Arrays I

Arrays is a collection of datatype.

Store a roll no. ⟶ int a = 19;

Store a name ⟶ String name = "Kunal Tamhane";

Store 5 roll nos. ⟶ int roll1 = 23;
int roll2 = 24,
int roll3 = 63,
int roll4 = 34,
int roll5 = 44;
.
.
.

} rollnos

Store 500 roll nos

Syntax of Array.

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

Q) Store five roll numbers using arrays.

Method 1.

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

Method 2.

```
int[] rnos2 = {23, 44, 34, 63, 33}
```

```
int [ ] rnos = new int [5]
```

keyword to create an object

means rnos reference variable is pointing to an array object that containts type integer

this datatype represents, the type of data stored inside the array.

- All the type of data in the Array should be same.
  that means if first element is an integer all elements of the array needs to be integers.

declaration

`int [ ] ros,` ⟶ declaration of array ros is getting defined in the stack

initialization

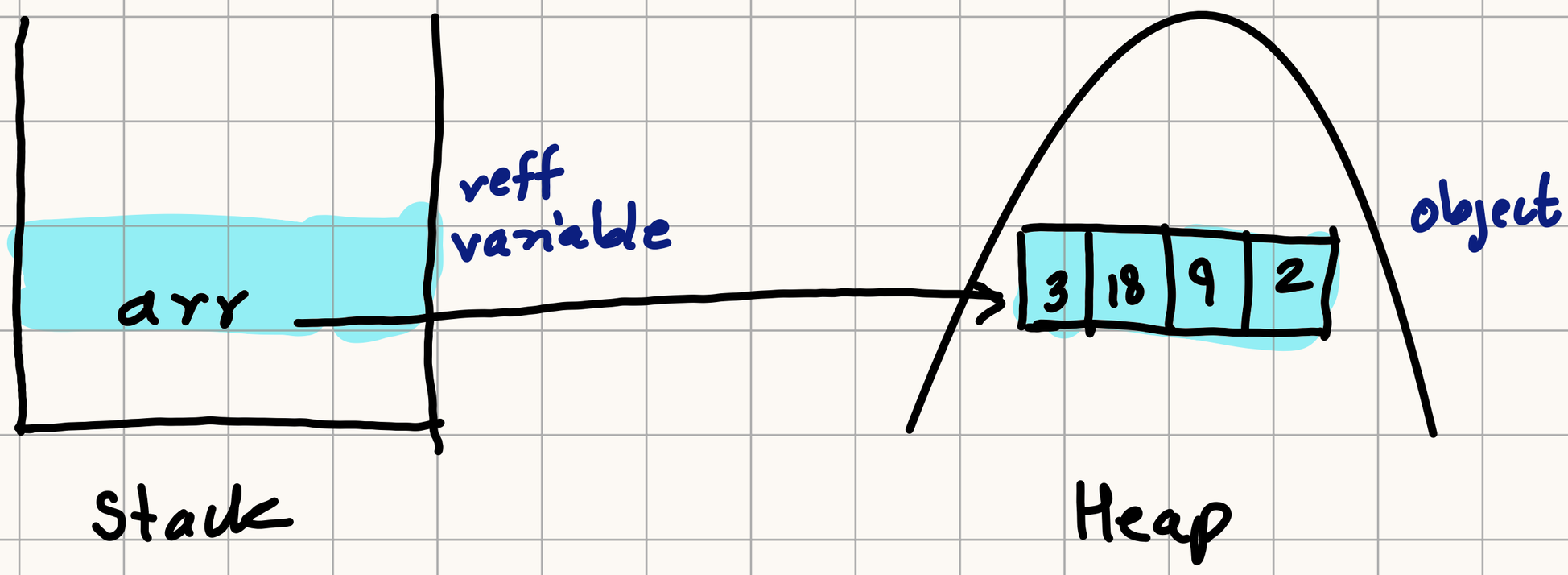
`ros = new int [5]` ⟶ actually here object is being created in heap

