=> Functional interface =>

Anonymous inner class

parameters → Lambda expression

() -> {

implementation

};

array - string - EH - Threads - collection

=> **Array** :-

=> int a;   int a, b, c, d, e;

int b;   int a, b, c, d, e, f, g, h, i, d, ----;

:

{ int a = 10;   a [~~10~~ 20]

a = 20;

s.o.p(a); }

int a, b, c, d, - z, aa, ab, --; ✗

int az = 20; } **Array** :-

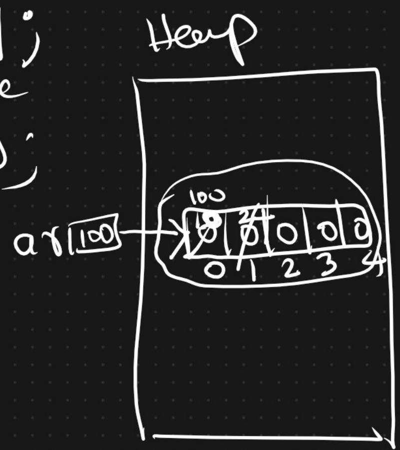
=> Array -> Treated as Object

⇓ hence memory

Heap

⟹ To store marks of 5 students :-

```
int [] ar = new int[5];
int ar[] = new int [5];   // 5 = size
```




```
ar[0] = 10;
ar[1] = 24;
⋮
s.o.p(ar[1]);   // 24
```

```
for(int i=0;i<5; i++)
    {
        System.out.println("Enter the marks of student : "+ i);
        ar[i]=sc.nextInt();
    }
```

i = 0 ~~0~~ ~~1~~ 2

i < 5

0 < 5 ✓

1 < 5 ✓

10 < 5

ar[i] =

ar[0] =

ar[1] =

ar | 10 | 14 | . | . | . |

0 1 2 3 4

ar.length

5

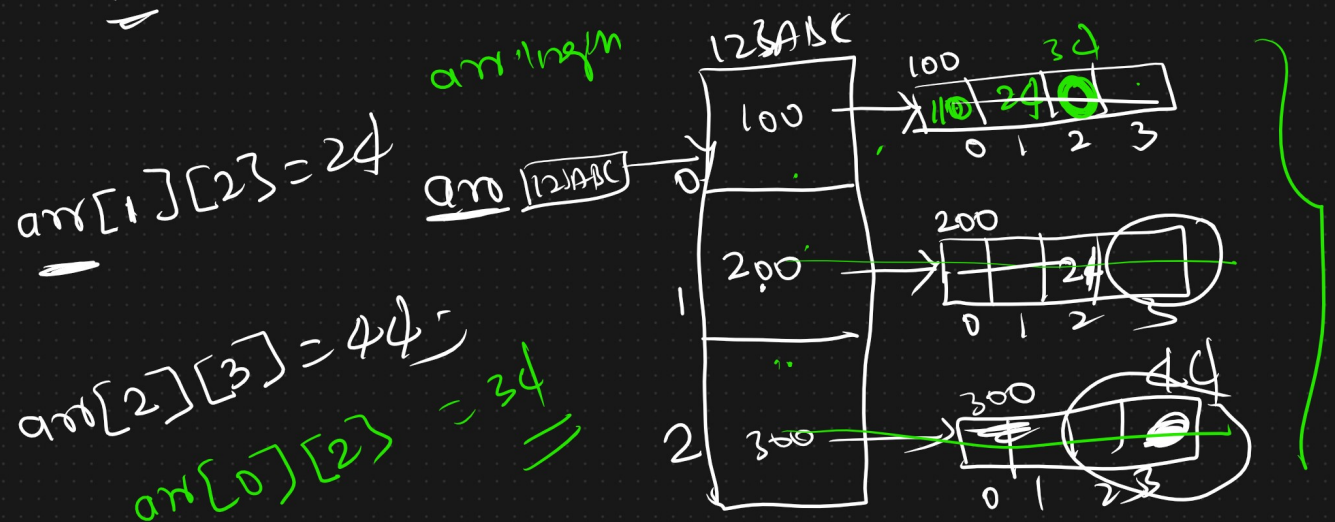
i — ar.length — j

classes	students
0	4
1	4
2	4

⟺

2D array :

int[][] ar = new int[3][4];




```
for(int i=0; i<ar.length;i++)
    {
      for(int j=0;j<ar[i].length;j++)
      {
        System.out.println("Enter the marks of class : "+ i + " Student : "+ j);
        ar[i][j]=scan.nextInt();
      }
    }
```

i = 0 1 2

i < ar.length

0 < 3

1 < 3

2 < 3

j = 0 1 2 3

j < ar[i].length

0 < ar[0].length

0 < 4

1 < 4

2 < 4

3 < 4

4 < 4 ✗

ar[i][j] =

ar[0][0]

