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

Флик Є.В.

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 22:**



**Текст програми:**

|  |
| --- |
| ***Лістинг CalcException***  */\*\*  \* package  \*/* package KI35.Flyk.lab6; */\*\*  \* Class <code>CalcException</code> more precises ArithmeticException  \** ***@author*** *EOM Stuff  \** ***@version*** *1.0  \*/* public class CalcException extends ArithmeticException {  */\*\*  \* Constructor  \*/* public CalcException() {  }   */\*\*  \* Constructor with string  \** ***@param*** *cause  \*/* public CalcException(String cause)  {  super(cause);  } }  ***Лістинг Calculation***  */\*\*  \* package  \*/* package KI35.Flyk.lab6;   import java.io.\*; import java.util.\*; */\*\*  \* Class <code>Calculation</code> implements method for (tg(4\*x) / x) calculate  \* calculation  \** ***@author*** *Flyk Yevhenii  \** ***@version*** *1.0  \*/* public class Calculation {  private double result, radian;   */\*\*  \* Method implements write to file  \** ***@param*** *fName  \** ***@throws*** *FileNotFoundException  \*/* public void writeResTxt(String fName) throws FileNotFoundException  {  PrintWriter f = new PrintWriter(fName);  f.printf("%f ",result);  f.close();  }   */\*\*  \* Method implements read from file  \** ***@param*** *fName  \*/* 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\n");  }  catch (FileNotFoundException ex)  {  System.*out*.print(ex.getMessage());  }  }   */\*\*  \* Method implements write to bin file  \** ***@param*** *fName  \** ***@throws*** *FileNotFoundException  \** ***@throws*** *IOException  \*/* public void writeResBin(String fName) throws FileNotFoundException, IOException  {  DataOutputStream f = new DataOutputStream(new FileOutputStream(fName));  f.writeDouble(result);  f.close();   }   */\*\*  \* Method implements read from bin file  \** ***@param*** *fName  \** ***@throws*** *FileNotFoundException  \** ***@throws*** *IOException  \*/* public void readResBin(String fName) throws FileNotFoundException, IOException  {  DataInputStream f = new DataInputStream(new FileInputStream(fName));  result = f.readDouble();  f.close();  }  */\*\*  \* Method calculates the (tg(4\*x) / x) expression  \** ***@param*** *X Angle in degrees  \** ***@throws*** *CalcException  \*/* public void calculate(double X) throws CalcException  {  radian = Math.*toRadians*(4\*X);   try {  result = Math.*tan*(radian);    if (result==Double.*NaN* || result==Double.*NEGATIVE\_INFINITY* || result==Double.*POSITIVE\_INFINITY* || X\*4==90 || X\*4== -90|| X==0)  throw new ArithmeticException();  result = result/X;  }  catch(ArithmeticException ex){  if (radian == Math.*PI*/2.0 || radian == -Math.*PI*/2.0)  throw new CalcException("Exception reason: Illegal value of X for tangent calculation");  else if (X == 0) {  throw new CalcException("X is equal to 0, it is not possible to divide by 0");  } else  throw new CalcException("Unknown reason of the exception during exception calculation");  }   }   */\*\*  \* Method return result  \** ***@return*** *(tg*(*4*\**x) / x)  \*/* public double getResult()  {  return result;  }  }  ***Лістинг CalculationApp***  */\*\*  \* package  \*/* package KI35.Flyk.lab6; import java.util.Scanner; import java.io.\*; import static java.lang.System.*out*; */\*\*  \* Class <code>CalculationApp</code> Implements driver for Calculation class  \** ***@author*** *Flyk Yevhenii  \** ***@version*** *1.0  \*/* public class CalculationApp {   public static void main(String[] args) throws IOException, FileNotFoundException {  boolean IsCatched = true;  Calculation obj = new Calculation();  Scanner s = new Scanner(System.*in*);  System.*out*.print("Enter data: ");  double data = s.nextDouble();   try  {  obj.calculate(data);  }  catch (CalcException ex)  {  IsCatched = false;  System.*out*.println(ex.getMessage());  }  if (IsCatched) {  System.*out*.println("Result is: " + obj.getResult());  obj.writeResTxt("textRes.txt");  obj.writeResBin("BinRes.bin");   obj.readResBin("BinRes.bin");  System.*out*.println("Result is(read from bin): " + obj.getResult());  obj.readResTxt("text1Res.txt");  System.*out*.println("Result is(read from txt): " + obj.getResult());  }   } } |

