**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 4**

по дисциплине:

«Визуальные средства разработки программных приложений»

на тему:

«Коллекции»

Выполнил: студент гр. ИТП-31

Коркуц С. И.

Принял: ассистент

Гуменников Е.Д.

Гомель 2020

**Цель работы:** изучить работу с коллекциями в *Java*.

**Задание:** Вариант 7. Условия задания на рисунке 1.

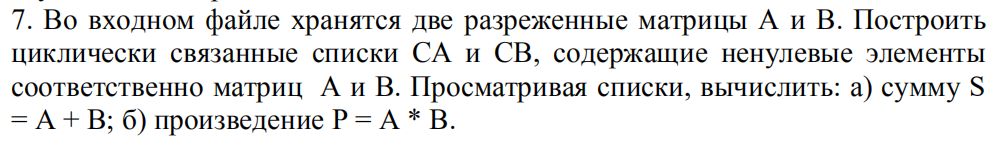


Рисунок 1 – Вариант задания

Структура созданных классов на рисунке 2.

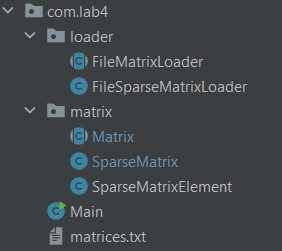


Рисунок 2 – Структура созданных классов

Изначально был придуман формат хранения разреженных матриц. Далее был создан файл *matrices.txt*, в котором хранятся разреженные матрицы. Содержимое файла, в котором разреженные матрицы хранятся в определённом формате на рисунке 3.

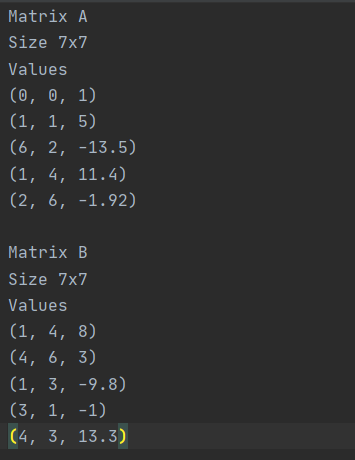


Рисунок 3 – Содержимое файла, в котором хранятся разреженные матрицы

Далее был написан абстрактный класс для считывания матриц из файла. От него был унаследован конкретный класс для считывания именно разреженной матрицы.

После этого был написан абстрактный класс матрицы, от которого был унаследован конкретный класс разреженной матрицы.

Во время выполнения выполняются задания из варианта. Результат выполнения заданий на рисунке 4.

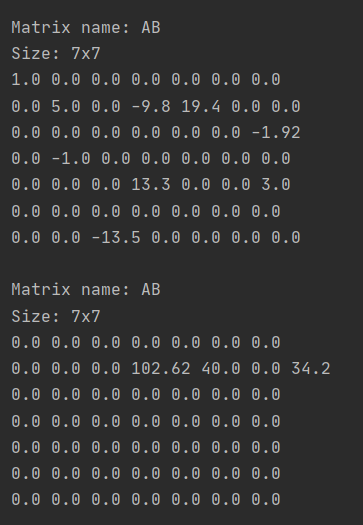
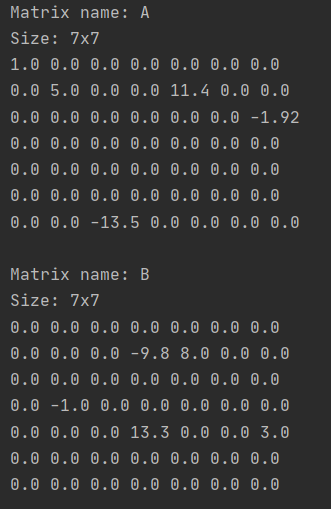


Рисунок 4 – Результат выполнения задания

**Вывод:** Были написаны классы для работы с разреженными матрицами. Была изучена работа с коллекциями в *Java*.

**ПРИЛОЖЕНИЕ А**

Листинг созданных классов

Содержимое файла

Matrix A  
Size 7x7  
Values  
(0, 0, 1)  
(1, 1, 5)  
(6, 2, -13.5)  
(1, 4, 11.4)  
(2, 6, -1.92)  
  
