* Introduction to Arrays & Arraylist in Java

- 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 situation in almost all programming language we have a concept called Array.

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

=> Syntax of an Array:

datatype[] variable_name = new datatype[size];

eg:- we want to store roll numbers:

int[] rollnos = new int[5]; store 5 roll numbers.

OR

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

represent the type of data stored in array.

All the type of data in array should be same!

=> Internal working of array:

int[] rollnos; // declaration of array

↳ rollnos are getting defined in stack.

rollnos = new int[5]; // initialisation

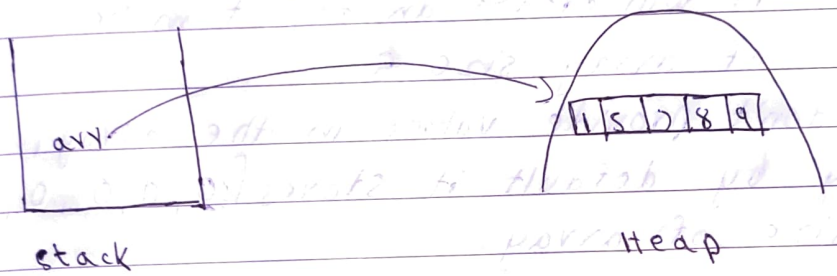
↳ actual memory allocation happens here, object is being created in heap memory

declaration of array (compile time) — initialisation (runtime)

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

- `int [ ]` → datatype
- `arr` → ref var
- `new int [5]` → creating object in heap memory

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




⇒ Internal Representation of Array:

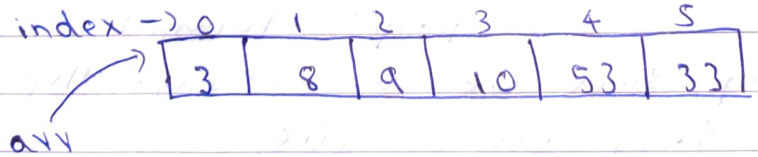
- Internally in Java, memory allocation totally depends on JVM whether it be continuous or not!

Reason 1: Objects are stored in heap memory.

Reason 2: In JLS (Java language specification) it is mentioned that heap objects are not continous.

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

⇒ Index of an array:



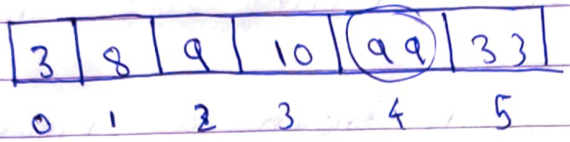

arr[0] = 3 arr[2] = 9 arr[4] = 53

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

Suppose to change the value of certain index:

arr[4] = 99

new array will be



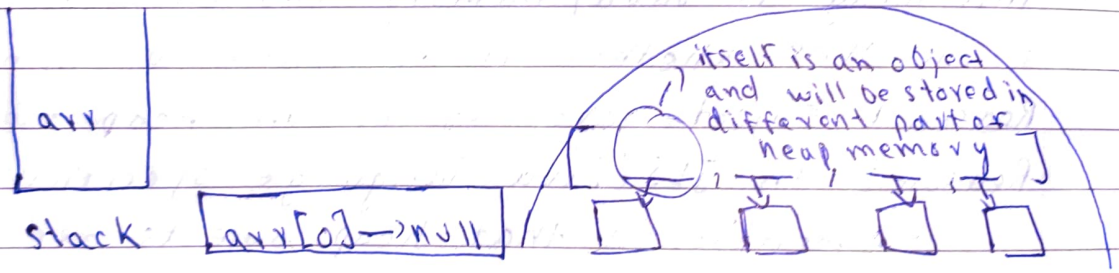

⇒ new keyword: → used to create an object

int[] arr = new int[5];

it will create an object in heap memory of array size 5

⇒ 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];

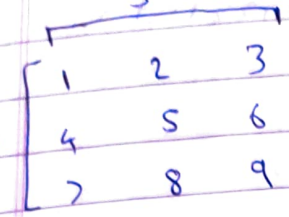



* primitives (int, char, etc) are stored in stack
* All other objects are stored in heap memory

⇒ Arrays to string (array) → internally uses for loop and gives the output in proper format

* In an array, since we can change the objects, hence they are mutable.
* strings are immutable.

