第7章 数组（涵盖教材第7,8章）

目 录

contents

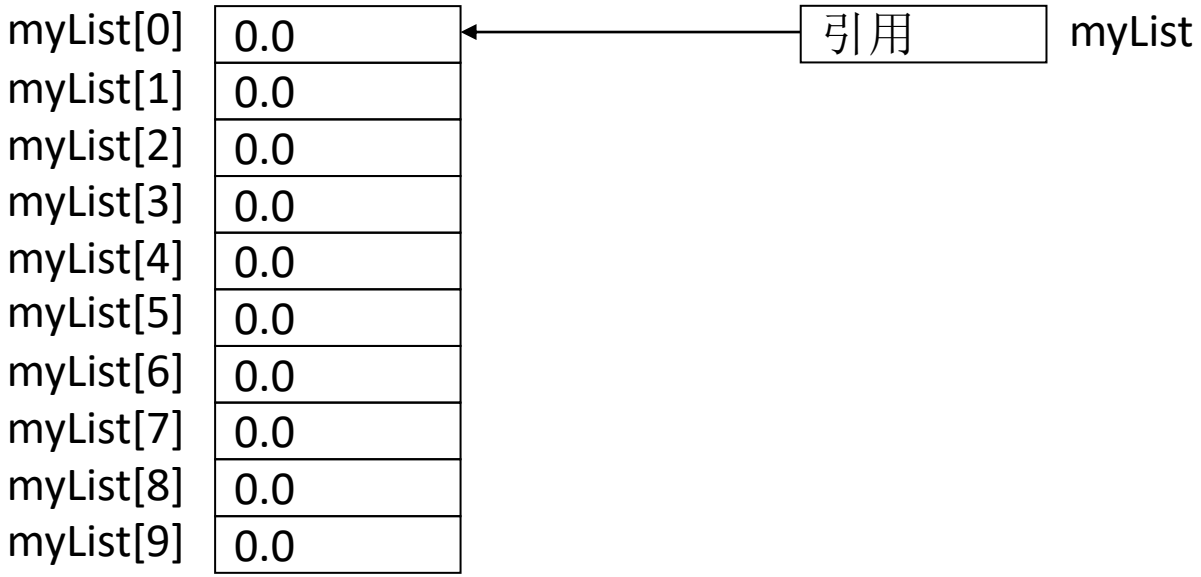
7.1 数组的基础知识

什么是数组

◆数组(array)是相同类型变量集合(这里的集合不是JDK的Collection)

◆数组类型的变量是引用相同类型变量集合的引用变量

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



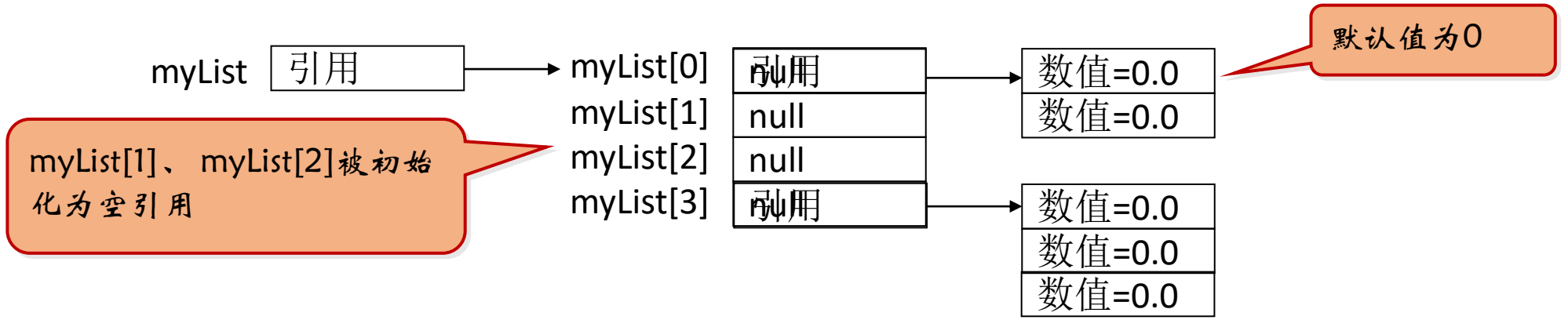
◆凡使用new后，内存单元都初始化为0（值）或null（引用）

7.1 数组的基础知识

什么是数组

◆数组元素本身也可以是引用变量。

```
double[ ][ ] myList = new double[4][ ];          //创建一个二维数组
myList[0]=new double[2];          // myList[0]是一个引用变量，指向一个一维数组（2个元素）
myList[3]=new double[3];          // myList[3]是一个引用变量，指向一个一维数组（3个元素）
```



