Arrays

In C & C++, Array is defined as a continuous allocation of memory.

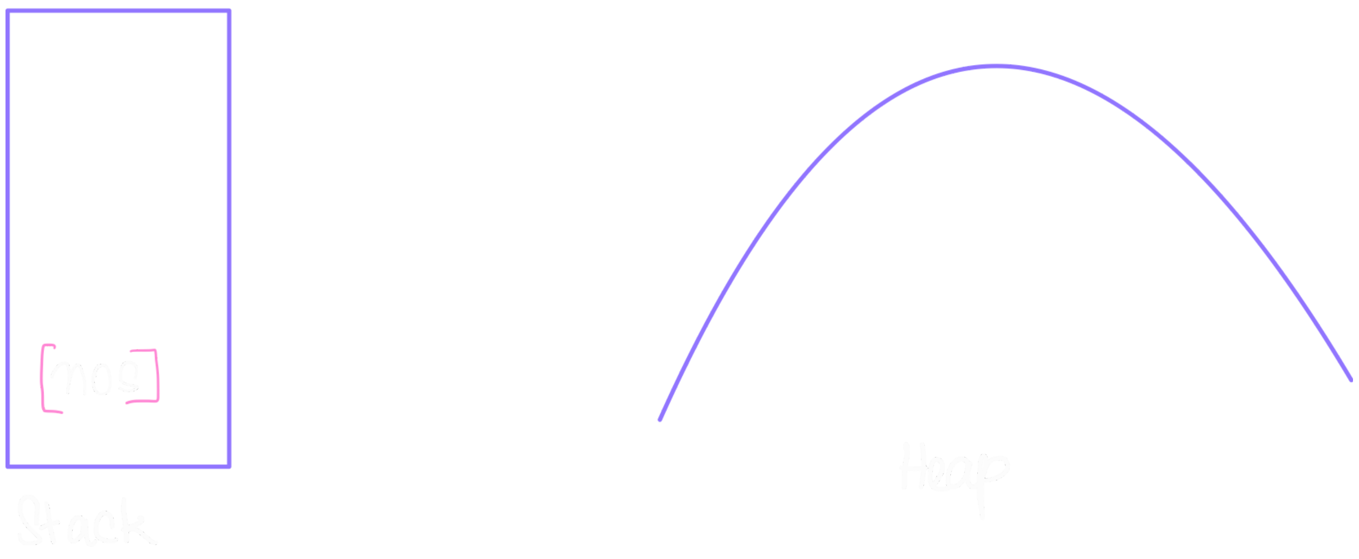
Java does not have any concept of pointers, it totally depends on JVM [Java Virtual Machine] whether arrays will have continuous or discontinuous allocation of memory.

But in Java documentation, it is said that objects in heap are not stored continuously.

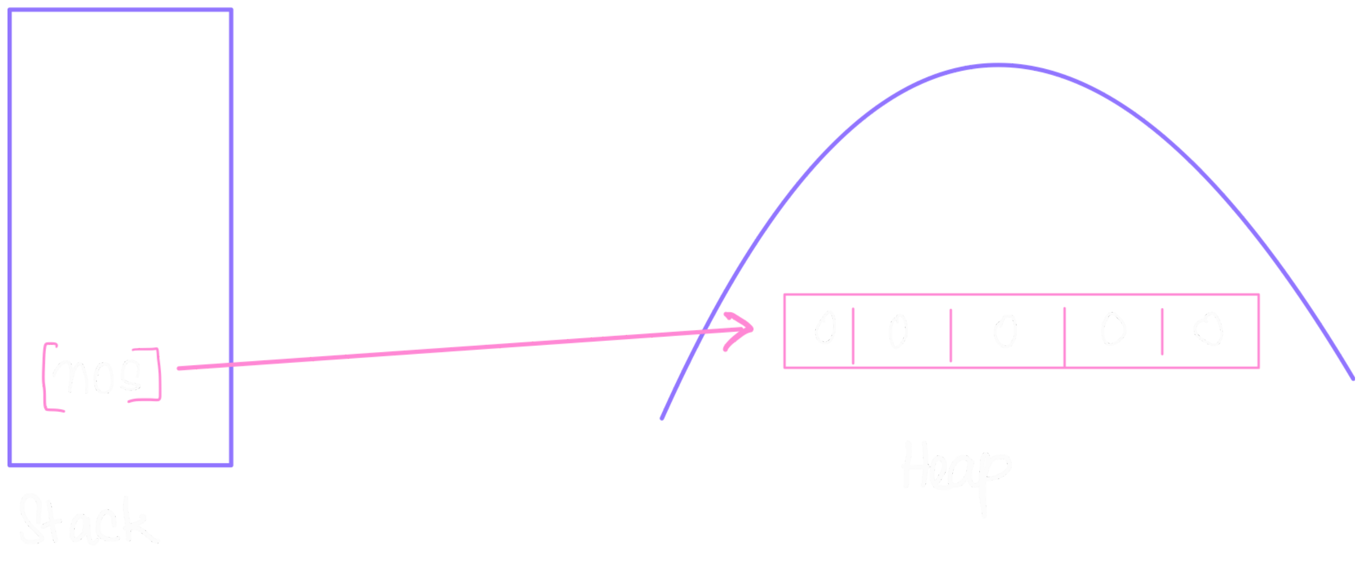
Hence, Arrays in Java may not be continuous.

```
public classArray {
           public static void main(String[] args) {
     int[] nos;
                       nos = new int[5];
                       System.out.println(Arrays.toString(nos));
  }
}

/* Output : [0, 0, 0, 0, 0] */
```

