Міністерство освіти і науки України

Національний університет «Львівська політехніка»



Звіт

до лабораторної роботи №2

з дисципліни: “Кросплатформні засоби програмування”

на тему: «Класи та пакети»

Варіант №12

Виконала:

ст.гр. КІ-307

Коваль К. О.

Прийняв:

Іванов Ю. С.

**Львів 2023**

**Мета:** ознайомитися з процесом розробки класів та пакетів мовою Java.

**Завдання:**

1. Написати та налагодити програму на мові Java, що реалізує у вигляді класу предметну область згідно варіанту. Програма має задовольняти наступним вимогам:

• програма має розміщуватися в пакеті Група.Прізвище.Lab2;

• клас має містити мінімум 3 поля, що є об’єктами класів, які описують складові частини предметної області;

• клас має містити кілька конструкторів та мінімум 10 методів;

• для тестування і демонстрації роботи розробленого класу розробити клас-драйвер;

• методи класу мають вести протокол своєї діяльності, що записується у файл;

• розробити механізм коректного завершення роботи з файлом (не надіятися на метод finalize());

• програма має володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.

2. Автоматично згенерувати документацію до розробленої програми.

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:  
1. Синтаксис визначення класу.

2. Синтаксис визначення методу.

3. Синтаксис оголошення поля.

4. Як оголосити та ініціалізувати константне поле?

5. Які є способи ініціалізації полів?

6. Синтаксис визначення конструктора.

7. Синтаксис оголошення пакету.

8. Як підключити до програми класи, що визначені в зовнішніх пакетах?

9. В чому суть статичного імпорту пакетів?

10. Які вимоги ставляться до файлів і каталогів при використанні пакетів?

**Варіант 12: Водойма**

**Виконання:**

**WaterBodyApp.java:**

package KI307.Koval.Lab2;

/\*\*

\* A sample application to demonstrate the functionalities of the WaterBody class.

\*/

public class WaterBodyApp {

public static void main(String[] args) {

WaterBody river = new WaterBody("Nile", 6650, "River");

System.***out***.println("Initial Details:");

river.printDetails();

river.setName("Amazon");

river.setArea(7000);

river.setType("River");

river.depthChange(10);

river.absorbWater(50);

river.evaporate(20);

river.pollute();

System.***out***.println("\nModified Details:");

river.printDetails();

river.closeLogFile();

}

}

**WaterBody.java:**

package KI307.Koval.Lab2;

import java.io.FileWriter;

import java.io.IOException;

/\*\*

\* Represents a body of water with various attributes and behaviors.

\*/

public class WaterBody {

private String name;

private int area;

private String type;

private boolean isPolluted;

private int depth;

/\*\*

\* Constructs a WaterBody with specified name, area, and type.

\* @param name The name of the water body.

\* @param area The area covered by the water body.

\* @param type The type of the water body.

\*/

public WaterBody(String name, int area, String type) {

this.name = name;

this.area = area;

this.type = type;

this.isPolluted = false;

this.depth = 0;

}

/\*\*

\* Constructs a WaterBody with a default name and unknown type.

\* @param name The name of the water body.

\*/

public WaterBody(String name) {

this.name = name;

this.area = 0;

this.type = "Unknown";

this.isPolluted = false;

this.depth = 0;

}

/\*\*

\* Sets the name of the water body.

\* @param name The name to set for the water body.

\*/

public void setName(String name) {

this.name = name;

logActivity("Set name to " + name);

}

/\*\*

\* Sets the area of the water body.

\* @param area The area to set for the water body.

\*/

public void setArea(int area) {

this.area = area;

logActivity("Set area to " + area);

}

/\*\*

\* Sets the type of the water body.

\* @param type The type to set for the water body.

\*/

public void setType(String type) {

this.type = type;

logActivity("Set type to " + type);

}

/\*\*

\* Gets the name of the water body.

\* @return The name of the water body.

\*/

public String getName() {

logActivity("Accessed name: " + name);

return name;

}

/\*\*

\* Gets the area of the water body.

\* @return The area of the water body.

\*/

public int getArea() {

logActivity("Accessed area: " + area);

return area;

}

/\*\*

\* Gets the type of the water body.

\* @return The type of the water body.

\*/

public String getType() {

logActivity("Accessed type: " + type);

return type;

}

/\*\*

\* Increases the area of the water body by a specified amount.

\* @param additionalArea The additional area to increase.

\*/

public void increaseArea(int additionalArea) {

this.area += additionalArea;

logActivity("Increased area by " + additionalArea + " to " + area);

}

/\*\*

\* Changes the type of the water body.

\* @param newType The new type to be set.

\*/

public void changeType(String newType) {

this.type = newType;

logActivity("Changed type to " + newType);

}

/\*\*

\* Prints details about the water body including name, area, type, pollution status, and depth.