◆多维数组只是数组的数组，故数组元素也可能是引用类型变量

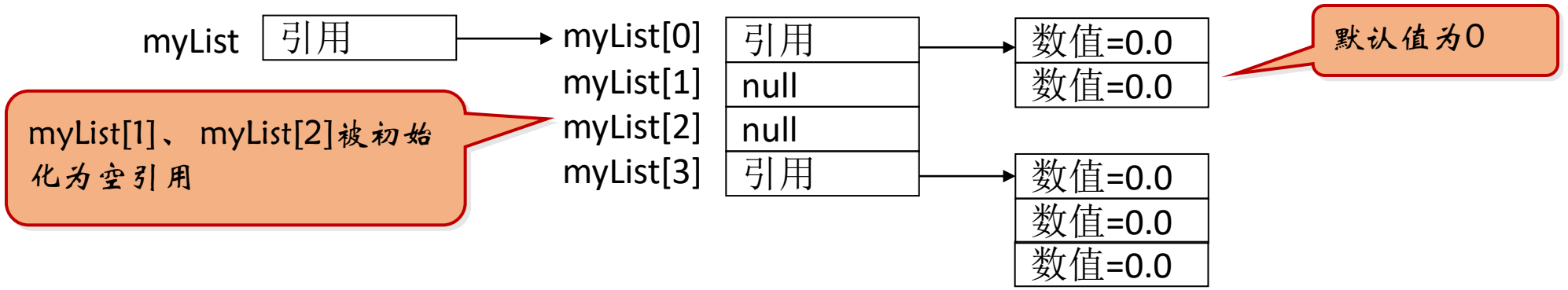
◆凡使用new后，内存单元都初始化为0或null

7.1 数组的基础知识

什么是数组

◆数组元素本身也可以是引用变量。

```
double[ ][ ] myList = new double[4][ ];          //创建一个二维数组
myList[0]=new double[2];          // myList[0]是一个引用变量，指向一个一维数组（2个元素）
myList[3]=new double[3];          // myList[3]是一个引用变量，指向一个一维数组（3个元素）
```



◆多维数组只是数组的数组，故数组元素也可能是引用类型变量

◆凡使用new后，内存单元都初始化为0或null

7.1 数组的基础知识

声明一维数组引用变量

◆任何实例化的数组都是Object的子类。数组引用变量声明语法：

datatype[ ] arrayRefVar;　//提倡的写法：类型在前，[ ]在后

例如：

double[ ] myList;　//这时myList为null

或者

datatype arrayRefVar[ ];

例如：

double myList[ ];

double [ ] a[ ];　//double[][] a;

◆数组变量是引用类型的变量，声明数组引用变量并不分配数组内存空间。必须通过new实例化数组来分配数组内存空间。

7.1 数组的基础知识

创建数组-new

◆使用new操作符创建数组。

arrayRefVar = new datatype[arraySize];

例如：

myList = new double[10]; //这时才分配内存

◆声明和创建在一条语句中。

datatype[ ] arrayRefVar= new datatype[arraySize];

或者

datatype arrayRefVar[ ] = new datatype[arraySize];

例如：

double[ ] myList = new double[10];

或者

double myList[ ] = new double[10];

7.1 数组的基础知识

数组元素初始化

◆新创建的数组对象，其元素根据类型被设置为默认的初始值（实际上都为0）。

- ➢数值类型为0
- ➢字符类型为’ \u0000’ //u后面为十六进制，必须4位写满
- ➢布尔类型为false
- ➢引用类型为null

◆数组可以在声明后的花括号中提供初始值。

```
double[ ] myList = {1.9, 2.9, 3, 3.5}//可以将int转化为double类型，这时不用指定维度size
或者
double[ ] myList;
myList = new double[ ] {1.9, 2, 3.4, 3.5} //可以将int转化为double类型，声明和创建不在一
条语句时，不能直接用{ }来初始化
```

7.1 数组的基础知识

访问数组

◆数组的大小在创建这个数组之后不能被改变。用以下语法访问数组的长度：

arrayRefVar.length

例如：

myList.length的值为10。

◆数组元素通过索引进行访问。元素的索引从0开始，范围从0到length-1。

arrayRefVar[index]

例如：

myList[0]表示数组的第一个元素

myList[9]表示数组的最后一个元素

7.1 数组的基础知识

数组示例

编写程序，读入6个整数，找出它们中的最大值。

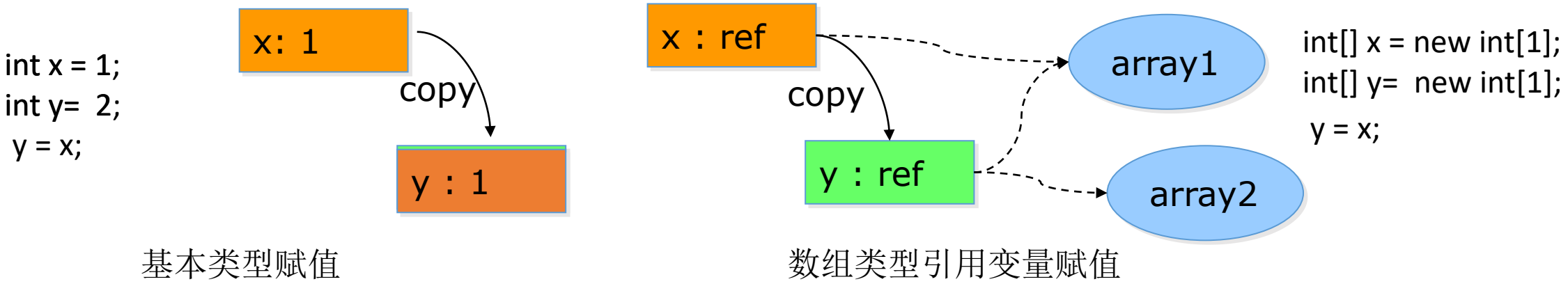
```
public class TestArray {
        public static void main(String[ ] args) {/** Main method */
                final int TOTAL_NUMBERS = 6;
                int[] numbers = new int[TOTAL_NUMBERS];

                // Read all numbers
                for (int i = 0; i < numbers.length; i++) {
                        String numString = JOptionPane.showInputDialog("Enter a number:");

                        numbers[i] = Integer.parseInt(numString);
                }
                // Find the largest
                int max = numbers[0];
                for (int i = 1; i < numbers.length; i++) {
                        if (max < numbers[i])           max = numbers[i];
                }
                System.out.println("Max number is " + max);
        }
}
```

