7.2 成员方法

7.2.1 基本介绍

在现实世界中，各类事物不仅具有属性，还具备行为。以人类为例，除了拥有年龄、姓名等属性外，还能进行说话、跑步等行为，甚至通过学习具备做算术题的能力。在 Java 编程中，为了完整地描述和处理这些具有行为特征的对象，我们需要定义成员方法（简称方法）。接下来，我们将对之前定义的`Person`类进行完善，使其能够体现人类的行为。

7.2.2 成员方法快速入门

示例代码（Method01.java）

```
public class Method01 {
    // 编写一个main方法
    public static void main(String[] args) {
        // 方法使用
        // 1. 方法写好后，如果不去调用(使用)，不会输出
        // 2. 先创建对象,然后调用方法即可
        Person p1 = new Person();
        p1.speak(); // 调用方法
        p1.cal01(); // 调用cal01方法
        p1.cal02(5); // 调用cal02方法，同时给n = 5
        p1.cal02(10); // 调用cal02方法，同时给n = 10
        // 调用getSum方法，同时num1=10, num2=20
        // 把方法getSum返回的值，赋给变量returnRes
        int returnRes = p1.getSum(10, 20);
        System.out.println("getSum方法返回的值=" + returnRes);
    }
}

class Person {
    String name;
    int age;

    // 方法(成员方法)
    // 添加speak成员方法,输出“我是一个好人”
    // 老韩解读
    // 1. public表示方法是公开
    // 2. void:  表示方法没有返回值
    // 3. speak() : speak是方法名,  ()形参列表
    // 4. {}方法体，可以写我们要执行的代码
    // 5. System.out.println("我是一个好人");表示我们的方法就是输出一句话
    public void speak() {
        System.out.println("我是一个好人");
    }

    // 添加cal01成员方法,可以计算从1+..+1000的结果
    public void cal01() {
        // 循环完成
        int res = 0;
```

```
        for (int i = 1; i <= 1000; i++) {
            res += i;
        }
        System.out.println("cal01方法 计算结果=" + res);
    }

    // 添加cal02成员方法,该方法可以接收一个数n，计算从1+..+n的结果
    // 老韩解读
    // 1. (int n)形参列表，表示当前有一个形参n, 可以接收用户输入
    public void cal02(int n) {
        // 循环完成
        int res = 0;
        for (int i = 1; i <= n; i++) {
            res += i;
        }
        System.out.println("cal02方法 计算结果=" + res);
    }

    // 添加getSum成员方法,可以计算两个数的和
    // 老韩解读
    // 1. public表示方法是公开的
    // 2. int :表示方法执行后，返回一个int值
    // 3. getSum方法名
    // 4. (int num1, int num2)形参列表，2个形参，可以接收用户传入的两个数
    // 5. return res;表示把res的值， 返回
    public int getSum(int num1, int num2) {
        int res = num1 + num2;
        return res;
    }
}
```

代码解读

1. `speak` **方法**：该方法用于输出一句话。`public` 修饰符表示此方法是公开的，其他类可以访问。`void` 表示该方法没有返回值。`speak` 是方法名，括号 `()` 内没有参数，意味着该方法不需要接收额外的数据。方法体 `{}` 中只有一条语句 `System.out.println("我是一个好人");`，执行该方法时会在控制台打印这句话。
2. `cal01` **方法**：用于计算从 1 累加到 1000 的结果。同样是 `public void` 类型，方法体内通过 `for` 循环实现累加计算，最后将结果输出到控制台。
3. `cal02` **方法**：与 `cal01` 类似，但它可以接收一个整数参数 `n`，计算从 1 累加到 `n` 的结果。在 `main` 方法中调用 `cal02` 时，传入不同的参数值，就可以得到不同范围的累加结果。
4. `getSum` **方法**：这是一个有返回值的方法。`public int` 表示该方法是公开的，并且执行后会返回一个 `int` 类型的值。方法名 `getSum`，形参列表 `(int num1, int num2)` 接收两个整数参数。方法体中计算两个参数的和，并通过 `return` 语句将结果返回。在 `main` 方法中调用 `getSum` 时，将返回值赋给 `returnRes` 变量，并输出。通过这些方法的定义和使用，我们可以看到如何在类中添加行为功能，并且学会如何调用这些方法来实现具体的操作。

