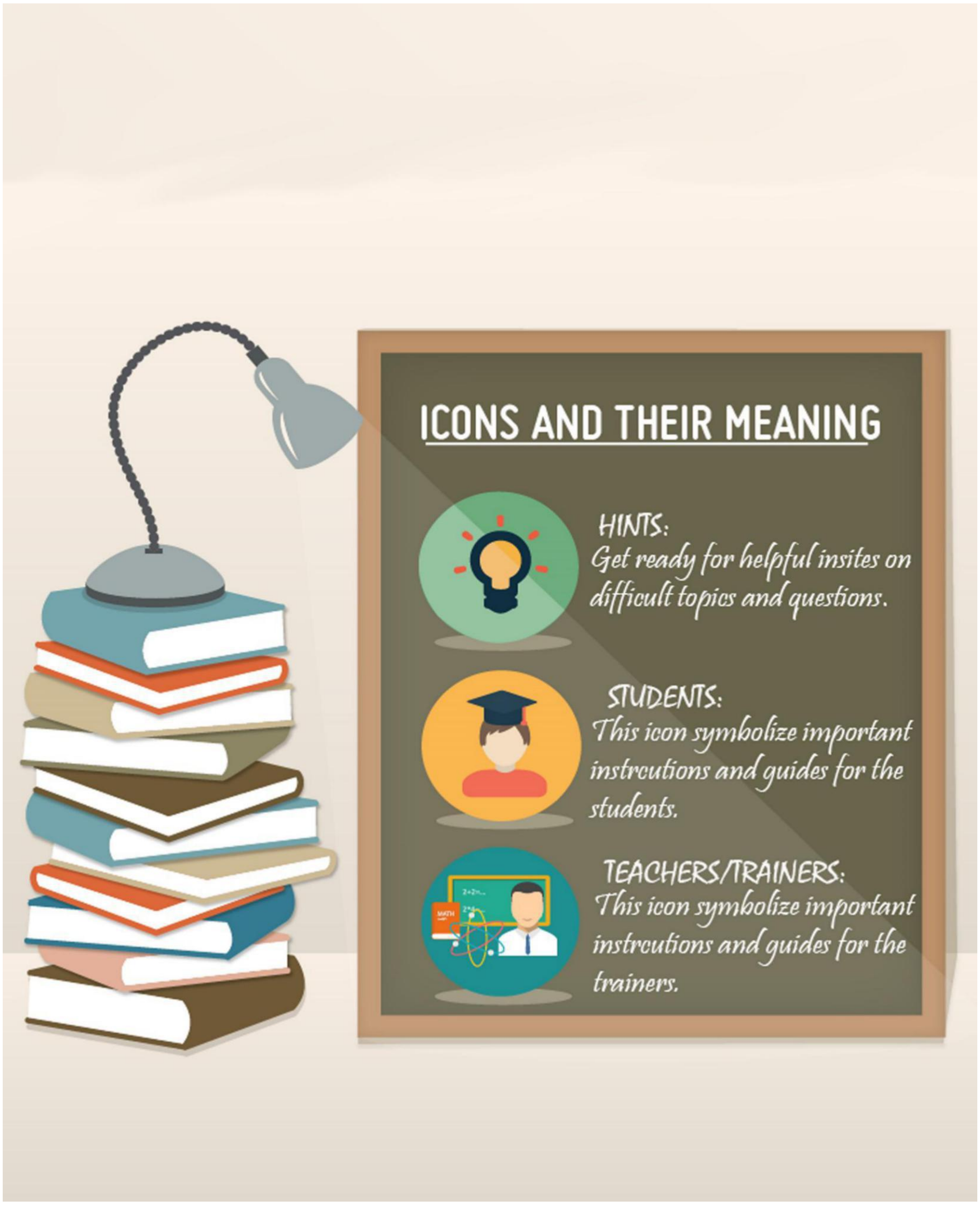
# CORE JAVA

MANUAL V8.3

**MODULE CODE:**

**ANUDIP FOUNDATION**


ICONS AND THEIR MEANING
HINTS:
Get ready for helpful insites on difficult topics and questions.
STUDENTS:
This icon symbolize important instrcutions and guides for the students.
TEACHERS/TRAINERS:
This icon symbolize important instrcutions and guides for the trainers.