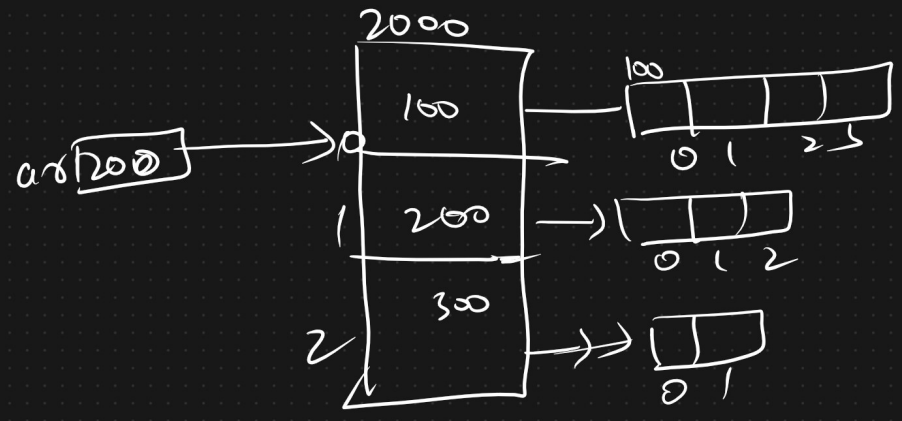
ar[0][1]

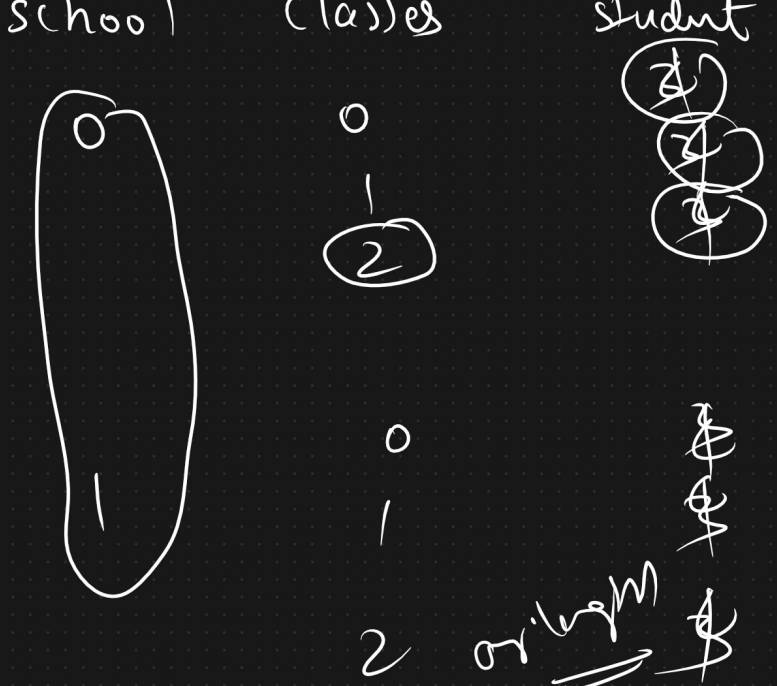
2D

classes	students
0	4
1	3
2	2

2D ⇒ Jagged array / irregular

```
int [][] ar = new int[3][];
ar[0] = new int[4];
ar[1] = new int[3];
ar[2] = new int[2];
```





new int[2][3][4];

i -> school

j -> class

k -> student

ar[i].length

ar[i][j].length.

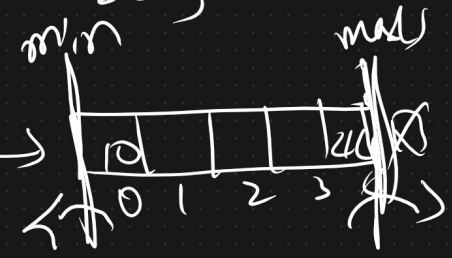
=>

C => BufferOverRun :-

(5) array

```
ar[0] = 10;
ar[1] = 24;
ar[2] = 4;
ar[4] = 45;
ar[5] = 54;
```


```
int ar[] = new int[5];
ar[0] = 10
  :
ar[4] = 44;
ar[5] = 54;
```




```
import java.util.Scanner;
```

String => lang

util —

lang

sql —

io —

net —

```
#include<stdio>
<cmath>
(main)
```

=> `int ar[] = new int[x];`

size → int → non-negative

=> ① It can store only homogenous type data
not heterogenous → only similar.

② Array size is fixed → cannot grow or shrink.

<=> ③ Array demands → contiguous memory location.

12

`new int[3];`

10	20	30
12		

RAM