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
Довганюк О.С.
Прийняв:
доцент кафедри ЕОМ
Іванов Ю. С.

Львів – 2023

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

Завдання:

1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі №2, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі №2, зробити абстрактним. Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab3 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.

2. Автоматично згенерувати документацію до розробленого пакету.

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 6:

6. Бомбардувальник

Вихідний код програми

Airplane.java

```
package KI306.Dovganiuk.Lab2;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalTime;
import java.time.temporal.ChronoUnit;

/**
 * Клас, що представляє літак
 */
public class Airplane {
```

```
private String model;
private Engine engine;
private Avionics avionics;
private Body body;
private double fuelCapacity;
private boolean isFlying;
private int currentAltitude;
private boolean landingGearDeployed;
private int passengersOnBoard;
private boolean lightsOn;

/**
 * Конструктор, що створює літак з заданими параметрами
 * @param model модель літака
 * @param engine двигун літака
 * @param avionics авіоніка літака
 * @param body кабіна літака
 */
public Airplane(String model, Engine engine, Avionics avionics, Body body) {
    this.model = model;
    this.engine = engine;
    this.avionics = avionics;
    this.body = body;
}

/**
 * Конструктор, що створює літак з заданою моделлю
 * @param model модель літака
 */
public Airplane(String model) {
    this.model = model;
    this.engine = null;
    this.avionics = null;
    this.body = null;
}

/**
 * Метод, що викликає взлет літака
 */
public void takeOff() {
    logAction("Взлетів");
    isFlying = true;
}

/**
 * Метод, що викликає приземлення літака
 */
public void land() {
    logAction("Приземлився");
    logAction("Приземлення успішне!!!! \n==========================================");
    isFlying = false;
}

/**
 * Метод, що змінює висоту польоту літака
 * @param altitude висота польоту
 */
public void fly(int altitude) {
```

```
    if (isFlying) {
        currentAltitude = altitude;
        logAction("Летить на висоті " + altitude + " метрів");
    } else {
        logAction("Літак не в повітрі. Неможливо змінити висоту.");
    }
}

/**
 * Метод, що встановлює двигун літака
 * @param engine двигун літака
 */
public void setEngine(Engine engine) {
    this.engine = engine;
    logAction("Двигун встановлено");
}

/**
 * Метод, що видаляє двигун з літака
 */
public void removeEngine() {
    this.engine = null;
    logAction("Двигун видалено");
}

/**
 * Метод, що висунути шасі
 */
public void deployLandingGear() {
    if (isFlying) {
        landingGearDeployed = true;
        logAction("Висування шасі");
    } else {
        logAction("Літак не в повітрі. Неможливо висунути шасі.");
    }
}

/**
 * Метод, що закрити шасі
 */
public void retractLandingGear() {
    if (isFlying) {
        landingGearDeployed = false;
        logAction("Закриття шасі");
    } else {
        logAction("Літак не в повітрі. Неможливо закрити шасі.");
    }
}

/**
 * Метод, що додає пасажирів на борт
 * @param count кількість пасажирів
 */
public void addPassengers(int count) {
    if (!isFlying || body.seats >= count) {
        passengersOnBoard += count;
        logAction("Пасажирів на борту: " + passengersOnBoard);
    } else if (isFlying) {
```

```
            logAction("Літак у повітрі. Неможливо додати пасажирів.");
        } else {
            logAction("Літак вміщує не більше " + body.seats + " пасажирів");
        }
    }

    /**
     * Метод, що увімкне освітлення
     */
    public void turnOnLights() {
        lightsOn = true;
        logAction("Увімкнено освітлення");
    }

    /**
     * Метод, що вимкне освітлення
     */
    public void turnOffLights() {
        lightsOn = false;
        logAction("Вимкнено освітлення");
    }

    /**
     * Метод, що поповнює пальне
     * @param liters кількість літрів пального
     */
    public void refuel(double liters) {
        fuelCapacity += liters;
        logAction("Поповнено пальне. Поточний об'єм пального: " + fuelCapacity + " л");
    }

    /**
     * Метод, що записує дію в файл логу
     * @param action дія
     */
    public void logAction(String action) {
        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter("log_lab2.txt", true));
            writer.write(LocalTime.now().truncatedTo(ChronoUnit.SECONDS)+"   "+model + ": " + action);
            writer.newLine();
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Клас, що представляє двигун
     */
    public static class Engine {
        private String type;

        /**
         * Конструктор, що створює двигун з заданим типом
         * @param type тип двигуна
         */
        public Engine(String type) {
            this.type = type;
```

```
        }
    }

    /**
     * Клас, що представляє авіоніку
     */
    public static class Avionics {
        private String equipment;

        /**
         * Конструктор, що створює авіоніку з заданим обладнанням
         * @param equipment обладнання авіоніки
         */
        public Avionics(String equipment) {
            this.equipment = equipment;
        }
    }

    /**
     * Клас, що представляє кабіну
     */
    public static class Body {
        private int seats;

        /**
         * Конструктор, що створює кабіну з заданою кількістю місць
         * @param seats кількість місць
         */
        public Body(int seats) {
            this.seats = seats;
        }
    }

/**
 * Метод, що закриває файл
 */
    public static void closeLogFile() {
        //logAction("--------------File Close----------------");
        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter("log_lab2.txt", true));
            writer.write(LocalTime.now().truncatedTo(ChronoUnit.SECONDS) + "    " + "-------------- Закриття
файлу логу--------------\n");
            writer.newLine();
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Bomber.java

