LAB 4 - Java (Nov. 6 - 17)

Assessment: 5% of the total course mark.

Submission Deadline: You will receive a mark only for the codes that are demonstrated in front of a TA by the end of the lab. Additionally, you have to submit on Avenue a text file (with extension `txt`) containing each `java` class that you write (just change the extension `java` to `txt`). The name of the file should indicate the name of the class and your student ID number. The online submission has to be done by the end of the lab session.

Description: Write two `Java` classes: `Matrix`, which represents matrices with integer elements, and `UpperTriangularMatrix` to represent upper triangular matrices with integer elements stored efficiently. The accompanying file `Matrix.java` contains an incomplete declaration of the `Matrix` class. You need to complete the declarations of incomplete methods or constructors according to the specifications given below. Additionally, you need to write the declaration of the class `UpperTriangularMatrix`. A class to test class `Matrix` and class `UpperTriangularMatrix` are provided as samples. However, you may need to perform additional testing to ensure that your code meets all the specifications. A demo class will be provided for the demonstration in the lab.

Specifications:

- Class `Matrix` has only the following instance fields, which have to be `private`:
  - an integer to store the number of rows
  - an integer to store the number of columns.
  - a two dimensional array of integers to store the matrix elements.
- Class `Matrix` contains at least the following constructors:
  - `public Matrix(int row, int col)` - constructs a `row`-by-`col` matrix with all elements equal to 0; if `row` ≤ 0, the number of rows of the matrix is set to 3; likewise, if `col` ≤ 0 the number of columns of the matrix is set to 3.
  - `public Matrix(int[][] table)` - constructs a matrix out of the two dimensional array `table`, with the same number of rows, columns, and the same element in each position as array `table`.
- Class `Matrix` contains at least the following methods:
  1) `public int getElement(int i, int j) throws IndexOutOfBoundsException` - returns the element on row `i` and column `j` of `this` matrix; it throws an exception if any of indexes `i` and `j` is not in the required range (rows and columns indexing starts with 0); the detail message of the exception should read: ”Invalid indexes”.
  2) `public boolean setElement(int x, int i, int j)` - if `i` and `j` are valid indexes of `this` matrix, then the element on row `i` and column `j` of `this` matrix is assigned the value `x` and `true` is returned; otherwise `false` is returned and no change in the matrix is performed.

3) `public Matrix copy()` - returns a **deep** copy of `this Matrix`. Note: A **deep** copy does not share any piece of memory with the original. Thus, any change performed on the copy will not affect the original.

4) `public void addTo(Matrix m) throws ArithmeticException` - adds `Matrix m` to `this Matrix` (**note**: `this Matrix` WILL BE CHANGED) ; it throws an exception if the matrix addition is not defined (i.e, if the matrices do not have the same dimensions); the detail message of the exception should read: "Invalid operation".

5) `public Matrix subMatrix(int i, int j) throws ArithmeticException` - returns a new `Matrix` object, which represents a submatrix of `this Matrix`, formed out of rows `0` through `i` and columns `0` through `j`. The method should first check if values `i` and `j` are within the required range, and throw an exception if any of them is not. The exception detail message should read: "Submatrix not defined". **Note**: The new object should be constructed in such a way that changes in the new matrix do not affect `this Matrix`.

6) `public boolean isUpperTr()` - returns `true` if `this Matrix` is upper triangular, and `false` otherwise. A matrix is said to be upper triangular if all elements below the main diagonal are 0. Note that the main diagonal contains the elements situated at positions where the row index equals the column index. In the following examples the main diagonal contains elements `1,9,3`.

   Example of a 3-by-3 upper triangular matrix:

   ```
   1  4  1
   0  9  0
   0  0  3
   ```

   Example of a 3-by-4 upper triangular matrix:

   ```
   1  5  1  4
   0  9  6  6
   0  0  3  8
   ```

   Example of a 4-by-3 upper triangular matrix:

   ```
   1  4  2
   0  9  6
   0  0  3
   0  0  0
   ```

7) `public static Matrix sum(Matrix[] matArray) throws ArithmeticException`- returns a new matrix representing the sum of all matrices in `matArray`. The method throws an exception if the matrices do not have the same dimensions. This method MUST USE method `addTo()` to perform the addition of two matrices.

