Lab 07: Arrays

Due date: By the end of your lab session of the week of Monday, October 3rd

Goals

- Learn to use 1 dimensional and 2 dimensional **arrays**
- Learn to use **for loops** to control the flow of the programs and to transverse arrays

Description

In mathematics, a matrix is a rectangular array of numbers, symbols, or expressions, arranged in rows and columns. In Java, a matrix is a two-dimensional array of elements of the same type. The dimensions of a matrix are given as *m*×*n*, where *m* is the number of rows and *n* is the number of columns in the matrix. To access an element in a matrix, you must specify a row and column **index**. The row index is given first, followed by the column index; remind that indexing starts at 0.

For example:

```
int[][] matrix = {{1, 2, 3},{4, 5, 6},{7, 8, 9}};
// matrix looks like:
// 1 2 3
// 4 5 6
// 7 8 9
int n = matrix[1][2]; //gets the value 6
```

In this lab you will use some properties of matrices. In case you are unfamiliar with them, here is a quick overview of the terms you should know.

- ***m*×*n* matrix:** a matrix with *m* rows and *n* columns

- ***n*×*n* matrix:** a matrix with *n* rows and *n* columns (square matrix)

- **(*i*,*j*) entry:** refers to an element in a matrix at row *i* and column *j*

- **main diagonal:** for an *m×n* matrix, these are the entries in the matrix at entries (i, j) where *i=j*

$$maindiagonal = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **upper diagonal:** the first adjacent diagonal above the main diagonal

$$upperdiagonal = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- **lower diagonal:** the first adjacent diagonal below the main diagonal

$$lowerdiagonal = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

In this lab you will create a class `Matrix` in a `lab07` module and implement a series of methods to validate properties of matrixes.

- `public boolean isSymmetric(int[][] matrix)`

Determines if a matrix is symmetric with respect to its main diagonal. To be symmetric, the matrix must be a square *n×n* matrix and the (*i,j*) entry must be equal to the (*j,i*) entry.

- `public boolean isDiagonal(int[][] matrix)`

Determines if a matrix is a diagonal matrix. To be a diagonal matrix, all entries that do not lie on the main diagonal must equal 0.

- `public boolean isIdentity(int[][] matrix)`

Determines if a matrix is an identity matrix. To be an identity matrix, the matrix must be a square *n*×*n* matrix, all entries along the main diagonal must equal 1, and all other entries must equal 0.

- `public boolean isUpperTriangular(int[][] matrix)`

Determines if a matrix is an upper triangular matrix. To be an upper triangular matrix, the matrix must be a square *n*×*n* matrix and all entries below the main diagonal must equal 0.

- `public boolean isTridiagonal(int[][] matrix)`

Determines if a matrix is a tridiagonal matrix. To be a tridiagonal matrix, the matrix must be a square *n*×*n* matrix, all entries must equal 0 **except** the main diagonal, the upper diagonal and the lower diagonal.

Turning in Your Work

You **must** turn in your "lab07" directory before leaving the lab session. Change your current directory to "cs180" and execute the following command:

```
turnin -c cs180=COMMON -p lab07 lab07
```

Rubric

This lab is worth 50 points. The points breakdown is as follows:

- 8 pts: Correctly implements `isSymmetric` method
- 8 pts: Correctly implements `isDiagonal` method
- 8 pts: Correctly implements `isIdentity` method
- 13 pts: Correctly implements `isUpperTriangular` method
- 13 pts: Correctly implements `isTridiagonal` method