

Міністерство освіти і науки України  
Національний університет «Львівська політехніка»



### **Звіт**

до лабораторної роботи №6

З дисципліни: «Кросплатформенні засоби програмування»

На тему: «Файли»

Виконав:

Студент групи КІ-34

Романів В. А.

Прийняв:

Іванов Ю. С.

Львів 2022

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

### *Виконання роботи*

#### **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.

*Завдання:*

$$18. y = \text{tg}(x) / (\sin(4x) - 2\cos(x))$$

*Код програми:*

Клас FioApp

```
import java.io.*;
import java.util.*;

public class FioApp {
    public static void main(String[] args) throws FileNotFoundException,
IOException {
        Scanner s = new Scanner(System.in);
        CalcWfio obj = new CalcWfio();
        System.out.print("Enter x -> ");
        double y = obj.calculate(s.nextInt());
        System.out.println("Result of example is: " + y);
        obj.writeResTxt("textRes.txt");
        obj.writeResBin("BinRes.bin");
        obj.readResBin("BinRes.bin");
        System.out.println("Result(bin) is: " + y);
        obj.readResTxt("textRes.txt");
        System.out.println("Result(text) is: " + y);
        s.close();
    }
}
```

## Клас CalcWfio

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

class CalcWfio {
    public void writeResTxt(String fName) throws FileNotFoundException {
        PrintWriter f = new PrintWriter(fName);
        f.printf("%.10f ", y);
        f.close();
    }
    public void readResTxt(String fName) {
        try {
            File f = new File(fName);
            if (f.exists()) {
                Scanner s = new Scanner(f);
                y = 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(y);
        f.close();
    }
    public void readResBin(String fName) throws FileNotFoundException,
    IOException {
        DataInputStream f = new DataInputStream(new FileInputStream(fName));
        y = f.readDouble();
        f.close();
    }
}
```

```

private double y;
public double calculate(int x) throws CalcException {
    double rad;
    rad = x * Math.PI / 180.0;
    try{
        y = Math.tan(x) / (Math.sin(4*rad) - 2 * Math.cos(rad));
        if(y == Double.NaN || y == Double.NEGATIVE_INFINITY ||
            y == Double.POSITIVE_INFINITY || x == 90 || x == -90){
            throw new ArithmeticException();
        }
    } catch (ArithmeticException ex) {
        // створимо виключення вищого рівня з поясненням причини
        // виникнення помилки
        if (rad==Math.PI/2.0 || rad==Math.PI/2.0)
            throw new CalcException("Exception reason: Illegal value of " +
                "X for tangent calculation");
        else
            throw new CalcException("Unknown reason of the exception " +
                "during exception calculation");
    }
    return y;
}
public double getResult() {
    return y;
}
}

```

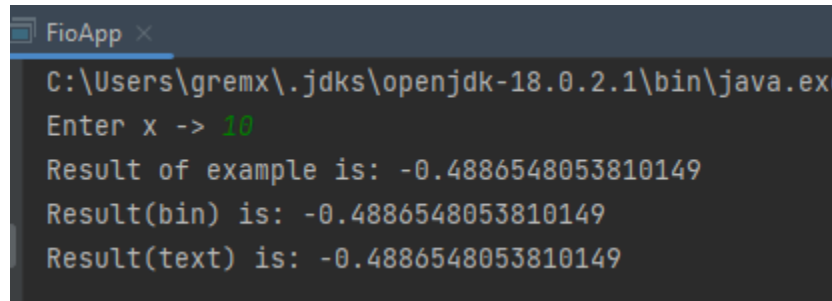
## Клас CalcException

```

class CalcException extends ArithmeticException {
    public CalcException() {
    }
    public CalcException(String cause) {
        super(cause);
    }
}

