Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

Кафедра моделювання систем і технологій

**ЛАБОРАТОРНА РОБОТА №2**

з дисципліни «Об’єктно-орієнтоване програмування»

Тема: Трикутник Флойда, файли та обробка строкових типів даних

Виконав студент 2 курсу

Групи КС-21

Клочко Андрій Володимирович

Перевірив:

к.т.н., доц. Нагорний К.А.

Харків – 2020

# ВСТУП

**Тема:** Розробка додатку для обробки строк тексту отриманого із текстового файлу.

**Мета роботи:** Набути базові навики роботи з текстовими файлами, такі як: читання вмісту файлу, запис обробленого тексту до файлу. Набути навики обробки даних строкового типу.

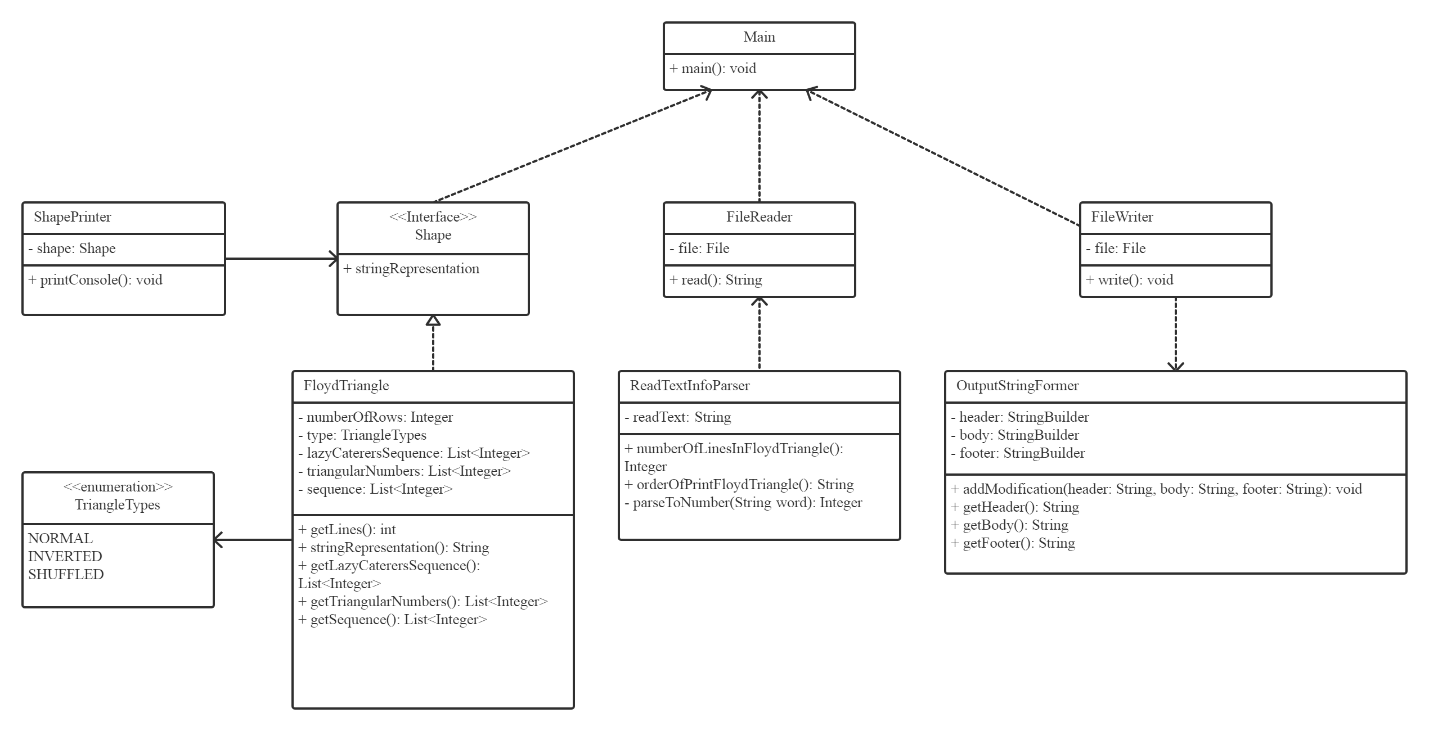
**Завдання:** дано два текстових фрагменти, що містяться у двох текстових файлах. Реалізувати зчитування тесту із вхідних файлів, обробку отриманого тексту, та запис результатів обробки у вихідні файли.

Формат вихідних файлів – HEADER – BODY. Де HEADER містить опис строкових перетворень. BODY містить фрагменти тексту (ів), отриманих в результаті модифікації строк вхідного тексту.

