第6章
Introduction to Java Programming, 8E

一维数组

学习目标

- 描述数组在程序设计中的必要性（6.1节）。
- 声明数组引用变量、创建数组（6.2.1～6.2.2节）。
- 初始化数组中的值（6.2.3节）。
- 使用下标变量访问数组元素（6.2.4节）。
- 利用一条数组初始化语法声明、创建和初始化数组（6.2.5节）。
- 编写程序实现常用的数组操作（显示数组，对所有元素求和，求最小和最大元素，随意打乱，移动元素）（6.2.6节）。
- 使用for-each循环简化程序设计（6.2.7节）。
- 在LottoNumbers和DeckOfCards问题中应用数组（6.3～6.4节）。
- 将一个数组的内容复制到另一个数组（6.5节）。
- 开发和调用带数组参数和数组返回值的方法（6.6～6.7节）。
- 定义带变长参数列表的方法（6.8节）。
- 使用线性查找算法（6.9.1节）或二分查找算法（6.9.2节）查找数组的元素。
- 使用选择排序法对数组排序（6.10.1节）。
- 使用插入排序法对数组排序（6.10.2节）。
- 使用Arrays类中的方法（6.11节）。

6.1 引言

在执行程序的过程中，经常需要存储大量的数据，例如，假设需要读取100个数，计算它们的平均值，然后找出有多少个数大于平均值。首先，程序读入这些数并且计算它们的平均值，然后用每个数与平均值进行比较判断它是否大于平均值。为了完成这个任务，必须将这些数全部存储到变量中。必须声明100个变量，并且重复书写100次几乎完全相同的代码。这样编写程序的方式是不太现实的，那么该如何解决这个问题呢？

这就需要一个高效的有条理的方法。Java和许多高级语言都提供了一种称作数组（array）的数据结构，可以用它来存储一个元素个数固定且元素类型相同的有序集。在现在这个例子中，可以将所有的100个数存储在一个数组中，通过一个一维数组变量访问它。解决方案看起来就像这样：

```
1 public class AnalyzeNumbers {
2   public static void main(String[] args) {
3     final int NUMBER_OF_ELEMENTS = 100;
4     double[] numbers = new double[NUMBER_OF_ELEMENTS];
5     double sum = 0;
6
7     java.util.Scanner input = new java.util.Scanner(System.in);
8     for (int i = 0; i < NUMBER_OF_ELEMENTS; i++) {
9       System.out.print("Enter a new number: ");
10      numbers[i] = input.nextDouble();
11      sum += numbers[i];
12    }
13
14    double average = sum / NUMBER_OF_ELEMENTS;
15
```

```
16      int count = 0; // The number of elements above average
17      for (int i = 0; i < NUMBER_OF_ELEMENTS; i++)
18        if (numbers[i] > average)
19          count++;
20
21      System.out.println("Average is " + average);
22      System.out.println("Number of elements above the average "
23        + count);
24    }
25 }
```

程序的第4行创建了一个有100个元素的数组，第10行将这些数存储在数组中，第11行将每个数加到sum中，并在第14行获得其平均值。然后，将数组中的每个数和平均值进行比较，计算比平均值大的数的个数（第16～19行）。

本章介绍一维数组。第7章将介绍二位数组和多维数组。

6.2 数组的基本知识

数组是用来存储数据的集合，但是，通常我们会发现把数组看做一个存储具有相同类型的变量集合会更有用。无须声明单个变量，例如：number0，number1，...，number99，只要声明一个数组变量numbers，并且用numbers[0]，numbers[1]，...，numbers[99]来表示单个变量。本节介绍如何声明数组变量、创建数组以及使用下标变量处理数组。

6.2.1 声明数组变量

为了在程序中使用数组，必须声明一个引用数组的变量，并指明数组的元素类型。下面是声明数组变量的语法：

```
elementType[ ] arrayRefVar;(元素类型[ ] 数组引用变量;)
```

elementType可以是任意数据类型，但是数组中所有的元素都必须具有相同的数据类型。例如：下面的代码声明变量myList，它引用一个具有double型元素的数组。

```
double[] myList;
```

注意 也可以用elementType arrayRefVar[]（元素类型 数组引用变量[]）声明数组变量。这种来自C语言的风格被Java采纳以适用于C程序员。推荐使用elementType[] arrayRefVar（元素类型[]数组引用变量）风格。

6.2.2 创建数组

不同于基本数据类型变量的声明，声明一个数组变量时并不在内存中给数组分配任何空间。它只是创建一个对数组的引用的存储位置。如果变量不包含对数组的引用，那么这个变量的值为null。除非数组已经被创建，否则不能给它分配任何元素。声明数组变量之后，可以使用下面的语法用new操作符创建数组：

```
arrayRefVar = new elementType[arraySize];(数组旨用变量=new元素类型[数组大小];)
```

这条语句做了两件事情：1）使用new elementType[arraySize]创建了一个数组；2）把这个新创建的数组的引用赋值给变量arrayRefVar。

声明一个数组变量、创建数组、然后将数组引用赋值给变量这三个步骤可以合并在一条语句里，如下所示：

```
elementType[] arrayRefVar = new elementType[arraySize];
(元素类型[]数组旨用变量=new元素类型[数组大小];)
```

或

```
elementType arrayRefVar[] = new elementType[arraySize];
(元素类型 数组旨用变量=new元素类型[数组大小];)
```

下面是使用这条语句的一个例子：

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

这条语句声明了数组变量myList，创建一个由10个double型元素构成的数组，并将该数组的引用赋值给myList。使用以下语法给这些元素赋值：

```
arrayRefVar[index] = value;
```

例如，下面的代码可以初始化数组：

```
myList[0] = 5.6;
myList[1] = 4.5;
myList[2] = 3.3;
myList[3] = 13.2;
myList[4] = 4.0;
myList[5] = 34.33;
myList[6] = 34.0;
myList[7] = 45.45;
myList[8] = 99.993;
myList[9] = 11123;
```

图6-1给出这个数组的全貌。

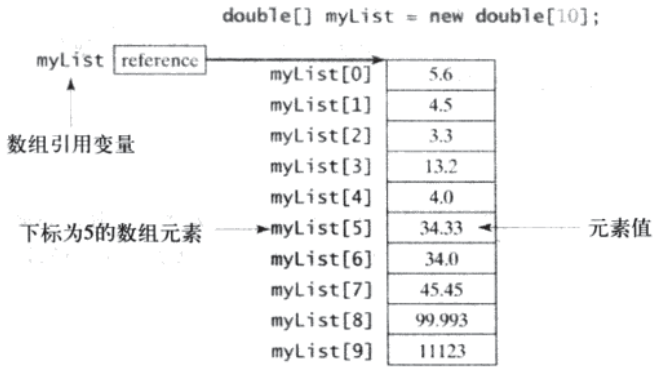


图6-1 数组myList包含10个double型元素并且下标从0到9为int型

注意 一个数组变量看起来似乎是存储了一个数组，但实际上它存储的是指向数组的引用。严格地讲，一个数组变量和一个数组是不同的，但多数情况下它们的差别是可以忽略的。因此，为了简化，通常可以说myList是一个数组，而不用更长的陈述：myList是一个含有10个double型元素数组的引用变量。

6.2.3 数组大小和默认值

当给数组分配空间时，必须通过指定该数组能够存储的元素个数来确定数组大小。创建数组之后就不能再修改它的大小。可以使用arrayRefVar.length求得数组的大小。例如：myList.length为10。

当创建数组后，它的元素被赋予默认值，数值型基本数据类型的默认值为0，char型的默认值为'\u0000'，boolean型的默认值为false。

6.2.4 数组下标变量

数组元素可以通过下标访问。数组下标是基于0的，也就是说，其范围从0开始到arrayRefVar.length-1结束。例如，在图6-1中，数组myList包含10个double值，而且下标从0到9。

数组中的每个元素都可以使用下面的语法表示，称为下标变量（indexed variable）：

```
arrayRefVar[index];(数组引用变量[下标];)
```

例如：myList[9]表示数组myList的最后一个元素。

警告 一些语言使用圆括号引用数组元素，例如myList(9)。而Java语言使用方括号，例

如myList[9]。

创建数组后，下标变量与正常变量的使用方法相同。例如：下面的代码是将myList[0]和myList[1]的值相加赋给myList[2]。

```
myList[2] = myList[0] + myList[1];
```

下面的循环是将0赋给myList[0]，1赋给myList[1]，...，9赋给myList[9]：

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

6.2.5 数组初始化语法

Java有一个简捷的记法，称作*数组初始化语法*（array initializer），它使用下面的语法将声明数组、创建数组和初始化数组结合到一条语句中：

```
elementType[] arrayRefVar = {value0, value1, ..., valuek};
```

（元素类型[]数组引用变量={值0，值1，…，值k};）

例如：

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

这条语句声明、创建并初始化包含4个元素的数组myList，它等价于下列语句：

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

警告　数组初始化语法中不使用运算符new。使用数组初始化语法时，必须将声明、创建和初始化数组都放在一条语句中。将它们分开会产生语法错误。因此，下面的语句是错误的：

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

6.2.6 处理数组

处理数组元素时，经常会用到for循环，理由有以下两点：

1）数组中所有元素都是同一类型的。可以使用循环以同样的方式反复处理这些元素。

2）由于数组的大小是已知的，所以很自然地就使用for循环。

假设创建如下数组：

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

下面是一些处理数组的例子：

1）（使用输入值初始化数组）下面的循环使用用户输入的数值初始化数组myList。

```
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）（使用随机数初始化数组）下面的循环使用0.0到100.0之间，但小于100.0的随机值初始化数组myList。

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

3）（显示数组）为了打印数组，必须使用类似下面的循环，打印数组中的每一个元素。

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

提示　对于char[]类型的数组，可以使用一条打印语句打印。例如：下面的代码显示Dallas：

