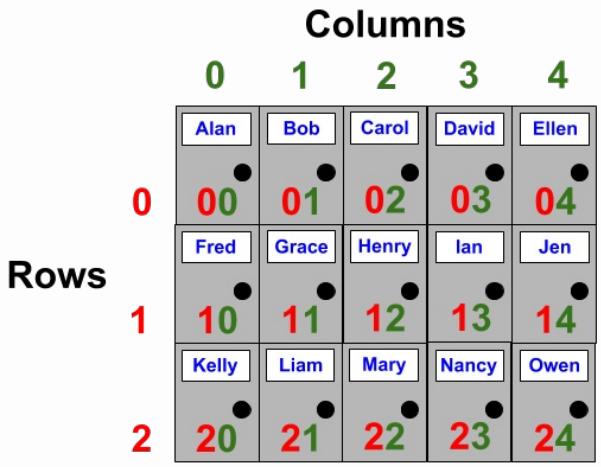
Learning Objectives: 2D Arrays

- **Create a 2D array using both the *initializer list* and *new* methods**
- **Access and modify 2D array elements**
- **Iterate through 2D arrays using both a regular `for` loop and an *enhanced* `for` loop**
- **Determine 2D array output**

Creating a 2D Array

An Array Inside Another Array

An array inside another array is called a **2D array**. A 2D arrays is symbolic of a table where there are rows and columns. The first index number represents the **row** position and the second index number represents the **column** position. Together, the row and column indices enable elements to be stored at specific locations.



.guides/img/2DArray

```
String[][] names = new String[3][5];
```

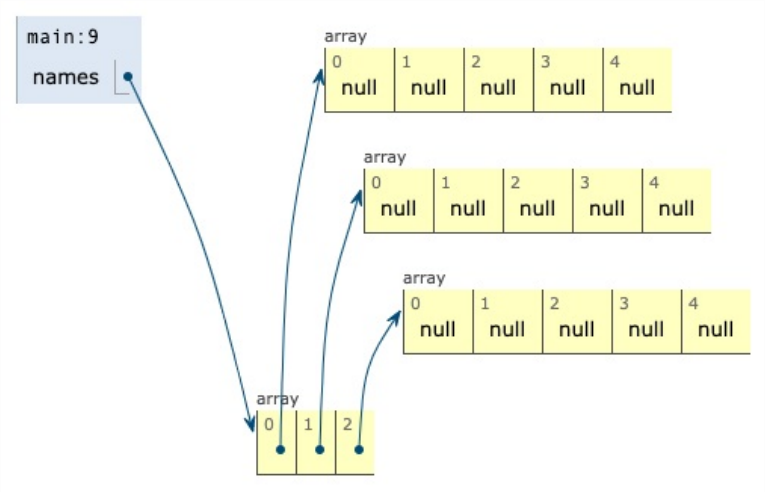
Code Visualizer

2D Array Syntax

- array type followed by **two** empty brackets `[][]` followed by a name for the 2D array.
- The = operator followed by the keyword `new` followed by the array type and two brackets `[][]`.
- The number of **rows** goes inside the **first** bracket and the number of **columns** goes inside the **second** bracket.

Why Array Inside Array?

The way 2D arrays store elements is a little unique. Rather than creating an actual table like shown above, each initial *row* of the 2D array actually refers to another *column* array. This is why a 2D array is considered to be *an array inside of another array*.



.guides/img/2DArrayReference

To determine the number of rows and columns in the 2D array, we can use the instance variable `length` like we did for arrays.

```
String[][] names = new String[3][5];

System.out.println(names.length + " rows");
System.out.println(names[0].length + " columns");
```

Code Visualizer

important

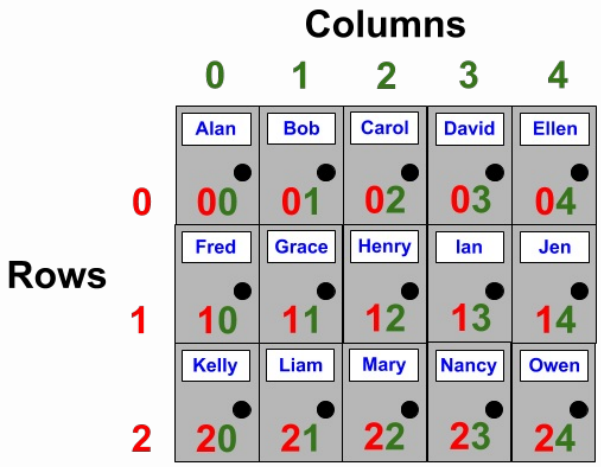
IMPORTANT

Note that when determining column length, you must refer to the 2D array's initial row index. For example, `names.length[0]` will calculate how many elements are inside row index 0 and these elements determine how many *columns* there are in the row.