Matrix B  
Size 7x7  
Values  
(1, 4, 8)  
(4, 6, 3)  
(1, 3, -9.8)  
(3, 1, -1)  
(4, 3, 13.3)

package com.lab4.loader;  
  
import com.lab4.matrix.Matrix;  
  
import java.io.Reader;  
import java.util.List;  
  
public abstract class FileMatrixLoader {  
 public abstract List<Matrix> load(Reader reader) throws Exception;  
}

package com.lab4.loader;  
  
import com.lab4.loader.FileMatrixLoader;  
import com.lab4.matrix.Matrix;  
import com.lab4.matrix.SparseMatrix;  
import com.lab4.matrix.SparseMatrixElement;  
  
import java.io.Reader;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
public class FileSparseMatrixLoader extends FileMatrixLoader {  
 @Override  
 public List<Matrix> load(Reader reader) throws Exception{  
 String text = getRowTextFromFile(reader);  
 return getMatricesFromText(text);  
 }  
  
 private String getRowTextFromFile(Reader reader) throws Exception {  
 StringBuilder builder = new StringBuilder();  
 int c;  
 while((c = reader.read()) != -1) {  
 builder.append((char)c);  
 }  
 return builder.toString();  
 }  
  
 private List<Matrix> getMatricesFromText(String text) throws Exception {  
 Pattern pattern = Pattern.compile("\\s\*Matrix (\\s\*\\w+\\s\*)+\\s\*" +  
 "\\s\*Size \\s\*\\d+x\\d+\\s\*" +  
 "\\s\*Values\\s\*" +  
 "(\\s\*(\\((\\s\*-?(\\d+(\\.?\\d+)?)\\s\*),(\\s\*-?(\\d+(\\.?\\d+)?)\\s\*),(\\s\*-?(\\d+(\\.?\\d+)?)\\s\*)\\))\\s\*)\*");  
 Matcher matcher = pattern.matcher(text);  
 List<Matrix> matrices = new ArrayList<>();  
 while(matcher.find()) {  
 matrices.add(getMatrixFromText(matcher.group()));  
 }  
 return matrices;  
 }  
  
 private Matrix getMatrixFromText(String text) throws Exception {  
 String[] lines = text.split("(\\r\\n)+");  
 String name = String.join(" ", Arrays.copyOfRange(lines[0].split("\\s+"), 1, lines[0].split("\\s+").length));  
 String[] sizes = lines[1].split("\\s+")[1].split("\\s\*x\\s\*");  
 int rows, columns;  
 try {  
 rows = Integer.parseInt(sizes[0]);  
 columns = Integer.parseInt(sizes[1]);  
 }  
 catch (Exception exception){  
 throw new Exception("Неправильное описание размера матрицы в файле");  
 }  
 List<SparseMatrixElement> elements = new ArrayList<>();  
 for(int i = 3; i < lines.length; i++) {  
 String numbersString = lines[i].replaceAll("\\s\*\\(\\s\*", "").replaceAll("\\s\*\\)\\s\*", "");  
 String[] numbers = numbersString.split("\\s\*,\\s\*");  
 elements.add(new SparseMatrixElement(Integer.parseInt(numbers[0]), Integer.parseInt(numbers[1]), Double.parseDouble(numbers[2])));  
 }  
 return new SparseMatrix(name, rows, columns, elements);  
 }  
}

package com.lab4.matrix;  
  
public class SparseMatrixElement {  
 int rowIndex;  
 int columnIndex;  
 double value;  
  
 public SparseMatrixElement(int rowIndex, int columnIndex, double value) {  
 this.rowIndex = rowIndex;  
 this.columnIndex = columnIndex;  
 this.value = value;  
 }  
  
 public static SparseMatrixElement add(SparseMatrixElement firstElement, SparseMatrixElement secondElement) throws Exception {  
 if(firstElement.rowIndex == secondElement.rowIndex && firstElement.columnIndex == secondElement.columnIndex) {  
 return new SparseMatrixElement(firstElement.rowIndex, firstElement.columnIndex, firstElement.value + secondElement.value);  
 }  
 else  
 throw new Exception("Эти элементы находятся не на одинаковых позициях");  
 }  
}

package com.lab4.matrix;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class SparseMatrix extends Matrix {  
 List<SparseMatrixElement> elements;  
  
 public SparseMatrix(String name, int rows, int columns, List<SparseMatrixElement> elements) {  
 super(name, rows, columns);  
 this.elements = elements;  
 }  
  
 public SparseMatrix(String name, double[][] matrix) {  
 super(name, matrix);  
 this.elements = new ArrayList<>();  
 for (int i = 0; i < rows; i++) {  
 for(int j = 0; j < columns; j++) {  
 if(matrix[i][j] != 0) {  
 elements.add(new SparseMatrixElement(i, j, matrix[i][j]));  
 }  
 }  
 }  
 }  
  
 @Override  
 public Matrix transpose() {  
 List<SparseMatrixElement> newElements = new ArrayList<>();  
 for (SparseMatrixElement element:  
 this.elements) {  
 newElements.add(new SparseMatrixElement(element.columnIndex, element.rowIndex, element.value));  
 }  
 return new SparseMatrix("Transposed " + this.name, this.columns, this.rows, newElements);  
 }  
  
