Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



**Звіт**

З лабораторної роботи №3

Варіант – 24

З дисципліни: «Кросплатформні засоби програмування»

На тему: «Файли у Java»

Виконав: ст. гр. КІ-301

Чоланюк О.Ю.

Прийняв:

Майдан М.В.

Львів 2023

**ФАЙЛИ У JAVA**

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. Потаками 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.

**Принципи роботи з файловими потоками**

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:

DataInputStream din = new DataInputStream(

new BufferedInputStream(

new FileInputStream)));

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.

**Читання з текстових потоків**

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:

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();

}

До виходу Java 5.0 єдиним класом для обробки вхідних текстових даних був клас BufferedReader, який мав метод readLine для читання одного рядку тексту**.**

**Запис у текстові потоки**

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 Systen.out. Приклад використання класу PrintWriter:

PrintWriter out = new PrintWriter (“file.txt”); out.print(“Hello ”);

out.print(1070);

out.println(“! I’m World.”);

out.close();

**Читання і запис двійкових даних**

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.

Приклад читання двійкових даних з файлу:

DataInputStream in = new DataInputStream(new FileInputStream (“binarydata.dat”));

int n = in.readInt();

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.

Метод writeUTF здійснює запис рядка у модифікованому машиннонезалежному форматі UTF-8, який крім Java мало хто використовує. Тому для потоків, які не призначені суто для Java, слід використовувати метод writeChars. Приклад запису двійкових даних у файл:

DataOutputStream out = new DataOutputStream(new FileOutputStream (“binarydata.dat”));

out.writeInt(5);

ЗАВДАННЯ

1. Створити клас, що реалізує методи читання/запису у текстовому і двійковому форматах результатів роботи класу, що розроблений у лабораторній роботі №4. Написати програму для тестування коректності роботи розробленого класу.

2. Для розробленої програми згенерувати документацію.

3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.

4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.

5. Дати відповідь на контрольні запитання.

Варіант 24: y=sin(x-9)/(x-cos(2x))

Код:

package lab5;  
  
import java.io.IOException;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 try {  
  
 System.*out*.print("Введіть x: ");  
 double x = new Scanner(System.*in*).nextDouble();  
  
 double y = *calculateY*(x);  
 System.*out*.print("Введіть назву текстового файлу: ");  
 String txtFileName = new Scanner(System.*in*).nextLine();  
  
  
 System.*out*.print("Введіть назву бінарного файлу: ");  
 String binFileName = new Scanner(System.*in*).nextLine();  
  
 FileWrite.*writeResultToTxt*(txtFileName, y);  
  
 FileWrite.*writeResultToBin*(binFileName, y);  
  
 System.*out*.println("Результат було успішно записано в файли.");  
  
 double resultFromTxt = FileWrite.*readResultFromTxt*(txtFileName);  
 System.*out*.println("Результат з текстового файлу: " + resultFromTxt);  
  
 double resultFromBin = FileWrite.*readResultFromBin*(binFileName);  
 System.*out*.println("Результат з бінарного файлу: " + resultFromBin);  
  
 } catch (IOException | NumberFormatException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static double calculateY(double x) {  
 try {  
  
 Equations equations = new Equations();  
 return equations.calculate(x);  
 } catch (CalcException e) {  
 e.printStackTrace();  
 return Double.*NaN*;  
 }  
 }  
}

package lab5;  
  
import java.io.\*;  
import java.util.Scanner;  
  
public class FileWrite {  
 public static void writeResultToTxt(String fileName, double number) throws IOException {  
 try (PrintWriter writer = new PrintWriter(fileName)) {  
 writer.printf("%f ", number);  
 }  
 }  
  
 public static void writeResultToBin(String fileName, double number) throws IOException {  
 try (DataOutputStream dos = new DataOutputStream(new FileOutputStream(fileName))) {  
 dos.writeDouble(number);  
 }  
 }  
  
 public static double readResultFromTxt(String fileName) throws IOException {  
 try (Scanner scanner = new Scanner(new File(fileName))) {  
 return scanner.nextDouble();  
 }  
 }  
  
 public static double readResultFromBin(String fileName) throws IOException {  
 try (DataInputStream dis = new DataInputStream(new FileInputStream(fileName))) {  
 return dis.readDouble();  
 }  
 }  
}

package lab5;  
  
class CalcException extends ArithmeticException {  
 public CalcException() {  
 }  
  
 public CalcException(String cause) {  
 super(cause);  
 }  
}

package lab5;  
  
public class Equations {  
 public double calculate(double x) throws CalcException {  
 validateInput(x);  
  
 double y;  
 double rad = Math.*toRadians*(x);  
  
 try {  
 y = Math.*sin*(x - 9) / (x - Math.*cos*(2 \* x));  
 System.*out*.println(y);  
 if (Double.*isNaN*(y) || Double.*isInfinite*(y)) {  
 throw new ArithmeticException();  
 }  
 } catch (ArithmeticException ex) {  
 throw new CalcException("Причина виліту: помилка в розрахунку формули");  
 }  
 return y;  
 }  
  
 private void validateInput(double x) throws CalcException {  
 if (Double.*isInfinite*(x) || Double.*isNaN*(x)) {  
 throw new CalcException("Причина виліту: некоректне введення X");  
 }  
  
 if (x - Math.*cos*(2 \* x) == 0) {  
 throw new CalcException("Причина виліту: ділення на нуль");  
 }  
 }  
}

Результат:

