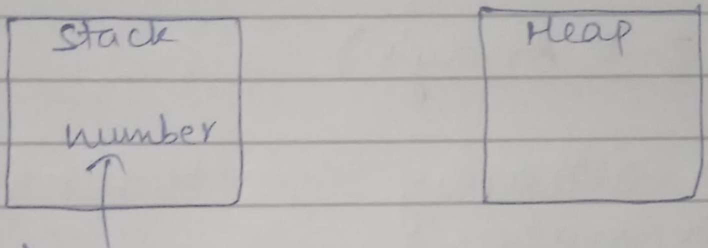
Arrays (1-D)

An array is a collection of similiar datatype values.

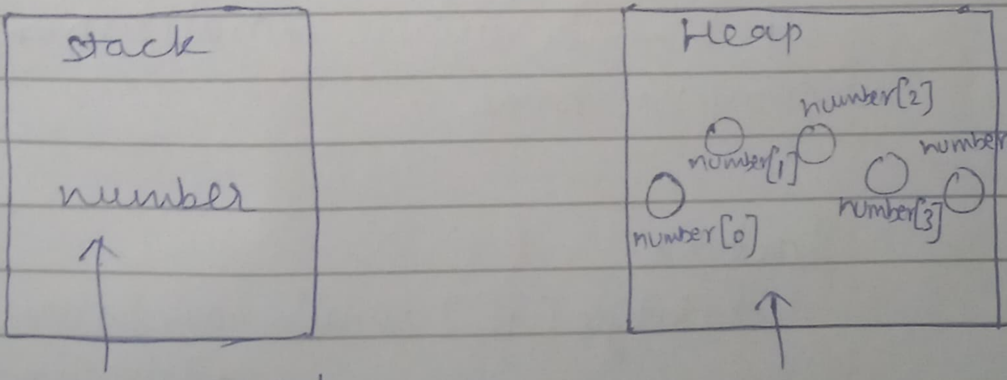
datatype[] variable_name;

Example→ int[] number;



Reference variable initialized (compile time)

int[] number = new int[5];



Reference variable

A new object is created with a size of 5 reference variables.

(Runtime / Dynamic memory allocation)

Note: Heap objects may/may not be continuous.

Objects are stored in heap, eg. array, string etc.
Primitive are stored in stack eg. int char etc.

As String is itself a class & create objects, it is also stored in heap.

(2-D Arrays)

A 2-D array can be visualized as a matrix.

```
int[ ][ ] num;
    ↑   ↑ columns
  rows
```

Reference variable initialization at compile time.

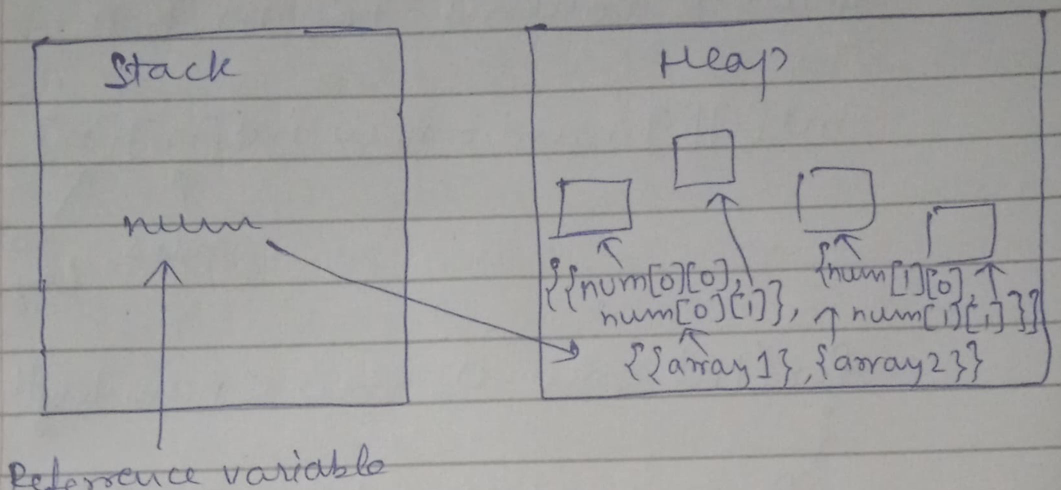
Syntax:

```
datatype[][] variable_name = new
                  datatype[row_size][col_size];
```

OR

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

int[ ][ ] num = new int[2][2];