**Результат роботи програми:**

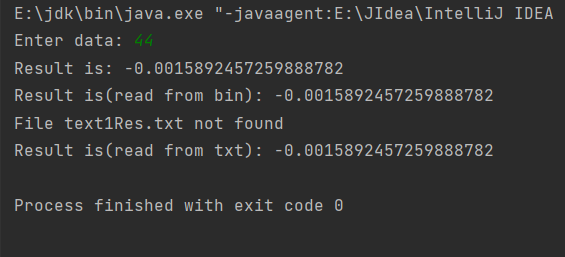


Рис.1.1 Результат роботи програми

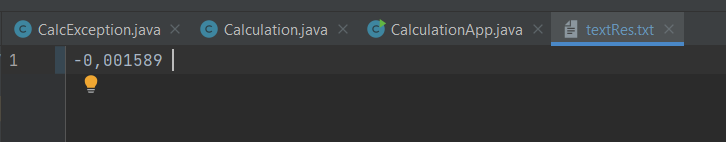


Рис.1.2 Результат роботи програми у файлі textRes.txt

**Фрагмент згенерованої документації**

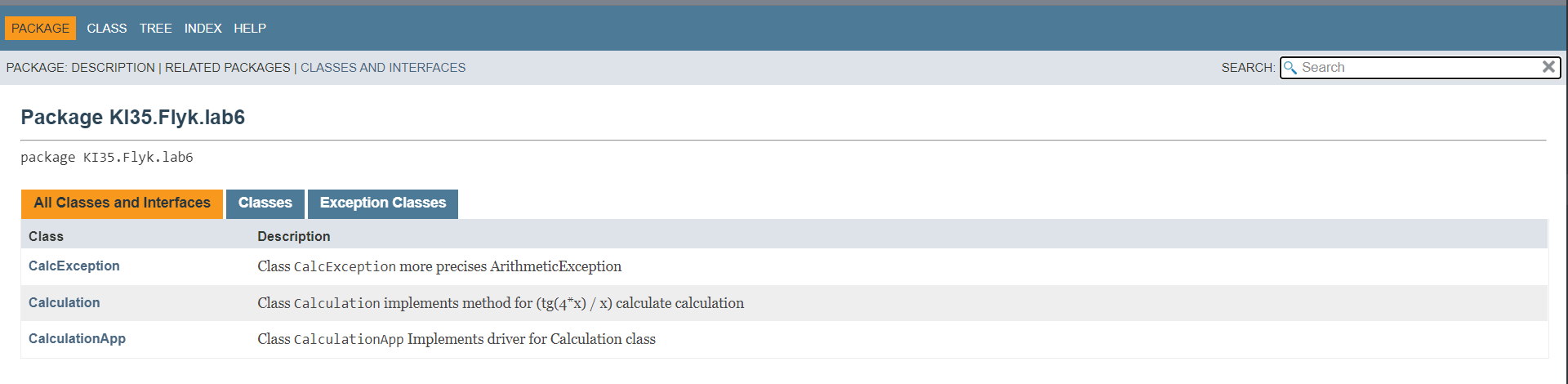


