

* What is an array?

- An array is a data structure that stores a fixed-size sequential collection of elements of the same type.
- It stores data in contiguous memory locations.
- Each element of the array can be accessed using indexing where index starts from 0 & goes upto length-1.
- We can only store a fixed set of elements in an array.
- We can store primitive values or objects in an array.

* Syntax:-

Syntax of array

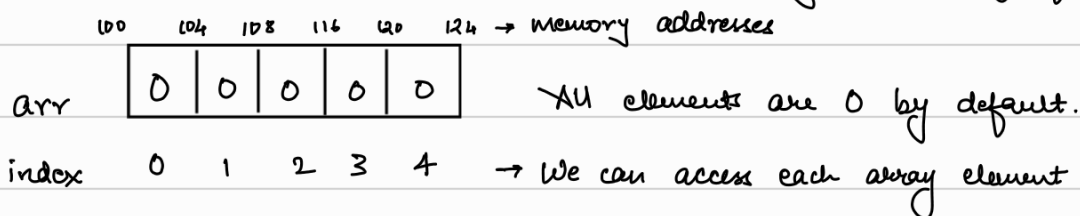
```
// Syntax to declare an array
dataType[] arr; // (OR)
dataType []arr; // (OR)
dataType arr[];

// Instantiation of an array
arr = new dataType[size];

// Defining an array
dataType[] arrReference = new dataType[size];
```

→ Memory gets assigned only when we instantiate an array.

`int[] arr = new int[5];` → This creates a contiguous memory for 5 int elements.



* Initializing an array:-

`int[] arr1 = {1, 2, 3, 4, 5};` → Using array initializer

`int[] arr2 = new int[3];`

`arr2[0] = 1;`

`arr2[1] = 2;`

`arr2[2] = 3;`

} manual initialization using indexes.

* Accessing array elements :-

```
int[] arr = {1, 2, 3};
```

```
cout (arr[0]); → 1
```

```
cout (arr[1]); → 2
```

```
cout (arr[2]); → 3
```

```
cout (arr[3]); → ArrayIndexOutOfBoundsException
```

* length property :-

→ Returns the count of total elements present in the array.

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

```
cout (arr.length); → 5
```

* Traversing an array :-

1) Using for loop :-

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

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

```
    cout (arr[i]);
```

```
}
```

2) Using for-each loop :-

```
for (int element : arr) {
```

```
    cout (element);
```

```
}
```

→ Can only read the array.

→ Doesn't include indexes.

→ Prefer for-each, when you don't need index access.

* Copying an array :-

1) Shallow copy :- It only copies the reference of the array. Changes in the copied array will reflect in the original array.

```
Main.java > ...
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5
6          int[] arr1 = { 1, 2, 3 };
7          int[] arr2 = arr1; // Shallow copy
8
9          System.out.println("arr1: " + Arrays.toString(arr1));
10         System.out.println("arr2: " + Arrays.toString(arr2));
11
12         arr2[0] = 69; // Changing values in new array
13         System.out.println("After modification");
14         System.out.println("arr1: " + Arrays.toString(arr1));
15         System.out.println("arr2: " + Arrays.toString(arr2));
16     }
17 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
• → java-workspace java Main.java
arr1: [1, 2, 3]
arr2: [1, 2, 3]
After modification
arr1: [69, 2, 3]
arr2: [69, 2, 3]
```

2) Deep copy :- Creating a new array & copying each element individually. Modifying the new array doesn't change anything in the original array.

```
Main.java > ...
1  public class Main {
2      public static void main(String[] args) {
3
4          int[] arr1 = { 1, 2, 3 };
5
6          // Deep copy
7          int[] arr2 = new int[arr1.length];
8          for (int i = 0; i < arr1.length; i++) {
9              arr2[i] = arr1[i];
10         }
11     }
12 }
```

* Important methods of java.util.Arrays class:-

1) Arrays.toString():-

→ Converts an array to a string representation.

```
int[] arr = {1, 2, 3};
```

```
System.out.println(Arrays.toString(arr)); → [1, 2, 3]
```

→ It returns a string.

2) Arrays.sort():-

→ Sorts an array in ascending order.

→ Return type is void.

```
int[] arr = {3, 2, 1};
```

```
Arrays.sort(arr);
```

```
System.out.println(Arrays.toString(arr)); → [1, 2, 3]
```

3) Arrays.fill():-

→ Fills an array with a specified value.

```
int[] arr = new int[5];
```

```
Arrays.fill(arr, 10); → [10, 10, 10, 10, 10]
```

```
Arrays.fill(arr, 1, 4, 69); → [10, 69, 69, 69, 10]
```

↑ ↑ ↑
From To Value
index index

4) Arrays.copyOf():-

→ Create a deep copy of the original length.

→ It will truncate or pad with additional 0 as per the **newLength** param.

```
int[] arr = {1, 2, 3};
```

```
int[] newArr = Arrays.copyOf(arr, 5); → [1, 2, 3, 0, 0]
```

↑
new length

5) Arrays.copyOfRange() :-

→ Creates a deep copy of the original array with the specified range.

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

```
int[] subArr = Arrays.copyOfRange(arr, 1, 4); → [2, 3, 4]
```

↑ ↑
from to (-1)

6) Arrays.equals() :-

→ Checks if two arrays are equal.

→ It checks if all corresponding elements are same or not.

```
int[] arr1 = {1, 2, 3};
```

```
int[] arr2 = {1, 2, 3};
```

```
System.out.println(Arrays.equals(arr1, arr2)); → true
```

7) Arrays.mismatch() :-

→ Finds & returns the index of the first mismatch between two arrays.

→ Returns -1 if no mismatch is found.

```
int[] arr1 = {1, 2, 3};
```

```
int[] arr2 = {1, 5, 6};
```

```
System.out.println(Arrays.mismatch(arr1, arr2)); → 1
```