Arrays, Part 3 of 3

Christopher Simpkins
chris.simpkins@gatech.edu

Multidimensional Arrays

You can create arrays of any number of dimensions simply by adding additional square-bracketed sizes. For example:

```
char[][] grid;
```

The declaration statement above:

- Declares a 2-dimensional array of `char`.
- As with one-dimensinal arrays, `char` is the base type.
- Each element of `grid`, which is indexed by two `int` expressions, is a `char` variable.

Initializing Multidimensional Arrays

Initialization of 2-dimensional arrays can be done with `new`:

```
grid = new char[10][10];
```

or with literal initialization syntax:

```
char[][] grid = {{' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
                 {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
                 {' ', '*', '*', ' ', ' ', ' ', ' ', '*', '*', ' '},
                 {' ', '*', '*', ' ', ' ', ' ', ' ', '*', '*', ' '},
                 {' ', ' ', ' ', ' ', '*', '*', ' ', ' ', ' ', ' '},
                 {' ', ' ', ' ', ' ', '*', '*', ' ', ' ', ' ', ' '},
                 {' ', '*', ' ', ' ', ' ', ' ', ' ', ' ', '*', ' '},
                 {' ', ' ', '*', ' ', ' ', ' ', ' ', '*', ' ', ' '},
                 {' ', ' ', ' ', '*', '*', '*', '*', ' ', ' ', ' '},
                 {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '}};
```

Notice that a 2-dimensional array is an array of 1-dimensional arrays (and a 3-dimensional array is an array of 2-dimensional arrays, and so on).

Visualizing Multidimensional Arrays

Our 2-dimensional `grid` array can be visualized as a 2-d grid of cells.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
grid[0]	' '	' '	' '	' '	' '	' '	' '	' '	' '	' '
grid[1]	' '	' '	' '	' '	' '	' '	' '	' '	' '	' '
grid[2]	' '	'*'	'*'	' '	' '	' '	' '	'*'	'*'	' '
grid[3]	' '	'*'	'*'	' '	' '	' '	' '	'*'	'*'	' '
grid[4]	' '	' '	' '	' '	'*'	'*'	' '	' '	' '	' '
grid[5]	' '	' '	' '	' '	'*'	'*'	' '	' '	' '	' '
grid[6]	' '	'*'	' '	' '	' '	' '	' '	' '	'*'	' '
grid[7]	' '	' '	'*'	' '	' '	' '	' '	'*'	' '	' '
grid[8]	' '	' '	' '	'*'	'*'	'*'	'*'	' '	' '	' '
grid[9]	' '	' '	' '	' '	' '	' '	' '	' '	' '	' '

And an individual cell can be accessed by supplying two indices:

```
grid[3][2] == '*'; // true
```

Traversing Multidimensional Arrays

Traverse 2-dimensional array by nesting loops. The key to getting it right is to use the right `length`s.

```
for (int row = 0; row < grid.length; ++row) {
    for (int col = 0; col < grid[row].length; ++col) {
        System.out.print(grid[row][col]);
    }
    System.out.println();
}
```

Note that the for loops above traverse the grid in row-major order. We can traverse the grid in column-major order by reversing the nesting of the for-loops:

```
for (int col = 0; col < grid[0].length; ++col) {
    for (int row = 0; row < grid.length; ++row) {
        System.out.print(grid[row][col]);
    }
    System.out.println();
}
```

Ragged Arrays

It's possible to create *ragged arrays* by creating nested arrays of variable length. For example:

```
double [][] ragged = new double[3][];
ragged[0] = new double[5];
ragged[1] = new double[10];
ragged[2] = new double[4];
```

Can we traverse array `ragged` in row-major order?
Can we traverse array `ragged` in column-major order?

Programming Exercise

- Download array-data.csv.
- Let's write a program to read the data from array-data.csv into an array.