```
char[] city = {'D', 'a', 'l', 'l', 'a', 's'};
System.out.println(city);
```

4）（对所有元素求和）使用名为total的变量存储和。total的值初始化为0。使用如下循环将数组中的每个元素加到total中：

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

5）（找出最大元素）使用名为max的变量存储最大元素。将max的值初始化为myList[0]。为了找出数组myList中的最大元素，将每个元素与max比较，如果该元素大于max，则更新max。

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

6）（找出最大元素的最小下标值）经常需要找出数组中的最大元素。如果数组中含有多个最大元素，那么找出最大元素的最小下标值。假设数组myList为{1，5，3，4，5，5}。最大元素为5，5的最小下标为1。使用名为max的变量存储最大元素，使用名为indexOfMax的变量表示最大元素的下标。将max的值初始化为myList[0]，而将indexOfMax的值初始化为0。将myList中的每个元素与max比较，如果这个元素大于max，则更新max和indexOfMax。

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

如果用（myList[i]>=max）替换（myList[i]>max），那么结果会是什么？

7）（随意打乱）在很多应用程序中，需要对数组中的元素进行任意的重新排序。这称作打乱（shuffling）。为完成这种功能，针对每个元素myList[i]，随意产生一个下标j，然后将myList[i]和myList[j]互换，如下所示：

```
for (int i = 0; i < myList.length; i++) {
  // Generate an index j randomly
  int index = (int) (Math.random()
    * mylist.length);

  // Swap myList[i] with myList[j]
  double temp = myList[i];
  myList[i] = myList[index]
  myList[index] = temp;
}
```

8）（移动元素）有时候，需要向左或向右移动元素。这里的例子就是将元素向左移动一个位置并且将第一个元素放在最后一个元素的位置：

```
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;
```

6.2.7 for-each循环

Java支持一个简便的for循环，称之为for-each循环（for-each loop）或增强型for循环（enhanced for

loop)，不使用下标变量就可以顺序地遍历整个数组。例如，下面的代码就可以显示数组myList的所有元素：

```
for (double u: myList) {
  System.out.println(u);
}
```

此代码可以读作“对myList中每个元素u进行以下操作”。注意，变量u必须声明为与myList中元素相同的数据类型。

通常，for-each循环的语法为：

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

但是，当需要以其他顺序遍历数组或改变数组中的元素时，还是必须使用下标变量。

警告 越界访问数组是经常会出现的程序设计错误，它会抛出一个运行错误ArrayIndexOutOfBoundsException。为了避免错误的发生，在使用时应确保所使用的下标不超过arrayRefVar.length-1。

程序员经常错误地使用下标1引用数组的第一个元素，但其实第一个元素的下标应该是0。这称为下标过1错误（off-by-one error）。它是在循环中该使用<的地方误用<=时会犯的错误。例如，下面的循环是错误的：

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

应该用<替换<=。

6.3 问题：乐透号码

选10乐透的每张筹码都有10个取值范围在1到99的独特数字。假设你买了很多筹码，希望它们涵盖从1到99的所有数字。编写一个程序，从文件中读取筹码上的数字，并且检查是否涵盖所有的数。将文件中最后一个数设定为0。假设文件包含如下数字：

```
80 3 87 62 30 90 10 21 46 27
12 40 83 9 39 88 95 59 20 37
80 40 87 67 31 90 11 24 56 77
11 48 51 42 8 74 1 41 36 53
52 82 16 72 19 70 44 56 29 33
54 64 99 14 23 22 94 79 55 2
60 86 34 4 31 63 84 89 7 78
43 93 97 45 25 38 28 26 85 49
47 65 57 67 73 69 32 71 24 66
92 98 96 77 6 75 17 61 58 13
35 81 18 15 5 68 91 50 76
0
```

程序应该显示：

```
The tickets cover all numbers
```

假设文件包含如下数字：

```
11 48 51 42 8 74 1 41 36 53
52 82 16 72 19 70 44 56 29 33
0
```

程序应该显示：

```
The tickets don't cover all numbers
```

该如何标定一个数字被涵盖呢？可以创建一个由99个布尔元素构成的数组。每个数组中的元素都可以用来标定是否涵盖了一个数字。假定这个数组是isCovered。初始状态时，每个元素都是false，如

图6-2a所示。当读取一个数时，它对应的元素设置为true。假如输入的数字是1、2、3、99、0。当读取数字1时，isCovered[0]就设置为true（参见图6-2b）。当读取数字2时，isCovered[2-1]就设置为true（参见图6-2c）。当读取数字3时，isCovered[3-1]就设置为true（参见图6-2d）。当读取数字99时，isCovered[99-1]就设置为true（参见图6-2e）。

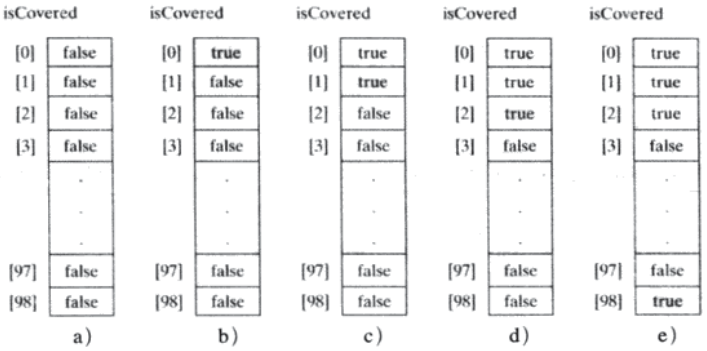


图6-2 如果在乐透筹码中出现数字i，就将isCovered[i-1]设置为true

这个程序的算法可以描述为如下所示：

```
对于从文件中读出的每个数字k，
    通过将isCovered[k-1]设置为true将数字k标记为涵盖的；
if 每一个isCovered[i]都为true
    那么该筹码涵盖所有的数字
else
    该筹码并未涵盖所有数字
```

程序清单6-1给出完整的程序。

程序清单6-1 LottoNumbers.java

```
1 import java.util.Scanner;
2
3 public class LottoNumbers {
4   public static void main(String args[]) {
5     Scanner input = new Scanner(System.in);
6     boolean[] isCovered = new boolean[99]; // Default is false
7
8     // Read each number and mark its corresponding element covered
9     int number = input.nextInt();
10    while (number != 0) {
11      isCovered[number - 1] = true;
12      number = input.nextInt();
13    }
14
15    // Check whether all covered
16    boolean allCovered = true; // Assume all covered initially
17    for (int i = 0; i < 99; i++)
18      if (!isCovered[i]) {
19        allCovered = false; // Find one number not covered
20        break;
21      }
22
23    // Display result
24    if (allCovered)
25      System.out.println("The tickets cover all numbers");
26    else
27      System.out.println("The tickets don't cover all numbers");
28  }
29 }
```

假设已经创建了一个名为LottoNumbers.txt的文本文件，该文件包括输入数值2 5 6 5 4 3 23 43 2 0。可以使用下面的命令运行该程序：

```
java LottoNumbers < LottoNumbers.txt
```

可以如下跟踪这个程序：

这个程序创建了一个由99个布尔元素构成的数组，并且将每个元素初始化为false（第6行）。它从文件中读取第一个数字（第9行）。然后程序在循环中重复下面的操作：

- 如果数字非0，就将数组isCovered中的对应值设置为true（第11行）。
- 读取下一个数字（第12行）。

line	Representative elements in array isCovered							number	allCovered
	[1]	[2]	[3]	[4]	[5]	[22]	[42]		
6	false	false	false	false	false	false	false		
9								2	
11	true								
12								5	
11				true					
12								6	
11					true				
12								5	
11				true					
12								4	
11			true						
12								3	
11		true							
12								23	
11						true			
12								43	
11							true		
12								2	
11	true								
12								0	
16									true
18(i=0)									false

当输入为0时，输入结束。程序第16～21行检测它是否涵盖所有数字，第24～27行显示结果。

6.4 问题：一副牌

这里的问题是要编写一个程序，从一副52张的牌中随机挑出4张牌。所有的牌可以用一个名为deck的数组表示，这个数组用从0到51的初始值来填充，如下所示：

```
int[] deck = new int[52];

// Initialize cards
for (int i = 0; i < deck.length; i++)
  deck[i] = i;
```

牌号从0到12、13到25、26到38以及39到51分别表示13张黑桃、13张红桃、13张方块、13张梅花，

如图6-3所示。在打乱数组deck之后，从deck中选出前四张牌。cardNumber/13决定牌的花色而cardNumver%13决定是具体花色中的哪张牌。

程序清单6-2给出这个问题的解决方案。

程序清单6-2 DeckOfCards.java

```
1 public class DeckOfCards {
2   public static void main(String[] args) {
3     int[] deck = new int[52];
4     String[] suits = {"Spades", "Hearts", "Diamonds", "Clubs"};
5     String[] ranks = {"Ace", "2", "3", "4", "5", "6", "7", "8", "9",
6       "10", "Jack", "Queen", "King"};
7
8     // Initialize cards
9     for (int i = 0; i < deck.length; i++)
10      deck[i] = i;
11
12    // Shuffle the cards
13    for (int i = 0; i < deck.length; i++) {
14      // Generate an index randomly
15      int index = (int)(Math.random() * deck.length);
16      int temp = deck[i];
17      deck[i] = deck[index];
18      deck[index] = temp;
19    }
20
21    // Display the first four cards
22    for (int i = 0; i < 4; i++) {
23      String suit = suits[deck[i] / 13];
24      String rank = ranks[deck[i] % 13];
25      System.out.println("Card number " + deck[i] + ": "
26        + rank + " of " + suit);
27    }
28  }
29 }
```

