Міністерство освіти та науки України

Національний університет «Львівська політехніка»

Кафедра «Електронних обчислювальних машин»



**ЗВІТ**

**з лабораторної роботи № 6**

**з дисципліни:** «Кросплатформенні засоби програмування»

**на тему:** «ФАЙЛИ»

Виконав: ст. гр. КІ-35

Гоцій М.М.

Прийняв:

доцент кафедри ЕОМ

Іванов Ю.С

Львів-2022

**Мета роботи:** 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 11:**



**Текст програми:**

|  |
| --- |
| ***Лістинг CalcException***  package KI35.Hotsii.Lab6;  */\*\*  \* Class <code>CalcException</code> more precises ArithmeticException  \** ***@author*** *Mykyta Hotsii  \** ***@version*** *1.0  \*/* public class CalcException extends ArithmeticException {  public CalcException(String message) {  super(message);  } }  ***Лістинг AccountFio***  package KI35.Hotsii.Lab6;  import java.io.\*;  */\*\*  \* Class <code>AccountantFio</code> implements method for tg(x)/ctg(x) expression  \* calculation  \** ***@author*** *Mykyta Hotsii  \** ***@version*** *1.0  \*/* public class AccountantFio {  private double rad;  private double result;  private String BINARY\_FILE\_PATH;  private String TXT\_FILE\_PATH;   */\*\*  \* Method writes the result to a binary file  \** ***@throws*** *IOException  \*/* public void writeResBin() throws IOException{  DataOutputStream output = new DataOutputStream(new FileOutputStream(BINARY\_FILE\_PATH));  output.writeDouble(result);  output.close();  }   */\*\*  \* Method writes the result to a txt file  \** ***@throws*** *IOException  \*/* public void writeResTxt() {  try (PrintWriter writer = new PrintWriter(new File(TXT\_FILE\_PATH))) {  writer.println(result);  } catch (FileNotFoundException e) {  e.printStackTrace();  }  }   */\*\*  \* Method reads the result from a binary file  \** ***@throws*** *IOException  \*/* public void readResBin() throws IOException {  DataInputStream input = new DataInputStream(new FileInputStream(BINARY\_FILE\_PATH));  result = input.readDouble();  input.close();  }   */\*\*  \* Method reads the result from a txt file  \** ***@throws*** *IOException  \*/* public void readResTxt() {  try (FileInputStream input = new FileInputStream(TXT\_FILE\_PATH);  InputStreamReader inputStreamReader = new InputStreamReader(input);  BufferedReader reader = new BufferedReader(inputStreamReader)) {  result = Double.*parseDouble*(reader.readLine());  } catch (IOException e) {  e.printStackTrace();  }  }   */\*\*  \* Method returns the result  \** ***@return*** *\*/* public double getResult() {  return result;  }   */\*\*  \* Method calculates the tg(x)/ctg(x) expression  \** ***@param*** *x - angle in degrees  \** ***@throws*** *CalcException  \*/* public void calculate(double x) throws CalcException{  try {  rad = x \* Math.*PI* / 180;  result = Math.*pow*(Math.*tan*(rad), 2);   if (result == Double.*POSITIVE\_INFINITY* || result == Double.*NEGATIVE\_INFINITY*) {  throw new ArithmeticException();  }  } catch (ArithmeticException e) {   if (rad==Math.*PI*/2.0 || rad==-Math.*PI*/2.0)  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");  }  }   public void setBinaryFilePath(String path) {  BINARY\_FILE\_PATH = path;  }   public void setTxtFilePath(String path) {  TXT\_FILE\_PATH = path;  } }  ***Лістинг Main***  import KI35.Hotsii.Lab6.AccountantFio;  import java.io.IOException; import java.util.Scanner;  */\*\*  \* Class <code>Accountant</code> Implements driver for Accountant class  \** ***@author*** *Mykyta Hotsii  \** ***@version*** *1.0  \*/* public class Main {  */\*\*  \*  \** ***@param*** *args  \** ***@throws*** *IOException  \*/* public static void main(String[] args) throws IOException {  AccountantFio fio = new AccountantFio();  fio.setBinaryFilePath("binary.bin");  fio.setTxtFilePath("text.txt");  Scanner s = new Scanner(System.*in*);  System.*out*.print("Enter data: ");  double data = s.nextDouble();  fio.calculate(data);  System.*out*.println("Result is: " + fio.getResult());  fio.writeResTxt();  fio.writeResBin();  fio.readResBin();  System.*out*.println("Result is: " + fio.getResult());  fio.readResTxt();  System.*out*.println("Result is: " + fio.getResult());   } } |

**Результат роботи програми:**

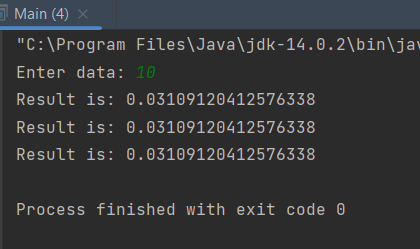
****

Рис.1.1 Результат роботи програми

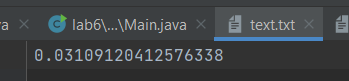


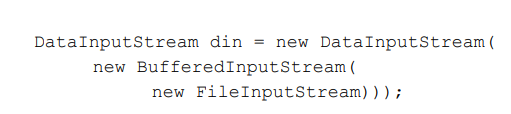
Рис.1.2 Результат роботи програми у файлі textRes.txt

Рис.2.2 Фрагмент згенерованої документації файлу Calculation

**Відповіді на контрольні запитання:**

1. Розкрийте принципи роботи з файловою системою засобами мови Java.

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:



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.

1. Охарактеризуйте клас Scanner

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. Приклад читання даних за допомогою класу Scanner з стандартного потоку вводу:



**Висновок:**

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