1. Файл «Digital values» містить фрагмент тексту, що записаний в одну строку. Речення в якій розділені крапкою «.». Обробка тексту передбачає отримання декількох варіантів результуючих текстових фрагментів при якій одна і та ж вхідна послідовність строк зазнала різні перетворення (наприклад п.6 та п.7 і т.д.). Перетворення:
   1. Видобути вміст файлу у відповідний об’єкт;
   2. Розділити строку по роздільнику «.» на речення;
   3. Розділити кожне речення на окремі слова або сполучення символів, наприклад "3";
   4. Порахувати загальну кількість слів;
   5. Порахувати загальну кількість символів;
   6. Замінити у кожному слові строкову першу літеру на прописний літерал;
   7. Замінити у кожному другому слові усі строкові літери на прописні, наприклад: замінити У кожному ДРУГОМУ слові УСІ строкові ...
   8. Скласти нові «слова», які складаються із перших і останніх двох літер кожного слова, наприклад: приклад 🡪 прад;
   9. Усі цифри мають бути записані літерами, наприклад: 1 🡪 One;
   10. Порахувати кількість входжень у текст кожного окремого слова.
2. Файл «Flyby missions» містить табличні дані, які організовані у три стовпчики. Перетворення:
   1. Вилучення вмісту файла у відповідний об’єкт;
   2. Розділити строку у відповідності із роздільником;
   3. Порахувати загальну кількість слів;
   4. Порахувати загальну кількість символів;
   5. Представити усі числа у форматі, в якому кожні три цифри розділяються комою «,», наприклад: 1000 🡪 1,000;
   6. Усі дати мають бути представлені у форматі «dd.mm.yyyy», для цього потрібно використати відповідний клас, із стандартної бібліотеки Java 8, для маніпуляцій датами та часом – java.time.LocalDate;
   7. Провести конкатинацію назв однотипних космічних апаратів, наприклад: Pioneer 10 и Pioneer11 🡪 Pioneer10ANDPioneer11.

# РЕЗУЛЬТАТИ РОБОТИ

## UML – діаграма класів



## Лістинг вихідного коду

Лістинг класу «FloydTriangle»

**package** Objects.Shapes.FloydTriangle;  
  
**import** Objects.Shapes.Shape;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** FloydTriangle **implements** Shape {  
  
 **private final static** String ***NEXT\_LINE*** = System.*lineSeparator*();  
  
 **private** Integer **lines**;  
 **private** TriangleTypes **type**;  
 **private** List<Integer> **lazyCaterersSequence**;  
 **private** List<Integer> **triangularNumbers**;  
 **private** List<Integer> **sequence**;  
  
 **public** FloydTriangle(**int** lines, TriangleTypes type){  
 **this**.**lines** = lines;  
 **this**.**type** = type;  
 **lazyCaterersSequence** = calculateLazyCaterersSequence();  
 **triangularNumbers** = calculateTriangularNumbers();  
 **sequence** = calculateSequence();  
 }  
  
 **public** Integer getLines(){  
 **return lines**;  
 }  
  
 @Override  
 **public** String stringRepresentation(){  
 **if**(**type** == TriangleTypes.***NORMAL***){  
 **return** normalRepresentation();  
 }**else if**(**type** == TriangleTypes.***INVERTED***){  
 **return** invertedRepresentation();  
 }**else if**(**type** == TriangleTypes.***SHUFFLED***){  
 **return** shuffledRepresentation();  
 }**else**{  
 **return ""**;  
 }  
 }  
  
 **public** List<Integer> getLazyCaterersSequence(){  
 **return new** ArrayList<>(**lazyCaterersSequence**);  
 }  
  
 **public** List<Integer> getTriangularNumbers(){  
 **return new** ArrayList<>(**triangularNumbers**);  
 }  
  
 **public** List<Integer> getSequence(){  
 **return new** ArrayList<>(**sequence**);  
 }  
  
  
 **private** String normalRepresentation(){  
 StringBuilder triangleString = **new** StringBuilder();  
 **for** (**int** i = 0; i < **lines**; i++) {  
 triangleString.append(createNormalLine(i)).append(***NEXT\_LINE***);  
 }  
 **return** triangleString.toString();  
 }  
  
 **private** String invertedRepresentation(){  
 StringBuilder triangleString = **new** StringBuilder();  
 **for** (**int** i = **lines** -1; i >= 0; i--) {  
 triangleString.append(createInvertedLine(i)).append(***NEXT\_LINE***);  
 }  
 **return** triangleString.toString();  
 }  
  
 **private** String shuffledRepresentation(){  
 StringBuilder triangleString = **new** StringBuilder();  
 **for** (**int** i = 0; i < **lines**; i++) {  
 **if**(i%2 == 0){  
 triangleString.append(createNormalLine(i)).append(***NEXT\_LINE***);  
 }**else**{  
 triangleString.append(createInvertedLine(i)).append(***NEXT\_LINE***);  
 }  
 }  
 **return** triangleString.toString();  
 }  
  
