Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра “Електронних обчислювальних машин”



**з дисципліни**

**Кросплатформні засоби програмування**

**Звіт з Лабораторної роботи № 5**

**ФАЙЛИ У JAVA**

**Виконав:**

КІ-307

Герега Р. О.

**Перевірив:**

Іванов Ю.С

**2023**

**Мета роботи:** оволодіти навиками використання засобів мови Java для роботи з потоками і файлами.

**Теоретичний вступ**

Бібліотека класів мови Java має більше 60 класів для роботи з потоками. Потаками у мові Java називаються об’єкти з якими можна здійснювати обмін даними. Цими об’єктами найчастіше є файли, проте ними можуть бути стандартні пристрої вводу/виводу, блоки пам’яті і мережеві підключення тощо. Класи по роботі з потоками об’єднані у кілька ієрархій, що призначені для роботи з різними видами даних, або забезпечувати додаткову корисну функціональність, наприклад, підтримку ZIP архівів. Класи, що спадкуються від абстрактних класів InputStream і OutputStream призначені для здійснення байтового обміну інформацією. Підтримка мовою Java одиниць Unicode, де кожна одиниця має кілька байт, зумовлює необхідність у іншій ієрархії класів, що спадкується від абстрактних класів Reader і Writer. Ці класи дозволяють виконувати операції читання/запису не байтних даних, а двобайтних одиниць Unicode. Принцип здійснення читання/запису даних нічим не відрізняється від такого принципу у інших мовах програмування. Все починається з створення потоку на запис або читання після чого викликаються методи, що здійснюють обмін інформацією. Після завершення обміну даними потоки необхідно закрити щоб звільнити ресурси.

**Принципи роботи з файловими потоками**

Для створення файлових потоків і роботи з ними у Java є 2 класи, що успадковані від InputStream і OutputStream це - FileInputStream і FileOutputStream. Як і їх суперкласи вони мають методи лише для байтового небуферизованого блокуючого читання/запису даних та керуванням потоками. На відміну від, наприклад, мови програмування С, де для виконання усіх можливих операцій з файлами необхідно мати один вказівник на FILE у мові Java реалізовано інший набагато складніший і гнучкіший підхід, який дозволяє формувати такі властивості потоку, які найкраще відповідають потребам рішення конкретної задачі. Так у Java розділено окремі функціональні можливості потоків на різні класи. Компонуючи ці класи між собою досягається необхідна кінцева функціональність потоку. Так одні класи, як FileInputStream,

забезпечують елементарний доступ до файлів, інші, як PrintWriter, надають

додаткової функціональності по високорівневій обробці даних, що пишуться у файл. Ще інші, наприклад, BufferedInputStream забезпечують буферизацію. Таким чином, наприклад, щоб отримати буферизований файловий потік для читання інформації у форматі примітивних типів (char, int, double,…) слід створити потік з одночасним сумісним використанням функціональності класів FileInputStream, BufferedInputStream і DataInputStream. Для цього слід здійснити наступний виклик: DataInputStream din = new DataInputStream( new BufferedInputStream( new FileInputStream))); Класи типу BufferedInputStream, DataInputStream, PushbackInputStream (дозволяє читати з потоку дані і повертати їх назад у потік) успадковані від класу FilterInputStream. Вони виступають так званими фільтрами, що своїм комбінуванням забезпечують додаткову лише необхідну функціональність при читанні даних з файлу. Аналогічний підхід застосовано і при реалізації класів для обробки текстових даних, що успадковані від Reader і Writer.

**Варіант 6**



**Завдання:**

1. Створити клас, що реалізує методи читання/запису у текстовому і двійковому форматах результатів роботи класу, що розроблений у лабораторній роботі №4.
2. Написати програму для тестування коректності роботи розробленого класу. Для розробленої програми згенерувати документацію.
3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.
4. Скласти звіт про виконану роботу з приведенням тексту програми, результату їївиконання та фрагменту згенерованої документації та завантажити його у ВНС.
5. Дати відповідь на контрольні запитання.