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

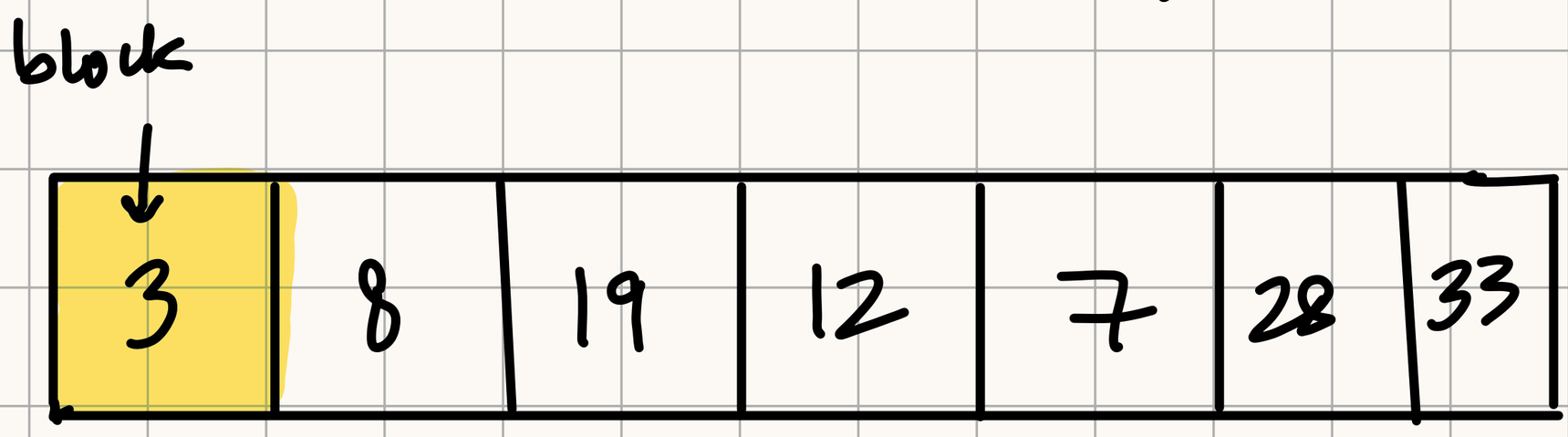
datatype — reference variable — compile time

creating the object in heap memory. — Run time

Dynamic memory allocation.

