Chapter 7 – Single-Dimensional Arrays

Contents

[What is an array? 2](#_Toc38234351)

[Declaring Arrays: 2](#_Toc38234352)

[Creating Arrays: 2](#_Toc38234353)

[Combining – declare & create: 2](#_Toc38234354)

[To assign values to the elements: 2](#_Toc38234355)

[Size 3](#_Toc38234356)

[Default values 3](#_Toc38234357)

[How to Access Arrays Elements: 3](#_Toc38234358)

[Index: 3](#_Toc38234359)

[Array Initializers: 3](#_Toc38234360)

[for loops are often used to process arrays 3](#_Toc38234361)

[Processing Arrays 4](#_Toc38234362)

[Copying Arrays: 5](#_Toc38234363)

[Passing Arrays to Methods: 6](#_Toc38234364)

[Returning an Array from a Method: 8](#_Toc38234365)

[Searching Arrays: 8](#_Toc38234366)

[Linear Search: 8](#_Toc38234367)

[ Binary Search 9](#_Toc38234368)

[Sorting Arrays: 10](#_Toc38234369)

[ Selection Sort 10](#_Toc38234370)

[Arrays class 12](#_Toc38234371)

[Command-Line Arguments: 12](#_Toc38234372)

# What is an array?

Array: stores a fixed-size sequential collection of elements of the same data type.

* A single array variable can reference a large collection of data [of the same data type].
* Once an array is created, its size is fixed.
* An array reference variable is used to access the elements in an array using an index.

# Declaring Arrays:

elementType [] arrayVariableName;

Example: double [] aryList;

* Declaring an array does **not** create memory for the array (unlike primitive data types).
* Creates a storage location for the [future] reference to the array.

# Creating Arrays:

arrayVariableName = new elementType [size];

Example: aryList = new double[10];

* Create an array – This is when the memory space is allocated – the reference is assigned to the variable
* Assigns the reference of the newly created array to the variable name

# Combining – declare & create:

elementType [] arrayVariableName = new elementType [size];

Example: double [] aryList = new double[10];

* Declared and created an array of type: double, with variable name: aryList, with 10 elements in the array
* Default values are assigned to the data elements
  + i.e. if int data type, then 0 is assigned to all the elements, if double data type, then 0.0 is assigned to the elements , on so on.

# To assign values to the elements:

arrayVariableName[index] = value;

Example: aryList[0] = 5.6;

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, but most of the time the distinction can be ignored.

Size:

* When the array is created, the size is given, then space is allocated
* the size of the array can be obtained using arrayVariableName.length
* the size **cannot be changed after** the array is created

Default values:

* Primitive numeric data types (int, double) are assigned value of 0
* Character data types are assigned: \u0000
* boolean data types are assigned false

# How to Access Arrays Elements:

* the array elements are accessed through the index

# Index:

* counting starts at 0 and ends at arrayVariableName.length - 1
* after an array is created, an indexed variable can be used in the same way as a regular variable

# Array Initializers:

* shorthand notation
* combines the declaration, creation, and initialization of an array into one statement.

elementType [] arrayVariableName = {value0, value1, …, valuek};

Example: double [] aryList = {1.9, 2.9, 3.4, 3.5};

for loops are often used to process arrays, for 2 reasons:

1. All of the elements in an array are of the same type. They are evenly processed in the same fashion repeatedly using a loop.
2. Since the size of the array is known, it is natural to use a for loop.

# Processing Arrays

**MAKE SURE YOU KNOW HOW TO DO #1, 3, 4, & 5**

The following are some examples of processing arrays.

1. Initializingarrays with input values: The following loop initializes the array myList with user input values.

Scanner input = new Scanner(System.in);

System.out.print("Enter " + myList.length + " values: ");

for (int i = 0; i < myList.length; i++){

myList[i] = input.nextDouble();

}

1. Initializing arrays with random values: The following loop initializes the array myList with random values between 0.0 and 100.0, but less than 100.0.

for (int i = 0; i < myList.length; i++) {

myList[i] = Math.random() \* 100;

}

1. Displaying arrays: To print an array, you have to print each element in the array using a loop like the following:

for (int i = 0; i < myList.length; i++) {

System.out.print(myList[i] + " ");

