Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ

ФАЙЛИ У JAVA

Методичні вказівки
до лабораторної роботи № 5 з курсу “Кросплатформні засоби програмування”
для студентів базового напряму 6.123 - “Комп’ютерна інженерія”

Затверджено
на засіданні кафедри
”Електронні обчислювальні машини”
Протокол № 1 від 28.08.2023 р.

Львів – 2023

Файли у Java: Методичні вказівки до лабораторної роботи № 5 з курсу “Кросплатформні засоби програмування” для студентів базового напряму 6.123 - “Комп’ютерна інженерія” / Укладач: Олексів М.В. – Львів: Національний університет “Львівська політехніка”, 2023, 13 с.

Укладач Олексів М. В., к.т.н.

Рецензент

Відповідальний за випуск: Мельник А. О., професор, завідувач кафедри

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 C, 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);
```

Файли з довільним доступом

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`):

```
RandomAccessFile(File file, String mode);
RandomAccessFile(String name, String mode).
```

Параметр mode може приймати такі значення:

- "r" – читання;
- "rw" – читання/запис;
- "rws" – читання/запис даних з негайним синхронним записом змін у файл або метадані файлу;
- "rwd" – читання/запис даних з негайним синхронним записом змін у файл, метадані файлу не міняються одразу.

Файли, що керуються класом `RandomAccessFile`, оснащені вказівником на позицію наступного байту, що має читатися або записуватися. Для того, щоб перемістити даний вказівник на довільну позицію в межах файлу використовується метод `void seek(long pos)`. Параметр `long pos` визначає номер байту, що має читатися або записуватися.

Щоб дізнатися поточну позицію вказівника на позицію наступного байту слід використати метод `long getFilePointer()`.

Для визначення довжини файлу використовується метод `long length()`, а для закриття файлу – `void close()`.

Для читання і запис даних клас `RandomAccessFile` реалізує методи інтерфейсів `DataInput` і `DataOutput`, які описані у попередньому пункті.

Довільний доступ до файлу є найповільнішим способом доступу до файлів. Трішки швидшим є звичайний потік даних і набагато швидшим є буферизований потік даних.

КОНТРОЛЬНІ ПИТАННЯ

1. Розкрийте принципи роботи з файловою системою засобами мови Java.
2. Охарактеризуйте клас `Scanner`.
3. Наведіть приклад використання класу `Scanner`.
4. За допомогою якого класу можна здійснити запис у текстовий потік?
5. Охарактеризуйте клас `PrintWriter`.
6. Розкрийте методи читання/запису двійкових даних засобами мови Java.
7. Призначення класів `DataInputStream` і `DataOutputStream`.
8. Який клас мови Java використовується для здійснення довільного доступу до файлів.
9. Охарактеризуйте клас `RandomAccessFile`.
10. Який зв'язок між інтерфейсом `DataOutput` і класом `DataOutputStream`?

ЛІТЕРАТУРА

1. Schildt H. Java: The Complete Reference, 12th Edition. / Herbert Schildt. – McGraw Hill, 2021. – 1280 p.
