Міністерство освіти і науки України
Національний університет «Львівська політехніка»

Звіт

до лабораторної роботи №4

З дисципліни: «Кросплатформенні засоби програмування»

На тему: «Спадкування та інтерфейси»

Виконав:

Студент групи КІ-34

Сенета Б.Р.

Львів 2022

Мета: ознайомитися зі спадкуванням та інтерфейсами на мові Java.

Виконання роботи

ЗАВДАННЯ

1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі №3, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі №3, 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.
4. Дати відповідь на контрольні запитання.

Завдання:

ВАРІАНТИ ЗАВДАННЬ

1. Спортсмен
2. Багаторазовий космічний корабель
3. Піддослідний пес
4. Піддослідний кіт
5. Вантажна машина
6. Бомбардувальник
7. Ноутбук
8. Цифрова відеокамера
9. Дерево
10. Офісний центр

11. Сенсорний екран
12. Море
13. Мобільний телефон
14. Телевізор з тюнером
15. Фрегат
16. Диктофон
17. Відеомагнітофон
18. Копіювальний апарат
19. Багатофункціональний пристрій
20. Чоботи
21. Водяний Пістолет
22. Штурмова гвинтівка
23. Газова плитка
24. Спорядження військового альпініста
25. Пристрій кліматконтролю
26. Моторний човен
27. Інтелектуальний патрон
28. Енергозберігаюча лампочка
29. Протигаз командира
30. Електричка

Клас Main:

```
import KI34.Seneta.Lab3.*;

public class App {
    public static void main(String[] args) throws Exception {
        Copier copier = new Copier();
        copier.connectPowerCordConnector();
        copier.turnOnStartButton();
        copier.turnOnPrinter();
        copier.putSmthOnTable();
        copier.set_copy(true);
        copier.startCopping(true);
        copier.dispose();
        copier.turnOffPrinter();
        copier.disconnectPowerCordConnector();
    }
}
```

