Project2: Matrix Multiplication Algorithms

PART 1 (*15% of the credit for Project2*): General algorithm for matrix multiplication

Submission type: in-person demo during **lab** or **office hours** (***both partners must be present***).
Deadline: 5 p.m. on Thursday, 4/15/21

Define a class named *MatrixWork* containing two *static* methods: *matrixProduct* and *main*

- *matrixProduct*: The goal of this method is to multiply given A and B matrices of $n\times k$ and $k\times m$ sizes respectively, and return the result-matrix C of size $n\times m$ (i.e. return C where C = A·B). A matrix is given in a 2-dimensional array. Thus, the method-signature is as follows:

public static int[][] matrixProduct (int [][] A, int[][] B)

Note: you can assume that A and B are valid arrays with > 0 number of rows and columns.

Attention: 1. The 2-dimensional array C has as many rows as A, and as many columns as B.
2. Elements of C are computed based on the following formula (each $c_{i,j}$ element is individually calculated as a sum):

$$c_{i,j} = \sum_{l=1}^{k} a_{i,l} \cdot b_{l,j}$$

Note: indexing in this formula is ***natural***.

Requirement: to multiply A and B matrices, the **number of columns in A must be equal to the number of rows in B**. You must first do this validity check and throw an ***IllegalArgumentException*** type exception if the requirement is not met.

Reminder: the number of rows in a two-dimensional array *arr* is *arr.lenght* and the number of columns is *arr[0].length* (assuming all rows have the same number of elements, which is the case when representing a matrix).

- *main*: In this method you will do the following (*no need to do any validity checks in main*):

1. Prompt the user to enter the input-file's name, then input the file-name and set up a scanner.
2. Create two 2-dimensional arrays for matrices A and B, and fill A and B with numbers, inputting values from the input-file as described below.

In the input-file you will have integer numbers only (there will not be any other symbols or signs). All numbers will be separated by whitespaces. The first two numbers will indicate the size of the A matrix (i.e. the number of rows and columns respectively), then values of the A matrix will follow (row after row, listing elements from left to right). After that, the next two numbers will indicate the size of the matrix B (i.e. the number of rows and columns respectively), then values of the B matrix will follow (row after row, listing elements from left to right). *You can assume that the 4 numbers representing A and B matrix-sizes will be positive numbers* (no need for validity check).

An example file-content is given below. Note that the data are shown in a clear and visually convenient look; however, there is no formatting requirement for the file, i.e. you should **not** assume/expect any mandatory line-breaks or empty lines.

```
2 3
1 2 3
4 5 6

3 3
1 2 3
4 5 6
7 8 9   //No specific formatting: e.g. data may be saved all on one line: 2 3 1 2 3 4 5 6 3 3 1 2 3 4 5 6 7 8 9
        //                          or they can be broken into lines in an arbitrary manner
```

3. Call *matrixProduct* method giving A and B matrices, and get the product-matrix (matrix C).

   Note: be aware of a potential *IllegalArgumentException* from *matrixProduct*; if this happens, arrange to catch it and output an error message. **You should not let the program crash.**

4. Output the content of matrix C on the screen, in an appropriate "matrix"-format: one line per matrix-row, separating values on the same row with space(s).

   For the above example your output should look like this (**only bold italic text is the output**):

```
Product matrix:
30  36  42          //c1,1=1·1+2·4 + 3·7=30    c1,2=1·2+2·5 + 3·8=36    c1,3=1·3+2·6 + 3·9=42
66  81  96          //c2,1=4·1+5·4 + 6·7=66    c2,2=4·2+5·5 + 6·8=81    c2,3=4·3+5·6 + 6·9=96
```

Compile and run your program, test it thoroughly:
Make sure the program works as expected and gives correct results (there are many online tools for matrix-multiplication so you can use one of them to check your data).
Try different size input matrices, including matrices that have only one row or only one column.
Also make sure that your program works as expected when input matrices are not eligible/valid for matrix-multiplication (they do not meet the above mentioned size-requirement).

How to get credit:
Get demo instructions from Canvas: go to *Project2* assignment and click the link for *demo*.
Run your program according to the **demo-instructions**. Finish all test-runs keeping the results of **all** tests on your screen. **After** all tests are **done**, send a request for me to come to your breakout room. Once I am in your room, share your screen (with the results) with me. *I may ask you to show your code and/or your input files, or I may ask you to run one of the tests in my presence.*
Both partners must be present for the demo.