2. Java SE Documentation at a Glance [електронний ресурс]. – Режим доступу до документації: http://www.oracle.com/technetwork/java/javase/documentation/index.html

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.

ВАРІАНТИ ЗАВДАННЬ

1. y=tg(x)
2. y=ctg(x)
3. y=sin(x)/cos(x)
4. y=cos(x)/sin(x)
5. y=2x/sin(x)
6. y=tg(x)/sin(2x)
7. y=ctg(x)/sin(7x-1)
8. y=sin(x)/sin(2x-4)
9. y=tg(x)/3x
10. y=tg(x)/ctg(x)
11. y=ctg(x)/tg(x)
12. y=sin(x)/tg(4x)
13. y=sin(x)/ctg(8x)
14. y=cos(x)/tg(2x)
15. y=cos(2x)/ctg(3x-1)
16. y=7x/tg(2x-4)
17. y=(x-4)/sin(3x-1)
18. y=tg(x)/(sin(4x) – 2cos(x))
19. y=ctg(x)/(sin(2x) + 4cos(x))
20. y=tg(x)ctg(2x)
21. y=sin(3x-5)/ctg(2x)
22. y=tg(4x)/x
23. y=ctg(8x)/x
24. y=sin(x-9)/(x-cos(2x))
25. y=1/sin(x)
26. y=1/cos(4x)
27. y=1/tg(2x)
28. y=1/ctg(2x)
29. y=sin(x)/(x+tg(x))
30. y=cos(x)/(x+2ctg(x))

ПОРЯДОК ВИКОНАННЯ

1. Запустити на виконання середовище Eclipse IDE.
2. Дослідити тестову програму, що наведена в методичних вказівках.
3. Почати створення власного проекту. Для цього слід викликати підпункут меню File->New->Project… У вікні, що відкриється слід вибрати Java Project (рис. 2) та натиснути кнопку ”Next>”.

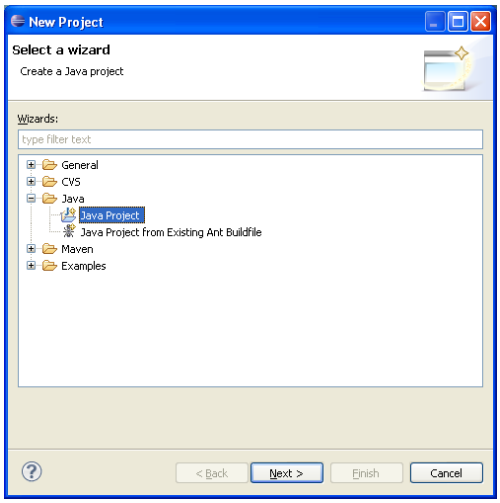

Рис. 2. Діалогове вікно вибору типу проекту в Eclipse IDE.

4. У вікні, що відкрилося, необхідно вказати назву нового проекту, місце розташування проекту, версію JRE на яке орієнтована програм, топологію проекту (Project layout) та натиснути кнопку ”Next>” (рис. 3).

