МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Лабораторна робота №1

з курсу

«Інженерія даних та знань»

*на тему:*

**«Основи побудови додатків для обробки наборів даних мовою**

**програмування Java»**

Виконав:

ст. гр. ITIT-12

Галах А.Б.

Прийняла:

Рибчак З.Л.

**Львів 2017**

**Мета роботи:**

Вивчення можливостей мови програмування Java для реалізації алгоритмів інженерії даних та набуття навичок опрацювання векторів та таблиці

**Індивідуальне завдання:**

**Варіант 3**

Ознайомитися з основними типами даних мови програмування Java та реалізувати калькулятор простих чисел з підтримкою зворотного польського запису. Додати можливість по елементного +,-,/,\* векторів та таблиць (розріджених матриць). Розробити процедури і функції для забезпечення доступу (читання-запис) до елементів матриці. В контрольному прикладі забезпечити читання і запис всіх елементів матриці. Оцінити час виконання операцій.

|  |  |
| --- | --- |
| **№** | **Завдання** |
| 3 | Всі нульові елементи розміщені в шаховому порядку,починаючи з 2-го елементу 1-ї стрічки |

**Порядок виконання:**

**Калькулятор простих чисел з підтримкою зворотного польського запису**

Було реалізовано калькулятор з підтримкою зворотного польського запису, що парсить математичний вираз та переводить його в зворотний польський запис для подальшого обчислення проходячи з ліва направо записуючи елементи в стек та обчислюючи їх при находжені оператора. Калькулятор підтримує дужки та такі оператори як:

1. Піднесення до степеню
2. Множення
3. Ділення
4. Остача від ділення
5. Додавання
6. Віднімання

Код програми:

Клас Calculator.java

**public class** Calculator {  
  
 **private static final** Map<Character, Integer> ***sOperators*** = **new** HashMap<>();  
 **static** {  
 ***sOperators***.put(**'^'**, 3);  
 ***sOperators***.put(**'\*'**, 2);  
 ***sOperators***.put(**'/'**, 2);  
 ***sOperators***.put(**'%'**, 2);  
 ***sOperators***.put(**'+'**, 1);  
 ***sOperators***.put(**'-'**, 1);  
 }  
  
 **public static void** main(String[] args) **throws** Exception {  
 BufferedReader d = **new** BufferedReader(**new** InputStreamReader(System.***in***));  
 String sIn;  
  
 **try** {  
 System.***out***.println(**"Input expression for calculation. Supported digits, operations +,-,\*,/,^,% and priority in the form of brackets ( and ):"**);  
 sIn = d.readLine();  
 d.close();  
 sIn = *toReversePolandNotation*(sIn);  
 System.***out***.println(String.*format*(**"Reverse poland notation: %s"**, sIn));  
 System.***out***.println(String.*format*(**"Result: %s"**, String.*valueOf*(*calculate*(sIn))));  
 } **catch** (Exception e) {  
 System.***out***.println(e.getMessage());  
 }  
}  
  
 */\*\*  
 \* Converts string in reverse poland notation  
 \** ***@param input*** *Input string  
 \** ***@return*** *Result string in reverse poland notation  
 \*/* **private static** String toReversePolandNotation(String input) **throws** Exception {  
 StringBuilder operationStack = **new** StringBuilder();  
 StringBuilder output = **new** StringBuilder();  
 **char** inputSymbol;  
 **char** tempSymbol;  
  
 **for** (**int** i = 0; i < input.length(); i++) {  
 inputSymbol = input.charAt(i);  
 **if** (*isOperator*(inputSymbol)) {  
 **while** (operationStack.length() > 0) {  
 tempSymbol = operationStack.charAt(operationStack.length() - 1);  
 **if** (*isOperator*(tempSymbol) && (*getOperatorPriority*(inputSymbol) <= *getOperatorPriority*(tempSymbol))) {  
 output.append(**" "**).append(tempSymbol).append(**" "**);  
 operationStack.setLength(operationStack.length() - 1);  
 } **else** {  
 output.append(**" "**);  
 **break**;  
 }  
 }  
 output.append(**" "**);  
 operationStack.append(inputSymbol);  
 } **else if** (**'('** == inputSymbol) {  
 operationStack.append(inputSymbol);  
 } **else if** (**')'** == inputSymbol) {  
 tempSymbol = operationStack.charAt(operationStack.length() - 1);  
 **while** (**'('** != tempSymbol) {  
 **if** (operationStack.length() < 1) {  
 **throw new** Exception(**"Wrong expression. Check it again!"**);  
 }  
 output.append(**" "**).append(tempSymbol);  
 operationStack.setLength(operationStack.length() - 1);  
 tempSymbol = operationStack.charAt(operationStack.length() - 1);  
 }  
 operationStack.setLength(operationStack.length() - 1);  
 } **else** {  
 output.append(inputSymbol);  
 }  
 }  
 **while** (operationStack.length() > 0) {  
 output.append(**" "**).append(operationStack.charAt(operationStack.length() - 1));  
 operationStack.setLength(operationStack.length() - 1);  
 }  
 **return** output.toString();  
 }  
  
 **private static boolean** isOperator(**char** c) {  
 **return *sOperators***.containsKey(c);  
 }  
  
  
 **private static int** getOperatorPriority(**char** operator) {  
 **return *sOperators***.getOrDefault(operator, -1);  
 }  
  
  
 **private static double** calculate(String input) **throws** Exception {  
 **double** dA = 0;  
 **double** dB = 0;  
 String temp;  
 Deque<Double> stack = **new** ArrayDeque<Double>();  
 StringTokenizer st = **new** StringTokenizer(input);  
  
 **while**(st.hasMoreTokens()) {  
 **try** {  
 temp = st.nextToken().trim();  
 **if** (1 == temp.length() && *isOperator*(temp.charAt(0))) {  
 **if** (stack.size() < 2) {  
 **throw new** Exception(**"Wrong data amount in stack for operation"** + temp);  
 }  
 dB = stack.pop();  
 dA = stack.pop();  
 **switch** (temp.charAt(0)) {  
 **case '+'**:  
 dA += dB;  
 **break**;  
 **case '-'**:  
 dA -= dB;  
 **break**;  
 **case '/'**:  
 dA /= dB;  
 **break**;  
 **case '\*'**:  
 dA \*= dB;  
 **break**;  
 **case '%'**:  
 dA %= dB;  
 **break**;  
 **case '^'**:  
 dA = Math.*pow*(dA, dB);  
 **break**;  
 **default**:  
 **throw new** Exception(**"Unacceptable operation"** + temp);  
 }  
 stack.push(dA);  
 } **else** {  
 dA = Double.*parseDouble*(temp);  
 stack.push(dA);  
 }  
 } **catch** (Exception e) {  
 **throw new** Exception(**"Unacceptable symbol in operation"**);  
 }  
 }  
  
 **if** (stack.size() > 1) {  
 **throw new** Exception(**"Operators number and operands number are different"**);  
 }  
 **return** stack.pop();  
 }  
}

**Результат виконання програми**

Результати виконання програми представлений на Рис.1. Як видно з рисунку вираз парситься та переводиться в зворотній польський запис з подальшим обчисленням. Перевірка показала, що результат обчислення виразу програми повністю збігається з обчисленням виразу на калькуляторі (Рис.2).

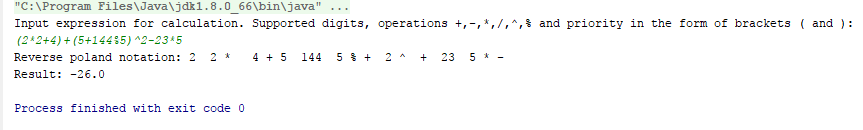


Рис.1 Результат виконання програми

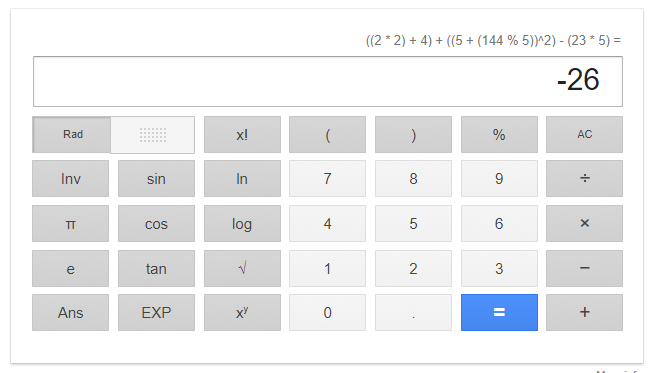


Рис.2 Обчислення виразу на калькуляторі

**Реалізація операцій з матрицями**

Було додано можливість по елементного додавання матриць. Розроблено процедури для читання та запису матриць в файл використовуючи формат Json. Для конвертації матриці в формат Json було використано бібліотеку GSON від Google. Було реалізовано доступ та редагування елементів матриці. Було реалізовано генерацію матриці згідно з індивідуальним завданням.