 @Override  
 public double getValueAt(int row, int column) throws IndexOutOfBoundsException {  
 if(row > rows || column > columns || row < 0 || column < 0) {  
 throw new IndexOutOfBoundsException();  
 }  
 for (SparseMatrixElement element :  
 elements) {  
 if(element.rowIndex == row && element.columnIndex == column) {  
 return element.value;  
 }  
 }  
 return 0;  
 }  
  
 public static SparseMatrix sum(SparseMatrix firstMatrix, SparseMatrix secondMatrix) throws Exception {  
 if(!(firstMatrix.rows == secondMatrix.rows && firstMatrix.columns == secondMatrix.columns)) {  
 throw new Exception("Разные размеры матриц");  
 }  
 List<SparseMatrixElement> firstMatrixElements = new ArrayList<>(firstMatrix.elements);  
 List<SparseMatrixElement> secondMatrixElements = new ArrayList<>(secondMatrix.elements);  
 List<SparseMatrixElement> newMatrixElements = new ArrayList<>();  
 for(int i = 0; i < firstMatrixElements.size(); i++) {  
 boolean flag = true;  
 for(int j = 0; j < secondMatrixElements.size(); j++) {  
 if(firstMatrixElements.get(i).rowIndex == secondMatrixElements.get(j).rowIndex &&  
 firstMatrixElements.get(i).columnIndex == secondMatrixElements.get(j).columnIndex) {  
 newMatrixElements.add(SparseMatrixElement.add(firstMatrixElements.get(i), secondMatrixElements.get(j)));  
 secondMatrixElements.remove(secondMatrixElements.get(j));  
 flag = false;  
 break;  
 }  
 }  
 if(flag) {  
 newMatrixElements.add(firstMatrixElements.get(i));  
 }  
 }  
 newMatrixElements.addAll(secondMatrixElements);  
 return new SparseMatrix(firstMatrix.name + secondMatrix.name, firstMatrix.rows, firstMatrix.columns, newMatrixElements);  
 }  
  
 public static SparseMatrix prod(SparseMatrix firstMatrix, SparseMatrix secondMatrix) throws Exception {  
 if(firstMatrix.columns != secondMatrix.rows) {  
 throw new Exception("Матрицы не могут быть умножены");  
 }  
 double[][] matrix = new double[firstMatrix.rows][secondMatrix.columns];  
 for(int i = 0; i < firstMatrix.rows; i++) {  
 for(int j = 0; j < secondMatrix.columns; j++) {  
 double accum = 0;  
 for(int k = 0; k < firstMatrix.columns; k++) {  
 accum += firstMatrix.getValueAt(i, k) \* secondMatrix.getValueAt(k, j);  
 }  
 matrix[i][j] = accum;  
 }  
 }  
 return new SparseMatrix(firstMatrix.name + secondMatrix.name, matrix);  
 }  
  
 @Override  
 public String toString() {  
 double[][] matrix = new double[rows][columns];  
 for (SparseMatrixElement element :  
 elements) {  
 matrix[element.rowIndex][element.columnIndex] = element.value;  
 }  
 StringBuilder builder = new StringBuilder();  
 builder.append(String.format("Matrix name: %s\nSize: %dx%d\n", name, rows, columns));  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < columns; j++) {  
 builder.append(String.valueOf(matrix[i][j]) + " ");  
 }  
 builder.append("\n");  
 }  
 return builder.toString();  
 }  
}

package com.lab4.matrix;  
  
public abstract class Matrix {  
 String name;  
 int rows;  
 int columns;  
  
 public Matrix(String name, double[][] matrix) {  
 this.name = name;  
 rows = matrix.length;  
 columns = matrix[0].length;  
 }  
  
 public Matrix(String name, int rows, int columns) {  
 this.name = name;  
 this.rows = rows;  
 this.columns = columns;  
 }  
  
 public abstract Matrix transpose();  
  
 public abstract double getValueAt(int row, int column) throws IndexOutOfBoundsException;  
}

package com.lab4;  
  
import com.lab4.loader.FileSparseMatrixLoader;  
import com.lab4.matrix.Matrix;  
import com.lab4.matrix.SparseMatrix;  
  
import java.io.FileReader;  
import java.io.Reader;  
import java.util.List;  
  
public class Main {  
  
 public static void main(String[] args) {  
 try(Reader reader = new FileReader("C:\\Users\\stani\\OneDrive\\Cloud\\University\\5sem\\5semJAVA\\lab4\\src\\com\\lab4\\matrices.txt")) {  
 List<Matrix> matrices = new FileSparseMatrixLoader().load(reader);  
 try {  
 System.out.println(matrices.get(0));  
 System.out.println(matrices.get(1));  
 Matrix sumMatrix = SparseMatrix.sum((SparseMatrix)matrices.get(0), (SparseMatrix)matrices.get(1));  
 System.out.println(sumMatrix);  
 Matrix prodMatrix = SparseMatrix.prod((SparseMatrix)matrices.get(0), (SparseMatrix)matrices.get(1));  
 System.out.println(prodMatrix);  
 }  
 catch (Exception exception) {  
 System.out.println(exception.getMessage());  
 }  
 } catch (Exception e) {  
 System.out.println(e.getMessage());  
 }  
 }  
}