Java Arrays

* Array is a collection of similar data items or elements.
* It is used to store group of data simultaneously.
* It can store data of same data type means an integer array can store only integer value, character array can store only character value and so on.
* Each location of an element in an array can be accessed by using their index.
* Every array in java has 'length' as its property which can be accessed by using ~~arry~~ -

`arrayName.length` → It gives the length of the array.

* Types of Array -

i) One-Dimensional Array -

→ Declaration of one dimensional-Array

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

Or,

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

→ Declaration and Initialisation -

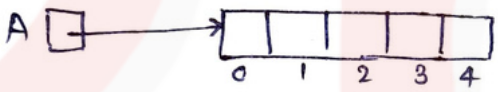
```
int A[] = {1, 2, 3, 4, 5};
```

Note:

In java array size is given after creating the new obj.

```
int A[] = new int[5]
```

Reference — obj



Here object is created in heap and reference is either in stack or heap.

* When we doesn't initialise the array, then zero is stored as default value.

Eg:-
```
Public class Test
{
    Public static void main (String a[])
    {
        int A[] = {2, 4, 6, 8, 10};
        for (int i = 0; i < A.length ; i++)
        {
            S.o.pln (A[i]);
        }
    }
```

→ Using for-each loop-

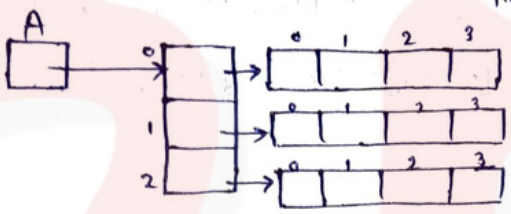
```
for (int x : A)
{
    S.o.pln (x);
}
```

o/p -
```
2
4
6
8
10
```

ii) Two-dimensional Array -

Two dimensional array are suitable for tables or matrix and it can be easily visulazed as having rows and columns.

```
int A[][] = new int [3] [4]
                    ↓       → Column
                   Rows
```



* It is also known as array of arrays or collection of arrays.

* "arrayName.length" gives number of rows.

* "arrayName[index].length" gives the no. of columns.

* Different way of declaring a 2-D array -

1) `int A[][] = new int [5] [5];`

2) `int B[][] = {{1,2,3}, {4,5,6}, {7,8,9}};` (3×3)

3) `int c [][];`

   `c = new int [3] [3];`

4) `int [][] D = new int [5][5];`

5) `int [] D[] = new int [5][5];`

6) `int[] E, F[];`

`E = new int[5]; // Here 'E' is a 1-D array`

`F = new int[5][5]; // and 'F' is 2D array.`

Note - 1) `int[] G, H, I, J;`

(In this way, we can declare multiple 1-D array at a time.)

2) `int [] G, H, I, J[];`

→ G, H, I are 1-D array.

→ J is a 2-D array.

Eg:-
```
public class Test
{
    public static void main (String a[])
    {
        int A[][] = new int [2][3];

        A[0][0] = 5
        A[0][1] = 10
        A[0][2] = 15

        A[1][0] = 20
        A[1][1] = 25
        A[1][2] = 30

    for ( int i = 0 ; i < A.length; i++)
     {
        for (int j= 0; j< A[i].length ; j++)
          {
              S.o.p ( A[i][j]+ " ");
          }
           S.o.pln ();
     }
```

Or

```
int A[][] =
{{5,10,15},{20,25,30}}
```

Using for-each loop -

```
for ( int x[] : A)
 {  for (int y : x)
     {
        s.o.p(y+ " ");
     }
     S.o.pln();
}
```

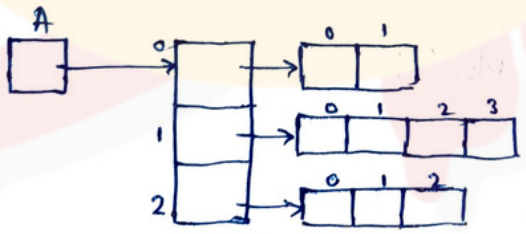
o/p - 5 10 15
20 25 30

* Jagged (Ragged Array) -

→ Jagged array is a type of array in which the members are of different sizes.

→ In jagged array, the members of an array are created separately a/c to their sizes using their indices.

Eg:-
```
int A[][];
A = new int [3][];
A[0] = new int [2];
A[1] = new int [4];
A[2] = new int [3];
```



* **Methods** -

Method is a collection of statements that are grouped together to perform an action.

* We can write our own methods in the class.
* When a method returns a value then the method itself takes the value.
* A method will have its own copy of variable.
* Whoever is called a method is called as a 'caller' or a 'method call'.
* The method which is called by a caller is known as called method.
* The parameters/argument passed in calling method are called as actual parameters.
* And the parameter of a called method are called as formal parameters.
* Formal parameters are nothing but input into a method where the return type is known as output to a method.
* When the method is called, the value of actual parameters are copied in formal parameter which is the only parameter passing method in java.

* Skeleton of method -

Signature or header ← return type methodName (parameter list)

```
{
    .....
    .....
    .....
}
```

Eg:-

```
class Test
{
    public static void main (String a[])
    {
        int a = 10, b = 15, c;
        c = max(a, b);
        S.o.pln(c);
    }

    static int max (int x, int y)
    {
        if (x > y)
            return x;
        else
            return y;
    }
}
```

Note: If a method is declared using static keyword, then no need to create an object to access it.

→ But if we create the object then there is no need to write 'static' keyword at method header or signature.

Eg-
```
class Test
{
    public static void main(String a[])
    {
        int a = 10, b = 15, c;
        Test t = new Test();
        c = t.max(a, b);
        System.out.println(c);
    }
    int max(int x, int y)    // No need to write 'static' keyword
    {
        if (x > y)
            return x;
        else
            return y;
    }
}
```

* Method Overloading :

→ Method overloading means writing more than one methods having same name but different parameter list or data type.

→ Compiler will call the corresponding method depending upon the parameter list.

Eg:-

```
Class Test
{
    public int max (int a, int b)
    {
        return a>b ? a : b ;
    }
    public int max (int a, int b, int c)
    {
        if (a>b && a>c) return a;
        else if (b>c) return b;
        return c;
    }
}
public class overloading
{
    public static void main (String ar[])
    {
        Test t = new Test ();
    int m = t.max (10, 5);
    int n = t.max (10, 15, 5);
        S.o.pln (m);
        S.o.pln (n);
    }
```