}

1. Summing all elements: Use a variable named total to store the sum. Initially total is 0. Add each element in the array to total using a loop like this:

double total = 0;

for (int i = 0; i < myList.length; i++) {

total += myList[i];

}

1. Finding the largest element: Use a variable named max to store the largest element. Initially max is myList[0]. To find the largest element in the array myList, compare each element with max, and update max if the element is greater than max.

double max = myList[0];

for (int i = 1; i < myList.length; i++) {

if (myList[i] > max)

max = myList[i];

}

1. Finding the smallest index of the largest element: Often you need to locate the largest element in an array. If an array has multiple elements with the same largest value, find the smallest index of such an element. Suppose the array myList is {1, 5, 3, 4, 5, 5}. The largest element is 5 and the smallest index for 5 is 1. Use a variable named max to store the largest element and a variable named indexOfMax to denote the index of the largest element. Initially max is myList[0], and indexOfMax is 0. Compare each element in myList with max, and update max and indexOfMax if the element is greater than max.

double max = myList[0];

int indexOfMax = 0;

for (int i = 1; i < myList.length; i++) {

if (myList[i] > max) {

max = myList[i];

indexOfMax = i;

}

}

# Copying Arrays:

* To copy the contents of one array into another, you have to copy the array’s **individual elements** into the other array.
* The following statement does not copy the contents, it copies the reference to the same array (meaning 2 references to the same array):

list1 = list2;

* There are 3 ways to copy arrays:
  + Use a loop to copy the elements 1 by 1

int[] sourceArray = {2, 3, 1, 5, 10};

int[] targetArray = new int[sourceArray.length];

for (int i = 0; i < sourceArray.length; i++) {

targetArray[i] = sourceArray[i];

}

* + Use the static arraycopy method in the System calls.

int []a = {1, 8, 3};

       // Create an array b[] of same size as a[]

        int []b = new int[a.length];

        // Copy elements of a[] to b[]

        System.arraycopy(a, 0, b, 0, 3);

* + Use the clone method

int []a = {1, 8, 3};

        // Copy elements of a[] to b[]

        int []b = a.clone();

* The **incorrect** way to copy:

int[] a = {1, 8, 3};

// Create an array b[] of same size as a[]

int[] b = new int[a.length];

// Doesn't copy elements of a[] to b[], only makes

// b refer to same location

b = a;

# Passing Arrays to Methods:

* When passing an array to a method, the reference of the array is passed to the method.

public static void main(String[] args){

int [] ary = {1, 2, 3};

printArrary(ary);

}

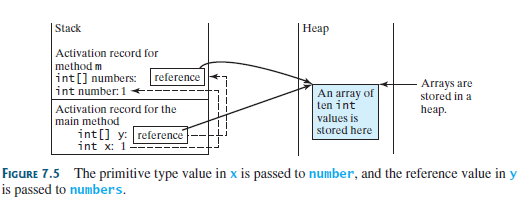
public static void printArray(int[] array) {

for (int i = 0; i < array.length; i++) {

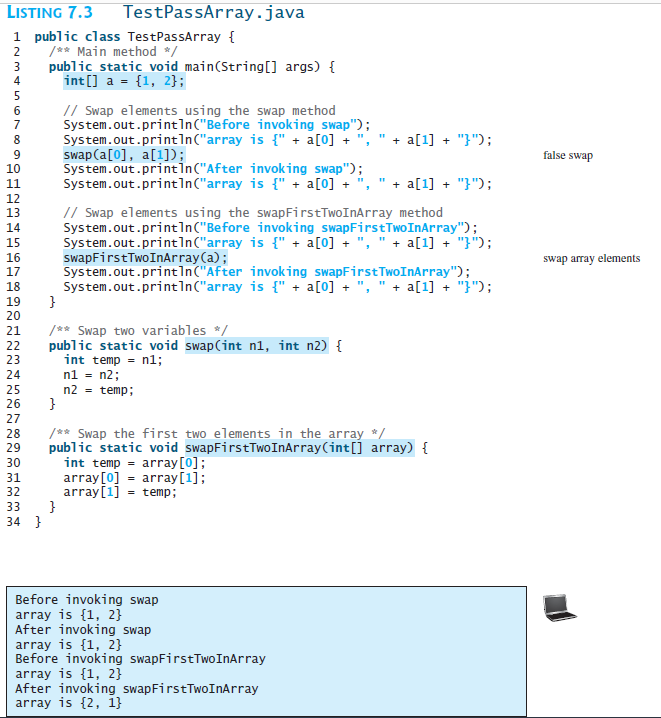
System.out.print(array[i] + " ");

