

* Intro To Array & ArrayList *

Page No.

Date:

* why do we need Arrays ?

⇒ It was simple, when we had to store just five integer numbers and now let's assume we have to store 5000 integer numbers. ~~It~~ is possible to use 5000 variable? [No]

To handle these situations, in almost programming language we have a concept called **Array**

Array is a data structure use to store a collection of data.

Syntax of Array -

`datatype[] variable-name = new datatype(size);`

Eg :- we want to store roll numbers:

`int[] rollnos = new int[5]`

or

`int[] rollnos = {51, 82, 13, 15, 16}`

* Note :-

- The datatype basically represents what is the data stored inside the array
- All the type of the data in the array should be same

* new keyword is used to create an object.
It will create an object in heap memory of array size size 5.

⇒ Internal working of an array

`int[] rollnos;` // declaration of array

↳ rollnos are getting defined in stack

`rollnos = new int[5];` // initialization

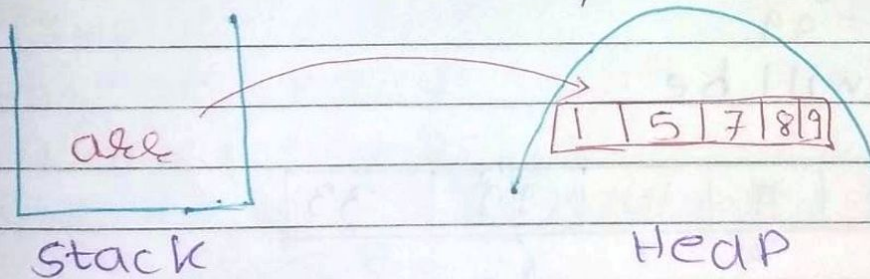
↳ actual memory allocation happens here. Here, object is being created in heap memory

declaration of
~~array~~
compile time

initialization
run time.

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

⇒ This above concept is known as Dynamic memory Allocation which means at runtime OR at execution time memory is allocated.



⇒ Internal Representation of Array:-

- Internally in Java, memory allocation totally depends on JVM whether it be continuous or not!

Reason 1: Objects are stored in heap memory

Reason 2: In JLS (Java Language Specification) it is mentioned that heap objects are not continuous.

Reason 3: Dynamic memory allocation. Hence, array objects in Java may not be continuous (depends on JVM)

* Index of an array:

index \rightarrow	0	1	2	3	4	5
arr \rightarrow	3	8	9	10	53	33

arr[0] = 3 arr[2] = 9 arr[4] = 53

arr[1] = 8 arr[3] = 10 arr[5] = 33

Suppose to change the value of certain index:

arr[4] = 99

New array will be.

3	8	9	10	99	33
0	1	2	3	4	5

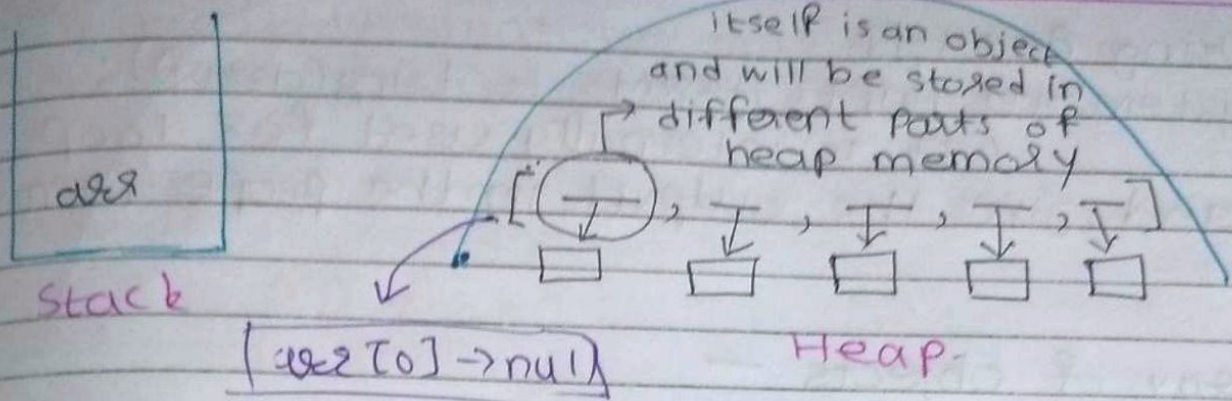
\Rightarrow If we don't provide values in the array, internally by default it stores [0, 0, 0, 0, 0] for above size of array

String arr = new String[4];

Primitives are stored in the stack memory only

Page No.

Date:



- * Primitives (int, char etc) are stored in stack
- * All other objects are stored in Heap memory.

• Input an array:-

- Input using for loops.

```
for (int i = 0; i < arr.length; i++) {  
    arr[i] = in.nextInt();  
}
```

← Inp

• Printing an array

- Using for loops.

```
for (int i = 0; i < arr.length; i++) {  
    arr[i] =  
    System.out.print(arr[i] + " ");  
}
```

- Using for each loops.

```
for (int j : arr) {  
    System.out.print(j + " ");  
}
```

// Here j represents the elements of an array

- Using Arrays.

```
System.out.print(Arrays.toString(array));
```

// It is internally used for loop and gives the output in the proper format

* Array of objects —

```
String[] str = new String[4];
```

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

```
    str[i] = in.next();
```

```
}
```

```
System.out.print(Arrays.toString(str));
```

* Passing ⁱⁿ functions —

```
{
```

```
int[] nums = {3, 4, 5, 12};
```

```
System.out.print(Arrays.toString(nums));
```

```
change(nums);
```

```
System.out.print(Arrays.toString(nums));
```

```
}
```

```
static void change(int[] arr){
```

```
    arr[0] = 99;
```

```
}
```

output : —

[3, 4, 5, 12]

