

Session 4- Facilitation Guide

INDEX

- I. Recap from the last session
- II. Array Declaration, Construction, and Initialization
- III. Declaring an Array
- IV. Constructing an Array
- V. Initializing an Array
- VI. One dimensional array
- VII. Sorting

For (2 hrs) ILT

I. Recap from the last session

In the last session we discussed polymorphism in detail. We explained how polymorphism helps us to use the same method for multiple purposes. Polymorphism in Java allows us to achieve different functionalities from methods that have the same name. We also discussed method and constructor overloading. We explained how different overloaded constructors can be used to initialize objects in different ways based on scenarios.

II. Array Declaration, Construction, and Initialization

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

So you got an idea of arrays but you are curious why we should use arrays when we can use other types of variables. Let us consider you are analyzing the marks of 100 students, or marks of each student in 10 subjects or salaries of 1000 employees. Either you can store them in multiple variables or store them in an array. If you store them in multiple variables then not only code will be ugly but also it will not be maintainable. In those cases we should use a data type that can hold multiple values. Array is one of those data types. We can also use Collection classes for this purpose, which will be discussed later.

Following diagram explains how the memory for array elements are allocated and value is stored.



Each item in an array is called an element, and each element is accessed by its numerical index. As shown in the preceding illustration, Java uses zero-based indexing for arrays. That means numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.

III. Declaring an Array

Similar to the declaration of normal variables , an array declaration contains two parts:

- the type of the array
- the name of the array

To declare an array in Java we should use the following syntax:

```
<ArrayType> <arrayVariableName> [ ];
```

Or

```
<ArrayType> [ ] <arrayVariableName>;
```

For example following code snippets can be used to declare an int array:

```
Java
...
// Preferred way
int[] myArray; // Square brackets are placed after the variable type

//Not preferred way of declaring an array
int anotherArray [ ]; //Square brackets are placed after the variable name

....
```

However, convention discourages the second form of placing brackets.

Arrays can be of primitive type or non-primitive data types. All the elements in the array are the same data type as declared and trying to modify any array element to another data type will cause error.

An array's name can be a valid Java identifier and it should follow the rules and conventions as previously discussed in the naming section. Here are some more examples of declaring arrays:

```
Java
// Primitive data types
```

```

byte[] anArrayOfBytes;
short[] anArrayOfShorts;
long[] anArrayOfLongs;
float[] anArrayOfFloats;

// Non-primitive data types
// Types can be name of class, interface or enums
String[] anArrayOfStrings;
MyClass[] myarray;
Month [] monthsArray;

```

IV. Constructing an Array

Constructing an array in Java refers to the process of creating an array object and allocating memory to hold a specific number of elements of a certain data type. Arrays are used to store collections of similar data items, such as integers, characters, or objects, in a contiguous block of memory.

When we declare an array, we do not specify the size of the array. To define the size of the array we use the new operator. The syntax is:

```
<datatype> <ArrayName> = new <data type> [<size of the array>];
```

Here is some examples of allocating size to arrays,

```

...
int arr [] = new int [100]; // int array of length 100
String stringArr [] = new String [5]; // String array of length 5
...

```

Array in Java is a special type of built-in object dynamically created, and may be assigned to variables of type Object. All methods of class Object may be invoked on an array. Similar to other reference type fields and methods of arrays can be accessed through “.” operator. Length of an array can be accessed through <arrayName>.length field.

