

第 5 天 Java 基础语法

今日内容介绍

◆ 方法

第1章 方法

1.1 方法概述

在我们的日常生活中，方法可以理解为要做某件事情，而采取的解决办法。

如：小明同学在路边准备坐车来学校学习。这就面临着一件事情（坐车到学校这件事情）需要解决，解决办法呢？可采用坐公交车或坐出租车的方式来学校，那么，这种解决某件事情的办法，我们就称为方法。

在 java 中，方法就是用来完成解决某件事情或实现某个功能的办法。

方法实现的过程中，会包含很多条语句用于完成某些有意义的功能——通常是处理文本，控制输入或计算数值。

我们可以通过在程序代码中引用方法名称和所需的参数，实现在该程序中执行（或称调用）该方法。方法，一般都有一个返回值，用来作为事情的处理结果。

1.2 方法的语法格式

在 Java 中，声明一个方法的具体语法格式如下：

```
修饰符 返回值类型 方法名 (参数类型 参数名 1, 参数类型 参数名 2, ..... ) {  
    执行语句  
    .....  
    return 返回值;  
}
```

对于上面的语法格式中具体说明如下：

- 修饰符：方法的修饰符比较多，有对访问权限进行限定的，有静态修饰符 **static**，还有最终修饰符 **final** 等，这些修饰符在后面的学习过程中会逐步介绍
- 返回值类型：用于限定方法返回值的数据类型
- 参数类型：用于限定调用方法时传入参数的数据类型
- 参数名：是一个变量，用于接收调用方法时传入的数据
- **return** 关键字：用于结束方法以及返回方法指定类型的值
- 返回值：被 **return** 语句返回的值，该值会返回给调用者

需要特别注意的是，方法中的“参数类型 参数名 1, 参数类型 参数名 2”被称作参数列表，它用于描述方法在被调用时需要接收的参数，如果方法不需要接收任何参数，则参数列表为空，即()

内不写任何内容。方法的返回值必须为方法声明的返回值类型，如果方法中没有返回值，返回值类型要声明为 `void`，此时，方法中 `return` 语句可以省略。

接下来通过一个案例来演示方法的定义与使用，如下图所示。MethodDemo01.java

```
public class MethodDemo01 {  
    public static void main(String[] args) {  
        int area = getArea(3, 5); // 调用 getArea 方法  
        System.out.println(" The area is " + area);  
    }  
  
    // 下面定义了一个求矩形面积的方法，接收两个参数，其中 x 为高，y 为宽  
    public static int getArea(int x, int y) {  
        int temp = x * y; // 使用变量 temp 记住运算结果  
        return temp; // 将变量 temp 的值返回  
    }  
}
```

运行结果如下图所示。

```
D:\java>java MethodDemo03  
The area is 15
```

图1-1 运行结果

在上述代码中，定义了一个 `getArea()` 方法用于求矩形的面积，参数 `x` 和 `y` 分别用于接收调用方法时传入的高和宽，`return` 语句用于返回计算所得的面积。在 `main()` 方法中通过调用 `getArea()` 方法，获得矩形的面积，并将结果打印。

1.3 方法调用图解

接下来通过一个图例演示 `getArea()` 方法的整个调用过程，如下图所示。

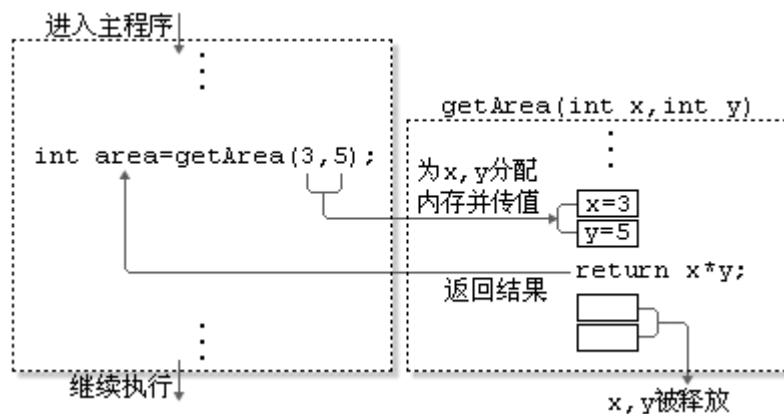


图1-2 `getArea()` 方法的调用过程

从上图中可以看出，在程序运行期间，参数 `x` 和 `y` 相当于在内存中定义的两个变量。当调用 `getArea()` 方法时，传入的参数 `3` 和 `5` 分别赋值给变量 `x` 和 `y`，并将 `x*y` 的结果通过 `return` 语句返回，整个方法的调用过程结束，变量 `x` 和 `y` 被释放。