[99, 4, 5, 12]

* In an array since we can change the objects, hence they are mutable. (changeable)

* Strings are immutable (not changeable)

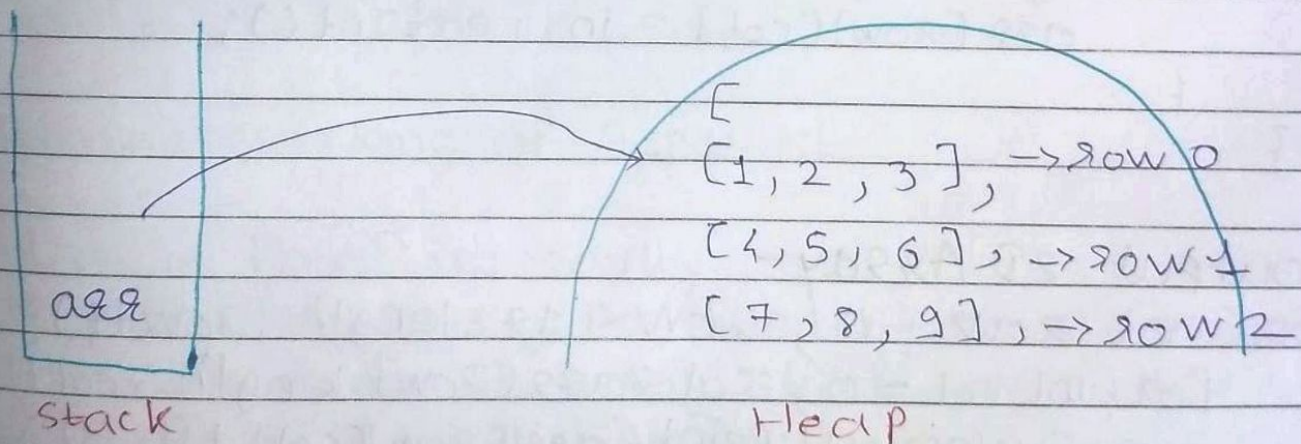
2D Arrays

3. $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Rightarrow \text{int}[][] \text{arr} = \text{new int} [\text{size}] []$

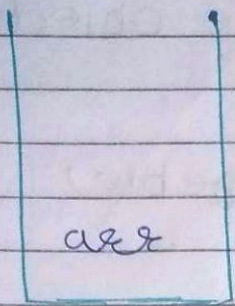
row column.
 $\downarrow \quad \downarrow$
 \uparrow
 mandatorily to give size of row.

OR

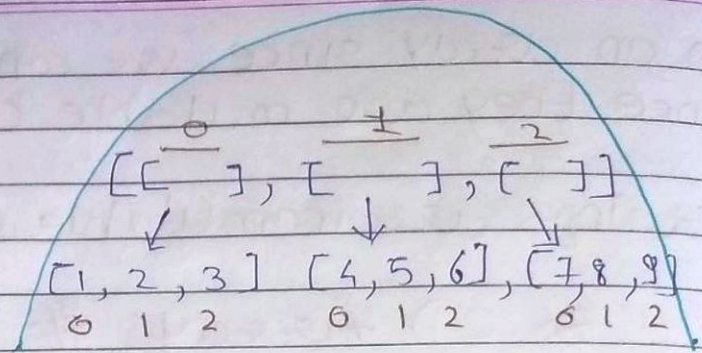
$\text{int}[][] \text{arr} = \{$
 $\{1, 2, 3\},$
 $\{4, 5, 6\},$
 $\{7, 8, 9\}$
 $\}$



Imagine this as an array of arrays.



stack



Heap

`arr[0] = [1, 2, 3]`

`arr[0][2] = 3.`

^{2d}
* Input Array —

```
int[][] arr = new int[3][2];
```

```
for(int row = 0; row < arr.length; row++){
    // for each col in every row
```

```
    for(int col = 0; col < arr[row].length; col++){
        arr[row][col] = in.nextInt();
```

```
    }
```

```
}
```

* Output 2D Array —

```
for(int row = 0; row < arr.length; row++){
    for(int col = 0; col < arr[row].length; col++){
        System.out.print(arr[row][col] + " ");
    }
```

```
    System.out.println();
```

```
}
```

OR


```
for (int row = 0; row < arr.length; row++) {
    System.out.print (Arrays.toString (arr [row]));
}
```

OR

```
for (int[] ar : arr) {
    System.out.print (Arrays.toString (ar));
}
```

ArrayList

'ArrayList' is a part of collection framework and is present in java.util.package. It provides us with dynamic arrays in Java. It is slower than standard arrays.'

Syntax:-

```
ArrayList <Integer> list = new ArrayList <> ();
```

(add wrapper here.)

* Internal working of ArrayList

- Size is fixed internally
- Suppose ArrayList gets filled by some amount
 - a) it will make an arrayList of say double the size of arrayList initially
 - b) old elements are copied in the new arrayList
 - c) old ones are deleted.

* Multi Dimensional ArrayList.

• Syntax:-

```
ArrayList < ArrayList < Integer > > List = new  
    ArrayList < > ();
```

// Initialization

```
for (int i=0 ; i<3; i++)  
{
```

```
    List.add (new nextInt
```

```
    List.add (new ArrayList < >());
```

```
}
```

// Input

```
for (int i=0 ; i<3 ; i++){
```

```
    for (int j=0 ; j<3 ; j++){
```

```
        list.get(i).add (in.nextInt());
```

```
    }
```

```
}
```