Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра «Електронних обчислювальних машин»



**Звіт**

з лабораторної роботи № 2

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

**на тему:** «Класи та пакети»

**Виконав:**

студент групи *КІ-306*

*Ярмола Юрій*

**Прийняв:**

доцент кафедри ЕОМ

Іванов Ю. С.

**Львів – 2023**

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

**Завдання ( Варіант 30 «Локомотив» ):**

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.Lab3;

• 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 розробленого пакету.

3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її

виконання та фрагменту згенерованої документації.

4. Дати відповідь на контрольні запитання.

**Вихідний код програми**

**Файл LocomotiveAPP**

package KI306.Yarmola.Lab2;

import java.io.FileNotFoundException;

/\*\*

\*

\* Locomotive Application class implements main method for Locomotive

class possibilities demonstration

\* @author Yurii Yarmola

\* @version 1.0

\*

\*/

public class LocomotiveApp {

/\*\*

\*

\* @param args - arguments in main function

\* @throws FileNotFoundException - call when iis not file

\*

\*/

public static void main(String[] args) throws FileNotFoundException

{

Locomotive train;

train = new Locomotive(false, 10, false);

train.start\_locomotive();

train.add\_car(2);

train.add\_speed();

train.add\_speed();

train.add\_speed();

train.is\_door\_opened();

train.minus\_car(5);

train.minus\_speed();

train.stop\_locomotive();

train.dispose();

}

}

**Файл Locomotive**

package KI306.Yarmola.Lab2;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.PrintWriter;

/\*\*

\* The <code>Locomotive</code> class represents a locomotive and its operations.

\* It includes functionality for managing the locomotive's engine, doors, speed,

\* and train cars. This class also logs events to a file named "Log.txt".

\*

\* @author Yurii Yarmola

\* @version 1.0

\*/

public class Locomotive {

private Engine engine;

private int train\_car\_num;

private boolean is\_door\_open;

private PrintWriter fout;

/\*\*

\* Default constructor for the locomotive.

\*

\* @throws FileNotFoundException Thrown when "Log.txt" file is not found.

\*/

public Locomotive() throws FileNotFoundException {

engine = new Engine();

train\_car\_num = 0;

is\_door\_open = false;

fout = new PrintWriter(new File("Log.txt"));

fout.println();

fout.write("Create locomotive");

}

/\*\*

\* Parameterized constructor for the locomotive.

\*

\* @param engine\_working Specifies if the engine is initially working.

\* @param train\_car\_number The initial number of train cars.

\* @param open\_door Specifies if the door is initially open.

\* @throws FileNotFoundException Thrown when "Log.txt" file is not found.

\*/

public Locomotive(boolean engine\_working, int train\_car\_number, boolean open\_door) throws FileNotFoundException {

engine = new Engine(engine\_working);

train\_car\_num = train\_car\_number;

is\_door\_open = open\_door;

fout = new PrintWriter(new File("Log.txt"));

fout.println();

fout.write("Create working locomotive");

}

/\*\*

\* Opens the door of the locomotive.

\*/

public void open\_door() {

fout.println();

fout.write("Door in locomotive is open");

is\_door\_open = true;

}

/\*\*

\* Closes the door of the locomotive.

\*/

public void close\_door() {

fout.println();

fout.write("Door in locomotive is closed");

is\_door\_open = false;

}

/\*\*

\* Checks if the door of the locomotive is open.

\*

\* @return True if the door is open, false otherwise.

\*/

public boolean is\_door\_opened() {

fout.println();

if (is\_door\_open)

fout.write("Door in locomotive is open");

else

fout.write("Door in locomotive is closed");

return is\_door\_open;

}

/\*\*

\* Stops the locomotive's engine.

\*/

public void stop\_locomotive() {

engine.stop\_engine();

fout.println();

fout.write("Stop locomotive");

}

/\*\*

\* Increases the speed of the locomotive.

\*

\* @return The current speed of the locomotive.

