Java Chapter 6: Arrays and ArrayLists

Contents

Time required: 120 minutes

Do: Online Tutorial

Go through the following shorter tutorials.

- Java Arrays
- Loop Through an Array
- Multidimensional Arrays
- Java ArrayLists
- Java YouTube Tutorial - 01 - Declaring Arrays & Accessing Elements

Do: Think Java, 2nd Ed. (Interactive Edition)

- Chapter 7 Arrays

Do: Introduction to Programming with Java

- Chapter 7 Arrays
- Chapter 8 ArrayLists
- Chapter 9 2D Arrays

Arrays

We will learn how to store multiple values of the same type by using a single variable. This language feature will enable you to write programs that manipulate larger amounts of data.

An array is a sequence of values; the values in the array are called elements. You can make an array of ints, doubles, Strings, or any other type, but all the values in an array must have the same type.

Creating Arrays

To create an array, you declare a reference variable with an array type, then create the array itself. Array types look like other Java types, except they are followed by square brackets ([]).

For example, the following lines declare that **counts** is an "**integer array**" and **values** is a "**double array**".

```
int[] counts;
double[] values;
```

To create the array itself, you use the "**new**" operator. The new operator allocates memory for the array and automatically initializes all its elements to zero:

```
counts = new int[4];
values = new double[size];
```

The first assignment makes "**counts**" refer to an array of four integers. The second makes "**values**" refer to an array of doubles, but the number of elements depends on the value of size (at the time the array is created).

You can also declare the variable and create the array with a single line of code:

```
int[] counts = new int[4];
double[] values = new double[size];
```

You can use any integer expression for the size of an array. The value must be nonnegative. If you try to create an array with -4 elements, for example, you will get a "**NegativeArraySizeException**".

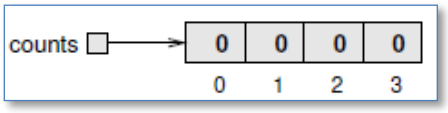
You can initialize an array with a comma-separated sequence of elements enclosed in braces, like this:

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

This statement creates an array variable, "**myArray**", and creates an array with four elements. In this instance, you don't need the new keyword, Java handles that automatically.

Accessing Arrays

When you create an array with the new operator, the elements are initialized to zero. The following diagram shows a memory diagram of the "**counts**" array.



The arrow indicates that the value of "**counts**" is a reference to the array. You should think of the array and the variable that refers to it as two different things. As you'll soon see, we can assign a different variable to refer to the same array, and we can change the value of counts to refer to a different array.

The boldface numbers inside the boxes are the elements of the array. The lighter numbers outside the boxes are the indexes used to identify each location in the array. As with strings, the index of the first element is 0, not 1.

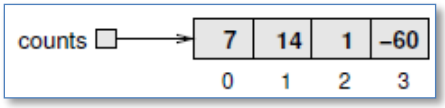
The [] operator selects elements from an array:

```
System.out.println("The first element is " + counts[0]);
```

You can use the [] operator anywhere in an expression:

```
counts[0] = 7;
counts[1] = counts[0] * 2; counts[2]++;
counts[3] -= 60;
```

The following diagram shows the results of these statements.



Arrays and For Loops

For loops are commonly used to through an array one element at a time.

```
public class ArrayForLoop {
    public static void main(String[] args) {
        // Initialize array
        int[] counts = { 1, 2, 3, 4, 5 };
        // Iterate through array one element at a time
        for (int i = 0; i < counts.length; i++) {
            System.out.println(counts[i]);
        }
    }
}
```

Example run:

```
1
2
3
4
5
```

The loop variable "**i**" is used as the index element to iterate through the array. "**i**" is commonly used for a counter variable that has no meaning.

For the counts array, the only legal indexes are 0, 1, 2, 3, and 4. If the index is negative or greater than 3, the result is an "**ArrayIndexOutOfBoundsException**".

Many computations can be implemented by looping through the elements of an array and performing an operation on each element. Looping through the elements of an array is called traversing or iterating.

```
public class ArraySquares {
    public static void main(String[] args) {
        // Square each element of an array
        int[] anArray = { 1, 2, 3, 4, 5 };
        for (int i = 0; i < anArray.length; i++) {
            // Print original element
            System.out.print(anArray[i]);
            // Square each element
            anArray[i] = anArray[i] * anArray[i];
            // Print squared element
            System.out.print(" " + anArray[i] + "\n");
        }
    }
}
```

Example run:

```
1 1
2 4
3 9
4 16
5 25
```

This example traverses an array and squares each element. At the end of the loop, the array has the values {1, 4, 9, 16, 25}.

Enhanced For Loop

Traversing arrays is so common, Java provides an alternative syntax that makes the code more compact. Consider a for loop that displays the elements of an array on separate lines:

```
int[] values = {1, 2, 3, 4, 5};
for (int i = 0; i < values.length; i++){
    int value = values[i];
    System.out.println(value);
}
```

We could rewrite the loop like this:

```
int[] values = {1, 2, 3, 4, 5};
for (int value : values) {
    System.out.println(value);
}
```

This statement is called an enhanced for loop, also known as the "**for each**" loop. You can read the code as, "**for each value in values**". It's conventional to use plural nouns for array variables and singular nouns for element variables.

Notice how the single line for " **(int value : values)**" replaces the first two lines of the standard for loop. It hides the details of iterating each index of the array, and instead, focuses on the values themselves.

Using the enhanced for loop, we can write an accumulator very concisely:

```
int[] counts = new int[100];
for (int score : scores){
    counts[score]++;
}
```

Tutorial 6.1: Arrays and Loops

```java
/**
 * Accessing and Looping Arrays
 */
public class ArrayAccessLoops {

    public static void main(String[] args) {
        // Declare constant for array size
        final int SIZE = 4;
        int[] counts = new int[SIZE];

        // Traversal with a for loop
        System.out.println("\nFor loop with original empty array");
        for (int i = 0; i < 4; i++) {
            System.out.print(counts[i] + " ");
        }

        // Access individual array elements
        counts[0] = 7;
        counts[1] = counts[0] * 2;
        counts[2]++;
        counts[3] -= 60;

        // Traversal with a while loop
        System.out.println("\nWhile loop");
        // Counter for while loop
        int j = 0;
        while (j < 4) {
            System.out.print(counts[j] + " ");
            j++;
        }

        // Traversal with a for each loop
        System.out.println("\nFor each loop");
        // For each element in the array
        for (int count : counts) {
            System.out.print(count + " ");
        }

        // Multiply each element by 2
        System.out.println("\nFor loop multiply each element by 2");
        for (int i = 0; i < 4; i++) {
            int countsTwo = (counts[i] * 2);
            System.out.print(countsTwo + " ");
        }
    }
}
```

Example run:

```
For loop with original empty array
0 0 0 0
While loop
7 14 1 -60
For each loop
7 14 1 -60
For loop multiply each element by 2
14 28 2 -120
```

Tutorial 6.2: Arrays with Methods

When using an array as a parameter in a method, a reference to the original array is passed. There is only one array. If the array is changed in a method, it is changed everywhere as there is only one array referenced. This is different than primitive types (int, double, etc.) where a copy of the variable is passed.

```java
// Provides Array methods
import java.util.Arrays;

public class ArraysFunctions {
    public static void main(String[] args) {

        int[] array = { 1, 2, 3, 4 };

        // Displaying array with function with array argument
        System.out.println("Print Array using function with array argument");
        printArray(array);

        // Printing an array as an object
        // This shows the memory address of the object
        System.out.println("Print Array's memory address");
        System.out.println(array);

        // Printing with Arrays class
        System.out.println("Print Array using Arrays class method");
        System.out.println(Arrays.toString(array));

```

```
22          // Copying an array is done by copying the elements
23          // from one array to the next
24          double[] a = { 1.0, 2.0, 3.0 };
25          double[] b = new double[a.length];
26          System.out.println("Print copy b of Array a");
27          for (int i = 0; i < a.length; i++) {
28              b[i] = a[i];
29              System.out.print(b[i] + " ");
30          }
31
32          System.out.println("\nPrint copy c of Array a");
33          // Copying with Arrays class method
34          double[] c = Arrays.copyOf(a, a.length);
35          for (double i : c) {
36              System.out.print(i + " ");
37          }
38
39
40          // Search an array using a method with parameters
41          int index = search(a, 2.0);
42          System.out.println("\nindex = " + index);
43
44          // reduce
45          double total = sum(a);
46          System.out.println("total = " + total);
47      }
```

```java
    /**
     * Function to print the elements of an array
     */
    public static void printArray(int[] a) {
        System.out.print("{" + a[0]);
        for (int i = 1; i < a.length; i++) {
            System.out.print(", " + a[i]);
        }
        System.out.println("}");
    }

    /**
     * Returns the index of the target in the array, or -1 if not found.
     */
    public static int search(double[] array, double target) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == target) {
                return i;
            }
        }
        return -1; // not found
    }

    /**
     * Returns the total of the elements in an array
     */
    public static double sum(double[] array) {
        double total = 0.0;
        for (int i = 0; i < array.length; i++) {
            total += array[i];
        }
        return total;
    }
}
```