Look at the following example:

```

Java
public class ArrayTest {

    public static void main(String[] args) {

        double [] myArr = new double [200];
        String [] messages = new String [15];
    }
}

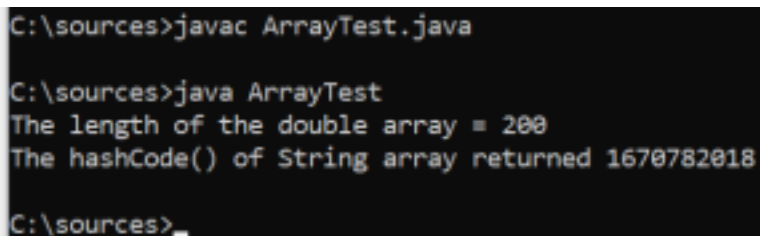
```

```

        System.out.println("The length of the double array = " +
myArr.length);
        // As arrays is also an Object (Base class of all Java class)
        // we can invoke Object's method for an array
        System.out.println("The hashCode() of String array returned " +
messages.hashCode());
    }
}

```

The output is:



```

C:\sources>javac ArrayTest.java

C:\sources>java ArrayTest
The length of the double array = 200
The hashCode() of String array returned 1670782018

C:\sources>

```

V. Initializing an Array

When we instantiate the array with a new operator all the elements will be initialized with the default values of that particular data type. Here are the default values of various Java data types.

- Numeric Types:
 - byte: 0
 - short: 0
 - int: 0
 - long: 0L
 - float: 0.0f
 - double: 0.0
- Boolean Type:
 - boolean: false
- Character Type:
 - char: '\u0000' (null character)
- Reference Types:
 - Objects (including arrays): null

It's important to note that these default values are assigned when arrays are declared and size is specified with the new operator but not explicitly initialized. Our next example will show how to create an array and allocate memory with the new operator. We will not initialize the values of array elements.

Java

```

public class ArrayInitialization {

    public static void main(String[] args) {

```

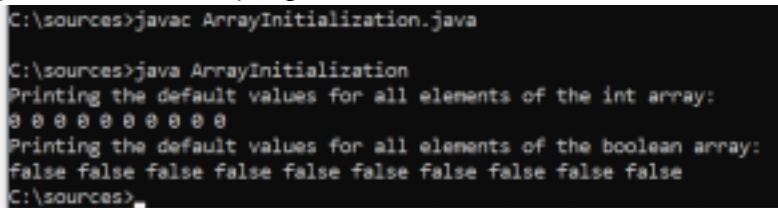
```

        int[] arr=new int[10];

        System.out.println("Printing the default values for all
elements of the int array:");
        for(int i=0;i<arr.length;i++)
        {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
        boolean[] boolArr = new boolean[10];
        System.out.println("Printing the default values for all
elements of the boolean array:");
        for(int i=0;i<arr.length;i++)
        {
            System.out.print(boolArr[i] + " ");
        }
    }
}

```

The output of the above program is:



```

C:\sources>javac ArrayInitialization.java

C:\sources>java ArrayInitialization
Printing the default values for all elements of the int array:
0 0 0 0 0 0 0 0 0 0
Printing the default values for all elements of the boolean array:
false false false false false false false false
C:\sources>_

```

To initialize the values of the array we can assign values to each element. To access or assign the value of a particular element in an array we should use the square brackets and index of the array element. So to access the 5th element in an array named myArr we will use the following expression.

Java

```

...
int myArr[] = new int [10];
myArr [4] =35; // Java uses 0 based index
...

```

In this Demo class we will explain how to initialize the values of array elements.

Java

```

import java.util.Scanner;
public class ArrayDemo {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int[] arr=new int[5];
        System.out.println("Enter 5 elements in an array:");
        for(int i=0;i<arr.length;i++)
        {
            arr[i]=sc.nextInt();
        }
        System.out.println("The elements are:");
        for(int i=0;i<arr.length;i++)
        {
            System.out.println(arr[i]);
        }
    }
}

```

We can also avoid using the new operator and assign values directly. Following example will explain how to initialize the values of an array without using the new operator or without specifying size of the array.