1.4 方法定义练习

分别定义如下方法：

定义无返回值无参数方法，如打印 3 行，每行 3 个*号的矩形

定义有返回值无参数方法，如键盘录入得到一个整数

定义无返回值有参数方法，如打印指定 M 行，每行 N 个*号的矩形

定义有返回值有参数方法，如求三个数的平均值

- 无返回值无参数方法，如打印 3 行，每行 3 个*号的矩形

```
public static void printRect(){
    //打印 3 行星
    for (int i=0; i<3; i++) {
        //System.out.println("***"); 相当于是打印 3 颗星，换行
        //每行打印 3 颗星
        for (int j=0; j<3; j++) {
            System.out.print("*"); // ***
        }
        System.out.println();
    }
}
```

- 有返回值无参数方法，如键盘录入得到一个整数

```
public static int getNumber(){
    Scanner sc = new Scanner(System.in);
    int number = sc.nextInt();
    return number;
}
```

- 无返回值有参数方法，如打印指定 M 行，每行 N 个*号的矩形

```
public static void printRect2(int m, int n){
    //打印 M 行星
    for (int i=0; i<m; i++) {
        //每行中打印 N 颗星
        for (int j=0; j<n; j++) {
            System.out.print("*");
        }
        System.out.println();
    }
}
```

- 有返回值有参数方法，如求三个数的平均值

```
public static double getAvg(double a, double b, double c) {
    double result = (a+b+c)/3;
    return result;
}
```

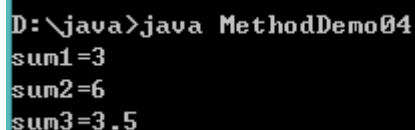
```
}
```

1.5 方法的重载

我们假设要在程序中实现一个对数字求和的方法，由于参与求和数字的个数和类型都不确定，因此要针对不同的情况去设计不同的方法。接下来通过一个案例来实现对两个整数相加、对三个整数相加以及对两个小数相加的功能，具体实现如下所示。MethodDemo02.java

```
public class MethodDemo02 {  
    public static void main(String[] args) {  
        // 下面是针对求和方法的调用  
        int sum1 = add01(1, 2);  
        int sum2 = add02(1, 2, 3);  
        double sum3 = add03(1.2, 2.3);  
        // 下面的代码是打印求和的结果  
        System.out.println("sum1=" + sum1);  
        System.out.println("sum2=" + sum2);  
        System.out.println("sum3=" + sum3);  
    }  
  
    // 下面的方法实现了两个整数相加  
    public static int add01(int x, int y) {  
        return x + y;  
    }  
  
    // 下面的方法实现了三个整数相加  
    public static int add02(int x, int y, int z) {  
        return x + y + z;  
    }  
  
    // 下面的方法实现了两个小数相加  
    public static double add03(double x, double y) {  
        return x + y;  
    }  
}
```

运行结果如下图所示。



```
D:\java>java MethodDemo04  
sum1=3  
sum2=6  
sum3=3.5
```

图1-3 运行结果

从上述代码不难看出，程序需要针对每一种求和的情况都定义一个方法，如果每个方法的名称都不相同，在调用时就很难分清哪种情况该调用哪个方法。

为了解决这个问题，Java 允许在一个类中定义多个名称相同的方法，但是参数的类型或个数必须不同，这就是方法的重载。

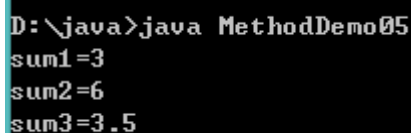
下面的三个方法互为重载关系

- `public static int add(int x, int y)` {逻辑} //两个整数加法
- `public static int add(int x, int y, int z)` {逻辑} //三个整数加法
- `public static int add(double x, double y)` {逻辑} //两个小数加法

