Arrays

Syntax →

datatype[] var_name = new datatype[size];

or datatype[] var_name = {-----};

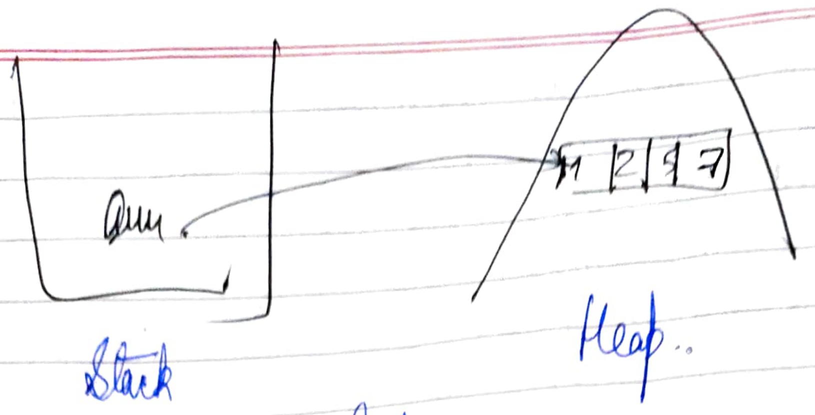
10th feb

$$\underbrace{\overset{\text{datatype}}{\text{int[]}}\ \overset{\text{ref variable}}{\text{arr}}}_{\text{happens at compile time}} = \underbrace{\overset{\text{new is used to create an object}}{\text{new}}\ \text{int[5]}}_{\text{happens at run time}}$$

— happens at run time

— This is known as dynamic memory allocation

— creating the object in heap memory



- array objects are in heap
- heap objects are not continous
- hence in java, array objects may not be continous.
  It all depends on the JVM!

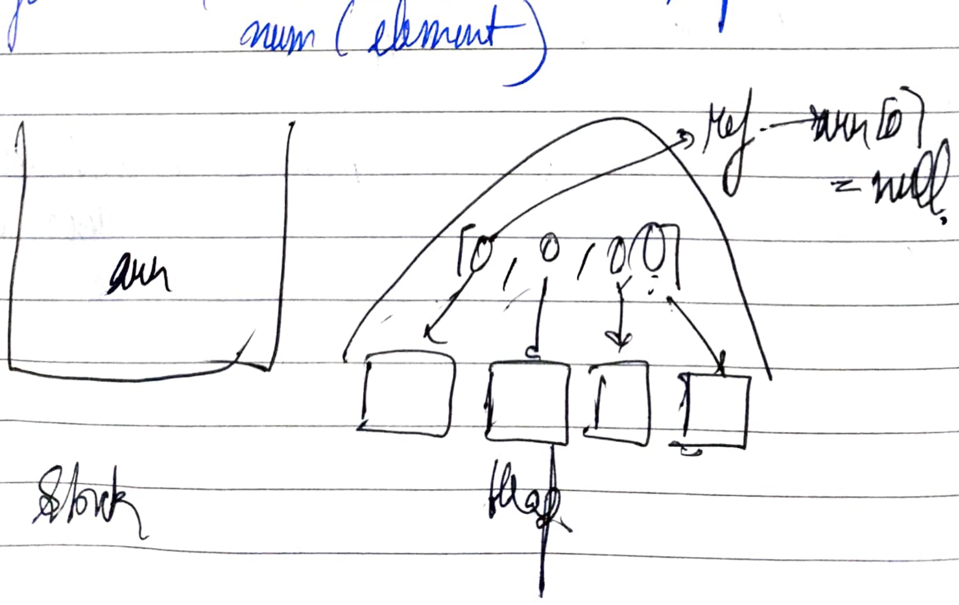
null →

String str = null; ✓
int num = null; ✗ this will give an error.

- null is a special value that is assigned to any reference variable (except primitives)

```
for( int num : arr){
    sysout( num + " " );
}
```

for every num(element) in arr, print num(element)



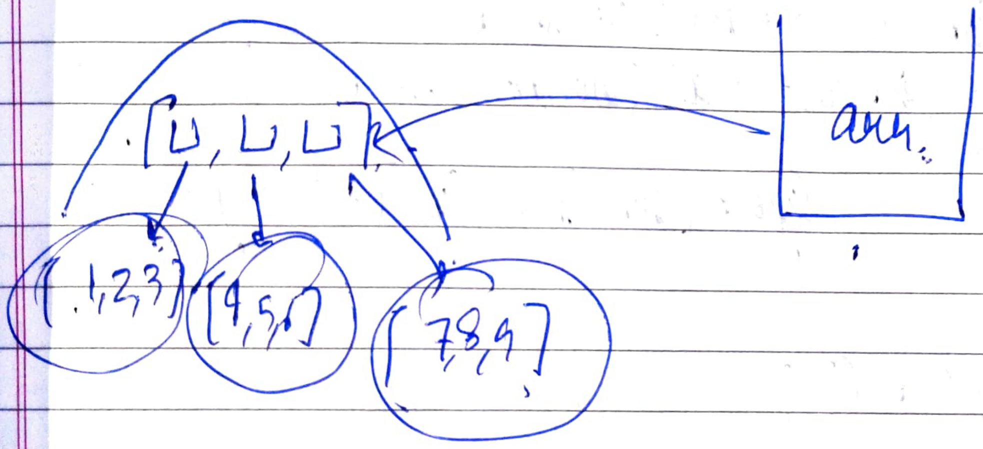
Multidimensional Array :→

→ int [][] arr = new int [] []

No. of Rows
Columns (Not Mandatory to add)

→ (Can be considered as an array of arrays).

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



→ int ~~2~~[][] arr2 = new int [3][];
~~int [][]~~ arr ~~2~~ = {
{1, 2, 3 },
{4, 5},
{6, 7, 8, 9}
};

→ print(arr.length) // no. of rows

→ // Input

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

arr[row].length

// output

```
for (int row = 0; row < arr.length; row++){
   for (int col = 0; col < arr[row].length; col++){
          Sysout (arr[row][col] + " ");
   }
   sysout.(); // prints a new line.
}
```

// output -

```
for (int[] a : arr) {
      sout ( Arrays.toString(a));
}
```

classmate
Date
Page

ArrayList →

- when we do not know what the size of our Array will be that's when we use. ArrayList

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

Integer → generics
- we can't use primitives
- we have to use wrapper class

() → initial capacity

```
→ list.add(23);
→ list.add(10);
  .
  .
→ sout(list);
```

list has several methods inside of it.

```
list.set(0, 99);
```

0 → index, 99 → value.

- size of our ArrayList is fixed internally.
- when the ArrayList fills up to particular amount, a new ArrayList of bigger size is created.
- old elements are copied into the new ArrayList
- The old ArrayList is deleted.

} Time complexity is Constt. O(1).

Multidimensional ArrayList -

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

// initialize
// input values
// output -