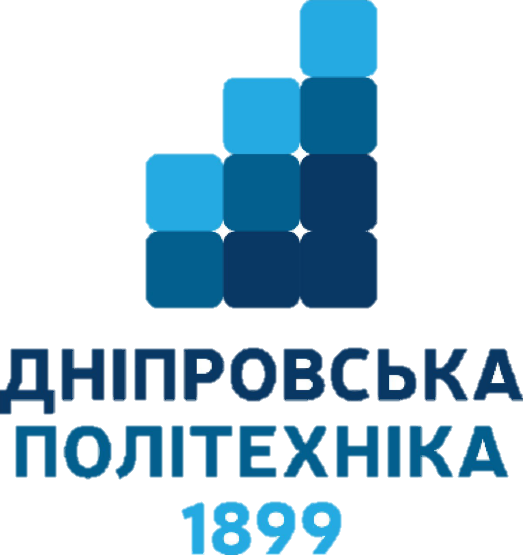
Міністерство освіти і науки України

Національний технічний університет

«Дніпровська політехніка»



Факультет інформаційних технологій

Кафедра системного аналізу і управління

*Звіт*

З дисципліни:

“ОПСJ”

**Виконав**

студент групи 124-18ск-1

Владимиров Я.Д

**GitHub**

https://github.com/gotoindex/javalabs.git

**Перевірив**

Мінєєв О.С

м. Дніпро

2020

Лабораторна робота №2

Хід роботи

Постановка задачі:

Розробити програму, що дозволить вам створити, як з клавіатури так і рандомно матрицю цілих чисел типу int заданої ширини та висоти(ввести з клавіатури), але не більше 20 на 20. Створити можливість пошуку в цій матриці мінімального і максимального елементу та розрахунок середнього арифметичного. Програма може бути написана в одному класі, обов'язково розбиття на методи. Обов'язкове використання клавіатури, під час вибору ручного чи рандомного створення матриці. Створення системи зчитування з клавіатури зробити будь-яким способом, наприклад завдяки класу Scanner. Scanner являє собою найпростішу систему сканування клавіатури. Диапазон рандомних чисел для створення елементів матриці повинен зверігатись в спеціальних константах.

Лістинг програми:

*Matrix.java:*

**package** lr2;  
  
**import** java.util.Random;  
**import** java.util.Scanner;  
  
*/\*\*  
 \* This class contains a custom implementation of a regular matrix with  
 \* additional methods that help enumerate certain parameters of the matrix,  
 \* like its maximum or minimum value.  
 \* The class uses a list of default java utility classes like {****@code*** *Random}  
 \* and {****@code*** *Scanner}.  
 \*  
 \* <p>Please make sure you are using the correct version of Idea,  
 \* if you're not sure which version you are using please contact  
 \* the administrator. The version intended for usage is maven-14.  
 \*  
 \* <p>The documentation for the methods contained in this class includes  
 \* brief descriptions of the <i>implementations</i>. Such descriptions should  
 \* be regarded as <i>implementation notes</i>, rather than parts of the  
 \* <i>specification</i>. Implementors should feel free to substitute other  
 \* algorithms, so long as the specification itself is adhered to. (For  
 \* example, the algorithm used by {****@code*** *sort(Object[])} does not have to be  
 \* a MergeSort, but it does have to be <i>stable</i>.)  
 \*  
 \* <p>This class is a member of the  
 \* <a href="https://github.com/gotoindex/javalabs">  
 \* Java Learning Course</a>.  
 \*  
 \** ***@author*** *Akim Vladimirov  
 \*/***public class** Matrix {  
  
 */\*  
 \* The following are constants used to manage matrix size and its  
 \* contents. As defined in the task, max matrix length and height  
 \* are both 20 elements.  
 \*  
 \* As for the random values, all randomly generated elements are  
 \* always between 5 and 10.  
 \*/* **private static final int *ROWS\_MAX*** = 20;  
 **private static final int *COLS\_MAX*** = 20;  
 **private static final int *RAND\_MIN*** = 5;  
 **private static final int *RAND\_MAX*** = 10;  
  
 *// This constant initializes a stream checking object for all input procedures.* **private static final** Scanner ***INPUT*** = **new** Scanner(System.***in***);  
  
 */\*  
 \* The matrix contains one necessary parameter which  
 \* describes the contents of the matrix.  
 \*  
 \* All of the parameters listed above are private since the  
 \* class is not meant to be operated outside the task's range.  
 \*/* **private final int** [][] **cells**;  
  