\*/

public void printDetails() {

System.out.println("Name: " + name);

System.out.println("Area: " + area);

System.out.println("Type: " + type);

System.out.println("Polluted: " + isPolluted);

System.out.println("Depth: " + depth);

logActivity("Printed details");

}

/\*\*

\* Simulates the effect of rainfall on the water body's area.

\* @param rainfall The amount of rainfall.

\*/

public void rainfallEffect(int rainfall) {

if (rainfall > 50) {

increaseArea(rainfall);

logActivity("Rainfall increased area by " + rainfall);

} else {

logActivity("Rainfall was not sufficient to increase area");

}

}

/\*\*

\* Simulates the pollution of the water body.

\*/

public void pollute() {

this.isPolluted = true;

logActivity("Water body is now polluted");

}

/\*\*

\* Cleans the water body from pollution.

\*/

public void clean() {

this.isPolluted = false;

logActivity("Water body is now clean");

}

/\*\*

\* Sets the depth of the water body.

\* @param depth The depth to set.

\*/

public void setDepth(int depth) {

this.depth = depth;

logActivity("Set depth to " + depth);

}

/\*\*

\* Changes the depth of the water body.

\* @param change The change in depth.

\*/

public void depthChange(int change) {

this.depth += change;

logActivity("Changed depth by " + change + " to " + depth);

}

/\*\*

\* Simulates the process of water evaporation from the water body.

\* @param amount The amount of water to be evaporated.

\*/

public void evaporate(int amount) {

if (amount < depth) {

depth -= amount;

logActivity("Evaporated " + amount + " of water");

} else {

depth = 0;

logActivity("Evaporated all water, depth is now 0");

}

}

/\*\*

\* Simulates the process of water absorption by the water body.

\* @param amount The amount of water to be absorbed.

\*/

public void absorbWater(int amount) {

depth += amount;

logActivity("Absorbed " + amount + " of water");

}

/\*\*

\* Logs an activity to a file.

\* @param activity The activity to be logged.

\*/

private void logActivity(String activity) {

try (FileWriter writer = new FileWriter("log.txt", true)) {

writer.write(activity + "\n");

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Closes the log file after finishing with logging activities.

\* This method should be called at the end to ensure proper file closure.

\*/

public void closeLogFile() {

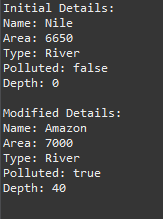
// Method to close the file or handle the file closing process

// This method should be called after finishing working with the log file

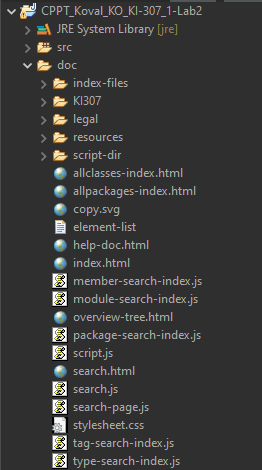
}

}

**Результат:**

****

**Документація:**

****

**Відповіді на контрольні запитання:**

1. \*\*Синтаксис визначення класу в Java:\*\*

```java

access\_modifier class ClassName {

// поля, методи, конструктори

}

```

2. \*\*Синтаксис визначення методу в Java:\*\*

```java

access\_modifier return\_type methodName(parameter\_list) {

// тіло методу

}

```

3. \*\*Синтаксис оголошення поля в Java:\*\*

```java

access\_modifier data\_type variable\_name;

```

4. \*\*Оголошення та ініціалізація константного поля в Java:\*\*

```java

access\_modifier static final data\_type CONSTANT\_NAME = value;

```

5. \*\*Способи ініціалізації полів в Java:\*\*

- Ініціалізація при оголошенні.

- В конструкторі класу.

- В блоку ініціалізації (наприклад, блок `static`).

6. \*\*Синтаксис визначення конструктора в Java:\*\*

```java

access\_modifier ClassName(constructor\_parameters) {

// тіло конструктора

}

```

7. \*\*Синтаксис оголошення пакету в Java:\*\*

```java

package package\_name;

```

8. \*\*Підключення класів з інших пакетів в Java:\*\*

- Використання імпорту: `import package\_name.ClassName;`

- Вказання повного шляху до класу при використанні.

9. \*\*Суть статичного імпорту пакетів в Java:\*\*

- Статичний імпорт дозволяє використовувати статичні (static) члени класу без зазначення імені класу.

- Синтаксис: `import static package\_name.ClassName.\*;`

10. \*\*Вимоги до файлів і каталогів при використанні пакетів в Java:\*\*

- Файл класу повинен знаходитися в тому ж пакеті, що і відповідний каталог в файловій системі.

- Ім'я файлу повинно відповідати імені класу (з розширенням `.java` для файлів з кодом і `.class` для скомпільованих файлів).

**Висновок:** на даній лабораторній роботі я ознайомилась з процесом розробки класів та пакетів мовою Java.