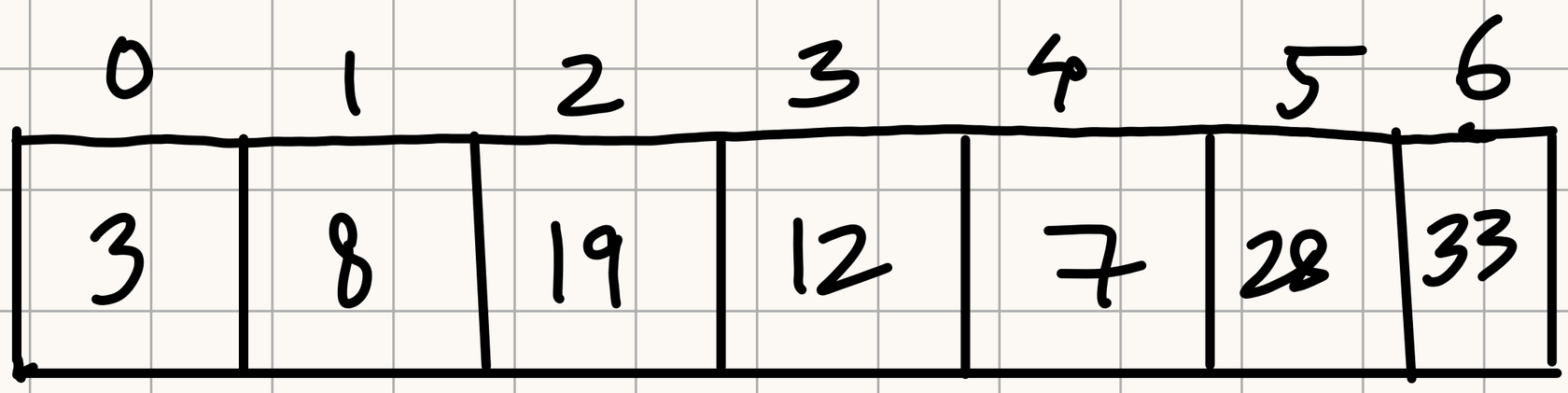


Internal Representation of an Array



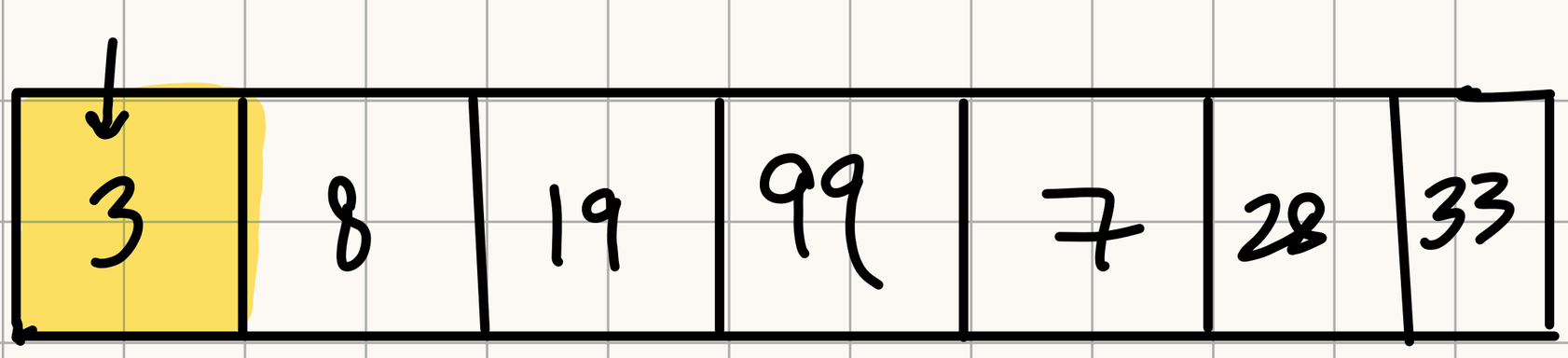
- In C/C++ array it is continious memory allocation.
- In Java array objects are in heap
- In heap memory is not stored continously (heap objects are not continous)
- Declaration happens at compile time while creation of array memory happens at runtime (Dynamic memory Allocation)
- Hence Array objects in Java may not be continous

Indices of array



print (arr[0]) ⟹ 3
print (arr[2]) ⟹ 19

arr[3] = 99

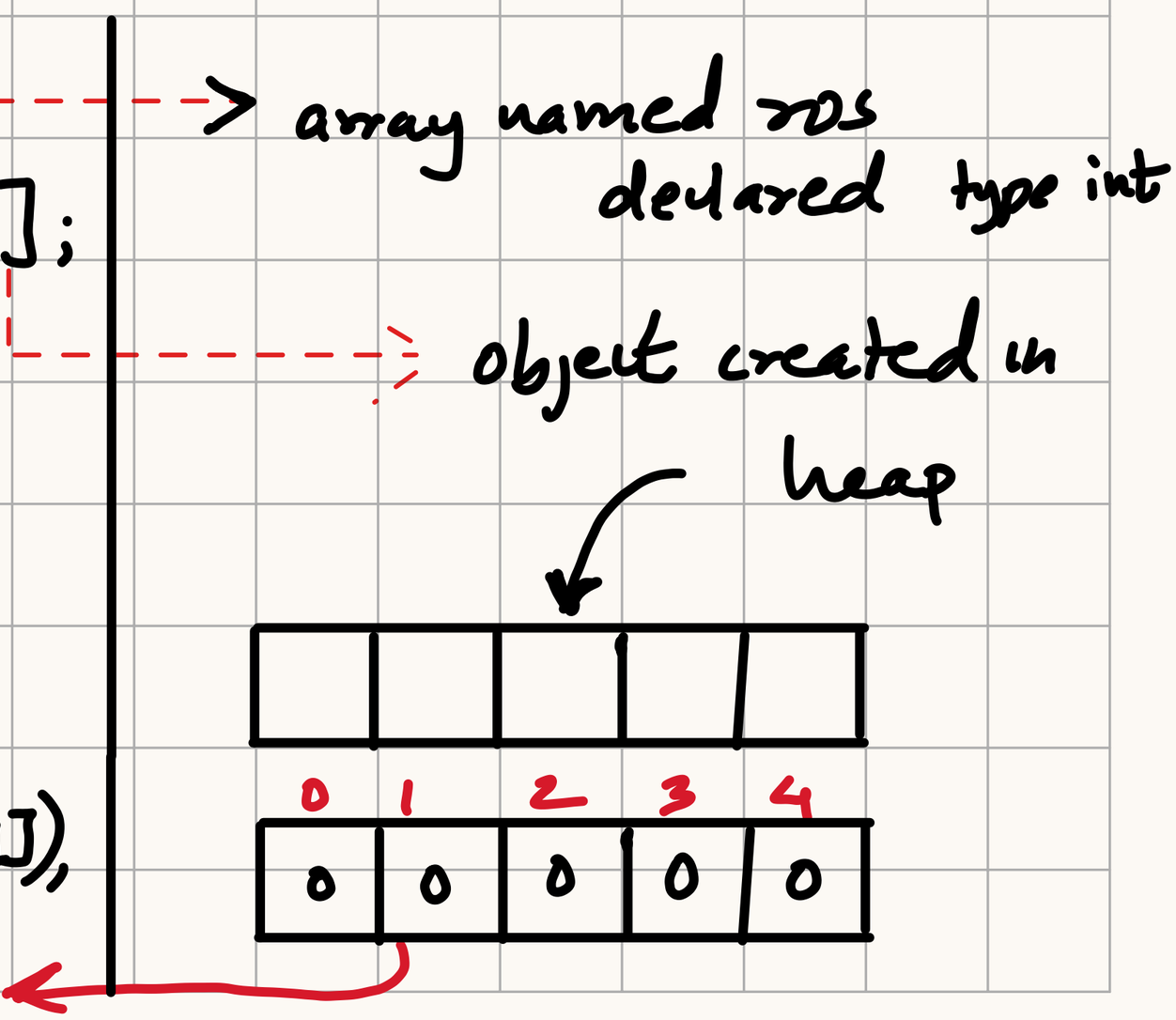


Empty Arrays:

int [] ros,
ros = new int [5];

