REVIEW OF JAVA PROGRAMMING: MATRIX MANIPULATION

1. OBJECTIVES

(a) To review the concepts of Java programming, the pre-requisite of CSC/CPE105.
(a) To review techniques of performing input / output from a plain text file.
(b) To review techniques for processing references and 2D-arrays.

2. LABORATORY

This lab will be conducted in the Computing Lab 1 (N4-B1-8) in SCE. This is an individual experiment. You cannot use the classes that implement the **Collection** interface (please refer to http://java.sun.com/docs/books/tutorial/collections/), except for Array class. We are going to implement these classes (like Vector, LinkedList, Stack, TreeSet, etc.) in CPE/CSC105, instead of using them directly from Java.

No make-up is allowed for students who have missed their stipulated lab classes without any acceptable excuse like having a valid leave of absence from the school or on medical leave. The students will be deemed to have failed the particular lab work. In case you have valid reasons to do makeup, you must inform the lecturer in charge. The makeup should be done within the same week, during the lab sessions attended by other groups.

3. EQUIPMENT

Hardware: The PCs running under the LINUX environment in Computing Lab 1.
Software: NetBeans, SUN JAVA compiler (**javac**) and interpreter (**java**). They can be downloaded from http://java.sun.com/javase/downloads/index.jsp. Choose "JDK 6 Update 10 with NetBeans 6.5". Your programming will only be tested by the lab markers using javac and java on the lab PCs. None other is accepted.

4. EXPERIMENT

A **matrix** is defined as a rectangular array of elements arranged by rows and columns. A matrix with *n* rows and *m* columns is said to have **row dimension** *n*, **column dimension** *m*, and **order** $n \times m$. A matrix *M* of order $n \times m$ can be written as follows:

$$M = \begin{bmatrix} a_{11} & a_{12} & \cdots\cdots & a_{1m} \\ a_{21} & a_{22} & \cdots\cdots & a_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots\cdots & a_{nm} \end{bmatrix}$$

where a_{ij} represents the element at row *i* and column *j.*

Matrices are used in engineering, scientific work and mathematics. The simplest and most often used operations on matrices are addition, subtraction, multiplication, transpose, determinants, and extracting the elements in a matrix. In this lab, we construct a simple matrix manipulation system using Java programming language. This will, in effect, involve creating a class for the representation of a matrix that is stored in computer memory. This class will consist of accessing routines that perform the operations such as addition and multiplication. (Find out the definitions of matrix addition, transpose, etc. from NTU library.)

In order to make the matrix implementation as flexible as possible, we will not place an *a priori* limit on the size of the matrix that can be represented. This will require the use of dynamic memory allocation. Consequently, the class definition for matrix is given below. You are required to use and expand this class to include various operations on matrices.

```
class Matrix {
      double[][] element;
}
```

Your program should define 3 matrix objects (variables), m1, m2 and m3. It should also display the following menu:

(1) Add two matrices
(2) Multiply two matrices
(3) Take transpose of a matrix
(4) Display a matrix
(5) Exit

When choice 1 or 2 is selected, your program should prompt the user to enter two **filenames** storing the two input matrices and perform the corresponding arithmetic operation. The two input matrices should be stored in m1 and m2 while m3 is for storing the result. Both the inputs and the result should be displayed.

When choice 3 is selected, your program should prompt the user again to enter the **filename** storing a single matrix. The input should be read into m1 and the result should be stored in m3. Both the input and the result should be displayed.

When choice 4 is selected, your program should prompt the user again for a number (i.e. 1, 2 or 3) to display content of m1, m2 or m3 correspondingly.

You can look up the definitions of "matrix transpose" and "matrix multiplication" from NTU library or simply from Google/Wikipedia. Implement the necessary methods to read a matrix from a file, perform addition, multiplication and transpose operations, and display the resultant matrix to the user. Although each of the tasks is to be implemented as a single method, it may be necessary to create additional sub-method(s) to handle portions of the method. This is especially true if the original method is too long or contains more than one functionally related group of statements. When you write a method, remember that this method is to work for all possible inputs. Not on just your test inputs. You must test for all conditions that might possibly arise; print out error messages as needed.

