Array

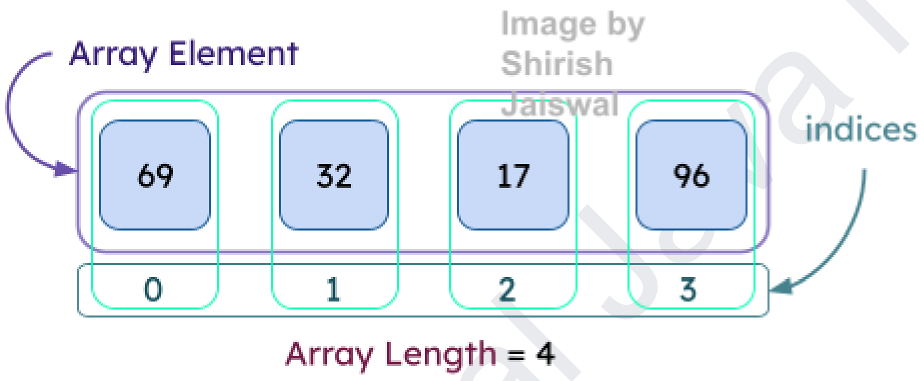
INDEX

Definition :

Array is a collection of the same datatype (int, char, String etc) element stored in a data structure of fixed size.

- Contiguous memory location
- Only Homogeneous data can be stored
- Indexing of the array start from 0
- Only Index based accessing is allowed
- It is a static data structure. Once the size of an array is declared it cannot be changed.
- All the elements are stored under one variable name

Contiguous means it needs memory adjacent to each other.



Types of array

1. **One Dimensional Arrays**

   Is a linear array in which elements are stored in a continuous row.

2. **Two-Dimensional Arrays**

   It is array of array in which elements are stored in continuous rows and columns

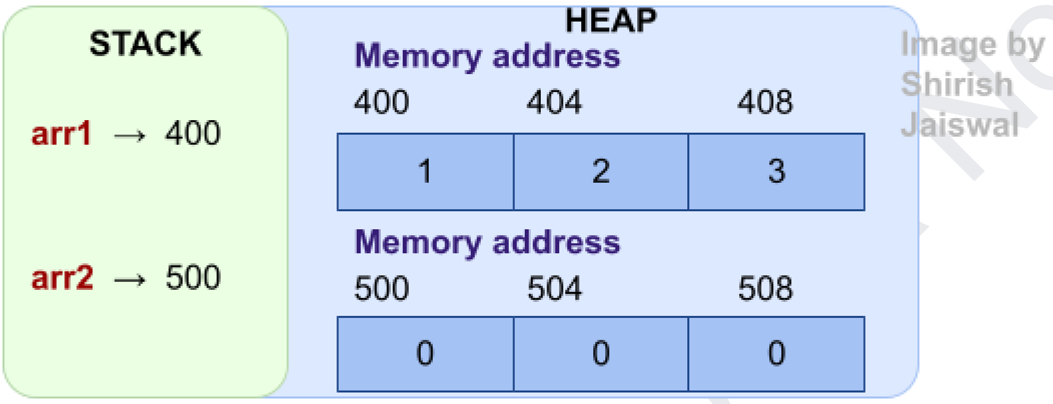
Note: *All the memory addresses are just for understanding.*

Note:

- stack will store the reference of the object
- reference will store the memory address of the heap where the object is created
- Objects will be created in the heap

One Dimensional array

Sr. no.	Reference Type []	Reference variable	=	Memory allocation	Object type []	{value1, value2 value3} //preknown values
1)	int []	arr1	=	new	int []	{1, 2, 3};
2)	int []	arr2	=	new	int [3];	// 3 is the size of array



Memory addresses are **increasing by 4**. → 400 → 404 → 408

Because the datatype of the array is int which takes **4 bytes** of memory.

arr1 is referring to 400 *(it can be anything)* because the first element of the array is on memory address 400.

Values of arr2 are 0 that means these are **default values.** Default values will be allocated if we do not add any value.

Access the value to array

To access the value from the array, the array name and the index is used. Indexing starts from 0.