```
Card number 6: 7 of Spades
Card number 48: 10 of Clubs
Card number 11: Queen of Spades
Card number 24: Queen of Hearts
```

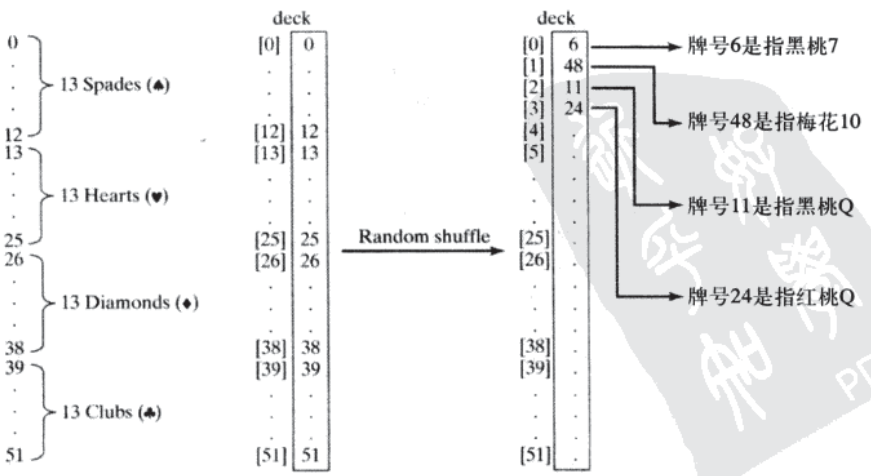


图6-3 52张牌存储在一个名为deck的数组中

程序为四种花色定义了一个数组suits（第4行），而为一个花色中的13张牌定义一个数组ranks（第5～6行）。这些数组中的每个元素都是一个字符串。

在第9～10行用0到51初始化deck。deck的值为0表示黑桃A，1表示黑桃2，13表示红桃A，14表示红桃2。

第13～19行随意地打乱这副牌。在牌被打乱后，deck[i]中放的是一个任意的值。deck[i]/13的值为0、1、2或3，该值确定这张牌是哪种花色（第23行）。deck[i]%13的值在0到12之间，该值确定这张牌是花色中的哪张牌（第24行）。

6.5 数组的复制

在程序中，经常需要复制一个数组或数组的一部分。这种情况下，可能会尝试使用赋值语句（=），如下所示：

```
list2 = list1;
```

该语句并不能将list1引用的数组内容复制给list2，而只是将list1的引用值复制给了list2。在这条语句之后，list1和list2都指向同一个数组，如图6-4所示。list2原先所引用的数组不能再引用，它就变成了垃圾，会被Java虚拟机自动收回。

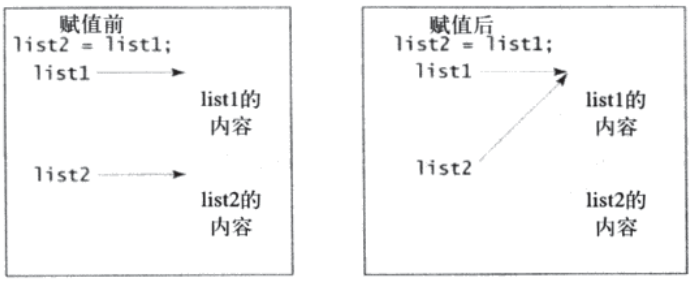


图6-4　赋值语句执行前，list1和list2指向各自的内存地址。
在赋值之后，数组list1的引用被传递给list2

在Java中，可以使用赋值语句复制基本数据类型的变量，但不能复制数组。将一个数组变量赋值给另一个数组变量，实际上是将一个数组的引用复制给另一个变量，使两个变量都指向相同的内存地址。

复制数组有三种方法：

1）使用循环语句逐个地复制数组的元素。

2）使用System类中的静态方法arraycopy。

3）使用clone方法复制数组，这将在第14章中介绍。

可以使用循环将源数组中的每个元素复制到目标数组中的对应元素。例如，下述代码使用for循环将sourceArray复制到targetArray：

```
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];
}
```

另一种方式是使用java.lang.System类的arraycopy方法复制数组，而不是使用循环。arraycopy的语法如下所示：

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

其中，参数src_pos和tar_pos分别表示在源数组sourceArray和目标数组targetArray中的起始位置。从sourceArray复制到targetArray中的元素个数由参数length指定。例如，可以使用下面的语句改写上述循环：

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

arraycopy方法没有给目标数组分配内存空间。复制前必须创建目标数组以及分配给它的内存空间。复制完成后，sourceArray和targetArray具有相同的内容，但占有独立的内存空间。

注意 arraycopy方法违反了Java命名习惯。根据命名习惯，该方法应该命名为arrayCopy（即字母C大写）。

6.6 给方法传递数组

正如前面给方法传递基本数据类型的值一样，也可以给方法传递数组。例如，下面的方法显示int型数组的元素：

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

可以通过传递一个数组调用上面的方法。例如，下面的语句调用printArray方法显示3、1、2、6、4和2：

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

注意 前面的语句使用下述语法创建数组：

```
new elementType[]{value0, value1, ..., valuek}; (new数据类型[]{值0,值1,…,值接量k};)
```

该数组没有显式地引用变量，这样的数组称为匿名数组（anonymous array）。

Java使用值传递（pass by value）的方式将实参传递给方法。传递基本数据类型变量的值与传递数组值会有很大的不同。

- 对于基本数据类型参数，传递的是实参的值。
- 对于数组类型参数，参数值是数组的引用，给方法传递的是这个引用。从语义上来讲，最好的描述就是参数传递的共享信息（pass-by-sharing），即方法中的数组和传递的数组是一样的。所以，如果改变方法中的数组，将会看到方法外的数组也变化了。

例如，采用下面的代码：

```
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]
  }
}
```

```
x is 1
y[0] is 5555
```

将会看到，在调用m之后，x仍然是1，但是y[0]却变成了5555。这是因为尽管y和numbers是两个独立的变量，但它们指向同一数组，如图6-5所示。当调用m(x,y)时，x和y的值传递给number和numbers。因为y包含数组的引用值，所以，numbers现在包含的是指向同一数组的相同引用值。

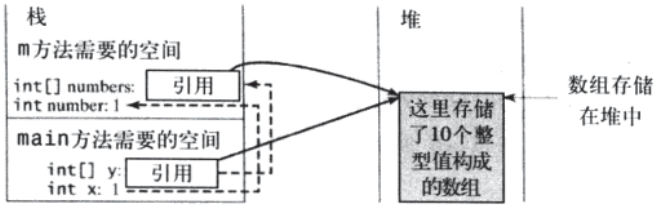


图6-5 x中的基本数据类型值被传递给number，而y中的引用值被传递给numbers

注意 JVM将数组存储在一个称作堆（heap）的内存区域中，堆用于动态内存分配，在堆中内存块可以按随意的顺序分配和释放。

传递数组参数

程序清单6-3给出另外一个例子，说明传递基本数据类型值与传递数组引用变量给方法的不同之处。

程序包含两个交换数组中元素的方法。第一个方法名为swap，它没能将两个整型参数对换。第二个方法名为swapFirstTwoInArray，它成功地将数组参数中前两个元素进行互换。

程序清单6-3 TestPassArray.java

```
1 public class TestPassArray {
2   /** Main method */
3   public static void main(String[] args) {
4     int[] a = {1, 2};
5
6     // Swap elements using the swap method
7     System.out.println("Before invoking swap");
8     System.out.println("array is {" + a[0] + ", " + a[1] + "}");
9     swap(a[0], a[1]);
10    System.out.println("After invoking swap");
11    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
12
13    // Swap elements using the swapFirstTwoInArray method
14    System.out.println("Before invoking swapFirstTwoInArray");
15    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
16    swapFirstTwoInArray(a);
17    System.out.println("After invoking swapFirstTwoInArray");
18    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
19  }
20
21  /** Swap two variables */
22  public static void swap(int n1, int n2) {
23    int temp = n1;
24    n1 = n2;
25    n2 = temp;
26  }
27
28  /** Swap the first two elements in the array */
29  public static void swapFirstTwoInArray(int[] array) {
30    int temp = array[0];
31    array[0] = array[1];
32    array[1] = temp;
33  }
34 }
```

```
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}
```

如图6-6所示，使用swap方法没能对换两个元素。但是，使用swapFirstTwoInArray方法就实现了对换。因为swap方法中的参数为基本数据类型，所以调用swap(a[0],a[1])时，a[0]和a[1]的值传给了方法内部的n1和n2。n1和n2的内存位置独立于a[0]和a[1]的内存位置。这个方法的调用没有影响数组的内容。

swapFirstTwoInArray方法的参数是一个数组。如图6-6所示，数组的引用传给方法。这样，变量a（方法外）和array（方法内）都指向在同一内存位置中的同一个数组。因此，在方法swapFirstTwoInArray内交换array[0]与array[1]和在方法外交换a[0]与a[1]是一样的。

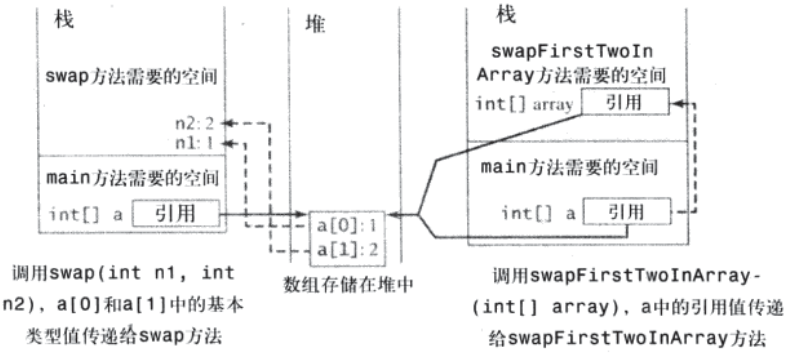
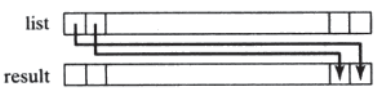


图6-6 将数组传给方法时，传给方法的是数组的引用

6.7 从方法中返回数组

可以在调用方法时向方法传递一个数组。方法也可以返回一个数组。例如，下面的方法返回一个与输入数组顺序相反的数组：

