起的若干个类成员的初始化项构成的。静态初始化器不是方法, 没有类型、方法名和参数表。

静态初始化器是对整个类进行初始化操作的, 它是在它所属的类被加载入内存时由系统自动实现。

在例 3-7 中, 可在类 A 体内添加如下的静态初始化器, 用来给静态变量 b 初始化:

```
static
{
    b=10;
}
```

3.2.6 抽象类和抽象方法

使用修饰符 abstract 修饰的类和方法称为抽象类和抽象方法。

1. 抽象类

抽象类是用修饰符 `abstract` 说明的类。抽象类的特点是该类没有具体的对象。因此,又称抽象类为概念类。例如,“人类”便是一个抽象类,因为人类指的是各国家各民族的人。人类没有一个具体的对象,而人类又分成若干个子类,比如,中国人、美国人等,这些子类才具有对象。例如,张三是中国人,Tom 是美国人。人类仅仅作为一个抽象的概念,它代表所有人的共同属性,而任何一个具体人却是由“人类”经过特殊化形成的某个子类的对象。

抽象类的作用在于它抽象概括了某类事物的共同属性,在描述其子类时只需简单描述其子类的特殊之处,不必再重复其抽象类的共同特点。因此,使用抽象类的优点是可以充分利用它所具有的公共属性来提高开发程序的效率。在实际应用中,抽象类一定有它的子类,而子类通常不是抽象类,如果其子类是抽象类,则该抽象类一定会有不是抽象类的子类。抽象类与其子类的关系是继承关系,详情在后面 3.4 节中介绍。

抽象类不能定义对象,但是抽象类可以有构造方法,它可以被其子类所调用,具体实施过程在 3.6.2 节中介绍。

2. 抽象方法

使用关键字 `abstract` 修饰的方法称为抽象方法。抽象方法是一种仅有方法头,而无方法体的一种特殊方法。该方法不能实现任何操作,它只能作为所有子类中重载该方法的一个统一接口。

抽象方法定义格式如下:

```
abstract <类型> <方法名>(<参数表>);
```

在该定义中,使用一个分号来替代了方法体。

抽象方法一定出现在抽象类中,该方法在抽象类中没有方法体,而该方法的实现表现在抽象类的若干子类中。子类继承了父类的抽象方法后,使用不同的实现(即方法体)来重载它。于是便形成了若干个名字、返回值类型和参数表都相同,而方法体不同的重载方法。

使用抽象方法的好处在于抽象方法在不同类中重载,可以隐藏具体的细节信息,在程序中只给出抽象方法名,而不必知道具体调用哪个方法。抽象方法提供了一个共同的接口,该抽象类的所有子类都可以使用该接口来实现该功能。

【例 3-8】 分析 Exam3_8.java 程序的输出结果。熟悉抽象类和抽象方法的定义和使用。

(1) 程序 Exam3_8.java:

```
abstract class B
{
    void outB()
    {
        System.out.println("in B.");
    }
}
```