凡治众如治寡，分数是也。

——《孙子兵法·势篇》

第3章 方　法

面向对象程序设计中将现实中的事物抽象为对象；把对相同对象的描述抽象为类。类的组成包括变量和方法。Java 中的方法用于实现类的行为，且只能作为类的一部分存在。方法(Method)在其他语言中也称为函数。

3.1 方法的概念

程序中相同的代码可能会多次重复出现，为了提高代码的复用性，将实现某个功能的语句集合在一起封装为方法。方法用于模拟类所具有的能力、动作或者行为。方法的优点有以下方面。

(1) 使程序变得更简短而清晰。

(2) 有利于程序维护。

(3) 可以提高程序开发的效率。

(4) 提高了代码的复用性。

3.2 如何定义方法

视频 3-1

一般情况下，定义一个方法包含以下语法：

```
修饰符 返回值类型 方法名(参数类型 参数名){
    …
    方法体
    …
    return 返回值;
}
```

方法包含一个方法头和一个方法体。方法有以下内容。

修饰符是可选的,定义了该方法的访问类型。

返回值类型是方法返回值的数据类型。如果方法没有返回值,返回值类型为 void。

方法名是方法的名称。方法名命名规则为:第一个单词以小写字母作为开头,后面的单词则用大写字母开头,尽量不使用连接符。例如 addPerson,方法名和参数列表共同构成方法签名。

参数类型指定了参数的类型,参数像是一个占位符。当方法被调用时,传递值给参数。这个值被称为实参。参数列表是指方法的参数类型、顺序和参数个数的集合。参数是可选的,方法可以不包含任何参数。

方法体包含具体的语句,定义该方法的功能。

下面给出一个方法的代码片段。该方法的功能是在登录时,显示提示信息。

```
static  void  displayMessage() {                          //方法头
    System.out.println("请输入用户名和密码: ");           //方法体
}
```

方法名为 displayMessage(),该方法不需要输入参数,方法即使没有参数也要保留圆括号()。该方法没有返回值,没有返回值时需要声明返回类型为 void。static 关键字表示该方法是静态方法,main()方法也是静态方法,静态方法会在后续章节详细介绍,静态方法可以直接调用。方法体内完成了输出提示信息的功能。

3.3 方法调用

Java 中,除了 main()方法是系统自动调用外,其他方法必须明确被调用。通过方法名和参数列表调用方法,形式如下:

```
方法名(实际参数表);
```

实际参数又称为实参,用来初始化调用方法的参数列表。实际参数必须与方法定义中的参数一致,包括参数的个数和类型。如果不一致,则会编译出错。

【例 3.1】 方法声明和调用示例。

```
//MyMethod.java
import java.util.*;
class MyMethod {
    static void displayMessage() {                        //方法定义
        System.out.println("请注意保护个人密码。");       //方法体
        System.out.println("请输入取款密码.");
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        String myPSW;
```

```
            displayMessage();                    //调用 displayMessage()方法
            myPSW = sc.next();
        }
    }
```

displayMessage()方法定义独立于 main()方法，它们是并列关系，都属于同一个类。方法定义都要在某个类的内部，不可以单独编写。方法定义在类中的顺序不会影响编译器的执行过程，因此 displayMessage()方法在 main()方法之前还是之后是无关紧要的。程序执行总是从 main()方法开始。在 main()方法中，执行到方法调用语句 displayMessage()时，才会真正运行 displayMessage()方法。

需要强调的是，当一个方法调用另一个方法时，调用方法会在调用点暂停执行，而跳转到被执行方法中执行相应语句；当被调用方法结束时程序又返回调用方法中再次继续运行。调用过程如图 3.1 所示。图 3.1 展示了 main()方法调用其他方法的过程，其实方法之间也可以相互调用。