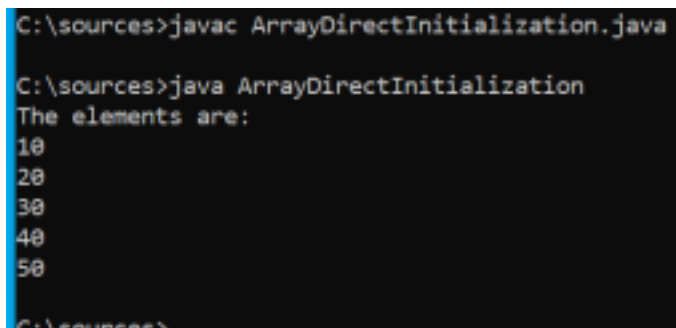
```

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

        int[] arr= {10,20,30,40,50};
        System.out.println("The elements are:");
        for(int i=0;i<arr.length;i++)
        {
            System.out.println(arr[i]);
        }
    }
}

```

The output will be:



```

C:\sources>javac ArrayDirectInitialization.java
C:\sources>java ArrayDirectInitialization
The elements are:
10
20
30
40
50
C:\sources>

```

Array of objects

The array of objects, as defined by its name, stores an array of objects. An object represents a single record in memory, and thus for multiple records, an array of objects

must be created. It must be noted that the arrays can hold only references to the objects, and not the objects themselves.

Let us see how we can declare an Array of objects in Java. We use the class name Object, followed by square brackets to declare an Array of Objects. Object[]
JavaObjectArray;

Another declaration can be as follows:

Object JavaObjectArray[];

In the following example we will create multiple Book objects and store them in an object array.

Java

```
import java.util.Scanner;
class Book
{
    int bookId;
    String bookName;
    float bookPrice;
    int pages;
    // Getter and setter methods
    public int getBookId() {
        return bookId;
    }
    public void setBookId(int bookId) {
        this.bookId = bookId;
    }
    public String getBookName() {
        return bookName;
    }
    public void setBookName(String bookName) {
        this.bookName = bookName;
    }
    public float getBookPrice() {
        return bookPrice;
    }
    public void setBookPrice(float bookPrice) {
        this.bookPrice = bookPrice;
    }
    public int getPages() {
        return pages;
    }
    public void setPages(int pages) {
```

```

        this.pages = pages;
    }
    public Book(int bookId, String bookName, float bookPrice, int
        pages) { this.bookId = bookId;
        this.bookName = bookName;
        this.bookPrice = bookPrice;
        this.pages = pages;
    }
    public Book() {
    }
    //
    void display()
    {
        System.out.println(" Book Id: "+this.bookId+
            "\n Book Name: "+this.bookName+
            "\n Book Price:"+this.bookPrice+
            "\n Book Pages: "+this.pages);
    }
}
public class ArrayOfObjectEx {
    public static void main(String[] args) {
        Book books[]=new Book[10];
        int size;
        int id;
        String bookName;
        float bookPrice;
        int pages;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter how many book you want to
enter:");
        size=sc.nextInt();
        System.out.println("Enter "+size+" book details in an
array!!");
        for(int i=0;i<size;i++)
        {
            System.out.println("Enter book id: ");
            id=sc.nextInt();
            sc.nextLine();
            System.out.println("Enter book Name: ");
            bookName=sc.nextLine();
            System.out.println("Enter book price: ");
            bookPrice=sc.nextFloat();
            System.out.println("Enter book pages: ");
            pages=sc.nextInt();
            //add details in the array
            books[i]=new Book(id, bookName, bookPrice, pages);

```



```
System.out.println("=====");
    }
    System.out.println("=====");
    System.out.println("All Book details:");
    for(int i=0;i<size;i++)
    {
        books[i].display();

    System.out.println("=====");
    }
}
}
```

Output:

```

C:\sources>javac  ArrayOfObjectEx.java

C:\sources>java ArrayOfObjectEx
Enter how many book you want to enter:
2
Enter 2 book details in an array!!
Enter book id:
123
Enter book Name:
Java for Dummies
Enter book price:
200
Enter book pages:
300
=====
Enter book id:
190
Enter book Name:
Python Made Easy
Enter book price:
600
Enter book pages:
340
=====
=====
All Book details:
Book Id: 123
Book Name: Java for Dummies
Book Price:200.0
Book Pages: 300
=====
Book Id: 190
Book Name: Python Made Easy
Book Price:600.0
Book Pages: 340
=====

C:\sources>

```

VI. One dimensional array

Arrays can be one dimensional or multidimensional. A one-dimensional array in Java is a linear collection of elements of the same data type. It is a fundamental data structure that allows you to store and manipulate multiple values under a single variable name. We already explained that each element in a one-dimensional array is accessed using an index, which starts from 0 for the first element and increments by 1 for each subsequent element.

