27-11-21

→ **Arrays**: Data structure use to store collection of same type of data.

why array :- To handle large storage of data, in almost all programming language.

Syntax -

datatype [] variable_name = new datatype [size];
(size → array size)

or

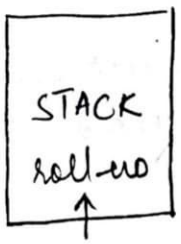
datatype variable_name [] = { data, data, };

Working of Array internally -

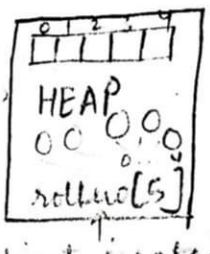
```
int rollno;          // declaration of array
rollno = new int[5]; //initialization
```

execute: compile time | run-time (dynamic memory allocation)

* we can change array, they are mutable



reference variable declared

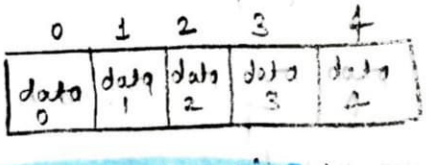


object created with size [5] starting from 0.

* primitives → stack
* objects / non-primitive → heap

- variable Rollno are getting defined in stack
- Actual memory allocation happens in heap.
  size & data

: Array memory indices starts with 0



Dynamic Memory allocation: Assigning the memory space during execution time and run-time

** In java, array memory allocation may not be continuous or continuous which totally depends upon JVM as stack & heap is handled by JVM.

→ Why depends on JVM?

Reason1 - obj. stored in heap memory

Reason 3 - Dynamic memory allocation

Reason2 - In Java language specification it is mentioned that heap objects are not continuous.

→ 'new' keyword : use to create an object.
It creates object in heap memory.

→ 2D Array :- visualised as a matrix

col ↓
row → 1 2 3
4 5 6
7 8 9

`int[][] arr = new int[size][];` (row size not mandatory, column)

(Row size is mandatory)

or

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

(col index: 0 1 2; row index: 0 1 2)

Storage can be imagined as array of array i.e. array inside array.

• length : Array method to measure length (size of array)

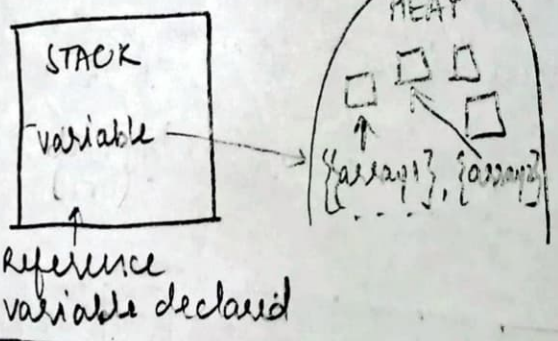
Syntax –

`datatype[][] variable_name = new datatype[row size][col size];`

or

```
datatype[][] variable_name = {
    {array1},
    {array2},
    ....
    {arrayN}};
```

Internal working of 2D Array



Reference variable declared

NULL

* Default value of any variable is null
* Null cannot be assigned to any primitive data type.

• toString :

.toString : Array class method converts array to string.
Syntax – Array.toString(arr)

→ ArrayList :- part of collection framework, present in java.util.package.

- provides dynamic arrays in Java.
- slower than standard arrays.

Why ArrayList :- if we don't want to set fixed size of length of array or want to update size in future.

Syntax –

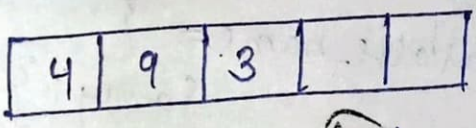
```
ArrayList<Integer> list = new ArrayList<>(1);
```

- ArrayList : inbuilt class
- Integer : Datatype (Wrapper class) — cannot add primitives
- list : reference variable
- new ArrayList<>(1) : ArrayList constructor (size)

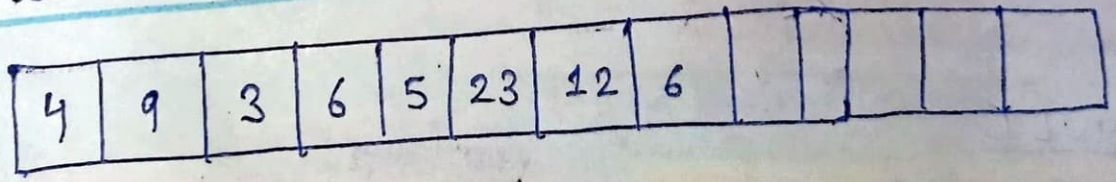
Internal working

* Size is fixed internally but not permanently. It varies according to the user input.

for eg: Inputted 5 data



Then you want to input 5 more.
But we do not have space
so java create a new list with new size (more than initial) to store new elements/data.
Then it'll copy the old list to new list and delete old list.



* Array is a non-primitive type of datatype.
So array can be a return type of fy and method.

Example –

```
public int[] arr(){
        datatype
int a1=20; int a2=23; int a3=87;
return new int[]{a1,a2,a3};} //returning new array
                              of type int storing
psvm(String args[]){          [a1,a2,a3]
int[] n = arr();
//invoking method and storing
returned array into n.
```

* Why, we can use Arraylist as returntype as it is ~~also~~ dynamic array. Type of array only

Example.

```
ArrayList<Integer> funame(){
datatype
ArrayList<Integer> list = new ArrayList<>();
int a = 1; int b = 2;
list.add(a);
list.add(b);
sout(list);
return list;}
```

ArrayList with Recursive fns

[24-01-22]

* ArrayList return type by passing parameters

Example-

```
ArrayList<Integer> find(int[] arr, int target, int index,
                        ArrayList<Integer> list){
```

ArrayList<Integer> — return type

ArrayList parameter → passing new list

```
if (index == arr.length){
    return list; }
if (arr[index] == target){
    list.add(index); }   //Adding result in list
return find(arr, target, index+1, list);
}
```

By passing list in fn params, obj is created once and with every call only reference is passed of same list object. So result of every call is saved in that list only.

* ArrayList return type without passing parameters.

Example-

```
ArrayList<Integer> find(int[] arr, int target, int index){
    ArrayList<Integer> list = new ArrayList<>();
    if (index == arr.length){
        return list; }
    if (arr[index] == target){ // this will contain answer for
        list.add(index); }     that fn call only
    ArrayList<Integer> answer = find(arr, target, index+1);
    list.addAll(answer); // here we are storing result of
    return list; }          recursive call in answer list.
                            And returning list as it
                            contains answer list data.
```

* Agar hum isko direct karte to har bar naya object create hoga or different list banayegi har fn call ke liye.