Chapter 5
Arrays

5.1 Introduction
- Array is a data structure that stores a fixed-size sequential collection of elements of identical types.

5.2 Array Basics
- This section introduces how to declare array variables, create arrays, and process arrays

5.2.1 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;

5.2.2 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];
  or
  dataType arrayRefVar [ ] = new dataType[arraySize];

- Here is an example of such a statement
  
  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.

### 5.2.3 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.

### 5.2.4 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.
- 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
```

```java
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]
```

### 5.2.6 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};
```

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.

### EXAMPLE 5.1 Testing Arrays (Page 179)

**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.

```java
// TestArray.java: Count the occurrences of the largest number
import javax.swing.JOptionPane;
public class TestArray {
    /** Main method */
    public static void main(String[] args) {
        int[] numbers = new int[6];
```
// Read all numbers
for (int i = 0; i < numbers.length; i++) {
    String numString = JOptionPane.showInputDialog(null,
        "Enter a number:",
        "Example 5.1 Input", JOptionPane.QUESTION_MESSAGE);

    // 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,
    "Example 5.1 Output", JOptionPane.INFORMATION_MESSAGE);
System.exit(0);
}

- 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.
**EXAMPLE 5.2 Assigning Grades (Page 181)**

- **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 $\geq$ best–10;
  - Grade is B if score is $\geq$ best–20;
  - Grade is C if score is $\geq$ best–30;
  - Grade is D if score is $\geq$ best–40;
  - Grade is F otherwise.

- 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.

```java
// AssignGrade.java: Assign grade
import javax.swing.JOptionPane;

public class AssignGrade {
    /** Main method */
    public static void main(String[] args) {
        int numOfStudents = 0; // The number of students
        int[] scores; // Array scores
        int best = 0; // The best score
        char grade; // The grade

        // Get number of students
        String numOfStudentsString = JOptionPane.showInputDialog(null,
                          "Please enter number of students:",
                          "Example 5.2 Input", JOptionPane.QUESTION_MESSAGE);
        numOfStudents = Integer.parseInt(numOfStudentsString);

        // Create array scores
        scores = new int[numOfStudents];

        // Read scores and find the best score
        for (int i = 0; i < scores.length; i++) {
            String scoreString = JOptionPane.showInputDialog(null,
                          "Please enter a score:",
                          "Example 5.2 Input", JOptionPane.QUESTION_MESSAGE);
            scores[i] = Integer.parseInt(scoreString);
            if (scores[i] > best)
                best = scores[i];
        }

        // 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,
    "Example 5.2 Output", JOptionPane.INFORMATION_MESSAGE);

System.exit(0);

}
5.3 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:

```java
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.

![Diagram showing before and after assignment](image)

FIGURE 5.4 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 8.”

- 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:

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

  Example:

  ```java
  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.
### 5.4 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
  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.

```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]
    }
}
```

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.

**EXAMPLE 5.3 Passing Arrays as Arguments (Page 186)**

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;
    }
}
```
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.

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.
5.5 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:

```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);
```

**EXAMPLE 5.4 Counting the Occurrences of Each Letter (Page 188)**

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

```java
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
```
```java
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 6
    Heap
    Space required for the createArray method char[] chars: ref
    Array of 100 characters
    Stack
    Space required for the main method char[] chars: ref

(B) After exiting createArray in Line 6
    Stack
    Space required for the main method char[] chars: ref
    Heap
    Array of 100 characters

FIGURE 5.9 (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
```
5.8 Multidimensional 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.

5.8.1 Declaring Variables of Multidimensional Arrays and Creating Multidimensional Array

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

  ```java
dataType [][] arrayRefVar;
```

  or

  ```java
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

  ```java
  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:

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

![Matrix Diagram]

FIGURE 5.12 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}
};
```

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

```
double[ ][ ][ ] scores = new double[10][5][2];
```

### 5.8.2 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

```
int[ ][ ][ ] x = new int[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[0][3].length is 4, x[1][0].length is 4, x[1][1].length is 4, x[1][2].length is 4
5.8.3 Ragged Arrays

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

```java
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 ragged array in advance, but know the sizes, say the same as before, you can create a ragged array using the syntax that follows:

```java
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];
```

- You can now assign random values to the array using the following loop:

```java
for (int row = 0; row < triangleArray.length; row++)
    for (int column = 0; column < triangleArray[row].length; column++)
        triangleArray[row][column] = (int) (Math.random() * 1000);
```
EXAMPLE 5.5 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', 'B', 'E', 'E', 'A', 'D'},
            {'E', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'B'},
            {'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D'},
            {'A', 'B', 'D', 'C', 'A', 'E', 'E', 'A', 'D'},
            {'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
            {'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
            {'C', 'B', 'E', 'C', 'C', 'B', 'E', 'E', 'A', 'D'}
        };

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