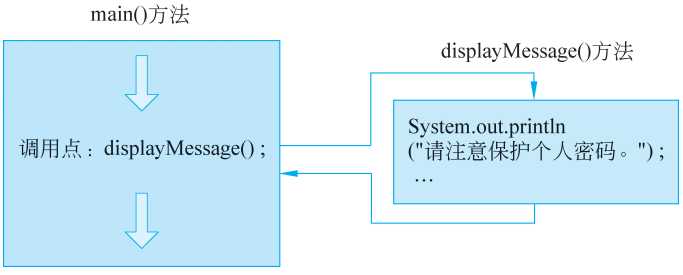


图 3.1　方法调用示意图

3.4　设计有参数和返回值的方法


视频 3-2

前面介绍的方法没有参数和返回值，较为简单。下面介绍有参数和返回值的方法。

以判断闰年为例，可以把闰年判断的逻辑编写为独立的方法。计算闰年时，需要给定一个确定的年份，根据此年份才可计算是不是闰年。计算完毕，需要告诉调用者判断结果。因此，方法有一个输入参数——年份；还有一个返回值——是否为闰年的判断结果。方法体完成判定的任务。

【例 3.2】　闰年的判断方法。

```
static  boolean  isLeapYear ( int  yearIn )
{
  boolean  resultIn=false;
    if((yearIn % 4 ==0)&& (yearIn % 100 !=0) ||
        (yearIn % 400 == 0))
      resultIn = true;
  return resultIn;
}
```


视频 3-3

方法的首部，修饰为静态的，表示方法可以直接调用。boolean 是返回值类型，表示方法调用时将返回 boolean 类型的判断结果。isLeapYear 是方法名，调用要通过方法名实现。方法有一个整数类型参数 yearIn，表示在调用时需提供一个整数类型的参数，即待判断年。方法体将对 yearIn 进行计算，判断其是否为闰年，并将 boolean 类型的判断结果返回。

例 3.2 中，方法内部的变量命名均带有 In，表示在方法内部有效。最后，要把判定结果返回，使用 return 语句：

```
return resultIn;
```

return 语句有两个功能。首先，它是方法的结束语句，一旦程序运行到 return 语句，方法将结束同时程序控制权跳回到调用方法。第二个功能是可以携带参数，将参数返回给调用者。例 3.2 中，它返回了判定结果。注意，返回值的类型必须与方法头部声明的类型一致。

方法什么时候结束运行呢？通常发生在下列情况：

(1) 运行完方法体的最后一条语句。

(2) 执行到 return 语句。

编程人员根据方法的复杂性，可能在程序的多个位置设置 return 语句，方法运行时只要执行到某个 return 语句，方法就立即结束，不管后面是否还有其他语句。

return 语句有以下两种形式。

(1) return。

(2) return 表达式；或 return （表达式）。

形式(1)适合方法返回值类型是 void 的方法。

形式(2)适合方法返回值类型是非 void 的方法。换句话说，void 返回类型的方法不需要也不能返回数据，而返回值类型为其他类型的方法必须使用 return 语句返回表达式结果。

例如，main()方法的返回值类型是 void，因此无须返回数据(main()方法执行结束即代表程序运行结束，相当于返回到操作系统对该方法的调用位置)。接下来讨论调用带参数的方法。

【例 3.3】 有一个参数和返回值的方法。

```
//LeapYear.java
import java.util.*;
public  class LeapYear {
    static boolean isLeapYear(int yearIn){
           boolean  resultIn = false;
    if((yearIn % 4 ==0) && (yearIn % 100 !=0) ||
                                        (yearIn % 400 == 0))
           resultIn = true;
           return resultIn;
        }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int year;
        boolean result;
```

```
            year = sc.nextInt();
            //调用 isLeapYear()方法,传入实际参数 year
            result = isLeapYear(year);
            System.out.println(year + " is  Leap Year?" + result);
        }
    }
```

正如前面介绍过,Java 程序运行时虚拟机会先找到 main()方法,从 main()方法开始执行。在 main()方法中,当执行到调用方法语句:

```
result = isLeapYear(year);
```

程序会跳转到 isLeapYear()方法内部执行。执行前,先将实际参数(year)赋值给形式参数(yearIn),即 yearIn = year。在 isLeapYear()方法内部执行到 return resultIn 语句时,会把 resultIn 的值返回给 main()方法。接着回到 main()方法中,把执行结果赋给布尔变量 result。这就是方法参数传值的整个过程,如图 3.2 所示。

