**Day-01**

**10-03-2025**

**=================**

**Arrays:**

**=====**

How variables internally work?

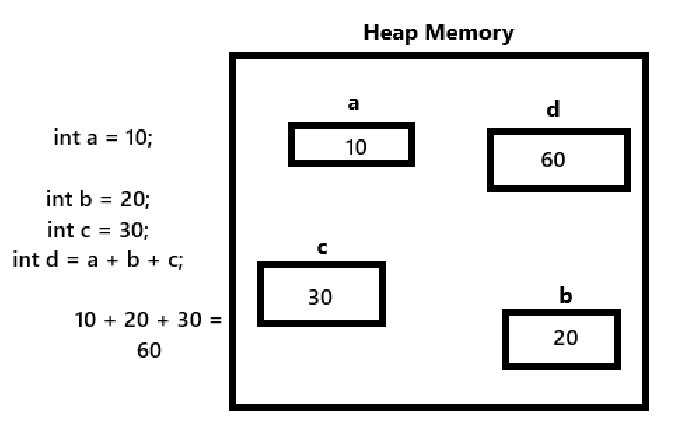
----------------------------------

->

int a= 10;

when we can initialize the variable,

the JVM can create the memory block for variable with the size based on the datatype then store the assigned value into that memory block.



Why Arrays?

---------------

-> When we have more number of variables in the java program it can increase the complexity or damage the processing time.

More variables can take more processing time.

-> To overcome this drawback we can use arrays.

-> When we want to store more than one value with same type into the single variable, we can use "arrays".

Array Definition:

-----------------

Array is one of the reference datatype.

It is a collection of similar (same datatype) data items.

Array Declaration:

---------------------

Syntax:

datatype array-name[] = new datatype[size];

or

datatype[] array-name = new datatype[size];

How Array can internally work?

-----------------------------------

After the declaration of an array, the JVM can create the memory in the heap with the size of (number-of-elements X size of datatype).

Ex: int[] a = new int[5];

in this case, the JVM can create 20-bytes of memory in the heap area.

and the total 20-bytes can divide into 5-blocks.

Each block with 4-bytes of size (int size).

And each block can be represented with an index value.

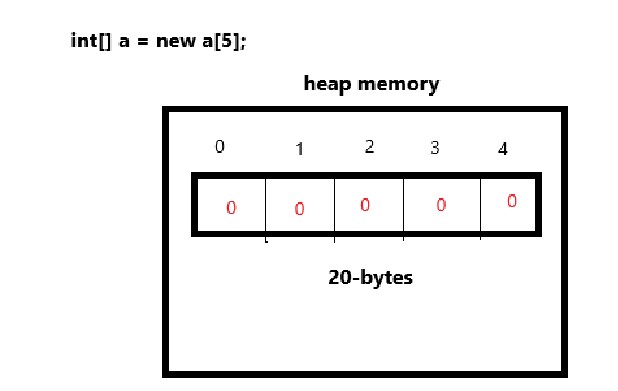
-> When we have not assigned any values for the array which we have declared then, the JVM can store default values according to the specified datatype.

Ex: if the array is integer type:

default value = 0

String type ==> null

boolean type ==> false etc.



-> To access individual array elements, we can use an index.

Syntax:

array-name[index];

public class ArrayDeclaration {

public static void main(String[] args) {

int a[] = new int[7];

char[] b = new char[5];

String[] c = new String[4];

System.out.print(a[0]);

System.out.print(a[1]);

System.out.print(a[2]);

System.out.print(a[3]);

System.out.print(a[4]);

System.out.print(a[5]);

System.out.print(a[6]);

System.out.println();

System.out.print(b[0]);

System.out.print(b[1]);

System.out.print(b[2]);

System.out.print(b[3]);

System.out.print(b[4]);

System.out.println();

System.out.println(c[0]);

}

}

**Day-02**

**11-03-2025**

**================**

Array Declaration:

--------------------

datatype array-name[] = new datatype[size];

datatype[] array-name = new datatype[size];

datatype array-name[];

array-name = new datatype[size];

public class ArrayDeclaration {

public static void main(String[] args) {

String fruits[];

fruits = new String[4];

System.out.println(fruits[0]);

}

}

**length Vs length()**

**============**

length:

-------

-> an attribute

we can use to find/calculate the length or size of the array

Syntax:

array-name.length;

length():

---------

-> a method

-> we can use this to get the size or length or total number of characters of string.

