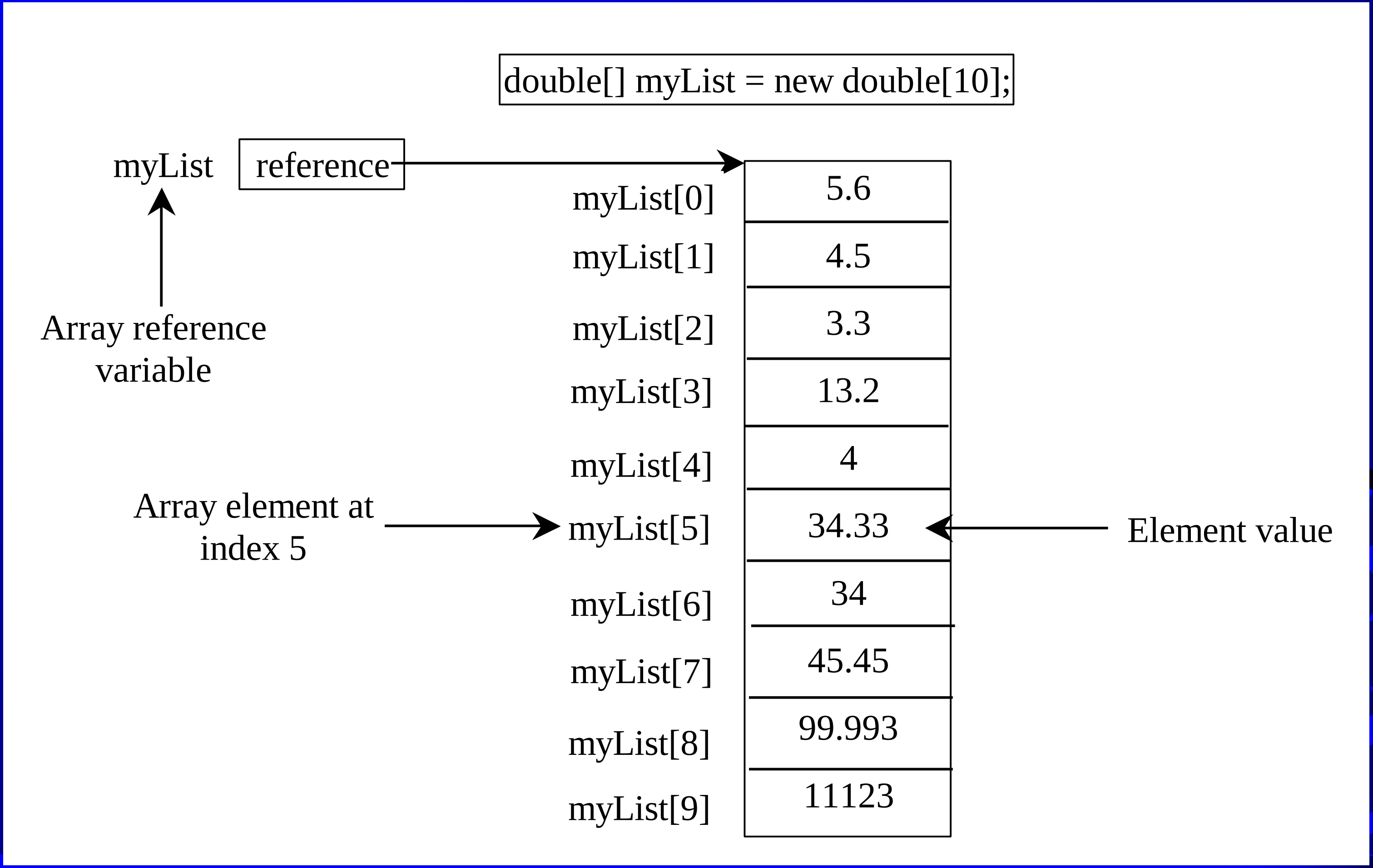
Chapter 7 Single-Dimensional Arrays

关于数组

数组是一组**相同类型**数据的集合。你不需要关心这些数据在内存中如何排列，因为Java不能通过地址访问元素。



声明数组变量

☞ `datatype[] arrayRefVar;` **//推荐写法**

例如：

`double[] myList;` //声明myList是数组变量，但是目前数组为**null**，还没有空间存放数组元素

☞ `datatype arrayRefVar[];` **//可用，不推荐**

例如：

```
double myList[];
```

创建数组

```
arrayRefVar = new datatype[arraySize];
```

例如：

`myList = new double[10];` //Java数组是动态创建的，new的作用是分配数组空间。此处**一定要用new**，这点与C语言不同

`myList[0]` 表示数组第一个元素

`myList[9]` 表示数组最后一个元素

声明的同时创建数组

- `datatype[] arrayRefVar = new datatype[arraySize];`

```
double[] myList = new double[10]; //推荐写法
```

- `datatype arrayRefVar[] = new datatype[arraySize];`

```
double myList[] = new double[10]; //不推荐
```

数组长度

数组一旦创建（主意不是声明）后，长度就固定不变了。其实数组是一个对象，这个对象有一个属性length存储了数组长度。用法如下：

arrayRefVar.**length**

例如前一个例子,

myList.length 是 10

默认值

和C不同，Java会自动为数组赋初值，所以创建数组（new）之后，每一个元素都有默认值。具体规定如下：

<u>0</u> 是所有数值类型的初值（整数、浮点数）

<u>‘\u0000’</u> 是 <u>char</u> 类型的初值

<u>false</u> 是 <u>boolean</u> 类型的初值

数组下标

数组元素通过下标访问，下标从0开始编号，最大下标是数组长度-1。例如有一个数组arrayRefVar，无论它类型如何，可用下标一定是：

0…arrayRefVar.length-1

也就是这个数组的元素是：

arrayRefVar[0], arrayRefVar[1], arrayRefVar[2], …, arrayRefVar[arrayRefVar.length-1]

数组初始化

☞ 声明，创建和初始化数组也可以一步完成（注意此处不用new）：

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

此时数组myList大小为4。

注意：采用数组初始化，不能出现new。

两种写法对比

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

上个例子的等价写法:

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

注意

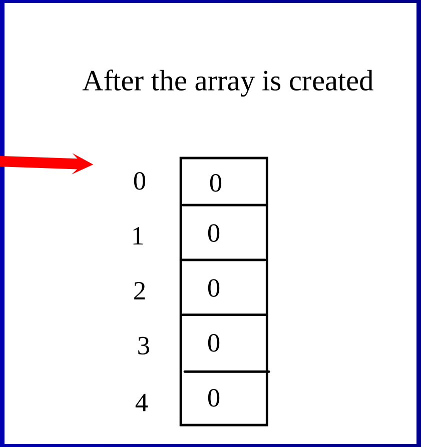
使用简化方式的初始化，所有代码必须在一条语句写完，不能拆开，所以下面这个写法就是错误的：

```
double[] myList;
myList = {1.9, 2.9, 3.4, 3.5};
```

单步执行一下

Declare array variable values, create an array, and assign its reference to values

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

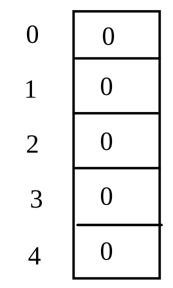


单步执行一下

i becomes 1

```
public class Test {
  public static voi[illegible]ain(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

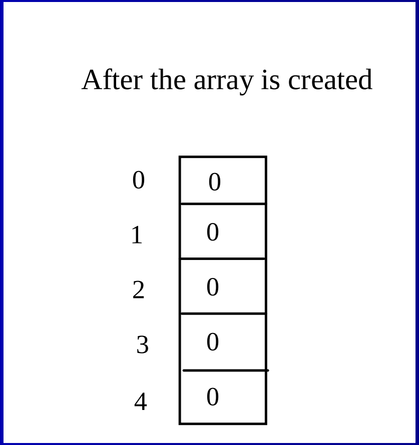
After the array is created



单步执行一下

i (=1) is less than 5

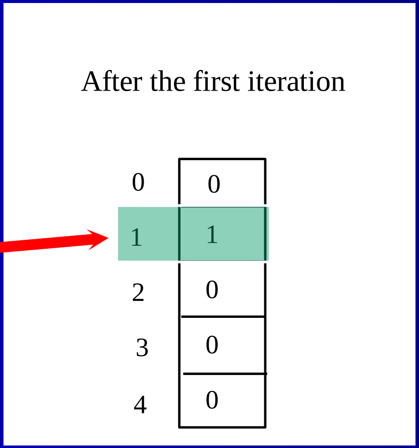
```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



单步执行一下

After this line is executed, value[1] is 1

```
public class Test {
  public static void main(Stri      gs) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



单步执行一下

After i++, i becomes 2

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

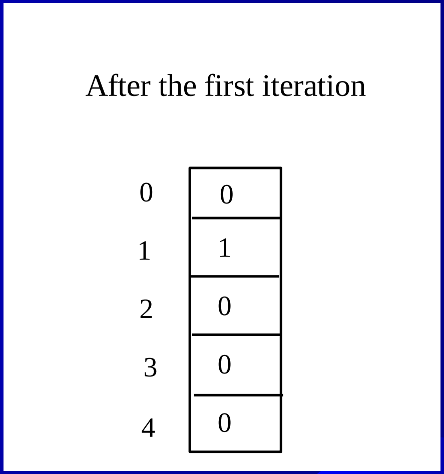
After the first iteration

0	0
1	1
2	0
3	0
4	0

单步执行一下

```
public class Test {
  public static void main(String[]
        args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] +
        values[4];
  }
}
```

i (= 2) is less than 5

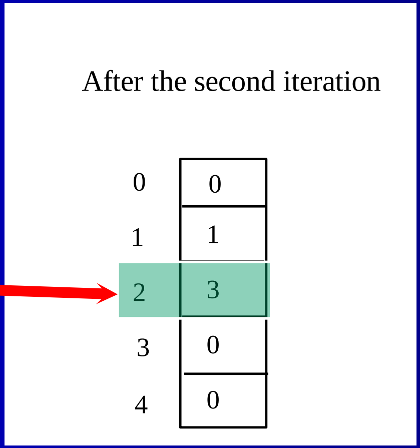

After the first iteration

0	0
1	1
2	0
3	0
4	0

单步执行一下

After this line is executed,
values[2] is 3 (2 + 1)

```
public class Test {
  public static void main(Stri        gs) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



单步执行一下

After this, i becomes 3.

```
public class Test {
  public static void main(S    ng[] args) {
    int[] values = new int[ ];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

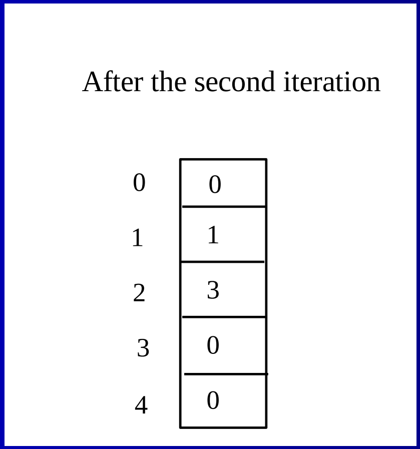
After the second iteration

0	0
1	1
2	3
3	0
4	0

单步执行一下

i (=3) is still less than 5.

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

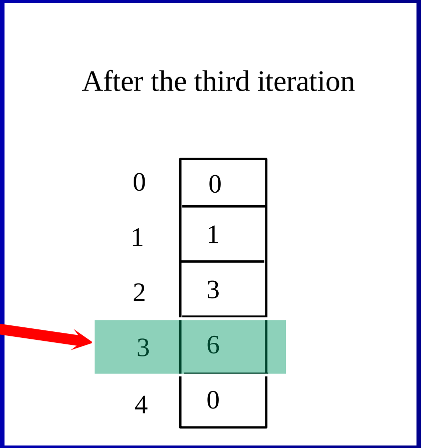

After the second iteration

0	0
1	1
2	3
3	0
4	0

单步执行一下

After this line, values[3] becomes 6 (3 + 3)

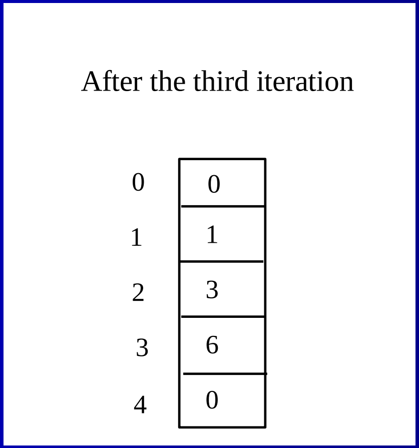
```
public class Test {
  public static void main(String[] arg
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



单步执行一下

After this, i becomes 4

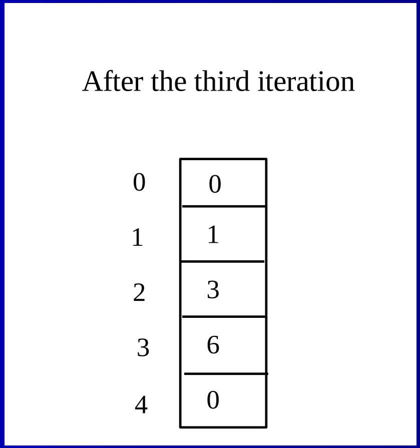
```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



单步执行一下

i (=4) is still less than 5

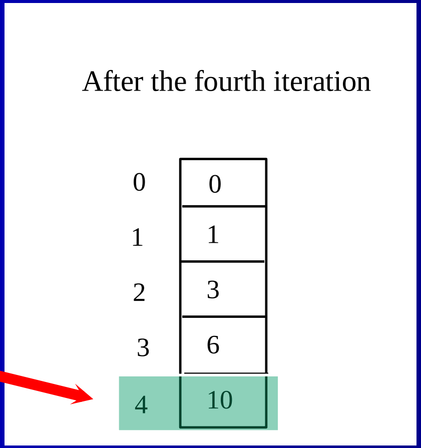
```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