System.out.println (ros [1]);

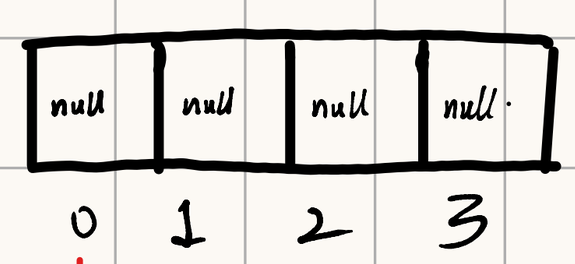
output → 0



```
String [] arr = new String [4];
System out println (arr [0]),
```

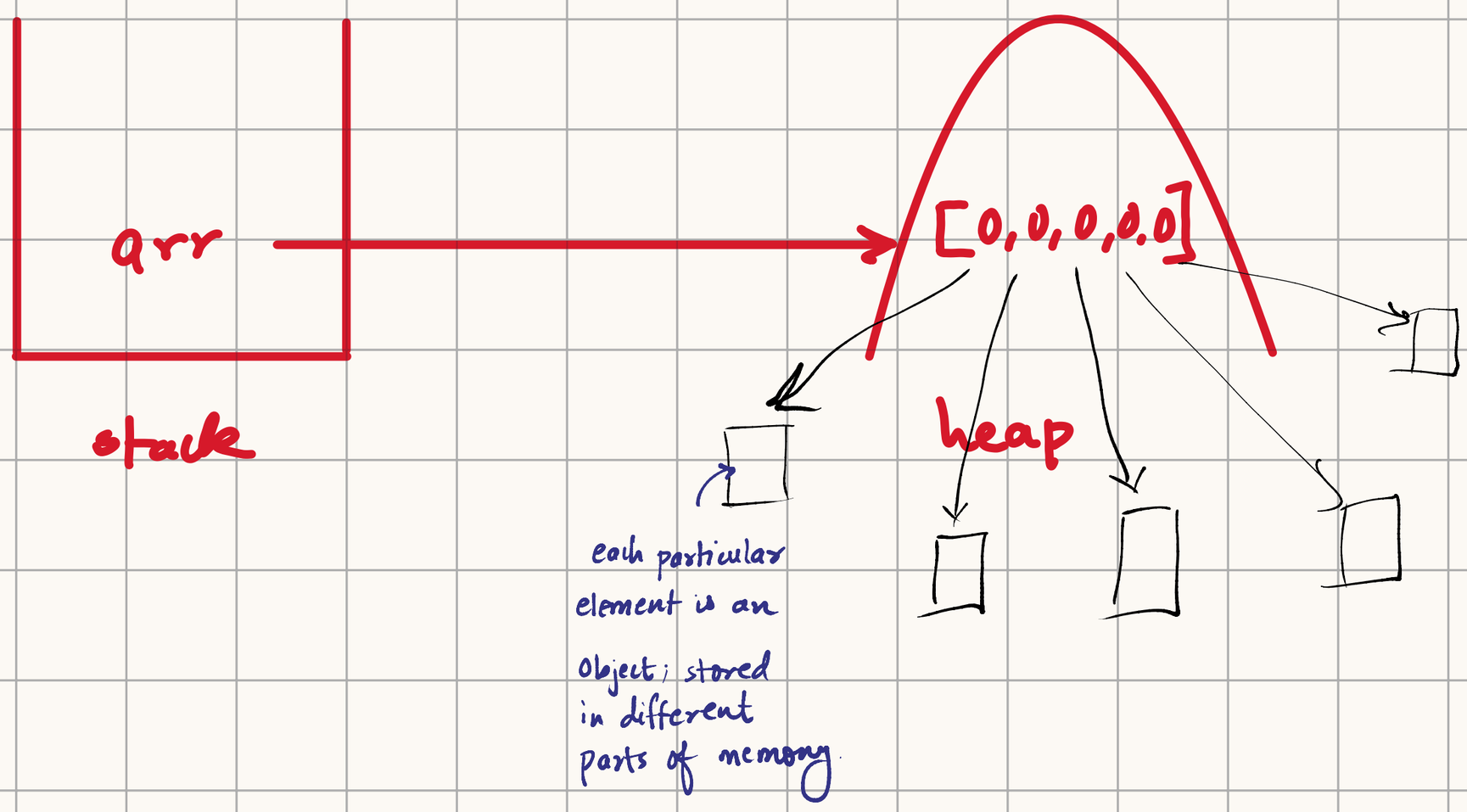
Output → null.