7.2 数组的复制

◆直接使用赋值语句不能实现数组复制，结果是两个数组引用变量指向同一个数组对象（浅拷贝赋值）。



u 复制数组的方法

- ➢使用循环来复制每个元素
- ➢使用System.arraycopy方法：两个数组都预先实例化了
- ➢调用数组的clone方法复制：被复制的数组变量可以没有实例化

7.2 数组的复制

◆复制数组的方法

➢使用循环来复制每个元素

```
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];//=赋值对引用类型是浅拷贝
}
```

➢使用System.arraycopy方法： sourceArray， targetArray都已经实例化好

```
arraycopy(sourceArray,srcPos,targetArray,tarPos,length);
System. arraycopy(sourceArray,0,targetArray,0, sourceArray.length);//target要求已存在
```

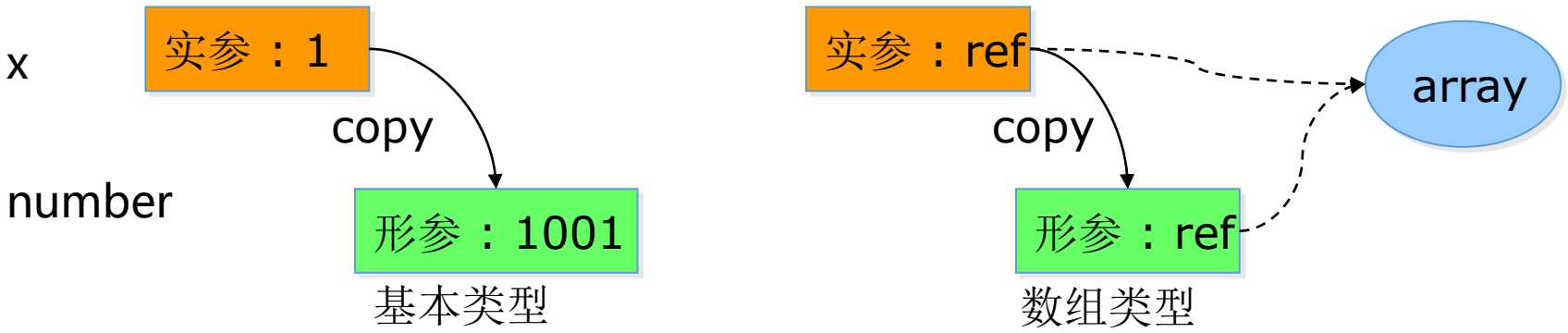
➢使用数组的clone方法： **targetArray可先不实例化**

```
int[] targetArray= sourceArray.clone( );
```

7.3 将数组传递给方法（数组作为方法参数）

◆可以将数组变量作为实参传递给方法。

- ➢基本数据类型传递的是实际值的拷贝，修改形参，不影响实参。
- ➢数组引用变量传递的是对象的引用，修改形参引用的数组，将改变实参引用的数组。



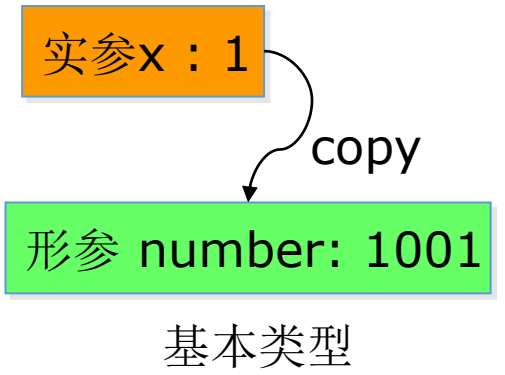
◆也可以从方法中返回数组，返回的也是引用。

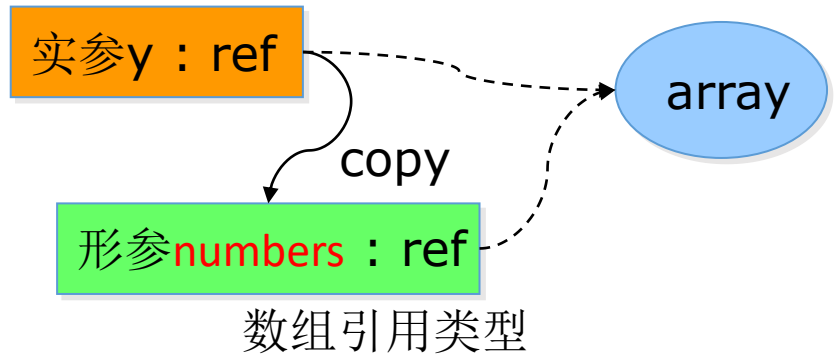
7.3 将数组传递给方法

数组传递给方法示例

```
public class TestPassArraySimple{
        /** Main method */
        public static void main(String[ ] args) {
                int x =1;
                int[ ] y = new int[10];
                y[0] = 20;

                m(x, 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;      //不改变x的值：值参传递
                numbers[0] = 5001; //改变y[0]
        }
}
```





