Todays Agenda: 10/04/2024

|  |
| --- |
| **Arrays in Java** |
| Definition and characteristics |
| Implement Basic operations |
| Access |
| Insert |
| Delete |
| Update |
| Reverse |
| Slice |
| Sorting an Array - Brute Force Method |
| Searching an Array - Linear Search Method |

Definition and characteristics:

Java Arrays

**Java array** is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.



int[] arr = new int[10];

0-9; arr[0]=1; arr[1] = ‘a’, arr[2]=”Sam”, arr[3]=2.0

int x =arr[1];

arr[6]=53;

### **Advantages**

* **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
* **Random access:** We can get any data located at an index position.

### **Disadvantages**

* **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

### **Types of Array in java**

There are two types of array.

* Single Dimensional Array
* Multidimensional Array

## **Single Dimensional Array in Java**

**Syntax to Declare an Array in Java**

1. dataType[] arr; (or)  //int[] arr;
2. dataType []arr; (or)  //int []arr;
3. dataType arr[];  //int arr[];
4. int[] arr = new int[3];
5. int []arr;
6. int arr[]

**Instantiation of an Array in Java**

1. arrayRefVar=**new** datatype[size];

### **Example of Java Array**

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

//Java Program to illustrate how to declare, instantiate, initialize

//and traverse the Java array.

**class** Testarray{

**public** **static** **void** main(String args[]){

**int** a[]=**new** **int**[5];//declaration and instantiation

a[0]=10;//initialization

a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

//traversing array 5

**for**(**int** i=0;i<a.length;i++)//length is the property of array

System.out.println(a[i]);

}}

Output:

10

20

70

40

50

## **Declaration, Instantiation and Initialization of Java Array**

We can declare, instantiate and initialize the java array together by:

**int** a[]={33,3,4,5};//declaration, instantiation and initialization

## **For-each Loop for Java Array**

**for**(data\_type variable:array){

//body of the loop

}

## **ArrayIndexOutOfBoundsException**

The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException if length of the array in negative, equal to the array size or greater than the array size while traversing the array.

## **2) Multidimensional Array in Java(2-D array)**

In such case, data is stored in row and column based index (also known as matrix form).

**Syntax to Declare Multidimensional Array in Java**

1. dataType[ ][ ] arrayRefVar; (or)  int[ ][ ] arr;
2. dataType [ ][ ]arrayRefVar; (or)  int [][]arr;
3. dataType arrayRefVar[][]; (or)   int arr[][];
4. dataType []arrayRefVar[];   int []arr[];

**Example to instantiate Multidimensional Array in Java**

**int**[][] arr=**new** **int**[3][3];//3 row and 3 column

**Example to initialize Multidimensional Array in Java**

arr[0][0]=1;

arr[0][1]=2;

arr[0][2]=3;

arr[1][0]=4;

arr[1][1]=5;

arr[1][2]=6;

arr[2][0]=7;

arr[2][1]=8;

arr[2][2]=9;

### **Example of Multidimensional Java Array**

//Java Program to illustrate the use of multidimensional array

**class** Testarray3{

**public** **static** **void** main(String args[]){

//declaring and initializing 2D array

**int** arr[][]={{1,2,3},//0

{2,4,5},//1

{4,4,5}};  //2

//0,1,2

//printing 2D array

**for**(**int** i=0;i<3;i++){

**for**(**int** j=0;j<3;j++){

   System.out.print(arr[i][j]+" ");  //0,0//0,1///0,2////123

 }

 System.out.println();

}

}}

Output:

1 2 3

2 4 5

## **Basic operations**

Now, let's discuss the basic operations supported in the array -

* Traversal - This operation is used to print the elements of the array.
* Insertion - It is used to add an element at a particular index.
* Deletion - It is used to delete an element from a particular index.
* Search - It is used to search an element using the given index or by the value.
* Update - It updates an element at a particular index.