⇒ 2D Array :




⟹ int [ ][ ] arr = new int [size][ ]

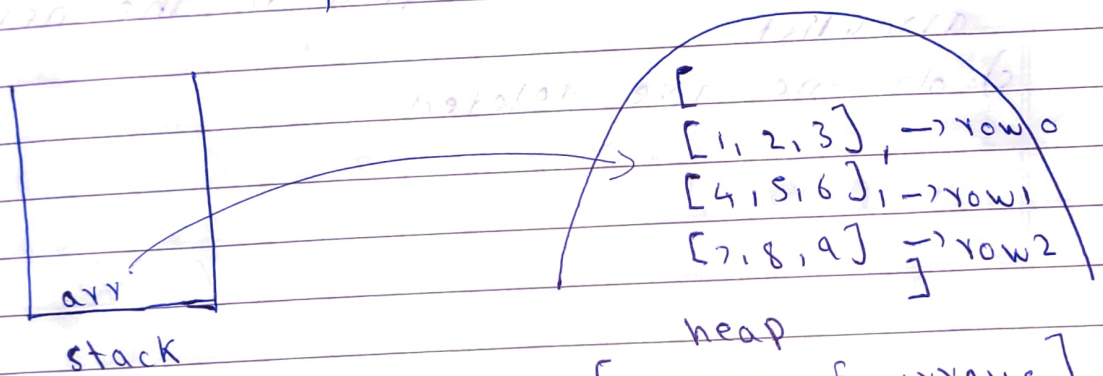
row column

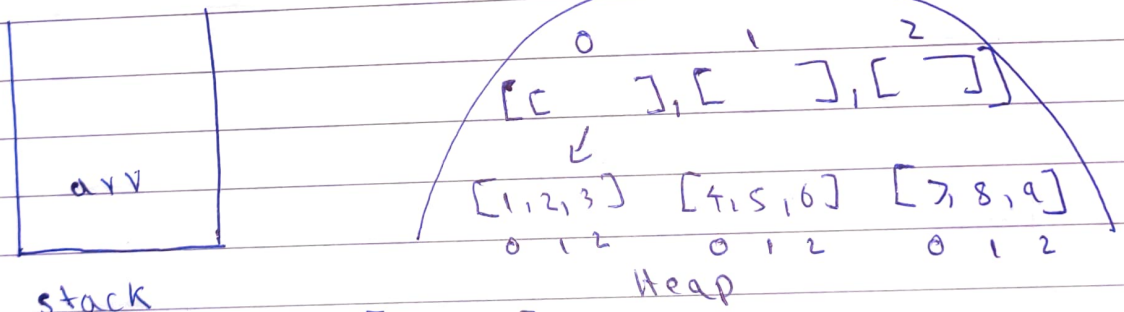
mandatory to give size of row

not mandat

OR

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




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

arr [0] [2] = 3

⇒ Arraylists :

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 wrappers

=) 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

* Ways to print array

- For loop

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

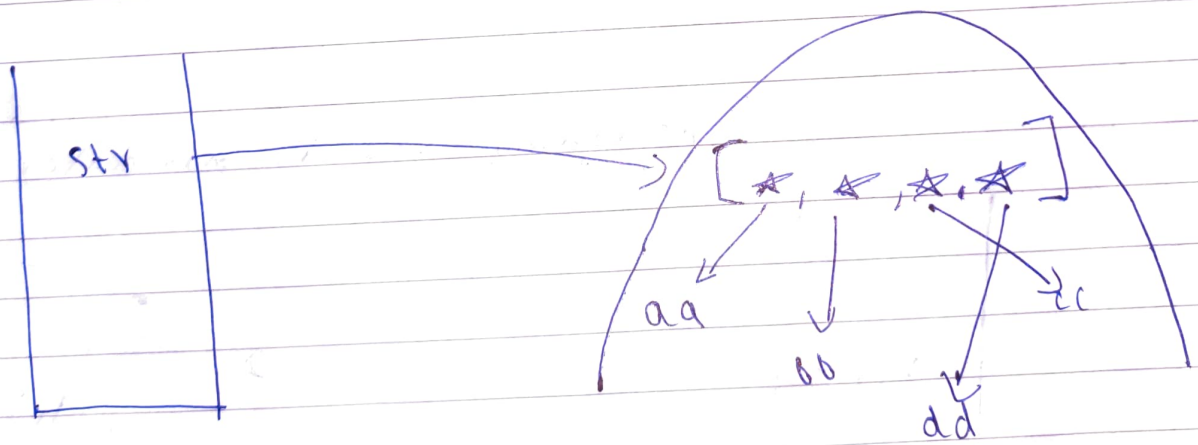
- To string

```
System.out.println(Arrays.tostring(arr));
```

* array of objects

```
String[] str = new string[4];
for(int i=0 ; i < str.length ; i++){
    str[i] = in.next();
}

System.out.println(Arrays.to string (str));
```



* Passing array in functions

```
Public static void main(String[] args){
    int[] nums = {3,4,5,12};
    System.out.println(Array.toString(nums));
    change(nums);
    System.out.println(Array.toString(nums));
}
static void change(int[] arr){
    arr[0] = 99;
}
```

* Strings are imutable in java and
* Arrays are mutable in java since we can change the object