单步执行一下

After this, values[4] becomes 10 (4 + 6)

```
public class Test {
  public static void main(Stri      args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

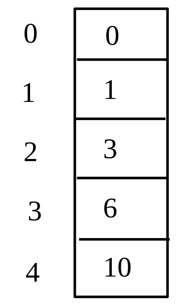


单步执行一下

After i++, i becomes 5

```
public class Test {
  public static void main(String[] arg
           {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

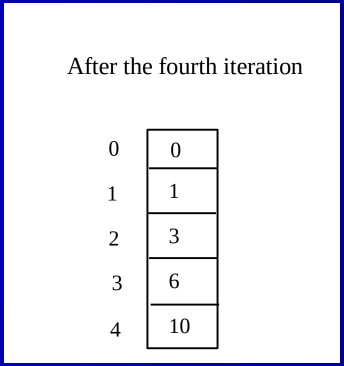
After the fourth iteration



单步执行一下

i (=5) < 5 is false. Exit the loop

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

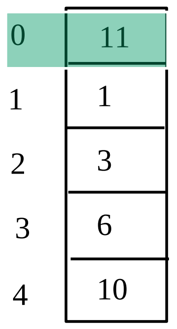

After the fourth iteration

0	0
1	1
2	3
3	6
4	10

单步执行一下

After this line, values[0] is 11 (1 + 10)

```
public class Test {
  public static void m        ring[] args) {
    int[] values = ne        [5];
    for (int i = 1; i       , i++) {
      values[i] = i     values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



数组例程

接下来有9个数组例程，欢迎模仿：

1. (用输入值初始化数组)
2. (用随机值初始化数组)
3. (打印数组)
4. (数组求和)
5. (找数组最大值)
6. (找数组最大元素的最小下标值)
7. (打乱数组)
8. (数组元素移动)
9. (用数组查表)

1. 用输入值初始化数组

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

2. 用随机值初始化数组

```
for (int i = 0; i < myList.length; i++) {
  myList[i] = Math.random() * 100;
}
```

3. 打印数组

```
for (int i = 0; i < myList.length; i++) {
  System.out.print(myList[i] + " ");
}
```

4. 数组求和

```
double total = 0;
for (int i = 0; i < myList.length; i++) {
  total += myList[i];
}
```

5. 找数组最大值

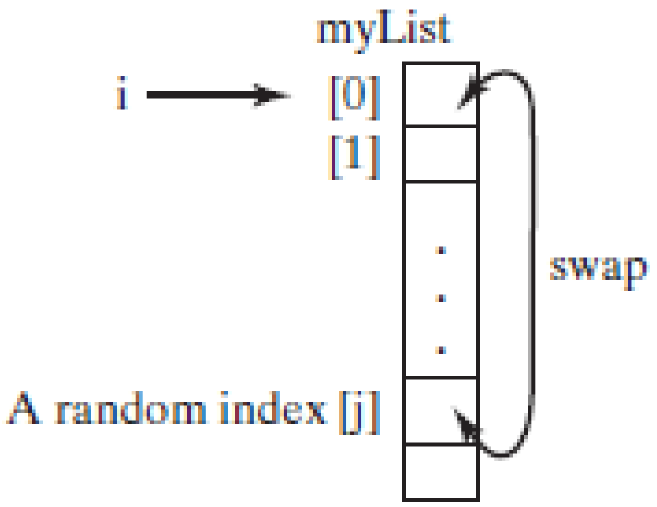
```
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
  if (myList[i] > max) max = myList[i];
}
```

6. 找数组最大元素的最小下标值

```
double max = myList[0];
int indexOfMax = 0;
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) {
        max = myList[i];
        indexOfMax = i;
    }
}
```

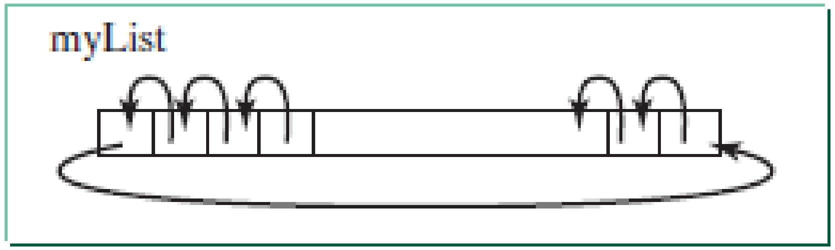
7. 打乱数组

```
for (int i = 0; i < myList.length; i++) {
    // Generate an index j randomly
    int j = (int) (Math.random() * mylist.length);
    // Swap myList[i] with myList[j]
    double temp = myList[i];
    myList[i] = myList[j]
    myList[j] = temp;
}
```



8. 元素移动（左移1位）

```
double temp = myList[0]; // Retain the first element
// Shift elements left
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;
```



9. 用数组查表

查找数字对应的英文月份名，如1对应January，2对应February，不使用数组，可能需要这样写代码：

```
if (monthNumber == 1)
  System.out.println("January");
else if (monthNumber == 2)
  System.out.println("February");
...
else
  System.out.println("December");
```

用switch也可实现上述代码。但两种写法都比较麻烦。

9. 用数组查表—续

查找数字对应的英文月份名，如1对应January，2对应February，使用数组，只需要这样写代码：

String[] months = {**"January"**, **"February"**, ..., **"December"**};

System.out.println(months[monthNumber - **1**]);

可以看到这个版本的代码更简洁高效。

增强型 for 循环 (for-each 循环)

JDK 1.5开始，for循环多了一种用法：从头到尾遍历。这种用法称为for-each循环。例如打印 myList数组的每一个元素值，可以这样写:

```
 for (double value: myList)
    System.out.println(value);
```

更一般的，for-each循环的语法是:

```
 for (elementType value: arrayRefVar) {
    // Process the value
 }
```

如果不想从头到尾遍历数组，或者需要跳过某几个下标，那只能老老实实用以前版本的for。

编程练习：数字覆盖问题

从键盘输入一批以0结尾的整数，编程判断这批数是否覆盖了1 到 99 之间的每一个整数。换句话说，就是判断1,2,3…,99的每一个整数是否都出现过。

解题思路

- 首先创建具有100个元素的**boolean**数组**isCovered**。用下标i表示数字i是否出现。下标0这里不用，因为只需要判断1..99。
- 初始的时候，所有元素的值都是false，表示数字未出现。每读到一个整数，就将对应的元素设为true。
- 所有数据读完后，判断数组元素是否全为true即可。

代码—1/2

```
import java.util.Scanner;
public class LottoNumbers {
public static void main(String[] args) {
Scanner input = new Scanner(System.in);
boolean[] isCovered = new boolean[100]; //默认false

// 读数并设置相应标志
int number = input.nextInt();
while (number != 0) {
  isCovered[number] = true;
  number = input.nextInt();
}
```

代码—2/2

```
// 检查是否全覆盖
boolean allCovered = true; // 假设全覆盖
for (int i = 1; i < isCovered.length; i++)
  if (!isCovered[i]) {
    allCovered = false; //找到1个未覆盖
    break;
  }
if (allCovered)
  System.out.println("Cover all numbers");
else
  System.out.println("Don't cover all numbers");
}
}
```

编程练习：发扑克牌

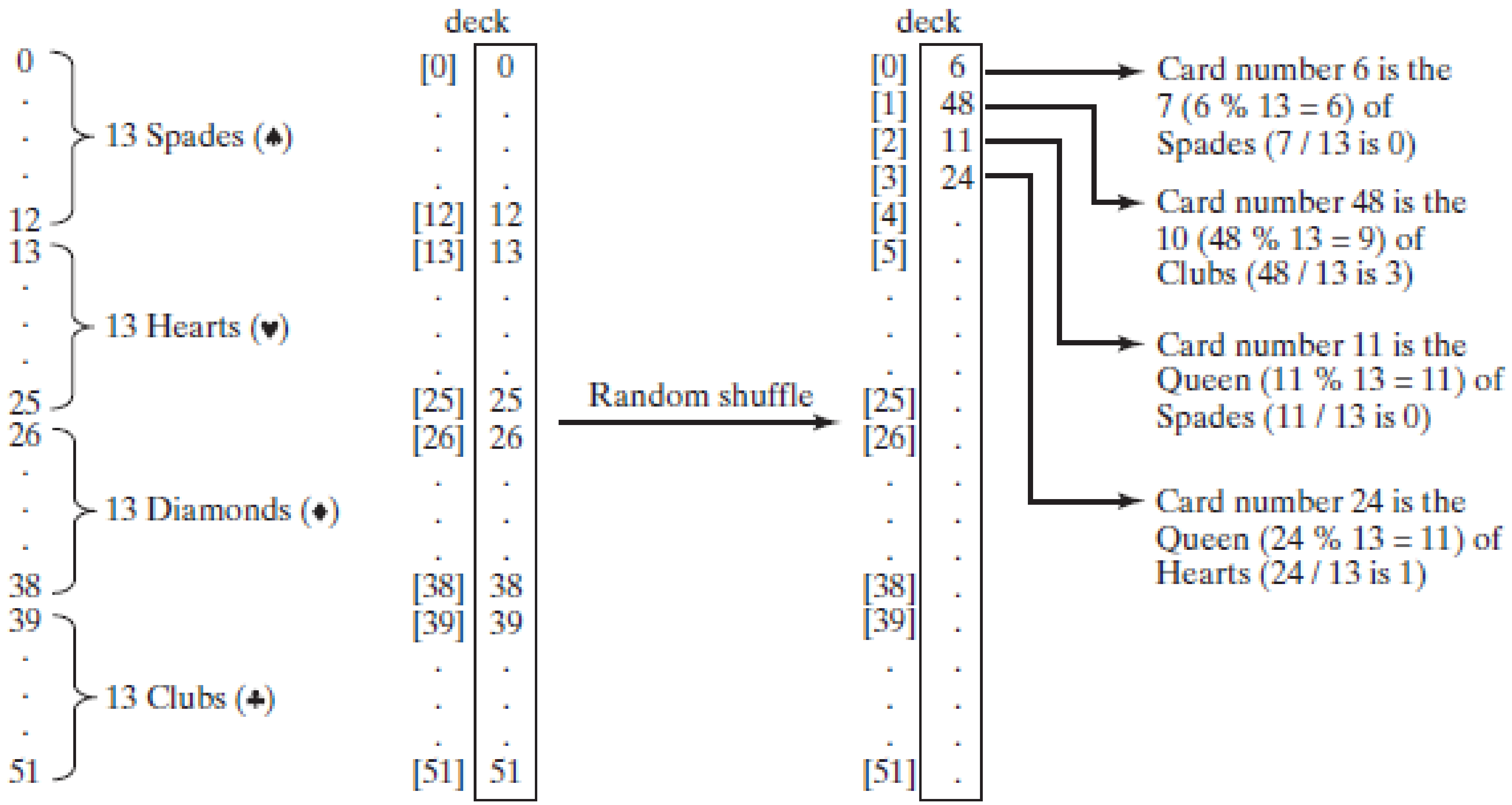
编程模拟发扑克牌。

解题思路：

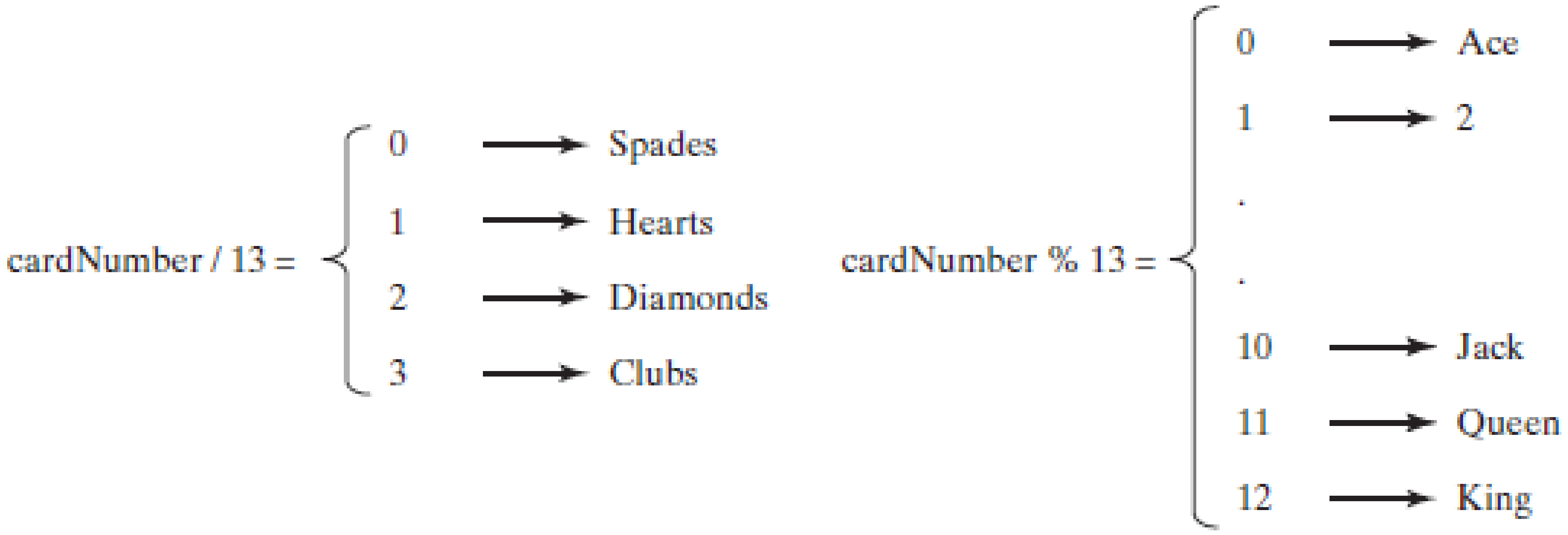
☞ 扑克牌共有52张（不含大小王）。首先用一个52个元素的数组表示，并且用0..51的值依次填充数组。约定0..12表示黑桃1..13，13..25表示红桃1..13，26..38表示方块1..13，39..51表示梅花1..13。

☞ 接下来，只需要打乱这个数组，然后根据元素值计算对应的花色，就可以完成发牌过程。

解题思路图示


0
12
13 Spades (♠)
13
25
13 Hearts (♥)
26
38
13 Diamonds (♦)
39
51
13 Clubs (♣)
deck
[0] 0
[12] 12
[13] 13
[25] 25
[26] 26
[38] 38
[39] 39
[51] 51
Random shuffle
deck
[0] 6
[1] 48
[2] 11
[3] 24
[4]
[5]
[25]
[26]
[38]
[39]
[51]
Card number 6 is the 7 (6 % 13 = 6) of Spades (7 / 13 is 0)
Card number 48 is the 10 (48 % 13 = 9) of Clubs (48 / 13 is 3)
Card number 11 is the Queen (11 % 13 = 11) of Spades (11 / 13 is 0)
Card number 24 is the Queen (24 % 13 = 11) of Hearts (24 / 13 is 1)

花色计算图示


cardNumber / 13 =
0 Spades
1 Hearts
2 Diamonds
3 Clubs
cardNumber % 13 =
0 Ace
1 2
.
.
10 Jack
11 Queen
12 King

代码—1/3