8) `public String toString()` - returns a string representing the matrix, with each row on a separate line, and the elements in a row being separated by 2 blank spaces.

- An `n`-by-`n` matrix `a` is said to be upper triangular if all elements below the main diagonal are 0. Such a matrix can be represented efficiently by using only a one dimensional array of size `n(n+1)/2`, which stores the matrix elements row by row, skipping the zeros below the diagonal, i.e., in the following order: `a(0,0)`, `a(0,1)`, `a(0,2)`, ⋯, `a(0,n-1)`, `a(1,1)`, `a(1,2)`, ⋯, `a(1,n-1)`, `a(2,2)`, `a(2,3)`, ⋯, `a(2,n-1)` ⋯ `a(n-1,n-1)`. The `Java` class `UpperTriangularMatrix` has to model square upper triangular matrices of integers, stored in efficient format as described above. Class `UpperTriangularMatrix` should have two private instance variables: an integer to represent the matrix size (i.e. the number of rows `n`), and a one dimensional array to store the matrix elements in efficient format.

- Class `UpperTriangularMatrix` contains at least the following constructors:
  - `public UpperTriangularMatrix(int n)` - if `n` ≤ 0, changes `n` to 1; initializes the `UpperTriangularMatrix` object to represent the all-zero `n`-by-`n` matrix.
  - `public UpperTriangularMatrix(Matrix upTriM) throws IllegalArgumentException` - initializes the `UpperTriangularMatrix` object to represent the upper triangular matrix `upTriM`. Note that `upTriM` is an object of the class `Matrix` that you have to write for this assignment. The method throws an exception if `upTriM` is not upper triangular. To check if the upper triangular condition is satisfied you MUST USE the method `isUpperTr()` of class `Matrix`.
- Class `UpperTriangularMatrix` contains at least the following instance methods:
  - `public int getDim()` - returns the number of rows of `this` matrix.
  - `public int getElement(int i, int j) throws IndexOutOfBoundsException` - returns the matrix element on row `i` and column `j` if `i` and `j` are valid indices of `this` matrix (indexing starts at 0); otherwise an `IndexOutOfBoundsException` is thrown, with message ”Invalid index”.
  - `public void setElement(int x, int i, int j) throws IndexOutOfBoundsException, IllegalArgumentException` - if `i` and `j` are valid indexes of the matrix, then the element on row `i` and column `j` of the matrix is assigned the value `x`; however, if indexes `i` and `j` correspond to a position in the lower part of the matrix and `x` is not 0 then an `IllegalArgumentException` has to be thrown with message ”Incorrect argument”; finally, if indexes `i` and `j` do not represent a valid position in the matrix then an `IndexOutOfBoundsException` is thrown, with message ”Invalid indexes”.
  - `public Matrix fullMatrix()` - returns a `Matrix` object corresponding to `this UpperTriangularMatrix`. Note that the `Matrix` object will store the full matrix including all the zeros from the lower part.
  - `public void print1DArray()` - prints the elements of the one dimensional array that stores the elements of the upper triangular part, separated by two spaces.
  - `public String toString()` - returns a string representing `this UpperTriangularMatrix` object. The representation should show all elements of the full matrix with each row on a separate line.
  - `public int getDet()` - returns the determinant of the matrix, which equals the product of the elements on the main diagonal.
  - `public double[] effSolve(double[] b) throws IllegalArgumentException` - This method solves the matrix equation `Ax=b`, where `A` is `this UpperTriangularMatrix`,

if the determinant of A is non-zero. Otherwise it throws an exception. The method returns array x. The method has to be efficient, which means that it has to use an efficient way to solve the equation and implement it without wasting time or memory resources, in other words, without allocating arrays (except for x) or invoking other methods. Partial marks will be awarded for correct, but less efficient solutions. Note that the method should also check if the dimension of b is appropriate and if not throw an exception.

INSTRUCTIONS: You may implement public methods in class Matrix to return the number of rows and the number of columns.