Syntax:

string-name.length();

Null Pointer Exception:

-------------------------

-> It is one of the Java run-time error.

-> The run-time error is also called as "Exception".

-> When the string is "null" and we are defining some operation then the JVM can throw "NullPointerException".

public class ArrayLength {

public static void main(String[] args) {

byte[] a1 = new byte[7];

float a2[] = new float[6];

double a3[];

a3 = new double[8];

boolean a4[] = new boolean[4];

char a5[] = new char[5];

String a6[] = new String[9];

String a7 = "Java programming language";

String a8 = null;

String a9 = "null";

System.out.println("size of a1 = "+a1.length);

System.out.println("size of a2 = "+a2.length);

System.out.println("size of a3 = "+a3.length);

System.out.println("size of a4 = "+a4.length);

System.out.println("size of a5 = "+a5.length);

System.out.println("size of a6 = "+a6.length);

System.out.println("The size of a7 = "+a7.length());

// System.out.println("The size of a8 = "+a8.length());

System.out.println("The size of a9 = "+a9.length());

}

}

Array Initialization

--------------------

-> Arrays can be initialized in two ways:

1) Compile-time initialization/Static Array Initialization

-------------------------------------------------------------

-> If a programmer have specified the size of the array at the time of declaration of an array is called as "Static array".

Syntax:

datatype array-name[] = new datatype[size];

here: value for the size must be defined here only

Ex: int a[] = new int[7];

Syntax for the static array:

datatype[] array-name = new datatype[]{v1, v2, v3, ....};

Here:

the size of the array depend on the total number of elements which we have assigned.

public class CArrayPractice {

public static void main(String[] args) {

char ch[] = new char[] {'a','e','i','o','u'};

char ch1[];

ch1 = new char[] {'1','2','3','4'};

System.out.println("The size = "+ch.length);

System.out.println("The size = "+ch1.length);

}

}

How the array can behave during the initialization?

---------------------------------------------------------

i) if number of elements of the array is less than the given size:

-----------------------------------------------------------------------

the elements according to the index can store and remaining fill with default values based on the specified datatype.

ii) if number of elements == size of the array

----------------------------------------------------

the elements can fit into whole memory.

iii) if number of elements > size of the array

--------------------------------------------------

Array index out of bounds exception

-----------------------------------------

-> it is one of the run-time error

-> when we can give more number of elements than the given size, we can get "ArrayIndexOutOfBoundsException"

public class CompileTimeArray {

public static void main(String[] args) {

int size1 = 10;

int size2 = 5;

int size3 = 3;

int a[] = new int[size1];

int b[] = new int[size2];

boolean c[] = new boolean[size3];

a[0] = 11;

a[1] = 22;

a[2] = 33;

b[0] = 1;

b[1] = 3;

b[2] = 5;

b[3] = 7;

b[4] = 9;

c[0] = true;

c[1] = false;

c[2] = true;

c[3] = false;

System.out.println(a.length);

// to access individual elements of array

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

System.out.print(a[i]+"\t");

}

System.out.println();

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

System.out.print(b[i]+"\t");

}

System.out.println();

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

System.out.print(c[i]+"\t");

}

System.out.println();

}

}

2) Run-time initialization/Dynamic Array Initialization

-----------------------------------------------------------

-> If a programmer can provide the size of the array using Scanner object/while running of the program is called as "Dynamic array".

Syntax:

Scanner scan = new Scanner(System.in);

datatype[] array-name = new datatype[size];

int size = scan.nextInt();

**Day-03**

**12-03-2025**

**=================**

Dynamic Array Creation:

---------------------------

-> During the execution time, if the size of the array has defined with scanner object that array is called as "Dynamic array".

