Chapter 6

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

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

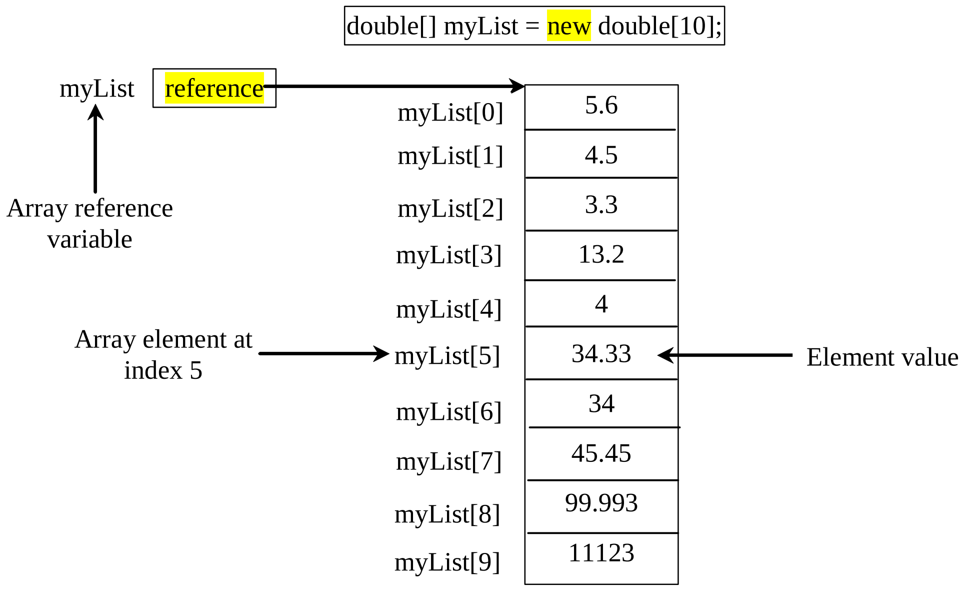


FIGURE 6.1 The array *myList* has ten elements of `double` type and `int` indices from 0 to 9.

- 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:

```
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.

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

- This shorthand notation is **equivalent** to the following statements:

```
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**:

```
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)

- (Printing arrays)
- (Summing all elements)
- (Finding the largest element)
- (Finding the smallest index of the largest element)

6.2.7 foreach 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:

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

  - In general, the syntax is

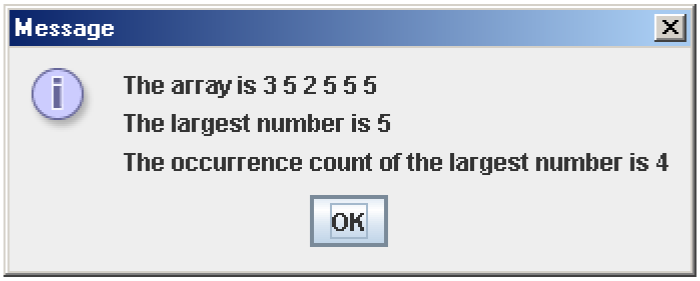
    ```
    for (elementType element: arrayRefVar) {
      // Process the value
    }
    ```

  - 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.

6.2.8 Example: Testing Arrays

LISTING 6.1 Testing Arrays (Page 174)

- Objective: The program receives **6 integers** from the keyboard, finds the largest number and counts the **occurrence** of the largest number entered from the keyboard.
- Suppose you entered 3, 5, 2, 5, 5, and 5, the largest number is 5 and its occurrence count is 4.



```
The array is 3 5 2 5 5 5
The largest number is 5
The occurrence count of the largest number is 4
```

```
import javax.swing.JOptionPane;

public class TestArray {
  /** Main method */
```

```
  public static void main(String[] args) {
    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:");

      // Convert string into integer
      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];
    }

    // Find the occurrence of the largest number
    int count = 0;
    for (int i = 0; i < numbers.length; i++) {
      if (numbers[i] == max) count++;
    }

    // Prepare the result
    String output = "The array is ";
    for (int i = 0; i < numbers.length; i++) {
      output += numbers[i] + " ";
    }

    output += "\nThe largest number is " + max;
    output += "\nThe occurrence count of the largest number "
      + "is " + count;