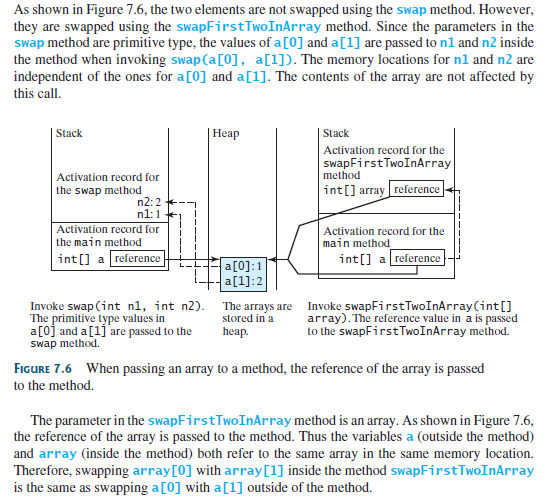
}

}



* Java uses *pass-by-value*:
  + For primitive data types, the value of the variable is passed to the method
  + For array data types, the value of the reference to an array is passed to the method. Meaning, that the changes in the array in the method will be seen outside the method.
  + Listing 7.3, TestPassArray.java (code listed below), shows the difference between passing a primitive data type and an array reference variable to a method.





# Returning an Array from a Method:

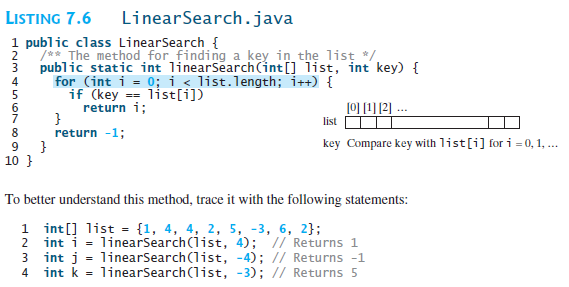
* When a method returns an array, the reference of the array is returned

# Searching Arrays:

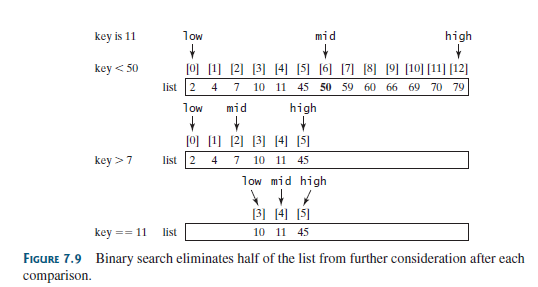
* Searching is the process of looking for a specific element in an array
* If the array is sorted, a binary search is more efficient than a linear search.

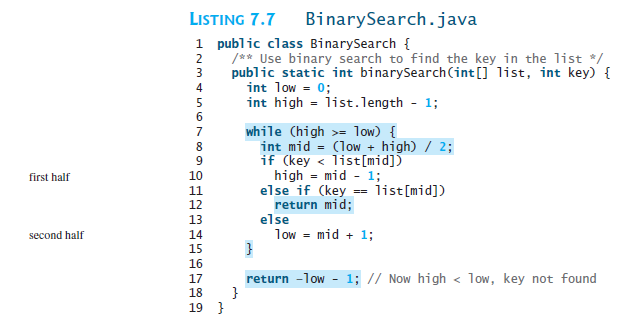
## Linear Search:

* + Compares the key (the element you are looking for) sequentially to each element in the array until the key matches or returns a -1 if a match is not found
  + Inefficient for large arrays
    - The linear search method compares the key with each element in the array. The elements can be in any order.
    - n elements means, worst case scenario, it take n iterations of a for loop to not find the key