Syntax:

Scanner scan = new Scanner(System.in);

int size = scan.nextInt();

datatype[] array-name = new datatype[size];

**/\***

**\* Write a java program to accept the size of the array as an input.**

**\* And display the array elements of the defined array with specified size.**

**\*/**

import java.util.Scanner;

public class DynamicArray {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter the size of the array:");

int size = scan.nextInt();

dynamicArray(size);

}

private static void dynamicArray(int size) {

Scanner scan1 = new Scanner(System.in);

double[] arr = new double[size];

// array definition

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

System.out.println("Enter an element for the index "+i);

arr[i] = scan1.nextDouble();

}

// array accessing

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

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

}

System.out.println();

}

}

**/\***

**\* Write a java program to accept the array as an input.**

**\* And find the sum of even elements and odd elements separately.**

**\*/**

import java.util.Scanner;

public class SumOfEvenOdd {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter the size of the array:");

int size = scan.nextInt();

int[] a = new int[size];

System.out.println("Enter the array:");

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

a[i] = scan.nextInt();

}

sumOfEvenOdd(a,size);

}

public static void sumOfEvenOdd(int[] arr, int n) {

int s\_even = 0,s\_odd = 0;

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

if(arr[i] % 2 == 0) {

s\_even = s\_even + arr[i];

}

else {

s\_odd = s\_odd + arr[i];

}

}

System.out.println("The sum of even elements of array = "+s\_even);

System.out.println("The sum of odd elements of array = "+s\_odd);

}

}

**/\***

**\* Write a java program to find the second largest element of the array.**

**\*/**

/\*

\* a = [-1, 10, 2, 52, 32, 27, 79, 21]

\* second largest ==> 52

\*

\* large = a[0]

\* secondLarge = minimum value of the integer (-2^31 to 2^31 - 1)

\*/

public class SecondLargestElement {

public static void main(String[] args) {

int a[] = {10,100,2,52,32,27,79,21};

int n = a.length;

secondLarge(a, n);

}

static void secondLarge(int[] a, int n) {

int large = a[0];

int secondLarge = Integer.MIN\_VALUE;

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

if(a[i] > large) {

secondLarge = large; // 10

large = a[i]; // 100

}

else if(a[i] != large && a[i] > secondLarge) {

secondLarge = a[i]; // 52 79

}

}

System.out.println("The second largest element = "+secondLarge);

}

}

Assignment:

---------------

1) Write a java program to accept an array as an input.

And find the sum of all elements of an array.

**Day-04**

**13-03-2025**

**=====================**

Assignment:

--------------

1) Write a java program to find the third largest number from the array.

**Linear Search:**

**==========**

/\* Write a java program to search the element in the given array.

And find the position of that element using linear search \*/

public class LinearSearch {

public static void main(String[] args) {

int a[] = {11,22,33,44,55,27,26,66,77,101};

int search = 126;

boolean flag = false;

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

if(a[i] == search) {

System.out.println("The given element "+search+" is found at:"+i);

flag = true;

break;

}

}

if(flag == false) {

System.out.println("The given element "+search+" is not found.");

}

}

}

**Binary Search:**

**==========**

/\* Write a program using java for finding element in an array using binary search technique.

\*/

Why Binary Search?

-----------------------

There is a drawback with linear search

In the worst case, after the maximum number of comparisons, we can get the clarity that element not found in the array.

Ex: if the array with 1000 elements

and search object need to compare on all 1000 elements in the worst to print "that element is not found".

Here, in this situation we can understand that linear search takes more processing time.

-> To overcome this only, we need to implement the binary search.

How Binary search can work?

--------------------------------

1) sort the array.

sort():

------

-> by default the sorting can be in ascending order

Syntax:

import java.util.Arrays;

Arrays.sort(array);

-> define low = 0

and high = length - 1

-> check condition:

low <= high{

if it is "true":

mid = (low + high)/2

-> a[mid] == search ==> mid is the searching element

a[mid] < search or search > a[mid]

low = mid + 1

-> a[mid] > search or search < a[mid]

high = mid - 1

}

import java.util.Arrays;

public class BinarySearch {

public static void main(String[] args) {

int a[] = {11,99,10,27,88,26,55,101,23};

int search = 88;

binarySearch(a,search);

}

static void binarySearch(int[] a,int s) {

Arrays.sort(a);

int low = 0;

int high = a.length - 1;

boolean flag = false;

while(low <= high) {

int mid = (low + high)/2;

if(a[mid] == s) {

System.out.println("The element found.");

flag = true;

break;

}

else if(s > a[mid]) {

low = mid + 1;

}

else {

high = mid-1;

}

}

if(flag == false) {

System.out.println("Element not found.");

}

}

}

**Day-05**

**15-03-2025**

**======================**

**Q: is main() method user-defined or pre-defined method?**

**--------------------------------------------------------------------**

Ans: the definition of main() method is always according to the application requirement/business requirement. Hence we can say the main() is always user-defined. But, the name 'main' is pre-defined.