Three input text files **matrix1.txt**, **matrix2.txt** and **matrix3.txt** have been created in the directory **/home/staff1/csc105/lab1/** in Computing lab 1 to assist the development and testing of your program. Its content and even the filenames will be changed (while still adhering to the same data format) during grading in order to test your programs' robustness. The objective is to ensure that you do not hardcode your program to work exclusively for the given sample matrix. Your program should read input of menu choices from the keyboard. However, one can save typing by using redirection as "**java Lab1 < commands**" where **Lab1** is the name of your class file (i.e. it contains the main() method) and **commands** is a text file which records your desired input of menu choices from the keyboard. A typical **commands** file is provided for your convenience in **/home/staff1/csc105/lab1/**. Check also the sample output file (according to the menu options in **commands**) and a **readme** file in the same directory for any last minute hints or changes. The example input files and commands file can also be found in the appendices below.

Create a sub-directory called **cpe105/lab1** or **csc105/lab1** (depending on whether you are taking CPE105 or CSC105) in your home directory. Please note that the grader will only look into this sub-directory to find, compile (like "`javac Lab1.java`") and test run (like "`java Lab1 < commands`") your program. You should provide a **readme** file that gives instructions to compile and run your program. You should not have irrelevant source files in the directory.

5. ERROR HANDLING

Your programs should do appropriate error handling like checking for incompatible matrix dimensions for multiplication and incorrect menu option selection. Apart from these, you do not need to worry about other types of errors.

6. REPORT

(a) This lab is due six working days after your lab session. A working day includes Monday to Friday but excludes Saturday, Sunday and public holidays.

(b) For the report, submit hard copies of the readme file (see section 4), program listing (see the second and third points of section 6(c)) and **script** file(s) recording results of the testing of your program. (Type "man script" in Computing Lab 1 to learn more about script)

(c) The grading of your work will be based on the following criteria:

- correctness of program (whether the methods work according to specifications);
- structure, organization, presentation of JAVA codes;
- documentation of codes (how easy to understand your codes); avoid excessive comments, such as commenting every line of your code.
- comprehensiveness of the test cases.

(d) Take note of the following concerning the style and presentation of your program codes:

- All indentations (or tabs) should be **four-character spaces** wide only.
- Each line of code should not contain more than 80 characters (spaces included). No lines should wrap around to the next line in your hardcopy printout.
- There should not be any occurrences of more than one consecutive blank line in your program.
- Your hardcopy printout should be dark and clear. Faintly printed outputs are not acceptable. Separate your printout into individual sheets; remove the periphery of each sheet; staple the sheets together at the top-right (or top-left) hand corner only.
- You could get 3 marks reduction from each of the above non-ideal (non-standard) formats.

7. ACADEMIC HONESTY AND COLLABORATION

Cooperation is recommended in understanding various concepts and system features. But the actual solution of the assignments, the programming and debugging must be your individual work, except for what you specifically credit to other sources. (Your grade will be based on your own contribution.) For example, copying without attribution any part of someone else's program is plagiarism, even if you modify it and even if the source is a textbook. You can document the credit to other sources at the start of your program code listing. The University takes acts of cheating and plagiarism very seriously: first time violators may fail the coursework component of CPE105 / CSC105. Any wholly (or partly) copied (or being copied) programs will receive zero mark.

8. REFERENCES

[1] Text and reference books for CSC/CPE 105.

[2] http://java.sun.com/docs/index.html; http://java.sun.com/j2se/

Appendix A: matrix1.txt

```
<matrix>
	rows = 2
	cols = 2

1 2
2 4
</matrix>
```

Appendix B: matrix2.txt

```
<matrix>
	rows = 2
	cols = 2

1 2
2 4
</matrix>
```

Appendix C: matrix3.txt

```
<matrix>
	rows = 3
	cols = 2

1 2
2 4
1 1
</matrix>
```

Appendix D: commands

```
1	matrix1.txt	matrix2.txt
2
matrix3.txt	matrix1.txt
3	m4.txt
4	1
4	2
5
```

Appendix E: Grading scheme example

Please refer to the appendix of Lab 3.