7.2.3 方法的调用机制原理（重要！- 示意图）

以 `getSum` 方法为例，当在 `main` 方法中调用 `p1.getSum(10, 20);` 时，程序的执行过程如下：

1. 首先，程序在栈内存中为 `main` 方法分配一块空间（栈帧），用于存储 `main` 方法中的局部变量（如 `p1`、`returnRes` 等）和方法执行的相关信息。

2. 当执行到`p1.getSum(10, 20);`时，程序会在栈内存中为`getSum`方法分配一块新的栈帧。在这个栈帧中，会为形参`num1`和`num2`分配空间，并将实参`10`和`20`的值传递给形参。
3. 接着，程序进入`getSum`方法体执行。在方法体中，计算`num1`和`num2`的和，并将结果存储在局部变量`res`中。
4. 当遇到`return res;`语句时，`getSum`方法执行结束，将`res`的值返回给调用它的地方（即`main`方法中调用`getSum`的语句处）。同时，`getSum`方法的栈帧从栈内存中移除。
5. 在`main`方法中，将`getSum`方法返回的值赋给`returnRes`变量，继续执行`main`方法后续的代码。可以通过绘制如下示意图来更直观地理解这个过程：
   （此处插入一个简单的栈内存调用方法的示意图，展示`main`方法栈帧和`getSum`方法栈帧的创建、参数传递、返回值等过程）通过理解方法的调用机制，我们能更好地把握程序在执行过程中数据的传递和方法的执行顺序，有助于编写更高效、准确的代码。

7.2.4 为什么需要成员方法

需求引入

请遍历一个二维数组，并输出数组的各个元素值。例如数组`int [][] map = {{0,0,1},{1,1,1},{1,1,3}};`。

解决思路

1. **传统方法**：使用单个`for`循环嵌套来遍历数组并输出元素值。代码如下：

```
for(int i = 0; i < map.length; i++) {
    for(int j = 0; j < map[i].length; j++) {
        System.out.print(map[i][j] + "\t");
    }
    System.out.println();
}
```

这种方法虽然能够实现功能，但存在明显的问题。如果在程序的多个地方都需要遍历输出类似的数组，就需要重复编写这段代码，导致代码冗余度高，且后期维护困难。例如，当输出格式需要修改时，就需要在所有遍历数组的地方都进行修改，容易遗漏。
\2. **使用成员方法**：定义一个类`MyTools`，在其中编写一个成员方法来实现数组的遍历输出功能。代码如下：

```
public class Method02 {
    // 编写一个main方法
    public static void main(String[] args) {
        // 请遍历一个数组，输出数组的各个元素值
        int [][] map = {{0,0,1},{1,1,1},{1,1,3}};
        // 使用方法完成输出，创建MyTools对象
        MyTools tool = new MyTools();
        // 遍历map数组
        // 传统的解决方式就是直接遍历
        // for(int i = 0; i < map.length; i++) {
        //     for(int j = 0; j < map[i].length; j++) {
        //         System.out.print(map[i][j] + "\t");
        //     }
        //     System.out.println();
        // }
        // 使用方法
```

```
        tool.printArr(map);
        //....
        // 要求再次遍历map数组
        // for(int i = 0; i < map.length; i++) {
        //     for(int j = 0; j < map[i].length; j++) {
        //         System.out.print(map[i][j] + "\t");
        //     }
        //     System.out.println();
        // }
        tool.printArr(map);
        //...再次遍历
        // for(int i = 0; i < map.length; i++) {
        //     for(int j = 0; j < map[i].length; j++) {
        //         System.out.print(map[i][j] + "\t");
        //     }
        //     System.out.println();
        // }
        tool.printArr(map);
    }
}

// 把输出的功能，写到一个类的方法中,然后调用该方法即可
class MyTools {
    // 方法，接收一个二维数组
    public void printArr(int[][] map) {
        System.out.println("=======");
        // 对传入的map数组进行遍历输出
        for(int i = 0; i < map.length; i++) {
            for(int j = 0; j < map[i].length; j++) {
                System.out.print(map[i][j] + "_");
            }
            System.out.println();
        }
    }
}
```

在`main`方法中，创建`MyTools`类的对象`tool`，然后通过`tool.printArr(map);`调用`printArr`方法来输出数组。当需要再次遍历数组时，只需再次调用该方法即可。这种方式不仅减少了代码的重复，提高了代码的可读性和可维护性，还将数组遍历输出的实现细节封装在`printArr`方法中，其他地方只需要关心如何调用该方法，而不需要了解具体的遍历实现过程。