Example run:

```
Print Array using function with array argument
{1, 2, 3, 4}
Print Array's memory address
[I@85ede7b
Print Array using Arrays class method
[1, 2, 3, 4]
Print copy b of Array a
1.0 2.0 3.0
Print copy c of Array a
1.0 2.0 3.0
index = 1
total = 6.0
```

Random Numbers and Arrays

Most computer programs do the same thing every time they run; programs like that are called deterministic. Usually, determinism is a good thing, since we expect the same calculation to yield the same result. But for some applications, we want the computer to be unpredictable. Games are an obvious example, but there are many others, like scientific simulations.

Making a program nondeterministic turns out to be hard, because it's impossible for a computer to generate truly random numbers. But there are algorithms that generate unpredictable sequences called pseudorandom numbers. For most applications, they are as good as random.

The "**java.util.Random**" library generates pseudorandom numbers. The method "**nextInt**" takes an integer argument, n, and returns a random integer between 0 and n - 1 (inclusive).

If you generate a long series of random numbers, every value should appear, at least approximately, the same number of times. One way to test this behavior of" **nextInt**" is to generate many values, store them in an array, and count the number of times each value occurs.

Tutorial 6.3: Random Numbers Method

Create a Java program named **RandomArray.java**

The following main method generates an array using the "**randomArray()**" method and displays it by using the "**printArray()**".

```java
/**
 * Name: RandomArray.java
 * Written by:
 * Written on:
 * Purpose: Create array with random numbers using method
  */

import java.util.Random;

public class RandomArray {
    public static void main(String[] args) {
        final int SIZE = 10;
        // Create array with SIZE elements using randomArray method
        int[] array = randomArray(SIZE);
        printArray(array);
    }
```

The following method creates an int array and fills it with random numbers between 1 and 99 inclusive. The argument specifies the desired size of the array, and the return value is a reference to the new array.

```java
    /**
     * Fill and returns an array of random integers
     */
    public static int[] randomArray(int size) {
        // Create random object
        Random random = new Random();
        int[] randomNumbersArray = new int[size];
        for (int i = 0; i < randomNumbersArray.length; i++) {
            // Get a random integer from 1 - 99
            randomNumbersArray[i] = random.nextInt(1, 100);
        }
        // Return array
        return randomNumbersArray;
    }
```

```
33      /**
34       * Loop through and print the elements of an array
35       */
36      public static void printArray(int[] a) {
37          // Print array up until the last element
38          for (int i = 0; i < a.length - 1; i++) {
39              System.out.print(a[i] + ", ");
40          }
41          // Print the last element of the array
42          System.out.println(a[a.length - 1]);
43      }
44  }
```

Each time you run the program; you should get different values. The output will look something like this:

```
16, 90, 75, 37, 61, 43, 22, 81, 42, 98
```

Array Search

Another common pattern is a search, which involves traversing an array and "searching" for a particular element. For example, the following method takes an array and a value, and it returns the index where the value appears:

```
public class ArraySearchMethod {
    public static void main(String[] args) {
        double[] anArray = { 1.0, 2.0, 3.0 };
        // Search an array using a method with parameters
        int index = search(anArray, 2.0);
        System.out.println("\nindex = " + index);
    }
    /**
     * Returns the index of the target in the array, or -1 if not found
     */
    public static int search(double[] array, double target) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == target) {
                return i;
            }
        }
        return -1; // not found
    }
}
```

If we find the target value in the array, we return its index immediately. If the loop exits without finding the target, it returns -1, a special value chosen to indicate a failed search. (This code is essentially what the "**String.indexOf**" method does.)

