

Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра «Електронних обчислювальних машин»



Звіт
з лабораторної роботи № 5
з дисципліни: «Кросплатформенні засоби програмування»
на тему: «“ФАЙЛИ У JAVA”»

Виконав:

студент групи *KI-306*

Олесько Б. А.

Прийняв:

доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2023

Мета: 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. 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.

Варіант 11 $y = \text{ctg}(x)/\text{tg}(x)$

Код програми

```
package KI306.Olesko.lab5;
import java.io.*;
import java.util.*;
public class App {
    public static void main(String[] args) {
        Calco equation = new Calco();
        FileUtils fileUtils = new FileUtils();
        double result;
        double fileResult;
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter X: ");
        try {
            int x = scanner.nextInt();
            result = equation.calculate(x);
            System.out.println("Result is " + result);
            // Write the result to a text file
            fileUtils.writeResTxt("E:\\REPOS\\CPPT_LABS\\LAB_5\\textRes.txt", result);
            // Write the result to a binary file
            fileUtils.writeResBin("E:\\REPOS\\CPPT_LABS\\LAB_5\\binRes.bin", result);
            // Read the result from the binary file
            fileResult = fileUtils.readResBin("E:\\REPOS\\CPPT_LABS\\LAB_5\\binRes.bin");
            System.out.println("Result from binary file is: " + fileResult);
            // Read the result from the text file
            fileResult = fileUtils.readResTxt("E:\\REPOS\\CPPT_LABS\\LAB_5\\textRes.txt");
            System.out.println("Result from txt file is: " + fileResult);
        } catch (CalcException e) {
```

```

        System.out.println(e.getMessage());
    }
}

```

```

package KI306.Olesko.lab5;
public class CalcException extends ArithmeticException {
    /**
     * Constructs a new `CalcException` with no detail message.
     */
    public CalcException() {
    }
    /**
     * Constructs a new `CalcException` with the specified detail message.
     *
     * @param cause The detail message describing the reason for the exception.
     */
    public CalcException(String cause) {
        super(cause);
    }
}

```

```

package KI306.Olesko.lab5;
public class Calco {
    /**
     * Calculates the  $y = \frac{\sin(x)}{\cos(x)}$  expression for a given angle in degrees.
     * @param x The angle in degrees for which the expression is calculated.
     * @return The result of the expression calculation.
     * @throws CalcException If an error occurs during the calculation, this exception is thrown
     */
    public double calculate(int x) throws CalcException {
        double y, rad;
        int i=1;
        if(x>360) {
            for(i=1 ;x>i*360;i++) { }
            x=x-((i-1)*360);
        }
        rad = Math.toRadians(x);
        try {
            double tanValue = Math.tan(rad);
            y = 1.0 / (tanValue * tanValue);
            if (Double.isNaN(y) || y == Double.NEGATIVE_INFINITY || y == Double.POSITIVE_INFINITY || rad==Math.PI
||x==0 ||x==360||x==90||x==270) {
                throw new ArithmeticException();
            }
        }
        catch (ArithmeticException e) {
            // create a higher-level exception with an explanation of the reason for the error
            if (rad==Math.PI/2.0 || rad==Math.PI/2.0){
                throw new CalcException("Exception reason: Illegal value of X = 90");
            }
            else if (rad==Math.PI || rad==Math.PI) {
                throw new CalcException("Exception reason: X = 180");
            }
            else if (rad==Math.PI * 3 / 2 || rad==Math.PI * 3 / 2) {
                throw new CalcException("Exception reason: X = 270");
            }
            else if (rad==Math.PI * 2 || rad==Math.PI * 2) {
                throw new CalcException("Exception reason: X = 360");
            }
            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;
    }
}