7.2.5 成员方法的好处

1. **提高代码的复用性**：通过将特定功能封装在方法中，在不同的地方需要实现相同功能时，只需调用该方法，而无需重复编写代码。如上述遍历数组的例子，`printArr`方法可以在多个需要遍历二维数组的地方被调用。
2. **封装实现细节**：将方法的实现细节隐藏起来，对外只提供一个调用接口。其他用户只需要知道方法的功能和如何调用，而不需要了解方法内部是如何实现的。这样可以降低程序的复杂性，提高代码的安全性和可维护性。例如，在`printArr`方法中，内部的双重`for`循环遍历数组的细节对于调用者来说是透明的，调用者只需要传入正确的数组参数并调用方法即可得到输出结果。

7.2.6 成员方法的定义

成员方法的定义语法如下：

```
访问修饰符 返回数据类型 方法名（形参列表..） {
    //方法体
    语句;
    return 返回值;
}
```

1. **形参列表**：表示成员方法输入，它可以接收调用方法时传入的数据。例如`cal(int n)`中，`n`就是形参，用于接收外部传入的一个整数；`getSum(int num1, int num2)`中有两个形参`num1`和`num2`，用于接收两个整数。形参列表可以为空，此时方法不需要接收额外的数据。
2. **返回数据类型**：表示成员方法输出。如果方法执行后有结果需要返回给调用者，就需要指定返回数据类型，如`int`、`double`、`String`等基本数据类型或引用类型。如果方法不需要返回任何值，则使用`void`关键字。
3. **方法主体**：为了实现某一功能的代码块，包含了一系列执行语句，用于完成方法的具体功能。例如在`cal01`方法中，`for`循环及相关语句构成了方法主体，实现了从 1 到 1000 的累加计算。
4. `return`**语句**：不是必须的。当方法有返回数据类型时，方法体中最后的执行语句必须为`return`语句，用于将计算结果返回给调用者，并且返回值类型必须和方法定义的返回数据类型一致或兼容。例如`getSum`方法中，`return res;`将计算得到的和`res`返回。如果方法返回类型是`void`，方法体中可以没有`return`语句，或者只写`return;`，此时`return`语句的作用是提前结束方法的执行。

7.2.7 注意事项和使用细节

示例代码（MethodDetail.java）

```java
public class MethodDetail {
    public static void main(String[] args) {
        AA a = new AA();
        int[] res = a.getSumAndSub(1, 4);
        System.out.println("和=" + res[0]);
        System.out.println("差=" + res[1]);
        // 细节: 调用带参数的方法时，一定对应着参数列表传入相同类型或兼容类型的参数
        byte b1 = 1;
        byte b2 = 2;
        a.getSumAndSub(b1, b2);//byte -> int
        //a.getSumAndSub(1.1, 1.8);//double ->int(x)
        // 细节: 实参和形参的类型要一致或兼容、个数、顺序必须一致
        //a.getSumAndSub(100);//x 个数不一致
        a.f3("tom", 10); //ok
        //a.f3(100, "jack"); // 实际参数和形式参数顺序不对
    }
}

class AA {
    // 细节: 方法不能嵌套定义
    public void f4() {
        //错误
        // public void f5() {
        // }
    }

    public void f3(String str, int n) {
    }
```

```
    // 1. 一个方法最多有一个返回值 [思考，如何返回多个结果 返回数组 ]
    public int[] getSumAndSub(int n1, int n2) {
        int[] resArr = new int[2];
        resArr[0] = n1 + n2;
        resArr[1] = n1 - n2;
        return resArr;
    }

    // 2. 返回类型可以为任意类型，包含基本类型或引用类型(数组，对象)
    // 具体看getSumAndSub
    //
    // 3. 如果方法要求有返回数据类型，则方法体中最后的执行语句必须为return值;
    // 而且要求返回值类型必须和return的值类型一致或兼容
    public double f1() {
        double d1 = 1.1 * 3;
        int n = 100;
        return n; // int ->double
        //return d1; //ok? double -> int
    }

    // 如果方法是void，则方法体中可以没有return语句，或者只写return ;
    // 老韩提示：在实际工作中，我们的方法都是为了完成某个功能，所以方法名要有一定含义
    // ，最好是见名知意
    public void f2() {
        System.out.println("hello1");
        System.out.println("hello1");
        System.out.println("hello1");
        int n = 10;
        //return ;
    }
}
```