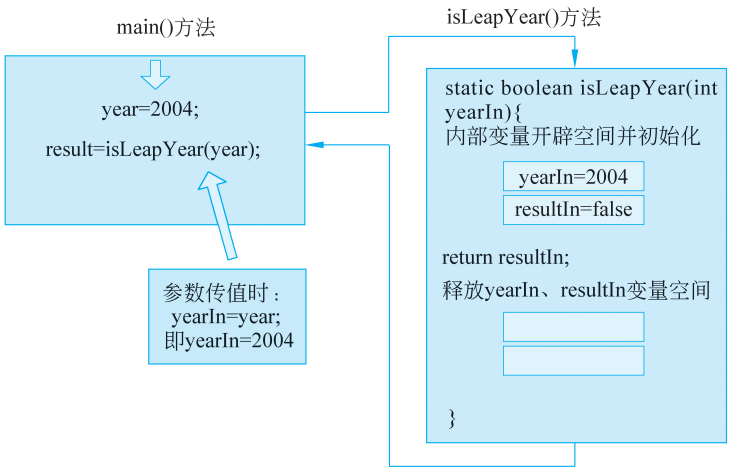


图 3.2 方法参数传递示意图

要注意变量的作用域问题。在方法内部定义的变量仅在方法内部有效。例如,在 isLeapYear()方法中,yearIn、resultIn 变量是方法内部变量。调用方法时,会创建存储两个变量的内存空间;方法执行完毕,会释放空间。因此,main()方法无法访问到 yearIn、resultIn 变量。

3.5 方法应用

在定义方法的过程中,一般要考虑 4 个问题。

- 方法名。
- 方法的输入(即形参)。
- 方法的输出(即返回值)。

• 方法体(完成相应功能的语句)。

方法名的定义要符合标识符规则,尽量做到见名知意。如果方法名由多个单词组成,一般第一个单词全部小写,后续单词的首字母要大写,如 myMethod、isEven。

3.5.1 单个参数的方法

【例 3.4】 计算圆的面积。

【编程思路】 计算圆的面积,方法命名为 circleArea。要完成计算,需先确定圆的半径,因此该方法需要一个输入参数(半径)。计算完毕,需要把计算结果返回给调用方法 main(),因此也需要一个返回值。计算面积的算法是 $\pi * r^2$,方法体内按照公式计算并返回结果。用方法实现计算圆的面积。

```
//MethodExp3.java
public class MethodExp3 {
  static double circleArea(double radiusIn){
       final double PI = 3.14;                    //定义常量 PI
       //根据传入参数计算圆的面积,并将结果返回
       return ( PI * radiusIn * radiusIn );
  }

  public static void main(String[] args) {
       double result=0.0;
       //调用 circleArea()方法,计算半径数值为 2.4 的圆面积
       result = circleArea(2.4);
       System.out.println(result);
       //调用 circleArea()方法,计算半径数值为 4.0 的圆面积
       System.out.println(circleArea(4.0));
  }
}
```

程序中通过方法名和参数实现调用,调用语句为:

```
result = circleArea(2.4);
System.out.println(circleArea(4.0));
```

调用时,传入的参数即实参赋值给形参(radiusIn),即

```
radiusIn = 2.4;                                   //第一次调用
radiusIn = 4.0;                                   //第二次调用
```

两次调用的结果处理方式有所不同。第一次调用,将结果返回给 main()方法内定义的变量 result;第二次调用,直接将结果输出显示。

3.5.2 多个参数的方法

方法有时需要多个参数,下面通过具体问题来了解多参数方法的处理。

【例 3.5】 实现由任意符号组成的 $i \times j$ 的矩形。

【编程思路】 方法的功能是绘制矩形，方法命名为 drawRectangle。绘制时，需指定绘制的符号，以及 $i \times j$ 的尺寸(行列数)。因此，输入参数应该有 3 个，分别为绘制符号、矩形宽、矩形高。方法体实现矩形的输出显示，该方法不需要返回结果，因此返回值类型是 void。方法体内绘制算法，回顾第 2 章介绍过的 for 循环的实例，它实现了由 10×4 个“*”组成的矩形。

```
for( i=0; i <4; i++ )                          //外部循环,实现打印出多行
{
    for( int j =0; j<9; j++){
    System.out.print("* ");                    //内部循环实现打印出一行"* "
}
System.out.println();                          //每行结束后,换行
}
```