 */\*\*  
 \* This is the only constructor necessary for the task completion and thus the  
 \* only constructor in the code.  
 \*  
 \** ***@implNote*** *All parameters are entered from stream using a {****@code*** *Scanner} object.  
 \* Additionally, all outputs use {****@code*** *System}'s {****@code*** *print()} and {****@code*** *println()}  
 \* methods to pass the output information to the user.  
 \*/* **public** Matrix() {  
  
 */\*  
 \* Both {@code rows} and {@code cols} are temporary and are only needed to  
 \* initialize the size of the matrix.  
 \*/* **int** rows = **this**.inputMatrixSize(**"height"**, ***ROWS\_MAX***);  
 **int** cols = **this**.inputMatrixSize(**"width"**, ***COLS\_MAX***);  
  
 */\*  
 \* The matrix is only initialized in the current method. Its further  
 \* filling in is done in a separate method called {@code fillMatrix()}.  
 \*/* **this**.**cells** = **new int** [rows][cols];  
 System.***out***.println(**"Created a "** + rows + **"x"** + cols + **" matrix."**);  
 **this**.fillMatrix();  
  
 }  
  
 */\*\*  
 \* A basic input method used for acquiring all matrix sizes. It additionally checks  
 \* all inputs and constrains them to fit a selected constant (or any other variable).  
 \*  
 \* <p>In the context of the current task, the method is called twice to get and correct  
 \* the sizes of the matrix.  
 \*  
 \** ***@implNote*** *The result is entered from stream using a {****@code*** *Scanner} object.  
 \* In this case an input filter is present to prevent unexpected errors.  
 \*  
 \** ***@param size\_name*** *a name for the metric to display on the output. For example could be  
 \* <i>length</i> or <i>height</i>.  
 \** ***@param max*** *the max possible size of the current metric.  
 \*/* **private int** inputMatrixSize(String size\_name, **int** max) {  
  
 System.***out***.print(**"Please enter the "** + size\_name + **" of the matrix (1-20): "**);  
  
 *// The following cycle filters out all unwanted inputs.* **while** (!***INPUT***.hasNextInt()) ***INPUT***.nextLine();  
  
 */\*  
 \* When the correct input type is captured, its value is received and  
 \* corrected to fit between 1 and a selected positive integer value.  
 \*/* **return** Math.*min*(Math.*max*(((***INPUT***.hasNextInt()) ? ***INPUT***.nextInt() : 1), 1), max);  
  
 }  
  
 */\*\*  
 \* This method doesn't directly initialize each cell of the matrix. Alternatively, it  
 \* asks the user in which way do they wish to initialize them: randomly (using the  
 \* {****@code*** *Random} class), or by hand.  
 \*  
 \** ***@implNote*** *All parameters are entered from stream using a {****@code*** *Scanner} object.  
 \* In this case an input filter is present to prevent unexpected errors.  
 \* Additionally, all outputs use {****@code*** *System}'s {****@code*** *print()} and {****@code*** *println()}  
 \* methods to pass the output information to the user.  
 \*/* **private void** fillMatrix() {  
  
 System.***out***.print(**"Fill the matrix in by hand? [Y/n]: "**);  
  
 *// This additional nextLine is used to resolve any previous inputs.* ***INPUT***.nextLine();  
  
 */\*  
 \* When the correct input type is captured, its value is received and  
 \* corrected to be fully lowercase, so both 'Y' and 'y' are accepted.  
 \*  
 \* Additionally, any other inputs that are not 'Y' or 'y', are accounted  
 \* for as the 'n' answer.  
 \*/* **boolean** random = ***INPUT***.hasNextLine() && !(***INPUT***.nextLine().toLowerCase().equals(**"y"**));  
 System.***out***.println(**"You chose to fill it in "** + (random ? **"randomly"** : **"by hand"**) + **"."**);  
 **if** (random) **this**.fillMatrixRandomly(); **else this**.fillMatrixByHand();  
  
 }  
  
 */\*\*  
 \* This method directly initializes each cell of the matrix using the  
 \* {****@code*** *Random} class to generate all elements.  
 \*  
 \** ***@implNote*** *This is where RAND\_MIN and RAND\_MAX constants are used to  
 \* define the range of each random number.  
 \*/* **private void** fillMatrixRandomly() {  
  
 *// The following variable is a temporary object for generating values for the elements.* Random generator = **new** Random();  
  
 **for** (**int** row = 0; row < **this**.**cells**.**length**; row++)  
 **for** (**int** col = 0; col < **this**.**cells**[0].**length**; col++)  
 **this**.**cells**[row][col] = generator.nextInt(***RAND\_MAX*** - ***RAND\_MIN***) + ***RAND\_MIN***;  
  