细节解读

1. **方法参数类型匹配**：调用带参数的方法时，必须对应着参数列表传入相同类型或兼容类型的参数。例如，`AA`类中的`getSumAndSub`方法需要两个`int`类型的参数，当传入`byte`类型参数时，由于`byte`可以自动转换为`int`，所以是允许的，如`a.getSumAndSub(b1, b2);`。但如果传入`double`类型参数，因为`double`不能自动转换为`int`，则会报错，如`a.getSumAndSub(1.1, 1.8);`是错误的。
2. **参数个数和顺序**：实参和形参的个数、顺序必须一致。例如，`f3`方法需要一个`String`类型和一个`int`类型的参数，调用时必须按照这个顺序传入正确类型和数量的参数，`a.f3("tom", 10);`是正确的，而`a.f3(100, "jack");`则因为参数顺序错误会导致编译错误。
3. **方法不能嵌套定义**：在一个方法内部不能再定义另一个方法。

课后练习

一、选择题

1. 以下关于成员方法的定义，语法正确的是（）
   A. `void myMethod() { }`
   B. `public myMethod() { }`

C. `int myMethod { }`

D. `void myMethod;`

2. 若有方法定义 `public int add(int a, int b) { return a + b; }`，调用该方法正确的是（ ）

A. `add(1, 2);`

B. `int result = add(1, 2);`

C. `int result = add("1", "2");`

D. `add();`

3. 一个方法定义为 `public void printInfo(String name, int age) { }`，以下调用正确的是（ ）

A. `printInfo("Tom");`

B. `printInfo(10, "Jerry");`

C. `printInfo("Lucy", 20);`

D. `printInfo();`

4. 方法中`return`语句的作用是（ ）

A. 结束方法的执行并返回值（如果有返回值）

B. 只是结束方法的执行，不返回任何值

C. 可以在方法的任何位置使用，用于暂停方法执行

D. 用于定义方法的返回类型

5. 若方法定义为 `public double calculate(double num1, double num2) { return num1 * num2; }`，该方法的返回类型是（ ）

A. `int`

B. `double`

C. `void`

D. `String`

6. 以下关于方法的说法，错误的是（ ）

A. 方法可以提高代码的复用性

B. 方法不能嵌套定义

C. 一个方法可以有多个返回值

D. 方法名应遵循命名规范，最好见名知意

7. 定义方法 `public int[] getArray() { int[] arr = {1, 2, 3}; return arr; }`，调用该方法后得到的返回值类型是（ ）

A. `int`

B. `int[]`

C. `void`

D. `Object`

8. 若有类 `class MathUtils { public static int multiply(int a, int b) { return a * b; } }`，调用该方法正确的是（ ）

A. `MathUtils mu = new MathUtils(); mu.multiply(3, 4);`

B. `MathUtils.multiply(3, 4);`

C. `multiply(3, 4);`

D. `int result = multiply(3, 4);`

9. 方法定义为 `public void showMessage() { System.out.println("Hello, World!"); }`，该方法的返回类型是（ ）

A. `String`

B. `int`

C. `void`

D. `boolean`

10. 以下关于方法参数的说法，正确的是（ ）
    A. 实参和形参的类型必须完全一致，不能兼容
    B. 实参和形参的个数可以不一致
    C. 方法调用时，实参按顺序传递给形参
    D. 形参在方法调用结束后仍然存在于内存中

二、填空题

1. 成员方法定义中，`public`是__修饰符，用于控制方法的__范围。
2. 若方法有返回值，在方法体中必须使用__语句返回与返回类型一致或兼容的值。
3. 方法定义中，形参列表用于接收__，实参是在__方法时传递给形参的值。
4. 一个方法最多有__个返回值，若要返回多个结果，可以返回__。
5. 方法的返回类型可以是__类型或__类型。
6. 若方法定义为 `public void process() { }`，该方法的返回类型是__，表示__。
7. 方法名应遵循__命名法，最好能__。
8. 调用方法时，实参和形参的__、__和顺序必须一致。
9. 方法定义中，返回数据类型在__修饰符之后，方法名在__之后。
10. 若有方法 `public int getMax(int a, int b) { return a > b? a : b; }`，调用 `getMax(5, 3)` 的返回值是__。