在此算法中，只需将固定数值和符号替换为传入的参数，即可实现该功能。

```
//MethodExp4.java
public class MethodExp4 {
    //三个输入参数的方法
static void drawRectangle(char symbolIn, int xIn, int yIn)
    {   //外部循环,实现打印出多行
        for( int i=0; i <yIn; i++ )  {
            //内部循环实现打印出一行 symbolIn
            for( int j =0; j<xIn; j++){
                System.out.print(symbolIn);
            }
            System.out.println();              //每行结束后,换行
        }
    }
    public static void main(String[] args) {
        drawRectangle('@',8,4);                //调用 drawRectangle()方法
        System.out.println();
        drawRectangle('#', 15, 5);             //调用 drawRectangle()方法
    }
}
运行结果:
@@@@@@@@
@@@@@@@@
@@@@@@@@
@@@@@@@@

###############
###############
###############
###############
###############
```

具有多个参数的方法定义与一个或无参数的方法类似，格式依然是方法名(参数)。只

是参数之间注意用“,”分开。方法调用时要注意参数的顺序和类型,必须严格按照定义的参数顺序和类型传递。

```
例如,调用语句:drawRectangle('@',8,4);
调用时,参数赋值严格按照方法定义 void drawRectangle(char symbolIn, int xIn, int
yIn)的顺序进行。实参列表赋值给形参列表:
symbolIn='@';
xIn=8;
yIn=4;
```

3.5.3 递归方法

递归的思想就是:把问题分解成规模更小的、具有与原问题相同解法的问题。既然递归的思想是把问题分解成规模更小且与原问题有着相同解法的问题,那么是不是这样的问题都能用递归来解决呢?答案是否定的。并不是所有问题都能用递归来解决。那么什么样的问题可以用递归来解决呢?一般来讲,能用递归来解决的问题必须满足两个条件。

(1) 可以通过递归调用来缩小问题规模,且新问题与原问题有着相同的结构和解法形式。

(2) 存在一种简单情境,可以使递归在简单情境下退出。

如果一个问题不满足以上两个条件,那么就不能用递归来解决。

方法的递归调用有两种方式,一种是直接递归,即方法直接调用自己;另一种是间接递归,即方法调用其他方法,而其他方法又再调用这个方法。不管是直接递归还是间接递归,如果没有一个终止条件结束递归,将会造成无休止的调用,因此在写递归程序时,一般要把终止条件写在方法的最前面。

下面通过求 n! 的例子来了解递归的具体过程。

首先,将原有问题逐步分解为与原有问题处理方法相同的更简单的问题,直至分解出来的问题结果已知。下面以 4! 为例进行分析。

4! = 4 *3!

3!=3 *2 !

2!=2 * 1!

1!=1 * 0!

0!=1;

归纳一下,可将分解过程描述为 n!= n * (n−1)! 直至分解到 0!。用伪代码描述为:

```
计算阶乘方法(参数为 n){
    当 n==1 时,n!=1;
    当 n>1 时,n! =n * 计算阶乘方法(n-1);
}
```

用 Java 语言实现为:

```
long Factorial(int n){
    if ( n == 1)
```

```
        return 1;                                    //递归停止条件,写在前面
    else
        return n * Factorial(n-1) ;                  //递归调用自身
}
```

【例 3.6】 求斐波那契(Fibonacci)数列第 n 项的方法 fibo(n)。斐波那契数列为 0、1、1、2、3、5、8、13、21、……,即从第 3 项开始,每一项为前两项之和。

【编程思路】 经过分析,斐波那契数列可表示为:

fibo(0)=0;

fibo(1)=1;

fibo(n)=fibo(n−1)+fibo(n−2)(n>=2)。

也就是说,当形参 n 为 0 时返回 0,为 1 时返回 1,不再递归,而是将 n 返回。当形参 n 接收的数大于或等于 2 时进行递归。

```
//MethodExp5.java
public class MethodExp5 {
    static int fibo(int n)
    {
        if(n == 0 || n == 1 )                        //递归终止条件
            return 1;
        else
            return fibo(n-1) + fibo(n-2);            //递归调用
    }

    public static void main(String[]  args){
        int n = 8;
        //调用递归方法
        System.out.println( "fibo " + n + " =" + fibo(n));
    }
}
运行结果:
fibo 8=34
```