```
1 public static int[] reverse(int[] list) {
2   int[] result = new int[list.length];
3
4   for (int i = 0, j = result.length - 1;
5        i < list.length; i++, j--) {
6     result[j] = list[i];
7   }
8
9   return result;
10 }
```



第2行创建了一个新数组result，第4～7行把数组list的元素复制到数组result中。第9行返回数组。例如，下面的语句返回元素为6、5、4、3、2、1的新数组list2。

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

案例研究：统计每个字母出现的次数

程序清单6-4中给出一个程序，统计字符数组中每个字母出现的次数。该程序完成下述任务：

1）随机生成100个小写字母，并将其放入一个字符数组中，如图6-7a所示。可以使用程序清单5-10中RandomCharacter类中的getRandomLowerCaseLetter()方法获取一个随机字母。

2）对数组中每个字母出现的次数进行计数。为了完成这个功能，创建一个具有26个int值的数组counts，每个值存放每个字母出现的次数，如图6-7b所示。也就是说，counts[0]记录a出现的次数，counts[1]记录b出现的次数，依此类推。

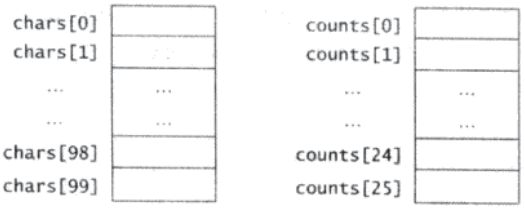


图6-7 数组char存储100个字符，数组counts存储26个计数器，每个计数器对一个字母进行计数

程序清单6-4 CountLettersInArray.java

```
1  public class CountLettersInArray {
2    /** Main method */
3    public static void main(String[] args) {
4      // Declare and create an array
5      char[] chars = createArray();
6
7      // Display the array
8      System.out.println("The lowercase letters are:");
9      displayArray(chars);
10
11     // Count the occurrences of each letter
12     int[] counts = countLetters(chars);
13
14     // Display counts
15     System.out.println();
16     System.out.println("The occurrences of each letter are:");
17     displayCounts(counts);
18   }
19
20   /** Create an array of characters */
21   public static char[] createArray() {
22     // Declare an array of characters and create it
23     char[] chars = new char[100];
24
25     // Create lowercase letters randomly and assign
26     // them to the array
27     for (int i = 0; i < chars.length; i++)
28       chars[i] = RandomCharacter.getRandomLowerCaseLetter();
29
30     // Return the array
31     return chars;
32   }
33
34   /** Display the array of characters */
35   public static void displayArray(char[] chars) {
36     // Display the characters in the array 20 on each line
37     for (int i = 0; i < chars.length; i++) {
38       if ((i + 1) % 20 == 0)
39         System.out.println(chars[i]);
40       else
41         System.out.print(chars[i] + " ");
42     }
43   }
44
45   /** Count the occurrences of each letter */
46   public static int[] countLetters(char[] chars) {
47     // Declare and create an array of 26 int
48     int[] counts = new int[26];
49
50     // For each lowercase letter in the array, count it
51     for (int i = 0; i < chars.length; i++)
```

```
52        counts[chars[i] - 'a']++;
53
54      return counts;
55    }
56
57    /** Display counts */
58    public static void displayCounts(int[] counts) {
59      for (int i = 0; i < counts.length; i++) {
60        if ((i + 1) % 10 == 0)
61          System.out.println(counts[i] + " " + (char)(i + 'a'));
62        else
63          System.out.print(counts[i] + " " + (char)(i + 'a') + " ");
64      }
65    }
66 }
```

```
The lowercase letters are:
e y l s r i b k j v j h a b z n w b t v
s c c k r d w a m p w v u n q a m p l o
a z g d e g f i n d x m z o u l o z j v
h w i w n t g x w c d o t x h y v z y z
q e a m f w p g u q t r e n n w f c r f

The occurrences of each letter are:
5 a 3 b 4 c 4 d 4 e 4 f 4 g 3 h 3 i 3 j
2 k 3 l 4 m 6 n 4 o 3 p 3 q 4 r 2 s 4 t
3 u 5 v 8 w 3 x 3 y 6 z
```

方法createArray（第21～32行）生成一个存放100个随机小写字母的数组。第5行调用该方法，并且将这个数组赋值给chars。如果将代码改写成如下形式，会出现什么错误？

```
char[] chars = new char[100];
chars = createArray();
```

将会创建两个数组。第一行使用new char[100]创建一个数组。第二行通过调用createArray()创建一个数组，并将这个数组的引用赋值给chars。第一行生成的数组就变成了“垃圾”，因为它不能再被引用。Java在后台自动回收“垃圾”。程序可以正常地编译和运行，但它会创建一个不必要的数组。

调用getRandomLowerCaseLetter()（第28行）返回一个随机小写字母。该方法是在程序清单5-10的RandomCharacter类中定义的。

countLetters方法（第46～55行）返回一个含26个int型值的数组，每个整数存放的是每个字母出现的次数。该方法处理数组中的每个字母，并将它对应的计数器加1。统计字母出现次数的穷举方法可能会如下所示：

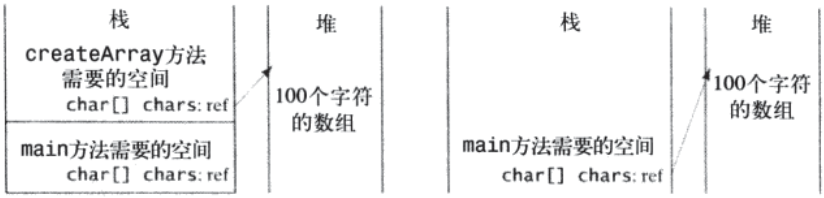
```
for (int i = 0; i < chars.length; i++)
  if (chars[i] == 'a')
    counts[0]++;
  else if (chars[i] == 'b')
    counts[1]++;
  ...
```

但是第51～52行给出一个更好的解决方案。

```
for (int i = 0; i < chars.length; i++)
  counts[chars[i] - 'a']++;
```

如果字母（chars[i]）是'a'，那么它对应的计数器就是counts['a'-'a']（即counts[0]）。如果字母是'b'，因为'b'的统一码比'a'的统一码大1，所以它对应的计数器为counts['b'-'a']（即counts[1]）。如果字母是'z'，因为'z'的统一码比'a'的大25，所以它对应的计数器为counts['z'-'a']（即counts[25]）。

图6-8显示在执行createArray方法的过程中和执行之后调用栈和堆的情况。参见复习题6.14会显示程序中其他方法对栈和堆的调用。



a) 执行第5行的createArray方法期间　　b) 退出第5行的createArray方法之后

图6-8　a）执行createArray方法创建100个字符的数组；
b）在main方法中返回这个数组并赋值给变量chars

6.8　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。方法中的参数声明如下：

typeName...parameterName（类型名...参数名）

在方法声明中，指定类型后紧跟着省略号（...）。只能给方法中指定一个可变长参数，同时该参数必须是最后一个参数。任何常规参数必须在它之前。

Java将可变长参数当成数组对待。可以将一个数组或可变的参数个数传递给可变长参数。当用可变的参数个数调用方法时，Java会创建一个数组并把参数传给它。程序清单6-5包括了打印出个数不定的数列中最大值的方法。

程序清单6-5　VarArgsDemo.java

```
1 public class VarArgsDemo {
2   public static void main(String[] args) {
3     printMax(34, 3, 3, 2, 56.5);
4     printMax(new double[]{1, 2, 3});
5   }
6
7   public static void printMax(double... numbers) {
8     if (numbers.length == 0) {
9       System.out.println("No argument passed");
10      return;
11    }
12
13    double result = numbers[0];
14
15    for (int i = 1; i < numbers.length; i++)
16      if (numbers[i] > result)
17        result = numbers[i];
18
19    System.out.println("The max value is " + result);
20  }
21 }
```

第3行将一个可变长参数列表传给数组numbers来调用printMax方法。如果没有传入参数，数组的长度为0（第8行）。

第4行调用一个数组调用printMax方法。

6.9　数组的查找

查找（searching）是在数组中寻找特定元素的过程，例如：判断某一特定分数是否包括在成绩列表

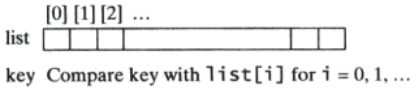
中。查找是计算机程序设计中经常要完成的任务。有很多用于查找的算法和数据结构。本节讨论两种经常使用的方法：线性查找（linear searching）和二分查找（binary searching）。

6.9.1 线性查找法

线性查找法将要查找的关键字key与数组中的元素逐个进行比较。这个过程持续到在列表中找到与关键字匹配的元素，或者查完列表也没有找到关键字为止。如果匹配成功，线性查找法返回与关键字匹配的元素在数组中的下标。如果没有匹配成功，则返回-1。程序清单6-6中的linearSearch方法给出解决方案：

程序清单6-6 LinearSearch.java

