3. 方法与数组

3.1. 方法的定义

- 方法（又叫函数）就是一段特定功能的代码块。方法提高程序的复用性和可读性
- 方法的格式：

```
访问权限修饰符 [其他修饰符(static)] 返回值类型 方法名 (参数类型1 形参1, 参数类型2 形参2,...) {
    // 形参列表
    // 方法体
    return 返回值;
}
```

  - 格式说明：
    - 修饰符：public，private，protected等等
    - 返回值类型：就是功能结果的数据类型
    - 方法名：见名知意，首字母小写，遵守驼峰命名法

3.2. 方法的形参与实参

- 实际参数：就是实际参与运算的
- 形式参数：就是方法定义上的，用于接收实际参数的
- 参数类型：就是参数的数据类型
- 参数名：就是变量名
- 方法体语句：就是完成功能的代码
- 注意：
  - 若当前方法中不要使用形参，那么形参列表可以为空

- 实参和形参的类型要相互兼容，而且实参的取值范围要小于或者等于形参类型的取值范围
- 在调用方法中，如果我们定义的方法有参数，就必须在调用方法的同时传入这个值，即给当前方法中的参数赋值，而这个传入的值我们称之为实际参数，也就是实参

3.3. 方法的返回值

- return：结束方法的
- 返回值：就是功能的结果，由return带给调用者
- 注意：
  - 若当前方法没有返回值类型，即返回值类型是void，那么当前方法中可以不写return
  - return即表示结束一个方法，也可以将返回值返回给调用当前方法的调用者
  - return返回值时一次只能返回一个值，不可以返回多个值
  - 一个方法中可以有多个return，但被执行的只能是一个，所以需要判断

3.4. 方法的重载

- 方法重载：overloading method
- 在类中可以创建多个方法，它们具有相同的名字，但具有不同的参数和不同的定义
- 注意：返回值不能作为重载的条件

3.5. Java数组

- 数组的定义：一组能够存储相同数据类型值得变量的集合
- 数组的赋值方式：
  - 使用默认的初始值来初始化数组中的每一个元素

```
// 语法：数组元素类型[] 数组名 = new 数组元素类型[数组中元素的个数（数组的长度）];
int[] scores = new int[3];
```

  - 先声明，然后再赋予默认的初始值

```
// 语法：数组元素类型[] 数组名;
//       数组名 = new 数组元素类型[数组中元素的个数（数组的长度）];
int[] scores;
scores = new int[3];
```

- 先声明，然后再使用指定的值进行初始化

```
// 语法：数组元素类型[] 数组名 = new 数组元素类型[]{元素1，元素2,...};
int[] scores = new int[]{56，34，89};
```

- 将第三种写法可以简化为（使用数组常量值给数组进行赋值）

```
// 语法：数组元素类型[] 数组名 = {元素1，元素2,...};
int[] scores = {56，34，89};
```

- 数组的遍历：
  - 求数组长度：数组名.length
  - 通过下标来访问数组中的元素：下标从0开始，到数组长度-1结束
  - 遍历：依次取出数组中的每一个元素
    - 普通的for循环：

```
// 语法
for (int i = 0; i < 数组的长度; i++) {
    // i：循环变量，同样也是数组的下标（取值范围[0，数组长度]）
    数组中元素的类型 变量 = 数组名[i];
}
```

    - 使用增强for循环[foreach循环]：

```
// 语法
for (数组中元素的类型 变量 : 数组名) {
    数组中元素的类型 临时变量 = 变量;
}

// 结合方法的定义，可以使用可变参数来代替数组作为参数
public static void print(int...变量名) {
    // 可变参数在使用时作为数组使用
}
```

    - 注意的问题：
      - 空指针异常（NullPointerException）
      - 数组越界异常（ArrayIndexOutofBoundsException）
- 数组内存分析：
  - 栈内存：大小固定，用于存储局部变量，临时变量（基本数据类型）和引用变量
  - 堆内存：大小不固定，用于存储对象
  - 数组是引用类型，会存放在堆内存中