 **private** String createNormalLine(**int** index){  
 StringBuilder createdLine = **new** StringBuilder();  
 **for** (**int** i = **lazyCaterersSequence**.get(index); i <= **triangularNumbers**.get(index); i++) {  
 createdLine.append(i).append(**"\t"**);  
 }  
 **return** createdLine.toString();  
 }  
  
 **private** String createInvertedLine(**int** index){  
 StringBuilder createdLine = **new** StringBuilder();  
 **for** (**int** i = **triangularNumbers**.get(index); i >= **lazyCaterersSequence**.get(index); i--) {  
 createdLine.append(i).append(**"\t"**);  
 }  
 **return** createdLine.toString();  
 }  
  
 **private** List<Integer> calculateSequence(){  
  
 List<Integer> sequence = **new** ArrayList<>();  
 **for** (**int** i = 0; i < **lines**; i++) {  
 **for** (**int** j = **lazyCaterersSequence**.get(i); j <= **triangularNumbers**.get(i); j++) {  
 sequence.add(j);  
 }  
 }  
 **return** sequence;  
 }  
  
 **private** List<Integer> calculateLazyCaterersSequence(){  
 List<Integer> lazyCaterersSequence = **new** ArrayList<>();  
 **for** (**int** i = 0; i < **lines**; i++) {  
 lazyCaterersSequence.add(getCentralPolygonalNumber(i));  
 }  
 **return** lazyCaterersSequence;  
 }  
  
 **private** List<Integer> calculateTriangularNumbers(){  
 List<Integer> triangularNumbers = **new** ArrayList<>();  
 **for** (**int** i = 1; i <= **lines**; i++) {  
 triangularNumbers.add(*getTriangularNumber*(i));  
 }  
 **return** triangularNumbers;  
 }  
  
 **private int** getCentralPolygonalNumber(**int** row){  
 **return** (row\*row + row + 2) / 2;  
 }  
  
 **private static int** getTriangularNumber(**int** row){  
 **return** (row\*(row + 1)) / 2;  
 }  
  
}

Лістинг інтерфейсу «Shape»

**package** Objects.Shapes;  
  
**public interface** Shape {  
 String stringRepresentation();  
}

Лістинг класу «ShapePrinter»

**package** Objects.Shapes;  
  
**public class** ShapePrinter {  
  
 **private** Shape **shape**;  
  
 **public** ShapePrinter(Shape shape){  
 **this**.**shape** = shape;  
 }  
  
 **public void** printConsole(){  
 System.***out***.println(**shape**.stringRepresentation());  
 }  
}

Лістинг перерахування «TriangleTypes»

**package** Objects.Shapes.FloydTriangle;  
  
**public enum** TriangleTypes {  
 ***NORMAL***, ***INVERTED***, ***SHUFFLED***}

Лістинг класу «OutputStringFormer»

**package** Objects.TextOperations;  
  
**public class** OutputStringFormer {  
  
 **private static final** String ***NEXT\_LINE*** = System.*lineSeparator*();  
  
 **private final** StringBuilder **header** = **new** StringBuilder();  
 **private final** StringBuilder **body** = **new** StringBuilder();  
 **private final** StringBuilder **footer** = **new** StringBuilder();  
  
 **public void** addModification(String header, String body, String footer){  
 addHeader(header);  
 addBody(body);  
 addFooter(footer);  
 }  
  
 **public** String getHeader(){  
 **return header**.toString();  
 }  
  
 **public** String getBody(){  
 **return body**.toString();  
 }  
  
 **public** String getFooter(){  
 **return footer**.toString();  
 }  
  
 **public** String getFormedModifications(){  
 **return header**.toString() + **body**.toString() + **footer**.toString();  
 }  
  
 **private void** addHeader(String header){  
 **this**.**header**.append(header).append(***NEXT\_LINE***);  
 }  
  
 **private void** addBody(String body){  
 **this**.**body**.append(body);  
 }  
  
 **private void** addFooter(String footer){  
 **this**.**footer**.append(footer).append(***NEXT\_LINE***);  
 }  
}

Лістинг класу «FileReader»

**package** Objects.FileOperations.FileReader;  
  
**import** java.io.BufferedReader;  
**import** java.io.File;  
**import** java.io.IOException;  
  
**public class** FileReader {  
  
 **private final** File **file**;  
  
 **private static final** String ***NEXT\_LINE*** = System.*lineSeparator*();  
  
 **public** FileReader(File file){  
 **this**.**file** = file;  
 }  
  
 **public** String read(){  
  
 StringBuilder allTextFromFile = **new** StringBuilder();  
  
 **try**(BufferedReader br = **new** BufferedReader(**new** java.io.FileReader(**file**))){  
 String line;  
 **while**((line = br.readLine()) != **null**){  
 allTextFromFile.append(line).append(***NEXT\_LINE***);  
 }  
 } **catch** (IOException exception) {  
 System.***out***.println(**"PROBLEM WITH READING FROM FILE"**);  
 }  
  
 **return** allTextFromFile.toString();  
 }  
  
}

Лістинг класу «FileWriter»

**package** Objects.FileOperations.FileWriter;  
  
**import** java.io.BufferedWriter;  
**import** java.io.File;  
**import** java.io.IOException;  
  
**public class** FileWriter {  
  
 **private final** File **file**;  
  
 **public** FileWriter(File file){  
 **this**.**file** = file;  
 }  
  
 **public void** write(String text){  
 **try**(BufferedWriter bw = **new** BufferedWriter(**new** java.io.FileWriter(**file**))){  
 bw.write(text);  
 }**catch** (IOException exception){  
 System.***out***.println(**"PROBLEM WITH WRITING TO FILE"**);  
 }  
 }  
}

## Результати роботи

## 

Рис. 1 – Результат прямого друку в файл

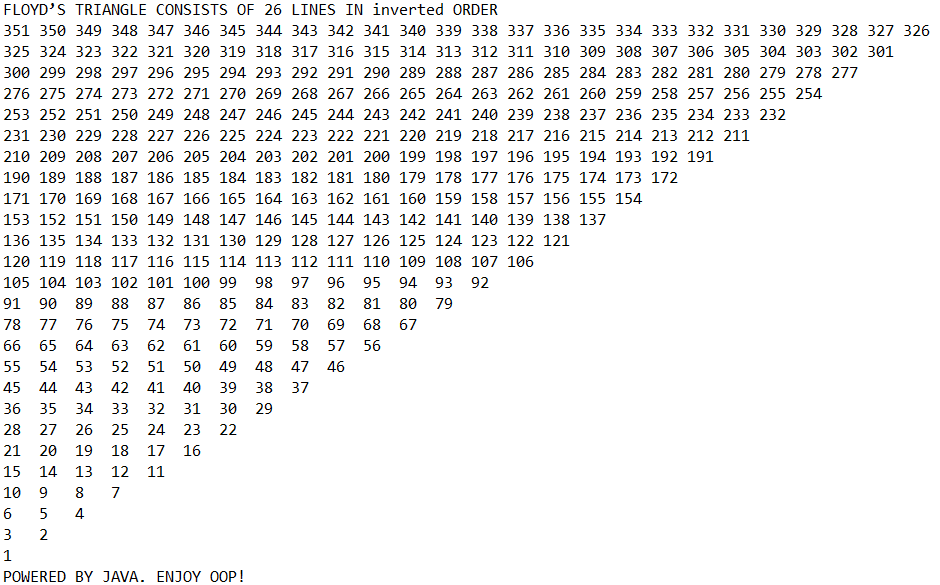


Рис. 2 – Результат оберненого друку в файл

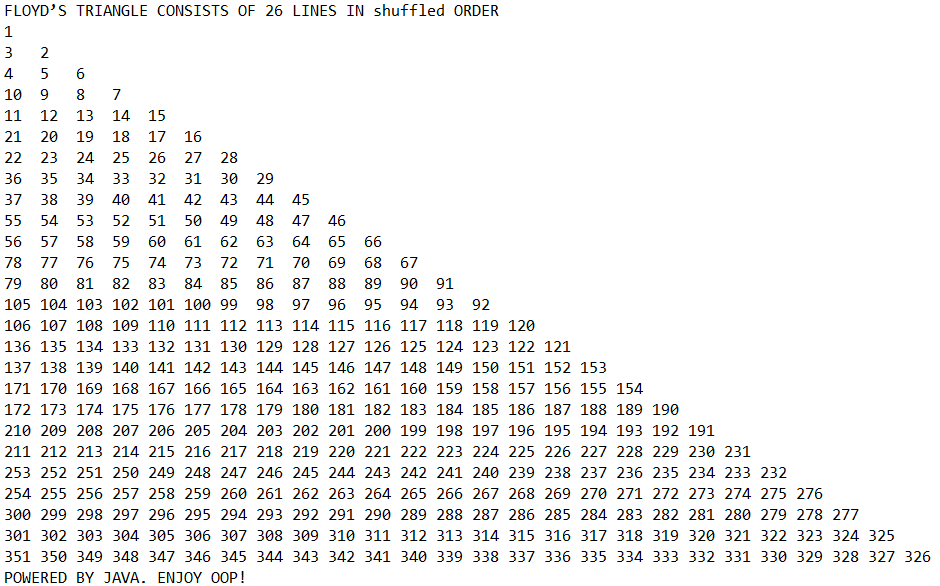


Рис. 3 – Результат перемішаного друку в файл