Here is some examples of one dimensional arrays:

```
...  
int[] numbers = {10, 20, 30, 40, 50};  
String [] currenaceis = {"INR", "USD", "GBP", "SGD"};  
...
```

One-dimensional arrays are commonly used to store lists of values, such as temperatures, exam scores, or other types of data. They provide a simple way to manage and manipulate collections of data of fixed length.

One-dimensional arrays are indeed versatile and provide a straightforward way to work with collections of data. However, it's important to note that one-dimensional arrays have a fixed size once they are created, which might limit their flexibility in some scenarios. For more dynamic storage needs, Java offers other data structures like ArrayLists or LinkedLists. (Will be discussing these later.)

VII. Sorting

Sorting is one of the most common as well as useful operations done on arrays. Suppose we want to select 10 best students from the class for sending them to an inter-school quiz competition. Then the school authority might sort the students according to their marks obtained in their class examination.

Sorting is the process of arranging data in a particular order. Computer data is a set of records consisting of one or more fields. In order to use data effectively and perform various operations such as searching, accessing etc., it is necessary to arrange the data in a certain order. For instance, if there are many records of student data then we could arrange the data according to the student id or the student name. This process is called sorting. Generally, sorting is performed to use data more effectively and easily.

There are multiple sorting algorithms that we can use depending on our needs. Either we can write our own sorting program or use the existing Java library to sort arrays or other Java collection items. Each sorting algorithm has its own computational time and space complexity which can be explained using BigO notation.

(**Notes:** There are many online resources to learn BigO notation. One such page is https://web.mit.edu/16.070/www/lecture/big_o.pdf)

There is no best sorting algorithm for all cases. To find out the best sorting algorithm for a particular set of data we should consider:

- Size of data - Only a few items, or a large set of data
- Are items mostly sorted already?
- Concerned about worst-case scenarios
- Interested in a good performance in average-cases
- Simple and less code.

We will now discuss two very popular and simple sorting algorithms. These two sorting

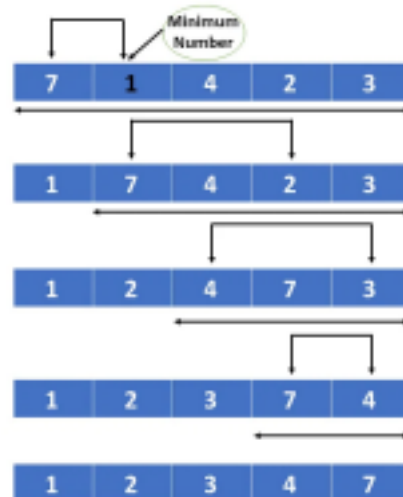
algorithms are not efficient for a large set of data, but they are simple, easy to learn and do not require additional memory. They are - Selection sort and Bubble sort. These sorting techniques are simple, yet they are effective sorting algorithms. When sorting a small amount of data, they can be used effectively. Besides this, there are many other effective sorting algorithms available like - merge sort, quick sort, insertion sort, heapsort etc.

Selection Sort:

Selection sort is the one of the simplest sorting algorithms. In the selection sort technique, the smallest element of the field is chosen and then the first element and this smallest element are swapped. Next, the second smallest element is found out and then exchanged with the second element of the array.

In this sorting, the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list. The time Complexity of this sorting method is $O(n^2)$ in BigO notation.

Let us take an input array [7, 1, 4, 2, 3].