- 多维数组：Java中没有真正的多维数组，多维数组的表示方式是数组中的元素还是数组

3.6. 冒泡排序算法

```
// BubbleSort
for (int i = 0; i < arrays.length - 1; i++) {
    for (int j = 0; j < arrays.length - 1 - i; j++) {
        if (arrays[j] > arrays[j + 1]) {
            swap(arrays[j], arrays[j + 1]);
        }
    }
}
```

- 算法执行过程：
    - 比较相邻的元素，如果第一个比第二个大，就交换它们两个
    - 对每一对相邻元素做相同的工作，从开始第一对到结尾最后一对，执行一次之后，最后面的元素应该是最大数
    - 针对所以元素重复上述操作，除了最后一个
    - 持续每次对越来越少的元素重复上述步骤，直到没有任何一对数字需要比较
- 注意：相同元素的前后顺序并没有改变，所以冒泡排序是一种稳定的排序算法

3.7. 选择排序算法

```
// SelectSort
int minIndex = 0;// 记录数组中的最小值的下标
for (int i = 0; i < arrays.length - 1; i++) {
    minIndex = i;// 每轮设定一个最小值下标
    for (int j = i + 1; j < arrays.length; j++) {
        if (arrays[j] > arrays[minIndex]) {
            minIndex = j;
        }
    }
    // 判断需要交换的数下标是否是自己
    if (minIndex != i) {
        arrays[minIndex] = arrays[minIndex] + arrays[i];
        arrays[i] = arrays[minIndex] - arrays[i];
        arrays[minIndex] = arrays[minIndex] - arrays[i];
    }
}
```

- 算法执行过程：
    - 每一趟从待排序的数据元素中选出最小（或最大）的一个元素，顺序放在已排序好的数列的最后，直到全部待排序的数据元素排完
- 注意：选择排序是不稳定的排序方法

3.8. 直接插入排序算法

```
// DirectInsertionSort
for (int i = 1; i < arrays.length; i++) {
    int temp = arrays[i];// 记录操作数
    for (int j = i - 1; j >= 0; j--) {
        if (arrays[j] > temp) {
            arrays[j + 1] = arrays[j];
        } else {
            break;
        }
    }
    if (arrays[j + 1] != temp) {
        arrays[j + 1] = temp;
    }
}
```

- 算法执行过程：
    - 每步将一个待排序的记录，按其顺序码大小插入到前面已经排序的子序列的合适位置（从后向前找到合适位置后），直到全部插入排序完为止

3.9. 二分查找算法

```
// BinarySearch
public int binarySearch(int[] arrays, int key) {}
    int start = 0;// 开始下标
    int end = arrays.length - 1;// 结束下标
    while (start <= end) {
        int middle = (start + end) / 2;
        if (arrays[middle] > key) {
            end = middle - 1;
        } else if (arrays[middle] < key) {
            start = middle + 1;
        } else {
            return middle;
        }
    }
    return -1;
}
```

- 二分查找（折半查找）：就是在已经排好序的数组中，通过将待查找的元素与中间索引值相应的元素进行比较，若大于中间索引值对应的元素，去右半部分查找，否则去左半部分查找。以此类推，直到找到为止，找不到返回一个负数

3.10. Arrays类

- 用来操作数组（比如排序和搜索）的各种方法
- 常用方法：

- 二分查找：

```
Arrays.binarySearch(int[] array, int value);
```

- 数组内容转成字符串的形式输出：

```
Arrays.toString(int[] array);
```

- 数组排序：

```
Arrays.sort(int[] array);
```

- 复制指定的数组：

```
Arrays.copyOf(int[] array, inr length);
Arrays.copyOf(int[] array, int from, int to);
System.arraycopy(Object src, int srcPos, Object dest, int destPos,
int length);
```

- 判断两个数组是否相等：

```
Arrays.equals();
```

- 使用指定元素填充数组：

```
Arrays.fill();
```