7.3 将数组传递给方法

String、Integer这样的对象作为参数传递要注意的问题：

```
public class CallByReferenceException {
    public static void main(String[] args) {
            Integer x = new Integer(10);
            testInteger(x);
            System.out.println("x = " + x);

            String y = "ABC";
            testString(y);
            System.out.println("y = " + y);
    }
    public static void testInteger(Integer i) {
            i = 20;
            System.out.println("i = " + i);
    }
    public static void testString(String s) {
            s = "abc";
            System.out.println("s = " + s);
    }
}
```

输出结果：
i = 20
x = 10
s = abc
y = ABC

为什么传引用但是形参变了而实参没有变？

引用类型的实参传递给形参后，实参、形参指向同一个对象。但是，对于String类、基本数据类型的包装类型的实参传递给形参，形参变了不会导致实参变化。这是为什么？

7.3 将数组传递给方法

String、Integer这样的对象作为参数传递要注意的问题：

Integer.class | String.class

```
831             return IntegerCache.cache[i + (-IntegerCache.low)];
832         return new Integer(i);
833     }
834
835     /**
836      * The value of the {@code Integer}.
837      *
838      * @serial
839      */
840     private final int value;
841
842     /**
843      * Constructs a newly allocated {@code Integer} object that
844      * represents the specified {@code int} value.
845      *
846      * @param   value   the value to be represented by the
847      *                  {@code Integer} object.
848      */
849     public Integer(int value) {
850         this.value = value;
851     }
852
```

这是因为String、Integer的内容是不可更改的
请看String、Integer的具体实现

在Integer内部，用
private final int value来保存整数值

7.3 将数组传递给方法

String、Integer这样的对象作为参数传递要注意的问题：

Integer.class | String.class

```
102  * @author  Martin Buchholz
103  * @author  Ulf Zibis
104  * @see     java.lang.Object#toString()
105  * @see     java.lang.StringBuffer
106  * @see     java.lang.StringBuilder
107  * @see     java.nio.charset.Charset
108  * @since   JDK1.0
109  */
110
111 public final class String
112     implements java.io.Serializable, Comparable<String>, CharSequence {
113     /** The value is used for character storage. */
114     private final char value[];
115
116     /** Cache the hash code for the string */
117     private int hash; // Default to 0
118
119     /** use serialVersionUID from JDK 1.0.2 for interoperability */
120     private static final long serialVersionUID = -6849794470754667710L;
121
```

这是因为String、Integer的内容是不可更改的
请看String、Integer的具体实现

在String内部，用
private final char value[] 来保存字符串内容

7.3 将数组传递给方法

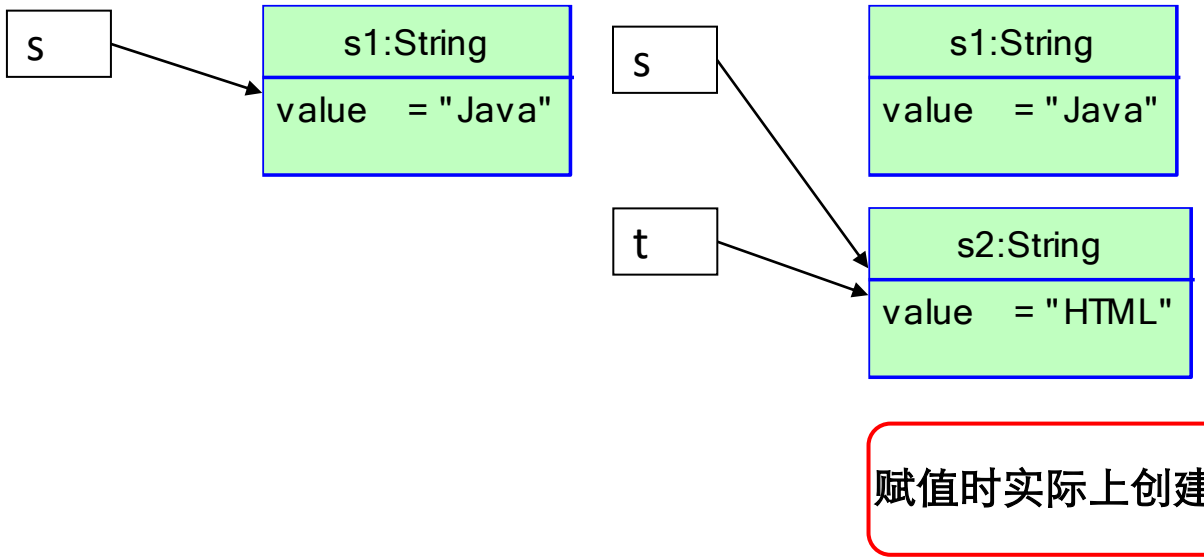
对于String、Integer这样内容不可改变的对象，当对其赋值时实际上创建了一个新的对象

◆例如，字符串对象创建之后，其内容是不可修改的。

String s = “Java”；

s = “HTML”；

String t =s;




```
public class CallByReferenceException {
    public static void main(String[] args) {
        Integer x = new Integer(10);
        testInteger(x);
        System.out.println("x = " + x);

        String y = "ABC";
        testString(y);
        System.out.println("y = " + y);
    }
    public static void testInteger(Integer i) {
        i = 20;
        System.out.println("i = " + i);
    }
    public static void testString(String s) {
        s = "abc";
        System.out.println("s = " + s);
    }
}
```