接下来通过方法重载的方式进行修改，如下所示。MethodDemo03.java

```
public class MethodDemo03 {  
    public static void main(String[] args) {  
        // 下面是针对求和方法的调用  
        int sum1 = add(1, 2);  
        int sum2 = add(1, 2, 3);  
        double sum3 = add(1.2, 2.3);  
        // 下面的代码是打印求和的结果  
        System.out.println("sum1=" + sum1);  
        System.out.println("sum2=" + sum2);  
        System.out.println("sum3=" + sum3);  
    }  
  
    // 下面的方法实现了两个整数相加  
    public static int add(int x, int y) {  
        return x + y;  
    }  
  
    // 下面的方法实现了三个整数相加  
    public static int add(int x, int y, int z) {  
        return x + y + z;  
    }  
  
    // 下面的方法实现了两个小数相加  
    public static double add(double x, double y) {  
        return x + y;  
    }  
}
```

MethodDemo02.java 的运行结果和 MethodDemo03.java 一样，如下图所示。



```
D:\java>java MethodDemo05  
sum1=3  
sum2=6  
sum3=3.5
```

图1-4 运行结果

上述代码中定义了三个同名的 `add()` 方法，它们的参数个数或类型不同，从而形成了方法的重载。

在 `main()` 方法中调用 `add()` 方法时，通过传入不同的参数便可以确定调用哪个重载的方法，如 `add(1,2)` 调用的是两个整数求和的方法。值得注意的是，方法的重载与返回值类型无关，它只有两个条件，一是方法名相同，二是参数个数或参数类型不相同。

1.5.1 重载的注意事项

- 重载方法参数必须不同：
 - 参数个数不同，如 `method(int x)` 与 `method(int x,int y)` 不同
 - 参数类型不同，如 `method(int x)` 与 `method(double x)` 不同
 - 参数顺序不同，如 `method(int x,double y)` 与 `method(double x,int y)` 不同
- 重载只与方法名与参数类型相关与返回值无关
 - 如 `void method(int x)` 与 `int method(int y)` 不是方法重载，不能同时存在
- 重载与具体的变量标识符无关
 - 如 `method(int x)` 与 `method(int y)` 不是方法重载，不能同时存在

1.5.2 参数传递

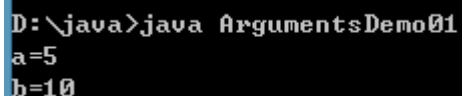
参数传递，可以理解当我们要调用一个方法时，我们会把指定的数值，传递给方法中的参数，这样方法中的参数就拥有了这个指定的值，可以使用该值，在方法中运算了。这种传递方式，我们称为参数传递。

- 在这里，定义方法时，参数列表中的变量，我们称为形式参数
- 调用方法时，传入给方法的数值，我们称为实际参数

我们看下面的两段代码，来明确下参数传递的过程：

```
public class ArgumentsDemo01 {  
    public static void main(String[] args) {  
        int a=5;  
        int b=10;  
  
        change(a, b); //调用方法时，传入的数值称为实际参数  
        System.out.println("a=" + a);  
        System.out.println("b=" + b);  
    }  
  
    public static void change(int a, int b) { //方法中指定的多个参数称为形式参数  
        a=200;  
        b=500;  
    }  
}
```

程序的运行结果如下：



```
D:\java>java ArgumentsDemo01  
a=5  
b=10
```

再看另一段代码

```
public class ArgumentsDemo02 {  
    public static void main(String[] args) {  
        int[] arr = { 1, 2, 3 };  
  
        change(arr); // 调用方法时，传入的数值称为实际参数
```

```

        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i]);
        }
    }

    public static void change(int[] arr) { // 方法中指定的多个参数称为形式参数
        for (int i = 0; i < arr.length; i++) {
            arr[i] *= 2;
        }
    }
}