So after the first pass we will see that the minimum element is 1. So minimum element 1 is replaced with the first element 7. So the array will become [1, 7, 4, 2, 3] after the first pass. Now the array has been divided into two subarrays [1] and [7, 4, 2, 3]. Similarly we will continue with [7, 4, 2, 3] and find that the minimum element is 2. So the second element 7 will be replaced with 2. The array will become [1, 2, 4, 7, 3]. The same process will be followed until the second there is no element in the second subarray and the array will become [1, 2, 3, 4, 7].

Here is the Java program that sorts the above array using the selection sort technique:

```
Java
import java.util.*;
/**
 *
 * Class to demonstrate selection sorting technique
```

```

*
*/
public class SelectionSort {
    // This method sort the input array using selection sort
    static void sortArray(int numArray[])
    {
        int n = numArray.length;
        // Traverse unsorted array
        for (int i = 0; i < n-1; i++)
        {
            // Find the minimum element in unsorted array
            int min_idx = i;
            for (int j = i+1; j < n; j++)
                if (numArray[j] < numArray[min_idx])
                    min_idx = j; // min_idx gets the index of the
Minimum element

            //Finding minimum element completed
            // Swap minimum element with compared element
            int temp = numArray[min_idx];
            numArray[min_idx] = numArray[i];
            numArray[i] = temp;
        }
    }
    public static void main(String args[])
    {
        //Declare and print the original array
        int numArray[] = {7,1,4,2,3};
        System.out.println("Original Array:" + Arrays.toString(numArray));
        //Call selection sort method
        sortArray(numArray);
        //Print the sorted array
        System.out.println("Sorted Array:" + Arrays.toString(numArray));
    }
}

```

The out will display the both input array and sorted array.

```

C:\sources>javac SelectionSort.java

C:\sources>java SelectionSort
Original Array:[7, 1, 4, 2, 3]
Sorted Array:[1, 2, 3, 4, 7]

C:\sources>

```

Bubble Sort:

Bubble Sort is the elementary sorting algorithm used for sorting a small set of data. It is a comparison-based sorting algorithm in which each element is compared with the next

element, and is swapped if those elements are not in the correct order. In case of Selection sort, swapping is performed on the minimum value of the subarray. But in bubble sort, swapping takes place whenever any element is smaller than the next element.

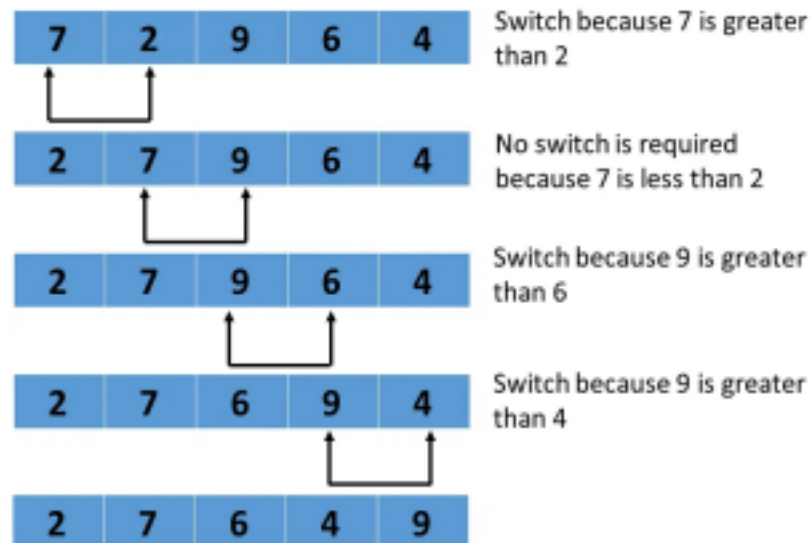
- The algorithm starts with comparing the first pair of elements.
- If the first element is smaller than the second element, then the elements are swapped
- This algorithm is not suitable for large data sets

The time Complexity of this sorting method is $O(n^2)$ in BigO notation.