使用递归方法编写的程序简洁清晰。不过,每次调用方法时,系统都需要分配内存来保存中间结果直至满足停止条件,系统开销比较大。在实际编程中,建议根据情况选择递归。

3.5.4 多个返回值的方法

前面介绍的方法返回值都只有一个。方法中虽然可以包含多个 return 语句,但程序执行到第一个 return 语句时方法就结束。因此,方法看似只能有一个返回值。那么,方法是否可以有多个返回值呢? 方法当然可以有多个返回值。多个返回值的实现要借助含有多个数据项的数据类型(如数组、类)作为返回类型,这部分将在后续介绍。

3.6 方法重载

方法的重载(Method overloading)指在一个类的内部,允许存在多个同名方法,只要它们的参数个数或者类型不同。

【例 3.7】 add()方法重载。

```
//MethodExp5.java
public class MethodExp6 {
    static  int  add(int x, int y)                  //两个整数相加
    {
        return x + y;
    }
    static  int  add(int x, int y, int z)           //三个整数相加
    {
        return x + y + z;
    }
    static  double  add(double x, double  y)        //两个双精度数相加
    {
        return x + y;
    }
    public static void main(String[] args) {
        int isum;
        double fsum;
        //调用 int  add(int x, int y) 方法
        isum = add(3,5);
        //调用  int  add(int x, int y, int z) 方法
        isum = add(1,10, 3);
        //调用 double add(double x, double y)方法
        fsum = add(7.6, 8.5);
    }
}
```

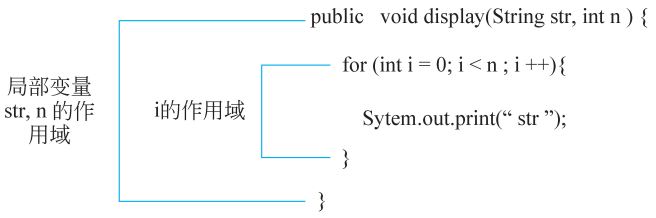
示例中有 add()同名方法,且方法的参数个数或者类型不同,这种现象就是方法重载。方法调用时根据方法名和参数来实现。当方法名相同时,唯一的决定因素就是参数,因此参数的个数和类型决定了程序调用哪个重载方法。注意:重载方法的参数列表必须不同,要么是参数的个数不同,要么是参数的类型不同,一般不将参数出现的顺序作为区分重载条件。重载方法的返回类型可以相同,也可以不同。

3.7 变量的作用域

变量的作用域是程序中该变量可以被访问或使用的范围。方法内定义的变量是局部变量。局部变量必须声明才可以使用。局部变量的作用域是从声明位置开始,到距离该变量

最近的一对花括号“{ }”所包括的代码块结束。

【例 3.8】 局部变量作用域示例。



例 3.8 中，变量 str，n 的声明在 display()方法内，是 display()方法的输入参数，也是该方法的局部变量。作用域涵盖整个方法。

for 循环的初始化部分声明的变量 i，其作用域仅限于整个 for 循环。

3.8 综合案例：验证码生成

1. 案例描述

本例生成一个指定长度的验证码，验证码由数字、大写字母、小写字母组成。验证码长度由用户指定。

2. 设计思路

设计一个验证码生成方法，该方法需要用户指定长度，因此有一个参数，表示用户指定的长度；验证码生成结果需返回给用户，因此该方法有返回值。验证码由三种字符类型组成，随机产生 0、1、2，代表三种类型。每种类型对应的具体字符也通过随机数产生。