### **Traversal operation**

This operation is performed to traverse through the array elements. It prints all array elements one after another. We can understand it with the below program –

public class TravesalArray {

public static void main(String[] args) {

int[] arr = {18, 30, 15, 70, 12}; //size-5,last index-4

System.out.println("Elements of the array are:");

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

System.out.printf("arr[%d] = %d, ", i, arr[i]);

}

}}

Output:

Array in DS

### **Insertion operation**

This operation is performed to insert one or more elements into the array. As per the requirements, an element can be added at the beginning, end, or at any index of the array. Now, let's see the implementation of inserting an element into the array.

public class InsertArray {

public static void main(String[] args) {

int[] arr = {20, 30, 15, 80, 12,70,100};//20, 30,60, 15, 80, 12,70

int x = 60; // element to be inserted

int pos = 3;///{18

int size = 7;

System.***out***.println("Array elements before insertion");

for (int k = 0; k < size; k++) {

System.***out***.print(arr[k] + " ");

}

System.***out***.println();

for (int i = size - 1; i >= pos; i--) { //6//3

arr[i] = arr[i - 1];

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

}//arr[6]

arr[pos - 1] = x;

System.***out***.println("Array elements after insertion");

for (int i = 0; i < size; i++)

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

System.***out***.println();

}

}

**Output**

Array elements before insertion

20 30 15 80 12 70 100

70 12 80 15 Array elements after insertion

20 30 60 15 80 12 70

### **Deletion operation**

As the name implies, this operation removes an element from the array and then reorganizes all of the array elements.

public class DeleteArray {

public static void main(String[] args) {

int[] arr = {18, 30, 15, 70, 12};

int k = 30;

int n = 5;

int i, j = 0,o = 0;

int[] b =new int[n-1];

System.***out***.println("Given array elements are:");

for (i = 0; i < n; i++) {

System.***out***.printf("arr[%d] = %d, ", i, arr[i]);

}

System.***out***.println();

for (i = 0; i < n; i++) {

if(arr[i]==k)

j=i;

}

for(int d=0;d<n;d++) {

if(d!=j) {

b[o] = arr[d];

o++;

}

}

//System.out.println("\nElements of array after deletion:");

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

System.***out***.printf("arr[%d] = %d, ", i, b[i]);

}

}

}

**Output**

Given array elements are:

arr[0] = 18, arr[1] = 30, arr[2] = 15, arr[3] = 70, arr[4] = 12,

arr[0] = 18, arr[1] = 15, arr[2] = 70, arr[3] = 12,

### **Search operation**

This operation is performed to search an element in the array based on the value or index.

public class SearchingInArray {

public static void main(String[] args) {

int[] arr = {18, 30, 15, 70, 12};

int item = 70;

int j = 0;

System.out.println("Given array elements are:");

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

System.out.printf("arr[%d] = %d, ", i, arr[i]);

}

System.out.println("\nElement to be searched = " + item);

while (j < arr.length) {//5// j=3<5

if (arr[j] == item) {// arr[3] = 70=70

break;

}

j = j + 1;

}

if (j == arr.length) {

System.out.println("Element " + item + " is not found in the array");

} else {

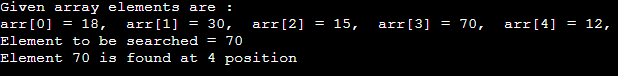
System.out.println("Element " + item + " is found at position " + (j + 1));

}

}

}

**Output**



### **Update operation**

This operation is performed to update an existing array element located at the given index.

public class UpdateArray {

public static void main(String[] args) {

int[] arr = {18, 30, 15, 70, 12};

int item = 50;

int pos = 3;

System.out.println("Given array elements are:");

for (int i = 0; i < 5; i++) {

System.out.printf("arr[%d] = %d, ", i, arr[i]);

}

arr[pos - 1] = item;

System.out.println("\nArray elements after updation:");

for (int i = 0; i < 5; i++) {

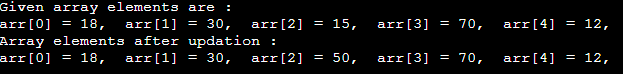
System.out.printf("arr[%d] = %d, ", i, arr[i]);

}

}

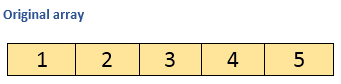
}

**Output**

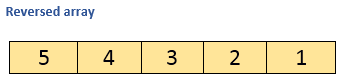


## **Reverse an Array:**

In this program, we need to print the elements of the array in reverse order that is; the last element should be displayed first, followed by second last element and so on.