System.out.println (arr1[0]);	**Output :**	1
System.out.println (arr1[1]);	**Output :**	2
System.out.println (arr1[2]);	**Output :**	3

You can also **update** the value of the array of specific index

```
arr1[0] = 5;
arr1[1] = 6;
arr1[2] = 7;
```

int [] arr2 = new int[3];

If declared as above then the value at each index inside the array will be the default value, till the time we don't add or update any value in it.

Default values are:

Data Type	Default Value	Data Type	Default Value	Data Type	Default Value
byte	0	short	0	int	0
long	0L	float	0.0f	double	0.0d
char	' '	boolean	false	Object	null

Update/Add, Output

If we update the values like we updated in the above example of arr1 then it will take a long time to update the whole array. So we can use for loop to update the value

```
int [] arr2 = new int [3];
for (int i = 0; i < 3; i++) {
        arr2 [i] = sc.nextInt();
}
```

As the size of the array is 3. But indexing in array start from 0 so, it will travel up-to *size - 1* in this case that is 3 - 1 = 2, so we have used <3

Length Function

Many times you may not know the size of an array. So to find out the size of an array we have a .length function which will return int value.

```
int [] arr2 = new int [10];
int len = arr2.length;
for (int i = 0; i < len; i++) {
        arr2 [i] = sc.nextInt();
}
```

Now we know how to **add** or **update** all the values in an array in one go.
By using a for loop we can provide **output**. Just by changing the body with S.*o*.p (arr2[i]); line.

Enhanced for loop / for - each loop

In Java, the **for-each** loop is used to iterate through elements of arrays and collections

Syntax :

```
for (dataType variable :  array reference variable) {
        // body of the loop
}
```

● array	-	an array or a collection
● variable	-	each item of array/collection is assigned to this variable
● dataType	-	the data type of the array/collection

```
for (int num : arr1) {
        System.out.println (num);
}
```

The value at indexes will be stored inside the variable `num` and it will be printed.

Limitations of for-each loop

- Not appropriate when modifying the array
- No index based accessing
- only iterates forward over the array in single steps
- cannot process two decision making statements at once

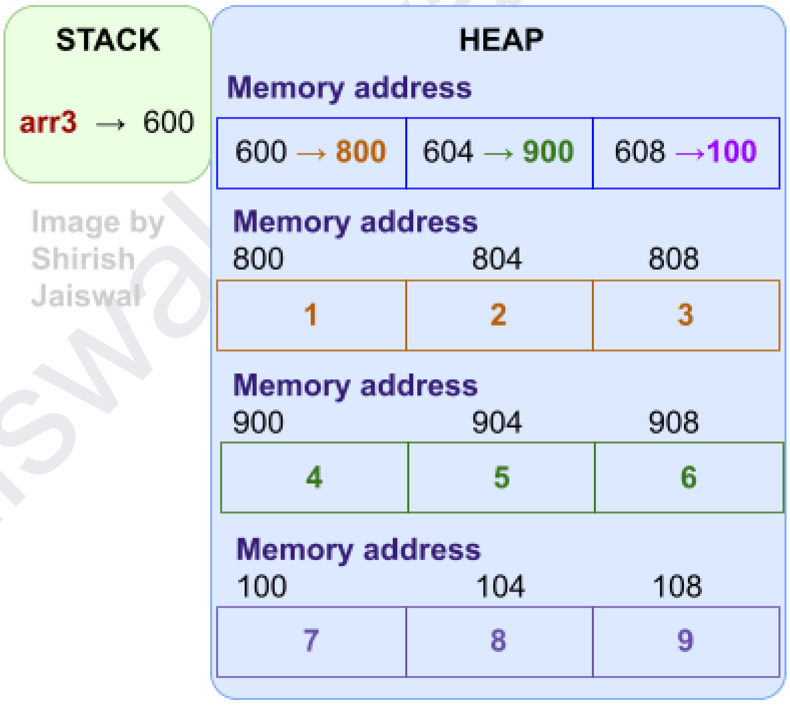
Two Dimensional array

Two dimensional array is an array of arrays in which elements are stored in continuous rows and columns.
This is not the matrix but an array having an array as its element.
If we have predefined values we can write as

```
int [][] arr3 = {      // 0      1       3       index of column
                   { 1,     2,      3 },     // 0 row index
                   { 4,     5,      6 },     // 1 row index
                   { 7,     8,      9 }      // 2 row index
               };
```

We can now clearly see that a 2d array is the array which contains arrays within itself.

int [row][column]



Update/Add, Output

If we want to update the values inside a 2D array we need to specify the index of row and column of the element that means row is the element of the main array and column is the element of the array's element array.

