Chapter 6
Single-Dimensional Arrays

6.1 Introduction

- Array is a data structure that stores a fixed-size sequential collection of elements of the same types.

6.2 Array Basics

- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- This section introduces how to declare array variables, create arrays, and process arrays

6.2.1 Declaring Array Variables

- Here is the syntax for declaring an array variable:

  ```
  dataType[ ] arrayRefVar;
  ```

- The following code snippets are examples of this syntax:

  ```
  double[ ] myList;
  ```

6.2.2 Creating Arrays

- Declaration of an array variable doesn’t allocate any space in memory for the array. Only a storage location for the reference to an array is created.
- If a variable doesn’t reference to an array, the value of the variable is null.
- You can create an array by using the new operator with the following syntax:

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

- This element does two things:
  1) It creates an array using new dataType[arraySize];
  2) It assigns the reference of the newly created array to the variable arrayRefVar.
- Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as follows:

  ```
  dataType[]arrayRefVar = new dataType[arraySize];
  ```
Here is an example of such a statement

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

This statement declares an array variable, myList, creates an array of ten elements of double type, and assigns its reference to myList.

**NOTE**

An array variable that appears to hold an array actually contains a reference to that array. Strictly speaking, an array variable and an array are different.

### 6.2.3 Array Size and Default Values

- When space for an array is allocated, the array size must be given, to specify the number of elements that can be stored in it.
- The size of an array cannot be changed after the array is created.
- Size can be obtained using arrayRefVar.length. For example, `myList.length` is 10.
- When an array is created, its elements are assigned the default value of 0 for the numeric primitive data types, ‘\u0000’ for char types, and `false` for Boolean types.
6.2.4 Array Indexed Variables

- The array elements are accessed through an index.
- The array indices are 0-based, they start from 0 to arrayRefVar.length-1.
- In the example, myList holds ten double values and the indices from 0 to 9. The element myList[9] represents the last element in the array.
- After an array is created, an indexed variable can be used in the same way as a regular variable. For example:

  ```java
  myList[2] = myList[0] + myList[1];  // adds the values of the 1st and 2nd elements into the 3rd one
  for (int i = 0; i < myList.length; i++) // the loop assigns 0 to myList[0]
      myList[i] = i;   // 1 to myList[1] .. and 9 to myList[9]
  ```

6.2.5 Array Initializers

- Java has a shorthand notation, known as the array initializer that combines declaring an array, creating an array and initializing it at the same time.

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

- This shorthand notation is equivalent to the following statements:

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

Caution

- Using the shorthand notation, you have to declare, create, and initialize the array all in one statement. Splitting it would cause a syntax error. For example, the following is wrong:

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

6.2.6 Processing Arrays

- When processing array elements, you will often use a for loop. Here are the reasons why:
  1) All of the elements in an array are of the same type. They are evenly processed in the same fashion by repeatedly using a loop.
  2) Since the size of the array is known, it is natural to use a for loop.
- Here are some examples of processing arrays (Page 173):
  - (Initializing arrays)
o (Printing arrays)
o (Summing all elements)
o (Finding the largest element)
o (Finding the smallest index of the largest element)

6.2.7 For-each Loops

- JDK 1.5 introduced a new for loop that enables you to traverse the complete array sequentially without using an index variable. For example, the following code displays all elements in the array myList:

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

- In general, the syntax is

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

- You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.
6.3 Problem: Lotto Numbers

- Suppose you play the Pick-10 lotto. Each ticket has 10 unique numbers ranging from 1 to 99. You buy a lot of tickets. You like to have your tickets to cover all numbers from 1 to 99. Write a program that reads the ticket numbers from a file and checks whether all numbers are covered. Assume the last number in the file is 0.