Above array in reversed order:



### **Algorithm**

* **STEP 1:** START
* **STEP 2:** INITIALIZE arr[] = {1, 2, 3, 4, 5}
* **STEP 3:** PRINT "Original Array:"
* **STEP 4:** REPEAT STEP 5 for(i=0; i<arr.length ; i++)
* **STEP 5:** PRINT arr[i]
* **STEP 6:** PRINT "Array in reverse order"
* **STEP 7:** REPEAT STEP 8 for(i= arr.length-1; i>=0; i--)
* **STEP 8:** PRINT a[i]
* **STEP 9:** END

### **Program:**

**public** **class** ReverseArray {

**public** **static** **void** main(String[] args) {

        //Initialize array

**int** [] arr = **new** **int** [] {1, 2, 3, 4, 5};

        System.out.println("Original array: ");

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

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

        }

        System.out.println();

        System.out.println("Array in reverse order: ");

        //Loop through the array in reverse order

**for** (**int** i = arr.length-1; i >= 0; i--) {

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

        }

    }

}

//int [] arr = new int [] {1, 2, 3, 4, 5}; //5/2=2//

int mid = arr.length/2;//2//6/2=3

if(arr.length%2==0) {

for(int i=mid;i>=0;i--) {

int temp = 0;

//arr[mid] =

//System.out.println();

}

}else {

for(int i=mid-1;i>=0;i--) {

int temp = 0;//1

temp = arr[i];

arr[i]= arr[arr.length-i];

arr[arr.length-i]=temp;

}

for(int q :arr) {

System.***out***.println(q);

}

}

**Output:**

Original array:

1 2 3 4 5

Array in reverse order:

5 4 3 2 1

# **Array Slicing in Java**

In Java, array **slicing** is a way to get a subarray of the given array. Suppose, a[] is an array. It has 8 elements indexed from a[0] to a[7].

a[] = {8, 9, 4, 6, 0, 11, 45, 21}

Now, we want to find a slice of the array index from a[3] to a[6]. Where a[3] is the startIndex and a[6] is the endIndex. Therefore, we get the following **sliced array**:

a[] = {6, 0, 11, 45}

## **By Copying Elements**

In this method, first, we find the start and end index of the given array. After that, we create an empty array (sliced array) of size **(endIndex - startIndex)+1.** From the given array, copy the elements (from startIndex) to the sliced array. At last, print the sliced array.

Let's implement the above approach in a [Java](https://www.javatpoint.com/java-tutorial) program to get a sliced array of the given array. In this program. we will use an array of primitive types.

**SliceArrayExample1.java**

**import** java.util.Arrays;

**public** **class** SliceArrayExample

{

//creating a functiion to the slice of an array

**public** **static** **int**[] getSlice(**int**[] array, **int** startIndex, **int** endIndex)

{

// Get the slice of the Array

**int**[] slicedArray = **new** **int**[endIndex - startIndex];

//copying array elements from the original array to the newly created sliced array

**for** (**int** i = 0; i < slicedArray.length; i++)

{

slicedArray[i] = array[startIndex + i];

}

//returns the slice of an array

**return** slicedArray;

}

//main() method

**public** **static** **void** main(String args[])

{

//array from which we will find the slice

**int**[] array = {23, 56, 78, 22, 45, 90, 67, 91, 0, 31};

//start index and end index denotes the part of the original array to be slice

**int** startIndex = 3, endIndex = 8;

//Get the slice of the array

**int**[] slicedArray = getSlice(array, startIndex, endIndex + 1);

//prints the slice array

System.out.println("Slice of Array: "+Arrays.toString(slicedArray));

}

}

**Output:**

Slice of Array: [22, 45, 90, 67, 91, 0]

## **Complexity of Array operations**

Time and space complexity of various array operations are described in the following table.

**Time Complexity**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Average Case** | **Worst Case** |
| Access | O(1) | O(1) |
| Search | O(n) | O(n) |
| Insertion | O(n) | O(n) |
| Deletion | O(n) | O(n) |

**Space Complexity**

In array, space complexity for worst case is **O(n)**.

Sorting an Array - Brute Force Method

### **Brute Force Approach**

The most common and easy approach is Brute force technique. The Brute force approach uses straight forward pattern which will not be bothering about the time complexity at all. The algorithm will simply iterates through all the elements and compare to their adjacent element. If the algorithm finds that the elements adjacent to each other are in wrong order, then it will swap it.

Example : Bubble sort, insertion sort, selection sort, etc.