# Module 4: Array, Enumeration and Collections

## Chapter 2

| | |
|---|---|
| **Objective:** After completing this lesson you will be able to : <br> * Learn about sorting and searching array elements <br> * Learn about using predefined array functions for manipulating arrays | **Materials Required:** <br> 1. Computer <br> 2. Internet access |
| **Theory Duration:** 60 minutes | **Practical Duration:** 60 minutes |
| **Total Duration:** 120 minutes | |

# Chapter 2

## 2.1 Array element sorting and Searching

Java enables programmers to sort the elements of arrays using sort(). Sort() is a class method of java.util.Arrays. Sorting and searching are important as they can help in looking for particular array elements.

Array sorting can be used to sort elements in both ascending and descending orders. It can be used for viewing array elements in an organized manner. The widely-used methods of sorting arrays include –

* Bubble sort
* Insertion sort

**An example of sorting code can help you get a better idea –**

```
package sortingarrays.sort;

import java.util.Arrays;

public class JavaArraySort {

    public static void main(String[] args) {
        int[] intArr = {6, 10, 9, 12, 8};
        String[] strArr = {'I', 'U', 'A','E','O'};
        Arrays.sort(intArr);
        Arrays.sort(strArr);

        System.out.println(Arrays.toString(intArr));
        System.out.println(Arrays.toString(strArr));
    }
}
```

**Output:**

```
[6, 8, 9, 10, 12]
[A, E, I, O, U]
```

**Example of ascending array sort for sorting integer elements in ascending order –**

```
import java.util.Arrays;
public class JavaSortingDemo
{
    public static void main(String[] args)

  {
        Integer[] numbers = new Integer[] { 18, 19, 13, 12, 15, 17, 16 };
        Arrays.sort(numbers);
        System.out.println(Arrays.toString(numbers));
    }
}
```

**Output:** [12, 13, 15, 16, 17, 18, 19]

**Example of descending array sort for sorting integer elements in descending order. It can be done with the Collections.reverseOrder() comparator in Java –**

```
Integer[] numbers = new Integer[] { 18, 19, 13, 12, 15, 17, 16 };

Arrays.sort(numbers, Collections.reverseOrder());
```

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

**Output:** [19, 18, 17, 16, 15, 13, 12]

*** Array Searching**

Binary search is the method used to search an element from a collection of elements. To perform a binary search, all elements of an array must be in ascending order. Take a look at a program showing this type of search below.

```
class BinarySearchDemo{
            public static void binarySearch(int first, int last,i nt arr[], int key){
              int mid = (first + last)/2;
              while( first <= last ){
              if ( arr[mid] < key ){

                   first = mid + 1;
                 }else if ( arr[mid] == key ){
               System.out.println('Element found at index: ' + mid);
                      break;
           }else{
           last = mid - 1;
                }
                 mid = (first + last)/2;
              }
              if ( first ı last ){
                 System.out.println('Element not found!');


              }
            }
            public static void main(String args[]){
```

```
        int arr[] = {20,30,40,50,60};

        int key = 40;

        int last=arr.length-1;

        binarySearch(arr,0,last,key);

    }
   }
```

**Output**: Element found at index: 2

## 2.2 **Manipulating Array by predefined Array functions**

Arrays can be manipulated by using some predefined functions. They all serve specific search and sort functions. Take a look –