```

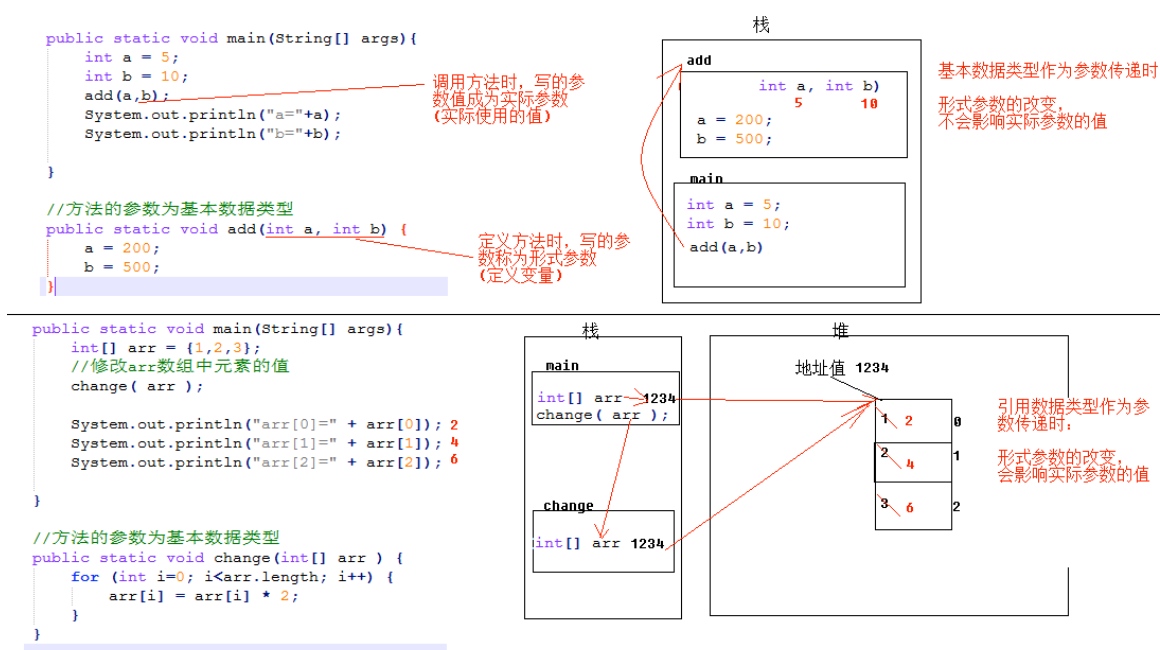
程序的运行结果如下：

```

D:\java>java ArgumentsDemo02
2
4
6

```

1.5.3 参数传递图解与结论



通过上面的两段程序可以得出如下结论：

- 当调用方法时，如果传入的数值为基本数据类型（包含 String 类型），形式参数的改变对实际参数不影响
- 当调用方法时，如果传入的数值为引用数据类型（String 类型除外），形式参数的改变对实际参数有影响

第2章 随机点名器案例

2.1 案例介绍

随机点名器，即在全班同学中随机的打印出一名同学名字。

要做的随机点名器，它具备以下 3 个内容：

- 存储所有同学姓名
- 总览全班同学姓名
- 随机点名其中一人，打印到控制台

2.2 案例需求分析

在全班同学中随机的打印出一名同学名字。

我们对本案例进行分析，得出如下分析结果：

1. 存储全班同学名字
2. 打印全班同学每一个人的名字
3. 在班级总人数范围内，随机产生一个随机数，查找该随机数所对应的同学名字

该案例须有以下 3 个内容：

- 存储所有同学姓名
- 总览全班同学姓名
- 随机点名其中一人，打印到控制台

随机点名器明确地分为了三个功能。如果将多个独立功能的代码写到一起，则代码相对冗长，我们可以针对不同的功能可以将其封装到一个方法中，将完整独立的功能分离出来。

而在存储同学姓名时，如果对每一个同学都定义一个变量进行姓名存储，则会出现过多孤立的变量，很难一次性将全部数据持有。此时，我们可以使用数组解决多个数据的存储问题。

2.3 实现代码步骤

编写 CallName.java 文件，完成程序的编写。

- main 方法中调用三个独立方法

```
public static void main(String[] args) {  
    System.out.println("-----随机点名器-----");  
    // 创建一个存储多个同学名字的容器（数组）  
    String[] students = new String[3];  
    /*  
     * 1. 存储全班同学名字  
     */  
    addStudentName(students);  
    /*  
     * 2. 打印全班同学每一个人的名字  
     */  
}
```



```
printStudentName(students);  
/*  
 * 3.获取随机点名到的学生姓名,并打印  
 */  
String randomName = randomStudentName(students);  
System.out.println("被点到名的同学是：" + randomName);  
}
```

