

Міністерство освіти і науки України Національний
університет «Львівська політехніка»



Звіт

до лабораторної роботи №6

З дисципліни: «Кросплатформенні засоби програмування»

На тему: «Файли»

Варіант 15

Виконав:
Ст. групи КІ-34
Ольховик О.С.

Прийняв:
к.т.н., доцент
Іванов Ю.С.

Львів 2022

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

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 ЗАВДАННЯ

$$15. \tilde{y} = \cos(\tilde{2x}) / \tilde{\text{ctg}}(\tilde{3x-1})$$

CalcWFioApp.java

```
import java.io.IOException;
import java.util.Scanner;

import KI34.Olkhovyk.Lab6.CalcWFio;

public class CalcWFioApp {

    public static void main(String[] args) throws 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("textRes.txt");
        obj.writeResBin("BinRes.bin");

        obj.readResBin("BinRes.bin");
        System.out.println("Result(Bin) is: " + obj.getResult());
        obj.readResTxt("textRes.txt");
        System.out.println("Result(Txt) is: " + obj.getResult());
        s.close();
    }
}
```

CalcException.java

```
package KI34.Olkhovyk.Lab6;

public class CalcException extends ArithmeticException {
    public CalcException() {
    }
}
```

```

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

```

CalcWFio.java

```

package KI34.Olkhovyk.Lab6;

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

public 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) {

```

```

        result = new Equations().calculate((int)x);
    }

    public double getResult() {
        return result;
    }

    private double result;
}

```

Equantions.java

```

package KI34.Olkhovyk.Lab6;

public class Equations {
    public double calculate(int x) throws CalcException {
        double y, rad;
        rad = x * Math.PI / 180.0;
        try {
            y = Math.cos(2 * rad) / Math.tan((3 * rad) - 1);
            if (y == Double.NaN || y == Double.NEGATIVE_INFINITY || y ==
Double.POSITIVE_INFINITY || x == 90 || x == -90)
                throw new ArithmeticException();

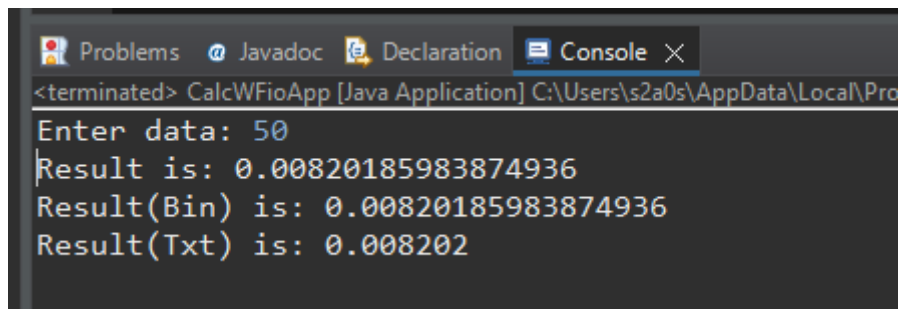
        } catch (ArithmeticException ex) {

            if (rad == 0 || rad == Math.PI || rad == 2 * Math.PI)
                throw new CalcException("Exception reason: Illegal value of X for
cotangent calculation");
            else if (rad == Math.PI / 2.0 || rad == -Math.PI / 2.0)
                throw new CalcException("Exception reason: Divide by zero");
            else
                throw new CalcException("Unknown reason of the exception during
exception calculation");
        }
        return y;
    }
}

```

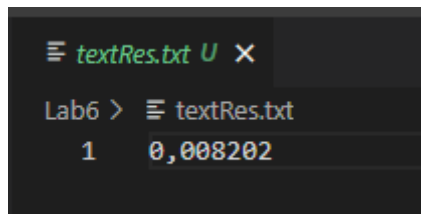
Результат виконання програми

- Консоль



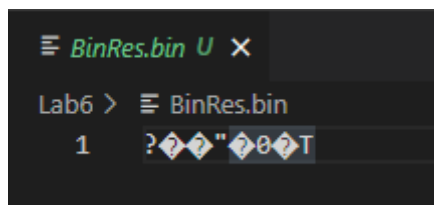
```
<terminated> CalcWFioApp [Java Application] C:\Users\s2a0s\AppData\Local\Pro
Enter data: 50
Result is: 0.00820185983874936
Result(Bin) is: 0.00820185983874936
Result(Txt) is: 0.008202
```

- Txt-файл з записаною інформацією:



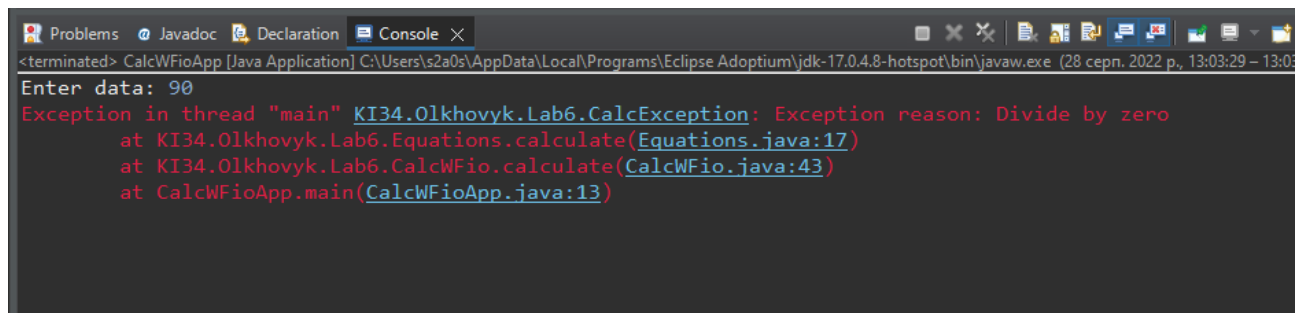
```
textRes.txt U X
Lab6 > textRes.txt
1 0.008202
```

- bin-файл з записаною інформацією:



```
BinRes.bin U X
Lab6 > BinRes.bin
1 0.008202
```

- Помилка:



```
Problems Javadoc Declaration Console X
<terminated> CalcWFioApp [Java Application] C:\Users\s2a0s\AppData\Local\Programs\Eclipse Adoptium\jdk-17.0.4.8-hotspot\bin\javaw.exe (28 серп. 2022 р., 13:03:29 - 13:03:30)
Enter data: 90
Exception in thread "main" KI34.Olkhovik.Lab6.CalcException: Exception reason: Divide by zero
    at KI34.Olkhovik.Lab6.Equations.calculate(Equations.java:17)
    at KI34.Olkhovik.Lab6.CalcWFio.calculate(CalcWFio.java:43)
    at CalcWFioApp.main(CalcWFioApp.java:13)
```

Відповіді на контрольні запитання:

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: я оволодів навиками використання засобів мови Java для роботи з потоками і файлами.