**binarySearch(short[], short**) – Uses binary search for searching particular 'short' array to find a given value

**binarySearch(int[], int**) – Uses binary search for searching a specific 'int' array to find a value

**binarySearch(double[], double**) – Uses binary search for searching a specific 'double' array to find a value

**binarySearch(A[], A, Comparator<A**I) – Uses binary search algorithm for a specific 'Object'

**binarySearch(A[], A**) – Uses binary search algorithm for a specific 'Object'

**binarySearch(float[], float**) – Uses binary search for searching a specific 'float' array to find a value

**binarySearch(byte[], byte**) – Uses binary search for searching a specific 'byte' array to find a value

**binarySearch(char[], char**) – Uses binary search for searching a specific 'char' array to find a value

**binarySearch(long[], long**) – Uses binary search for searching a specific 'long' array to find a value

**equals(short[], Object**) – Returns a true output if the specific 'short' array is equal to a given object

**equals(byte[], Object**) – Returns a true output if the specific 'byte' array is equal to a given object

**equals(double[], Object**) – Returns a true output if the specific 'double' array is equal to a given object

**equals(A[], Object**) – Returns a true output if the specific 'Object' array is equal to a given object

**equals(float[], Object**) – Returns a true output if the specific 'float' array is equal to a given object

**equals(boolean[], Object**) – Returns a true output if the specific 'boolean' array is equal to a given object

**equals**(**char[], Object**) – Returns a true output if the specific 'char' array is equal to a given object

**equals**(**int[], Object**) – Returns a true output if the specific 'int' array is equal to a given object

**equals**(**long[], Object**) – Returns a true output if the specific 'long' array is equal to a given object

**fill**(**char[], char**) – Sets a specific 'char' value for all elements of a particular 'char' array

**fill**(**boolean[], boolean**) – Sets a specific 'boolean' value for all elements of a particular 'boolean' array

**fill**(**float[], float**) – Sets a specific 'float' value for all elements of a particular 'float' array

**fill**(**A[], A**) – Sets a specific 'Object' value for all elements of a particular 'Object' array

**fill**(**double[], double**) – Sets a specific 'double' value for all elements of a particular 'double' array

**fill**(**byte[], byte**) – Sets a specific 'byte' value for all elements of a particular 'byte' array

**fill**(**int[], int**) – Sets a specific 'int' value for all elements of a particular 'int' array

**fill**(**short[], short**) – Sets a specific 'short' value for all elements of a particular 'short' array

**fill**(**long[], long**) – Sets a specific 'long' value for all elements of a particular 'long' array

**sort**(**short[]**) – Sorts a particular 'short' array in a numerical ascending order

**sort**(**byte[]**) – Sorts a particular 'byte' array in a numerical ascending order

**sort**(**float[]**) – Sorts a particular 'float' array in a numerical ascending order

**sort**(**A[], Comparator<A**I) – Sorts a particular 'object' array as directed by a specific order

**sort**(**A[]**) – Sorts a particular 'object' array in ascending order

**sort**(**double[]**) – Sorts a particular 'doubles' array in ascending order

**sort**(**char[]**) – Sorts a particular 'char' array in ascending order

**sort**(**int[]**) – Sorts a particular 'int' array in ascending order

**sort**(**long[]**) – Sorts a particular 'long' array in ascending order

**toList**(**A[]**) – Returns a fixed-size List output with the help of a particular array

**Practical** (60 **minutes**)

See the example programme for Java array sorting below. Write the same programme to sort the integers 8, 4, 3, 5, 6 and the alphabetical string C, O, I, P, U, in ascending order. Show the resulting output. Repeat the same for the integers 10, 7, 9, 13, 17 and the alphabetical string E, H, G, A, C.

```
package sortingarrays.sort;

import java.util.Arrays;

public class JavaArraySort {

    public static void main(String[] args) {

        int[] intArr = {6, 10, 9, 12, 8};

        String[] strArr = {'I', 'U', 'A','E','O'};

        Arrays.sort(intArr);

        Arrays.sort(strArr);

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

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

    }

}
```

**Instructions: The progress of students will be assessed with the exercises mentioned below.**

**MCQ (10 minutes)**

1. programmers to sort the elements of arrays using

a) sort()

b) search()

c) float()

d) None of the mentioned

2. Two popular array sorting methods are bubble sort and ________ sort.

a) object

b) insertion

c) list

d) None of the mentioned

3. Sorting is a _______ method of java.util.Arrays

a) class

b) array

c) subclass

d) None of the mentioned

4. What comparator can be used for descending array sort?

a) Collections.inverseOrder()

b) Collections.reverseOrder()

c) Collections.decreasingOrder()

d) None of the mentioned

5. To search an element from a collection of elements a programmer can use _________ search

a) non-binary

b) indexing

c) binary

d) None of the mentioned

6. A binary search requires array elements to be in ___________ order.

a) ascending

b) alphabetical descending

c) descending

d) None of the mentioned

7. Java arrays can be manipulated by ________ functions

a) custom

b) predefined

c) user-defined

d) None of the mentioned

8. binarySearch(int[], int) uses binary search for _______ a specific 'int' array.

a) sorting

b) unsorting

c) finding

d) None of the mentioned

9. equals(float[], Object) returns a _____ output if a particular 'float' array is equal to a given object

a) true

b) null

c) false

d) None of the mentioned

10. What function sorts a particular 'object' array in ascending order?

a) sort(A[])

b) sort(Object[])

c) sort(char[])

d) None of the mentioned