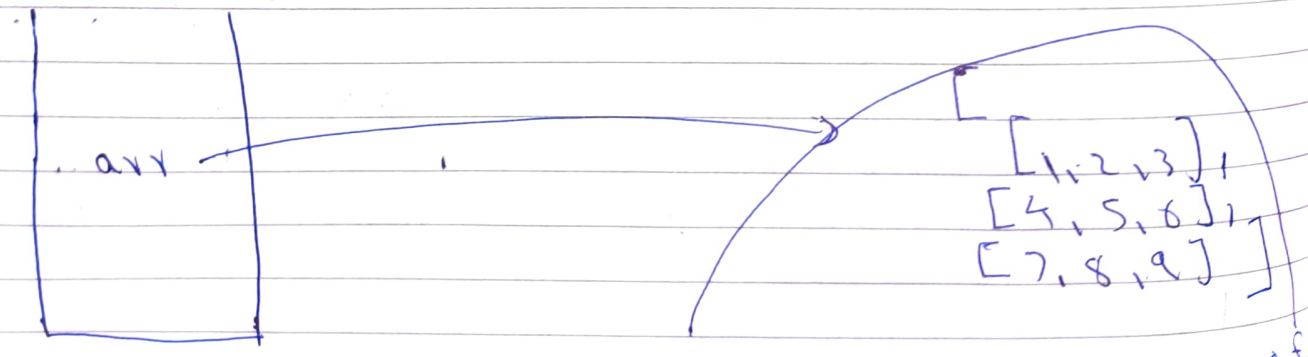
* 2D Arrays

Syntax

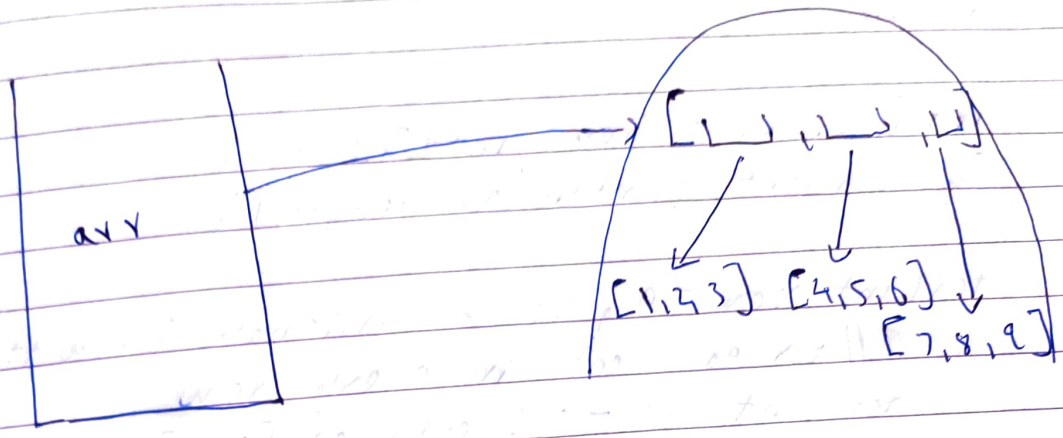
```
int[][] arr = new int[3][];
```

not mandatory

Assigning row is mandatory




Imagine this as an array of arrays




arr[1] = [4,5,6]
arr[1],[0] = 4

- Column ~~is~~ size is not mandatory

[ [1,2,3]
  [4,5]
  [6,7,8,9]
]

* Input of 2D-Array.

1st way

```
int[][] arr = {
                {1,2,3},
                {4,5},
                {6,7,8,9}
              };
```

2nd way

```
int [][]arr = new int[3][2];
System.out.println(arr.length);

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 of 2D Array

- 1st way

```
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();
}
```

- 2nd way

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

* Arraylist

Arraylist is a part of collection framework and is present in java.util. package. It provide us with dynamic arrays in Java. It is slower than standard array.

Syntax :-

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

↓ add wrappers

- Inputs/Output

~~list~~

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

list.add(67);
list.add(67);
  :
  :

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

Functions :-

① list.contains( );
② list.set(0, 99);
③ list.remove(2);

2nd way

```
for (int i =0 ; i<5 ; i++){
    list.add(in.nextInt());
}

for(int i=0 ; i<5 ; i++) {
    System.out.println(list.get(i));
}
// list[index] syntax will not work 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 array - list
c) old ones are deleted.

* Multidimensional array list

Syntax :-

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

```
// initialisation
for(int i = 0 ; i < 3 ; i++) {
    list.add(new ArrayList<>());
}

// add elements
for(int i = 0 ; i < 3 ; i++){
    for(int j = 0 ; j < 3 ; j++){
        list.get(i).add(in.nextInt());
    }
}
System.out.println(list);
```