

第3章

计算面积周长——方法与作用域

能力目标：

- 学会定义方法和调用方法,理解变量和字段的作用域;
- 能编写方法,计算圆、矩形的面积和周长。

3.1 任务预览

本章实训要编写计算圆、矩形面积和周长程序,程序运行结果如图 3-1 所示。



(a) 计算圆面积和周长

(b) 计算矩形面积和周长

图 3-1 实训程序运行界面

3.2 方法定义

方法是命名的语句有序集,是一系列执行步骤的汇总。在一些计算机语言中,方法也称为函数、子程序或过程。

方法由方法头和方法体两部分组成。方法定义包括声明方法头、编写方法体。

方法定义的一般语法形式：

```
可选 public 等 可选 static 返回类型 方法名( 可选参数表 ) {  
    ... //语句构成的方法体  
}
```

大括号前面是方法头,方法头可以声明方法的访问级别,级别有 public、private 等,分别表示公共的、私有的。Java 是面向对象的语言,方法必须在类或其他引用类型的内部定义,访问级别决定了方法的使用范围(作用域)。如果没有声明访问级别,则默认为包(package)范围,即包访问级别。

方法头还可添加关键字 static,用于声明静态方法(类方法)。静态方法是整个类(所有对象)共用的方法,调用时以类名作前缀。没有 static 的方法,是非静态方法(实例方法),调用时只能以对象名作前缀。

声明方法必须声明返回类型(构造方法除外)、方法名和圆括号。方法名属于标识符,圆括号是方法的重要标志,无论是否有参数,圆括号都不能省略。

方法的返回类型有 int、double、String 等,没有返回值的方法,其返回类型为 void,表示空类型。

参数表是可选的。方法可以没有参数,也可以有多个。如果有多个,则用英文逗号分隔。每个参数都要声明数据类型。方法声明中的参数,是没有确定值的,属于形式参数,简称“形参”。

方法体是方法的主体,由大括号括起的语句组成(是一个复合语句)。调用方法运行时,将按顺序逐个执行方法体的语句。因此方法体内各语句的顺序非常重要,不可以排错。

如计算圆面积的静态方法:

```
static double calcArea(double r){
    double area;
    area = 3.14 * r * r;
    return area;
}
```

又如计算圆周长的静态方法:

```
static double calcGirth(double r){
    return 2 * 3.14 * r;
}
```

上面两个方法的方法名为 calcArea 和 calcGirth。方法名属于标识符,要按标识符命名规则命名。方法是一系列动作的总称,建议命名时使用动词或动宾结构的词汇。

方法体中一般有返回语句,用于结束本方法,返回调用它的地方。返回语句格式如下:

return 可选的表达式;

返回语句中的 return 是关键字,后面的表达式不是必须的。即对于返回类型为 void 的方法,内部如果有 return 语句,则语句不带表达式;但对于返回类型非 void 的方法(如 double),则返回语句必带表达式,并且表达式的值要与返回类型相符。

注意: 返回类型为 void 的方法,执行时没有返回值,因此 return 语句不是必须的。

返回语句通常位于方法的尾部,因为它会结束方法的执行。

上述计算圆面积方法 calcArea 的方法体含有 3 条语句,最后一条语句返回局部变量 area 的值,该变量已在第 2 条语句中被赋值(其中参数 r 表示圆半径),因此,给出参数 r 的实际值,如 3.1,调用该方法,便可得到圆的面积,如 30.18。

试一试：计算圆周长方法 `calcGirth` 的方法体只有 1 个语句，能否在计算圆面积方法 `calcArea` 中也用 1 个语句替换那 3 个语句？

方法只需定义一次，允许多次调用。使用方法的目的是，就是避免重复编码。

3.3 方法调用

定义方法就是为了调用(使用)它。方法每调用一次，方法体就被执行一次。

调用方法的语法形式：

方法名(可选实参表)

调用方法时，参数个数和类型必须与方法声明一致。参数用于传递数据。调用时参数必须有值，称为实际参数，简称“实参”。

如果方法声明没有参数(称无参方法)，则方法调用也没有参数。

如果方法声明有参数，则调用时必须为每个参数(形参)提供一个参数值(实参)。

实参是一个能求值的表达式，可以是最简单表达式——常量，或有值的变量。如果是变量，则无须在调用时声明类型，因为在调用之前就已声明并赋值了。

例如，调用计算圆面积、周长方法：

```
calcArea(3.1)
calcGirth(3.1)
```

非 `void` 的方法，属于一个表达式，调用后有值，可直接输出，或赋给某一变量，或参与另一个表达式的运算，或作为另一个方法调用的实参。

例如，调用计算圆面积方法 `calcArea`，并把结果赋给一变量：

```
double radius, area;
radius = 3.1;
area = calcArea(radius);
```