```java
public class DeckOfCards {
public static void main(String[] args) {
int[] deck = new int[52];
String[] suits = { "Spades", "Hearts",
"Diamonds", "Clubs" };
String[] ranks = { "Ace", "2", "3", "4",
"5", "6", "7", "8", "9", "10",
"Jack", "Queen", "King" };
```

代码—2/3

```
// Initialize the cards
for (int i = 0; i < deck.length; i++)
  deck[i] = i;
// Shuffle the cards
for (int i = 0; i < deck.length; i++) {
// Generate an index randomly
int index = (int) (Math.random() * deck.length);
int temp = deck[i];
deck[i] = deck[index];
deck[index] = temp;
}
```

代码—3/3

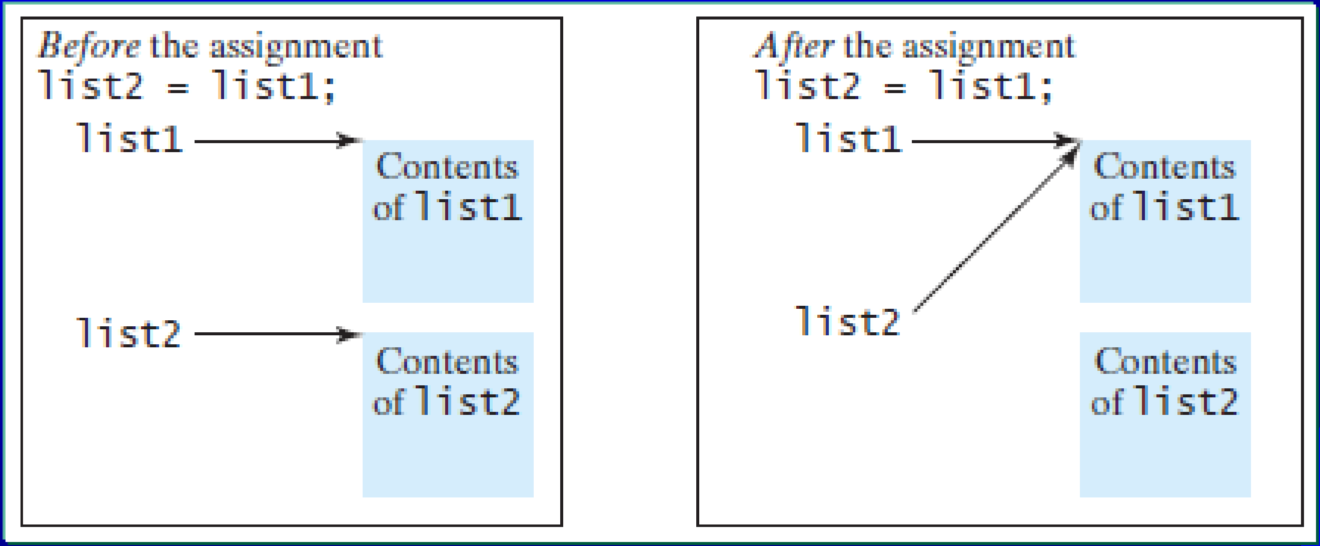
```
// Display the first four cards
for (int i = 0; i < 4; i++) {
String suit = suits[deck[i] / 13];
String rank = ranks[deck[i] % 13];
System.out.println("Card number " + deck[i]
+ ": " + rank + " of " + suit);
}
}
}
```

错版的数组拷贝

程序中常常需要把一个数组list1的内容复制给list2，这时候你可能会写出这样的代码：

list2 = list1;

这个代码不会有编译错，因为它的运行效果是这样：



解释一下：上述代码的意思是list2也指向list1数组，后果是list2数组丢失，而list1数组却有两个名字。

正确的数组拷贝

要把一个数组复制给另一个数组，需要老老实实用一个循环：

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new
  int[sourceArray.length];


for (int i = 0; i < sourceArrays.length; i++)
    targetArray[i] = sourceArray[i];
```

arraycopy 工具

实在想偷懒又想复制数组，可以借助 System.arraycopy方法：

```
arraycopy(sourceArray, src_pos,
  targetArray, tar_pos, length);
```

Example:

```
System.arraycopy(sourceArray, 0,
  targetArray, 0, sourceArray.length);
```

数组作为形参，可以有两种方式传递参数

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

有名数组作为实参

```
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

匿名数组作为实参

匿名数组

使用下列语法可以创建一个数组：

```
new dataType[]{literal0, literal1, ..., literalk};
```

例如这个语句：

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

由于这种数组并没有指定一个明确的变量名，所以这种数组叫做匿名数组。

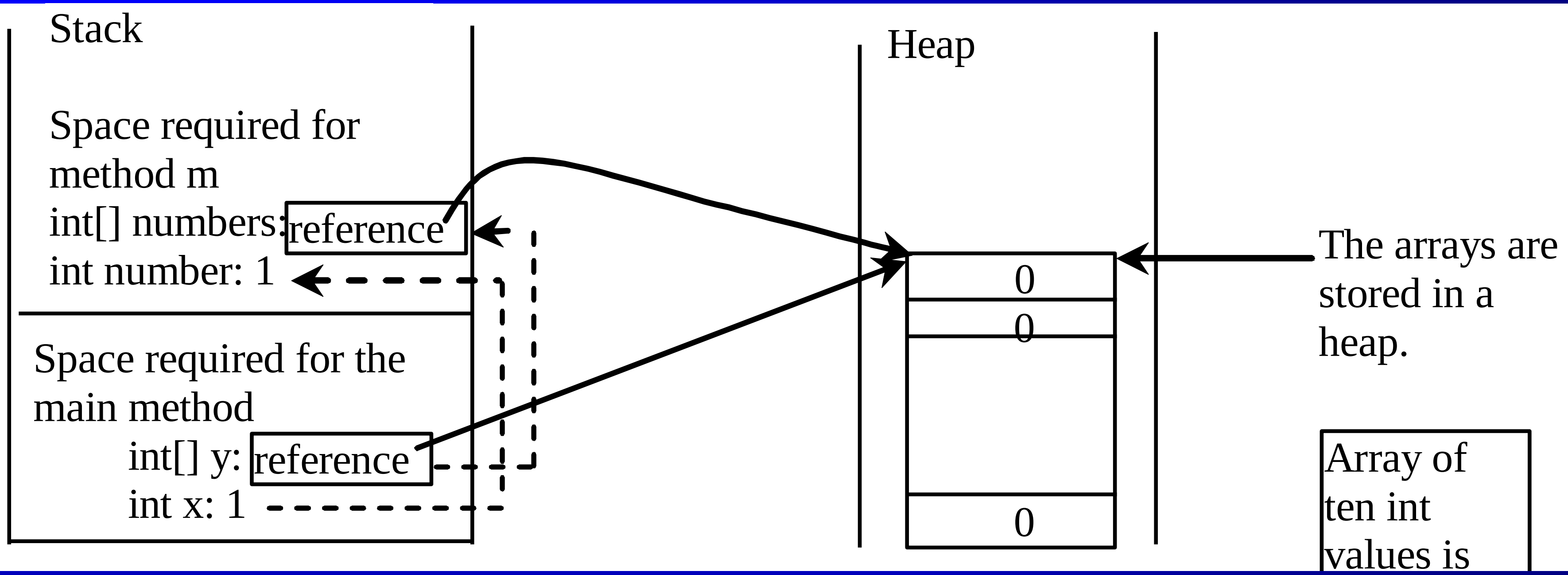
传值调用

- Java中实参与形参之间的数据传递只有一种方式：传值。
- 对**基本数据类型**来说，值传递就是把实参的值复制给形参，然后二者就再无关联了。
- 对**非基本数据类型**（例如数组、类、接口）来说，值传递也是把实参的值复制给形参，不过此时Java传递的实参值是一个引用（reference），这样造成的结果，就是实参和形参都指向同一个对象。**所以针对形参的任何修改，都相当于直接改动实参。**

一个简单的例子

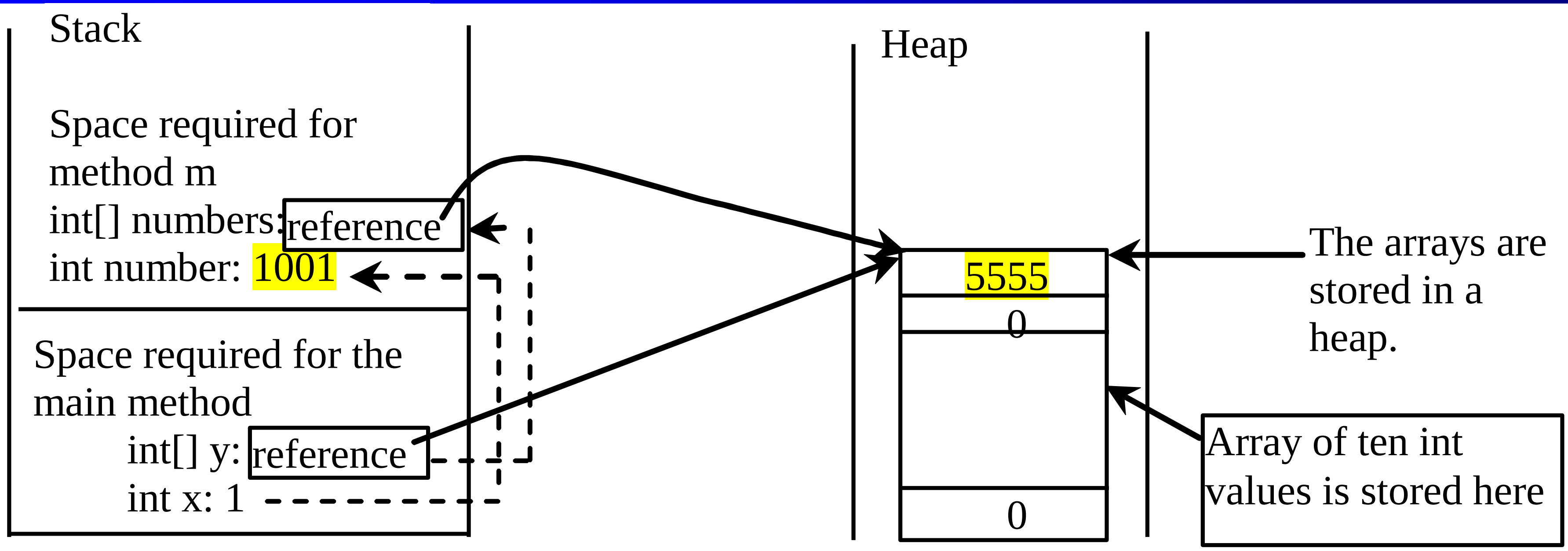
```
public class Test {
  public static void main(String[] args) {
    int x = 1; // x represents an int value
    int[] y = new int[10]; // y represents an array of int values

