MATRIX

① Multidimensional Array in Java.

* Not stored in contiguous locations.

```
Class Test
{ public static void main(String[] args)
  { int arr[][]={{1,2,3},
                 {4,5,6,7}};
```

No: of ele can be diff in diff rows.

for traversing:

```
    for(int i=0; i<arr.length; i++) {
        for(int j=0; j<arr[i].length; j++)
            { System.out.println(arr[i][j]+" ");
            }
        System.out.println();   → for tabular form printing.
    }
}
```

Note: — `arr[2][3];` ⟶ specification like this is not allowed in Java, only in C++.

⇒
```
int m=2, n=3;
int arr[][] = new int[m][n];
```

② Passing as fn.

① `func(arr);`

② `public static void func(int arr[][]) {`

`}`

(3) Matrix in Snake Pattern

I/p :→ Matrix given. ⟶ $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ } Snake pattern.

O/p :→ Matrix elements in Straight pattern → (1 2 3 6 5 4 7 8 9)

```
Void printSnake ( int mat[][] )
{
    for (int i=0; i<R; i++)
    {
        if (i%2==0)
        {
            for (int j=0; j<col; j++)
            { print (mat[i][j] + " "); }
        } else {
            for (int j=col-1; j>=0; j--)
            { print (mat[i][j] + " "); }
        }
    }
}
```

---

(4) Matrix Boundary Traversal

I/p:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

O/p: → 1 2 3 4 8 12 16 15 14 13 9 5

Idea :- Print arr[0][--]
arr[n-1][---]
arr[--][0]
arr[---][n-1]

not in java.

```
void bTraversal(int mat[R][C])
{
    if (R==1)
    {
        for(int i=0; i<C; i++)
            { print(mat[0][i] + " "); }
    }
    else if (C==1)
    {
        for (int i=0; i<R; i++)
            { print(mat[i][0] + " "); }
    }
    else
    {
        for(int i=0; i<C; i++)
            {print(mat[0][i] + " ");}
        for(int i=1; i<R; i++)
            {print(mat[i][C-1] + " "); }
        for(int i=C-2; i>=0; i--)
            {print(mat[R-1][i] + " "); }
        for(int i=R-2; i>=1; i--)
            {print(mat[i][0] + " "); }
    }
}
```

1 row & 1 column handled seperately

look at for statement carefully, as we don't want to print ele twice. C-2, R-2 used.

③ Transpose of a matrix

⇒

a) Naive

```
Void transpose ( int mat[][] )
{ int temp[n][n];
    for (int i=0 ---- )
    { for(int j=0- ------ )
        { temp=[i][j] = mat [j][i] ; }
    }
}
```

b)

⇒ ~~Eff~~

```
Void transpose (int mat [][] )
{ for(------ )
    { for(------)
        { swap (mat [i][j], mat [j][i]);
    }
}
```

---

④ Turn the matrix 90° (anticlockwise)

(Rotate)

⇒

```
1 2 3          3 6 9
4 5 6   ⟶      2 5 8
7 8 9          1 4 7
```

Naive

```
void rotate90(int mat[][])
{ int temp[n][n];
    for(int i=0....)
    { for(int j=0.....)
      { temp[n-j-1][i] = mat[i][j]; }
    }
  // temp is req matrix.
}
```

Eff sol

O(n2) & O(1) space.

→ Find transpose and reverse columns.

→ Code

```
void rotate90(int mat[][])
{
  for(int i=0; i<n; i++)
   { for(int j=i+1; j<n; j++)
     { swap(mat[i][j], mat[j][i]) }
   }

  for(int i=0; i<n; i++){
     int low=0, high=n-1;
     while(low<high)
     { swap(mat[low][i], mat[high][i]);
       low++;
     } high--;
} }
```

(5) Spiral Traversal of Matrix

I/p :- 1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

O/p :- 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

→ 4 pointers used. (top, bottom, right, left).

→ 4 loops used.