- 1. 存储所有同学姓名

```
/**  
 * 1.存储全班同学名字  
 * 创建一个存储多个同学名字的容器（数组）  
 * 键盘输入每个同学的名字,存储到容器中（数组）  
 */  
public static void addStudentName(String[] students) {  
    //键盘输入多个同学名字存储到容器中  
    Scanner sc = new Scanner(System.in);  
    for (int i = 0; i < students.length; i++) {  
        System.out.println("存储第"+i+"个名称:");  
        students[i] = sc.next();  
    }  
}
```

上述方法中,通过键盘录入,完成为指定数组元素赋值。方法定义时,将参数定义为字符串数组,用于接收存放的同学姓名。

- 打印全班同学每一个人的名字

```
/**  
 * 2 打印全班同学每一个人的名字  
 */  
public static void printStudentName(String[] students) {  
    //遍历数组,得到每个同学名字  
    for (int i = 0; i < students.length; i++) {  
        String name = students[i];  
        //打印同学名字  
        System.out.println("第"+i+"个学生名称: " + name);  
    }  
}
```

上述方法中,方法参数 `students` 数组中存储了所有学生的姓名。通过遍历将数组中的每一个元素访问一遍,得到每一个同学名称。

- 3. 获取随机点到的学生姓名

```
/**  
 * 3.在班级总人数范围内,随机产生一个随机数,返回随机数位置上的学生姓名  
 */  
public static String randomStudentName(String[] students) {  
    //根据数组长度,获取随机索引
```

```
int index = new Random().nextInt(students.length);  
//通过随机索引从数组中获取名称  
String name = students[index];  
//返回随机点到的名称  
return name;  
}
```

上述方法中，通过随机数类 `Random` 产生一个从 0 到数组长度的随机索引。使用该索引获取 `students` 数组中对应的值，便得到了全班同学的随机姓名。

第3章 库存管理案例

3.1 案例介绍

现在，我们将原有的库存管理案例，进行业务逻辑的封装。

```
-----库存管理-----  
1.查看库存清单  
2.修改商品库存数量  
3.退出  
请输入要执行的操作序号:  
_
```

将对下列功能进行方法封装：

- 打印库存清单功能
- 库存商品数量修改功能
- 退出程序功能

3.2 案例需求分析

管理员能够进行的操作有 3 项（查看、修改、退出），我们可以采用（`switch`）菜单的方式来完成。

```
-----库存管理-----  
1.查看库存清单  
2.修改商品库存数量  
3.退出
```

请输入要执行的操作序号：

每一项功能操作，我们采用方法进行封装，这样，可使程序的可读性增强。

选择“1.查看库存清单”功能，则控制台打印库存清单

选择“2.修改商品库存数量”功能，则对每种商品库存数进行更新

选择“3.退出”功能，则退出库存管理，程序结束

3.3 实现代码步骤

编写代码 `Demo 库存管理.java`，完成如下功能：

- 功能菜单

```
/**
 * 库存管理功能菜单
 * @return 管理员键盘输入的功能操作序号
 */
public static int chooseFunction() {
    System.out.println("-----库存管理-----");
    System.out.println("1.查看库存清单");
    System.out.println("2.修改商品库存数量");
    System.out.println("3.退出");
    System.out.println("请输入要执行的操作序号: ");
    //接收键盘输入的功能选项序号
    Scanner sc = new Scanner(System.in);
    int choose = sc.nextInt();
    return choose;
}
```