```
1 public class LinearSearch {
2   /** The method for finding a key in the list */
3   public static int linearSearch(int[] list, int key) {
4     for (int i = 0; i < list.length; i++) {
5       if (key == list[i])
6         return i;
7     }
8     return -1;
9   }
10 }
```



为了更好地理解这个方法，用下面的语句跟踪这个方法：

```
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
```

线性查找法把关键字和数组中的每一个元素进行比较。数组中的元素可以按任意顺序排列。平均来看，如果关键字存在，那么在找到关键字之前，这种算法必须与数组中一半的元素进行比较。由于线性查找法的执行时间随着数组元素个数的增长而线性增长，所以，对于大数组而言，线性查找法的效率并不高。

6.9.2 二分查找法

二分查找法是另一种常见的对数值列表的查找方法。使用二分查找法的前提条件是数组中的元素必须已经排好序。假设数组已按升序排列。二分查找法首先将关键字与数组的中间元素进行比较。考虑下面三种情况：

- 如果关键字小于中间元素，只需要在数组的前一半元素中继续查找关键字。
- 如果关键字和中间元素相等，则匹配成功，查找结束。
- 如果关键字大于中间元素，只需要在数组的后一半元素中继续查找关键字。

显然，二分法在每次比较之后就排除掉一半的数组元素，有时候是去掉一半的元素，有时候是去掉一半加1个元素。假设数组有n个元素。为方便起见，假设n是2的幂。经过第1次比较，只剩下n/2个元素需要进一步查找；经过第2次比较，剩下（n/2）/2个元素需要进一步查找。经过k次比较之后，需要查找的元素就剩下$n/2^k$个。当$k=\log_2 n$时，数组中只剩下1个元素，就只需要再比较1次。因此，在一个已经排序的数组中用二分查找法查找一个元素，即使是最坏的情况，也需要$\log_2 n+1$次比较。对于一个有1024（2^{10}）个元素的数组，在最坏情况下，二分查找法只需要比较11次，而在最坏的情况下线性查找要比较1023次。

每次比较后，数组要查找的部分就会缩小一半。用low和high分别表示当前查找数组的第一个下标和最后一个下标。初始条件下，low为0，而high为list.length-1。让mid表示列表的中间元素的下标。这样，mid就是（low + high）/2。图6-9显示怎样使用二分法从数列{2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79}中找出关键字11。

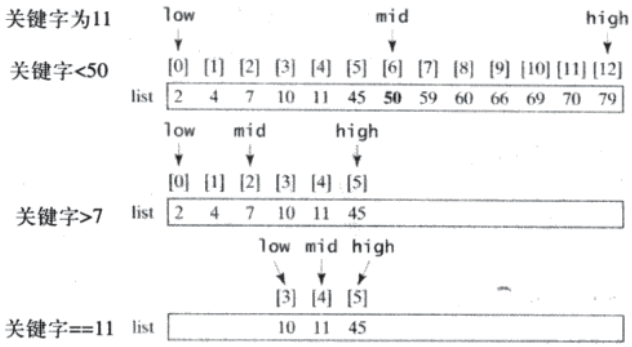


图6-9 二分查找法在每次比较后，数列中需要进一步考虑的元素减少一半

现在知道了二分查找法是如何工作的。下一个任务就是在Java中实现它。不要急于给出一个完整的实现。逐步地实现这个程序，一次一步。可以从查找的第一次迭代开始，如图6-10a所示。它将关键字key和低下标low为0、高下标high为list.length-1的数列的中间元素进行比较。如果key<list[mid]，就将下标high设置为mid-1；如果key==list[mid]，则匹配成功并返回mid；如果key>list[mid]，就将下标low设置为mid+1。

```
public static int binarySearch(
    int[] list, int key) {
  int low = 0;
  int high = list.length - 1;


    int mid = (low + high) / 2;
    if (key < list[mid])
      high = mid - 1;
    else if (key == list[mid])
      return mid;
    else
      low = mid + 1;


}
```

a) 版本1

```
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; // Not found
}
```

b) 版本2

图6-10 逐步实现二分查找法

接下来就要考虑增加一个循环，实现这个方法重复地完成查找，如图6-10b所示。如果找到这个关键字，或者当low>high时还没有找到这个关键字，就结束这个查找。

当没有找到这个关键字时，low就是一个插入点，这个位置将插入关键字以保持列表的有序性。一种更实用的方法是返回插入点减去1。这个方法必须返回一个负值，表明这个关键字不在该序列中。可以只返回-low吗？答案是：不可以。如果关键字小于list[0]，那么low就是0，-0也是0。这就表明关键字匹配list[0]。一个好的选择是，如果关键字不在该序列中，方法返回-low-1。返回-low-1不仅表明关键字不在序列中，而且还给出了关键字应该插入的地方。

完整的程序在程序清单6-7中给出。

程序清单6-7 BinarySearch.java

```
1 public class BinarySearch {
2   /** Use binary search to find the key in the list */
3   public static int binarySearch(int[] list, int key) {
4     int low = 0;
5     int high = list.length - 1;
6
7     while (high >= low) {
8       int mid = (low + high) / 2;
```

```
 9       if (key < list[mid])
10         high = mid - 1;
11       else if (key == list[mid])
12         return mid;
13       else
14         low = mid + 1;
15     }
16
17     return -low - 1; // Now high < low, key not found
18   }
19 }
```

如果关键字包含在列表中，二分查找法就返回查找的关键字的下标（第12行）；否则，返回-1 -low（第17行）。

如果用（high>low）替换第7行的（high>=low），会出现什么现象呢？这个查找也许会漏掉可能的匹配元素。假如数列只有一个元素，这个查找就会漏掉这个元素。

如果数列中有重复的元素，这个方法还能使用吗？回答是肯定的，只要数列中的元素是按递增顺序排列的。如果查找的元素在数列中，那么该方法就返回匹配元素中的一个下标。

为了更好地理解这个方法，使用下面的语句跟踪这个方法，当方法返回时确定low和high。

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
int i = BinarySearch.binarySearch(list, 2); // Returns 0
int j = BinarySearch.binarySearch(list, 11); // Returns 4
int k = BinarySearch.binarySearch(list, 12); // Returns -6
int l = BinarySearch.binarySearch(list, 1); // Returns -1
int m = BinarySearch.binarySearch(list, 3); // Returns -2
```

下面的表格列出当方法退出时和值从调用方法返回时，low和high的值。

方法	Low	High	返回值
binarySearch(list,2)	0	1	0
binarySearch(list,11)	3	5	4
binarySearch(list,12)	5	4	−6
binarySearch(list,1)	0	−1	−1
binarySearch(list,3)	1	0	−2

注意 线性查找法适用于在小数组或没有排序的数组中查找，但是对大数组而言效率不高。二分查找法的效率较高，但它要求数组已经排好序。

6.10 数组的排序

像查找一样，排序是计算机程序设计中的一个常用任务。已经开发出了很多不同的排序算法。本节将介绍两种简单、直观的排序算法：选择排序法（selection sort）和插入排序法（insertion sort）。

6.10.1 选择排序

假设要按升序排列一个数列。选择排序法先找到数列中最小的数，然后将它放在数列的最前面。接下来，在剩下的数中找到最小数，将它放到第一个数的后面，依此类推，直到数列中仅剩一个数为止。图6-11显示如何使用选择排序法对数列{2，9，5，4，8，1，6}进行排序。

已经知道了选择排序法是如何工作的。现在的任务是用Java语言实现它。对初学者来说，很难在第一次尝试时就开发出完整的解决方案。开始编写第一次迭代的代码，找出数列中的最大数，将其与最后一个元素互换，然后观察第二次迭代与第一次的不同之处，接着是第三次，依此类推。通过这样的观察可以写出推广到所有迭代的循环。

解决方案如下所述：

```
for (int i = 0; i < list.length - 1; i++) {
  select the smallest element in list[i..list.length-1];
  swap the smallest with list[i], if necessary;
    // list[i] is in its correct position.
    // The next iteration apply on list[i+1..list.length-1]
}
```

程序清单6-8实现这个解决方案。

程序清单6-8　SelectionSort.java

```
1 public class SelectionSort {
2   /** The method for sorting the numbers */
3   public static void selectionSort(double[] list) {
4     for (int i = 0; i < list.length - 1; i++) {
5       // Find the minimum in the list[i..list.length-1]
6       double currentMin = list[i];
7       int currentMinIndex = i;
8
9       for (int j = i + 1; j < list.length; j++) {
10        if (currentMin > list[j]) {
11          currentMin = list[j];
12          currentMinIndex = j;
13        }
14      }
15
16      // Swap list[i] with list[currentMinIndex] if necessary;
17      if (currentMinIndex != i) {
18        list[currentMinIndex] = list[i];
19        list[i] = currentMin;
20      }
21    }
22  }
23 }
```