```

```

package KI306.Olesko.lab5;
import java.io.*;
import java.util.Locale;
import java.util.Scanner;
public class FileUtils {
    /**
     * Writes a double value to a text file.
     * @param fName The name of the file to write to.
     * @param result The double value to write to the file.
     */
    public void writeResTxt(String fName, double result) {
        try {
            PrintWriter fileWriter = new PrintWriter(fName);
            fileWriter.print(result);
            fileWriter.close();
        } catch (FileNotFoundException e) {
            System.out.println("Exception reason: Possibly incorrect file path " + e);
        }
    }
    /**
     * Reads a double value from a text file.
     * @param fName The name of the file to read from.
     * @return The double value read from the file, or 0 if the file does not contain a valid double value.
     */
    public double readResTxt(String fName) {
        double result = 0;
        try {
            File file = new File(fName);
            Scanner scanner = new Scanner(file);
            scanner.useLocale(Locale.US);
            if (scanner.hasNextDouble()) {
                result = scanner.nextDouble();
                scanner.close();
            } else {
                System.err.println("The file does not contain a double value.");
            }
        } catch (FileNotFoundException e) {
            System.out.println("Exception reason: Possibly incorrect file path");
        }
        return result;
    }
    /**
     * Writes a double value to a binary file.
     * @param fName The name of the file to write to.
     * @param result The double value to write to the file.
     */
    public void writeResBin(String fName, double result) {
        try {
            DataOutputStream fileOutputStream = new DataOutputStream(new FileOutputStream(fName));
            fileOutputStream.writeDouble(result);
            fileOutputStream.close();
        } catch (FileNotFoundException e) {
            System.out.println("Exception reason: Possibly incorrect file path " + e);
        } catch (IOException e) {
            System.out.print(e.getMessage());
        }
    }
    /**
     * Reads a double value from a binary file.
     * @param fName The name of the file to read from.
     * @return The double value read from the file, or 0 if the file does not contain a valid double value.
     */
    public double readResBin(String fName) {
        double result = 0;
        try {
            DataInputStream fileInputStream = new DataInputStream(new FileInputStream(fName));
            result = fileInputStream.readDouble();
        }
    }
}

```

```

        fileInputStream.close();
    } catch (FileNotFoundException e) {
        System.out.println("Exception reason: Possibly incorrect file path " + e);
    } catch (IOException e) {
        System.out.print(e.getMessage());
    }
    return result;
}
}

```

Результат програми

```

Enter X: 1
Result is 3282.139703653887
Result from binary file is: 3282.139703653887
Result from txt file is: 3282.139703653887

```

Документація

Package `KI306.Olesko.lab5`

Class App

`java.lang.Object`
`KI306.Olesko.lab5.App`

`public class App`
`extends Object`

Constructor Summary

Constructors

Constructor	Description
<code>App()</code>	

Method Summary

All Methods

Static Methods

Concrete Methods

Modifier and Type	Method	Description
<code>static void</code>	<code>main(String[] args)</code>	

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Відповіді на контроль завдання

1. Робота з файловою системою в Java включає в себе використання класів `File` та `Path` для роботи з файлами та каталогами. Для читання та запису даних у файли використовуються класи `FileInputStream`, `FileOutputStream` для роботи з байтами, а класи `FileReader` та `FileWriter` для текстових даних.
2. Клас `Scanner` в Java використовується для зчитування введених користувачем даних з клавіатури або з інших джерел, таких як файли. Він дозволяє парсити та обробляти введені дані.
3. Приклад використання класу `Scanner` для зчитування числа з клавіатури:

```
```java  
Scanner scanner = new Scanner(System.in);
System.out.print("Enter a number: ");
int number = scanner.nextInt();
```
```

4. Для запису у текстовий потік використовується клас `PrintWriter`.
5. Клас `PrintWriter` в Java використовується для запису текстових даних у потік. Він дозволяє виводити рядки та інші типи даних у текстовий файл.
6. Для читання та запису двійкових даних використовуються класи `DataInputStream` та `DataOutputStream`. Вони дозволяють читати та записувати дані у бінарному форматі.
7. Класи `DataInputStream` і `DataOutputStream` використовуються для читання та запису примітивних типів даних у бінарному форматі. Вони допомагають зберігати дані у компактному бінарному вигляді.
8. Для здійснення довільного доступу до файлів використовується клас `RandomAccessFile`. Він дозволяє читати та записувати дані у файлі на конкретній позиції.
9. Клас `RandomAccessFile` використовується для довільного доступу до файлів, включаючи можливість читання та запису даних на конкретні позиції у файлі.
10. Інтерфейс `DataOutput` має методи для запису примітивних даних у бінарний потік. Клас `DataOutputStream` реалізує цей інтерфейс та дозволяє записувати дані у бінарний файл.

Висновок

На даній лабораторній роботі я зрозумів, як працює робота з файловою системою в мові програмування Java, включаючи роботу з файлами та каталогами. Набув знань про клас `Scanner`, який дозволяє зчитувати введені дані з клавіатури та інших джерел.