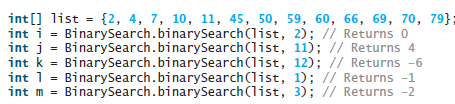


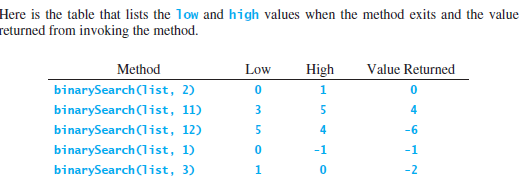
* Binary Search:
  + The array must be sorted!
  + Starts with the middle – compares the middle value to the key
    - if the key is less than the middle element’s value, it searches the first half of the array
    - If the key is equal to the middle element’s value, the search ends with a match
    - If the key is greater than the middle element’s value, it searches the last half of the array





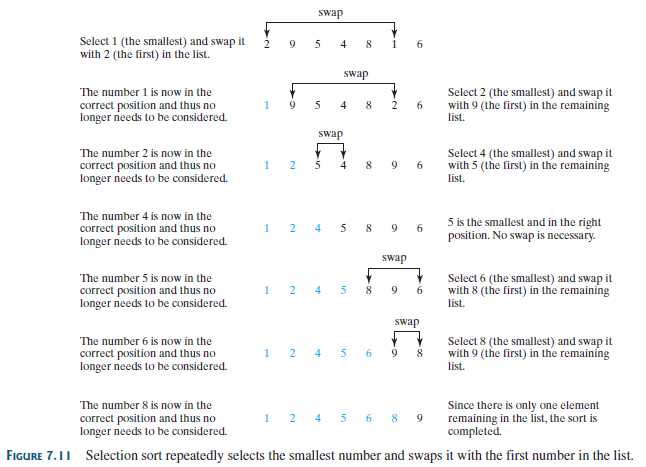
To better understand this method, trace it with the following statements and identify low and high when the method returns.

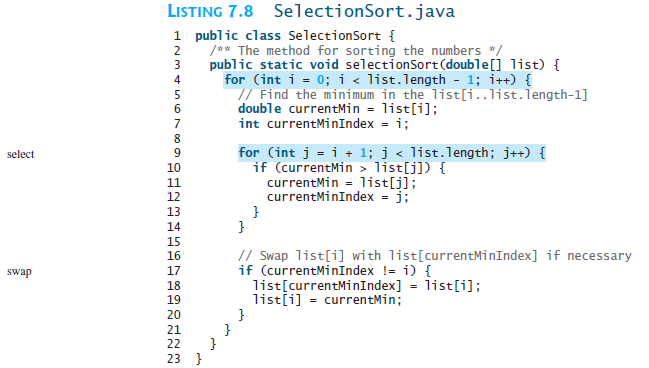




# Sorting Arrays:

* Putting the elements in order; many algorithms have been developed for this.
* Selection Sort:
  + Finds the smallest number in the array and swaps its position with the first element, then finds the next smallest and swaps it with the second position, and continues till the array is sorted.
  + Sorting list{2, 9, 5, 4, 8, 1, 6}





# Arrays class

The Arrays class contains useful methods for common array operations such as sorting and searching.

* Sort: sorts the whole array, or a partial section of the array – depending on what is passed to the method

Example:

double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};

Arrays.sort(numbers); // Sort the whole array

* binaraySearch: to search for the key in an array

Example:

int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};

System.out.println("1. Index is " +

java.util.Arrays.binarySearch(list, 11));

* equals: to check if 2 arrays are equal

Example:

int[] list1 = {2, 4, 7, 10};

int[] list2 = {2, 4, 7, 10};

int[] list3 = {4, 2, 7, 10};

System.out.println(java.util.Arrays.equals(list1, list2)); // true

System.out.println(java.util.Arrays.equals(list2, list3)); // false

* fill: to fill in all or part of the array – depending on what is passed to the method

Example:

int[] list1 = {2, 4, 7, 10};

int[] list2 = {2, 4, 7, 7, 7, 10};

java.util.Arrays.fill(list1, 5); // Fill 5 to the whole array

java.util.Arrays.fill(list2, 1, 5, 8); // Fill 8 to a partial array

* toString: returns a String that represents all the elements in the array

Example:

int[] list = {2, 4, 7, 10};

System.out.println(Arrays.toString(list));

# Command-Line Arguments:

* The main method can receive string arguments from the command line
* A main method is just a regular method that receives arguments from the command line

public static void main(String[] args) {

For example, if you are using a command line IDE, then you can type the following command to make a program called TestMain run. The values of arg0, arg1, and arg2 are the elements in the String array args.

java TestMain arg0 arg1 arg2