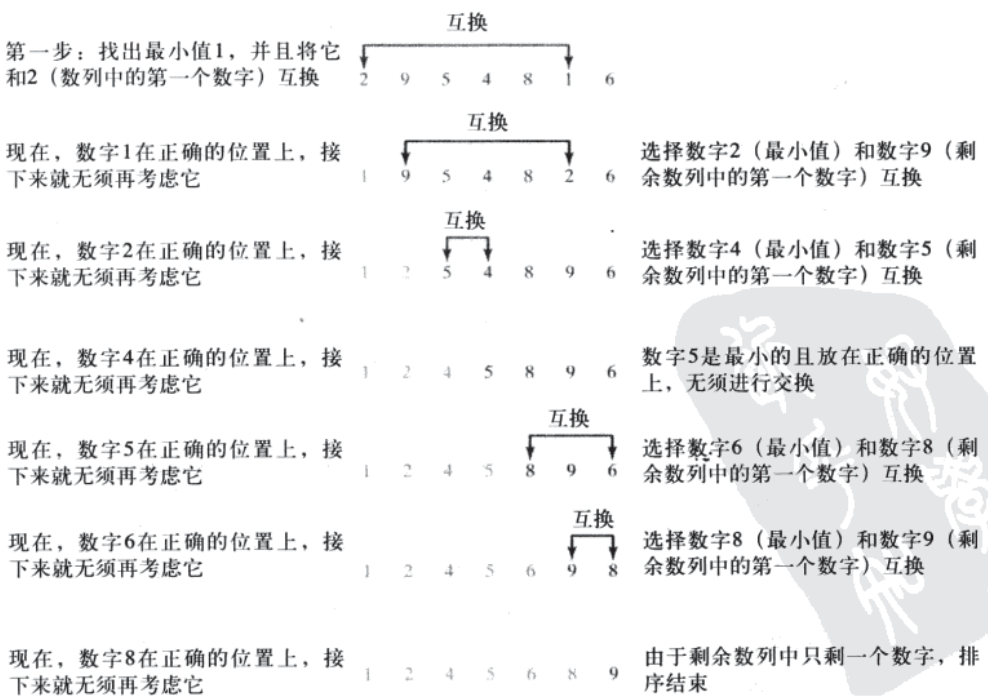


图6-11　选择排序重复选择数列中的最小数，然后将它和数列中的第一个数字互换

方法selectionSort (double[]list) 可以对任意一个double型元素数组进行排序。这个方法用嵌套的for循环实现。外层循环（循环控制变量i）（第4行）迭代执行以寻找从list[1]到list[list.length-1]的列表中最小的元素，然后将它和list[i]互换。

变量i的初值是0。在外层循环的每次迭代之后，list[i]都被放到正确的位置。最后，所有的元素都被放到正确的位置，因此，整个数列也就排好序了。

为了更好地理解这个方法，用下面的语句跟踪该方法：

```
double[] list = {1, 9, 4.5, 6.6, 5.7, -4.5};
SelectionSort.selectionSort(list);
```

6.10.2 插入排序

假设希望对一个数列进行递增排序。插入排序法的算法是在已排好序的子数列中反复插入一个新元素来对数列值进行排序的，直到整个数列全部排好序。图6-12描述如何用插入排序法对数列{2，9，5，4，8，1，6}进行排序。

这个算法可以描述如下：

```
for (int i=1; i<list.length;  i++) {
将list[i]插入已排好序的子数列中，这样list[0..i]也是排好序的。
}
```

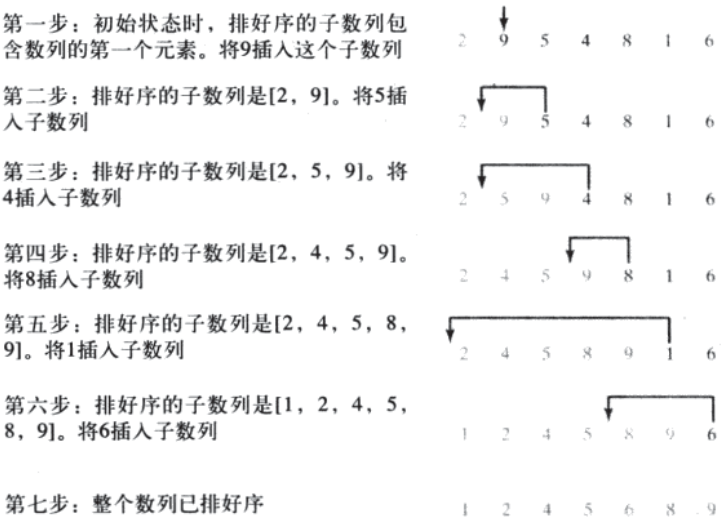


图6-12 插入排序将新元素重复插入已排好序的子数列中

为了将list[i]插入list[0..i-1]，需要将list[i]存储在一个名为currentElement的临时变量中。如果list[i-1]>currentElement，就将list[i-1]移到list[i]；如果list[i-2]>currentElement，就将list[i-2]移到list[i-1]，依此类推，直到list[i-k]<=currentElement或者k>i（传递的是排好序的数列的第一个元素）。将currentElement赋值给list[i-k+1]。例如：为了在图6-13的步骤3中将4插入{2，5，9}中，由于9>4，所以把list[2]（9）移到list[3]，又因为5>4，所以把list[1]（5）移到list[2]。最后，把currentElement（4）移到list[1]。

算法可以扩展和执行，如程序清单6-9所示。

程序清单6-9 InsertionSort.java

```
1 public class InsertionSort {
2   /** The method for sorting the numbers */
3   public static void insertionSort(double[] list) {
```

```
4     for (int i = 1; i < list.length; i++) {
5       /** insert list[i] into a sorted sublist list[0..i-1] so that
6           list[0..i] is sorted. */
7       double currentElement = list[i];
8       int k;
9       for (k = i - 1; k >= 0 && list[k] > currentElement; k--) {
10        list[k + 1] = list[k];
11      }
12
13      // Insert the current element into list[k + 1]
14      list[k + 1] = currentElement;
15    }
16  }
17 }
```

图6-13 新元素插入到已排好序的子数列中

`insertionSort(double[] list)`方法是对任意一个`double`类型元素构成的数组进行排序。该方法是用嵌套的`for`循环实现的。外层循环（循环控制变量`i`）（第4行）的迭代是为了获取已排好序的子数列，其范围从`list[0]`到`list[i]`。内层循环（循环控制变量`k`）将`list[i]`插入到从`list[0]`到`list[i-1]`的子数列中。

为了更好地理解这个方法，使用下面的语句跟踪这个方法：

```
double[] list = {1, 9, 4.5, 6.6, 5.7, -4.5};
InsertionSort.insertionSort(list);
```

6.11 Arrays类

为实现数组的排序和查找、数组的比较和对数组填充元素，`java.util.Arrays`类包括各种各样的静态方法。这些方法都有对所有基本类型的重载方法。

可以使用`sort`方法对整个数组或部分数组进行排序。例如，下面的代码对数值型数组和字符型数组进行排序。

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

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars, 1, 3); // Sort part of the array
```

调用`sort(numbers)`对整个数组`numbers`排序。调用`sort(chars,1,3)`对从`chars[1]`到`chars[3-1]`的部分数组排序。

可以采用二分查找法（`binarySearch`方法）在数组中查找关键字。数组必须提前按增序排列好。如果数组中不存在关键字，方法返回-（插入点下标+1）。例如，下面的代码在整数数组和字符数组中查找关键字：

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("(1) Index is " +
```

```
    java.util.Arrays.binarySearch(list, 11));
System.out.println("(2) Index is " +
    java.util.Arrays.binarySearch(list, 12));

char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("(3) Index is " +
    java.util.Arrays.binarySearch(chars, 'a'));
System.out.println("(4) Index is " +
    java.util.Arrays.binarySearch(chars, 't'));
```

前面代码的输出为：

(1) Index is 4
(2) Index is -6
(3) Index is 0
(4) Index is -4

可以采用equals方法检测两个数组是否相等。如果它们的内容相同，那么这两个数组相等。在下面的代码中，list1和list2相等，而list2和list3不相等。

```
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 10};
int[] list3 = {4, 2, 7, 10};
System.out.println(java.util.Arrays.equals(list1, list2)); // true
System.out.println(java.util.Arrays.equals(list2, list3)); // false
```

可以使用fill方法填充整个数组或部分数组。例如：下列代码将5填充到list1中，将8填充到元素list2[1]和list2[3-1]中。

```
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 10};
java.util.Arrays.fill(list1, 5); // Fill 5 to the whole array
java.util.Arrays.fill(list2, 1, 3, 8); // Fill 8 to a partial array
```

关键术语

anonymous array（匿名数组）
array（数组）
array initializer（数组初始化语法）
binary search（二分查找）
garbage collection（垃圾回收）
index（下标）
indexed variable（下标变量）
insertion sort（插入排序）
linear search（线性查找）
selection sort（选择排序）

本章小结

- 使用语法elementType[] arrayRefVar（元素类型[] 数组引用变量）或elementType arrayRefVar[]（元素类型 数组引用变量[]）声明一个数组类型的变量。尽管elementType[] arrayRefVar也是合法的，但还是推荐使用elementType[] arrayRefVar风格。
- 不同于基本数据类型变量的声明，声明数组变量并不会给数组分配任何空间。数组变量不是基本数据类型变量。数组变量包含的是对数组的引用。
- 只有创建数组后才能给数组元素赋值。可以使用new操作符创建数组，语法如下：new elementType[arraySize]（数据类型[数组大小]）。
- 数组中的每个元素都是使用语法arrayRefVar[index]（数组引用变量[下标]）表示的。下标必须是一个整数或一个整数表达式。
- 创建数组之后，它的大小就不能改变，使用arrayRefVar.length就可以得到数组的大小。由于数组的下标总是从0开始，所以，最后一个下标总是arrayRefVar.length-1。如果试图引用数组界外的元素，就会发生越界错误。

- 程序员经常会错误地用下标1访问数组的第一个元素，但是，实际上这个元素的下标应该是0。这个错误称为下标过1错误（index off-by-one error）。
- 当创建一个数组时，若它的元素是基本数据类型的数值，那么赋默认值0。字符类型的默认值为'\u0000'，布尔类型的默认值为false。
- Java有一个称为*数组初始化语法*（array initializer）的简捷表达式，它将数组的声明、创建和初始化合并为一条语句，其语法为：