    // Display the result
    JOptionPane.showMessageDialog(null, output);
  }
}
```

- Without using the *numbers* array, you would have to declare a variable for each number entered, because all the numbers are compared to the largest number to count its occurrences after it is found.

Caution

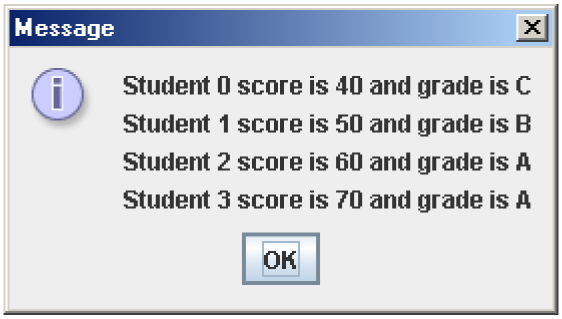
- Accessing an array out of bound is a common programming error. To avoid it, make sure that you **don't** use an index beyond *arrayRefVar.length-1*.
- Programmers often mistakenly reference the first element in an array with index 1, so that the index of the 10th element becomes 10. This is called *the-off-by-one-error*.

6.2.9 Example: Assigning Grades

LISTING 6.2 Assigning Grades (Page 176)

- **Objective**: read student scores (int) from the keyboard, get the best score, and then assign grades based on the following scheme:

  - Grade is A if score is >= best – 10;
  - Grade is B if score is >= best – 20;
  - Grade is C if score is >= best – 30;
  - Grade is D if score is >= best – 40;
  - Grade is F otherwise.



Student 0 score is 40 and grade is C
Student 1 score is 50 and grade is B
Student 2 score is 60 and grade is A
Student 3 score is 70 and grade is A

- The program prompts the user to **enter the total number** of students, then prompts the user to enter all of the scores, and concludes by displaying the grades.

```
import javax.swing.JOptionPane;

public class AssignGrade {
  /** Main method */
  public static void main(String[] args) {
    // Get number of students
    String numberOfStudentsString = JOptionPane.showInputDialog(
      "Please enter number of students:");

    // Convert string into integer
    int numberOfStudents = Integer.parseInt(numberOfStudentsString);

    int[] scores = new int[numberOfStudents]; // Array scores
    int best = 0; // The best score
    char grade; // The grade

    // Read scores and find the best score
    for (int i = 0; i < scores.length; i++) {
      String scoreString = JOptionPane.showInputDialog(
        "Please enter a score:");

      // Convert string into integer
      scores[i] = Integer.parseInt(scoreString);
      if (scores[i] > best)
        best = scores[i];
    }

    // Declare and initialize output string
    String output = "";

    // Assign and display grades
    for (int i = 0; i < scores.length; i++) {
      if (scores[i] >= best - 10)
        grade = 'A';
        else if (scores[i] >= best - 20)
          grade = 'B';
        else if (scores[i] >= best - 30)
          grade = 'C';
        else if (scores[i] >= best - 40)
          grade = 'D';
        else
          grade = 'F';

        output += "Student " + i + " score is " +
          scores[i] + " and grade is " + grade + "\n";
      }

      // Display the result
      JOptionPane.showMessageDialog(null, output);
    }
  }
```

- The program declares *scores* as an array of int type in order to store the student's scores. The size of the array is undetermined when the array is declared.
- After the user enters the number of students into *numOfStudents*, an array with the size of *numOfStudents* is created in Line 13 .
- The size of the array is set at runtime; it cannot be changed once the array is created.

6.3 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:

  ```
  list2 = list1;
  ```

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

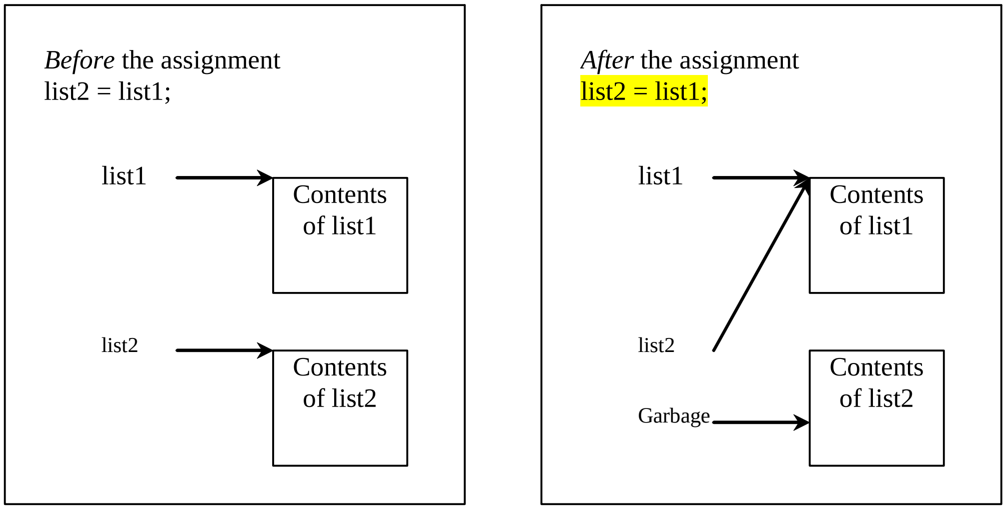


FIGURE 6.4 Before the assignment, list1 and list2 point to separate memory locations. After the assignments the reference of the list1 array is passed to list2

- The array previously referenced by *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**:

  ```
  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:

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

  Example:

  ```
  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.

6.4 Passing Arrays to Methods

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

```
    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.`

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

printArray(new int[]{3, 1, 2, 6, 4, 2}); // anonymous array; no explicit
                                          reference variable for the array
```

- 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.

```
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]
  }
}
```

Result is:

```
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.

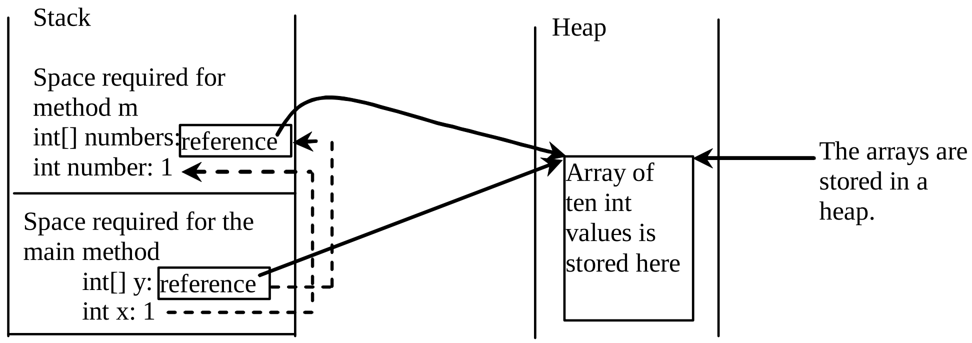

Stack
Space required for method m
int[] numbers:
reference
int number: 1
Space required for the main method
int[] y:
reference
int x: 1
Heap
Array of ten int values is stored here
The arrays are stored in a heap.

FIGURE 6.5 The primitive type value in x is passed to number, and the reference value in y is passed to numbers

6.4.1 Example: Passing Array Arguments

LISTING 6.3 Passing Arrays as Arguments (Page 180)

- 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.

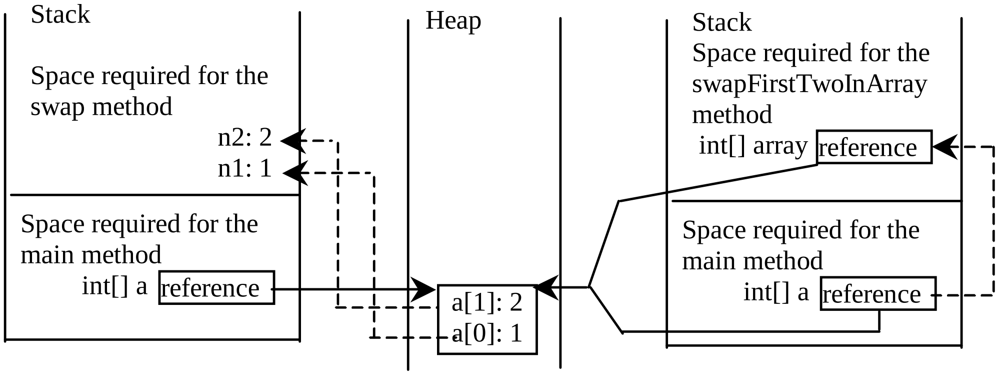
```
public class TestPassArray {
  /** Main method */
  public static void main(String[] args) {
    int[] a = {1, 2};

    // 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;
  }
}
```

Result:

```
   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.


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.

- 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.

6.5 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:

```
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:

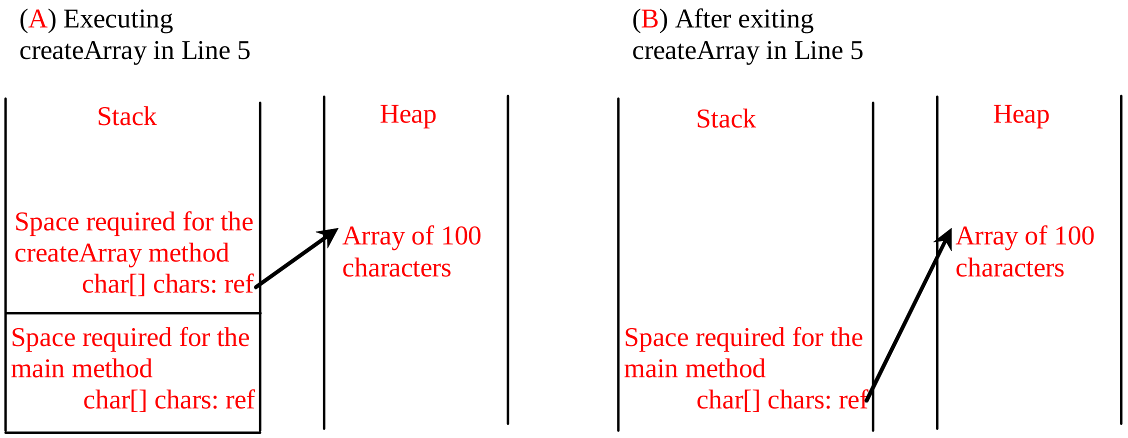
```
   int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
   int[] list2 = reverse(list1);
```

6.5.1 Example: Counting the Occurrences of Each Letters

LISTING 6.4 Counting the Occurrences of Each Letter (Page 182)

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

```
/* 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 d y t b e c p w 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') + " ");
    }
  }
}
```


(A) Executing createArray in Line 5

(B) After exiting createArray in Line 5

FIGURE 6.10 (a) An array of one hundred characters is created when executing createArray. (b) This array is returned and assigned to the variable chars in the main method

6.10 Two-Dimension Arrays

- You can use a two-dimensional array to represent a matrix or a table.
- Occasionally, you will need to represent n-dimensional data structures. In Java, you can create n-dimensional arrays for any integer n.

6.10.1 Declaring Variables of Multidimensional Arrays and Creating Multidimensional Array

- Here is the syntax for declaring a two-dimensional array:

  dataType [][] arrayRefVar;

  or

  dataType arrayRefVar[][];   // *This style is correct, but not preferred*

- As an example, here is how you would **declare** a two-dimensional array variable matrix of int values

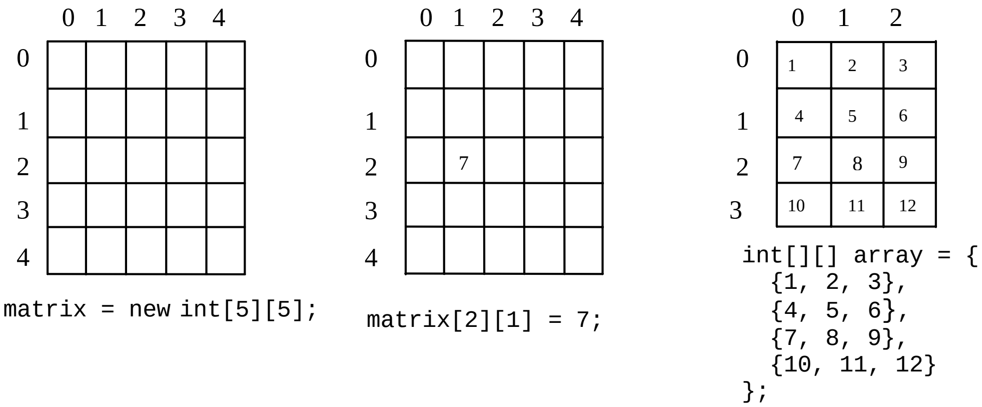
  int [][] matrix;

  or

  int matrix[][]; // *This style is correct, but not preferred*

- You can **create** a two-dimensional array of 5 by 5 int values and assign it to matrix using this syntax:

  matrix = **new** int[5][5];



```
matrix = new int[5][5];
```

```
matrix[2][1] = 7;
```

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

FIGURE 6.12 The index of each subscript of a multidimensional array is an int value starting from 0.