array named arr created and declared with datatype string and size 4



null used as default

```
String [ ] arr = new String [5];
// internal writing of object
```



Array Input

```
public class Input {
        public static void main (String [] args) {
         int [] arr = new int[5],
         arr[0] = 23,
         arr[1] = 45;
         arr[2] = 233,
         arr [3] = 543;
         arr [4] = 3,
         System.out println (arr[3]),
    }
}
```

Input using Loops:

```
public class Input {
    public static void main (String [] args) {
        Scanner in = new Scanner (System.in);
         int[] arr = new int [5];
        for (int i = 0, i < arr length ; i++ ) {
            arr [i] = in next Int ();
```

Printing an Array:

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

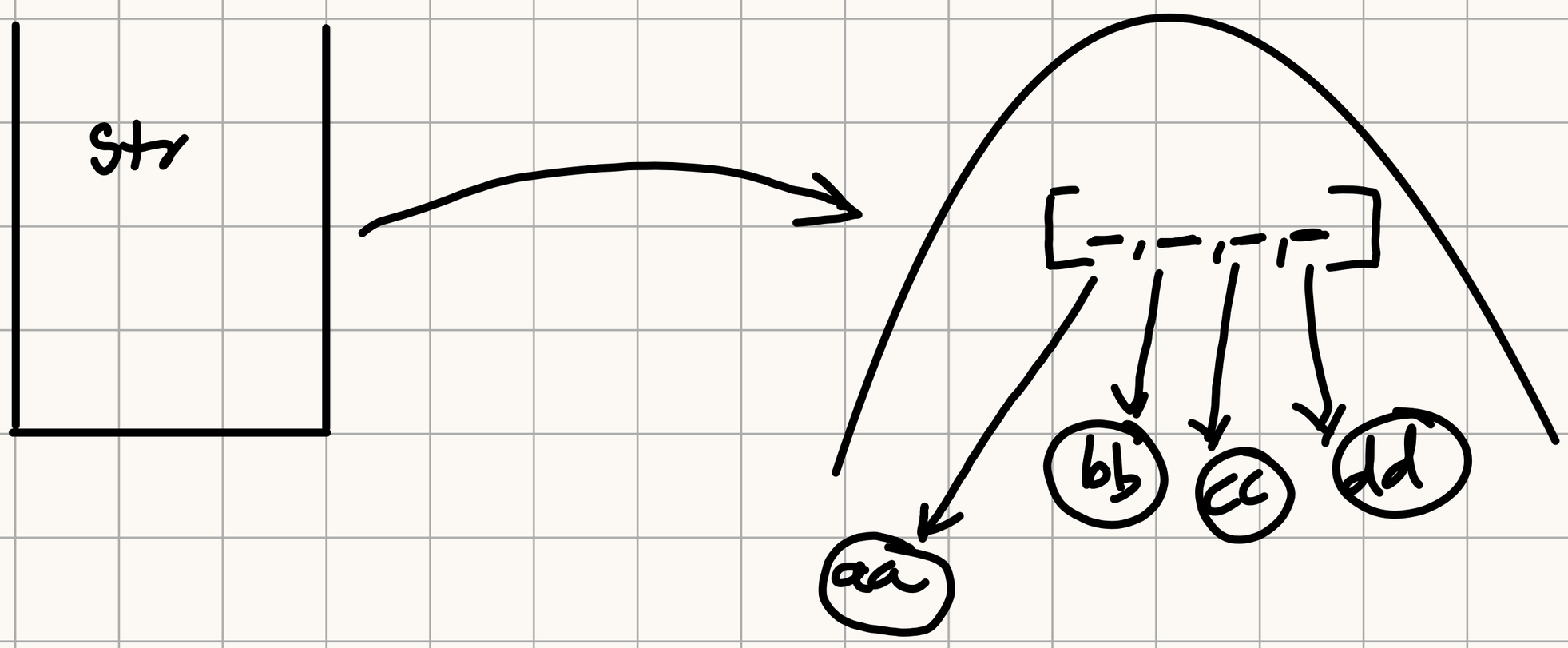
b)
```
for (int num : arr) { // for every element in array print elem
        System.out.println (num+ " "),
}       // here num represents element of array
```

c)
```
System.out.println (Arrays.toString (arr)),
```

Array of objects

```
public class Main {
        public static void main (String[] args) {
            String[] str = new String [4],
            for (int i = 0; i < str.length ; i++ ) {
              str[i] = in.next ( );
        }
          System.out.println (Arrays.toString (str)),
    }
}
```

Storage of Objects in Heap.