Accessing and Modifying a 2D Array

2D Array Access

To access and modify elements inside a 2D array, you need to specific the *row* and *column* indices at which the elements are located. For example `names[1][2]` refers to the element that's at row index 1 and column index 2.



.guides/img/2DArray

Below is a code block showcasing a 2D array that contains fifteen P.O. Box names from a postal office. Note that you can use the **initializer list** method to populate elements inside your 2D array. Each *column* array is separated by curly braces `{}` as well as a comma `,`.

```
String[][] names = { {"Alan", "Bob", "Carol", "David", "Ellen"},
                     {"Fred", "Grace", "Henry", "Ian", "Jen"},
                     {"Kelly", "Liam", "Mary", "Nancy", "Owen"}
        };

System.out.println(names[1][2]);
```

Code Visualizer

challenge

What happens if you:

- change `names[1][2]` within the print statement from above to `names[2][1]`?
- change `names[1][2]` within the print statement from above to `names[3][0]`?

Code Visualizer

important

IMPORTANT

Note that you will still get an `ArrayIndexOutOfBoundsException` error if you attempt to access or modify an element at a row or column index that does not exist. Like arrays, you cannot add additional rows or columns of elements to the 2D array after it has been initialized.

2D Array Modification

To modify elements within a 2D array, simply access the element and assign another element to it.

```
String[][] names = { {"Alan", "Bob", "Carol", "David", "Ellen"},
                     {"Fred", "Grace", "Henry", "Ian", "Jen"},
                     {"Kelly", "Liam", "Mary", "Nancy", "Owen"}
        };

System.out.println(names[1][2]);

names[1][2] = "Harry";
System.out.println(names[1][2]);
```

Code Visualizer

challenge

What happens if you:

- change all `names[1][2]` within the code above to `names[0][0]`?
- change `"Harry"` in the code above to `"Amy"`?

Code Visualizer

Iterating a 2D Array

2D Array Iteration

To iterate through a 2D array, we can use two `for` loops, one **nested** inside another. The outer `for` loop is for the rows while the inner `for` is for the columns.

```
int[][] digits = { {1, 2, 3},
                   {4, 5, 6},
                   {7, 8, 9} };

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

Code Visualizer

challenge

What happens if you:

- change `digits[0].length` in the inner `for` loop to `digits[1].length`
- change `println` in the print statement to `print`?

Code Visualizer

Note that all of the rows' lengths are the same, they each have three elements. Therefore, it doesn't matter if we use `digits[0].length`, `digits[1].length`, or `digits[2].length`. Also note that using `println` prints the elements vertically while `print` prints the elements horizontally. To print the elements so that the columns stay together but the rows separate, we can try something like this:

```
int[][] digits = { {1, 2, 3},
                   {4, 5, 6},
                   {7, 8, 9} };

for (int i = 0; i < digits.length; i++) {
  for (int j = 0; j < digits[0].length; j++) {
    if (j == digits[0].length - 1) {
      System.out.println(digits[i][j]);
    }
    else {
      System.out.print(digits[i][j] + " ");
    }
  }
}
```

Code Visualizer

The `if` conditional forces the elements to be printed with a newline every time the iterating variable reaches the end of the column index. Otherwise, the elements will be printed with a space instead.

2D Array with Enhanced For Loop

Like arrays and ArrayLists, 2D arrays can also make use of the **enhanced** `for` loop.

```
int[][] digits = { {1, 2, 3},
                   {4, 5, 6},
                   {7, 8, 9} };

for (int[] i : digits) {
  for (int j : i) {
    if ((j == 3) | (j == 6) | (j == 9)) {
      System.out.println(j);
    }
    else {
      System.out.print(j + " ");
    }
  }
}
```

Code Visualizer

Note that we cannot use an enhanced `for` loop to manipulate array indices. Our iterating variable goes through the 2D array and takes on each element value rather than each element index. This is why we have the conditional

statement `if ((j == 3) | (j == 6) | (j == 9))` rather than `if (j == digits[0].length - 1)`. Additionally, since we have an array inside of another array, our iterating variable `i` is of type `int[]` rather than just `int`.