Caution

- It is a common mistake to use matrix[2,1] to access the element at row 2 and column 1.
- In Java, each subscript must be enclosed in a pair of square brackets.
- You can also use an array initializer to declare, create and initialize a two-dimensional array. For example,

```
int[ ][ ] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

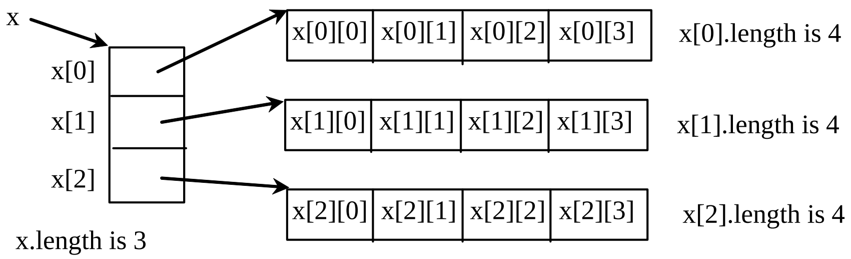
Equivalent

```
int[ ][ ] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

6.10.2 Obtaining the Lengths of Multidimensional Arrays

int[ ][ ] x = new int[3][4];

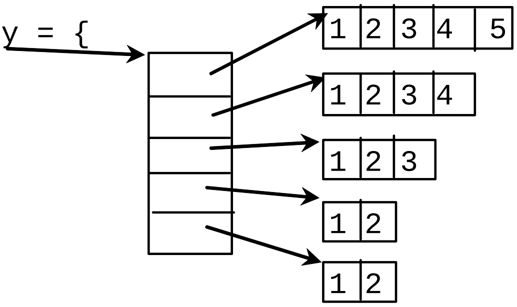
x.length is 3
x[0].length is 4, x[1].length is 4, x[2].length is 4



6.10.3 Ragged Arrays

- Each row in a two-dimensional array is itself an array. Thus, the rows can have different lengths.

```
int[][] triangleArray = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```



- If you **don't** know the values in a raged array in advance, but know the sizes, say the same as before, you can create a ragged array using the syntax that follows:

```
int [][] triangleArray = new int[5][];
triangleArray[0] = new int[5];
triangleArray[1] = new int[4];
triangleArray[2] = new int[3];
triangleArray[3] = new int[2];
triangleArray[4] = new int[1];
```

6.10.4 Processing Two-Dimensional Arrays

- Suppose an array matrix is declared as follows:

  ```
  int [ ] [ ] matrix = new int [10][10];
  ```

- Here are some examples of processing two-dimensional arrays:
  - (Initializing arrays with random values) You can now assign random values to the array using the following loop:

    ```
    for (int row = 0; row < triangleArray.length; row++)
        for (int column = 0; column < triangleArray[row].length; column++)
            triangleArray[row][column] = (int) (Math.random( ) * 1000);
    ```

  - (Printing arrays)
  - (Summing all elements)
  - (Summing elements by column)
  - (Which row has the largest sum?)

6.10.5 Example: Grading a Multiple-Choice Test

- Objective: write a program that grades multiple-choice test.

Students' Answers to the Questions:

	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

Key to the Questions:

	0	1	2	3	4	5	6	7	8	9
Key	D	B	D	C	C	D	A	E	A	D

```
/* Output
      Student 0's correct count is 7
      Student 1's correct count is 6
      Student 2's correct count is 5
      Student 3's correct count is 4
      Student 4's correct count is 8
      Student 5's correct count is 7
      Student 6's correct count is 7
      Student 7's correct count is 7

*/
```

LISTING 6.10 Grading a Multiple-Choice Test

```
public class GradeExam {
  /** Main method */
  public static void main(String args[]) {
    // Students' answers to the questions
    char[][] answers = {
      {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
      {'D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D'},
      {'E', 'D', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D'},
      {'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D'},
      {'A', 'B', 'D', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
      {'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
      {'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
      {'E', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'}};

    // Key to the questions
    char[] keys = {'D', 'B', 'D', 'C', 'C', 'D', 'A', 'E', 'A', 'D'};

    // Grade all answers
    for (int i = 0; i < answers.length; i++) {
      // Grade one student
      int correctCount = 0;
      for (int j = 0; j < answers[i].length; j++) {
        if (answers[i][j] == keys[j])
          correctCount++;
      }

      System.out.println("Student " + i + "'s correct count is " +
        correctCount);
    }
  }
}
```

6.11 Multidimensional Arrays

- The following syntax declares a three-dimensional array variable scores, creates an array, and assigns its reference to scores:

  double [ ] [ ] [ ] x = **new** double[2][3][4];

double[ ][ ] [ ] x = new double[2][3][4];

x.length is 2
x[0].length is 3, x[1].length is 3
x[0][0].length is 4, x[0][1].length is 4, x[0][2].length is 4,
x[1][0].length is 4, x[1][1].length is 4, x[1][2].length is 4