Modifying String Array .

```
public class Main {
    public static void main (String[] args) {

        String[] str = new String [4],
        for (int i = 0; i < str Length ; i++ ) {
            str [i] = in next ( );
    }
     System.out println (Arrays to String (str)),
     str [1] = "kunal",
     System out println (Arrays to string (str));
}
}
```

Array Passing in Functions (Strings are immutable, Arrays are mutable)

```
public class Passing in Functions {
    public static void main (String [ ] args ) {
        int [ ] nums = {3,4,5,12};
        System out println (Arrays to String (nums));
        change (nums);
        System out println (Arrays to String (nums)),
    }

    static void change (int [ ] arr ) {
        arr [0] = 99,
} }     // Output [99, 4, 5, 12].
```

Multidimensional Arrays:

```
public class Main {
    public static void main (String[] args) {
        int[][] arr = new int[3][3],
        // note: adding a row is
        mandatory in 2D array
    }
}
```

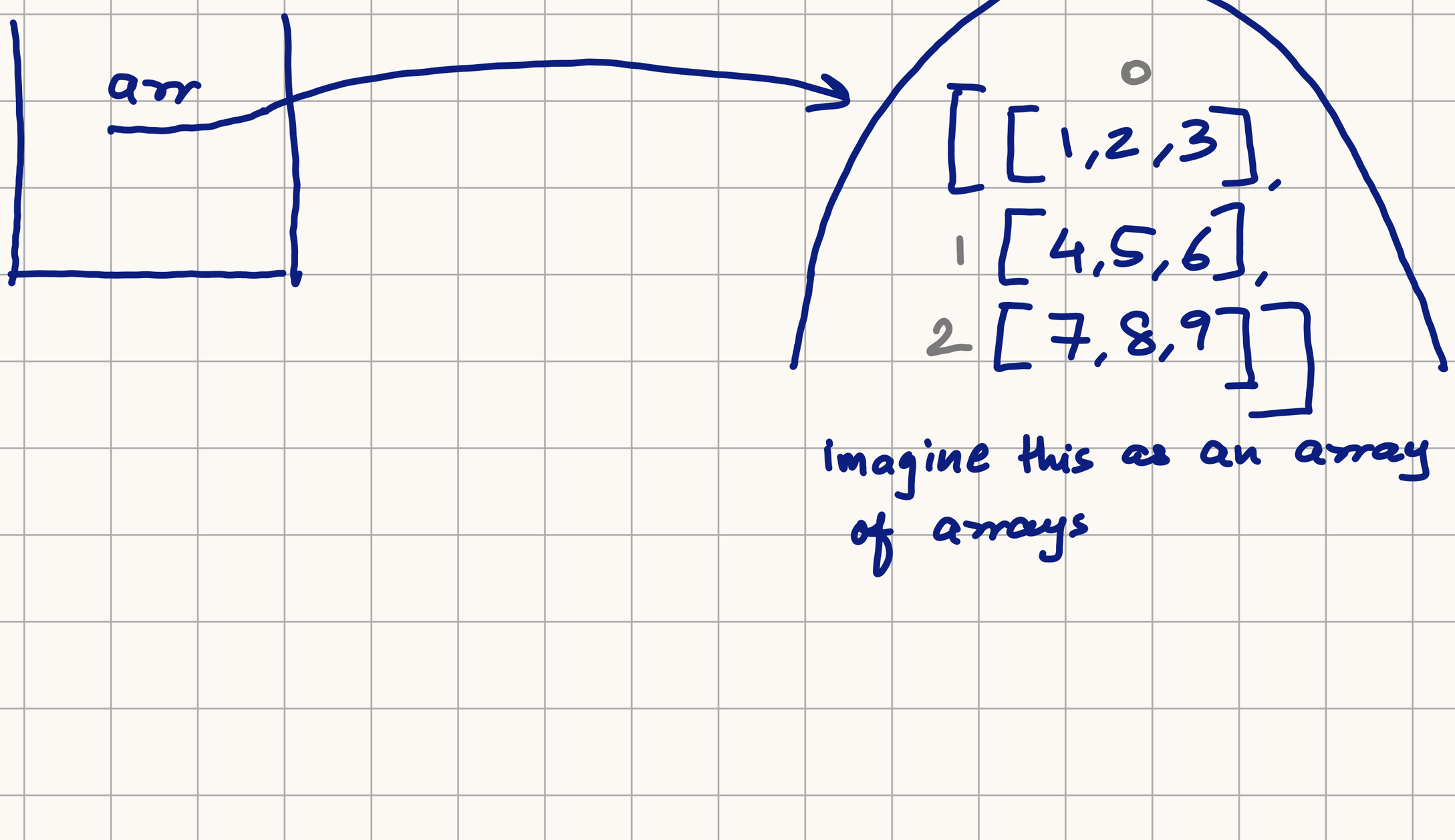
Syntax.

row ↑ column ↑

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

—— output.