7.3 将数组传递给方法

可以通过debug来观察对象引用：

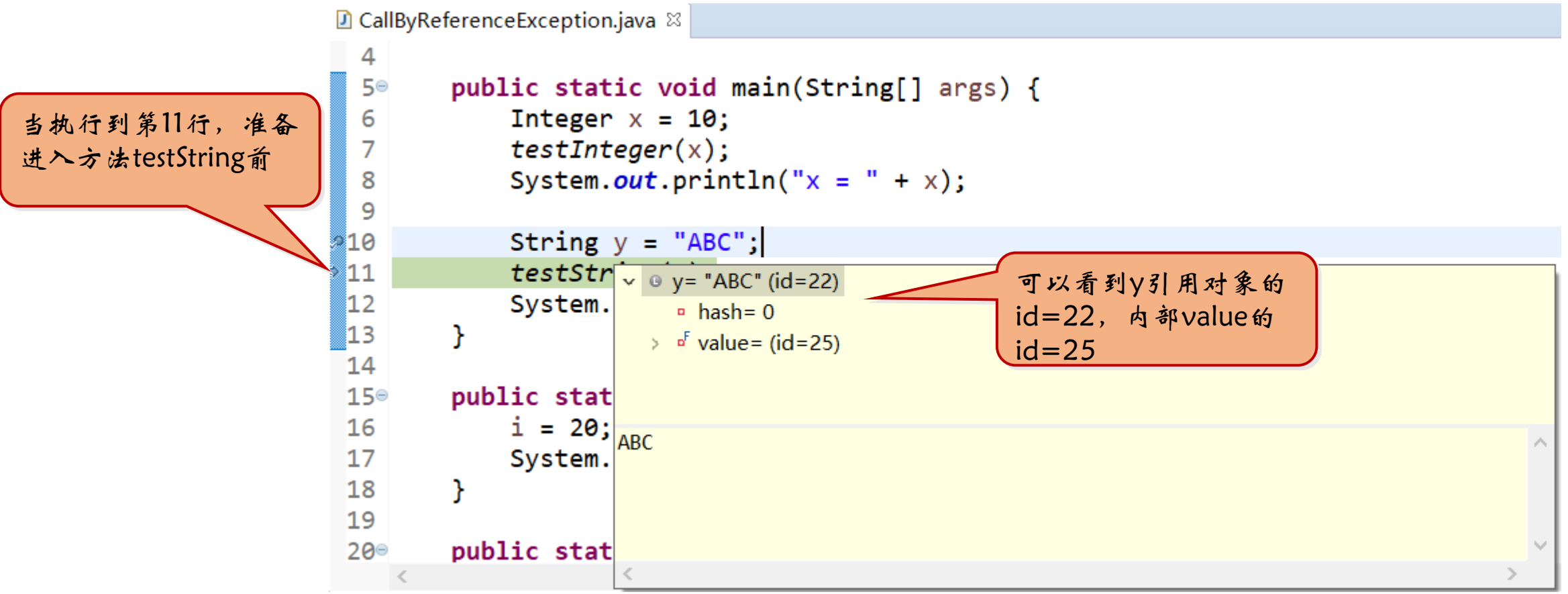
在第10行和第21行设置二个断点，观察实参引用y和形参引用s

CallByReferenceException.java

```
1  package ch9;
2
3  public class CallByReferenceException {
4
5      public static void main(String[] args) {
6          Integer x = 10;
7          testInteger(x);
8          System.out.println("x = " + x);
9
10         String y = "ABC";
11         testString(y);
12         System.out.println("y = " + y);
13     }
14
15     public static void testInteger(Integer i) {
16         i = 20;
17         System.out.println("i = " + i);
18     }
19
20     public static void testString(String s) {
21         s = "abc";
22         System.out.println("s = " + s);
23     }
24
25 }
26
```