Клас Printer:

```
/**
 * lab 3 package
 */
package KI34.Seneta.Lab3;

import java.io.*;

/**
 * Class <code>Printer</code> implements computer mouse
 *
 * @author Seneta Bohdan
 * @version 1.0
 */
abstract class Printer {
    private PrinterButton PrinterButton;
    private PrinterTablet PrinterTablet;
    private PrinterPort PrinterPort;
    private PrinterMatrix PrinterMatrix;
    private PrintWriter fout;

    /**
     * Constructor
     *
     * @throws FileNotFoundException
     */
    public Printer() throws FileNotFoundException {
        PrinterButton = new PrinterButton();
        PrinterTablet = new PrinterTablet();
        PrinterPort = new PrinterPort();
        PrinterMatrix = new PrinterMatrix();
        fout = new PrintWriter(new File("Log.txt"));
```

```
    }

    /**
     * Constructor
     *
     * @throws FileNotFoundException
     */

    public Printer(boolean colored) throws FileNotFoundException {
        PrinterButton = new PrinterButton(colored);
        PrinterTablet = new PrinterTablet();
        PrinterPort = new PrinterPort();
        PrinterMatrix = new PrinterMatrix();
        fout = new PrintWriter(new File("Log.txt"));
    }

    /**
     * Method releases used recourses
     */
    public void dispose() {
        fout.close();
    }

    /**
     * Method implements turning on Printer
     */
    public void turnOnPrinter() {
        if (PrinterPort.get_powerCordConnector()) {
            PrinterButton.set_Power(true);
            if (PrinterButton.get_Power()) {
                System.out.print("The Printer is on\n");
                fout.print("The Printer is on\n");
                fout.flush();
            } else {
                System.out.print("The Printer isn't on\n");
                fout.print("The Printer isn't on\n");
                fout.flush();
            }
        }
    }

    /**
     * Method implements turning off Printer
     */
    public void turnOffPrinter() {
        if (!PrinterPort.get_powerCordConnector()) {
            PrinterButton.set_Power(false);
            if (!PrinterButton.get_Power()) {
                System.out.print("The Printer isn't on\n");
                fout.print("The Printer isn't on\n");
                fout.flush();
            } else {
                System.out.print("The Printer is on\n");
```

```
                fout.print("The Printer is on\n");
                fout.flush();
            }
        }
    }

    /**
     * Method implements turn on colored Printning
     */
    public void turnOnColoredPrint() {
        PrinterButton.set_Colored(true);
        if (PrinterButton.get_Colored()) {
            System.out.print("The colored Print is on\n");
            fout.print("The colored Print is on\n");
            fout.flush();
        } else {
            System.out.print("The colored Print isn't on\n");
            fout.print("The colored Print isn't on\n");
            fout.flush();
        }
    }

    /**
     * Method implements turn off colored Printning
     */
    public void turnOffColoredPrint() {
        PrinterButton.set_Colored(false);
        if (PrinterButton.get_Colored()) {
            System.out.print("The colored Print is on\n");
            fout.print("The colored Print is on\n");
            fout.flush();
        } else {
            System.out.print("The colored Print isn't on\n");
            fout.print("The colored Print isn't on\n");
            fout.flush();
        }
    }

    /**
     * Method implements turn on Start button
     */
    public void turnOnStartButton() {
        PrinterButton.set_Start(true);
        if (PrinterButton.get_Start()) {
            System.out.print("The start button is pressed\n");
            fout.print("The start button is pressed\n");
            fout.flush();
        } else {
            System.out.print("The start button isn't pressed\n");
            fout.print("The start button isn't pressed\n");
            fout.flush();
        }
    }
```

```java
    /**
     * Method implements turn off Start button
     */
    public void turnOffStartButton() {
        PrinterButton.set_Start(false);
        if (PrinterButton.get_Start()) {
            System.out.print("The start button is pressed\n");
            fout.print("The start button is pressed\n");
            fout.flush();
        } else {
            System.out.print("The start button isn't pressed\n");
            fout.print("The start button isn't pressed\n");
            fout.flush();
        }
    }

    /**
     * Method implements putting something on Printer table
     * to Print it
     */
    public void putSmthOnTable() {
        PrinterTablet.set_Tablet(true);
        System.out.print("Something put on table\n");
        fout.print("Something put on table\n");
        fout.flush();
    }

    /**
     * Method implements check if something is on the table
     */
    public boolean canWePrint() {
        return PrinterTablet.get_Tablet();
    }

    /**
     * Method implements connecting USB
     */
    public void connectUSB() {
        PrinterPort.set_USB(true);
        if (PrinterPort.get_USB()) {
            System.out.print("The USB is connected\n");
            fout.print("The USB is connected\n");
            fout.flush();
        } else {
