Assignment 8

Due: November 19, 2014 at 11:55PM.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own. Any sharing or copying of assignments will be considered cheating. If you get significant help from anyone, you should acknowledge it in your submission.

You should not use any features of Java that we did not cover in class.

Problem 1 (60 points): Matrix

Matrix Class (40 points)

For this program you should revisit the code that you wrote for assignment 6. In this assignment you will encapsulate the data (i.e. two dimensional array) that represents the matrix and the operations that can be performed on it into a single class called Matrix.

Your class should be able to handle matrices for which the number of rows and number of columns are between 1 and 5 (inclusive). If the client of the class attempts to create a matrix with fewer rows/columns, the number should default to 1 (do not print any error message). If the client of the class attempts to create a matrix with more rows/columns, the number should default to 5 (do not print any error message). The matrix should be filled by randomly generated integers in the range from -10 to 10 (inclusive). HINT: this verification and assignments should be performed by the class constructor.

The class should provide the methods that provide information about number of rows and number of columns of the matrix called getNumOfRows() and getNumOfColumns() .

The class should provide methods for horizontal flip, vertical flip and transpose operations (see Assignment 6), but with one difference - the new modified matrix should replace the matrix on which the operation is performed. HINT: in case of the transpose operation, this may involve "modifying" the dimensions of the matrix.

The class should provide a toString() method that returns a String object containing the representation of the matrix. The string should be multi-lined and contain the data organized into aligned columns.

The class should provide an add method. The method should take another matrix object as a parameter and modify the current object by adding to it the parameter object. Two matrices can be added only if they have the same shape (same number of rows and same number of columns). The method should return true, if the addition can be carried out and false, if it cannot. Two matrices that have the same shape are added by adding their corresponding entries together. For example: if matrix m1 is

\[
m1 = \begin{bmatrix} 1 & 3 & -5 \\ 0 & -4 & 2 \end{bmatrix}
\]

and matrix m2 is

\[
m2 = \begin{bmatrix} 3 & -4 & 5 \\ 1 & -2 & 0 \end{bmatrix}
\],

then a call to m1.add(m2) should return true and change m1 to be

\[
m1 = \begin{bmatrix} 4 & -1 & 0 \\ 1 & -6 & 2 \end{bmatrix}
\].

If then we try to add matrix m3

\[
m3 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
\]
to \( m_1 \) by calling \( m_1.add(m_3) \), the method should return \texttt{false} and \( m_1 \) should not be changed.

The class should provide a multiply method. The method should take an integer as its parameter and modify the current object by multiplying each matrix entry by the value of the parameter.

**Driver/Test Class (20 points)**

Write a program containing the main() method that uses and tests your implementation of the Matrix class. To provide thorough testing, your main method should attempt ALL of the following tasks:

- Create several new matrices: one with the valid number of rows and columns (between 1 and 5, inclusive), one with the number of rows that is too small and one for which the number of rows is too large (and then the same thing for number of columns). Use getNumOfRows and getNumOfColumns to verify if the dimensions of all of the matrices were set according to the specification above.
- Create a matrix and apply to it the series of transformations: horizontal flip, vertical flip and transpose. Verify visually if the transformations are applied correctly.
- Create matrices of matching sizes and add them together. Print both matrices and their sum to verify if the operations are correct.
- Create matrices of different shapes. Try to add them together and verify that the method returns false and that the neither of the matrices was modified.
- Create a matrix and multiply it by different factors (positive, negative, zero). Verify visually that the operations were performed correctly.

The program should not require any user interaction.

Call your files: Matrix.java and TestMatrix.java.

**Extra credit** (10 points):

---

**Problem 2 (40 points): Expression Validation**

Write a program that uses the StackOfCharacters class that we discussed in class to validate mathematical expressions. Your program should read in the mathematical expression as its command line argument. It should determine if that mathematical expression is valid (based on the matching parenthesis) and print VALID or INVALID to the screen, accordingly.

The algorithm for validation is as follows:

```java
for each character in the input string (the mathematical expression)
    if the character is an opening bracket
        push it on the stack
    if the character is a closing bracket
        if the stack is empty
            return invalid
        if the top of the stack is an opening bracket
            pop it
    if the stack is empty
        return true
otherwise
    return false
```

**ASSUMPTIONS:** You can assume that the mathematical expression provided as the command line argument does not contain any spaces (only digits, mathematical operators and parenthesis are going to be used).

Call your files: BracketValidator.java.

**Extra credit** (10 points): Modify your program so that it works with mathematical expressions that allow parenthesis, square brackets and curly braces. You will need to modify the algorithm itself.
Grading

**Does the program compile?** If not, you will lose all the points for that problem.

**Is the program properly documented?** (worth ~30% of each problem)

Proper documentation includes:

- Javadoc preamble with the name of the author, date of creation and brief description of the program;
- Javadoc documentation of each method;
- appropriately chosen variable names, i.e., descriptive names;
- comments inside the code describing steps needed to be taken to accomplish the goal of the program;
- appropriate formatting, indentation and use of white space to make the code readable.

Remember that the code is read by humans and it should be easy to read for people who were not involved in its development.

**Is the program well developed?** (worth ~40% of each problem) The classes should represent particular things and only those things for which they are designed (it is not a good idea to add a method to a class just because you are not sure where else to put it). Make sure you create variables and data fields of appropriate types, use control statements (conditionals and loops) that are appropriate for the task, accomplish your task in a well designed and simple way (not a convoluted algorithm that happens to produce the correct output for some unknown reason). You should also design a friendly and informative user interface. The program must implement all the methods and classes mentioned above.

**Is the program correct?** (worth ~30% of each problem), Make sure that your program produces valid results that follow the specification of the problem every time it is run. At this point you can assume a "well behaved user" who enters the type of data that you request, but the program has to verify if the values are within valid ranges. If the program is not completely correct, you get credit proportional to how well it is developed and how close you got it to the completely correct code.

What and how to submit?

You should submit **four** source code files compressed into a single **zip** file to NYU Classes. Do not submit all the files that Eclipse creates, just the source code files that have .java extension.

Questions

Post any questions you have regarding this assignment to NYU Classes Forums under the Assignments topic.