Assignment 2: Matrix

Logistics

- The assignment is meant to be done individually.
- The **deadline** for this assignment is **11:59 PM on Oct 21, 2022, PDT**.
- Academic dishonesty is unacceptable and will not be tolerated in this course.
- **Last Modified**: Oct 9, 2022

Get Started

To get started, please download the zip file from Canvas and follow the structure and submissions guidelines below.

Repository Structure

After you unzip the zip file downloaded above, you should get the following files and folders:

README.pdf

Problem description and requirements

Assignment_Project

The assignment project folder. It contains `out` and `src` folders.

`out/artifacts/Assignment_Project_jar` is where you should put your JAR file.

In `src/Matrix/Matrix.java`, the template for this assignment is provided. You should implement each method as requested in this description to complete each task.

We also provide a sample test script in `src/SampleTest.java` to let you test your implementation conveniently.

Submission Instructions

- During implementing this assignment:
  - **Please do not modify the repository structure**
  - **Please do not change the template we provide to you**

  A simple way to check two requirements above is to run the sample tests we provided to you. If every sample test runs smoothly and gets passed, then you are fine.

  Fail to obey this rule may lead to reduction of your final score!
- In this project, you don't have to create a JAR file in your submission. We will use our own script (similar to `src/SampleTest.java`) to test your class `Matrix` assignment.

- In order to run the sample test script in `src/SampleTest.java`, you will need to:
    - Create an artifact configuration for the JAR
        - From the main menu, select `File | Project Structure` and click `Artifacts`.
        - Click `+`, point to `JAR` and select `From modules with dependencies`.
        - To the right of the `Main Class` field, click and select `SampleTest`
        - Apply the changes and close the dialog.
    - Build the JAR artifact
        - From the main menu, select `Build | Build Artifacts`.
        - Point to `SampleTest:jar` and select `Build`.
        - If you now look at the `out/artifacts` folder, you'll find your JAR there.
- After you complete this assignment, just zip up this project folder into `Assignment_Project.zip`.
- Finally, upload it to Canvas under Assignment 2 submission link.
- Sample tests are only for clearer explanation and simple testing during your implementation. Another set of test cases will be used to evaluate your assignment after due. But don't worry. All the test cases will strictly follow the description we gave below.

General Description

Matrix is a very important form to represent numerical data. In this assignment, you are required to implement a class called `Matrix` with basic functionalities. In detail, you need implement:

- input/output
- Basic arithmetic operations, such as
    - Addition
    - Subtraction
    - Matrix/Scalar Multiplication
    - Scalar Division
    - Equalization
    - Transposition
- Matrix indexing

Notice:

- Through the whole assignment 2, elements in matrices must be presented and stored in **`double`** data type.
- Please note that **all the arguments are valid**. In the other word, it is not necessary for you to pre-check the values and raise exceptions.
- Third-party libraries and/or JDKs with built-in matrix data types operations are **forbidden**.
- In order to avoid possible abuse of third-party libraries and/or JDKs with built-in matrix data types operations, we will use OpenJDK-19 for evaluation. It's recommended that you use the same JDK in your implementation process.

Task 1: Matrix Input and Output

In this task, you need to implement basic inputs and outputs for the self-defined class `Matrix`.

Task 1.1: Load the Matrix in

In order to do matrix operation, you first need to load it in. Therefore, in this task, you need to implement the Constructor of class `Matrix` in order to load the matrix in, and store it inside the class.

Normally, we represent an $m \times n$ matrix with a sequence number in the size of $m \times n$. However, under certain circumstances there are many duplicated values in matrices. If we use normal way of representation, many numbers inside the $m \times n$ sequence will be redundant, which is a waste.

In order to solve this waste, a better way is proposed: **sparse representation of Matrix**.

Sparse representation of Matrix is given in the following format below using EBNF form.

```
ValueKey = {blank}, "[", {blank}, Value, {blank}, ";", {blank}, Keys, {blank}, "]", {blank} | {blank}, "[", {blank}, "]", {blank};
Keys = {blank}, "[", {blank}, "]", {blank} | {blank}, "[", Key, {blank}, ";", Keys, {blank}, "]", {blank};
Key = {blank}, "(", {blank}, Value, {blank}, ",", {blank}, Value, {blank}, {blank}, ")", {blank};
Value = double;
blank = " " | "\n" | "\r" | "\t";
```

Above is strict definition for hyper-humans to understand without ambiguity.

In human-readable language, the expression is in the format of:

```
[[value1; [key1; key2; ...]; [value2; [key1; key2; ...]]; ...]
```

where for each `key` (i.e. `key1`, `key2` and so on in the expression above), it is in the format:

```
key = (row, col)
```

where `row` and `col` represents row and column number of the value, which is a.