            System.out.print("The USB isn't connected\n");
            fout.print("The USB isn't connected\n");
            fout.flush();
        }
    }

    /**
     * Method implements disconnecting USB
```

```
     */
    public void disconnectUSB() {
        PrinterPort.set_USB(false);
        if (PrinterPort.get_USB()) {
            System.out.print("The USB isn't disconnected\n");
            fout.print("The USB isn't disconnected\n");
            fout.flush();
        } else {
            System.out.print("The USB is disconnected\n");
            fout.print("The USB is disconnected\n");
            fout.flush();
        }
    }

    /**
     * Method implements connecting IEEE 1394
     */
    public void connectIEEE1394() {
        PrinterPort.set_IEEE1394(true);
        if (PrinterPort.get_IEEE1394()) {
            System.out.print("The IEEE1394 PORT is connected\n");
            fout.print("The IEEE1394 PORT is connected\n");
            fout.flush();
        } else {
            System.out.print("The IEEE1394 PORT isn't connected\n");
            fout.print("The IEEE1394 PORT isn't connected\n");
            fout.flush();
        }
    }

    /**
     * Method implements disconnecting IEEE 1394
     */
    public void disconnectIEEE1394() {
        PrinterPort.set_IEEE1394(false);
        if (PrinterPort.get_IEEE1394()) {
            System.out.print("The IEEE1394 PORT isn't disconnected\n");
            fout.print("TThe IEEE1394 PORT isn't disconnected\n");
            fout.flush();
        } else {
            System.out.print("The IEEE1394 PORT is disconnected\n");
            fout.print("The IEEE1394 PORT is disconnected\n");
            fout.flush();
        }
    }

    /**
     * Method implements connecting Additional Boards
     */
    public void connectAdditionalBoards() {
        PrinterPort.set_connectorForAdditionalBoards(true);
        if (PrinterPort.get_connectorForAdditionalBoards()) {
            System.out.print("The AdditionalBoards is connected\n");
```

```
            fout.print("The AdditionalBoards is connected\n");
            fout.flush();
        } else {
            System.out.print("The AdditionalBoards isn't connected\n");
            fout.print("The AdditionalBoards isn't connected\n");
            fout.flush();
        }
    }

    /**
     * Method implements disconnecting Additional Boards
     */
    public void disconnectAdditionalBoards() {
        PrinterPort.set_connectorForAdditionalBoards(false);
        if (PrinterPort.get_connectorForAdditionalBoards()) {
            System.out.print("AdditionalBoards isn't disconnected\n");
            fout.print("AdditionalBoards isn't disconnected\n");
            fout.flush();
        } else {
            System.out.print("AdditionalBoards is disconnected\n");
            fout.print("AdditionalBoards is disconnected\n");
            fout.flush();
        }
    }

    /**
     * Method implements connecting Power Cord
     */
    public void connectPowerCordConnector() {
        PrinterPort.set_powerCordConnector(true);
        if (PrinterPort.get_powerCordConnector()) {
            System.out.print("The Power Cord is connected\n");
            fout.print("The Power Cord is connected\n");
            fout.flush();
        } else {
            System.out.print("The Power Cord isn't connected\n");
            fout.print("The Power Cord isn't connected\n");
            fout.flush();
        }
    }

    /**
     * Method implements disconnecting Power Cord
     */
    public void disconnectPowerCordConnector() {
        PrinterPort.set_powerCordConnector(false);
        if (PrinterPort.get_powerCordConnector()) {
            System.out.print("The Power Cord isn't disconnected\n");
            fout.print("The Power Cord isn't disconnected\n");
            fout.flush();
        } else {
            System.out.print("The Power Cord is disconnected\n");
            fout.print("The Power Cord is disconnected\n");
```

```
            fout.flush();
        }
    }

    /**
     * Method implements Printning
     */
    public void Printning() {
        if (PrinterPort.get_powerCordConnector()) {
            if (PrinterButton.get_Power()) {
                if (PrinterButton.get_Start()) {
                    if (PrinterTablet.get_Tablet()) {
                        if (PrinterButton.get_Colored()) {
                            PrinterMatrix.Printned(true);
                            System.out.print("Colored Printning ...\n");
                            System.out.print("Printned\n");
                            fout.print("Colored Printning ...\n");
                            fout.print("Printned\n");
                            fout.flush();
                        } else {
                            PrinterMatrix.Printned(true);
                            System.out.print("White\\Black Printning ...\n");
                            System.out.print("Printned\n");
                            fout.print("White\\Black Printning ...\n");
                            System.out.print("Printned\n");
                            fout.flush();
                        }
                    } else {
                        System.out.print("Nothing to Print try again\n");
