Array 数组

Introduction 数组介绍

数组变量声明方式：

```
double[] myList = new double[10];

// 在声明时直接初始化数组
int[] arr = {1, 2, 3, 4, 5};
// or
int[] arr = new int[] {1, 2, 3, 4, 5};
```

Old syntax: 老语法（不推荐）

```
double myList[] = new double[10];
```

数组变量获取长度: arrayVariable.**length**

Java默认值

在Java中，不同类型的数据的默认值如下：

1. **整数类型**（如`int`, `long`, `byte`, `short`）：默认值是0。
2. **浮点类型**（如`float`, `double`）：默认值是0.0。
3. **布尔类型**（`boolean`）：默认值是`false`。
4. **字符类型**（`char`）：默认值是`\u0000`（即ASCII码中的NUL字符）。
5. **引用数据类型**（`reference`）：默认值是null；

搭配系统输入来创建数组

```
double[] myList = new double[10];
Scanner input = new Scanner(System.in);
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++) {
    myList[i] = input.nextDouble();
}
```

数组变化训练

整体向左移动，或者整体向右移动。

> 需要注意的是，开始的时候是闭区间，结束的时候是开区间

```
// shifting left
double temp = myList[0]; // Retain the first element
for (int i = 1; i < myList.length; i++) {
    myList[i - 1] = myList[i];
}
// Move the first element to fill in the last position
```

```
myList[myList.length - 1] = temp;

// Shifting right
double temp = myList[myList.length - 1]; // Retain the last element
for(int i=myList.length-1; i>0; i--) {
    myList[i] = myList[i-1];
}
// Move the last element to fill in the first position
myList[0] = temp;
```

随机洗牌

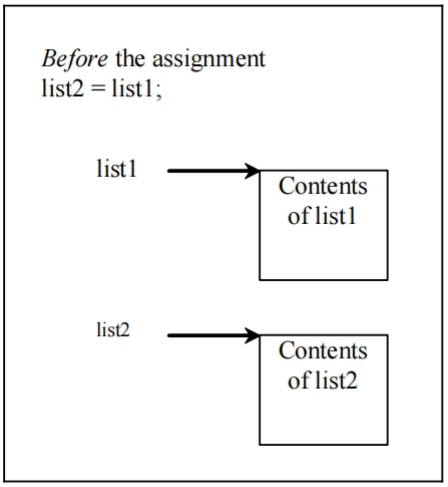
```
for (int i = 0; i < myList.length; i++) {
    // Generate an index j randomly
    // 注意random生成的数字是在0~1之间的随机小数，所以要乘上当前列表的长度,int会向下取整
    int j = (int)(Math.random() * myList.length);
    // Swap myList[i] with myList[j]
    double temp = myList[i];
    myList[i] = myList[j];
    myList[j] = temp;
}
```

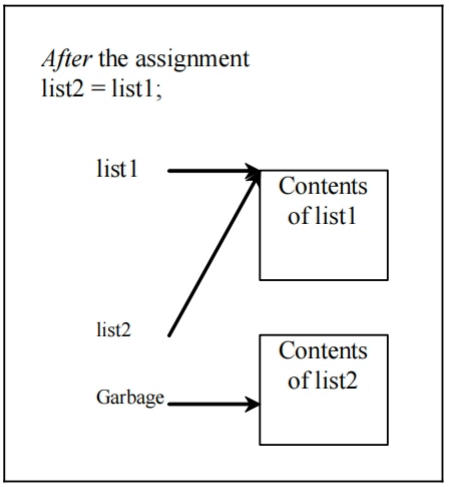
增强for循环

```
double[] myList = new double[10];
double total = 0;

// for循环
for (inti = 0; i < myList.length; i++) {
    total += myList[i];
}

// forEach循环 （需要搭配Java的集合类数据类型才能使用）
for (double d : myList) {
    total += d;
}
```

复制Arrays

在Java中不能简单地只使用=来复制一个对象的内容给另一个。使用=的含义是复制当前指向存储该对象的地址 (re-directing the pointer)。





正确的复制方式是创建一个新的数组对象，然后循环地将原对象中的值赋值进去

```
int[] sourceArray={2, 3, 1, 5, 10};
int[] targetArray=new int[sourceArray.length];
for (int i = 0; i < sourceArray.length; i++) {
    targetArray[i] = sourceArray[i];
}
```

Java复制工具 arraycopy utility

```
System.arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);
// Example:
int[] sourceArray={2, 3, 1, 5, 10};
int[] targetArray=new int[sourceArray.length];
System.arraycopy(sourceArray, 0,
targetArray, 0, sourceArray.length);
```