    m(x, y); // Invoke m with arguments x and y

    System.out.println("x is " + x);
    System.out.println("y[0] is " + y[0]);
  }

  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]
  }
}
```

调用栈


Stack
Space required for method m
int[] numbers: reference
int number: 1
Space required for the main method
int[] y: reference
int x: 1
Heap
0
0
0
The arrays are stored in a heap.
Array of ten int values is

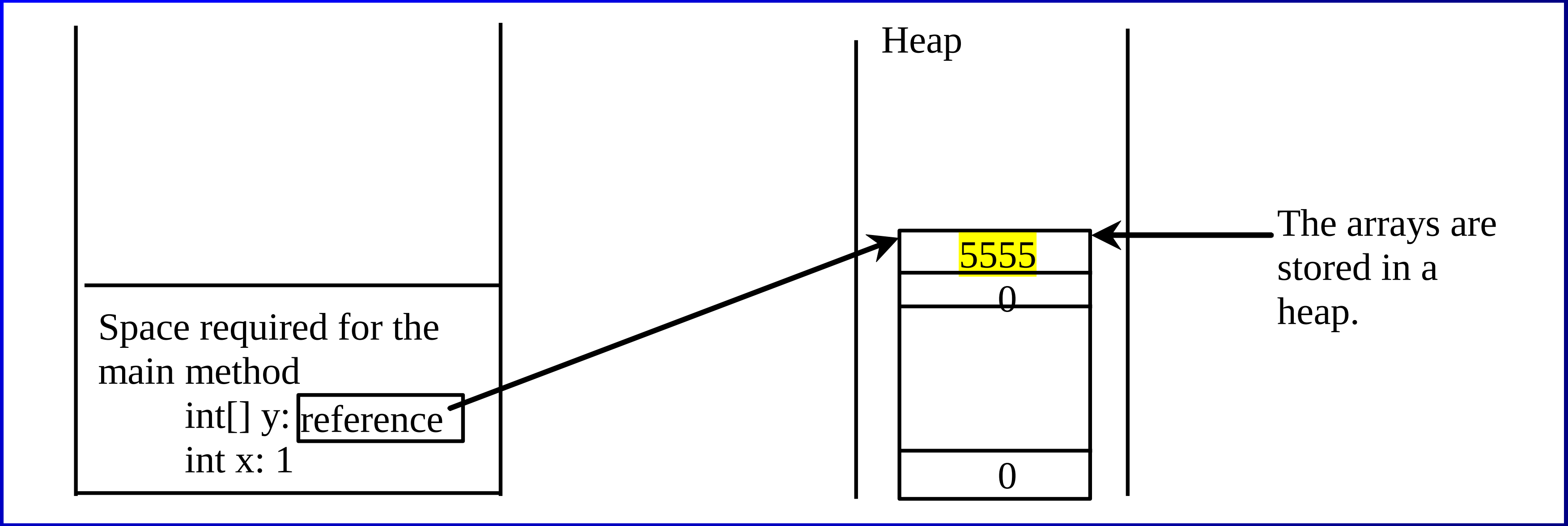
执行方法调用m(x, y)时, x 和 y 的值分别复制给 number 和 numbers。 由于 y 是数组， 复制的是引用值， 因此 numbers 和 y 现在都指向了同一个数组。

调用栈


Stack
Space required for method m
int[] numbers: reference
int number: 1001
Space required for the main method
int[] y: reference
int x: 1
Heap
5555
0
0
The arrays are stored in a heap.
Array of ten int values is stored here

当修改形参numbers的某个元素值时， 由于numbers和y都指向同一个数组， 所以相当于y数组的元素值也被同样修改了。

堆（Heap）

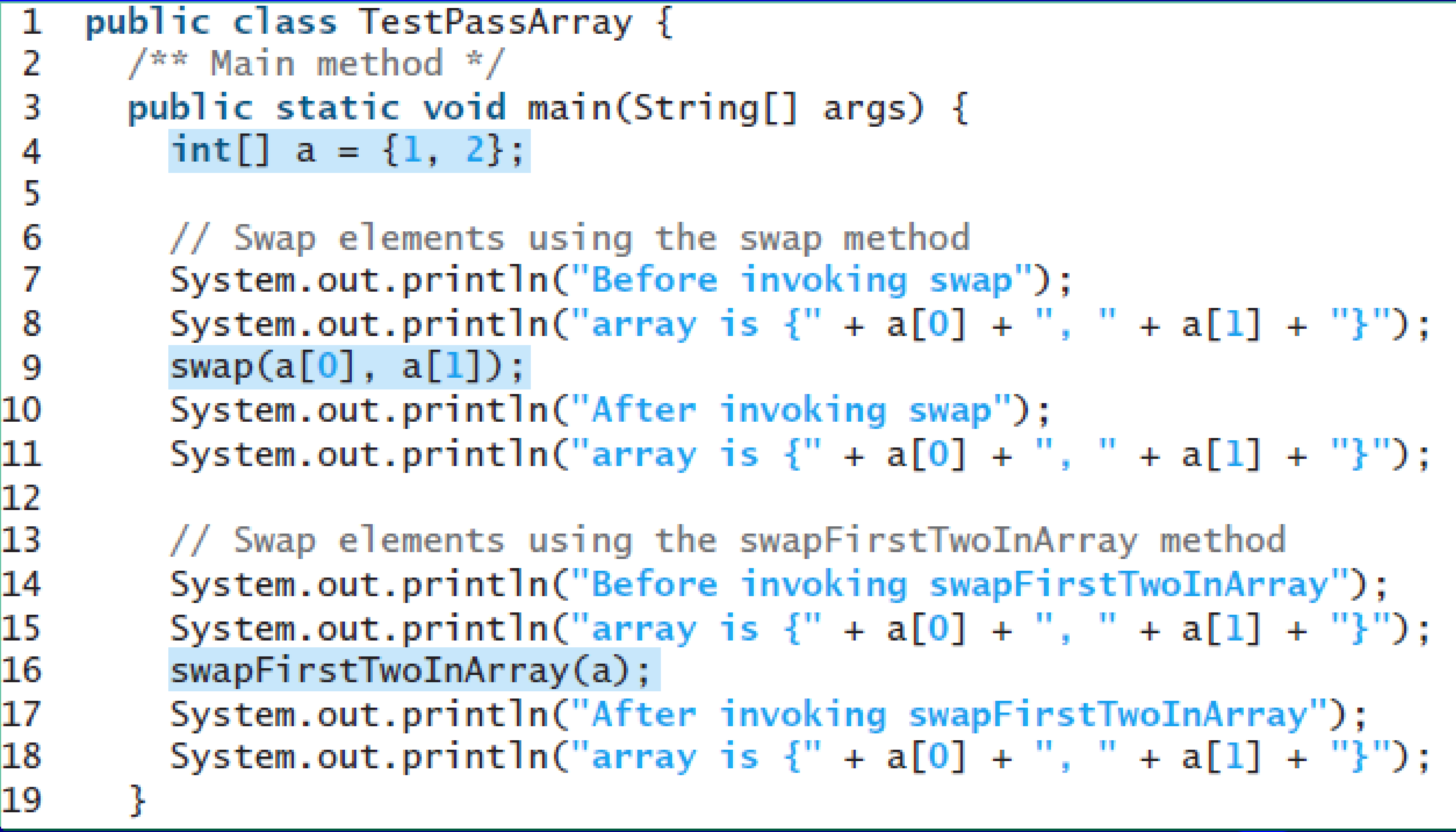

Space required for the main method
int[] y: reference
int x: 1
Heap
5555
0
0
The arrays are stored in a heap.

JVM会把数组存储在一个称为堆（ *heap* ）的内存区。这个区域会在JVM的控制下按需自动分配内存，以及在适当的时候自动回收无用内存（未必按照分配的顺序回收）。

参数传递的另一个例子

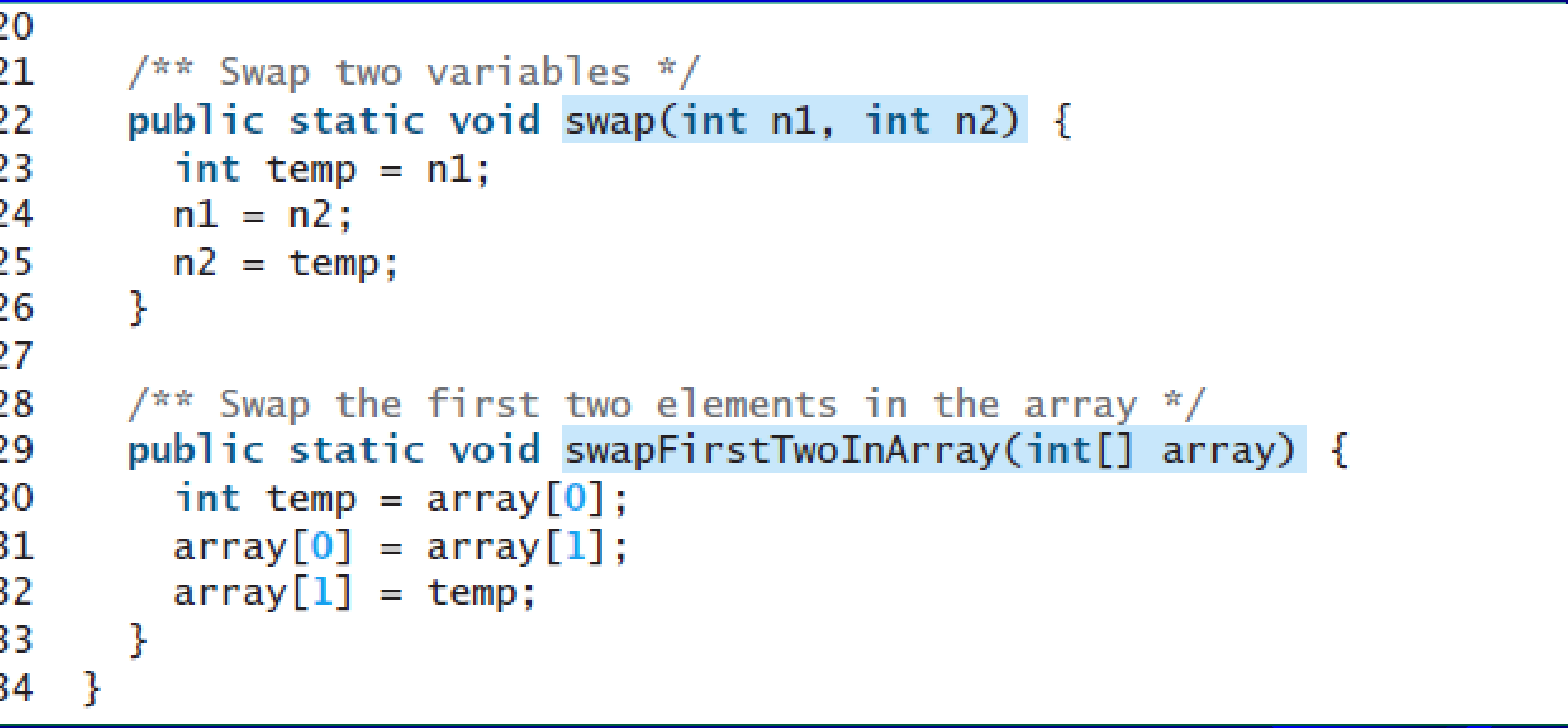
继续巩固一下基本数据类型和非基本数据类型（这里是数组）在作为参数传递时的区别。

前一半代码



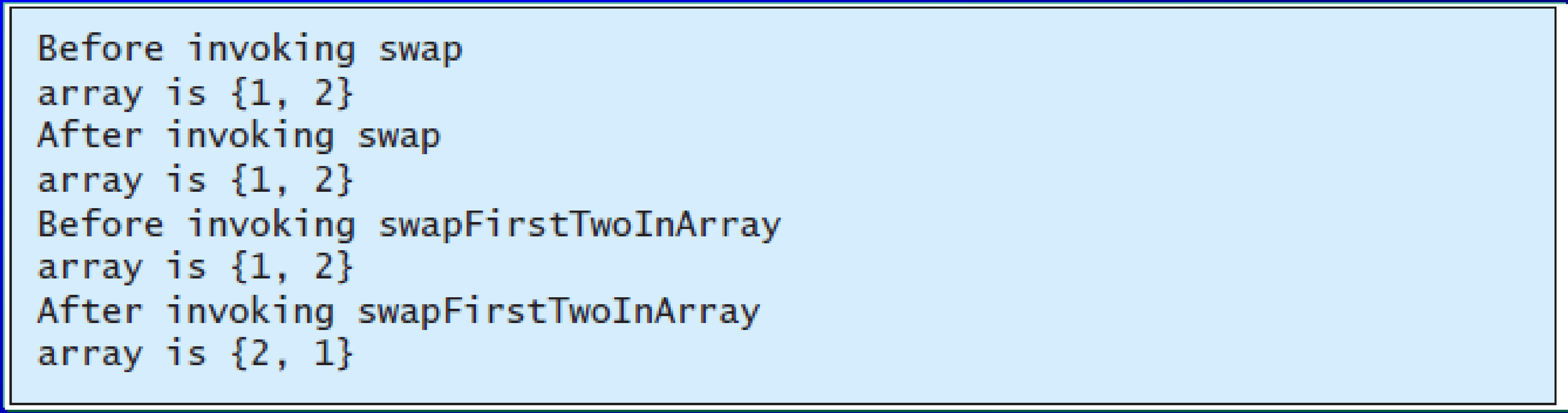
```java
public class TestPassArray {
  /** Main method */
  public static void main(String[] args) {
    int[] a = {1, 2};

    // Swap elements using the swap method
    System.out.println("Before invoking swap");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
    swap(a[0], a[1]);
    System.out.println("After invoking swap");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");

    // Swap elements using the swapFirstTwoInArray method
    System.out.println("Before invoking swapFirstTwoInArray");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
    swapFirstTwoInArray(a);
    System.out.println("After invoking swapFirstTwoInArray");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
  }
```

后一半代码



```

  /** Swap two variables */
  public static void swap(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
  }