**Actual Vs Formal Parameters/Arguments:**

**--------------------------------------------------**

When we can define the arguments/parameters in method call those are called as "Actual Parameters/Arguments".

When we can define the arguments/parameters in method definition are called as "Formal Parameters/Arguments".

Here: the formal parameters can always take/accept values from actual arguments.

**/\***

**\* Write a program in java to find the smallest and biggest element**

**\* from the given array without sort().**

**\* a = {-1,10,0,77,99,-7,-11,33};**

**\* smallest = -11**

**\* biggest = 99**

**\*/**

import java.util.Scanner;

public class FindBiggestSmallest {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter the size of the array:");

int size = scan.nextInt();

int[] a = new int[size];

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

System.out.println("Enter the element for the position:"+i);

a[i] = scan.nextInt();

}

findSmallBig(a); // method call

}

static void findSmallBig(int[] arr) {

int small = arr[0];

int big = arr[0];

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

if(arr[i] < small) {

small = arr[i];

}

if(arr[i] > big) {

big = arr[i];

}

}

System.out.println("The smallest element = "+small);

System.out.println("The biggets element = "+big);

}

}

**/\***

**\* Write a java program to remove the duplicates from the**

**\* given array.**

**\***

**\* a = {1,3,1,2,3,5,5,7};**

**\* output: {1,3,2,5,7}**

**\*/**

import java.util.Arrays;

public class RemoveDuplicates {

public static void removeDuplicates(int[] a,int s) {

Arrays.sort(a);

int i = 0;

for(int j = 1;j < s;j++) {

if(a[i] != a[j]) {

i++; // 1

a[i] = a[j];// 3

}

}

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

System.out.print(a[k]+"\t");

}

}

public static void main(String[] args) {

int a[] = {1,3,1,2,3,5,5,7};

int s = a.length;

removeDuplicates(a,s);

}

}

**/\***

**\* Write a program in java to copy the array into another array.**

**\*/**

Solution-1: (Using Assignment operator)

--------------------------------------------

public class CopyArray {

public static void main(String[] args) {

int a[] = {1,3,5,7,9,11,13,15};

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

b = a;

System.out.println("The Array b is = ");

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

System.out.print(b[i]+"\t");

}

System.out.println();

System.out.println("The array a is = ");

for(int j = 0;j < a.length;j++) {

System.out.print(a[j]+"\t");

}

}

}

Solution-2: copying element by element:

---------------------------------------------

public class CopyArray {

public static void main(String[] args) {

int a[] = {1,3,5,7,9,11,13,15};

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

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

b[i] = a[i];

}

System.out.println("The Array a is = ");

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

System.out.print(a[i]+"\t");

}

System.out.println();

System.out.println("The Array b is = ");

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

System.out.print(b[i]+"\t");

}

}

}

**Aliasing and Cloning:**

**==============**

Datatypes are classified into two types:

1) Immutable datatypes

2) Mutable datatypes

-> All primitive datatypes are immutable datatypes

-> The data once we can define we cannot make it modify are called as "Immutable"

-> The data can allow for modification within the same location of heap memory is called as "Mutable datatypes".

-> Arrays are mutable datatypes.