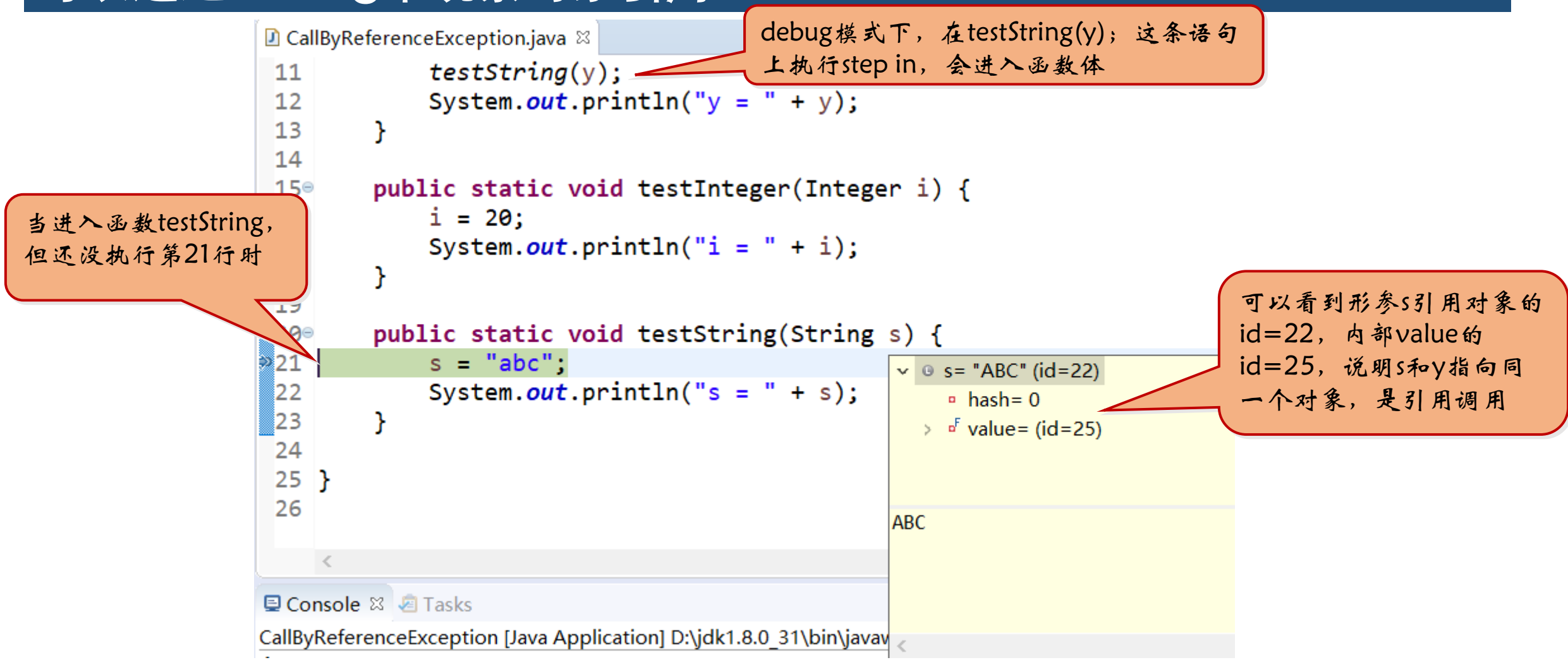
7.3 将数组传递给方法

可以通过debug来观察对象引用：



7.3 将数组传递给方法

可以通过debug来观察对象引用：



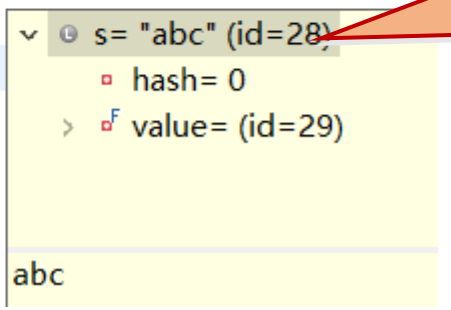
7.3 将数组传递给方法

可以通过debug来观察对象引用：

CallByReferenceException.java

```
11          testString(y);
12          System.out.println("y = " + y);
13      }
14
15      public static void testInteger(Integer i) {
16          i = 20;
17          System.out.println("i = " + i);
18      }
19
20      public static void testString(String s) {
21          s = "abc";
22          System.out.println("s = " + s);
23      }
24
25  }
26
```

当执行完第21行后（对s的赋值）



可以看到形参s引用对象的id=28，内部value的id=28，说明s这时和y指向的不是同一个对象

这也是为什么形参内容变了，实参内容没有变。为什么这么设计？
以Integer为例，因为Integer是int的包装类，它必须要和int的特性一致：
int作为方法参数调用时，方法内部对参数的改变不会影响实参。所以Integer必须这么设计

7.4 从方法中返回数组

◆调用方法时，可向方法传递数组引用，也可从方法中返回数组引用

➢下面的方法返回一个与输入数组顺序相反的数组引用

```
public static int[ ] reverse (int[ ] list){
      int[ ] result = new int [ list.length ];
     for(int i = 0, j = result.lenght - 1; i < list.length; i++,j--){
                        result [ j ] = list [i];
      }
      return result;
}
int[ ] list1 = {1, 2, 3, 4, 5, 6};
int[ ] list2 = reverse(list1);
```

7.5 可变长参数列表

- 可以把类型相同但个数可变的参数传递给方法。方法中的可变长参数声明如下

  typeName ... parameterName

- 在方法声明中，指定类型后面跟省略号
- 只能给方法指定一个可变长参数，同时该参数必须是最后一个参数
- Java将可变长参数当数组看待，通过length属性得到可变参数的个数

```
print(String... args){   //可看作String [ ]args

        for(String  temp:args)
                System.out.println(temp);
        System.out.println(args.length);
}
```

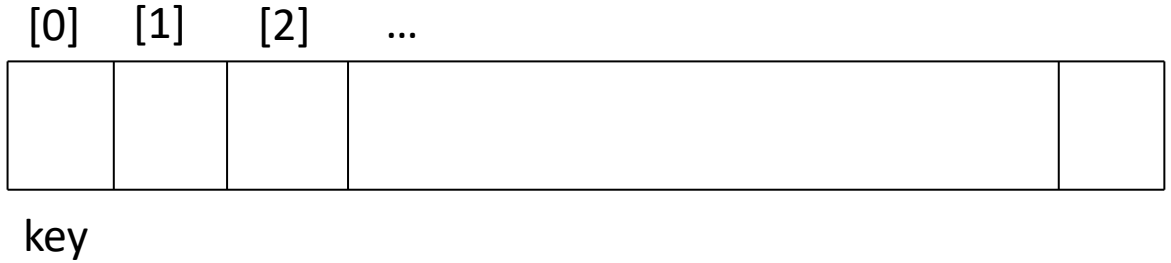
- 调用该方法

```
print("hello","lisy");
```

7.6 数组的查找和排序

数组的查找:线性搜索和二分搜索

◆线性搜索法(linear searching)将一个值与数组的每个元素进行比较。如果找到相同的元素，返回元素的索引；否则返回-1。




```
/** 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;
    }
```