又如，调用计算圆面积方法 `calcArea`，并把结果作为输出方法 `println` 的实参，即在控制台中输出圆的面积：

```
System.out.println(calcArea(3.1));
```

注意：方法调用必须包含一对圆括号，调用无参方法也不例外。记住，方法名和圆括号是方法的标记。

需要强调的是，上面方法调用的例子都没有使用前缀，是假定方法定义和方法调用均在同一个类内部进行。调用一个类内部声明的方法，无须加上前缀，当然，在非静态方法中调用非静态方法也可加上代表本类对象的关键字 `this` 作前缀，例如 `this.method()`；对于静态方法，还可用类名作前缀。

下面给出一个完整的程序，用于说明方法声明和方法调用。

【例 3-1】 定义并调用方法，计算圆的周长和面积。

```
public class Ex1 {
```

```
static double calcArea(double r){           //定义计算圆面积方法
    double area;
    area = 3.14 * r * r;
    return area;
}

static double calcGirth(double r){         //定义计算圆周长方法
    return 2 * 3.14 * r;
}

public static void main(String[] args) {
    double radius, area;
    radius = 3.1;
    area = calcArea(radius);               //调用计算圆面积方法
    System.out.printf("半径为 3.1 的圆的面积: %.2f", area);
    area = calcArea(10);                   //调用计算圆面积方法
    System.out.printf("\n半径为 10 的圆的面积: %.2f", area);
    System.out.printf("\n半径为 3.1 的圆的周长: %.2f", calcGirth(3.1));
    System.out.printf("\n半径为 10 的圆的周长: %.2f", calcGirth(10));
                                           //调用计算圆周长方法
}
}
```

程序运行结果如图 3-2 所示。

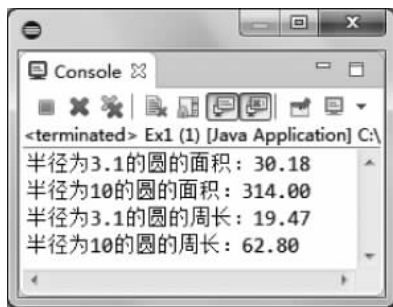


图 3-2 计算圆面积和周长

在例 3-1 程序中,输出面积和周长时调用了格式化输出方法 `printf`,以保留两位小数点。该方法详细说明见 3.4.2 小节。

3.4 在命令行窗口输入输出数据

在程序运行过程中,通常要进行数据输入输出的互动操作。在图形界面下传输数据,是通过文本框和标签等控件进行的。在字符界面的命令行窗口内传输数据,则通过调用系统预定义的方法进行。

注意: Eclipse 开发环境的命令行窗口是开发界面右下角的 Console(控制台)窗格。如果 Console 窗格没有出现,则可选择菜单 `Window|Show View|Console` 命令显示出来。

3.4.1 输入数据

调用 java.util 包的 Scanner 类对象的 nextBoolean、nextByte、nextShort、nextInt、nextLong、nextFloat、nextDouble、next 和 nextLine 等方法,可分别在命令行窗口(或其他输入源)中读入布尔型、字节型、短整型、整型、长整型、单精度浮点型、双精度浮点型、字符串和一行字符串等数据。

上述方法执行时,程序将停下来,等待用户在命令行窗口中输入数据,直到按 Enter 键确认,程序才继续运行下去。如果上述方法连续调用,即要输入多个数据,则各数据之间除了用 Enter 键分隔,也可用空格分隔(nextLine 方法除外)。

典型代码如下:

```
import java.util.Scanner;
...
Scanner scan = new Scanner(System.in);
double x = scan.nextDouble();
```

如果在命令行窗口中输入数据,则构建 Scanner 对象要使用标准输入流 System.in 作为构造方法的参数。使用 next 方法输入字符串时,由于空格用作分隔符,所以输入的字符串不能含有空格。不过,可调用 nextLine 方法输入一行(以 Enter 键)结束的字符串,这样该行字符串就允许存在空格了。

也可调用 java.io 包的 BufferedReader 类对象的 readLine 方法,输入一行允许存在空格的字符串。典型代码如下:

```
import java.io.*;
...
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
try{
    String str = br.readLine();
    ...
} catch(Exception e){}
```

构建 BufferedReader 类对象时要用到 InputStreamReader 对象,作为构造方法的参数。而构建 InputStreamReader 对象则以标准输入流 System.in 作为构造方法的参数。由于 readLine 方法会引发输入输出异常,故使用时要作异常处理,即编写 try-catch 代码块处理。

