* 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

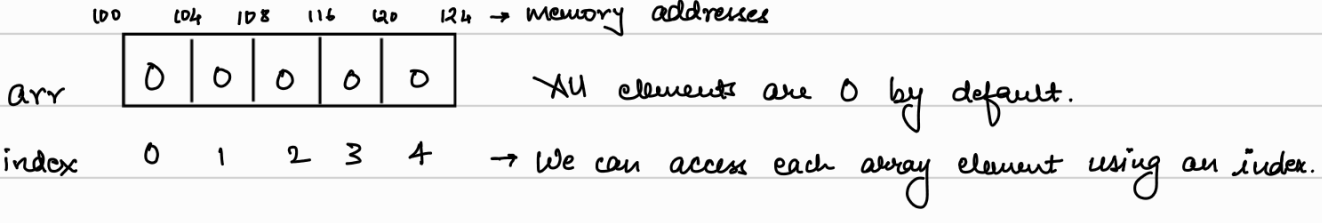
```
// Syntax to declare an array
dataType[] arr; // (OR)
dataType []arr; // (OR)
dataType arr[];

// 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.



All elements are 0 by default.

→ We can access each array element using an index.

* Initializing an array :-

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

```
int[] arr2 = new int[3];
arr2[0] = 1;
arr[1] = 2;
arr[2] = 3;
```

} manual initialization using indexes.

* Accessing array elements :-

```
int[] arr = {1, 2, 3};
sout(arr[0]); → 1
sout(arr[1]); → 2
sout(arr[2]); → 3
sout(arr[3]); → ArrayIndexOutOfBoundsException
```

* length property :-

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

```
int[] arr = {1, 2, 3, 4, 5};
sout(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++) {
    sout(arr[i]);
}
```

2) Using for-each loop :-

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

```java
import java.util.*;

public class Main {
    Run | Debug
    public static void main(String[] args) {

        int[] arr1 = { 1, 2, 3 };
        int[] arr2 = arr1; // Shallow copy

        System.out.println("arr1: " + Arrays.toString(arr1));
        System.out.println("arr2: " + Arrays.toString(arr2));

        arr2[0] = 69; // Changing values in new array
        System.out.println(x:"After modification");
        System.out.println("arr1: " + Arrays.toString(arr1));
        System.out.println("arr2: " + Arrays.toString(arr2));
    }
}
```

```
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.

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {

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

        // Deep copy
        int[] arr2 = new int[arr1.length];
        for (int i = 0; i < arr1.length; i++) {
            arr2[i] = arr1[i];
        }
    }
}
```

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

1) Arrays.toString():-

→ Converts an array to a string representation.

```
int[] arr = {1,2,3};
sout(Array.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);
sout(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]
                               0   1   2   3   4
```

from index — To index — Value

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]
```

5 ← 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 (1) 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};
sout (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};
sout (Arrays.mismatch(arr1, arr2)); → 1
```