```
package KI306.Dovganiuk.Lab3;

/**
 * Interface Bombardier, which describes methods for dropping bombs.
 */
interface Bombardier {
    /**
     * Method for dropping a specified number of bombs.
     * @param count the number of bombs to drop.
     */
    void dropBombs(int count);
}

/**
 * Class Bomber, which inherits from the Airplane class and implements the Bombardier interface.
 */
public class Bomber extends Airplane implements Bombardier {

    private int bombCount; // The number of bombs in the plane.

    /**
     * Constructor for the Bomber class with parameters.
     * @param model the model of the plane.
     * @param engine the engine of the plane.
     * @param avionics the avionics of the plane.
     * @param body the body of the plane.
     * @param bombCount the number of bombs in the plane.
     */
    public Bomber(String model, Engine engine, Avionics avionics, Body body, int bombCount) {
        super(model, engine, avionics, body);
        this.bombCount = bombCount;
    }

    /**
     * Constructor for the Bomber class with the model of the plane.
     * @param model the model of the plane.
     */
    public Bomber(String model) {
        super(model);
        this.bombCount = 0;
    }

    /**
     * Method for dropping a specified number of bombs.
     * @param count the number of bombs to drop.
     */
    public void dropBombs(int count) {
        if (isFlying && bombCount >= count) {
            bombCount -= count;
            logAction("Відпустив " + count + " бомб");
```

```
        } else if (isFlying && bombCount < count){
            logAction("В літаку всього " + bombCount + " бомб");
        } else {
            logAction("Літак не в повітрі. Неможливо скинути бомби.");
        }
    }

    /**
     * Method for loading a specified number of bombs.
     * @param count the number of bombs to load.
     */
    public void loadBombs(int count) {
        if (!isFlying) {
            bombCount += count;
            logAction("Завантажено " + count + " бомб");
        } else {
            logAction("Літак в повітрі. Неможливо завантажити бомби.");
        }
    }
}
```

Main.java

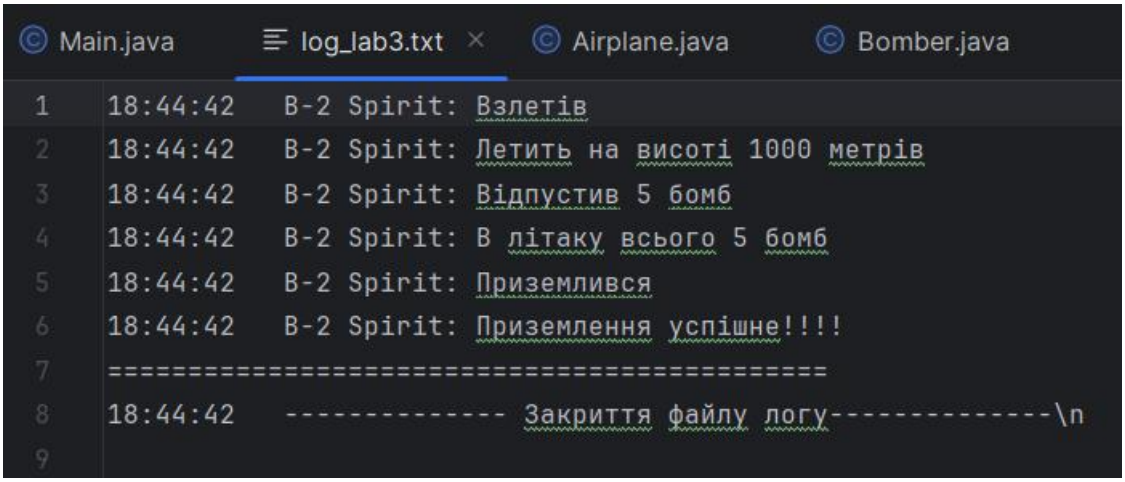
```
package KI306.Dovganiuk.Lab3;

public class Main {
    public static void main(String[] args)
    {
        Bomber bomber = new Bomber("B-2 Spirit", new Airplane.Engine("Twin-turbofan"), new Airplane.Avionics("GlobalSight"), new Airplane.Body(2), 10);

        bomber.takeOff();
        bomber.fly(1000);
        bomber.dropBombs(5);
        bomber.dropBombs(10);

        bomber.land();
        bomber.closeLogFile();
    }
}
```

Текстовий файл з результатом виконання програми



Висновок

Під час виконання даної лабораторної роботи я ознайомився з спадкуванням та інтерфейсами у мові Java.