元素类型[] 数组引用变量={value0,value1,...,valuek}

- 将数组参数传递给方法时，实际上传递的是数组的引用；更准确地说，被调用的方法可以修改调用者的原始数组的元素。

复习题

6.2节

6.1 如何声明和创建一个数组？

6.2 如何访问数组的元素？

6.3 声明数组时给数组分配内存吗？什么时候为数组分配内存？下面代码的输出结果是什么？

```
int x = 30;
int[] numbers = new int[x];
x = 60;
System.out.println("x is " + x);
System.out.println("The size of numbers is " + numbers.length);
```

6.4 指出下列语句是对还是错：

(1) 数组中的每个元素都有相同的类型。

(2) 一旦数组被声明，大小就不能改变。

(3) 一旦数组被创建，大小就不能改变。

(4) 数组中的元素必须是基本数据类型。

6.5 下面哪些语句是合法的数组声明？

```
int i = new int(30);
double d[] = new double[30];
char[] r = new char(1..30);
int i[] = (3, 4, 3, 2);
float f[] = {2.3, 4.5, 6.6};
char[] c = new char();
```

6.6 数组下标的类型是什么？最小的下标是多少？如何表示数组名为a的第三个元素？

6.7 编写语句完成：

(1) 创建一个含10个double值的数组。

(2) 将5.5赋值给数组中最后一个元素。

(3) 显示数组前两个元素的和。

(4) 编写循环计算数组中所有元素的和。

(5) 编写循环找出数组的最小值。

(6) 随机产生一个下标，然后显示该下标所对应的数组元素。

(7) 使用数组初始化语法创建另一个初始值为3.5、5.5、4.52和5.6的数组。

6.8 当程序尝试访问下标不合法的数组元素，会发生什么？

6.9 找出并修改下面的代码：

```
1 public class Test {
2   public static void main(String[] args) {
3     double[100] r;
```

```
4
5     for (int i = 0; i < r.length(); i++);
6       r(i) = Math.random * 100;
7   }
8 }
```

6.3节

6.10 使用arraycopy()方法将下述数组复制到目标数组t：

```
int[] source = {3, 4, 5};
```

6.11 一旦数组被创建，其大小就不能再改变。下列代码可以修改数组大小吗？

```
int[] myList;
myList = new int[10];
// Some time later you want to assign a new array to myList
myList = new int[20];
```

6.4～6.7节

6.12 当给方法传递数组时，就会创建一个新数组然后传递给方法。这种说法正确吗？

6.13 给出下面两个程序的输出：

```
public class Test {
  public static void main(String[] args) {
    int number = 0;
    int[] numbers = new int[1];

    m(number, numbers);

    System.out.println("number is " + number
      + " and numbers[0] is " + numbers[0]);
  }

  public static void m(int x, int[] y) {
    x = 3;
    y[0] = 3;
  }
}
```

a)

```
public class Test {
  public static void main(String[] args) {
    int[] list = {1, 2, 3, 4, 5};
    reverse(list);
    for (int i = 0; i < list.length; i++)
      System.out.print(list[i] + " ");
  }

  public static void  reverse(int[] list) {
    int[] newList = new int[list.length];

    for (int i = 0; i < list.length; i++)
      newList[i] = list[list.length - 1 - i];

    list = newList;
  }
}
```

b)

6.14 在程序执行过程中，数组存储在哪里？写出程序清单6-4中的createArray、displayArray、countLetters和displayCounts方法在执行中和执行后栈和堆的内容。

6.8节

6.15 下面的方法声明错在哪里？

```
public static void print(String... strings, double... numbers)
public static void print(double... numbers, String name)
public static double... print(double d1, double d2)
```

6.16 能用下列语句调用程序清单6-5中的printMax方法吗？

```
printMax(1, 2, 2, 1, 4);
printMax(new double[]{1, 2, 3});
printMax(new int[]{1, 2, 3});
```

6.9～6.10节

6.17 以图6-9为例，显示如何应用二分查找法在数列{2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79}中查找关键字10和关键字12。

6.18 以图6-11为例，显示如何应用选择排序方法对{3.4, 5, 3, 3.5, 2.2, 1.9, 2}进行排序。

6.19 以图6-12为例，显示如何应用插入排序方法对{3.4, 5, 3, 3.5, 2.2, 1.9, 2}进行排序。

6.20 应该如何修改程序清单6-8中的selectionSort方法，实现数字按递减顺序排序？

6.21 应该如何修改程序清单6-9中的insertionSort方法，实现数字按递减顺序排序？

6.11节

6.22 什么类型的数组能够用java.util.Arrays.sort方法进行排序？这个sort方法会创建一个新数组吗？

6.23 为了应用java.util.Arrays.binarySearch(array,key)，数组应按升序还是降序排列？还是可以既非升序也非降序？

6.24 显示执行完每一行之后数组的内容。

```
int[] list = {2, 4, 7, 10};
java.util.Arrays.fill(list, 7);
java.util.Arrays.fill(list, 1, 3, 8);
System.out.print(java.util.Arrays.equals(list, list));
```

编程练习题

6.2节

*6.1 （指定等级）编写一个程序，读入学生成绩，获取最高分best，然后根据下面的规则赋等级值：

- 如果分数>=best−10，等级为A
- 如果分数>=best−20，等级为B
- 如果分数>=best−30，等级为C
- 如果分数>=best−40，等级为D
- 其他情况下，等级为F

程序提示用户输入学生总数，然后提示用户输入所有的分数，最后显示等级得出结论。下面是一个运行示例：

```
Enter the number of students: 4 Enter
Enter 4 scores: 40 55 70 58 Enter
Student 0 score is 40 and grade is C
Student 1 score is 55 and grade is B
Student 2 score is 70 and grade is A
Student 3 score is 58 and grade is B
```

6.2 （倒置输入的数）编写程序，读取10个整数，然后按照和读入顺序相反的顺序将它们显示出来。

**6.3 （计算数字的出现次数）编写程序，读取在1到100之间的整数，然后计算每个数出现的次数。假定输入是以0结束的。下面就是这个程序的运行示例：

```
Enter the integers between 1 and 100: 2 5 6 5 4 3 23
  43 2 0 Enter
2 occurs 2 times
3 occurs 1 time
4 occurs 1 time
5 occurs 2 times
6 occurs 1 time
23 occurs 1 time
43 occurs 1 time
```

注意 如果一个数出现的次数大于一次，就在输出时使用复数“times”。

6.4 （分析成绩）编写一个程序，读入个数不确定的考试分数，并且判断有多少个分数是大于或等于平均分，多少个分数是低于平均分的。输入一个负数表示输入的结束。假设成绩的最高分为10分。

**6.5 （打印不同的数）编写一个程序，读入10个数并且显示互不相同的数（即一个数出现多次，但仅显示一次）。提示，读入一个数，如果它是一个新数，则将它存储在数组中。如果该数已经在数组中，则忽略它。输入之后，数组包含的都是不同的数。下面是这个程序的运行示例：

```
Enter ten numbers: 1 2 3 2 1 6 3 4 5 2 Enter
The distinct numbers are: 1 2 3 6 4 5
```

*6.6 （修改程序清单4-14）程序清单4-14通过检验2，3，4，5，6，...，n/2是否是数n的因子来判断n是否是素数。如果找到一个因子，n就不是素数。判断n是否素数的另一个更有效的方法是：检验小于等于 $\sqrt{n}$ 的素数是否都能整除n。如果不能，则n就是素数。使用这个方法改写程序清单4-11以显示前50个素数。需要使用一个数组存储这些素数，然后再检查它们是否是n的可能的因子。

*6.7 （统计一位数的个数）编写一个程序，生成0和9之间的100个随机整数，然后显示每一个数出现的次数。

提示 使用(int)(Math.random()*10)产生0到9之间的随机整数。使用一个名为counts的由10个整数构成的数组存放0，1，…，9的个数。

6.4～6.7节

6.8 （求数组的平均值）编写两个重载的方法，使用下面的方法头返回一个数组的平均数：

```
public static int average(int[] array)
public static double average(double[] array)
```

编写测试程序，提示用户输入10个double型值，调用这个方法，然后显示平均值。

6.9 （找出最小元素）编写一个方法，使用下面的方法头求出一个整数数组中的最小元素：

```
public static double min(double[] array)
```

编写测试程序，提示用户输入十个数字，调用这个方法返回最小值，显示其最小值。下面是该程序的运行示例：

```
Enter ten numbers: 1.9 2.5 3.7 2 1.5 6 3 4 5 2 Enter
The minimum number is: 1.5
```

6.10 （找出最小元素的下标）编写一个方法，求出整数数组中最小元素的下标。如果这样的元素个数大于1，则返回最小的下标。使用下面的方法头：

```
public static int indexOfSmallestElement(double[] array)
```

编写测试程序，提示用户输入10个数字，调用这个方法，返回最小元素的下标，然后显示这个下标值。

*6.11 （统计学方面：计算标准差）练习题5.21计算数字的标准差。本题使用一个和它不同但等价的公式来计算n个数的标准差。

$$\text{均值} = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{x_1 + x_2 + \cdots x_n}{n} \qquad \text{标准差} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \text{均值})^2}{n-1}}$$

要用这个公式计算标准差，必须使用一个数组存储每一个数，因此，可以在获取平均值后使用它们。

程序应该包含下面的方法：

```
/** Compute the deviation of double values*/
public static double deviation(double[] x)

/** Compute the mean of an array of double values*/
public static double mean(double[] x)
```

编写测试程序，提示用户输入10个数字，然后显示平均值和标准差，如下面的运行示例所示：

```
Enter ten numbers: 1.9 2.5 3.7 2 1 6 3 4 5 2 Enter
The mean is 3.11
The standard deviation is 1.55738
```

*6.12 （倒置数组）6.7节中的reverse方法通过把数组复制到新数组中，实现数组的倒置。改写方法将传递到实参的数组倒置，然后返回这个数组。编写一个测试程序，提示用户输入十个数字，调用这个方法倒置这些数字，然后显示这些数字。