三、判断题

1. 方法定义中，返回类型可以省略不写。（ ）
2. 一个类中只能定义一个成员方法。（ ）
3. 方法调用时，实参和形参的类型可以不一致，只要能进行类型转换即可。（ ）
4. 方法中可以没有`return`语句。（ ）
5. 方法名可以随意命名，不需要遵循任何规范。（ ）
6. 若方法返回类型是`void`，则方法体中不能有`return`语句。（ ）
7. 方法的形参在方法调用时分配内存空间。（ ）
8. 方法可以嵌套定义，即在一个方法内部可以再定义另一个方法。（ ）
9. 提高代码复用性是使用成员方法的好处之一。（ ）
10. 方法定义中，访问修饰符只能是`public`。（ ）

四、简答题

1. 简述成员方法的定义语法，并说明各部分的作用。
2. 说明方法调用机制的原理，以一个简单方法调用为例进行解释。
3. 为什么需要使用成员方法？它有哪些好处？
4. 方法的返回类型有什么作用？如果方法不需要返回值，应如何定义？
5. 实参和形参的区别是什么？在方法调用时它们是如何传递数据的？
6. 方法定义中，访问修饰符有哪些？它们的作用分别是什么？
7. 若一个方法需要返回多个值，应该如何实现？请举例说明。
8. 方法名的命名规范是什么？为什么要遵循这些规范？
9. 请举例说明方法定义和调用的过程，包括方法有返回值和无返回值的情况。
10. 简述方法在 Java 编程中的重要性。

五、编程题

1. 定义一个 `Calculator` 类，包含两个方法：`add` 用于计算两个整数的和，`subtract` 用于计算两个整数的差。在 `main` 方法中创建 `Calculator` 对象并调用这两个方法，输出计算结果。
2. 编写一个 `Circle` 类，包含半径属性 `radius`，定义一个方法 `calculateArea` 用于计算圆的面积（面积公式：`S = πr²`，`π` 取 3.14），在 `main` 方法中创建 `Circle` 对象，设置半径并调用 `calculateArea` 方法输出圆的面积。
3. 定义一个 `StringUtils` 类，包含一个方法 `reverseString` 用于将输入的字符串反转。在 `main` 方法中输入一个字符串，调用 `reverseString` 方法并输出反转后的字符串。
4. 编写一个 `ArrayUtils` 类，包含一个方法 `findMax` 用于找出整型数组中的最大值。在 `main` 方法中创建一个整型数组，调用 `findMax` 方法并输出数组中的最大值。
5. 定义一个 `Person` 类，包含姓名 `name` 和年龄 `age` 属性，编写一个方法 `printInfo` 用于输出个人信息（格式为：姓名是 [姓名]，年龄是 [年龄]）。在 `main` 方法中创建 `Person` 对象，设置姓名和年龄后调用 `printInfo` 方法。
6. 编写一个 `MathUtils` 类，包含一个方法 `isPrime` 用于判断一个整数是否为质数。在 `main` 方法中输入一个整数，调用 `isPrime` 方法并输出判断结果。
7. 定义一个 `Rectangle` 类，包含长 `length` 和宽 `width` 属性，编写两个方法：`calculatePerimeter` 用于计算矩形的周长，`calculateArea` 用于计算矩形的面积。在 `main` 方法中创建 `Rectangle` 对象，设置长和宽后调用这两个方法并输出结果。
8. 编写一个 `Student` 类，包含姓名 `name`、成绩 `score` 属性，编写一个方法 `getGrade` 用于根据成绩返回对应的等级（90 分及以上为 'A'，80 - 89 分为 'B'，70 - 79 分为 'C'，60 - 69 分为 'D'，60 分以下为 'E'）。在 `main` 方法中创建 `Student` 对象，设置成绩后调用 `getGrade` 方法并输出等级。
9. 定义一个 `FileUtils` 类，包含一个方法 `countWords` 用于统计一个字符串中单词的个数（假设单词之间用空格分隔）。在 `main` 方法中输入一个字符串，调用 `countWords` 方法并输出单词个数。
10. 编写一个 `Employee` 类，包含姓名 `name`、工资 `salary` 属性，编写一个方法 `raiseSalary` 用于给员工涨薪（涨薪幅度为 10%）。在 `main` 方法中创建 `Employee` 对象，设置工资后调用 `raiseSalary` 方法，输出涨薪后的工资。