**LISTING 6.1 LottoNumbers (Page 205)**

```java
import java.util.Scanner;

public class LottoNumbers {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        boolean[] isCovered = new boolean[99]; // default false

        // Read all numbers and mark corresponding element covered
        int number = input.nextInt();
        while (number != 0) {
            isCovered[number - 1] = true;
            number = input.nextInt();
        }

        // Check if all covered
        boolean allCovered = true; // Assume all covered
        for (int i = 0; i < 99; i++)
            if (!isCovered[i]) {
                allCovered = false; // Find one number is not covered
                break;
            }

        // Display result
        if (allCovered)
            System.out.println("The tickets cover all numbers");
        else
            System.out.println("The tickets don’t cover all numbers");
    }
}
```

```
1 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
The tickets cover all numbers
```
6.4 Problem: Deck of Cards

- The problem is to write a program that picks four cards randomly from a deck of 52 cards. All the cards can be represented using an array named deck, filled with initial values 0 to 52, as follows:

  ```java
  int[] deck = new int[52];
  // Initialize cards
  for (int i = 0; i < deck.length; i++)
    deck[i] = i;
  ```

**Listing 6.2 DeckOfCards.java (Page 207)**

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

        // Initialize 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;
        }

        // 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);
        }
    }
}
```

Card number 6: 7 of Spades  
Card number 48: 10 of Clubs  
Card number 11: Queen of Spades  
Card number 24: Queen of Hearts
6.5 Copying Arrays

- Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

  \[
  \text{list2} = \text{list1};
  \]

- This statement does **not** copy the contents of the array referenced by \textit{list1} to \textit{list2}, but merely **copies the reference value** from \textit{list1} to \textit{list2}. After this statement, \textit{list1} and \textit{list2} reference to the same array, as shown below.

![Figure 6.4 Before the assignment, list1 and list2 point to separate memory locations. After the assignments the reference of the list1 array is passed to list2](image)

- The array previously referenced by \textit{list2} is no longer referenced; it becomes **garbage**, which will be automatically collected by the Java Virtual Machine.

- You can use assignment statements to copy primitive data type variables, but not arrays.

- Assigning one array variable to another variable actually copies one reference to another and makes both variables point to the **same memory location**.

- There are three ways to copy arrays:
  - Use a **loop** to copy individual elements.
  - Use the static **arraycopy** method in the **System** class.
  - Use the **clone** method to copy arrays. “Introduced in chapter 9.”

- Using a **loop**:

  ```java
  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];
  ```
- The **arraycopy** method:

  \[
  \text{arraycopy(sourceArray, src_pos, targetArray, tar_pos, length)};
  \]

  Example:

  \[
  \text{System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length)};
  \]

- The number of elements copied from `sourceArray` to `targetArray` is indicated by `length`.
- The `arraycopy` does **not** allocate memory space for the target array. The target array must have already been created with its memory space allocated.
- After the copying take place, `targetArray` and `sourceArray` have the same content but independent memory locations.
6.6 Passing Arrays to Methods

- The following method displays the elements of an int array:

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

  The following invokes the method to display 3, 1, 2, 6, 4, and 2.

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

- Java uses **pass by value** to pass arguments to a method. There are important differences between passing the values of variables of primitive data types and passing arrays.
- For an argument of a primitive type, the argument’s **value** is passed.
- For an argument of an array type, the value of an argument contains a reference to an array; this **reference** is passed to the method.

  ```java
  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

- `y` and `numbers` reference to the same array, although `y` and `numbers` are independent variables.
- When invoking `m(x, y)`, the values of `x` and `y` are passed to `number` and `numbers`.
- Since `y` contains the reference value to the array, `numbers` now contains the same reference value to the same array.
- The JVM stores the array in an area of memory called **heap**, which is used by dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order.

**FIGURE 6.5** The primitive type value in \( x \) is passed to \( \text{number} \), and the reference value in \( y \) is passed to \( \text{numbers} \).
6.6.1 Example: Passing Array Arguments

LISTING 6.3 TestPassArray: Passing Arrays as Arguments (Page 211)

- For a parameter of an array type, the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body will affect the original array that was passed as the argument.
- **Example:** write two methods for swapping elements in an array. The first method, named `swap`, fails to swap two int arguments. The second method, named `swapFirstTwoInArray`, successfully swaps the first two elements in the array argument.

```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] + "}");
    }

    /** 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;
    }
}
```

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}
The first method doesn't work. The two elements are not swapped using the `swap` method.

