Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра «Електронних обчислювальних машин»



**Звіт**

з лабораторної роботи № 3

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

**на тему:** «Спадкування та інтерфейси»

**Виконав:**

студент групи *КІ-306*

*Ярмола Юрій*

**Прийняв:**

доцент кафедри ЕОМ

Іванов Ю. С.

**Львів – 2023**

**Мета роботи:** ознайомитися з спадкуванням та інтерфейсами у мові Java.

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

1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі No3, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі No3, зробити абстрактним. Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab4 та 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.

1. Дати відповідь на контрольні запитання.

**Вихідний код програми**

**Файл TrainAPP**

import KI306.Yarmola.Lab3.ElectricTrain;

import java.io.FileNotFoundException;

/\*\*

\*

\* Train Application class implements main method for electric train

class possibilities demonstration

\* @author Yurii Yarmola

\* @version 1.0

\*

\*/

public class TrainApp {

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

ElectricTrain train = new ElectricTrain();

train.start\_locomotive();

train.add\_car(4);

train.open\_train\_door();

train.close\_train\_door();

train.add\_temperature(5);

train.turn\_on\_light();

train.add\_speed();

train.add\_speed();

train.stop\_locomotive();

train.dispose();

}

}

**Locomotive.java**

package KI306.Yarmola.Lab3;

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 abstract class Locomotive {

private Engine engine;

private int train\_car\_num;

private boolean is\_door\_open;

protected 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 = 1;

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;

}

}

**ElectricTrain.java**

package KI306.Yarmola.Lab3;

import KI306.Yarmola.Lab3.Locomotive.\*;

import KI306.Yarmola.Lab3.TrainInterface.\*;

import java.io.FileNotFoundException;

/\*\*

\* The {@code ElectricTrain} class represents an electric train, which is a type of locomotive

\* implementing the {@code TrainInterface}. It extends the {@code Locomotive} class and adds

\* specific functionality for an electric train, including control of train doors, lights,

\* and temperature.

\*

\* @author Yura Yarmola

\* @version 1.0

\*/

public class ElectricTrain extends Locomotive implements TrainInterface {

boolean is\_train\_door\_open;

boolean is\_light;

int temperature;

/\*\*

\* Default constructor for an electric train. Initializes the train with default values.

\*

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

\*/

public ElectricTrain() throws FileNotFoundException {

super();

is\_train\_door\_open = false;

is\_light = false;

temperature = 20;

fout.println();

fout.write("Create Electric train");

}

/\*\*

\* Parameterized constructor for an electric train.

\*

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

\* @param light Specifies if the train light is initially on.

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

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

\*/

public ElectricTrain(boolean open, boolean light, int car\_number) throws FileNotFoundException {

super(false, car\_number, false);

is\_train\_door\_open = open;

is\_light = light;

temperature = 20;

fout.println();

fout.write("Create Electric train");

}

/\*\*

\* Opens the train door.

\*/

@Override

public void open\_train\_door() {

is\_train\_door\_open = true;

fout.println();

fout.write("Train door is opened");

}

/\*\*

\* Closes the train door.

\*/

@Override

public void close\_train\_door() {

is\_train\_door\_open = false;

fout.println();

fout.write("Train door is closed");

}

/\*\*

\* Turns on the train light.

\*/

@Override

public void turn\_on\_light() {

is\_light = true;

fout.println();

fout.write("Train light is turned on");

}

/\*\*

\* Turns off the train light.

\*/

@Override

public void turn\_off\_light() {

is\_light = false;

fout.println();

fout.write("Train light is turned off");

}

/\*\*

\* Increases the temperature of the train cabin.

\*

\* @param temp The temperature to add.

\* @return The updated temperature.

\*/

@Override

public int add\_temperature(int temp) {

temperature += temp;

fout.println();

fout.write("Temperature is " + Integer.toString(temperature));

return temperature;

}

/\*\*

\* Decreases the temperature of the train cabin.

\*

\* @param temp The temperature to subtract.

\* @return The updated temperature.

\*/

@Override

public int minus\_temperature(int temp) {

temperature -= temp;

fout.println();

fout.write("Temperature is " + Integer.toString(temperature));

return temperature;

}

/\*\*

\* Gets the current temperature of the train cabin.

\*

\* @return The current temperature.

\*/

@Override

public int get\_temperature() {

fout.println();

fout.write("Temperature is " + Integer.toString(temperature));

return temperature;

}

/\*\*

\* Closes the PrintWriter used for logging.

\*/

@Override

public void dispose() {

fout.close();

}

}**TrainInterface.java**

package KI306.Yarmola.Lab3;

public abstract interface TrainInterface {

public void open\_train\_door();

public void close\_train\_door();

public void turn\_on\_light();

public void turn\_off\_light();

public int add\_temperature(int temp);

public int minus\_temperature(int temp);

public int get\_temperature();

}

**Результат виконання програми**

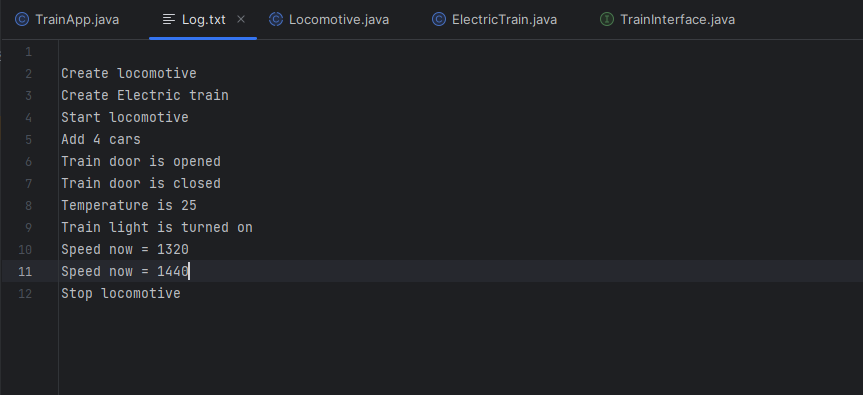
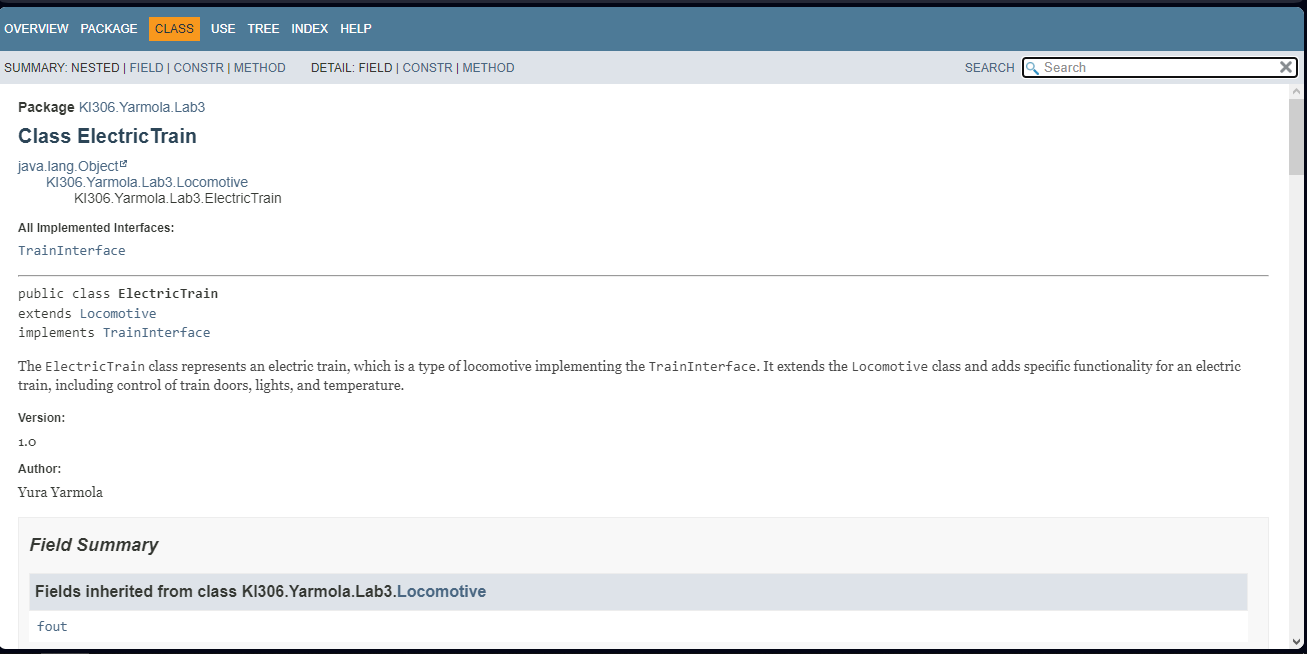
****

Рис.1 – Скріншот документу згенерованого програмою

**Фрагмент згенерованої документації**



**Відповіді на контрольні запитання**

1. Синтаксис реалізації спадкування.

class ChildClass extends ParentClass {

// Код підкласу тут

}

1. Що таке суперклас та підклас?

Суперклас (або батьківський клас) - це клас, від якого успадковуються властивості та методи.

Підклас (або дочірній клас) - це клас, який успадковує властивості та методи від суперкласу і може додавати власні.

1. Як звернутися до членів суперкласу з підкласу?

За допомогою ключового слова super. Наприклад: super.methodName(); для виклику методу суперкласу.

1. Коли використовується статичне зв’язування при виклику методу?

Статичне зв'язування відбувається під час компіляції, коли відомо, який метод викликати на основі типу посилання.

1. Як відбувається динамічне зв’язування при виклику методу?

Динамічне зв'язування відбувається під час виконання програми і визначається на основі об'єкта, який викликає метод.

1. Що таке абстрактний клас та як його реалізувати?

Абстрактний клас - це клас, який не може бути інстанційований і може містити абстрактні методи.

Для створення абстрактного класу використовується ключове слово abstract.

1. Для чого використовується ключове слово instanceof?

Використовується для перевірки, чи об'єкт належить до певного класу або інтерфейсу.

1. Як перевірити чи клас є підкласом іншого класу?

Використовується ключове слово extends. Наприклад, class ChildClass extends ParentClass.

1. Що таке інтерфейс?

Інтерфейс - це контракт, який описує методи, які клас повинен реалізувати. В інтерфейсах методи не мають реалізації.

1. Як оголосити та застосувати інтерфейс?

Оголошення інтерфейсу: interface InterfaceName { ... }

Застосування інтерфейсу в класі: class ClassName implements InterfaceName { ... }

**Висновок**

Ознайомився з спадкуванням та інтерфейсами у мові Java. Розробив розширений клас ElecticTrain за областю з варіанту «Електричка».