Benjamin Leibowitz
Lab 2

<u>Description of data structures</u>
In this lab, two primary data structures were used. To parse an individual row from the text file, the row's string was converted to a 1-D string array, separated by single spaces. The string array was then converted to a 2-D integer array for downstream determinant calculation.

<u>Justification of your data structure choices and implementation</u>
Due to the consistency of input matrices (integers separated by a single space), this made conversion of a row within the input text file to a valid string array a single line of code. Given more time and more difficult test cases, I would likely test for separators, and allow for different delineations (tab / comma / space). However, by placing a user requirement for spaces between integers within each matrix, 1-D string array conversion proves to be quite simple.

Next - to convert the string to a 2-D integer array, the length of the string array was used to initialize the 2-D array (in both dimensions due to square matrix). Then, subsequent lines can be used to input each element within the text file into an integer array by reading in new lines from the input text file and simple casting and a nested for loop over rows and columns.

The rationale for using arrays was simplicity. Looping and assigning elements within arrays proved to be the simplest route, so that's what I used. Additionally, useful array attributes (length) are already defined and easily accessed.

<u>Discussion of the appropriateness to the application</u>
The array data structures used seemed to be the simplest solution to the problem at hand. Given that all the values in the matrix were integers (arrays require a singular data type) separated by a space (separable with the split() function) with no empty lines between input lines (allowing for simple looping), an array seemed suitable. Also - it was mandatory to use the array structure =).

<u>Description and justification of your design decisions</u>
Several design decisions were made during this lab. First, the data structures used, for which decision rationale is explained above. Next, whether to have a string-parsing script be separate from the class with the algorithm for calculating determinants. Though splitting it made sense at first for modularity purposes, a single java script that combined both the string reading and the determinant calculation seemed to be most readable and easy to compile.

Overall, recursion is a difficult process to implement, and simplicity is important. Thus most decisions that were made were done to speed up the debugging process, which was the vast majority (95%) of the time spent on this lab.

<u>Efficiency with respect to both time and space</u>
Due to the small size of input matrices, space was not a big concern. With regard to time, recursion is likely far slower than an iterative process for calculating the determinant of a matrix, due to the multiple calls of the same function, testing of base cases, and back calculations that

need to be performed once the base cases are reached. However, the point of this lab was to implement recursion, and for that reason, it was used instead of iteration, which would have been much more efficient computationally.

What you learned

Many lessons were learned in this lab, including the fact that recursion is extremely difficult to implement due to the infinite calling that can happen. Many times I was left hanging after running the program, hoping that one day it would finish running. It never did.

The two processes that helped me debug were creating simplicity, thoughtful base cases, printing intermediate statuses of arrays. In particular, the Arrays.deepToString() function was probably the most helpful, which can print 2-D arrays quite easily.

Also, I learned that the more time you spend thinking about the overall algorithm and architecture of code before writing it, the less time you have to spend debugging. Unfortunately, I was rather careless at the beginning of this lab, and it meant I had to spend a great deal of time fixing errors, and incorrect determinant calculations.

What you might do differently next time

Next time, I would put in much more thought into the base cases and algorithm prior to writing it out. This would have resulted in a lot less time debugging. The approach I took this time was to get all my thoughts down, then go back and correct mistakes. Wrong approach for sure.

Discussion of anything you did as an enhancement

As enhancements, I test whether the file is valid, and print the string back to the user. If I were a user, there is no chance I'd remember which order I put in matrices, so having them returned to you in addition to getting the output desired (the determinant), would be helpful in my opinion.