  /** Swap the first two elements in the array */
  public static void swapFirstTwoInArray(int[] array) {
    int temp = array[0];
    array[0] = array[1];
    array[1] = temp;
  }
}
```

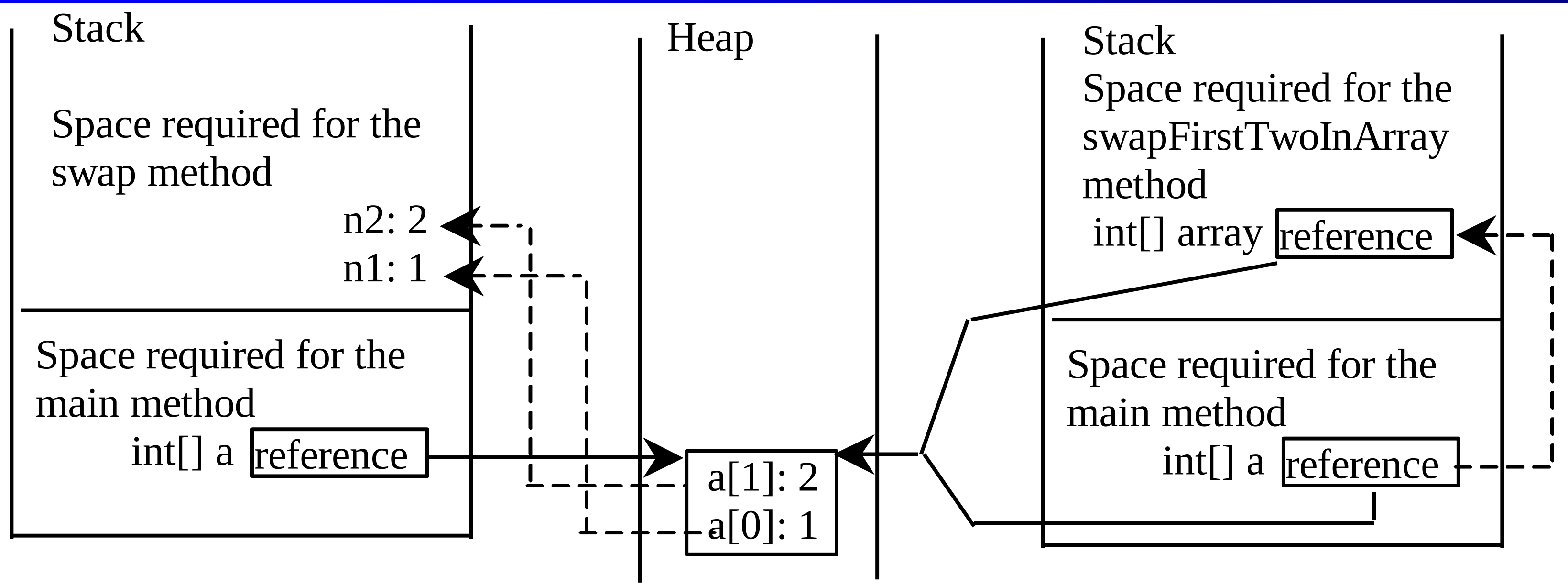
运行结果



```
Before invoking swap
array is {1, 2}
After invoking swap
array is {1, 2}
Before invoking swapFirstTwoInArray
array is {1, 2}
After invoking swapFirstTwoInArray
array is {2, 1}
```

- 结论就是，两个int作为参数的swap没有完成实参的值的交换，数组作为参数的swapFirstTwoInArray，成功改变了实参的值。

解释一下



Stack
Space required for the swap method
n2: 2
n1: 1
Space required for the main method
int[] a
reference
Heap
a[1]: 2
a[0]: 1
Stack
Space required for the swapFirstTwoInArray method
int[] array
reference
Space required for the main method
int[] a
reference

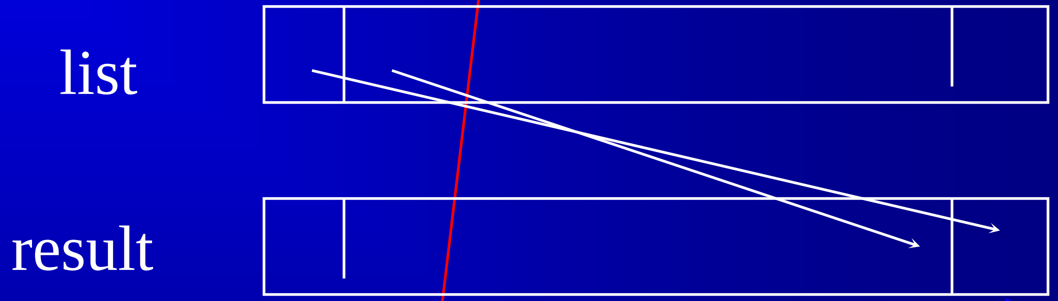
Invoke swap(int n1, int n2). The primitive type values in a[0] and a[1] are passed to the swap method.

The arrays are stored in a heap.

Invoke swapFirstTwoInArray(int[] array). The reference value in a is passed to the swapFirstTwoInArray method.

返回值是数组的方法

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

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

  return result;
}
```



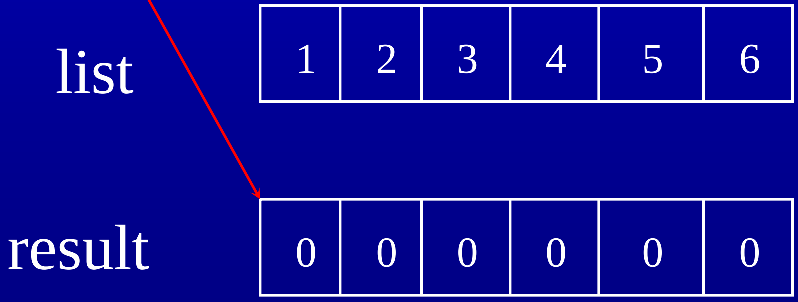
```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

单步执行一下数组翻转程序

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

声明并创建一个等大的数组

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

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

  return result;
}
```

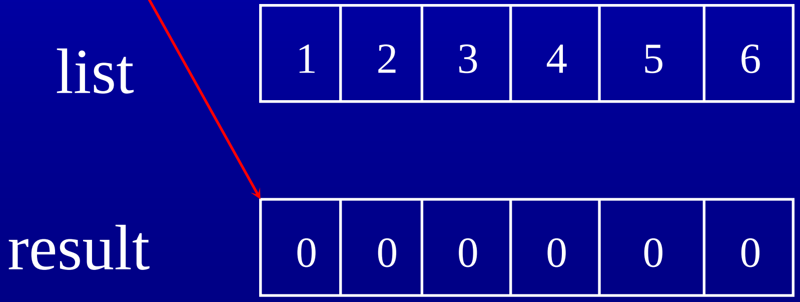


单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i = 0 and j = 5

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

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

  return result;
}
```

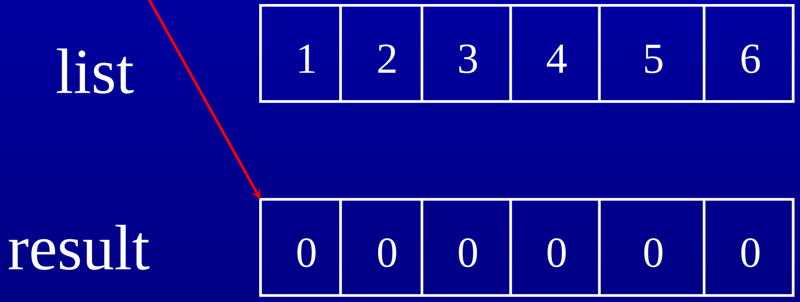


单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (= 0) is less than 6

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.lengt

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

  return result;
}
```

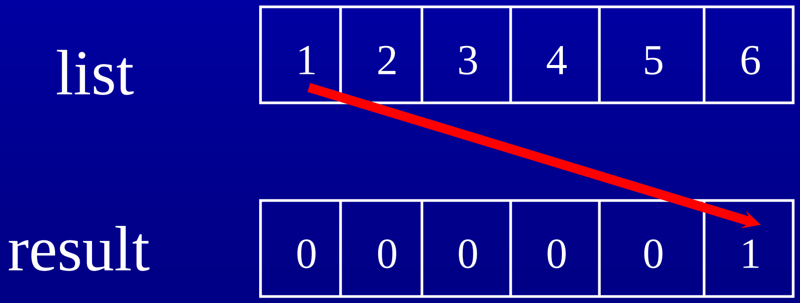


单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

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

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

  return result;
}
```

i = 0 and j = 5
Assign list[0] to result[5]



单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

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

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

  return result;
}
```

After this, i becomes 1 and j becomes 4

list

1	2	3	4	5	6

result

0	0	0	0	0	1

单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

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

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

  return result;
}
```

i (=1) is less than 6

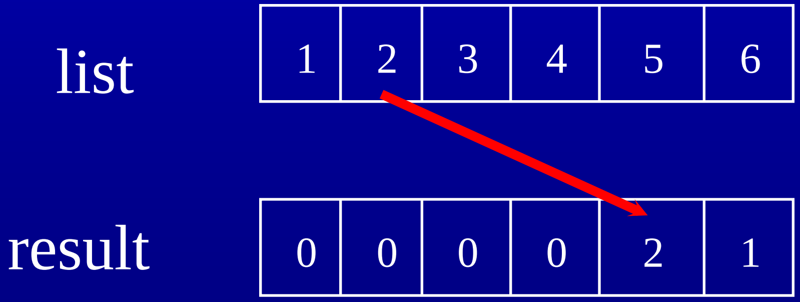
list	1	2	3	4	5	6
result	0	0	0	0	0	1

单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

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

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

  return result;
}
```

i = 1 and j = 4
Assign list[1] to result[4]



单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 2 and j becomes 3

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

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

  return result;
}
```

list	1	2	3	4	5	6
result	0	0	0	0	2	1

单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=2) is still less than 6

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

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

  return result;
}
```

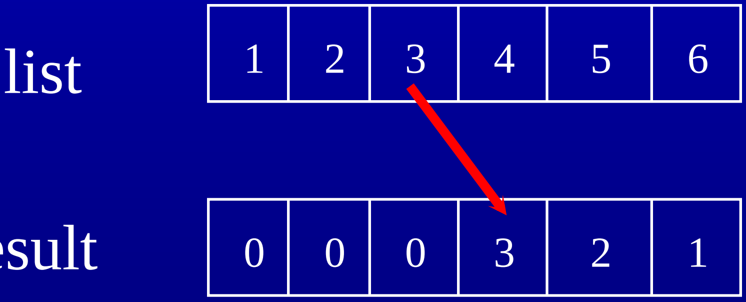
list	1	2	3	4	5	6
result	0	0	0	0	2	1

单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

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

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

  return result;
}
```

i = 2 and j = 3
Assign list[i] to result[j]

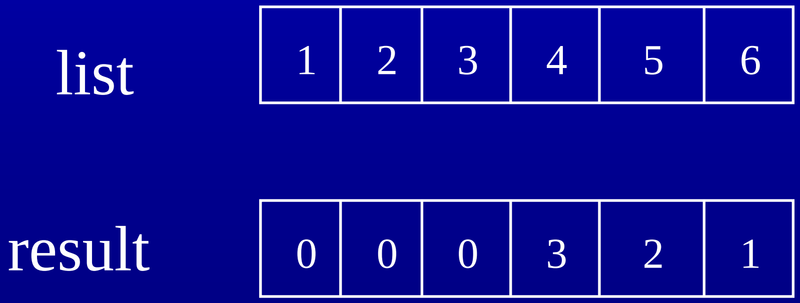


单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 3 and j becomes 2

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

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

  return result;
}
```



单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=3) is still less than 6

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

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

  return result;
}
```

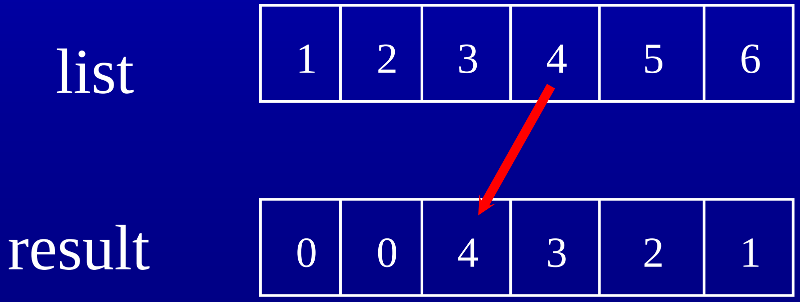
list	1	2	3	4	5	6
result	0	0	0	3	2	1

单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

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

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

  return result;
}
```

i = 3 and j = 2
Assign list[i] to result[j]



单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

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

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

  return result;
}
```

After this, i becomes 4 and j becomes 1

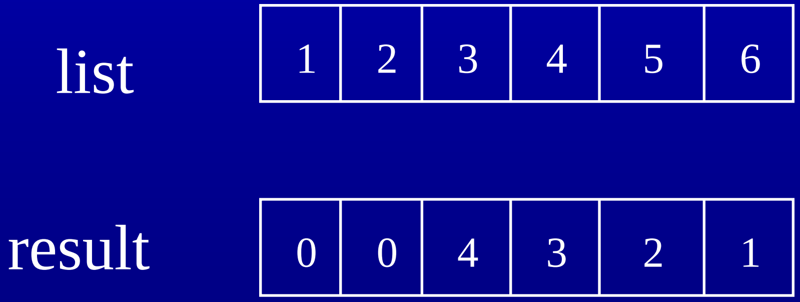
list	1	2	3	4	5	6
result	0	0	4	3	2	1

单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=4) is still less than 6

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

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

  return result;
}
```