Рис.2.1 Фрагмент згенерованої документації

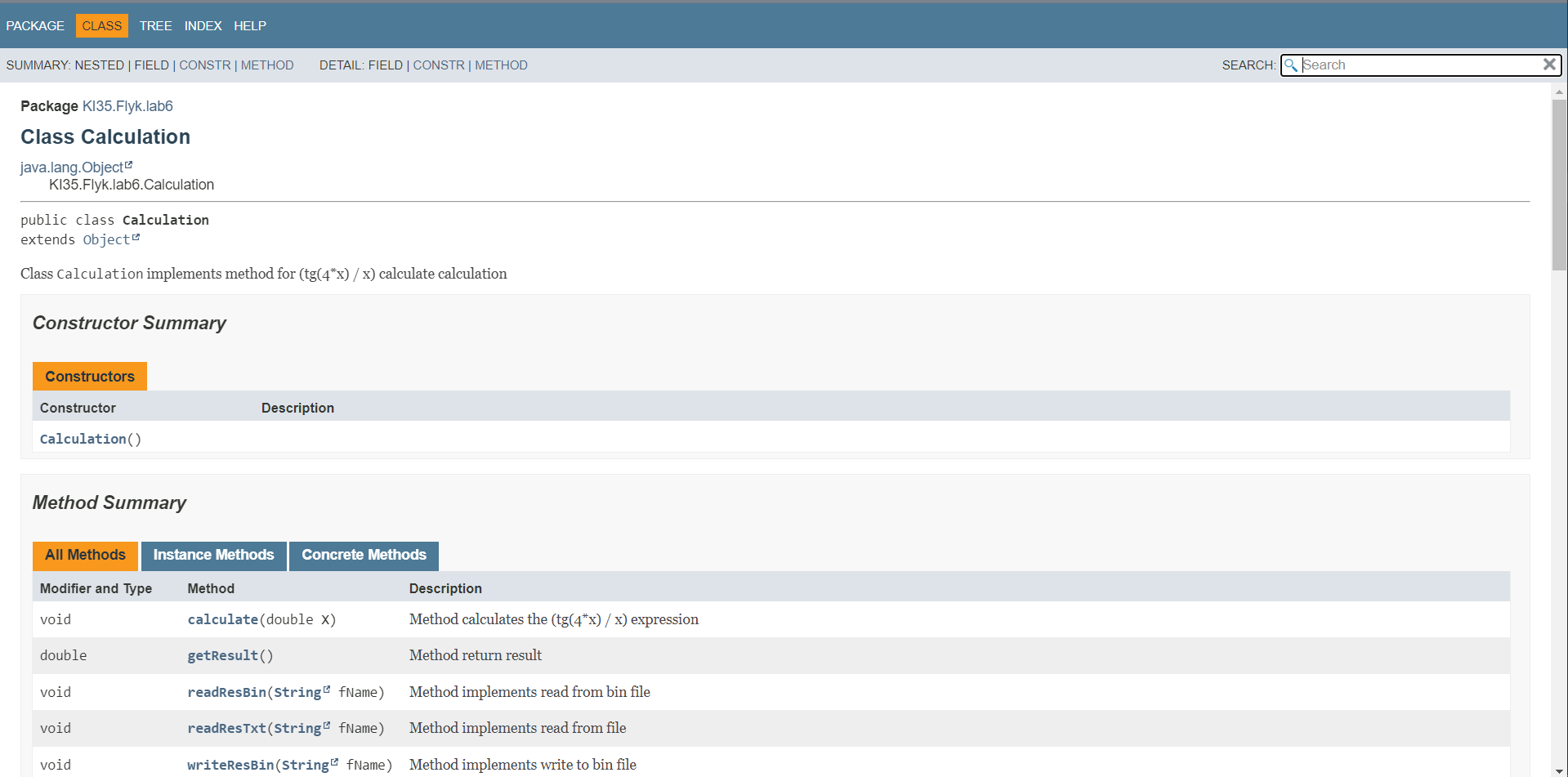
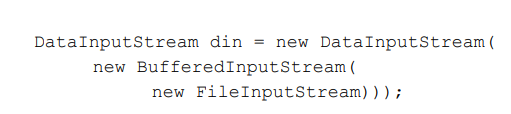


Рис.2.2 Фрагмент згенерованої документації файлу Calculation

**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.