\*/

public int add\_speed() {

int power = engine.add\_power();

int speed = power \* 3 / train\_car\_num;

fout.println();

fout.write("Speed now = " + Integer.toString(speed));

return speed;

}

/\*\*

\* Decreases the speed of the locomotive.

\*

\* @return The current speed of the locomotive.

\*/

public int minus\_speed() {

int power = engine.minus\_power();

int speed = power \* 3 / train\_car\_num;

fout.println();

fout.write("Speed now = " + Integer.toString(speed));

return speed;

}

/\*\*

\* Starts the locomotive's engine.

\*/

public void start\_locomotive() {

engine.start\_engine();

fout.println();

fout.write("Start locomotive");

}

/\*\*

\* Adds train cars to the locomotive.

\*

\* @param number The number of train cars to add.

\* @return The updated total number of train cars.

\*/

public int add\_car(int number) {

train\_car\_num += number;

fout.println();

String number\_string = Integer.toString(number);

fout.write("Add " + number\_string + " cars");

return train\_car\_num;

}

/\*\*

\* Removes train cars from the locomotive.

\*

\* @param number The number of train cars to remove.

\* @return The updated total number of train cars.

\*/

public int minus\_car(int number) {

if (train\_car\_num - number < 0) {

train\_car\_num = 0;

} else {

train\_car\_num -= number;

}

String number\_string = Integer.toString(number);

fout.println();

fout.write("Minus " + number\_string + " cars");

return train\_car\_num;

}

/\*\*

\* Closes the PrintWriter used for logging.

\*/

public void dispose() {

fout.close();

}

}

/\*\*

\* The <code>Engine</code> class represents the engine of the locomotive.

\* It includes properties and methods related to the engine's operation.

\*/

class Engine {

private int rotate\_num;

private boolean is\_working;

/\*\*

\* Default constructor for the engine.

\*/

public Engine() {

rotate\_num = 0;

is\_working = false;

}

/\*\*

\* Parameterized constructor for the engine.

\*

\* @param work Specifies if the engine is initially working.

\*/

public Engine(boolean work) {

is\_working = work;

if (work) {

rotate\_num = 1000;

} else {

rotate\_num = 0;

}

}

/\*\*

\* Checks if the engine is working.

\*

\* @return True if the engine is working, false otherwise.

\*/

public boolean is\_engine\_work() {

return is\_working;

}

/\*\*

\* Starts the engine.

\*/

public void start\_engine() {

is\_working = true;

rotate\_num = 1000;

}

/\*\*

\* Stops the engine.

\*/

public void stop\_engine() {

is\_working = false;

rotate\_num = 0;

}

/\*\*

\* Increases the engine's power and returns it.

\*

\* @return The current engine power.

\*/

public int add\_power() {

if (is\_working) {

rotate\_num += 100;

return rotate\_num \* 2;

} else {

return 0;

}

}

/\*\*

\* Decreases the engine's power and returns it.

\*

\* @return The current engine power.

\*/

public int minus\_power() {

if (is\_working) {

if (rotate\_num - 100 < 0) {

rotate\_num = 0;

} else {

rotate\_num -= 100;

}

return rotate\_num \* 2;

} else {

return 0;

}

}

/\*\*

\* Gets the current rotation of the engine.

\*

\* @return The current rotation value.

\*/

public int get\_rotation() {

return rotate\_num;