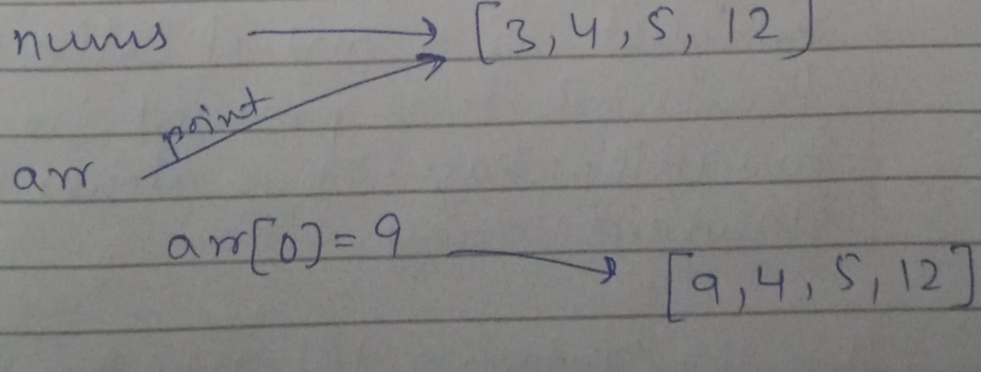


Note - ⊙ No. of rows mentioning is mandatory.

⊙ arr.length returns the number of rows in array arr.

⊙ arr[row_number].length returns the number of columns in that particular row.

⊙



∵ Arrays are mutable (can be changed).

* An array (2-D) can be of different rows & colums i.e., m & n.

```
int[][] num = new int[m][n];
                      ↑   ↖
                    rows   colums
                           (dynamic)

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

⇒ Arrays.toString(arr) → internally uses for loop and gives the output in proper format.

Arraylists (1-D)

Arraylist is a part of Collection framework & is present in java.util package. It is slower than standard arrays.

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

Internal working –

- 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.

```
list.add(67);
list.add(34);
list.add(25);
list.add(46);
```

```
System.out.println(list); →[67,34,25,46]
```

```
System.out.println(list.contains(654));
```

- It checks whether list contains the mentioned value and returns true/false

```
list.set(0,99);
      ↑    ↖value
    index
```

```
list.remove(2);
            ↑ index
```

```
list.get(i); → returns value at index i
```

2-D Arraylists

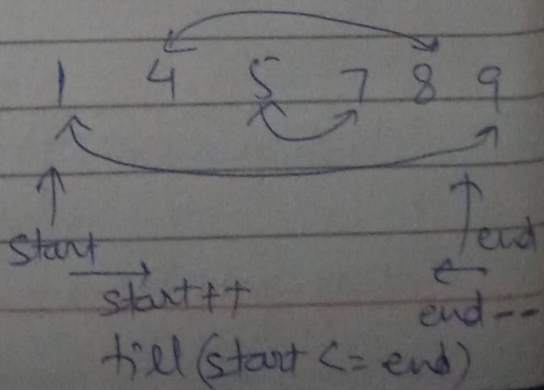
```
ArrayList<ArrayList<Integer>> list =
        new Arraylist<Arraylist<Integer>>();

Scanner in = new Scanner(System.in);

//initialization
 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);
```

Q.) (Based on Arrays)
Reverse the array without using reverse function & no extra space.



```
scanner s= new Scanner (system.in);
int n= s.nextInt();
ind[] arr = new int[n];

for(int i=0; i<n; i++){
    arr[i]=s.nextInt();
}

for(int start=0, end=n-1; start<=end; start++, end--)
{
    swap(arr, start, end);
}

public void swap(int[] arr, int s, int e){
    int temp= arr[s];
    arr[s] = arr[e];
    arr[e] = temp;
}
```

Linear Search

Searching - It is a process of finding a given value position in a list of values.

Linear/ Sequential Search -

- Comparison of target value with all the other elements given in the array.

```
             0    1    2    3    4    5
eg.→ arr = [18 , 12 , 19 , 77 , 29 , 50]  (unsorted array)
             ↑                ↘
           start               return
```

target = 77

In above example, the target value is compared with all the elements in array in linear way.

eg.→ Search in String (1-D)

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

        String str= "mayank";
        char target= 'a';
        System.out.println(search(str,target));
    }

    static boolean search(string str, char target)
    {
        if(str.length==0){
            return false;
        }
        for(int i=0; i<str.length(); i++){
            if(target==str.charAt(i)){
                return true;
            }
        }
        return false;
    }
}
```

for loop can be enhanced -

```
for(char ch: str.toCharArray()){
    if(ch==target){
        return true;
    }
}
```

Note - ⊙ Min/Max problems are solved using Linear Search concept.

Q.) Find no. of numbers that has even no. of digits.

```
int evenDigits(int[] nums){
    int ans=0;
    for(int i=0;i<nums.length;i++){
        if(even(nums[i])){
            ans++;
        }
    }
    return ans;
}
```

```
boolean even(int num){
    int countDigits=0;
    if(num<0){
        num *=-1;
    }
    if (num==0) {
        countDigits = 1;
    }
    while (num>0) {
        CountDigits++;
        num /=10;
    }
    return countDigits %2==0;

}
```

- To count no. of digits, we can also use log

```
static int digits(int num) {
    if(num <0)
        num *= -1;

    return (int)(Math.log10(num))+1;
}
```

- If its about binary digits → log 2 will be used.

2D array Linear Search

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

        if(arr[row][col]== target) {
            return new int[]{row, col};
        }
    }
}
return new int[]{-1,-1};
```