3.4.2 输出数据

调用标准输出流 System.out 的 println 和 print 方法,可在命令行窗口中输出字符串及各种基本型数据,其中,前一个方法名多了 ln,它是 line 单词的缩写,表示输出完数据后自动换行,而后一个方法则不会自动换行。

这两个方法都带有一个参数,参数可以是表达式形式。如果是由运算符+组成的表达式,则只要有一个操作数是字符串类型,都不做加法运算,而是执行字符串连接运算。例如:

```
double x = 2.1, y = 4;
System.out.println(x + " + " + y + " = " + (x+y));
```

方法 `println` 的参数是由 5 个十号组成的表达式,由于第一个十号右边是字符串 "+",因此该十号是串接运算符。最后的 $(x+y)$ 由于用小括号括起来,故里面的 + 运算优先执行,其左右两边都是数值,因此该十号才是加法运算符。其余 3 个十号也都是串接运算符。需要强调的是:字符串 "+" 里面的字符 + 不是运算符,只是照原样输出的普通字符。于是输出结果为:

```
2.1 + 4.0 = 6.1
```

注意: `println` 方法也可不带参数,这时只输出换行符,即自动换行。

除了上述两个方法外,从 JDK1.5 版开始,新增了与 C 语言 `printf` 函数类似的格式化输出方法 `printf`,该方法调用格式如下:

```
System.out.printf("格式控制字符串", 参数 1, 参数 2, ..., 参数 n)
```

其中,格式控制字符串由普通字符和格式控制符组成,普通字符照原样输出,格式控制符有 `%d`、`%f`、`%e`、`%c` 和 `%s` 等,用以输出后面的各参数的值。除方法第一个参数外,后面参数的个数要与格式控制符的个数一致。

格式控制符简介如下:

`%b`: 输出 boolean 型数。

`%d`: 输出 int 型数。

`%f`: 输出 float 或 double 浮点型数。

`%e`: 以指数形式输出 float 或 double 浮点型数。

`%c`: 输出 char 型数据。

`%s`: 输出 String 型数据。

输出数据的同时还可以控制数据的宽度。例如:

`%md`: 输出占 m 列的 int 型数。

`%.nf`: 输出小数保留 n 位的浮点型数。

`%m.nf`: 输出占 m 列、小数保留 n 位的浮点型数。

例如:

```
int radius = 10;
double area = 314;
System.out.printf("半径为 %d 的圆面积: %.2f", radius, area);
```

执行第 3 条语句,输出方法 `printf` 的第一个参数值,它是一个字符串,其中 `%d` 位置用 `radius` 值替换,`%.2f` 位置用 `area` 值替换并保留 2 位小数。输出结果如下:

```
半径为 10 的圆面积: 314.00
```

3.5 方法签名与方法重载

在一个类中允许定义多个同名的方法,条件是只要参数表不同就可以。

所谓参数表不同,是指参数的个数不同,或参数的类型和顺序不同。一个方法的方法名

和参数表,构成了“方法签名”。所以,只要方法签名不同,就允许在一个类中定义多个方法。

当定义两个以上名称相同而签名不同的方法时,就称为“方法重载”。

【例 3-2】 重载两数相加的方法,以便求两个整数或实数之和。

```
public class Ex2 {
    static int add(int a, int b){           //add 方法签名 1
        return a + b;
    }
    static double add(int a, double b) {   //add 方法签名 2
        return a + b;
    }
    static double add(double a, int b){    //add 方法签名 3
        return a + b;
    }
    static double add(double a, double b) { //add 方法签名 4
        return a + b;
    }
    public static void main(String[] args) { //主方法
        System.out.println("整整相加: " + add(1, 2));
        System.out.println("整实相加: " + add(2, 2.3));
        System.out.println("实整相加: " + add(3.4, 2));
        System.out.println("实实相加: " + add(4.1, 5.2));
    }
}
```

由于两数相加共有整整、整实、实整、实实 4 种搭配,因此在上述代码中,对两数相加的 add 方法进行重载,共有 4 个方法签名,并在 main 方法中依次调用了这 4 个方法。程序运行结果如图 3-3 所示。

需要说明的是,为了阐明方法重载的概念,例 3-2 特地编写了 4 个 add 方法,事实上,由于 int 型数能自动转换为 double 型数,因此只需最后一个 add 方法(签名 4),便能进行两个整、实数的混合加法运算。