3. 案例实现

Demo.java：

```
import java.util.Random;
public class Demo {
    public static void main(String[] args) {
        System.out.println(code(5));
    }

    public static String code(int n) {
        String code = "";
        Random r = new Random();
        for (int i = 0; i < n; i++) {
            int caseValue = r.nextInt(3);    //生成 0、1、2，分别代表生成数字、大写字母、
//小写字母
            switch (caseValue) {
                case 0:                      //生成数字 0~9
                    code += r.nextInt(10);   //code = code + 0-9
                    break;
                case 1:                      //生成 A~Z 大写字母
```

```
                    code += (char)(r.nextInt(26) + 'A');   //'A'的 ASCII 码是 65,
//'Z'的 ASCII 码是 90,生成 65~90 的随机数并转换为大写字母 A~Z
                    break;
                case 2:                          //生成 a~z 小写字母
                    code += (char)(r.nextInt(26) + 'a');   //'a'的 ASCII 码是 97,
//'z'的 ASCII 码是 122,生成 97~122 的随机数并转换为小写字母 a~z
                    break;
            }
        }
        return code;
    }
}
```

习题 3

1. 判断下列说法是否正确。

   (1) 方法只能返回一个值。 (　　)

   (2) 方法的参数允许有多个。 (　　)

   (3) 方法执行到 return 语句时,将立即退出。 (　　)

   (4) 方法的形参和实参,名称必须相同。 (　　)

2. 语句 int(Math.random() * 6)+ 1 的作用是(　　)。

   A. 产生 1～6 的随机数　　B. 产生 100～600 的随机数

   C. 产生 10～60 的随机数　　D. 产生 1000～6000 的随机数

3. 下面的方法头哪一个有效? 如果无效,请解释原因。

   (1) public static one(int a, int b)

   (2) public static int thisone(char x)

   (3) public static char another(int a, b)

   (4) public static double yetanother

4. 如下方法,调用 stery(5)时返回值是(　　)。

```
public staic int stery(int n){
    if (n == 0)
        return 1;
    else
        return 3 * stery( n - 1);
}
```

   A. 0　　B. 3　　C. 81　　D. 243　　E. 6561

5. 下列是方法定义的形参表,定义形式正确的是(　　)。

   A. int num1,num2　　B. num1,num2

   C. int num1,int　num2　　D. int num1;int num2

6. 若有以下调用语句，则正确的 fun()方法头部是(　　)。

```
public static void main(String args[])
{
    float x;
    int a;
    …
    fun( a, x );
}
```

A. void fun(int m, float n)　　B. void fun(float a, int x)

C. void fun(int m, float x[])　　D. void fun(int[] x,float a)

7. 关于方法的参数，以下说法正确的是(　　)。

A. Java 中每个方法都必须有参数

B. 方法的参数要么是一个，要么是多个

C. 有返回值的方法必须有参数

D. 形参变量是 int 型的方法，调用语句中所给的实参数据也必须是 int 型

8. 属于 main()方法的返回类型是(　　)。

A. public　　B. static　　C. void　　D. main

9. 若有方法定义：staitc int fun(int num){return 2 * num;}，以下调用语句正确的是(　　)。

A. System.out.println(fun('A'+'B'));

B. int a=fun(2.3) ;

C. fun(2 , 3);

D. if(fun(4)) System.out.println("正确");

编程练习

1. (1) 编写提供三个选项的菜单驱动程序。

第一选项，当用户输入摄氏温度时显示出华氏温度；

第二选项，当用户输入华氏温度时显示出摄氏温度；

第三选项，用户退出。

用到的公式(C 代表摄氏温度，F 代表华氏温度)：

F=(9C/5)+32

C=5(F-32)/9

(2) 为了使程序正常运行，用户输入的温度不能低于绝对零度，也就是-273.15℃，或者-459.67℉。

2. 编写方法 reverseDigit，将一个整数作为参数，并反向返回该数字。例如，reverseDigit(123)的值是 321。同时编写程序测试此方法。

3. 编写程序，一列数的规律如下：1、1、2、3、5、8、13、21、34、……求这列数的第 30 位数是多少。

4. 编写方法 distance，计算笛卡儿平面中两点(x1,y1)和(x2,y2)间的距离。此方法将表示平面上两个点的四个数作为参数，返回两点间的距离。

5. 编写程序，随机生成 100 个 0～9 的数，统计 0～9 每个数出现的概率。