                        fout.print("Nothing to Print try again\n");
                        fout.flush();
                    }
                } else {
                    System.out.print("Start button isn't pressed\n");
                    fout.print("Start button isn't pressed\n");
                    fout.flush();
                }
            } else {
                System.out.print("Power button isn't plugged in\n");
                fout.print("Power button isn't plugged in\n");
                fout.flush();
            }
        } else {
            System.out.print("Power isn't plugged in\n");
            fout.print("Power isn't plugged in\n");
            fout.flush();
        }
    }
}

class PrinterButton {
    private boolean isStart;
    private boolean isColored;
```

```
    private boolean isPower;

    /**
     * Constructor default
     */
    public PrinterButton() {
        isStart = false;
        isColored = false;
        isPower = true;
    }

    /**
     * Constructor with three parameters
     */
    public PrinterButton(boolean setStart, boolean setColored, boolean setPower) {
        isStart = setStart;
        isColored = setColored;
        isPower = setPower;
    }

    /**
     * Constructor with one parameter
     */
    public PrinterButton(boolean setColored) {
        isColored = setColored;
    }

    /**
     * Method sets start button
     */
    public void set_Start(boolean setStart) {
        isStart = setStart;
    }

    /**
     * Method sets colored button
     */
    public void set_Colored(boolean setColored) {
        isColored = setColored;
    }

    /**
     * Method sets power button
     */
    public void set_Power(boolean setPower) {
        isPower = setPower;
    }

    /**
     * Method sets start button
     */
    public boolean get_Start() {
        return isStart;
```

```
    }

    /**
     * Method get colored button
     */
    public boolean get_Colored() {
        return isColored;
    }

    /**
     * Method get power button
     */
    public boolean get_Power() {
        return isPower;
    }
}

class PrinterTablet {
    private boolean isOnTablet;

    /**
     * Constructor default
     */
    public PrinterTablet() {
        isOnTablet = false;
    }

    /**
     * Method sets tablet
     */
    public void set_Tablet(boolean sOnTablet) {
        isOnTablet = sOnTablet;
    }

    /**
     * Method get tablet
     */
    public boolean get_Tablet() {
        return isOnTablet;
    }
}

class PrinterPort {
    private boolean USB;
    private boolean IEEE1394;
    private boolean connectorForAdditionalBoards;
    private boolean powerCordConnector;

    /**
     * Constructor default
     */
    public PrinterPort() {
        USB = false;
```

```
        IEEE1394 = false;
        connectorForAdditionalBoards = false;
        powerCordConnector = false;
    }

    /**
     * Method sets USB connection
     */
    public void set_USB(boolean sUSB) {
        USB = sUSB;
    }

    /**
     * Method sets IEEE 1394 connection
     */
    public void set_IEEE1394(boolean sIEEE1394) {
        IEEE1394 = sIEEE1394;
    }

    /**
     * Method sets Additional Boards connection
     */
    public void set_connectorForAdditionalBoards(boolean
sConnectorForAdditionalBoards) {
        connectorForAdditionalBoards = sConnectorForAdditionalBoards;
    }

    /**
     * Method sets Cord Connector connection
     */
    public void set_powerCordConnector(boolean sPowerCordConnector) {
        powerCordConnector = sPowerCordConnector;
    }

    /**
     * Method get USB connection
     */
    public boolean get_USB() {
        return USB;
    }

    /**
     * Method get IEEE 1394 connection
     */
    public boolean get_IEEE1394() {
        return IEEE1394;
    }

    /**
     * Method get Additional Boards connection
     */
    public boolean get_connectorForAdditionalBoards() {
        return connectorForAdditionalBoards;
```

```java
    }

    /**
     * Method get Cord connection
     */
    public boolean get_powerCordConnector() {
        return powerCordConnector;
    }
}

class PrinterMatrix {
    AnalogDigitalDevice ADD;
    private boolean isTransformed;

    /**
     * Constructor default
     */
    public PrinterMatrix() {
        isTransformed = false;
        ADD = new AnalogDigitalDevice();
    }

    /**
     * Method sets transformation
     */
    public void set_transform(boolean sTransform) {
        isTransformed = sTransform;
    }

    /**
     * Method get transformation
     */
    public boolean get_transform() {
        return isTransformed;
    }

    /**
     * Method implement Printning
     */
    public boolean Printned(boolean run) {
        if (run == true) {
            ADD.set_convert(true);
            return true;
        }
        return false;
    }

}

class AnalogDigitalDevice {
    private boolean isConverted;

    /**
```

```
     * Constructor default
     */
    public AnalogDigitalDevice() {
        isConverted = false;
    }

    /**
     * Method sets convert
     */
    public void set_convert(boolean sConvert) {
        isConverted = sConvert;
    }

    /**
     * Method get convert
     */
    public boolean get_convert() {
        return isConverted;
    }

}
```