Internal working of 2D Array:



Q) Write a program to create a multidimensional array like given below

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

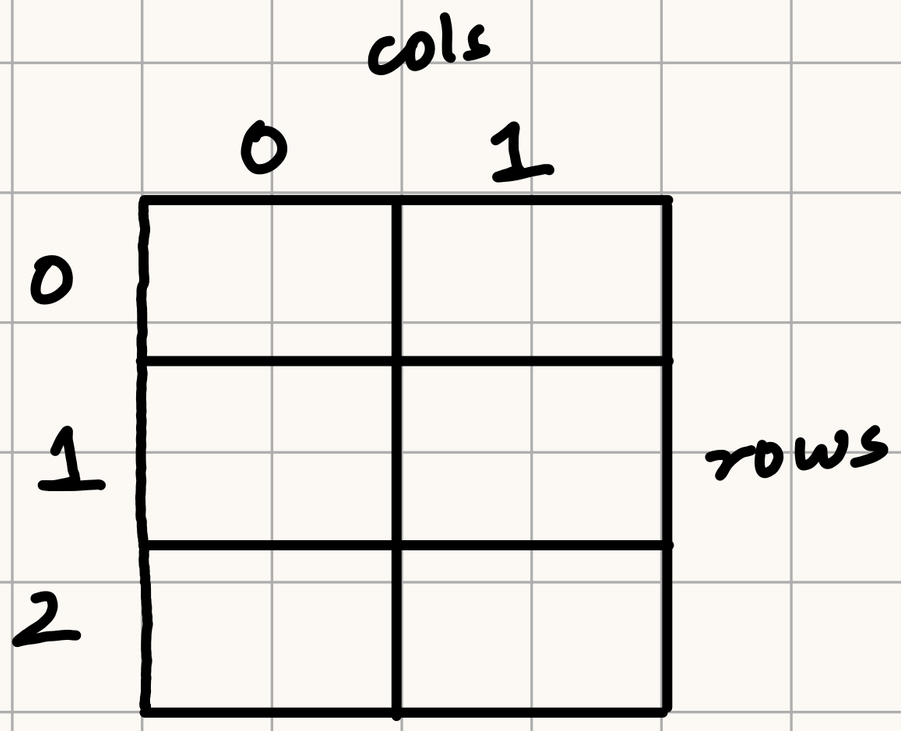
```
public class Main {
    public static void main (String [] args) {
        int [][] arr2D = {
                        {1, 2, 3},
                        {4, 5}
                        {6, 7, 8, 9}
                        };
    }
}
```

2D Array Input:

```
public class Main {
    public static void main (String [] args) {
        Scanner in = new Scanner (System.in);
        int [][] arr = new int [3][3];
        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();
} }
```

* Internal working of above program.




```
for (row = 0, row<3, row++){
    // now we take every row
    for (col = 0, col<2; col++){
        arr[r][c] = input
} }
```

row	col
~~0~~	~~0~~ ~~x~~ 2
~~x~~	~~0~~ ~~x~~ 2
~~2~~	~~0~~ ~~x~~ 2

arr[row] length

```
for (r=0, r<3, r++)
  for (c=0, c< size(r), c++)
      input
} }
```

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

OR

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

OR

(by enhanced for)

```
    for (int[] a · arr) {
            System out println(Arrays to String (a));
        }
    }
}
```

Dynamic Array: (Column not fixed size)

```
public class COLNofixed {
    public static void main (String[] args) {
        int [][] arr = {
            {1, 2, 3, 4},
            {5, 6},
            {7, 8, 9}
        },

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 ();
    }
}
}
```