-> When the new data object got created from another data object, in this case if the change on one object can reflect on another object is called as "Aliasing".

public class Aliasing {

public static void main(String[] args) {

int a[] = {1,3,5,7,9,11,13,15};

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

b = a;

System.out.println("The Array b is = ");

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

System.out.print(b[i]+"\t");

}

System.out.println();

System.out.println("The array a is = ");

for(int j = 0;j < a.length;j++) {

System.out.print(a[j]+"\t");

}

b[1] = 13;

System.out.println("The Array b is = ");

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

System.out.print(b[i]+"\t");

}

System.out.println();

System.out.println("The array a is = ");

for(int j = 0;j < a.length;j++) {

System.out.print(a[j]+"\t");

}

}

}

-> When the change on one object not reflecting on another object is called as "Cloning".

public class CloneArray {

public static void main(String[] args) {

int a[] = {11,22,33,44,55,66,77};

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

b = a.clone();

System.out.println("The Array before to change:");

System.out.println("The Array a = ");

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

System.out.print(a[i]+"\t");

}

System.out.println();

System.out.println("The Array b = ");

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

System.out.print(b[i]+"\t");

}

System.out.println();

a[6] = 777;

System.out.println("The Arrays after the change = ");

System.out.println("The Array a = ");

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

System.out.print(a[i]+"\t");

}

System.out.println();

System.out.println("The Array b = ");

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

System.out.print(b[i]+"\t");

}

System.out.println();

}

}

**Day-06**

**16-03-2025**

**==================**

**/\* Write a program in java to find the number of occurrences/frequency of each element in the given array\*/**

/\*

a = {1,3,5,7,9,1,2,3,4,5,5,4,3,2,1};

1 ==> 3-times

3 ==> 3-times

5 ==> 3-times

etc.

\*/

public class FrequencyOfElement {

public static void main(String[] args) {

int a[] = {1,3,5,7,9,1,2,3,4,5,5,4,3,2,1};

int n = a.length;

boolean cmp[] = new boolean[n];

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

if(cmp[i]) {

continue;

}

int count = 0;

for(int j = 1;j < n;j++) {

if(a[i] == a[j]) {

count++;

cmp[j] = true;

}

}

}

}

}

**Q: Is it possible to access/print the array without for loop?**

Ans: yes

toString():

-> a pre-defined or built-in method

can use to convert array to the string.

Syntax:

Arrays.toString(array-name);

one dimensional Array:

---------------------

Syntax:

datatype[] array-name = {v1,v2,v3,...};

Two dimensional Array:

----------------------

-> collection of 1-d arrays

Syntax:

datatype[][] array-name = {{1d},{1d},{1d},...};

Three Dimensional Array:

------------------------

-> collection of 2d arrays

Syntax:

datatype[][][] array-name = {{2d},{2d},{2d},...};

import java.util.Arrays;

public class ArraysWithDifferentDimensions {

public static void main(String[] args) {

//1-dimensional array

int[] a= {1,3,5,7,9};

// 2-dimensional array

int b[][] = {{1,2,3,4},{5,6,7,8},{10,11,12,13}};

int c[][][] = {{{1,2,3,4},{4,3,2,1}},{{1,3,5,7},{7,5,3,1}}};

System.out.println("The One Dimensional Array = "+Arrays.toString(a));

System.out.println("The Two Dimensional Array = ");

System.out.println("Row1"+Arrays.toString(b[0]));

System.out.println("Row2"+Arrays.toString(b[1]));

System.out.println("ROw3"+Arrays.toString(b[2]));

}

}

**/\***

**\* Write a java program to find the sum of two matrices.**

**\*/**

public class SumOfTwoMatrices {

public static void main(String[] args) {

int a[][] = {{1,2,3},{4,5,6},{7,8,9}};

int b[][] = {{9,8,7},{6,5,4},{3,2,1}};

int m = a.length; // number of rows

int n = a[0].length; // number of elements

int c[][] = new int[m][n];

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

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

c[i][j] = a[i][j] + b[i][j];

}

}

System.out.println("The Resultant Array = ");

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

for(int j = 0;j < c[i].length;j++) {

System.out.print(c[i][j]+"\t");

}

System.out.println();

}

}

}

**/\* Matrix Multiplication \*/**

public class MatrixMultiplication {

public static void main(String[] args) {

int a[][] = {{1,2,3},{4,5,6}}; // 2 X 3

int b[][] = {{1,2},{3,4},{5,6}}; // 3 X 2

int r1 = a.length;

int c1 = a[0].length;

int r2 = b.length;

int c2 = b[0].length;

int c[][] = new int[r1][c2];

if(c1 != r2) {

System.out.println("Matrix Multiplication is not possible.");