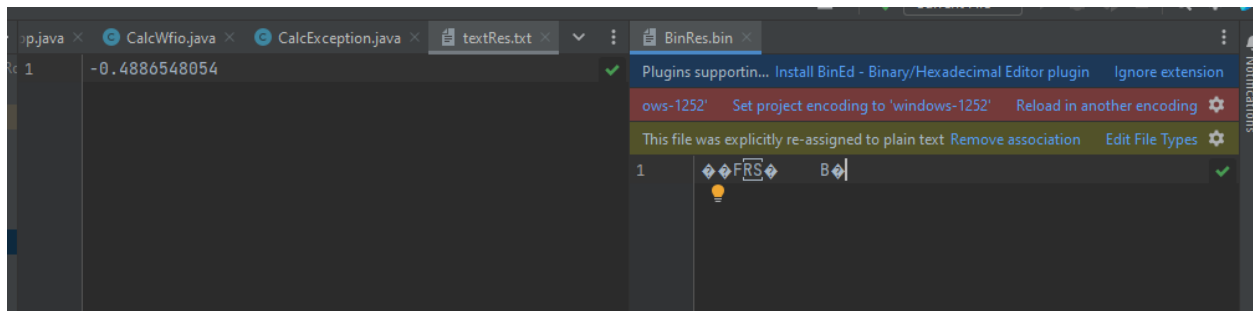
```

Результат роботи програми:

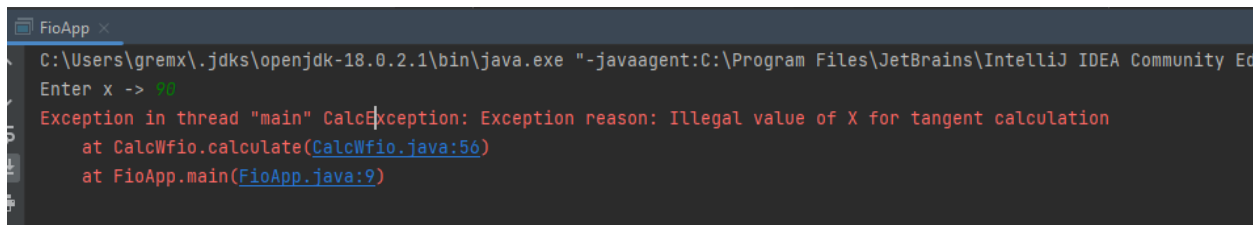


```
C:\Users\grems\jdk\openjdk-18.0.2.1\bin\java.exe
Enter x -> 10
Result of example is: -0.4886548053810149
Result(bin) is: -0.4886548053810149
Result(text) is: -0.4886548053810149
```

Створено текстовий та бінарний файли:



Успішно виводить повідомлення про помилку:



### *Відповіді на КЗ*

1. Бібліотека класів мови Java має більше 60 класів для роботи з потоками. Потоками 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.

2. Для читання текстових потоків найкраще підходить клас `Scanner`. На відміну від `InputStreamReader` і `FileReader`, що дозволяють лише читати текст, він має велику кількість методів, які здатні читати як рядки, так і окремі примітивні типи з подальшим їх перекодуванням до цих типів, робити шаблонний аналіз текстового потоку, здатний працювати без потоку даних та ще багато іншого.

3. Приклад читання даних за допомогою класу `Scanner` з стандартного потоку вводу: `Scanner sc = new Scanner(System.in); int i = sc.nextInt();` Приклад читання даних за допомогою класу `Scanner` з текстового файлу: `Scanner sc = new Scanner(new File("myNumbers")); while (sc.hasNextLong()) { long aLong = sc.nextLong(); }`

4. Для буферизованого запису у текстовий потік найкраще використовувати клас `PrintWriter`. Цей клас має методи для виводу рядків і чисел у текстовому форматі: `print`, `println`, `printf`, - принцип роботи яких співпадає з аналогічними методами `System.out`.

5. `PrintWriter` – надає додаткової функціональності по високорівневій обробці даних, що пишуться у файл.

6. Читання двійкових даних примітивних типів з потоків здійснюється за допомогою класів, що реалізують інтерфейс `DataInput`, наприклад класом `DataInputStream`. Інтерфейс `DataInput` визначає такі методи для читання двійкових даних:

- `readByte;`
- `readInt;`
- `readShort;`
- `readLong;`
- `readFloat;`
- `readDouble;`

- readChar;
- readBoolean;
- readUTF.

Запис двійкових даних примітивних типів у потоки здійснюється за допомогою класів, що реалізують інтерфейс `DataOutput`, наприклад класом `DataOutputStream`. Інтерфейс `DataOutput` визначає такі методи для запису двійкових даних:

- writeByte;
- writeInt;
- writeShort;
- writeLong;
- writeFloat;
- writeDouble;
- writeChar;
- writeChars;
- writeBoolean;
- writeUTF.

7. `DataInputStream` – читання двійкового файлу. `DataOutputStream` – запис двійкового файлу.

8. – 9. Керування файлами з можливістю довільного доступу до них здійснюється за допомогою класу `RandomAccessFile`. Відкривання файлу в режимі запису і читання/запису здійснюється за допомогою конструктора, що приймає 2 параметри – посилання на файл (`File file`) або його адресу (`String name`) та режим відкривання файлу (`String mode`).

10. `DataOutput` інтерфейс, який реалізований класом `DataOutputStream`.

**Висновок:** оволодів навиками використання засобів мови Java для роботи з потоками і файлами