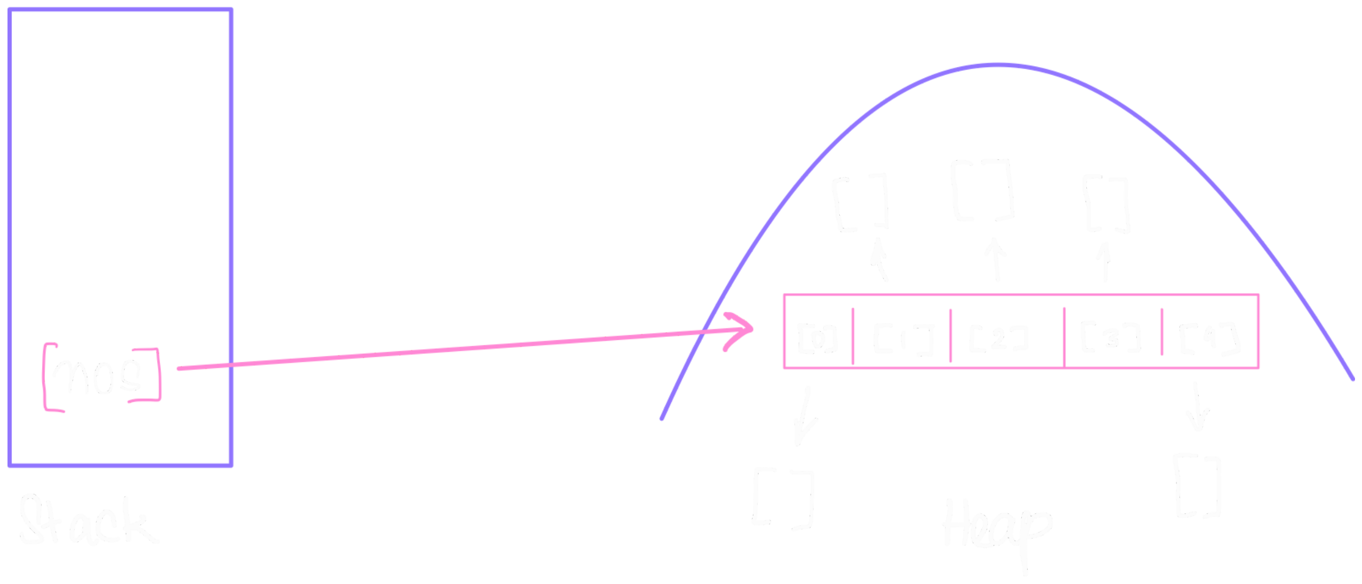
- `int[]` represents the type of data stored in the array, all elements in the array has to be the same datatype is declared.
- `int[] nos;` is the declaration of array, i.e. `nos` is defined as a reference variable in the stack memory at compile time.



- `nos = new int[5];` is the initialisation of array, i.e. object of null type is created in heap memory at runtime [dynamic memory allocation].



- Primitives are stored in stack memory and non-primitives are stored in heap memory.
- Elements in Array act as reference variables in the heap memory.