The following code searches an array for the value 1.23, which is the third element. Because array indexes start at 0, the output is 2:

```
double[] array = {3.14, -55.0, 1.23, -0.8};
int index = search(array, 1.23);
System.out.println(index);
```

Another common traversal is a reduce operation, which "**reduces**" an array of values down to a single value. This can also be a running total of the values in the array.

Examples include the sum or product of the elements, the minimum, and the maximum. The following method takes an array and returns the sum of its elements:

```
public static double sum(double[] array){
    double total = 0.0;
    for (int i = 0; i < array.length; i++){
        // Accumulate or sum the values in the array
        total += array[i];
    }
    return total;
}
```

Before the loop, we initialize total to 0. Each time through the loop, we update total by adding one element from the array. At the end of the loop, total contains the sum of the elements. A variable used this way is sometimes called an accumulator because it "**accumulates**" the running total.

ArrayLists

ArrayLists are like Arrays but are **dynamically sized.** Developers don't need to specify the capacity or the maximum size when declaring ArrayLists. As elements are added and removed, it **grows or shrinks** its size automatically.

ArrayLists use wrapper classes for the data types such as String, Character, Integer, Double, etc. Primitive types such as int and double will not work.

The previous Array operations hold for ArrayLists. There are additional methods as the ArrayList is mutable.

ArrayList Operations

- **add(element)** to add a new element to the end of this list or **add(index, element)** to add element at the specified index.
- **addAll(ArrayList)** to append another ArrayList) to it.
- **set(index, element)** replaces the element at the specified position in this list with the specified element.
- **get(index)** to get an element at the index.
- **isEmpty()** to check if there are any elements in this list. Returns true if this list contains no elements.
- **size()** returns the number of elements in this list.

- **clear()** to delete all elements. You can also use removeAll() to achieve the same, but it is slower than clear().
- **remove(index)** removes the element at the index in this list. Shifts any subsequent elements to the left (subtracts one from their indices).
- **remove(element)** removes the first occurrence of the specified element from this list.
- **removeIf(predicate)** removes element from the ArrayList that satisfy the predicate argument e.g. removeIf(s -> s.length == 3)
- **contains(element)**, **indexOf(element)** and **lastIndexOf(element)** methods are used for searching for elements in ArrayList.
- **listIterator()** to get an ordered list iterator over the elements in this list.
- **toArray()** converts this ArrayList into an array.

ArrayList

Let's create an ArrayList.

Import the `ArrayList` class: Import the `ArrayList` class from the `java.util` package at the beginning of your Java code to use `ArrayList`.

```
import java.util.ArrayList;
```

Declare an ArrayList: Use the following syntax:

```
// Replace 'type' with the type of elements you want to store in the
// ArrayList, and 'name' with the desired name of the ArrayList.
ArrayList<type> name = new ArrayList<type>();
```

For example, to create an `ArrayList` of integers:

```
// Declare an empty ArrayList of integers
ArrayList<Integer> myArrayList = new ArrayList<Integer>();
```

Access Elements in an ArrayList

Access elements in an `ArrayList` using the index (starting from 0) using the `get()` method.

```
// Accesses the first element of the ArrayList
int element = myArrayList.get(0);
```

Add Elements to an ArrayList

Add elements to an ArrayList by using **add() method**, this method has couple of variations, which we can use based on the requirement.

For example: If we want to add the element at the end of the List then simply do it like this:

```
alist.add("Steve"); //This will add "Steve" at the end of List
```

To add the element at the specified location in ArrayList, we can specify the index in the add method like this:

```
alist.add(3, "Steve"); //This will add "Steve" at the fourth position
```

Change an Element in an ArrayList

The **set()** method changes an element in ArrayList. We provide the index and new element, this method then updates the element present at the given index with the new given element. In the following example, we have given the index as 0 and new element as "Lucy" in the set() method.

```
names.set(0, "Lucy");
```

Remove Elements from an ArrayList

The **remove()** method removes elements from an ArrayList.

```
names.remove("Lucy"};
```