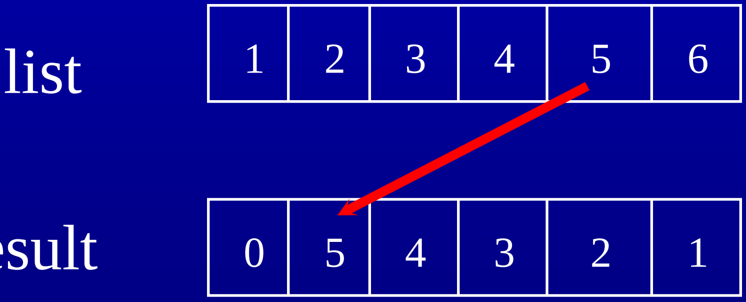


单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

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

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

  return result;
}
```

i = 4 and j = 1
Assign list[i] to result[j]



单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 5 and j becomes 0

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

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

  return result;
}
```

list	1	2	3	4	5	6
result	0	5	4	3	2	1

单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=5) is still less than 6

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length
  
  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }
  
  return result;
}
```

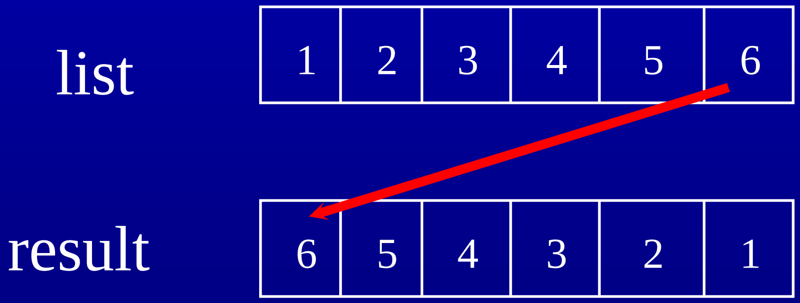
list	1	2	3	4	5	6
result	0	5	4	3	2	1

单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

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

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

  return result;
}
```

i = 5 and j = 0
Assign list[i] to result[j]

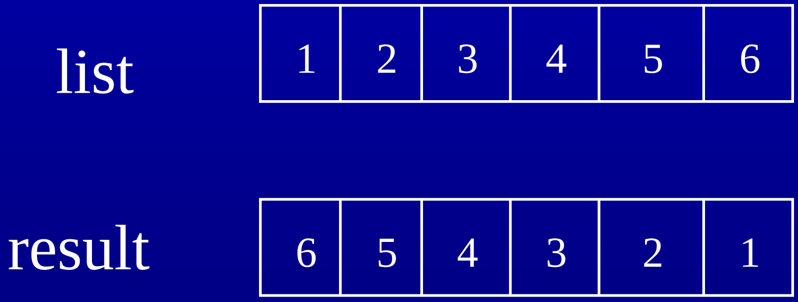


单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 6 and j becomes -1

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

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

  return result;
}
```



单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=6) < 6 is false. So exit the loop.

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

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

  return result;
}
```

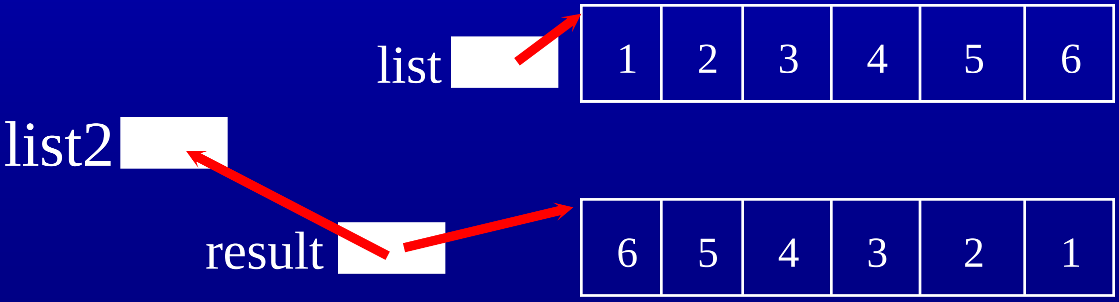
list	1	2	3	4	5	6
result	6	5	4	3	2	1

单步执行一下数组翻转程序

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Return result

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

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

  return result;
}
```



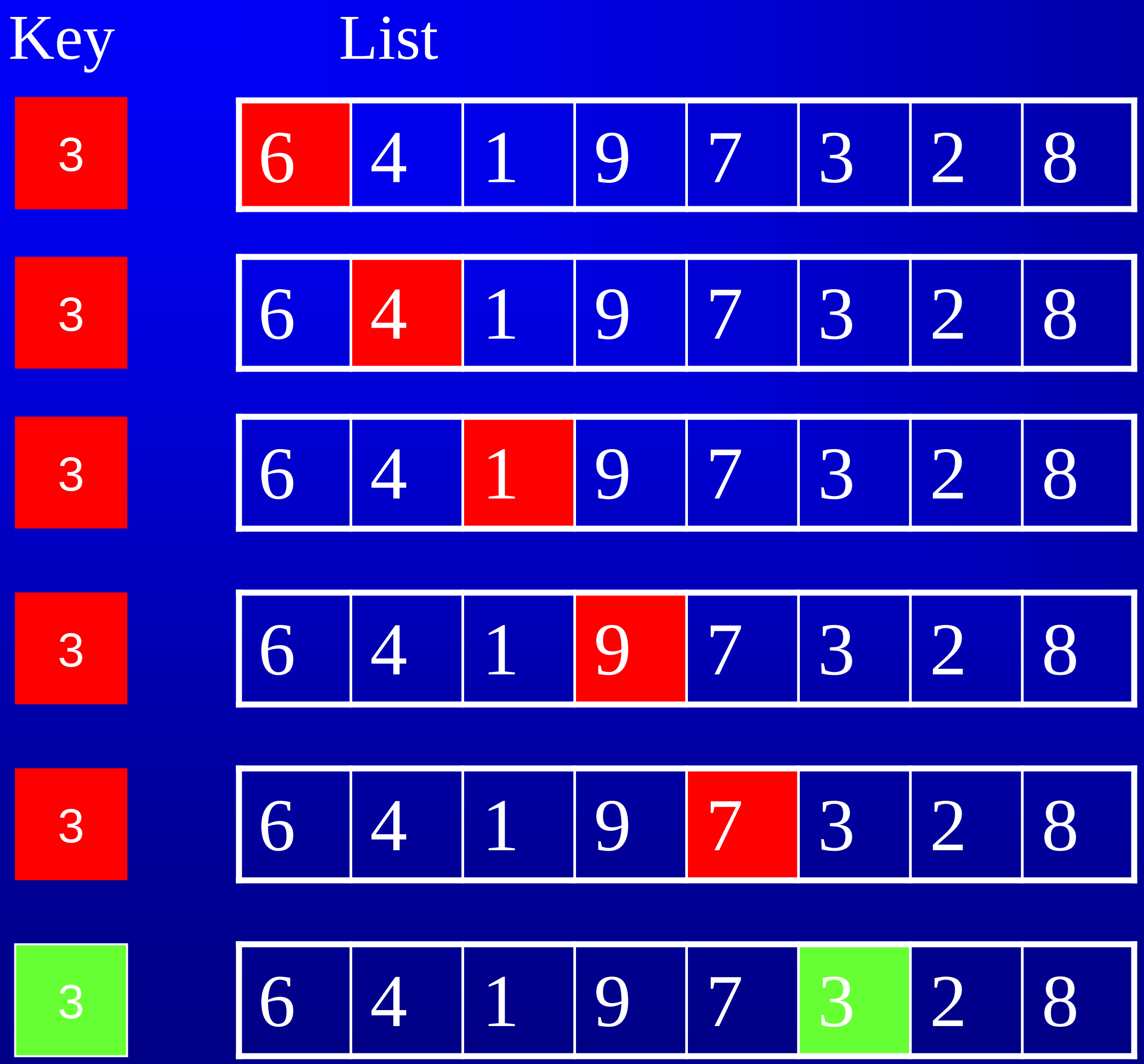
数组查找

- 数组查找是在数组中查找一个特定的数是否存在，如果存在的话，返回所在下标；如果不存在，返回一个负数（一般是-1，也可以是其它负数，只要不是有效下标就行）
- 查找有很多方法，这里只讨论其中两个，线性查找 *linear search* 和二分查找 *binary search*.

线性查找（Linear Search）

- 线性查找是从下标0开始，挨个向后查找指定关键字是否出现。一旦出现后立即退出查找，直接返回下标值；实在找不到的话，返回-1。
- 线性查找效率较低，编码简单，适用于查找无序的数组。

线性查找示意


Key
List
3 | 6 4 1 9 7 3 2 8
3 | 6 4 1 9 7 3 2 8
3 | 6 4 1 9 7 3 2 8
3 | 6 4 1 9 7 3 2 8
3 | 6 4 1 9 7 3 2 8
3 | 6 4 1 9 7 3 2 8

实现代码

```
/** The method for finding a key in the list */
public static int linearSearch(int[] list, int key) {
  for (int i = 0; i < list.length; i++)
    if (key == list[i])
      return i;
  return -1;
}
```

运行示例:

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4);  // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
```