6.8节

*6.13 （随机数选择器）编写一个方法，返回1到54之间的随机数，不包括传递到实参中的数。如下指定这个方法头：

```
public static int getRandom(int... numbers)
```

6.14 （计算gcd）编写一个方法，返回个数不确定的整数的最大公约数。指定这个方法头如下所示：

```
public static int gcd(int... numbers)
```

编写测试程序，提示用户输入5个数字，调用该方法找出这些数的最大公约数，并显示这个最大公约数。

6.9～6.10节

6.15 （消除重复）使用下面的方法头编写方法，消除数组中重复出现的值：

```
public static int[] eliminateDuplicates(int[] numbers)
```

编写一个测试程序，读取10个整数，调用该方法，然后显示结果。下面是程序的运行示例：

```
Enter ten numbers: 1 2 3 2 1 6 3 4 5 2 Enter
The distinct numbers are: 1 2 3 6 4 5
```

6.16 （执行时间）编写程序，随机产生100000个整数值和一个关键字。估算一下调用程序清单6-6中的linearSearch方法的执行时间。对该数组进行排序，然后估算调用程序清单6-7中的binarySearch方法的执行时间。可以使用下面的代码模板获取执行时间：

```
long startTime = System.currentTimeMillis();
perform the task;
long endTime = System.currentTimeMillis();
long executionTime = endTime - startTime;
```

*6.17 （修改选择排序法）在6.10.1节中，使用的是选择排序法对数组排序。选择排序法重复地在当前数组中找到最小值，然后将这个最小值与该数组中的第一个数进行交换。改写这个程序，重复地在当前数组中找到最大值，然后将这个最大值与该数组中的最后一个数进行交换。编写测试程序，读取10个double型的数字，调用该方法，然后显示排好序的数字。

**6.18 （冒泡排序）使用冒泡排序算法编写一个排序方法。冒泡排序算法遍历数组几次。在每次遍历中，对相邻的两个元素进行比较。如果这一对元素是降序，则交换它们的值；否则，保持值不变。由于较小的值像气泡一样逐渐“浮向”顶部，同时较大的值“沉向”底部，所以，这种技术称为冒泡排序法（bubble sort）或下沉排序法（sinking sort）。使用{6.0, 4.4, 1.9, 2.9, 3.4, 2.9, 3.5}测试这个方法。编写一个测试程序，读取10个double型的值，调用这个方法，然后显示排好序的数字。

**6.19 （对学生排序）编写一个程序，提示用户输入学生个数、学生姓名和他们的成绩，然后按照学生成绩的降序打印学生的姓名。

***6.20 （游戏：八皇后）经典的八皇后难题是要将八个皇后放在棋盘上，任何两个皇后都不能互相攻击（即没有两个皇后是在同一行、同一列或者同一对角上）。可能的解决方案有很多。编写程序显示一个这样的解决方案。一个示例输出如下所示：

```
|Q| | | | | | | |
| | | | |Q| | | |
| | | | | | | |Q|
| | | | | |Q| | |
| | |Q| | | | | |
| | | | | | |Q| |
| |Q| | | | | | |
| | | |Q| | | | |
```

***6.21 （游戏：豆机）豆机，也称为梅花瓶或高尔顿瓶，它是一个用来做统计实验的设备，是用英国科学家瑟弗兰克斯高尔顿的名字来命名的。它是一个三角形状的均匀放置钉子（或钩子）的直立板子，如图6-14所示。

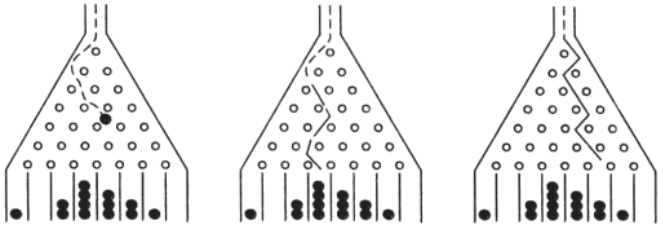


图6-14 每个球都选取一个随机路径，然后掉入一个槽中

球都是从板子口落下的。每当球碰到钉子，它就有50%的机会落向左边或落向右边。在板子底部的槽子中都会累积一堆球。

编写程序模拟豆机。程序应该提示用户输入球的个数以及机器的槽数。打印每个球的路径模拟它的下落。例如：在图6-14b中球的路径是LLRRLLR，而在图6-14c中球的路径是RLRRLRR。使用条形图显示槽中球的最终储备量。下面是程序的一个运行示例：

```
Enter the number of balls to drop: 5 ↵Enter
Enter the number of slots in the bean machine: 7 ↵Enter

LRLRLRR
RRLLLRR
LLRLLRR
RRLLLLL
LRLRRLR

  0
  0
000
```

提示 创建一个名为slots的数组。数组slots中的每个元素存储的是一个槽中球的个数。每个球都经过一条路径落入一个槽中。路径上R的个数表示球落下的槽的位置。例如：对于路径LRLRLRR而言，球落到slots[4]中，而对路径RRLLLLL而言，球落到slots[2]中。

***6.22 （游戏：多种八皇后问题的解决方案）练习题6.20找出八皇后问题的一种解决方案。编写一个程序，计算八皇后问题所有可能的解决方案，然后显示所有的解决方案。

**6.23 （游戏：储物柜难题）一个学校有100个储物柜和100个学生。所有的储物柜在上学第一天都是关着的。随着学生进来，第一个学生，用S1表示，打开每个柜子。然后，第二个学生，用S2表示，从第二个柜子开始，第二个柜子用L2表示，然后关闭其他的柜子。学生S3从第三个柜子开始，然后改变每个第三个柜子（如果它是开的就关上，如果它是关的就打开）。学生S4从柜子L4开始，然后改变每个第四个柜子。学生S5从L5开始，然后改变每个第五个柜子，依此类推，直到学生S100改变L100为止。

在所有学生都经过教学楼并且改变了柜子之后，哪些柜子是开的？编写程序找出答案。

提示 使用存放100个布尔型元素的数组，每个元素都表明一个柜子是开的（true）还是关的（false）。初始状态时，所有的柜子都是关的。

**6.24 （模拟：优惠券收集人问题）优惠券收集人问题是一个经典的统计问题，它有很多实际应用。这个问题重复地从一套对象中拿出一个对象，然后找出要将所有需要拿出的对象都至少拿出来

一次，需要拿多少次。从该问题衍生出的类似问题就是，从一副打乱的52张牌中重复选牌，找出在看到每种花色都有一张出现前，需要选多少次。假设在选下一张牌之前的那张牌是背面向上的。编写程序，模拟要得到四张不同花色的牌所需要的选取次数，然后显示选中的四张牌（有可能一张牌被选了两次）。下面是这个程序的运行示例：

```
Queen of Spades
5 of Clubs
Queen of Hearts
4 of Diamonds
Number of picks: 12
```

6.25 （*代数问题：解二次方程式*）使用下面的方法头编写一个解二次方程式的方法：

```
public static int solveQuadratic(double[] eqn, double[] roots)
```

二次方程式$ax^2+bx+c=0$的系数都传给数组`eqn`，然后将两个非复数的根存在roots里。方法返回根的个数。参见编程练习题3.1了解如何解二次方程。

编写程序，提示用户输入a、b和c的值，然后显示根的个数以及所有非复数的根。

6.26 （*完全相同的数组*）如果两个数组`list1`和`list2`的长度相同，而且对于每个`i`，`list1[i]`都等于`list2[i]`，那么认为`list1`和`list2`是完全相同的。使用下面的方法头编写一个方法，如果`list1`和`list2`完全相同，那么这个方法返回`true`：

```
public static boolean equal(int[] list1, int[] list2)
```

编写一个测试程序，提示用户输入两个整数数列，然后显示这两个数列是否完全相同。下面是运行示例。注意，输入的第一个数字表明数列中元素的个数。

```
Enter list1: 5 2 5 6 1 6 Enter
Enter list2: 5 2 5 6 1 6 Enter
Two lists are strictly identical
```

```
Enter list1: 5 2 5 6 6 1 Enter
Enter list2: 5 2 5 6 1 6 Enter
Two lists are not strictly identical
```

6.27 （*相同的数组*）如果两个数组`list1`和`list2`的内容相同，那么就说它们是相同的。使用下面的方法头编写一个方法，如果`list1`和`list2`是相同的，该方法就返回`true`：

```
public static boolean equal(int[] list1, int[] list2)
```

编写一个测试程序，提示用户输入两个整数数列，然后显示它们两个是否相同。下面是运行示例。注意，输入的第一个数字表示数列中元素的个数。

```
Enter list1: 5 2 5 6 6 1 Enter
Enter list2: 5 5 2 6 1 6 Enter
Two lists are identical
```

```
Enter list1: 5 5 5 6 6 1 Enter
Enter list2: 5 2 5 6 1 6 Enter
Two lists are not identical
```

6.28 （数学方面：组合）编写一个程序，提示用户输入10个整数，然后显示从这10个数中选出两个数的所有组合。

6.29 （游戏：选出四张牌）编写一个程序，从一副52张的牌中选出四张，然后计算它们的和。Ace、King、Queen和Jack分别表示1、13、12和11。程序应该显示若得到的和为24的选牌次数。

6.30 （模式识别方面：四个连续相等的数）编写下面的方法，测试某个数组是否有四个连续的值相同的数字。

```
public static boolean isConsecutiveFour(int[] values)
```

编写测试程序，提示用户输入一个整数数列，如果这个数列中有四个连续的具有相同值的数，那就显示true；否则，显示false。程序应该首先提示用户键入输入的大小，即数列中值的个数。