

Two-Dimensional Arrays

Nested Iteration Statements

In Java, nested iteration statements are iteration statements that appear in the body of another iteration statement. When a loop is nested inside another loop, the inner loop must complete all its iterations before the outer loop can continue.

```
for(int outer = 0; outer < 3; outer++){
    System.out.println("The outer index
is: " + outer);
    for(int inner = 0; inner < 4; inner++)
    {
        System.out.println("\tThe inner
index is: " + inner);
    }
}
```

Declaring 2D Arrays

In Java, 2D arrays are stored as arrays of arrays. Therefore, the way 2D arrays are declared is similar 1D array objects. 2D arrays are declared by defining a data type followed by two sets of square brackets.

```
int[][] twoDIntArray;
String[][] twoDStringArray;
double[][] twoDDoubleArray;
```

Accessing 2D Array Elements

In Java, when accessing the element from a 2D array using `arr[first][second]`, the `first` index can be thought of as the desired row, and the `second` index is used for the desired column. Just like 1D arrays, 2D arrays are indexed starting at `0`.

```
//Given a 2d array called `arr` which
stores `int` values
int[][] arr = {{1,2,3},
               {4,5,6}};
```

```
//We can get the value `4` by using
int retrieved = arr[1][0];
```


Initializer Lists

In Java, initializer lists can be used to quickly give initial values to 2D arrays. This can be done in two different ways.

1. If the array has not been declared yet, a new array can be declared and initialized in the same step using curly brackets.
2. If the array has already been declared, the `new` keyword along with the data type must be used in

```
// Method one: declaring and initializing
at the same time
double[][] doubleValues = {{1.5, 2.6,
3.7}, {7.5, 6.4, 5.3}, {9.8, 8.7, 7.6},
{3.6, 5.7, 7.8}};
```

order to use an initializer list

// Method two: declaring and  initializing separately:

```
String[][] stringValues;  
stringValues = new String[][] {{"working",  
"with"}, {"2D", "arrays"}, {"is", "fun"}};
```

Modify 2D Array Elements

In Java, elements in a 2D array can be modified in a similar fashion to modifying elements in a 1D array. Setting `arr[i][j]` equal to a new value will modify the element in row `i` column `j` of the array `arr`.

```
double[][] doubleValues = {{1.5, 2.6,  
3.7}, {7.5, 6.4, 5.3}, {9.8, 8.7, 7.6},  
{3.6, 5.7, 7.8}};
```

```
doubleValues[2][2] = 100.5;
```

// This will change the value 7.6 to 100.5

Row-Major Order

“Row-major order” refers to an ordering of 2D array elements where traversal occurs across each row - from the top left corner to the bottom right. In Java, row major ordering can be implemented by having nested loops where the outer loop variable iterates through the rows and the inner loop variable iterates through the columns. Note that inside these loops, when accessing elements, the variable used in the outer loop will be used as the first index, and the inner loop variable will be used as the second index.

```
for(int i = 0; i < matrix.length; i++) {  
    for(int j = 0; j < matrix[i].length;  
j++) {  
        System.out.println(matrix[i][j]);  
    }  
}
```

Column-Major Order

“Column-major order” refers to an ordering of 2D array elements where traversal occurs down each column - from the top left corner to the bottom right. In Java, column major ordering can be implemented by having nested loops where the outer loop variable iterates through the columns and the inner loop variable iterates through the rows. Note that inside these loops, when accessing elements, the variable used in the outer loop will be used as the second index, and the inner loop variable will be used as the first index.

```
for(int i = 0; i < matrix[0].length; i++)  
{  
    for(int j = 0; j < matrix.length; j++)  
{  
        System.out.println(matrix[j][i]);  
    }  
}
```

Traversing With Enhanced For Loops

In Java, enhanced for loops can be used to traverse 2D arrays. Because enhanced for loops have no index variable, they are better used in situations where you only

```
for(String[] rowOfStrings :  
twoDStringArray) {  
    for(String s : rowOfStrings) {  
        System.out.println(s);  
    }  
}
```

care about the values of the 2D array – not the location of those values

```
}
```

```
}
```

 [Print](#)  [Share](#) ▼