 }  
  
 */\*\*  
 \* This method directly initializes each cell of the matrix by asking the user  
 \* to enter the value of each element and then waiting for the correct input.  
 \*  
 \** ***@implNote*** *All parameters are entered from stream using a {****@code*** *Scanner} object.  
 \* In this case an input filter is present to prevent unexpected errors.  
 \* Additionally, all outputs use {****@code*** *System}'s {****@code*** *print()} and {****@code*** *println()}  
 \* methods to pass the output information to the user.  
 \*/* **private void** fillMatrixByHand() {  
  
 **for** (**int** row = 0; row < **this**.**cells**.**length**; row++)  
 **for** (**int** col = 0; col < **this**.**cells**[0].**length**; col++) {  
 System.***out***.print(**"Enter element ["** + (row + 1) + **", "** + (col + 1) + **"]: "**);  
  
 *// The following cycle filters out all unwanted inputs.* **while** (!***INPUT***.hasNextInt()) ***INPUT***.nextLine();  
  
 *// When the correct input type is captured, its value is assigned to the current cell.* **this**.**cells**[row][col] = ***INPUT***.nextInt();  
 }  
  
 }  
  
 */\*\*  
 \* This method simply displays the contents of the matrix to user in a  
 \* grid-like pattern.  
 \*/* **public void** showMatrix() {  
  
 System.***out***.println(**"Your matrix:"**);  
 **for** (**int** [] row: **this**.**cells**) {  
 **for** (**int** cell: row) System.***out***.print(cell + **" "**);  
 System.***out***.println();  
 }  
  
 }  
  
 */\*\*  
 \* This method finds the min value in the matrix and then displays it to the user.  
 \*  
 \** ***@implNote*** *The algorithm is a simple comparison of each element to the initial  
 \* value, which is considered to be the first element of the matrix.  
 \*/* **public void** showMatrixMin() {  
  
 **int** min = **this**.**cells**[0][0];  
 **for** (**int** [] row : **this**.**cells**) {  
 **for** (**int** cell : row) **if** (cell < min) min = cell;  
 }  
 System.***out***.println(**"Min matrix element: "** + min);  
  
 }  
  
 */\*\*  
 \* This method finds the max value in the matrix and then displays it to the user.  
 \*  
 \** ***@implNote*** *The algorithm is a simple comparison of each element to the initial  
 \* value, which is considered to be the first element of the matrix.  
 \*/* **public void** showMatrixMax() {  
  
 **int** max = **this**.**cells**[0][0];  
 **for** (**int** [] row : **this**.**cells**) {  
 **for** (**int** cell : row) **if** (cell > max) max = cell;  
 }  
 System.***out***.println(**"Max matrix element: "** + max);  
  
 }  
  
 */\*\*  
 \* This method calculates the average of the matrix and then displays it to the user.  
 \*  
 \** ***@implNote*** *The algorithm is a simple summarizing of all elements of the matrix and  
 \* a following division by the result of multiplication of its sizes.  
 \*/* **public void** showMatrixAvg() {  
  
 **int** sum = 0;  
 **for** (**int** [] row: **this**.**cells**)  
 **for** (**int** cell: row) sum += cell;  
 System.***out***.println(**"Avg of elements: "** + sum / (**this**.**cells**.**length** \* **this**.**cells**[0].**length**));  
  
 }  
  
 */\*\*  
 \* Initializes a {****@code*** *Matrix} object and displays its parameters to the user.  
 \*  
 \** ***@implNote*** *The algorithm uses a built-in method implemented by the {****@code*** *System} class  
 \* and has no particular algorithmic value.  
 \*  
 \** ***@implNote*** *This is the main function. It executes everytime the file is called.  
 \*  
 \** ***@param args*** *regular string-type arguments. Note, that all arguments in  
 \* this case will be ignored.  
 \*/* **public static void** main(String[] args) {  
  
 Matrix m = **new** Matrix();  
 m.showMatrix();  
 m.showMatrixMin();  
 m.showMatrixMax();  
 m.showMatrixAvg();  
  
 }  
  
}

Результати виконання:

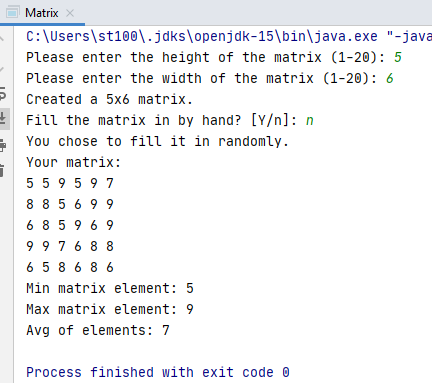


Рисунок 2.1 – Результат виконання програми

Висновок: Під час виконання лабораторної роботи №2 було розроблено програму, що дозволить створити матрицю цілих чисел типу int заданої ширини та висоти.