Клас Copy:

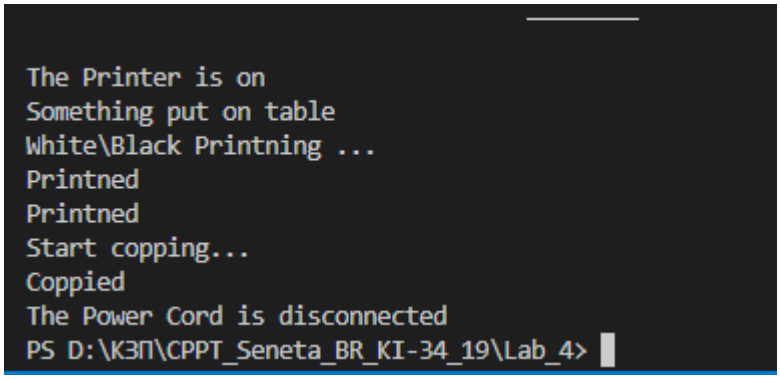
```
package KI34.Seneta.Lab3;

public interface Copy {
    boolean isCopied();

    void startCopping(boolean s_c);
}
```

Інтерфейс Copier:

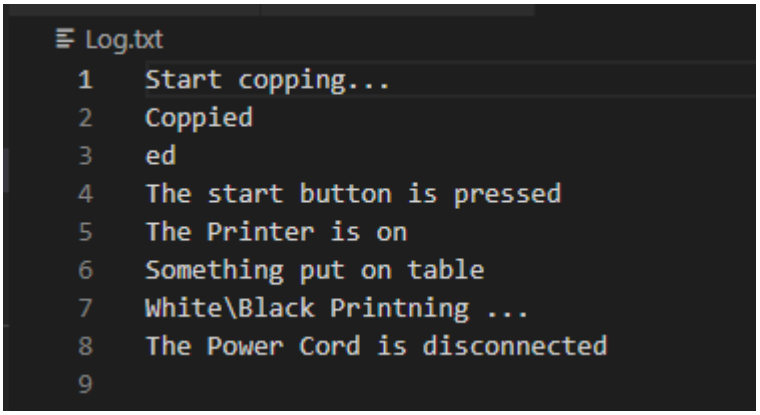
```
package KI34.Seneta.Lab3;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class Copier extends Printer implements Copy {

    private boolean turnOnCopy;
    private boolean isCoppied;

    private PrintWriter fout1;

    public Copier() throws FileNotFoundException {
        super();
        fout1 = new PrintWriter(new File("Log.txt"));
```

```
    }

    /**
     * Method releases used recourses
     */
    public void dispose() {
        fout1.close();
    }

    public void set_copy(boolean s_cpy) {
        this.turnOnCopy = s_cpy;
    }

    boolean get_copy() {
        return this.turnOnCopy;
    }

    void set_coppeid(boolean s_c) {
        this.isCoppied = s_c;
    }

    @Override
    public boolean isCopied() {
        return isCoppied;
    }

    @Override
    public void startCopping(boolean s_c) {
        Printning();
        if (get_copy()) {
            System.out.print("Start copping...\n");
            System.out.print("Coppied\n");
            set_coppeid(s_c);
            fout1.print("Start copping...\n");
            fout1.print("Coppied\n");
            fout1.flush();
        }
    }
}
```

Результат виконання:



Створено файл з записом у нього даних:



Відповіді на КЗ:

1. class Підклас extends Суперклас {
   Додаткові поля і методи
   }
2. Найчастіше супер-клас – це базовий клас, а підклас – це похідний клас від суперкласу.
3. Для звернення до методів чи полів суперкласу з підкласу потрібно використати ключове слово *super*.
   super.назваМетоду([параметри]); // виклик методу суперкласу
   super.назваПоля // звертання до поля суперкласу
4. Статичне зв’язування використовується при поліморфізмі. (компіляція). Лише тоді, коли метод є приватним, статичним або констрктором.
5. Поліморфізм реалізується за допомогою механізму динамічного (пізнього) зв’язування, який полягає у тому, що вибір методу, який необхідно викликати, відбувається *не на етапі компіляції*, а під час виконання програми.

6. Абстрактні класи призначені бути основою для розробки ієрархій класів та не дозволяють створювати об’єкти свого класу. Вони реалізуються за допомогою ключового 49 слова abstract. На відміну від звичайних класів абстрактні класи можуть містити абстрактні методи (а можуть і не містити).
7. Використовується для визначення типу об’єкта в момент виконання програми. Посилання на базовий клас.
8. При наслідувані у Java дозволяється перевизначення (перевантаження) методів та полів. При цьому область видимості методу, що перевизначається, має бути не меншою, ніж область видимості цього методу у суперкласі, інакше компілятор видасть повідомлення, про обмеження привілеїв доступу до даних. Перевизначення методу полягає у визначені у підкласі методу з сигнатурою методу суперкласу. При виклику такого методу з-під об’єкта підкласу викличеться метод цього підкласу. Якщо ж у підкласі немає визначеного методу, що викликається, то викличеться метод суперкласу. Якщо ж у суперкласі даний метод також відсутній, то згенерується повідомлення про помилку.
9. Інтерфейси вказують що повинен робити клас не вказуючи як саме він це повинен робити. Інтерфейси покликані компенсувати відсутність множинного спадкування у мові Java та гарантують визначення у класах оголошених у собі прототипів методів.
10. Синтаксис оголошення інтерфейсів:

```
[public] interface НазваІнтерфейсу {
    Прототипи методів та оголошення констант інтерфейсу
    }
```

Висновок: створено інтерфейс, абстрактний клас та реалізовано наслідування.