Printing Arrays

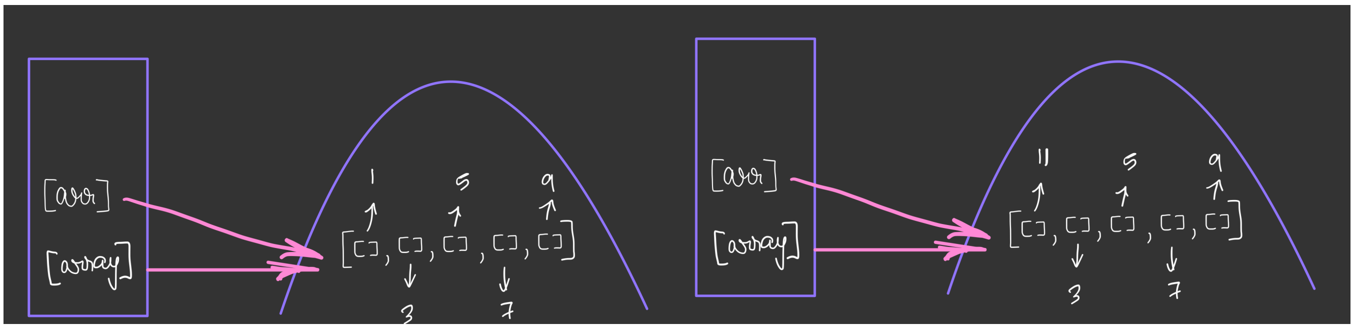
```java
package com.inclass;

import java.util.Arrays;

public class Print {
    public static void main(String[] args) {
        int[] arr = new int[5];

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

        for (int element : arr) { //for-each method
            System.out.print(element + ", ");
        }
        System.out.println();
        System.out.println(Arrays.toString(arr )); //toString method
    }
}

/* Output :
                                        0, 0, 0, 0, 0,
                                        0, 0, 0, 0, 0,
                                        [0, 0, 0, 0, 0] */
```

How are Local Variable of Arrays stored?

```
import java.util.Arrays;

public class Array {
    public static void main(String[] args) {
        int[] arr = {1, 3, 5, 7, 9};
        func(arr);
        System.out.println(Arrays.toString(arr));
    }

    static void func(int[] array) {
        array[0] = 11;
    }
}

/* Output : [11, 3, 5, 7, 9] */
```

This does not work for non-primitive data-types.

But this is not the same for arrays. Although `array` is changed, `arr` also gets changed, but in the case of arrays new objects are not created in heap memory.



How to return Arrays ?

```
package com.inclass;

import java.util.Arrays;

public class ReturningArrays {
    public static void main(String[] args) {
        System.out.println(Arrays.toString(createArr()));
    }

    static int[] createArr() {
        return new int[] {1, 2, 3, 4, 5};
    }
}
```