上述方法用来完成库存管理功能菜单的显示、接收管理员选择的功能操作序号。这是完成了案例的第一步。接下来完成“查看、修改、退出”这三项功能。

- 编写 main 主方法，调用库存管理功能菜单方法，与“查看、修改、退出”这三个方法。

```
public static void main(String[] args) {
    //记录库存商品信息
    //品牌型号
    String[] brands = new String[]{"MacBookAir", "ThinkpadT450"};
    //尺寸大小
    double[] sizes = new double[]{13.3, 14.0};
    //价格
    double[] prices = new double[]{6988.88, 5999.99};
    //库存个数
    int[] counts = new int[]{0, 0};

    //通过 while 循环模拟管理员进行功能重复选择操作
    while (true) {
        //打印功能菜单操作,接收键盘输入的功能选项序号
        int choose = chooseFunction();
        //执行序号对应的功能
        switch (choose) {
            case 1://查看库存清单
                printStore(brands, sizes, prices, counts);
                break;
            case 2://修改商品库存数量
                update(brands, counts);
                break;
            case 3://退出
                exit();
        }
    }
}
```

```

        return;
    default:
        System.out.println("-----");
        System.out.println("功能选择有误，请输入正确的功能序号!");
        break;
    }
}
}
}

```

在主方法中，创建了 5 个数组，分别用来存储商品的品牌型号、尺寸大小、价格、配置、库存个数，通过接收到的功能选项序号，进行 switch 语句判断后，调用对应的功能方法。

● 查看库存清单功能

```

/**
 * 查看库存清单
 * @param brands 商品品牌型号
 * @param sizes 商品尺寸大小
 * @param prices 商品价格
 * @param counts 商品库存个数
 */
public static void printStore(String[] brands, double[] sizes, double[] prices,
int[] counts) {
    //统计总库存个数、统计库存总金额
    int totalCount = 0;
    double totalMoney = 0.0;
    for (int i = 0; i < brands.length; i++) {
        totalCount += counts[i];
        totalMoney += counts[i] * prices[i];
    }
    //列表顶部
    System.out.println("----- 查 看 库 存 清 单
-----");
    System.out.println("品牌型号      尺寸 价格 库存数");
    //列表中部分
    for (int i = 0; i < brands.length; i++) {
        System.out.println(brands[i]+" "+sizes[i]+" "+prices[i]+"
"++counts[i]);
    }
    //列表底部

    System.out.println("-----
-");

    System.out.println("总库存数: "+totalCount);
    System.out.println("库存商品总金额: "+totalMoney);
}
}

```

上述方法用来完成打印库存清单功能，5 个方法参数用来打印的库存商品相关信息

- 修改商品库存数量功能

```
/**
 * 修改商品库存数量
 * @param brands 商品品牌型号
 * @param counts 商品库存个数
 */
public static void update(String[] brands, int[] counts){
    System.out.println("-----修改商品库存数量-----");
    for (int i = 0; i < brands.length; i++) {
        System.out.println("请输入"+ brands[i] +"商品库存数");
        counts[i] = new Scanner(System.in).nextInt();
    }
}
```

上述方法用来完成修改商品库存数量功能，2 个方法参数用来指定所要修改的商品与库存数

- 退出功能

```
/**
 * 退出
 */
public static void exit(){
    System.out.println("-----退出-----");
    System.out.println("您已退出系统");
}
```

上述方法用来完成退出程序的功能

第4章 总结

4.1 知识点总结

- 方法

- 格式:

```
修饰符 返回值类型 方法名(参数类型 参数名 1, 参数类型 参数名 2, ...){
    方法体;
    return 返回值;
}
```

- 方法使用的注意事项:

- 1, 方法不调用，自己不执行
- 2, 方法中不能定义方法，但是，方法中可以调用方法
- 3, 方法定义的位置在类中，其他方法的外面
- 4, 如果方法没有明确的返回值类型，使用'空'类型， void 表示
- 5, void 只能在方法返回值类型位置使用，不能作为 普通的数据类型使用
- 6, 如果方法返回值类型为 void 类型，可以省略 return ;

- 方法调用方式:

有明确返回值类型的方法调用方式：

- 1，单独调用
- 2，输出调用
- 3，赋值调用

没有明确返回值类型的方法调用方式：

- 1，单独调用

■ 方法重载：

方法重载，在同一个类中，出现了多个同名的方法，他们的参数列表不同（参数列表的个数不同，参数列表的数据类型不同，参数列表的顺序不同）。

方法重载特点：

与方法的返回值类型无关，与方法的参数名无关，只看方法名与参数列表；

方法重载，是通过 JVM 来完成同名方法的调用的，通过参数列表来决定调用的是哪一个方法。