For each `[valuei; [key1; key2; ...]]`, every indicated element is assigned with the value `valuei`. `valuei` is always a **double** number.

In the other word, suppose `[valuei; [key1; key2; ...]` is part of sparse expression for matrix `A`.

```
A[key1.row][key1.col] = A[key2.row][key2.col] = ... = valuei
```

Keys in the whole expression are all distinguished, i.e., there will be **no overwriting** for all the elements with values assigned in the expression.

After all sparse value assignment is done and there are some elements are assigned with values, they will be assigned with 0.0 by default.

Notice that redundant blanks may appear anywhere inside an expression.

If you are still confused about definition of sparse representation and its expression, please read the appendix, take a view of given test cases and take discussion and lab sessions of Week 3 for more details.

Input and Output

Input

- `m`: row number of the matrix. It should be a positive integer.
- `n`: column number of the matrix. It should be a positive integer.
- `expr`: sparse representation expression of the matrix. It should be `String`.

Output

There will be no output for this task.

Task 1.2: Print the Matrix out

In this task, you should implement the method `print()` in order to make it possible to print the matrix out.

The printing format follows the format defined below using EBNF form.

```
Matrix = "[" Rows "]" | "[" "]" ;
Rows = Row | Row ";" Rows ;
Row = element | element "," Row ;
element = double;
```

To make it simple, you should output the matrix in the format of:

```
[Matrix[0][0],Matrix[0][1],...,Matrix[0][n-1];Matrix[1][0],...,Matrix[1][n-1];...;Matrix[m01][0],...,Matrix[n-1][n-1]]
```

Where `Matrix[i][j]` means the value of the element on *i*-th row, *j*-th column.

Notice that redundant blanks **may not** appear anywhere inside a valid output expression.

Input and Output

Input

There will be no input

Output

You should output a string following the format defined above.

- Notice that **all the numbers must keep 5 digits after decimal**
  - hint: use `String.format`.

Task 2: Matrix Arithmetic Operations

In this task, you need to implement basic operations for the self-defined class `Matrix`.

Task 2.1: Matrix Addition and Subtraction

Implement methods `add()` and `sub()` for class `Matrix`.

Input and Output

Input

`other`: class `Matrix`

Output

`res`: class `Matrix`

Task 2.2: Matrix Multiplication

Implement method `mul()` for class `Matrix`.

There are 2 circumstances for matrix multiplication:

- Multiplication between 2 matrices
- Multiplication between one matrix and a scalar (which is `double`)

Input and Output

Input

`other`: class `Matrix` or scalar (`double`)

Output

`res`: class `Matrix`

Task 2.3: Matrix Scalar Division

Implement method `div()` for class `Matrix`.

Input and Output

Input

`other`: scalar (`double`)

Output

`res`: class `Matrix`

Task 2.4: Matrix Equalization

Implement method `isEq()` for class `Matrix` to judge whether the given matrix (`other`) is identical with each other.

Input and Output

Input

`other`: class `Matrix`

Output

`isEqual`: `boolean`

Task 2.5: Matrix Transpose

Implement method `transpose` for class `Matrix` to calculate the transposed matrix.

Input and Output

Input

No input for this task

Output

`res`: class `Matrix`

Task 3: Matrix Indexing

In this task, you need to implement method `getElements()` and `setElements()` so that part of the class `Matrix` could get accessed or modified.

The following are tasks you need to do (`x` is an instance of `Matrix` class):

- `x.getElements(row_num)` returns the `row_num`-th row (starting at `0`) which is also a matrix
  - **Input**:
    - `row_num`: integer
  - **Output**:
    - `res`: class `Matrix`
- `x.getElements(row_num, col_num)` returns the element at the `row_num`-th row and `col_num`-th column
  - Please return the result in `String` and **keep it 5 digits after decimal**.
  - **Input**:
    - `row_num`: integer
    - `col_num`: integer
  - **Output**:
    - `res`: String
- `x.setElements(row_num, other)` replaces the `row_num`-th row by the Matrix `other`
  - `other` only has one row
  - **Input**:
    - `row_num`: integer
    - `other`: class `Matrix`
  - **Output**:
    - There will be no output for this task.
- `x.setElements(row_num, col_num, other)` replaces the element at the `row_num`-th row and `col_num`-th column by the `double` float `other`
  - **Input**:
    - `row_num`: integer
    - `col_num`: integer
    - `other`: `double`
  - **Output**:
    - There will be no output for this task.

Please note that **all the arguments are valid**. In the other word, it is not necessary for you to pre-check the values and raise exceptions.