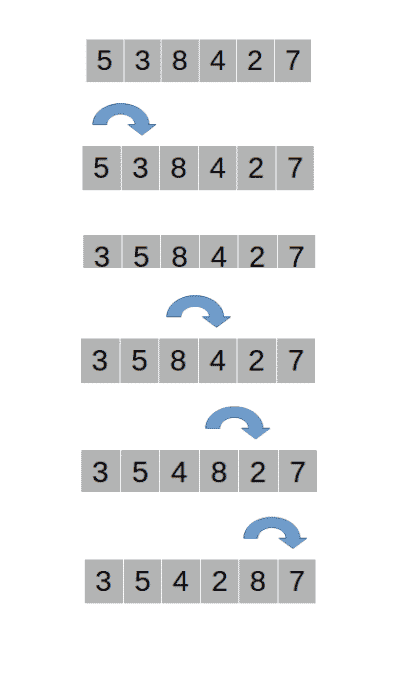
### **Bubble Sort**

Bubble sort is the easiest and simplest sorting algorithm which iterates through each element and swaps with its adjesent element if they were in wrong order. Bubble sort uses the Brute force approach to sort the elements.

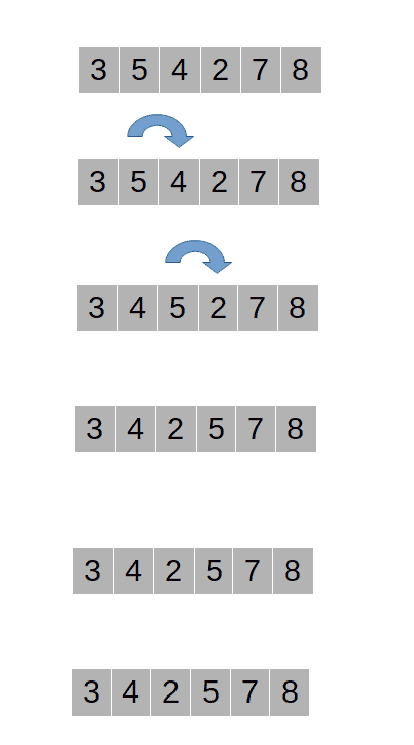
The iterating of elements will happen for multiple passes with each time ends with one element less (i.e 1st pass ends with nth element and 2nd pass will ends with (n-1)th element and so on).

The algorithm will move the bigger elements towards the last of the list, where the other brute force algorithms will pull the smallest number towards starting of the array. There are many types of sorting algorithms which performs the sorting in same time complexity as the Bubble sort but among them, the Bubble sort algorithm will be very much easy to implement.

PASS1:



**Pass #2**



#### **Algorithm**

public class BubbleSortSamp {

public static void bubbleSort(int arr[]) {

int temp, n = arr.length;

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n - i-1; j++) {//6-0

if (arr[j] > arr[j + 1]) {

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

public static void main(String args[]) {

int arr[] = { 5, 3, 4, 8, 7, 6, 9 };

*bubbleSort*(arr);

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

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

}

}

}

Output:

3, 4, 5, 6, 7, 8, 9

Similarly, the algorithm will loop until 5 Passes irrespective of whether the array is already sorted or not.

#### **Time complexity**

Best Case

The best case time complexity can be determined by providing a list of elements that are already sorted.

Eg: 1,2,3,4,5

The algorithm will take N x N number of iterations with zero swapping operation as there are no elements in the wrong order.

**Time Complexity**: O(N2)

Average/Worst Case

The average case time complexity can be determined by providing a list of elements that are partially sorted, likewise the worst case time complexity can be determined by providing a list of elements that are exactly in reverse order.

# **Linear Search in Java**

Linear search is used to search a key element from multiple elements. Linear search is less used today because it is slower than binary search and hashing.

**Algorithm:**

Step 1: Traverse the array

* Step 2: Match the key element with array element
* Step 3: If key element is found, return the index position of the array element
* Step 4: If key element is not found, return -1

Let's see an example of linear search in java where we are going to search an element sequentially from an array.

**public** **class** LinearSearchExample{

**public** **static** **int** linearSearch(**int**[] arr, **int** key){

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

**if**(arr[i] == key){

**return** i;

            }

        }

**return** -1;

    }

**public** **static** **void** main(String a[]){

**int**[] a1= {10,20,30,50,70,90};

**int** key = 50;

        System.out.println(key+" is found at index: "+linearSearch(a1, key));

    }

}

Output:

50 is found at index: 3