2D Arrays

Arrays in Arrays

Arrays can contain…

- ints
- doubles
- chars
- Strings
- **any other objects**

We've used a bunch of other objects.

- Some were built in to Java:
    - Scanner
    - StringTokenizer
    - Rectangle
- Some we've built ourselves:
    - VendingMachine
    - Theater
    - Needle

All of these types can be stored in arrays (`Rectangle[] shapes`, e.g.)

Arrays are also objects

```java
int[] citCourseNumbers = new int[3];
citCourseNumbers[0] = 591;
citCourseNumbers[1] = 592;
citCourseNumbers[2] = 593;
```

The array `citCourseNumbers` is itself an object!

- We initialized a new array using the `new` keyword
- The array has a field `length`

This means...

We Can Store Arrays Inside of Arrays.

These nested arrays are called "2D Arrays"

The syntax is similar to that of 1D Arrays:

```
type[][] arrayName = new type[numRows][numCols]
```

The above creates a 2D array that will store data with type `type` in a matrix with `numRows` rows and `numCols` columns.

Example

```
int[][] matrix = new int[3][4]
matrix[2][1] = 7;
matrix[1][3] = 12;
```

	col 0	col 1	col 2	col 3
row 0				
row 1				12
row 2		7		

Getting Types Straight

Expression	Type	Meaning
`matrix`	`int[][]`	Array of arrays, or 2D array.
`matrix[1]`	`int[]`	the second row inside of `matrix`
`matrix[1][3]`	`int`	the int at row `1`, col `3`

Iterating over 2D Arrays

The basic strategy is to iterate over rows, then within the rows iterate over columns.

```
double[][] fractions = new double[5][5];
for (int i = 0; i < fractions.length; i++) {
    for (int j = 0; j < fractions[i].length; j++) {
        fractions[i][j] = i / j;
    }
}
```

Iterating over 2D Arrays

Result:

```
[[NaN,      0.0, 0.0, 0.0,                0.0],
 [Infinity, 1.0, 0.5, 0.3333333333333333, 0.25],
 [Infinity, 2.0, 1.0, 0.6666666666666666, 0.5],
 [Infinity, 3.0, 1.5, 1.0,                0.75],
 [Infinity, 4.0, 2.0, 1.3333333333333333, 1.0]]
```

Explicit 2D Array Declaration

Same as with 1D Arrays, but with more braces.

```
String[][] seatingChart = {{"Harry", "Dana"}, {"Jintong", "Vivian", "Adrian"}};
```

or, for more clarity:

```
String[][] seatingChart = {
    {"Harry", "Dana"},
    {"Jintong", "Vivian", "Adrian"}
};
```

Jagged Arrays (did you catch that?)

2D arrays do not have to have the same number of columns in every row.

```
String[][] seatingChart = {
    {"Harry", "Dana"},
    {"Jintong", "Vivian", "Adrian"}
};
```

Row 0 is an array with a length of 2 and row 1 is an array with a length of 3.

Practice: Transposing a 2D array

For a given **rectangular** (non-jagged) 2D int array `A`, return a new 2D array `B` where `A[i][j] == B[j][i]` for all `i` and all `j`.

Solution: Transposing a 2D array

```
public int[][] transpose(int[][] A) {
    int numRows = A.length;
    int numCols = A[0].length;
    int[][] B = new int[numCols][numRows];
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            B[j][i] = A[i][j];
        }
    }
    return B;
}
```

Practice: Flattening a 2D array

For a given **rectangular** (non-jagged) 2D int array `A`, return a new 1D array `B` where `B` has all of the elements from the first row of `A`, then from the second row of `A`, then from the third row of `A`, etc.

Worked Example: Tic Tac Toe

- CRC
- Building the Game