Tutorial 6.4: ArrayList Fun

Create a Java program named **ArrayListFun.java**

```java
/**
 * Name: ArrayListFun.java
 * Written by:
 * Written on:
 * Purpose: ArrayList demonstration
 */

import java.util.ArrayList;
import java.util.Collections;

public class ArrayListFun {
    public static void main(String[] args) {
        // Create ArrayList to store Strings
        ArrayList<String> namesList = new ArrayList<>();

        // Add items to the ArrayLIst
        namesList.add("John");
        namesList.add("Kyle");
        namesList.add("Zorro");
        namesList.add("Amanda");
        namesList.add("Wendy");

        System.out.println("Original ArrayList");
        displayArrayList(namesList);

        // Get first item in ArrayList
        String nameZero = namesList.get(0);
        System.out.println(" First name in list: " + nameZero);

        // Remove Amanda
        namesList.remove("Amanda");

        // Add Floyd at the 2 position
        namesList.add(2, "Floyd");

        // Change the first item to Lucy
        namesList.set(0, "Lucy");

        System.out.println("Modified ArrayList");
        displayArrayList(namesList);

        // Use Collections library to sort the ArrayList
        Collections.sort(namesList);
```

```
45          System.out.println("Sorted ArrayList");
46          // Loop through ArrayList with for each loop
47          for (String name : namesList) {
48              System.out.print("- " + name + " -");
49          }
50          System.out.println();
51      }// end main
52
53      /**
54       * @summary Display ArrayList with for loop
55       * @param names ArrayList<String>
56       */
57      static void displayArrayList(ArrayList<String> names) {
58          // Get size of ArrayList
59          int size = names.size();
60          System.out.println("List size is " + size);
61          // Loop through ArrayList with standard for loop
62          for (int x = 0; x < size; ++x) {
63              System.out.print("- " + names.get(x) + " -");
64          }
65          System.out.println();
66      }
67  }
```

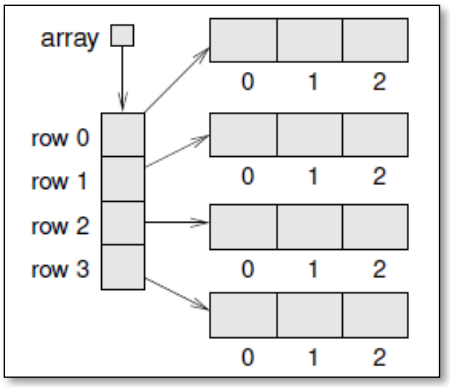
Example run:

```
Original ArrayList
List size is 5
- John -- Kyle -- Zorro -- Amanda -- Wendy -
 First name in list: John
Modified ArrayList
List size is 5
- Lucy -- Kyle -- Floyd -- Zorro -- Wendy -
Sorted ArrayList
- Floyd -- Kyle -- Lucy -- Wendy -- Zorro -
```

2D Arrays

A 2D array is a multidimensional array. We can go beyond 2 dimensions, but that is rarely used. A 2D array or arraylist looks like a spreadsheet.

In Java, a 2D array is really an array of arrays. You can think of it as an array of rows, where each row is an array.



Tutorial 6.5: 2D Arrays

An example of a use for a 2D array would be a Tic Tac Toe game.

```java
/**
 * Filename: TwoDArrayExample.java
 * Demonstrate a two-dimensional array
 */
public class TwoDArrayExample {
    public static void main(String[] args) {
        // Array of size 4x3 to hold integers.
        int[][] values = {
                { 10, 20, 30 },
                { 40, 50, 60 },
                { 70, 80, 90 },
                { 11, 21, 31 }
        };

        // Nested loops to print the array in tabular form.
        // Iterate through array one row at a time
        for (int row = 0; row < 4; row++) {
            // Iterate through current row one element at a time
            for (int col = 0; col < 3; col++) {
                // Use print to keep row elements on same line
                System.out.print(values[row][col] + " ");
            }
            // println to separate rows
            System.out.println();
        }
    }
}
```

Example run:

```
10 20 30
40 50 60
70 80 90
11 21 31
```

Assignment 1: Sum of Array Elements

Requirement: Write a Java program to calculate the sum of all elements in an array.

1. Create a method **sumArray()** that takes an integer array or arraylist as a parameter and returns the sum of its elements.
2. In the main method, ask the user to input the number of elements in the array.
3. Prompt the user to enter each element of the array.
4. Call the **sumArray** method and display the result.

Example run:

```
Enter the number of elements in the array: 3
Enter the elements of the array:
2
3
4
The sum of the array elements is: 9
```

Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.