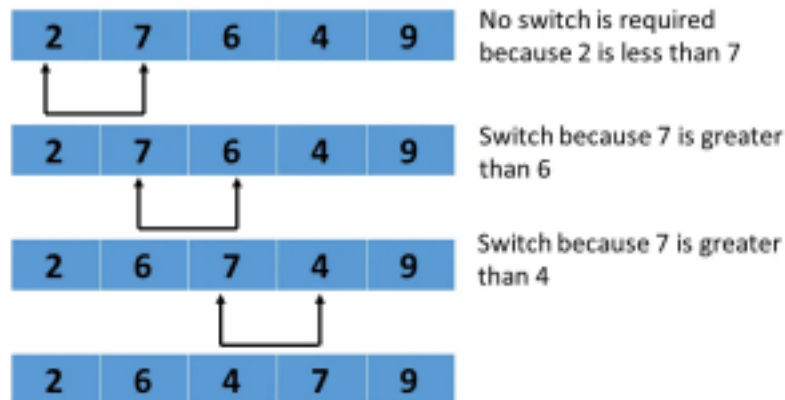
Here is an example to explain the bubble sort algorithm. Let us first start with the input array [7, 2, 9, 6, 4].

First Pass

Here is the first pass of bubble sort.



Second Pass



This process will continue and finally the sorted array will be:
[2, 4, 6, 7, 9].

Here is a Java program which follows exactly the above steps and sorted the array.

Java

```
//Bubble sort
//Time Complexity : O(N^2)
//Space Complexity : O(1)
public class BubbleSort
{
    static void bubbleSort(int a[])
    {
        int len = a.length;
        for (int i = 0; i < len-1; i++)
            for (int j = 0; j < len-i-1; j++)
                if (a[j] > a[j+1]) //comparing the pair of elements
                {
                    // swapping a[j+1] and a[i]
                    int temp = a[j];
                    a[j] = a[j+1];
                    a[j+1] = temp;
                }
    }
    /* Prints the array */
    static void printArray(int a[])
    {
        int len = a.length;
        for (int i = 0; i < len; i++)
            System.out.print(a[i] + " "); //print the sorted array
        System.out.println();
    }
}
```

```

// Main method to test the sorting
public static void main(String args[])
{
    int arr[] = {7, 2, 9, 6, 4};
    System.out.println("Initial array is:");
    printArray(arr); //Calling the printArray function
    bubbleSort(arr); //Calling the bubbleSort function
    System.out.println("Sorted array is:");
    printArray(arr); //Calling the printArray function
}
}

```

The output is:

```

C:\sources>javac BubbleSort.java

C:\sources>java BubbleSort
Initial array is:
7 2 9 6 4
Sorted array is:
2 4 6 7 9

C:\sources>

```

Java's Built-in Sorting Library:

To sort an array using Java's built-in library we can use `Arrays.sort()` method. Let us go through an example to sort an array of integers and Strings to understand this utility.

Java

```

import java.util.Arrays;
public class ArraySorting {

    public static void main(String[] args) {

        //Declare and initialize an int array
        int [] arr = {56, 32, 12, 1, 80, 2};

        // Arrays class in java's java.util package has a
        // sort() method that sort arrays of primitives and Objects

        Arrays.sort(arr);
        // Array arr is already sorted.
        System.out.println(Arrays.toString(arr)); // Print sorted array

        String [] currencies = {"INR", "USD", "GBP", "SGD"};
        Arrays.sort(currencies); // Sorts alphabetically
        System.out.println(Arrays.toString(currencies)); // Print
        alphabetically

    }
}

```


Output:

```
C:\sources>javac ArraySorting.java

C:\sources>java ArraySorting
[1, 2, 12, 32, 56, 80]
[GBP, INR, SGD, USD]

C:\sources>
```

After the 2 hr ILT, the student has to do a 1 hour live lab. Please refer to the Session 4 Lab doc in the LMS.