Pass By Value 传值或传引用

Passing arrays to Method 将数组传递给方法

```
// 假设需要传递一个array进入该方法
public static void printArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
}

// 两种传递方式，一种是传入一个声明好的数组变量
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
// 另一种是直接传入一个匿名数组
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

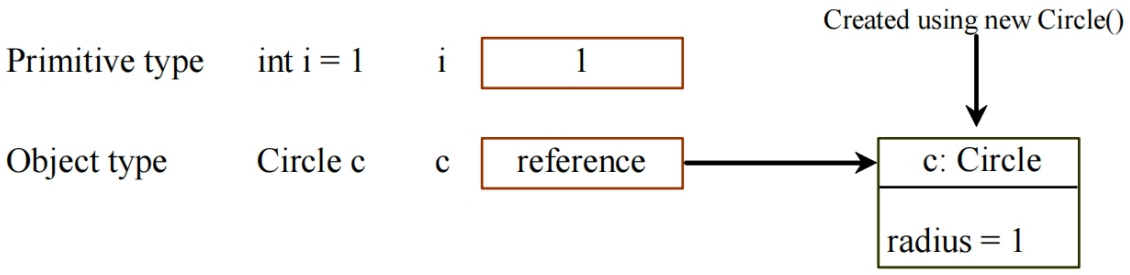
There is no explicit reference variable for the array. Such array is called an **anonymous array**.

数组没有显式引用变量。这样的数组称为 **匿名数组。**

Java uses **pass by value** to pass arguments to a method.

Java是**值传递**类型的语言。

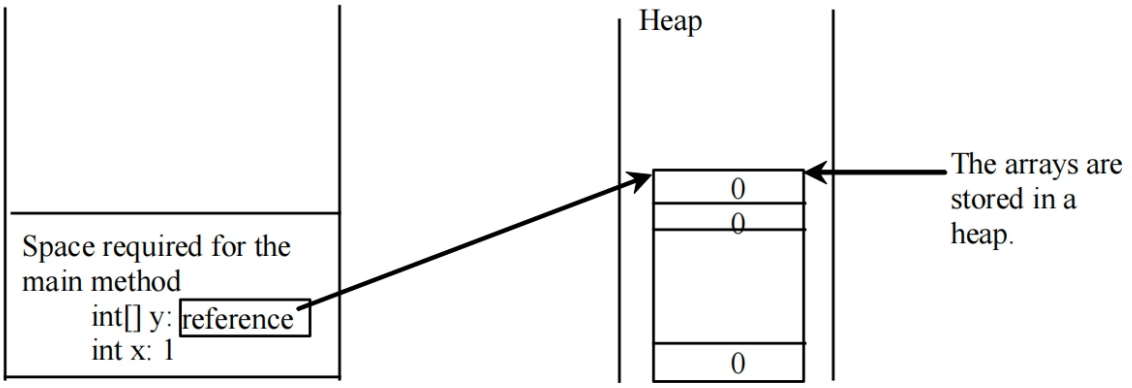
- For a parameter of an **array** reference type, the value of the parameter contains a **reference** to an array; this reference is passed to the method.
  - Any changes to the array that occur inside the method body **will affect the original array** that was passed as the argument.
- **传递引用数据类型（引用传递）**
  - **传递的是对象的引用**（内存地址），即将对象在内存中的地址传递给方法。
  - 在方法中通过引用修改对象的属性或调用方法会影响到原始对象。
- Different from a parameter of a **primitive type value** where the actual value is passed.
  - Changing the value of the local parameter inside the method **does not affect the value of the variable** outside the method
- **传递基础数据类型（值传递）**
  - **传递的是值的副本**，即将变量的值复制一份传递给方法。
  - 在方法中对参数值的修改不会影响原始变量的值。




```java
// 案例
public class Test {
    public static void main(String[] args) {
        int x = 1; // x represents an int value
        int[] y = new int[10]; // y represents an array of int values
        m(x, y); // Invoke m with arguments x and y
        System.out.println("x is " + x); // 1
        System.out.println("y[0] is " + y[0]); // 5555
    }