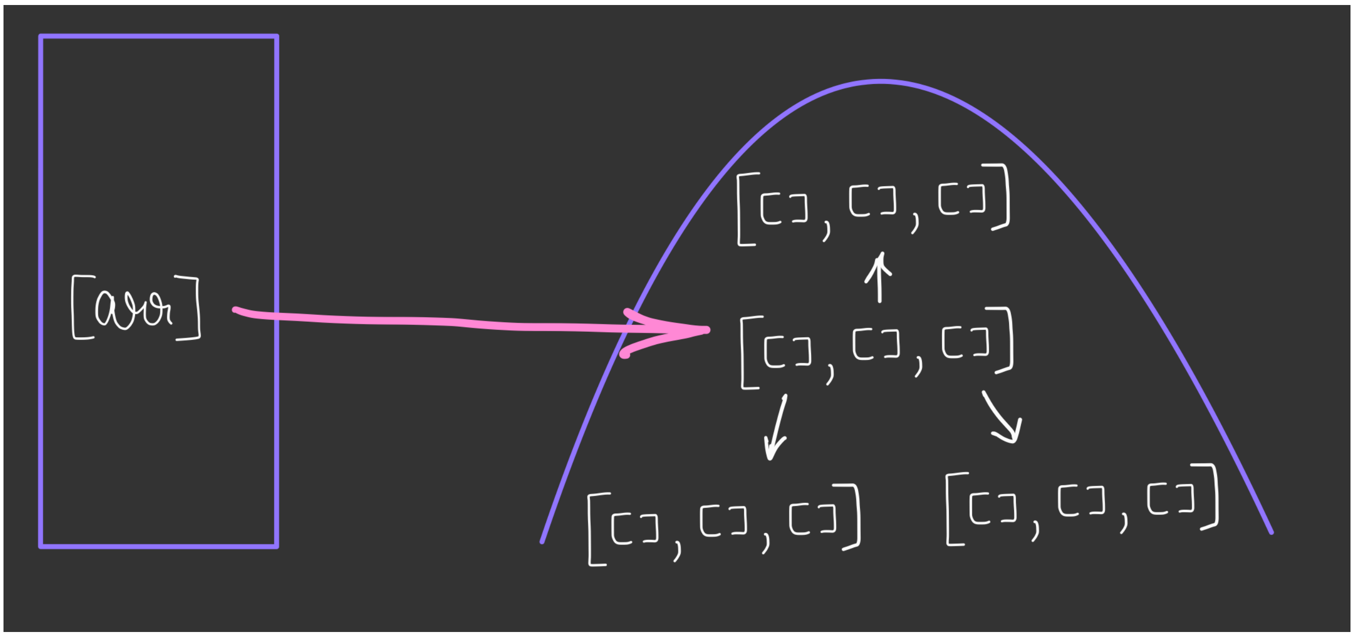
Because new objects of arrays are not created in heap memory, So `new` function is used to create a object of array in the heap.

For this we use `new int[]` to return an array.

2-D Arrays

2-D Arrays are like array of arrays.

```
package com.inclass;

import java.util.Arrays;

public class TwoDimension {
    public static void main(String[] args) {
        int[][] arr2D = new int[3][];
        print2D(arr2D);
        arr2D = new int[][] {{0, 0, 1, 0}, {0, 1, 0}, {1, 0}};
        print2D(arr2D);
    }
    static void print2D(int[][] arr2D) {
        for (int[] row : arr2D){
            System.out.println(Arrays.toString(row));
        }
        System.out.println();
    }
}

/* Output :
                                                null
                                                null
                                                null

                                                [0, 0, 1, 0]
                                                [0, 1, 0]
                                                [1, 0]
*/
```



`new int[3][]` : object of array of null elements is created in the heap. While declaration, only no of rows be stated.

How to Input for 2-D Array ?

```java
package com.inclass;

import java.util.Arrays;
import java.util.Scanner;

public class TwoDimension {
    public static void main(String[] args) {
        int[][] arr2D = new int[3][];
        for (int row = 0; row < arr2.length; row++) {
           for (int col = 0; col < arr2[row].length; col++) {
                arr2[row][col] = in.nextInt();
            }
        }
        print2D(arr2D);
    }

    static void print2D(int[][] arr2D) {
        for (int[] row : arr2D){
            System.out.println(Arrays.toString(row));
        }
        System.out.println();
    }
}
```

Array-Lists

```java
package com.inclass;

import java.util.ArrayList;

public class List {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>(5);

        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(5);
        System.out.println(list);

        list.set(0, 6);
        System.out.println(list);

        list.remove(0);
        System.out.println(list);

        System.out.println(list.get(0));
    }
}
/* Output :
[1, 2, 3, 4, 5]
[6, 2, 3, 4, 5]
[2, 3, 4, 5]
2
```

- Array size is always fixed internally.
- When array-list is filled,
    - a new array-list of maybe doubled the size of the initial size of the array-list
    - all the elements of initial array-list is copied to the new array list
    - the initial array-list is removed
- Q. Swap elements of an array

```
package com.inclass;

public class Swap {
    public static void main(String[] args) {
        int[] arr = {1, 3, 23, 9, 18};
        swap(arr, 1, 3);
    }
    static void swap(int[] arr, int index1,int index2) {
        int temp=arr[index1];
        arr[index1]=arr[index2];
        arr[index2]=temp;
    }
}
```

- Q. Reversing an array

```
package com.inclass;

import java.util.Arrays;

public class Reverse {

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

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

    static int[] reverse(int[] arr) {
        for (int i = 0; i < arr.length/2; i++) {
            int temp = arr[i];
            arr[i] = arr[arr.length-i-1];
            arr[arr.length-i-1] = temp;
        }
        return arr;
    }
}
```