Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Звіт

З лабораторної роботи №6

З дисципліни «Кросплатформенні засоби програмування»

На тему: «Файли»

Виконав:
ст.гр. КІ-36
Литовко С.Г
Прийняв:
Іванов Ю.С.

Львів-2022

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

ТЕОРЕТИЧНІ ВІДОМОСТІ

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 і 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 B*ufferedInputStream, 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*.

ЗАВДАННЯ

1. Створити клас, що реалізує методи читання/запису у текстовому і двійковому форматах результатів роботи класу, що розроблений у лабораторній роботі №5. Написати програму для тестування коректності роботи розробленого класу.
2. Для розробленої програми згенерувати документацію.
3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагмент згенерованої документації.
4. Дати відповідь на контрольні запитання.

Варіант

6. y=tg(x)/sin(2x)

Код програми:

Main.java

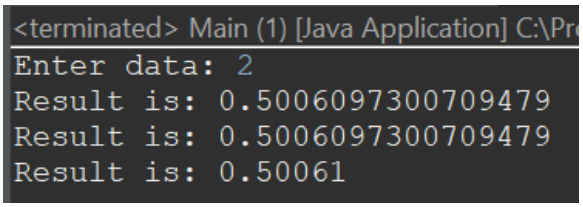
```
package KI36.Lytovko.Lab06;

import java.util.Scanner;
import java.io.*;

/**
 * Class <code>EquationsApp</code> Implements driver for Equations class
```

```
 * @version 1.0
 */
public class Main {
    /**
     * @param args
     */
    public static void main(String[] args) throws FileNotFoundException, IOException
    {
        CalcWFio obj = new CalcWFio();
        Scanner s = new Scanner(System.in);
        System.out.print("Enter data: ");
        double data = s.nextDouble();
        obj.calculate(data);
        System.out.println("Result is: " + obj.getResult());
        obj.writeResTxt("lab6.txt");
        obj.writeResBin("lab6.bin");

        obj.readResBin("lab6.bin");
        System.out.println("Result is: " + obj.getResult());
        obj.readResTxt("lab6.txt");
        System.out.println("Result is: " + obj.getResult());
    }
}
class CalcWFio
{
    public void writeResTxt(String fName) throws FileNotFoundException
    {
        PrintWriter f = new PrintWriter(fName);
        f.printf("%f ",result);
        f.close();
    }

    public void readResTxt(String fName)
    {
        try
        {
            File f = new File (fName);
            if (f.exists())
            {
                Scanner s = new Scanner(f);
                result = s.nextDouble();
                s.close();
            }
            else
                throw new FileNotFoundException("File " + fName + "not found");
        }
        catch (FileNotFoundException ex)
        {
            System.out.print(ex.getMessage());
        }
    }

    public void writeResBin(String fName) throws FileNotFoundException, IOException
    {
        DataOutputStream f = new DataOutputStream(new FileOutputStream(fName));
```

```
        f.writeDouble(result);
        f.close();
    }

    public void readResBin(String fName) throws FileNotFoundException, IOException
    {
        DataInputStream f = new DataInputStream(new FileInputStream(fName));
        result = f.readDouble();
        f.close();
    }

    public void calculate(double x)
    {
        Equations eq = new Equations();
        result = eq.calculate(x);
    }

    public double getResult()
    {
        return result;
    }
    private double result;
}

/**
 * Class <code>CalcException</code> more precises ArithmeticException
 * @version 1.0
 */
class CalcException extends ArithmeticException
{
    public CalcException(){}

    public CalcException(String cause)
    {
        super(cause);
    }
}


class Equations
{
    public double calculate(double x) throws CalcException
    {
        double y, rad;
        rad = x * Math.PI / 180.0;
        try
        {
            y = Math.tan(rad) / Math.sin(2 * rad);
            if (x==90 || x== -90 || x==0 || x==-180 || x==180 || 2*x == 90 || 2*x ==
-90 || 2*x == 180 || 2*x == -180 )
                throw new ArithmeticException();
        }
        catch (ArithmeticException ex)
        {
```

```
            if (x==90 || x== -90 || x==-180 || x==180 || 2*x == 90 || 2*x == -90 ||
2*x == 180 || 2*x == -180 )
                throw new CalcException("Exception reason: Illegal value of X for
tangent calculation");

            else if (x==0)
                throw new CalcException("Exception reason: X = 0");
            else
                throw new CalcException("Unknown reason of the exception during
exception calculation");

        }
        return y;
    }
}
```

Результат програми:



Висновок: Зробивши цю лабораторну роботу, я оволоділа навиками використання засобів мови Java для роботи з потоками і файлами.