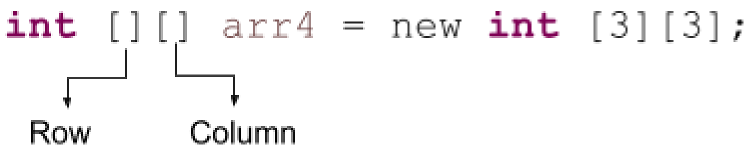
```
arr3 [0][0] = 3;
```

When we need to access the value of the first element of an array we write

```
System.out.println (arr3[0][0]);
```

Output : 1

When we don't have preknown values we need to add the size of row and column of the array
Syntax:



```
int [][] arr4 = new int [3][3];
```

This means we have length of an array arr4 is 3 and the length of its element array is 3.

Methods in arrays

```
int [] arr1  = {1, 2, 3};
int [] arr2  = {1, 2, 3};
int [] arr3  = {4, 5, 6};
```

- == operator & *.equals()* method - **NOT to USE in case of Array.**

```
System.out.println (arr1  == arr2 );          Output :      false
```

Even though the values of the array are the same, it's giving **`false`** output.

Because **==** operator will check the references of arrays which store the memory address.

If the reference variables are pointing to the same memory location then output is **`true`**.

For above output to be **`true`** then the snippet should be

```
int [] arr1  = {1, 2, 3};
int [] arr2;
arr2  = arr1;
System.out.println (arr1  == arr2 );                    Output :      true
```

In the above snippet both the references **`arr1`** and **`arr2`** are pointing toward the same memory location so the output is **`true`**.

.equals() method

```
System.out.println (arr1.equals(arr2));                 Output :      false
```

This method is an Object class method which compares actual values. But in the case of an array this **method is overridden** and it's the same as **== operator** checking the references of the arrays. So it will return output as **`false`**

- **Arrays.*equals* (obj1, obj2)**

  System.out.println (Arrays.*equals* (arr1, arr2)); **Output :** true

  System.out.println (Arrays.*equals* (arr1, arr3)); **Output :** false

  This function will compare the **actual value** inside the arrays, So the output of arr1 and arr2 is true and output of arr1 and arr3 is false.

- **Arrays.*sort* (obj1)**

  This will sort the array in ascending order.

- **Arrays.*toString* (obj1)**

  Returns a string representation of the contents of the specified array.

- **Arrays.*mismatch* (obj1, obj2)**

  Finds and returns the index of the first unmatched element between the two specified arrays. And if both the arrays are the same it returns -1.

  System.out.println (Arrays.*mismatch* (arr1, arr2)); output : -1

  System.out.println (Arrays.*mismatch* (arr1, arr3)); output : 0

- **Arrays.*fill(obj, from index, to Index, value)***

  Used to Fill an array to specific value at each index

  Arrays.*fill(*arr2, *0, 3, 3)*; output : 3 3 3

  It will always consider the ***to Index*** as *index -1*

Advantages and Dis-Advantages of array

Advantages

- We can store multiple homogeneous elements under a single name.
- Array has a safer cache positioning that improves performance
- Accessing an element is very fast because of contiguous memory and easy by using the index number
- Array allows random access to elements
- Arrays allow matrix data to be stored using a multidimensional array.

Dis-Advantages

- To create an Array, contiguous memory is required.
- The values stored in the array should have a homogeneous data type.
- The Array is a static data structure. It is of fixed size. We can not increase or decrease the size of the Array after creation
- When it comes to insertion and deletion it is a bit difficult because elements are stored sequentially and the shifting operation is expensive.