}

else {

for(int i = 0;i < r1;i++) {

for(int j = 0;j < c1;j++) {

for(int k = 0;k < c2;k++) {

c[i][k] += a[i][j] \* b[j][k];

}

}

}

System.out.println("The Resultant Matrix After the multiplication = ");

for(int i = 0;i < r1;i++) {

for(int j = 0;j < c2;j++) {

System.out.print(c[i][j]+"\t");

}

System.out.println();

}

}

}

}

**Matrix Transpose**

**============**

public class MatrixTranspose {

public static void main(String[] args) {

int a[][] = {{1,2,3},{4,5,6},{7,8,9}};

int m = a.length;

int n = a[0].length;

int b[][] = new int[n][m];

System.out.println("The array before the transpose = ");

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

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

System.out.print(a[i][j]+"\t");

}

System.out.println();

}

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

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

b[j][i] = a[i][j];

}

}

System.out.println("The array After the transpose = ");

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

for(int j = 0;j < m;j++) {

System.out.print(b[i][j]+"\t");

}

System.out.println();

}

}

}

**Day-07**

**17-03-2025**

**================**

**Frequency of Each element in an array:**

**=========================**

int[] a = {1,3,5,7,9,1,2,4,6,1,2};

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

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

{

if(visited[i] == 1){

continue;

}

int count = 1;

for(int j = 1;j < a.length; j++)

{

if(a[i] == a[j]{

count++;

visited[j] = 1;

}

}

System.out.println("The frequency of "+a[i]+" is = "+count);

}

**Sorting Techniques:**

**=============**

**Q: Is an array in java ordered or unordered?**

Ans: Ordered

Ex:

a = {100,500,200,400,600,300}

a[0] = 100

a[1] = 500

a[2] = 200

a[3] = 400

a[4] = 600

a[5] = 300

-> Sorting is a procedure can be used to arrange the elements in ascending order or descending order.

-> Sorting techniques are:

1) Bubble sort

2) Insertion sort

3) Selection sort

4) Merge Sort

5) Quick sort etc.

-> The sorting is possible with two types:

1) Forward Sort

2) Reverse Sort

-> when the elements can arrange in ascending order is called as "Forward Sort"

-> when the elements can arrange in descending order is called as "Reverse Sort"

1) Bubble Sort:

------------------

-> this technique can start compare the elements of the given array from begin to last and swap if the specified condition is satisfied.

import java.util.Arrays;

public class BubbleSort {

public static void main(String[] args) {

int[] a= {10,0,99,77,-1,32,-7,27,-9};

System.out.println("The Array before the sort = "+Arrays.toString(a));

for(int i = 0;i < a.length-1;i++) {

for(int j = 0;j < a.length-i-1;j++) {

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

int t = a[j];

a[j] = a[j+1];

a[j+1] = t;

}

}

}

System.out.println("The array after the sort = "+Arrays.toString(a));

}

}

2) Insertion Sort:

--------------------

1) Sorting can be takes place from index '1'

2) when the given condition (index >= 0 && array[index] > key),

i) move the current index towards right by one position

ii) insert/place the searching key (key) in that new index place.

3) repeat above two steps continuously until the last element.

import java.util.Arrays;

public class InsertionSort {

public static void main(String[] args) {

int a[] = {10,-1,0,99,7,32,-9,27,-11};

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

int key = a[i];

int j = i-1;

while(j >= 0 && a[j] > key) {

a[j+1] = a[j]; // move the index towards the right

j--;

}

a[j+1] = key;

}

System.out.println("The Array after the sort = "+Arrays.toString(a));

}

}

**How to merge two arrays into one:**

**-----------------------------------------**

int a[] = {10,20,30,40};

int b[] = {11,22,33,44,55};

int m = a.length;

int n = b.length;

int c[] = new int[m+n];

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

{

c[i] = a[i];

}

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

{

c[m + i] = b[i];

}

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

{

System.out.print(c[i]+"\t");

}

Result:

int c[] = {10,20,30,40,11,22,33,44,55};