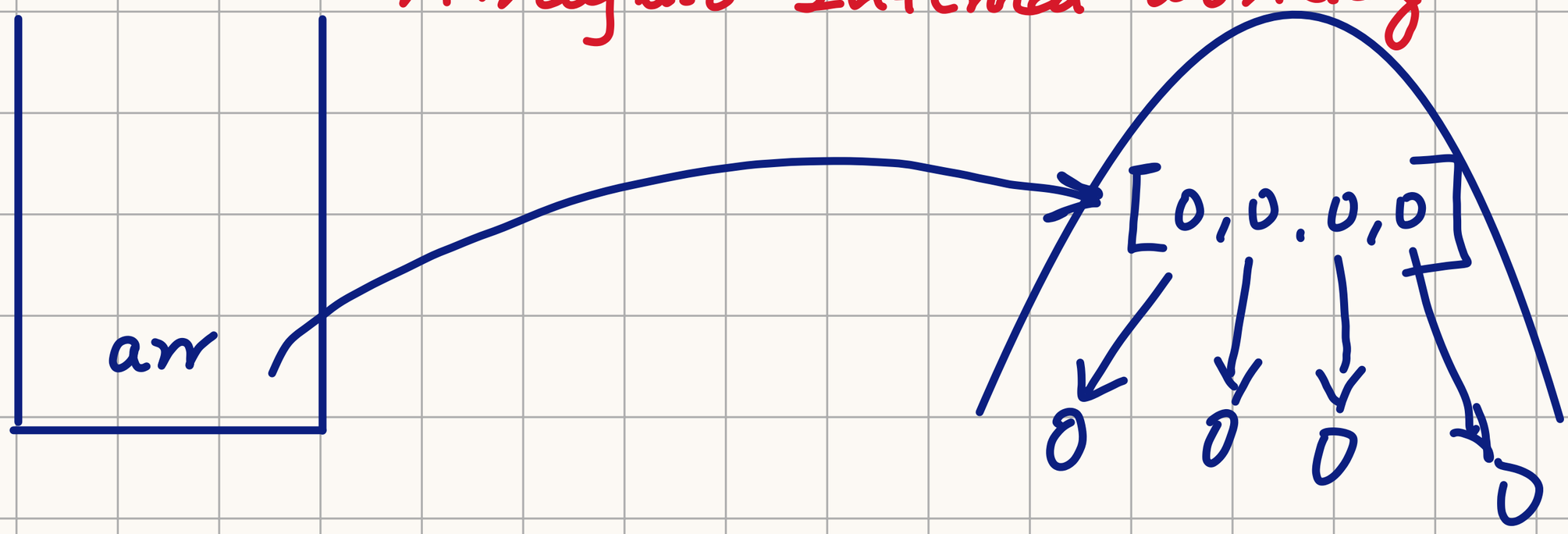
ArrayList.

```
//Syntax:
ArrayList<Integer> list = new ArrayList<          >(
                                        initial Capacity.10);

    list add (67);
    list add (68),
    list add (34),
    List add (204);
    list.add (112),
   List add (208);
   lut add (202),
  List add (111);
    list add (67);
    list add (68),
    list add (34),
   List add (204);
    list.add (112),
  List add (208);
   lut add (202),
 List add (111);

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

Arraylist Internal working



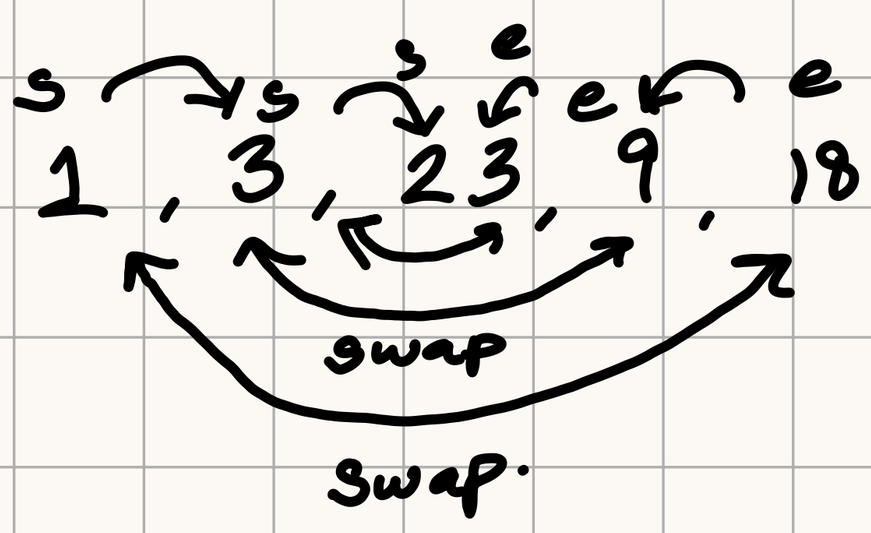
① Size is fixed internally.

② Say arraylist fills by some amount

⇒ It will create a new arraylist of say double the size.

⇒ old elements are copied in new o

Reversing an Array:



Q) Write a program to swap an array

```
public class reverse {
    public static void main (String[] args) {
        int [] arr = {1,3,23,9,18};
        reverse (arr).
```

```
        System.out.println(Arrays.toString(arr));
    }

    static void reverse(int[] arr){
        int start = 0;
        int end = arr.length-1;

    while (start < end){
            swap(arr, start, end);
            start++;
            end--;
        }
    }

static void swap(int[] arr, int index1, int index2){
        int temp = arr[index1];
        arr[index1] = arr[index2];

        arr[index2] = temp;
    }
}
```