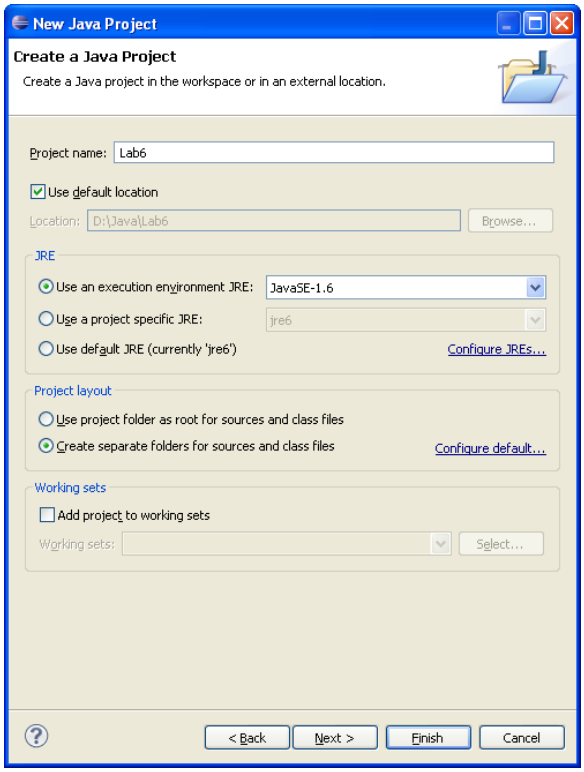

Рис. 3. Діалогове вікно створення нового проекту в Eclipse IDE.

5. У вікні, що відкрилося, натиснути кнопку ”Finish” (рис. 4).

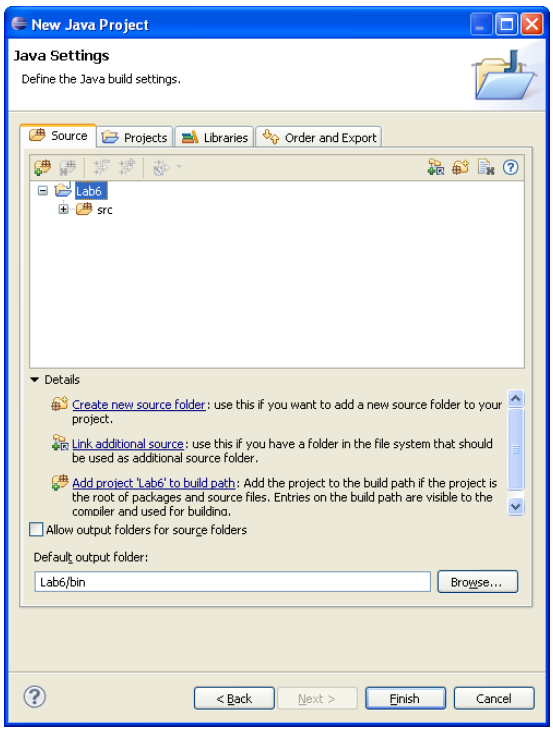

Рис. 4. Діалогове вікно налаштування властивостей побудови нового проекту в Eclipse IDE.

6. У вікні, що з’явилося (рис.5), викликати підпункт меню File->New->Class. У вікні, що відкрилося, слід задати кореневий каталог проекту, назву пакету, назву класу драйвера, що створюється, модифікатор доступу класу, базовий клас, методи, які слід згенерувати автоматично, вказати чи генерувати коментарі автоматично та натиснути кнопку ”Finish” (рис. 6). Аналогічним чином створити клас, що реалізує завдання.

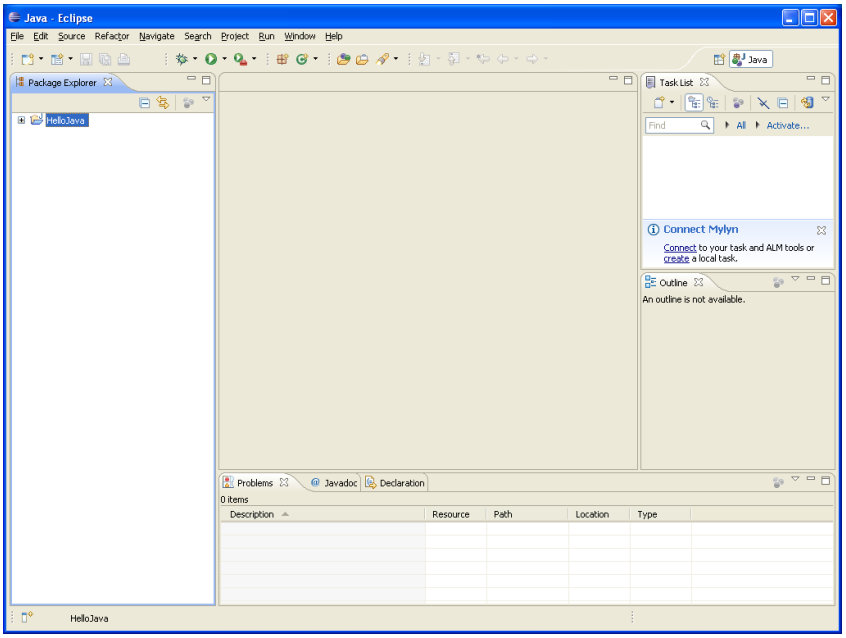

Рис. 5. Середовище Eclipse IDE з відкритим проектом Java.

6. У створеному файлі написати програму згідно завдання.

7. Запустити програму на виконання. Для цього слід вибрати підпункт меню Run-> Run.

8. Встановити точки переривання, запустити налагоджувач (Run->Debug) та покроково дослідити процес виконання програми (рис. 7).

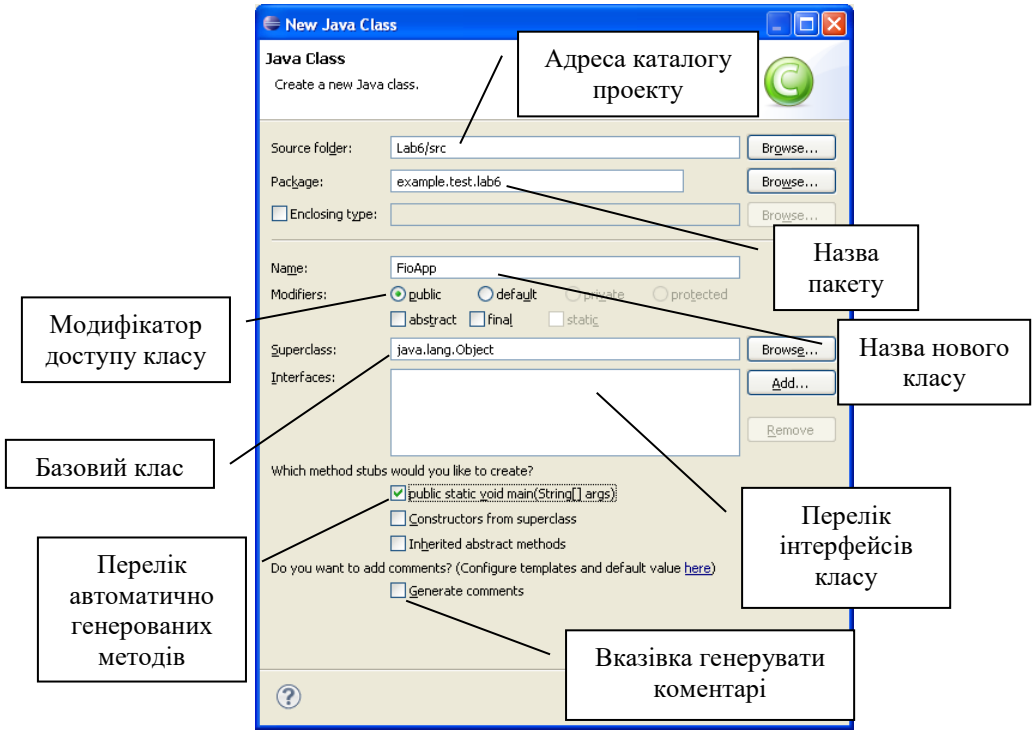

Рис. 6. Створення нового класу з використанням візуального конструктора класів.

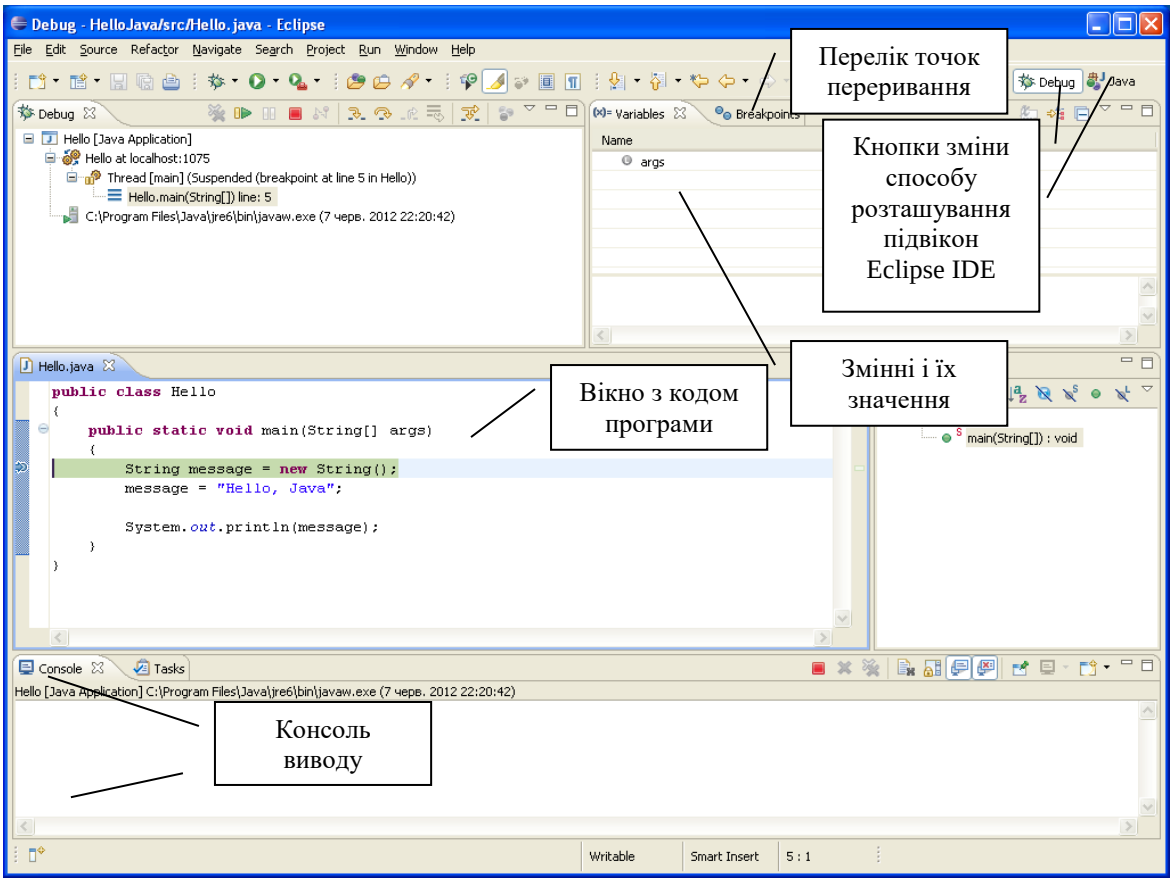

Рис. 7. Приклад вікна налагодження програми.

9. Перебуваючи у підкаталозі \src автоматично згенерувати документацію для всіх загальнодоступних класів у каталог \doc вашого проекту за допомогою команди:

```
javadoc -d ..\doc Група.Прізвище.Lab5
```

Проаналізувати автоматично згенеровану документацію.

ПРИКЛАД ПРОГРАМИ

Файл FioApp.java

```
/*******************************************************************************
 * Copyright (c) 2013-2023 Lviv Polytechnic National University. All Rights Reserved.
 *
 * This program and the accompanying materials are made available under the terms
 * of the Academic Free License v. 3.0 which accompanies this distribution, and is
 * available at https://opensource.org/license/afl-3-0-php/
 *
 * SPDX-License-Identifier: AFL-3.0
 *******************************************************************************/

package example.test.lab5;

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

public class FioApp {

	/**
	 * @param args
	 */
	public static void main(String[] args) throws FileNotFoundException, IOException
	{
		// TODO Auto-generated method stub
		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 is: " + obj.getResult());
		obj.readResTxt("textRes.txt");
		System.out.println("Result is: " + obj.getResult());
	}

}

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 = x*x;
	}

	public double getResult()
	{
		return result;
	}
	private double result;
}
```

НАВЧАЛЬНЕ ВИДАННЯ

ФАЙЛИ У JAVA

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторної роботи № 5 з дисципліни “ Кросплатформні засоби програмування ”
для студентів базового напряму 6.123 - “Комп’ютерна інженерія”

Укладач Олексів М. В., к.т.н.

Редактор

Комп’ютерне верстання

Здано у видавництво . Підписано до друку
Формат 70x100/16. Папір офсетний. Друк на різографі
Умовн. друк. арк. Обл..-вид. арк..
Тираж прим. Зам..

Видавництво Національного університету “Львівська політехніка”
Реєстраційне свідоцтво ДК №751 від 27.12.2001 р.

Поліграфічний центр Видавництва
Національного університету “Львівська політехніка”

Вул. Ф. Колесси, 2. Львів, 79000