二分查找（Binary Search）

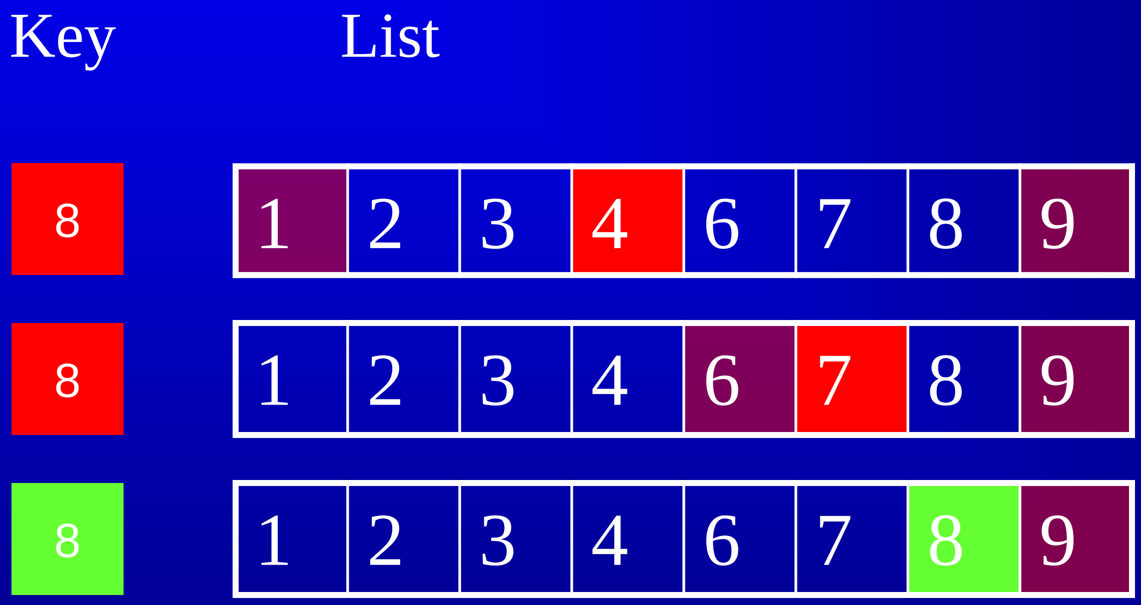
如果被查找的数组已经按照升序排列，二分查找是一个更好的选择。例如假设数组是：

2 4 7 10 11 45 50 59 60 66 69 70 79

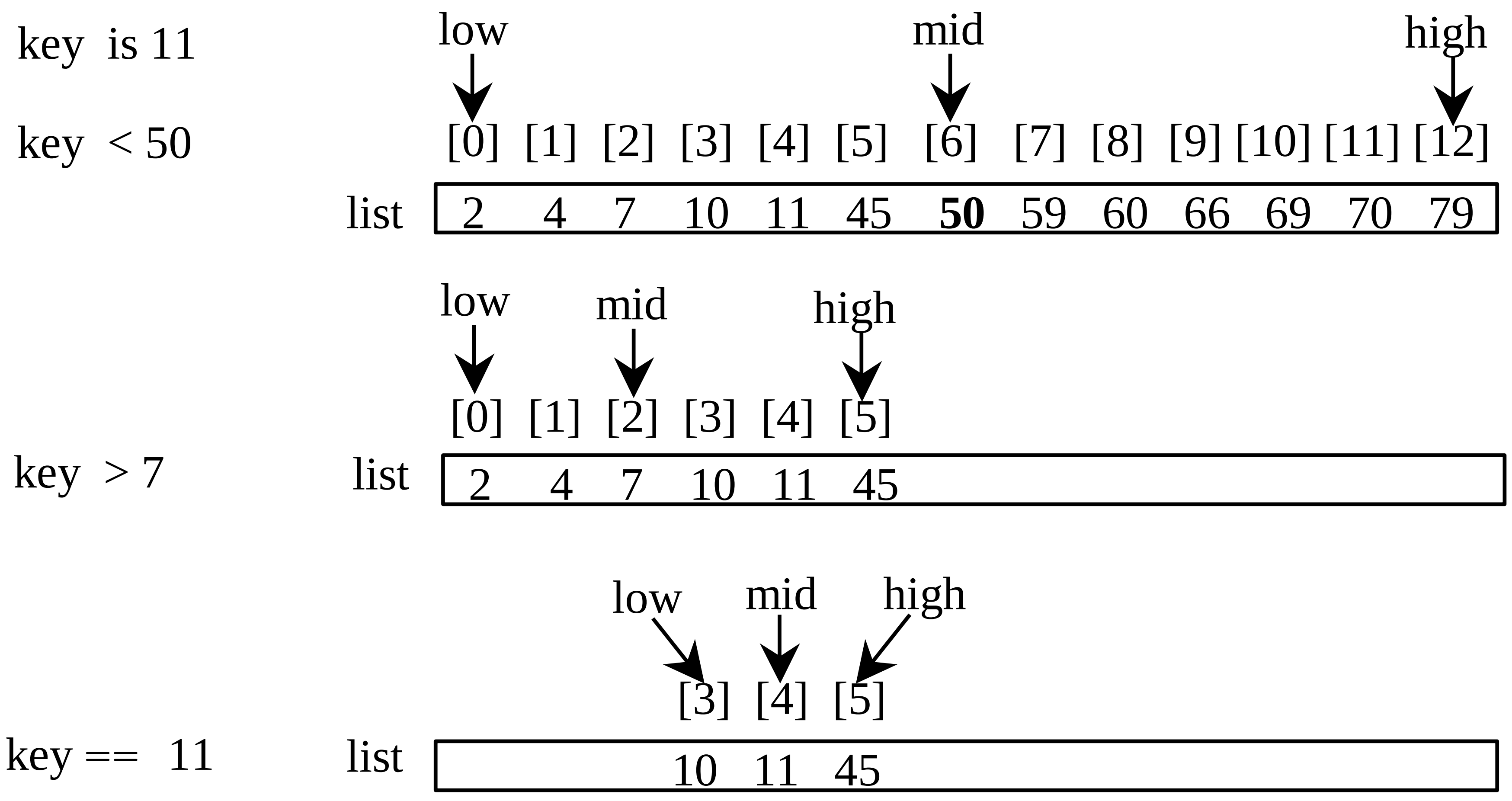
二分查找首先会定位在数组正中，然后将正中间的元素与关键字做一个比较：

- 如果关键字小于该元素，接下来你只需查找数组前一半；
- 如果二者相等，查找成功结束；
- 如果关键字大于该元素，接下来你只需要查找数组后一半。

二分查找示意



另一个例子

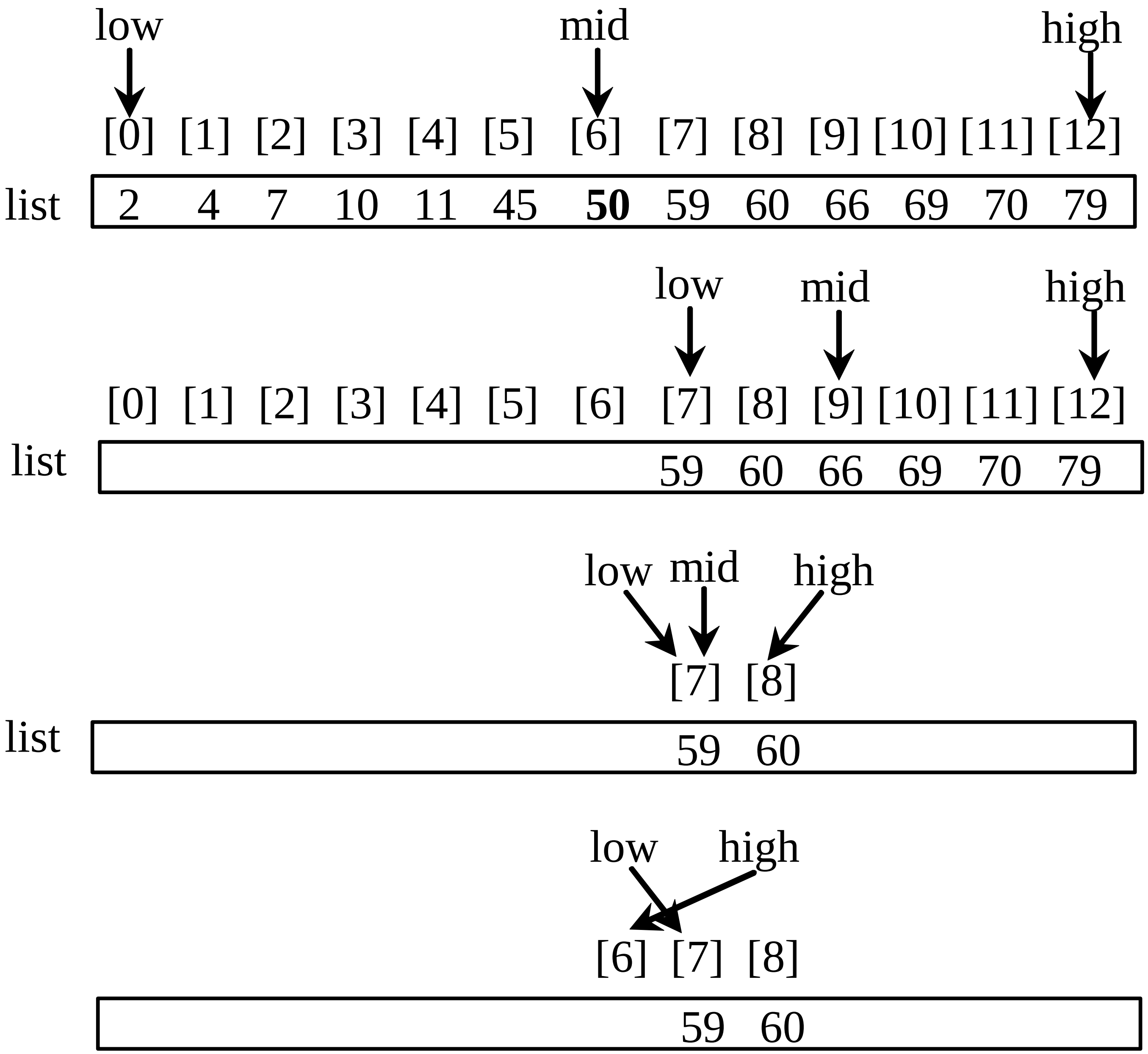

key is 11
key < 50
low
mid
high
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]
list
2 4 7 10 11 45 **50** 59 60 66 69 70 79
key > 7
low
mid
high
[0] [1] [2] [3] [4] [5]
list
2 4 7 10 11 45
key == 11
low
mid
high
[3] [4] [5]
list
10 11 45

key is 54

key > 50

key < 66

key < 59



实现代码

```
/** 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;

  while (high >= low) {
    int 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;
}
```

Arrays.binarySearch方法

Java的数组工具类提供了多个版本的 binarySearch供调用，支持对**已经按照升序排列**的int, double, char, short, long, float数组进行二分查找，例如：

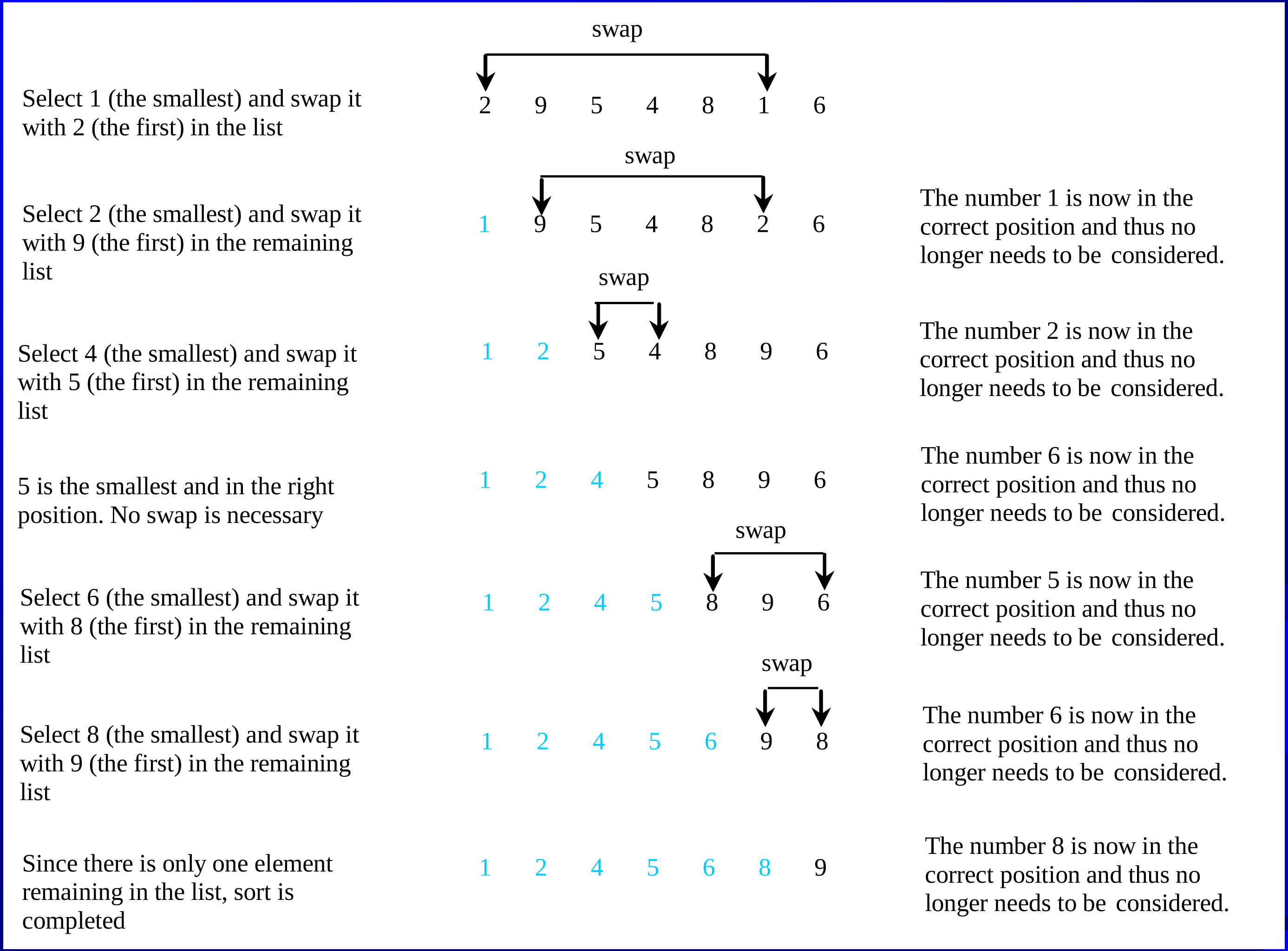
```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("Index is " +
  java.util.Arrays.binarySearch(list, 11)); //返回4，查找成功

char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("Index is " +
  java.util.Arrays.binarySearch(chars, 't')); //返回-4，查找失败
```

数组排序

- 排序是数组的常见任务。更多的排序算法会在《数据结构》课程里介绍，这里只介绍两个简单的排序算法：选择排序（*selection sort* ）和插入排序（*insertion sort*）。

选择排序

选择排序的算法是：每次选取一个数组的最小值，并把它交换到数组的最前面。下图是数组 {2, 9, 5, 4, 8, 1, 6}的排序过程。



Select 1 (the smallest) and swap it with 2 (the first) in the list

swap

2 9 5 4 8 1 6

Select 2 (the smallest) and swap it with 9 (the first) in the remaining list

swap

1 9 5 4 8 2 6

The number 1 is now in the correct position and thus no longer needs to be considered.

Select 4 (the smallest) and swap it with 5 (the first) in the remaining list

swap

1 2 5 4 8 9 6

The number 2 is now in the correct position and thus no longer needs to be considered.

5 is the smallest and in the right position. No swap is necessary

1 2 4 5 8 9 6

The number 6 is now in the correct position and thus no longer needs to be considered.

Select 6 (the smallest) and swap it with 8 (the first) in the remaining list

swap

1 2 4 5 8 9 6

The number 5 is now in the correct position and thus no longer needs to be considered.

Select 8 (the smallest) and swap it with 9 (the first) in the remaining list

swap

1 2 4 5 6 9 8

The number 6 is now in the correct position and thus no longer needs to be considered.

Since there is only one element remaining in the list, sort is completed

1 2 4 5 6 8 9

The number 8 is now in the correct position and thus no longer needs to be considered.

编码实现

```
for (int i = 0; i < list.length; i++)
{
  在 list[i..listSize-1]中找最小值;
  如果必要的话，把最小值交换到list[i];
  // list[i]排序完毕
  // 下一步要排序list[i+1..listSize-1]
}
```

```
for (int i = 0; i < list.length; i++)
{
  在 list[i..listSize-1]中找最小值;
  如果必要的话，把最小值交换到list[i];
  // list[i]排序完毕
  // 下一步要排序list[i+1..listSize-1]
}
```

展开

```
  double currentMin = list[i];
  int currentMinIndex = i;
  for (int j = i; j < list.length; j++) {
    if (currentMin > list[j]) {
      currentMin = list[j];
      currentMinIndex = j;
    }
  }
```

```
for (int i = 0; i < list.length; i++)
{
  在 list[i..listSize-1]中找最小值;
  如果必要的话，把最小值交换到list[i];
  // list[i]排序完毕
  // 下一步要排序list[i+1..listSize-1]
}
```

展开

```
    if (currentMinIndex != i) {
      list[currentMinIndex] = list[i];
      list[i] = currentMin;
    }
```

完整代码

```
/** The method for sorting the numbers */
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;
      }
    }
  }
```

插入排序

int[] myList = {2, 9, 5, 4, 8, 1, 6}; // 无序

算法的思想是每次在一个有序的列表中，新插入一个尚未排序的元素，让新的列表继续保持有序，直到整个数组插入完毕。

Step 1: Initially, the sorted sublist contains the first element in the list. Insert 9 to the sublist.

Step2: The sorted sublist is {2, 9}. Insert 5 to the sublist.

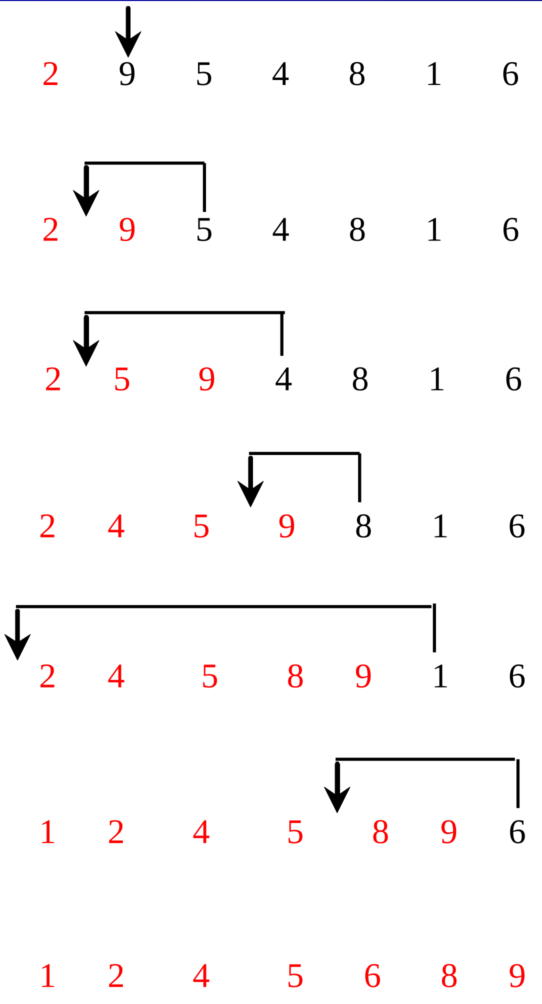
Step 3: The sorted sublist is {2, 5, 9}. Insert 4 to the sublist.

Step 4: The sorted sublist is {2, 4, 5, 9}. Insert 8 to the sublist.

Step 5: The sorted sublist is {2, 4, 5, 8, 9}. Insert 1 to the sublist.

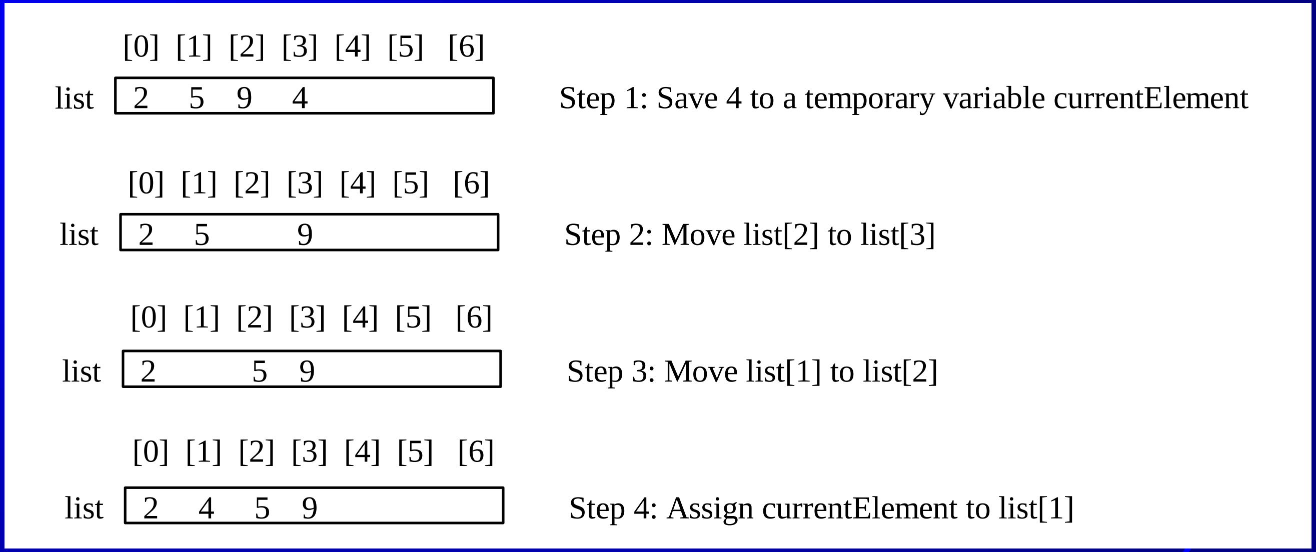
Step 6: The sorted sublist is {1, 2, 4, 5, 8, 9}. Insert 6 to the sublist.

Step 7: The entire list is now sorted



如何插入?

插入新元素的思路是，让元素从列表的最后一个元素往前移动，直到无法移动为止。


[0] [1] [2] [3] [4] [5] [6]
list 2 5 9 4
Step 1: Save 4 to a temporary variable currentElement
[0] [1] [2] [3] [4] [5] [6]
list 2 5 9
Step 2: Move list[2] to list[3]
[0] [1] [2] [3] [4] [5] [6]
list 2 5 9
Step 3: Move list[1] to list[2]
[0] [1] [2] [3] [4] [5] [6]
list 2 4 5 9
Step 4: Assign currentElement to list[1]

编程实现

```
for (int i = 1; 1; i < list.length; i++) {
  将list[i]元素插入已排序的list[0..i-1]中，并保持有序
  于是list[0..i]排序完毕
}
```

完整源代码

```
public static void insertionSort(double[] list) {
  for (int i = 1; i < list.length; i++) {
  /** Insert list[i] into a sorted sublist list[0..i-1] so that list[0..i] is sorted. */
    double currentElement = list[i];
    int k;
    for (k = i - 1; k >= 0 && list[k] > currentElement; k--) {
      list[k + 1] = list[k];
    }
    // Insert the current element into list[k + 1]
    list[k + 1] = currentElement;
  }
}
```

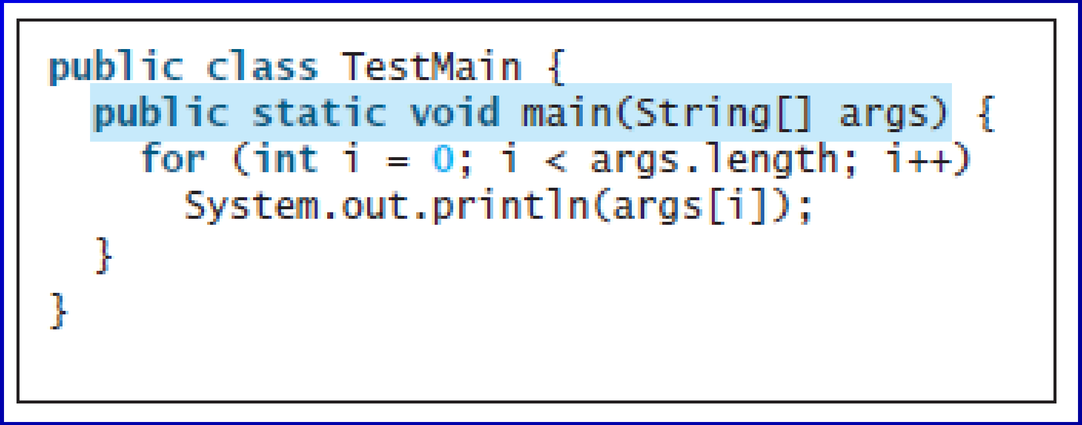
Arrays.sort方法

Java的数组工具类提供了多个版本的sort供调用，支持对int, double, char, short, long, float数组进行排序，例如：

```
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);
```

命令行参数

- Java的main函数有个形参args，用于接收命令行参数。例如下面的代码：



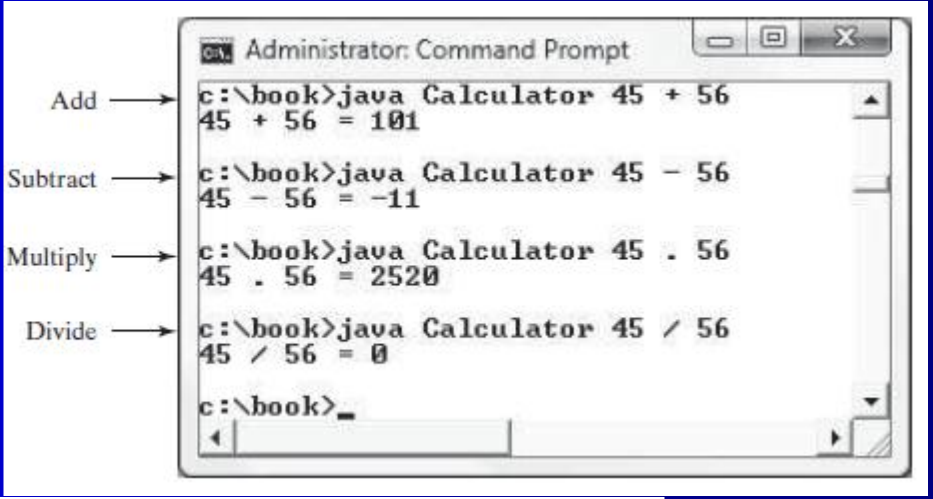
```
public class TestMain {
   public static void main(String[] args) {
      for (int i = 0; i < args.length; i++)
         System.out.println(args[i]);
   }
}
```

- 运行：java TestMain "First num" alpha 53
- 则args.length为3，3个元素分别为First num, alpha，53。与C/C++不同，TestMain本身不算参数。

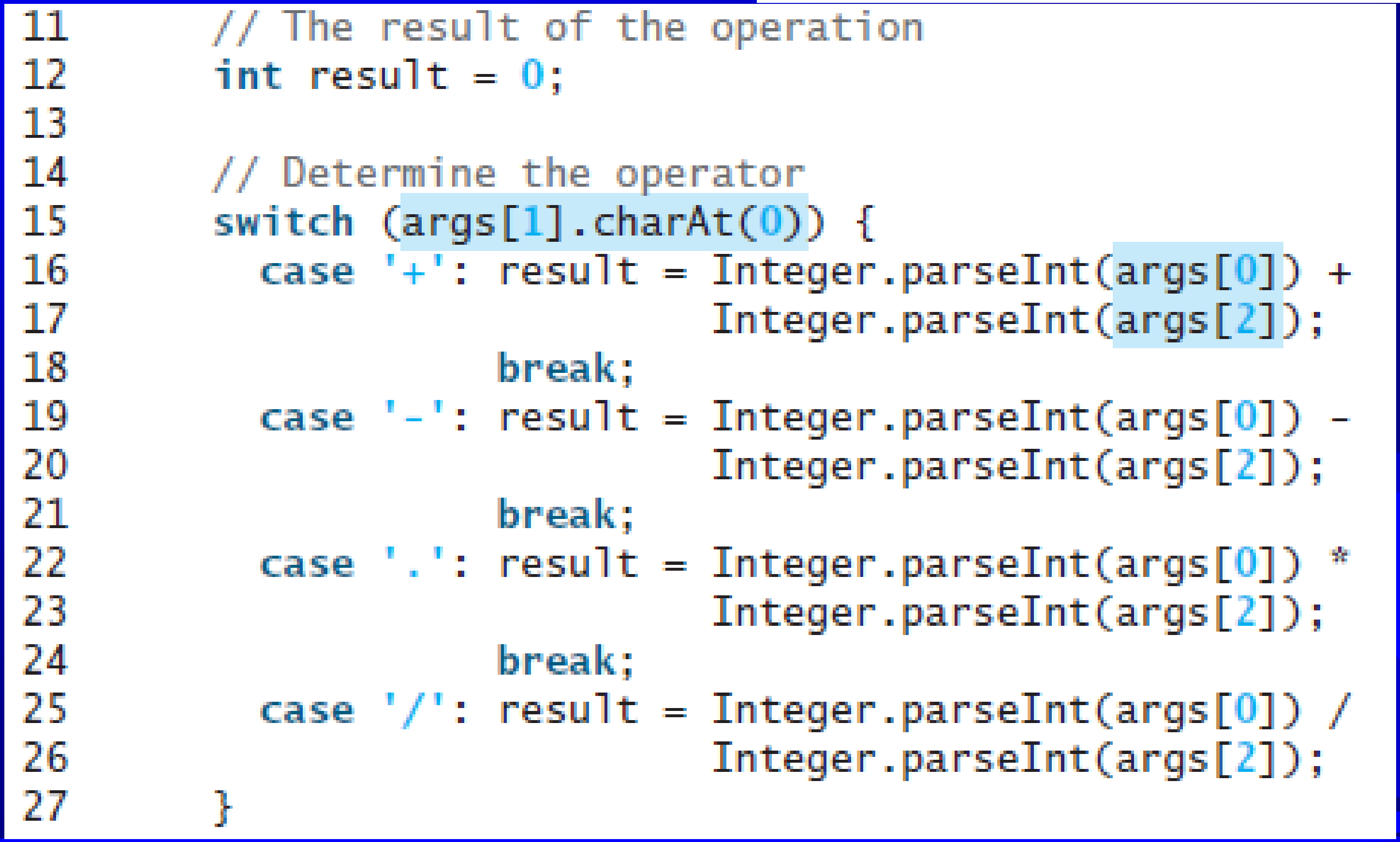
例题：命令行计算器

- 写一个命令行计算器，支持从参数中读取表达式并计算。输入格式如下：
  – java Calculator 2 + 3
- 输出格式如下：
  – 2 + 3 = 5
- 其中两个数与运算符之间都有空格，运算符有+-./四种。因为*有特殊含义（表示当前目录下所有文件），这里用.代替

关键代码及结果



```
c:\book>java Calculator 45 + 56
45 + 56 = 101

c:\book>java Calculator 45 - 56
45 - 56 = -11

c:\book>java Calculator 45 . 56
45 . 56 = 2520

c:\book>java Calculator 45 / 56
45 / 56 = 0

c:\book>_
```



```
    // The result of the operation
    int result = 0;

    // Determine the operator
    switch (args[1].charAt(0)) {
      case '+': result = Integer.parseInt(args[0]) +
                         Integer.parseInt(args[2]);
                break;
      case '-': result = Integer.parseInt(args[0]) -
                         Integer.parseInt(args[2]);
                break;
      case '.': result = Integer.parseInt(args[0]) *
                         Integer.parseInt(args[2]);
                break;
      case '/': result = Integer.parseInt(args[0]) /
                         Integer.parseInt(args[2]);
    }
```

THE END