}

}

**Результат виконання програми**

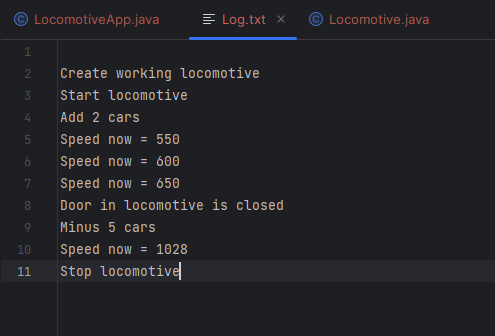
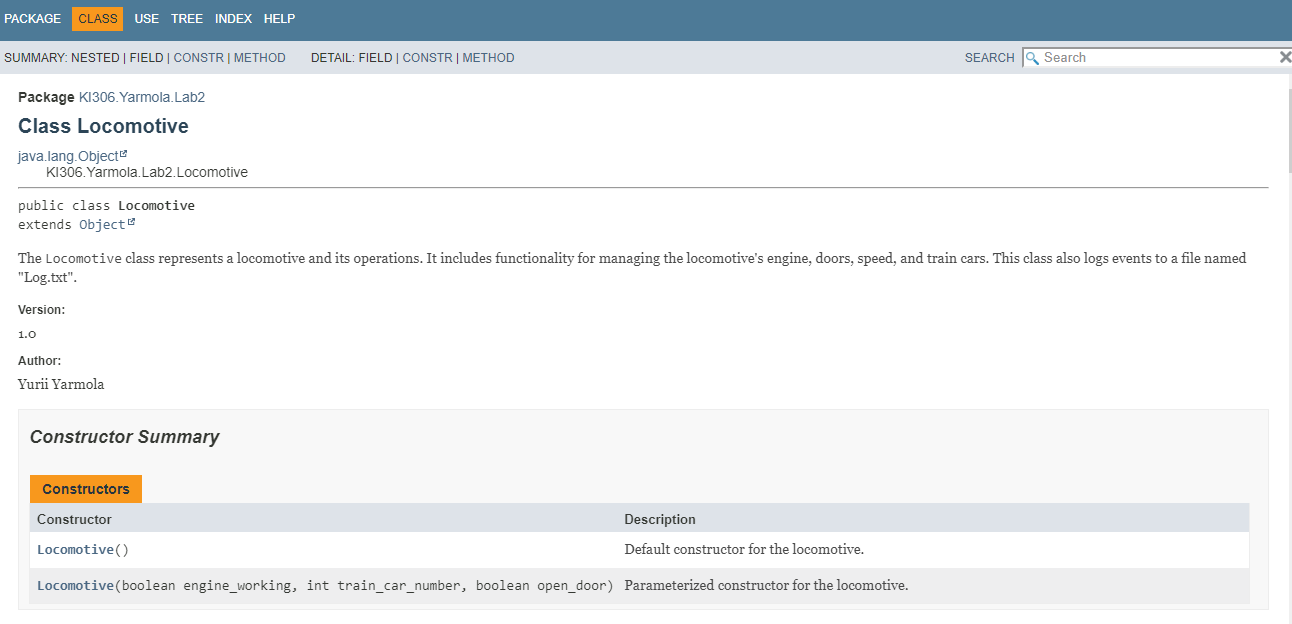
****

Рис.1 – Скріншот документу згенерованого програмою

**Фрагмент згенерованої документації**



**Відповіді на контрольні запитання**

1. Синтаксис визначення класу.

public class ClassName {

// Код класу тут

}

1. Синтаксис визначення методу.

returnType methodName(parameterType parameterName) {

// Тіло методу

}

1. Синтаксис оголошення поля.

dataType fieldName;

1. Як оголосити та ініціалізувати константне поле?

final dataType fieldName = value;

1. Які є способи ініціалізації полів?

конструктор, ініціалізація при оголошенні, блоки ініціалізації.

1. Синтаксис визначення конструктора.

public ClassName(constructorParameters) {

// Тіло конструктора

}

1. Синтаксис оголошення пакету.

package packageName;

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

import packageName.ClassName;

1. В чому суть статичного імпорту пакетів?

Статичний імпорт пакетів дозволяє використовувати статичні члени класу без префіксу

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

Вимоги до файлів і каталогів у використанні пакетів: файл класу повинен зберігатися в каталозі, що відповідає імені пакету, і шлях до нього повинен відповідати його імені простору імен.

**Висновок**

Ознайомився з роботою класів та пакетів у мові програмування Java. Розробив клас який реалізує предметну область «Локомотив».