◆最坏情况下需要比较N次。平均要比较N/2次。效率不高O(N)。

7.6 数组的查找和排序

数组的查找-二分搜索法

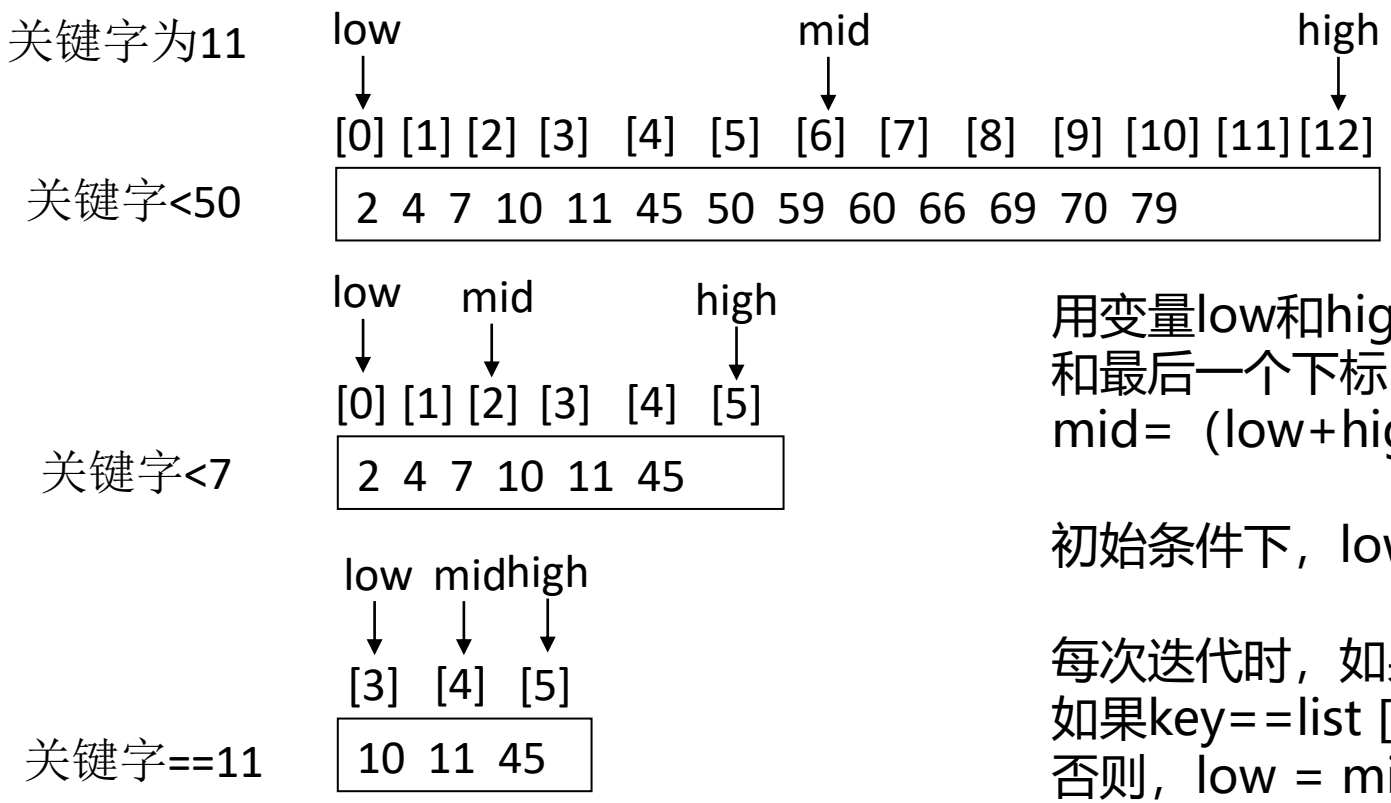
◆二分搜索法(binary searching)是在一个已排序的数组中搜索特定元素。假设数组已按升序排列，将关键字与数组中间元素进行比较：

- ➢如果关键字比中间元素小，则在前一半数组中搜索；
- ➢如果关键字与中间元素相同，查找结束；
- ➢如果关键字比中间元素大，则在后一半数组中搜索。

◆二分法每比较一次就排除一半元素。假设数组有N个元素，为讨论方便，设N是2的幂指数。经过第1次比较，剩下N/2个元素需要查找，经过第2次，剩下N/2/2个元素。经过k次，剩下$N/2^k$个元素。当$k=\log_2 N$时，只剩下一个元素。所以最坏情况下该算法需要比较$\log_2 N+1$次。假设N=1024（2^{10}），最多只需要比较11次，而线性查找最坏需要1024次。因此算法的复杂度$O(\log_2 N)$。

7.6 数组的查找和排序

数组的查找-二分搜索法



用变量low和high分别记当前剩下元素的第一个和最后一个下标，mid表示中间元素下标。
mid=（low+high）/2

初始条件下，low=0，high=list.length-1

每次迭代时，如果key<list [mid], high=mid-1；
如果key==list [mid],则返回mid
否则，low = mid + 1

迭代继续条件：high>= low

7.6 数组的查找和排序

数组的查找-二分搜索法

```
/** 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;
  }
```

7.6 数组的查找和排序

数组的排序

◆选择排序算法：假设要将数组按升序排列

- 将列表中的元素最大值放在最后一个位置
- 将剩下元素的最大值放在倒数第二的位置
- 以此类推，直到剩下一个数为止。

7.6 数组的查找和排序

数组的排序

```
static void selectionSort(double[] list) {
        for (int i = list.length - 1; i >= 0; i--) {
                // Find the maximum in the list[0..i]
                double currentMax = list[0];
                int currentMaxIndex = 0;

                for (int j = 0; j <= i; j++) {
                        if (currentMax < list[j]) {
                                currentMax = list[j];
                                currentMaxIndex = j;
                        }
                }

                // Swap list[i] with list[currentMaxIndex] if necessary;
                if (currentMaxIndex != i) {
                        list[currentMaxIndex] = list[i];
                        list[i] = currentMax;
                }
        }
}
```

7.7 Arrays类

- java.util.Arrays类包括各种静态方法，其中实现了数组的排序和查找
  - 排序

    double[ ] numbers={6.0, 4.4, 1.9, 2.9};

    java.util.Arrays.sort(numbers); //注意直接在原数组排序
  - 二分查找

    int[ ] list={2, 4, 7, 10, 11, 45, 50};

    int index = java.util.Arrays.binarySearch(list, 11);
- Arrays和String是常用的两个值得研究的类。

7.8 命令行参数

- 可以从命令行向java程序传递参数。参数以空格分隔，如果参数本身包含空格，用双引号括起来。格式：

  java 类名 参数1 参数2 …

  例如 java TestMain “First number” alpha 53

- 命令行参数将传递给main方法的args参数。args是一个字符串数组，可以通过数组下标访问每个参数。

  public static void main(String[ ] args)

  - 注意Java的命令行参数不包括类名， args.length==3

- 可变长参数用…定义。args是一个字符串数组，可以定义为可变长参数。String … args可以当成String[ ] args数组使用。

  public static void main(String … args) //也可以作为启动函数

- 注意在定义重载函数时，编译器认为String[] args和String … args类型相同

7.9 多维数组

二维数组：数组的数组

◆声明二维数组引用变量

dataType[ ][ ] refVar;

◆创建数组并赋值给引用变量：当指定了行、列大小，是矩阵数组（每行的列数一样）。非矩阵数组则需逐维初始化

refVar = new dataType[rowSize][colSize]; （这时元素初始值为0或null)

◆在一条语句中声明和创建数组

dataType[ ][ ] refVar = new dataType[rowSize][colSize];

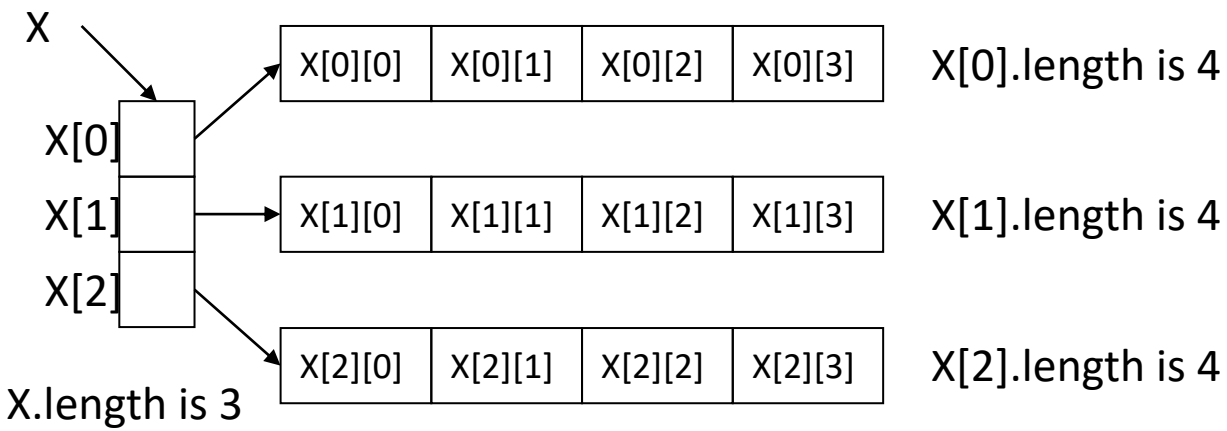
或者

dataType refVar[ ][ ] = new dataType[rowSize][colSize];

7.9 多维数组

二维数组的长度

- 二维数组的每个元素是一个一维数组。int[][] X =new int[3][4];
  - X指向的是内存里的一个一维数组，数组X的长度是数组X的元素的个数，可由X.length得到，X.length = 3。
  - 元素X[i]是引用，指向另一个一维数组，其长度可由X[i].length得到。



- X.length是final的，不可修改。

7.9 多维数组

不规则数组

- 二维数组每一行的的列数可以不同。
- 创建不规则二维数组时，可以只指定第一维下标。这时第一维的每个元素为null（如下所示），必须为每个元素创建数组。例如：

```
int[ ][ ] x = new int[5][ ]; //第一维的每个元素为null
x[0] = new int[5];//为每个元素创建数组
x[1] = new int[4];
x[2] = new int[3];
x[3] = new int[2];
x[4] = new int[1];
//x.length=5
//x[2].length=3
//x[4].length=1，只能取x[4].length的值(它是final的)
```