**Код**

/\*\*

\* lab 4 package **ExpressionCalculator.java**

\*/

package KI.Hereha.Lab5;

import java.io.FileWriter;

import java.io.IOException;

/\*\*

\* Клас <code>ExpressionCalculator</code> для обчислення виразу tan(x)/sin(2x)

\* @author Rostyslav Hereha

\* @version 1.0

\*/

public class ExpressionCalculator {

// функція для обчислення виразу

public static double calculateExpression(double x) {

double tanX = Math.tan(x);

double sin2X = Math.sin(2 \* x);

if (sin2X == 0) {

// створимо виключення вищого рівня з поясненням причини

throw new ArithmeticException("Ділення на нуль: sin(2x) = 0");

}

return tanX / sin2X;

}

public static void writeResultToFile(double result) throws IOException {

try (FileWriter writer = new FileWriter("result.txt")) {

writer.write("Результат обчислень: y = " + result);

}

}

}

package KI.Hereha.Lab5;

import java.io.IOException;

import java.util.InputMismatchException;

import java.util.Scanner;

import KI.Hereha.Lab5.ExpressionCalculator;

import KI.Hereha.Lab5.ResultFileManager;

public class ExpressionCalculatorDriver {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

try (Scanner scanner = new Scanner(System.in)) {

System.out.println("Введіть x: ");

double x = scanner.nextInt();

double result = ExpressionCalculator.calculateExpression(x); // Приклад виклику з обчисленням для x=45 градусів

System.out.println("Результат обчислень: y = " + result);

ExpressionCalculator.writeResultToFile(result);

ResultFileManager.writeResultToTextFile(result, "result.txt");

// Запис результату у двійковий файл

ResultFileManager.writeResultToBinaryFile(result, "result.dat");

// Читання результату з текстового файлу

double textResult = ResultFileManager.readResultFromTextFile("result.txt");

System.out.println("Прочитаний результат з текстового файлу: " + textResult);

// Читання результату з двійкового файлу

double binaryResult = ResultFileManager.readResultFromBinaryFile("result.dat");

System.out.println("Прочитаний результат з двійкового файлу: " + binaryResult);

} catch (ArithmeticException e) {

// Блок перехоплює помилки обчислень виразу

System.err.println("Помилка обчислень: " + e.getMessage());

} catch (IOException e) {

// Блок перехоплює помилки запису в файл

System.err.println("Помилка запису до файлу: " + e.getMessage());

} catch (InputMismatchException e) {

// Блок перехоплює помилки з неправильним вводом х

System.err.println("Помилка неправильний тип введених даних: " + e.getMessage());

} catch (Exception e) {

// Блок перехоплює інші помилки

System.err.println("Помилка: " + e);

}

}

}

package KI.Hereha.Lab5;

import java.io.\*;

public class ResultFileManager {

/\*\*

\* Запис результату у текстовий файл.

\* @param result Результат обчислень

\* @param fileName Ім'я файлу

\* @throws IOException Виникає у випадку помилки запису у файл

\*/

public static void writeResultToTextFile(double result, String fileName) throws IOException {

try (FileWriter writer = new FileWriter(fileName)) {

writer.write("Результат обчислень: y = " + result);

}

}

/\*\*

\* Читання результату з текстового файлу.

\* @param fileName Ім'я файлу

\* @return Результат обчислень

\* @throws IOException Виникає у випадку помилки читання з файлу

\*/

public static double readResultFromTextFile(String fileName) throws IOException {

try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {

String line = reader.readLine();

// Видаліть всі символи, що не є цифрами або крапкою

line = line.replaceAll("[^\\d.]", "");

return Double.parseDouble(line);

}

}

/\*\*

\* Запис результату у двійковий файл.

\* @param result Результат обчислень

\* @param fileName Ім'я файлу

\* @throws IOException Виникає у випадку помилки запису у файл

\*/

public static void writeResultToBinaryFile(double result, String fileName) throws IOException {

try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(fileName))) {

outputStream.writeDouble(result);

}

}

/\*\*

\* Читання результату з двійкового файлу.

\* @param fileName Ім'я файлу

\* @return Результат обчислень

\* @throws IOException Виникає у випадку помилки читання з файлу

\*/

public static double readResultFromBinaryFile(String fileName) throws IOException {

try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(fileName))) {

return inputStream.readDouble();

}

}

}

**Результат:**







**Короткий висновок**

Під час виконання даної лабораторної роботи я оволодів навиками використання засобів мови Java для роботи з потоками і файлами..