Java 系统有很多方法重载的例子。例如 System.out.print 方法有 9 个签名, System.out.println 方法有 10 个签名,它们可直接输出布尔型、整型、浮点型、字符串型、字符型等数据,部分调用形式如下:

```
System.out.println(true);
System.out.println(8);
System.out.println(3.14);
System.out.println("abc");
System.out.println('A');
```

不但一般方法可重载,构造方法(也叫构造函数)也可重载。构造方法是一种构建类对象的特殊方法。

注意: 可以重载一个方法的参数,但不能重载方法的返回类型,因为返回类型不构成方法签名。例如,不能在一个类中同时重载下面两个“方法”:



图 3-3 方法重载运行结果

```
static int add(int a, int b){
    return a + b;
}
static double add(int a, int b) {           //出错：方法重复(而不是重载)
    return a + b;
}
```

3.6 方法参数值传递——单向传递

调用方法时,参数的传递方式都是值传递,即把实参的一个副本传给对应的形参。传递过来的参数,在执行方法过程中,方法体内部对参数的更改不会影响原来的数据。因此值传递属于单向传递:只从实参传给形参,不能从形参传回给实参。

【例 3-3】 测试方法参数的单向值传递。

```
public class Ex3 {
    static void change(int a){           //定义没有返回值的 change 方法
        a = 28;
    }
    public static void main(String[] args) {
        int age = 18;
        change(age);                   //调用 change 方法
        System.out.println(age);
    }
}
```

上述程序的运行结果是输出 18 而非 28,说明了在调用 change 方法时,只把实参 age 的值 18 传给方法的形参 a,在 change 方法执行完后,不会把 a 的值 28 再传回给 age,即只是单向传递参数。

即使把例 3-3 的 change 方法参数 a 换成 age(这时里面的语句变为 age=28;),运行结果也是 18。这时虽然从表面上看,参数 age 与 main 方法内部的 age 同名,但它们分别是两个不同方法的局部变量,运行时存放在不同的内存空间中,main 方法内部调用 change 方法时仍是单向传值,故结果不变。

试一试: 下面程序运行时输出什么结果?请上机验证您的判断。

```
public class Ex4 {
    static void change(String s){
        s = "123";
    }
    public static void main(String[] args) {
        String s = "abc";
        change(s);
        System.out.println(s);
    }
}
```

3.7 变量作用域

声明方法时可在方法头加上 `public`,使方法能被别的类(对象)调用,这时方法的作用域超出了其所在的类。但如果方法头用 `private` 声明,则方法的作用域只局限于它所定义的类,只能在类的内部调用。

方法具有作用域,变量也有作用域。作用域(Scope)就是能够使用的代码区域,是变量或方法发挥作用的领域和范围。

如果变量能在一个特定位置使用,该变量便具有那个位置的作用域。

变量与方法类似,只能在声明、定义之后才能使用。

变量根据所声明的位置和作用范围,分为局部变量和字段两种。

3.7.1 局部变量作用域

方法内声明的变量,包括圆括号内的方法参数,都是局部变量(local variable)。

局部变量只限于方法内部使用,作用域从变量声明开始,到方法体结束为止。

例如:

```
class MyClass {
    void method1(int a) {
        double x;
        ...
    }
    void method2() {
        x = 3.12;           //出错: 变量超出作用域
        a = 18;           //出错: 变量超出作用域
        ...
    }
}
```

类 `MyClass` 中,声明了 `method1` 和 `method2` 两个方法,第一个方法声明了整型参数 `a` 和 `double` 型变量 `x`,它们都是方法的局部变量,只能在本方法内部使用,但却在第二个方法中对这两个变量进行赋值,显然超出了变量的使用范围,因而产生语法错误。

注意: 方法内部的代码块(如复合语句),也可声明在其范围内使用的局部变量。

3.7.2 字段作用域

类体中,作为类成员声明的变量(不是在方法内部声明的),称为字段(field)。

方法体开、闭大括号之间建立了局部变量的作用域(方法范围内的作用域);界定类主体的一对开、闭大括号也建立了一个作用域(类范围内的作用域)。

字段作用域在类范围内。使用字段能在方法之间共享数据。例如:

```
class MyClass2 {
    void method1() {
        x = 3.12;           //赋值字段 x
    }
}
```

```
...
}
void method2() {
    System.out.println("字段 x 的值为" + x); //输出字段 x
    ...
}
private double x; //声明字段 x
}
```

类的组成部分称为类的成员。类 MyClass2 中声明了 3 个成员：私有字段 x、两个方法 method1 和 method2。字段 x 在两个方法内均可使用，因为字段的作用域是整个类。