The second method works. The two elements are actually swapped using the `swapFirstTwoInArray` method.

Since the arguments in the first method are primitive type, the values of a[0] and a[1] are passed to n1 and n2 inside the method when invoking `swap(a[0], a[1])`.

The memory locations for n1 and n2 are independent of the ones for a[0] and a[1].

The contents of the array are not affected by this call.

- The parameter in the `swapFirstTwoInArray` method is an array.
- As shown above, the reference of the array is passed to the method.
- Thus the variables `a` (outside the method) and `array` (inside the method) both refer to the same array in the same memory location.
- Therefore, swapping array[0] with array[1] inside the method `swapFirstTwoInArray` is the same as swapping a[0] with a[1] outside of the method.
6.7 Returning an Array from a Method

- You can pass arrays to invoke a method. A method may also return an array.
- For example, the method below returns an array that is the reversal of another array:

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

    for (int i = 0, j = result.length - 1; // copies elements from array
        i < list.length; i++, j--) { // list to array result
        result[j] = list[i];
    }
    return result;
}
```

- The following statement returns a new array list2 with elements 6, 5, 4, 3, 2, 1:

```java
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```
6.7.1 Case Study: Counting the Occurrences of Each Letters

LISTING 6.4 CountLettersInArray.java: Counting the Occurrences of Each Letter (Page 213)

- Generate 100 lowercase letters randomly and assign to an array of characters.
- Count the occurrence of each letter in the array.

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

   The occurrences of each letter are:
   2 a 5 b 4 c 4 d 7 e 6 f 3 g 5 h 3 i 2 j
   2 k 2 l 3 m 5 n 4 o 4 p 3 q 0 r 2 s 4 t
   4 u 7 v 8 w 5 x 5 y 1 z */

public class CountLettersInArray {
  /** Main method */
  public static void main(String args[]) {
    // Declare and create an array
    char[] chars = createArray();

    // Display the array
    System.out.println("The lowercase letters are:");
    displayArray(chars);

    // Count the occurrences of each letter
    int[] counts = countLetters(chars);

    // Display counts
    System.out.println();
    System.out.println("The occurrences of each letter are:");
    displayCounts(counts);
  }

  /** Create an array of characters */
  public static char[] createArray() {
    // Declare an array of characters and create it
    char[] chars = new char[100];

    // Create lowercase letters randomly and assign
    // them to the array
    for (int i = 0; i < chars.length; i++)
      chars[i] = RandomCharacter.getRandomLowerCaseLetter();

    // Return the array
    return chars;
}
```
/** Display the array of characters */
public static void displayArray(char[] chars) {
    // Display the characters in the array 20 on each line
    for (int i = 0; i < chars.length; i++) {
        if ((i + 1) % 20 == 0)
            System.out.println(chars[i] + " ");
        else
            System.out.print(chars[i] + " ");
    }
}

/** Count the occurrences of each letter */
public static int[] countLetters(char[] chars) {
    // Declare and create an array of 26 int
    int[] counts = new int[26];

    // For each lowercase letter in the array, count it
    for (int i = 0; i < chars.length; i++)
        counts[chars[i] - 'a']++;

    return counts;
}

/** Display counts */
public static void displayCounts(int[] counts) {
    for (int i = 0; i < counts.length; i++) {
        if ((i + 1) % 10 == 0)
            System.out.println(counts[i] + " " + (char)(i + 'a'));
        else
            System.out.print(counts[i] + " " + (char)(i + 'a') + " ");
    }
}

FIGURE 6.8 (a) An array of 100 characters is created when executing createArray. (b) This array is returned and assigned to the variable chars in the main method.