    public static void m(int number, int[] numbers) {
        number = 1001; // Assign a new value to number
        numbers[0] = 5555; // Assign a new value to numbers[0]
    }
}
```

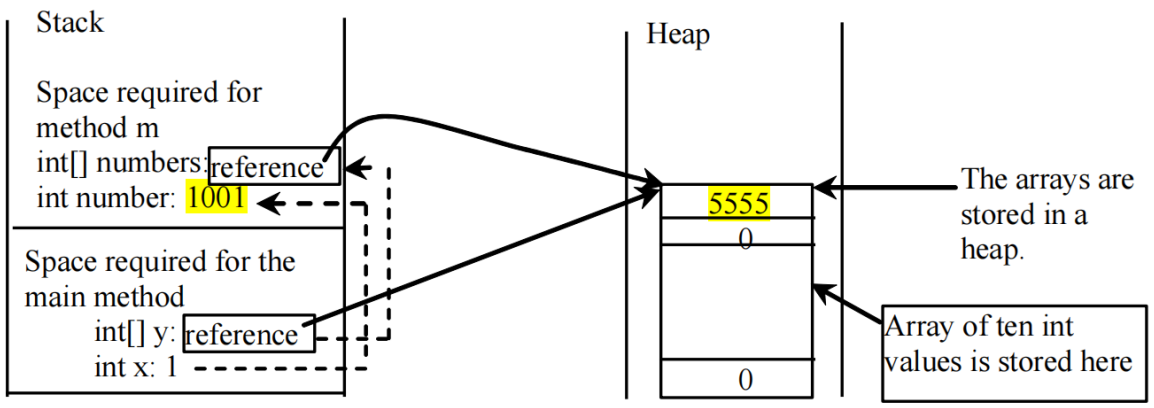
堆与栈

JVM 将数组和对象存储在称为 **堆（HEAP）** 的内存区域中，该区域用于动态内存分配，其中内存块以任意顺序分配和释放。



When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, numbers now contains the same reference value to the same array.

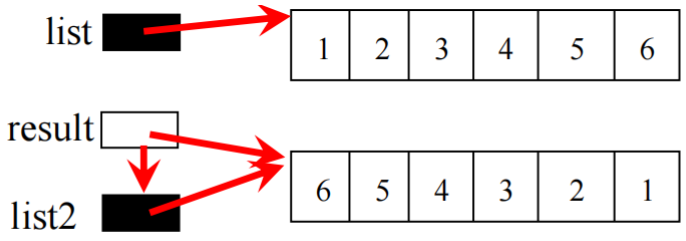
调用 m（x， y） 时，x 和 y 的值将传递给 number 和 numbers。由于 y 包含数组的引用值，因此 numbers 现在包含同一数组的相同引用值。



Reverse the list

> 方法1： 生成一个新的数组并且承接翻转之后的老数组

```
int[] list1 = new int[] {1, 2, 3, 4, 5, 6};
int[] list2 = new int[list1.length];

public int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list1.length; i++, j--) {
        result[j] = list[i];
    }

    return result;
}
```



> 方法2：直接在原数组上进行更改

```
public static void reverse(int[] list) {
    int temp;
    for (int i = 0, j = list.length - 1; i < list.length/2; i++, j--) {
        temp = list[j];
        list[j] = list[i];
        list[i] = temp;
    }
}
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
reverse(list1);
System.out.print(list1[0]);
```

Algorithm 数据相关算法

Binary Search 二分查找

1. 如果 key 小于 middle 元素，则只需搜索数组前半部分的 key。
2. 如果 key 等于 middle 元素，则搜索以匹配项结束。
3. 如果 key 大于中间元素，则只需要在数组的后半部分搜索 key。

```
/** Use binary search to find the key in the list */
public static int binarySearch(int[] list, int key) {
    int low = 0;
    int high = list.length - 1;
    int mid;
    // 这里使用的策略是左闭右闭的区间
    while (high >= low) {
        mid = (low + high) / 2;
        if (key < list[mid])
            high = mid - 1;
        else if (key == list[mid])
            return mid;
        else
            low = mid + 1;
    }
    return -1 - low;
}
```

Time complexity: O(logn)

Java内置方法:

> Arrays.binarySearch(list, targetNumber);

Selection Sort 选择排序

```
public static void selectionSort(double[] list) {
    for (int i = 0; i < list.length; i++) {
        // Find the minimum in the list[i..list.length-1]
        double currentMin = list[i];
        int currentMinIndex = i;
```

```
        for (int j = i + 1; j < list.length; j++) {
            if (currentMin > list[j]) {
                currentMin = list[j];
                currentMinIndex = j;
            }
        }
        // Swap list[i] with list[currentMinIndex] if necessary;
        if (currentMinIndex != i) {
            list[currentMinIndex] = list[i];
            list[i] = currentMin;
        }
    }
}
```

Time complexity $O(n^2)$

Insertion Sort 插入排序

```
public static void insertionSort(int[] a){
    for(int i=1; i<a.length; i++){
        int temp = a[i];
        if(temp < a[i-1]) {
            int j;
            for(j = i-1; j >= 0; j--){
                if(temp < a[j])
                    a[j+1] = a[j];
                else
                    break;
            }
            a[j+1] = temp;
        }
    }
}
```

Time complexity: $O(n) \sim O(n^2)$

Java内置的排序方法

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers);

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars);
```

Questions 练习题

1. What is the output of the following code:

```
int[] numbers = {1, 2, 3, 4, 5};
for (int num : numbers) {
    num *= 2;
}
System.out.println(numbers[2]);
```

输出的结果是: 3，因为并没有将计算之后的结果赋值给numbers[i]