注意：语法上，字段除了使用 private(私有的)修饰，也允许用 public(公共的)修饰。其中，private 字段只局限于本类使用，public 字段则像 public 方法一样，作用域突破了本类范围，可被其他类引用。不过，对象封装性要求字段为 private，因此一般不声明 public 字段。

需要强调的是，在方法中，必须先声明才能使用局部变量，因为语句是顺序执行的。但字段有所不同，字段的声明可以放在引用它的方法后面(参见上面类 MyClass2 的字段 x)。表面上，好像是“先使用再声明”，实际上，程序运行之前经过了编译阶段，在编译时系统已得知字段的声明。也就是说，如果字段之间不存在相互引用，则类成员之间的声明顺序无关紧要，正所谓“排名不分先后”。但成员方法内部，各语句之间的顺序不可随意颠倒。

3.8 本章小结

本章学习了方法定义和方法调用，方法定义包括方法头声明和方法体的建立。简单而言，方法是有名称的代码块。定义方法目的是减少重复编码，方便调用。调用方法即使用方法。方法是模块化编程的最小单位，是面向对象程序设计中的小模块(类则是较大的模块)。定义和调用方法是代码重用的体现。方法一次定义，便可反复调用。在命令行窗口中输入输出数据，就是通过调用系统预定义的方法来完成的。

Java 属于面向对象程序设计语言，方法定义和方法调用都在类的内部进行。一个类允许定义多个同名的方法，只要方法签名不同即可。所谓签名不同，就要参数个数、类型或顺序要有所不同。一个类定义了两个以上同名(但签名不同)的方法，称为方法重载。

方法的参数传递是值传递，方法调用时，把实参值传递给方法定义中的形参，再执行方法体语句，执行完毕，不会把形参值反过来再传回给实参。因此，方法的参数传递是单向传递，只从实参传给形参。

方法有使用范围，变量也有使用范围，这就是变量的作用域。变量分为两种：局部变量和字段。局部变量限于方法的内部使用，字段则以类成员的身份出现，因而可以被所在类的各个方法来引用。

本章的知识点归纳如表 3-1 所示。

表 3-1 本章知识点归纳

知 识 点	操作示例及说明
方法定义	<code>static double calcGirth(double r){ return 2 * 3.14 * r; }</code>
方法调用	<code>System.out.printf("半径为 10 的圆周长: %.2f", calcGirth(10));</code>
在命令行窗口输入数据	<code>Scanner scan = new Scanner(System.in); double x = scan.nextDouble(); ... BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); try{ String str = br.readLine(); ... } catch(Exception e){ }</code>
在命令行窗口输出数据	<code>System.out.println(x + y); System.out.printf("圆的面积: %.2f", area);</code>
方法签名与方法重载	<code>static int add(int a, int b){return a + b; } static double add(int a, double b) {return a + b; }</code>
方法参数值传递——单向传递	<code>static void change(int a){ a = 28; } ... int age = 18; change(age); System.out.println(age); //输出 18 而非 28</code>
局部变量作用域	<code>void method1(int a) { double x; ... //a 和 x 均只能在本方法内使用 }</code>
字段作用域	<code>class MyClass { void method1() { x = 3.12; ... } void method2() {System.out.println(x); ... } private double x; //声明字段 x }</code>

3.9 实训 3: 计算圆、矩形面积和周长

(1) 使用方法编写求圆面积和周长的程序,运行界面如图 3-1(a)所示,运行时提示输入圆半径,然后输出计算结果。

提示: 部分代码参考如下。

```
import java.util.Scanner;
public class ... {
    static double calcArea(double r){ return ... }
    ...
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        double radius, area, girth;
        System.out.println("请输入圆的半径: ");
        radius = scan.nextDouble( );
        area = ...
        System.out.printf("半径为 %.2f 的圆的面积: %.2f", radius, area);
        ...
    }
}
```

```
    }  
}
```

(2) 使用方法编写求矩形面积和周长的程序,运行界面如图 3-1(b)所示,运行时提示输入矩形的长度和宽度,然后输出计算结果。

提示: 部分代码参考如下。

```
import java.util.Scanner;  
public class Training2 {  
    static double calcArea(double len, double wid){ return ... }  
    ...  
    public static void main(String[] args) {  
        Scanner scan = new Scanner(System.in);  
        double length, width, area, girth;  
        System.out.println("请输入矩形的长度: ");  
        length = scan.nextDouble( );  
        ...  
        area = calcArea(length, width);  
        System.out.printf("长%.2f,宽%.2f 的矩形面积: %.2f", ...);  
        ...  
    }  
}
```