Код програми:

**public class** Main {  
  
 **private static** Gson *gson* = **new** GsonBuilder().create();  
 **private static** String *FILE* = **"G:\\навчання\\5 курс\\Інженерія даних та знань\\lab1\\lab1\\testData.json"**;  
  
 **public static void** main(String[] args) **throws** Exception {  
 *addMatrix*();  
 *writeToFile*(*FILE*, *gson*.toJson(*generateMatrix*()));  
 String json = *readFile*(**new** File(*FILE*));  
 **int**[][] matrix = *gson*.fromJson(json, **int**[][].**class**);  
 *printArray*(matrix);  
 *accessMatrix*(matrix);  
 }  
  
 **private static void** addMatrix() {  
 **long** t1 = System.*currentTimeMillis*();  
 **int** [][] m1 = {{1, 1}, {2, 2}};  
 **int** [][] m2 = {{3, 3}, {4, 4}};  
 **int** [][] m3 = **new int**[2][2];  
 **for** (**int** i = 0; i < m1.**length**; i++) {  
 **for** (**int** j = 0; j < m1[i].**length**; j++) {  
 m3[i][j] = m1[i][j] + m2[i][j];  
 }  
 }  
 System.***out***.println(**"Matrix1:"**);  
 *printArray*(m1);  
 System.***out***.println(**"Matrix2:"**);  
 *printArray*(m2);  
 System.***out***.println(**"Matrix1 + Matrix2:"**);  
 *printArray*(m3);  
 System.***out***.println(**"addMatrix Time: "** + String.*valueOf*(System.*currentTimeMillis*() - t1));  
 }  
  
 **private static void** accessMatrix(**int**[][] matrix) **throws** IOException {  
 BufferedReader d = **new** BufferedReader(**new** InputStreamReader(System.***in***));  
 **while** (**true**) {  
 System.***out***.println(String.*format*(**"rows number"**, matrix.**length**));  
 System.***out***.println(**"Input row"**);  
 **int** row = Integer.*parseInt*(d.readLine());  
 **if** (matrix.**length** <= row) {  
 System.***out***.println(**"Wrong row"**);  
 **return**;  
 }  
  
 System.***out***.println(**"Input column"**);  
 System.***out***.println(String.*format*(**"columns number"**, matrix[row].**length**));  
 **int** column = Integer.*parseInt*(d.readLine());  
 **if** (matrix[row].**length** <= column) {  
 System.***out***.println(**"Wrong column"**);  
 **return**;  
 }  
 System.***out***.println(**"Rquested element: "** + String.*valueOf*(matrix[row][column]));  
 System.***out***.println(**"Do you want to change it? [y/n]"**);  
 String answer = d.readLine();  
 **if** (**"y"**.equals(answer)) {  
 System.***out***.println(**"Input new value"**);  
 **int** value = Integer.*parseInt*(d.readLine());  
 matrix[row][column] = value;  
 String json = *gson*.toJson(matrix);  
 System.***out***.println(**"New Matrix: "**);  
 *printArray*(matrix);  
 *writeToFile*(*FILE*, json);  
 } **else** {  
 **break**;  
 }  
 }  
 d.close();  
 }  
  
 **private static void** printArray(**int** [][] array) {  
 **for** (**int** i = 0; i < array.**length**; i++) {  
 **for** (**int** j = 0; j < array[i].**length**; j++) {  
 System.***out***.print(String.*format*(**"%2d"**, array[i][j]));  
 }  
 System.***out***.println();  
 }  
 }  
  
 **private static int**[][] generateMatrix() **throws** IOException{  
 **long** t1 = System.*currentTimeMillis*();  
 BufferedReader d = **new** BufferedReader(**new** InputStreamReader(System.***in***));  
 System.***out***.println(**"Input row numbers"**);  
 **int** row = Integer.*parseInt*(d.readLine());  
 System.***out***.println(**"Input column numbers"**);  
 **int** column = Integer.*parseInt*(d.readLine());  
 **int** [][] matrix = **new int**[row][column];  
  
 **for** (**int** i = 0; i < row; i++) {  
 **for** (**int** j = 0; j < column; j++) {  
 matrix[i][j] = i % 2 != 0 ? (j % 2 == 0 ? 0 : 1) : (j % 2 != 0 ? 0 : 1) ;  
 }  
 }  
 System.***out***.println(**"generateMatrix Time: "** + String.*valueOf*(System.*currentTimeMillis*() - t1));  
 **return** matrix;  
 }  
  
 **private static void** writeToFile(String fileName, String content) {  
 **long** t1 = System.*currentTimeMillis*();  
 **try** {  
 *// Writing to a file* File file=**new** File(fileName);  
 FileWriter fileWriter = **new** FileWriter(file, **false**);  
 System.***out***.println(**"Writing JSON object to file"**);  
 System.***out***.println(**"-----------------------"**);  
 System.***out***.println(content);  
 fileWriter.write(content);  
 fileWriter.flush();  
 fileWriter.close();  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 System.***out***.println(**"writeToFile Time: "** + String.*valueOf*(System.*currentTimeMillis*() - t1));  
 }  
  
 **private static** String readFile(File file) **throws** FileNotFoundException, IOException {  
 **long** t1 = System.*currentTimeMillis*();  
 InputStream in = **new** FileInputStream(file);  
 **byte**[] b = **new byte**[(**int**) file.length()];  
 **int** len = b.**length**;  
 **int** total = 0;  
  
 **while** (total < len) {  
 **int** result = in.read(b, total, len - total);  
 **if** (result == -1) {  
 **break**;  
 }  
 total += result;  
 }  
  
 System.***out***.println(**"readFile Time: "** + String.*valueOf*(System.*currentTimeMillis*() - t1));  
 **return new** String(b, **"UTF-8"**);  
 }  
}

**Результат виконання програми**

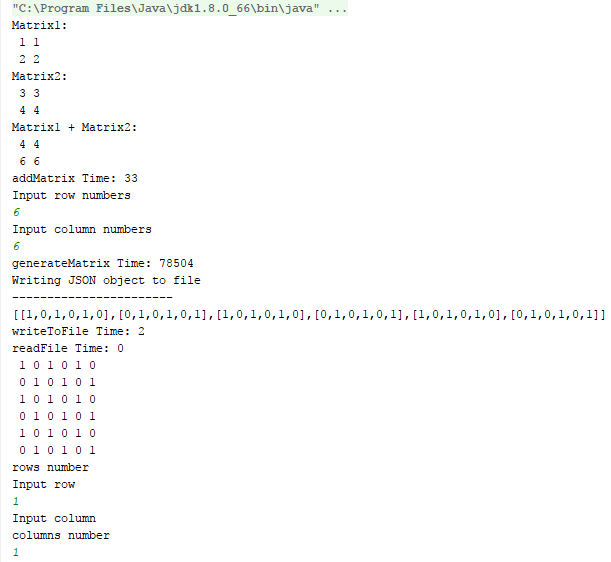
Результат виконання програми представлений на Рис.3.

1. Було проведено по елементне додавання двох матриць та виведений результат обчислення. Час виконання операції 33 мс.
2. Було згенеровано матрицю згідно індивідуального завдання розміром 6 на 6, розміри було визначено через консольний ввід.
3. Після генерації матрицю було згенеровано в формат Json виведено на екран і записано в текстовий файл.
4. В подальшому матриця була відредагована (був змінений 1 елемент 1 рядка матриці) записана в файл та відображена на екрані.

Результуюча матриця в форматі Json:

[[1,0,1,0,1,0],[0,7,0,1,0,1],[1,0,1,0,1,0],[0,1,0,1,0,1],[1,0,1,0,1,0],[0,1,0,1,0,1]

**Висновок:** При виконанні даної лабораторної було реалізовано калькулятор на основі зворотного польського запису, та проведено основні операції на матрицями на мові програмування Java. Для серіалізації матриць було використано Json формат та бібліотеку GSON.



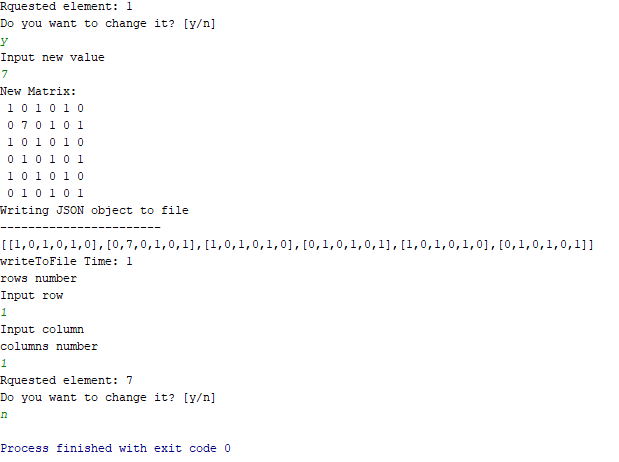


Рис.3 Результат виконання операцій з матрицями