

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: «“Мета: ознайомитися з спадкуванням та інтерфейсами у мові
Java.”»

Виконав:
студент групи *KI-306*
Олесько Б. А.

Прийняв:
доцент кафедри ЕОМ
Іванов Ю. С.

Львів – 2023

Мета: ознайомитися з спадкуванням та інтерфейсами у мові Java.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Спадкування

Спадкування в ООП призначене для розширення функціональності існуючих класів

шляхом утворення нових класів на базі вже існуючих. У Java реалізована однокоренева

архітектура класів згідно якої всі класи мають єдиного спільного предка (кореневий клас в

ієрархії класів) – клас Object. Решта класів мови Java утворюються шляхом успадковування даного класу. Будь-яке спадкування у мові Java є відкритим, при цьому

аналогів захищеному і приватному спадкуванню мови C++ не існує. На відміну від C++ у

Java можливе спадкування лише одного базового класу (множинне спадкування відсутнє).

Спадкування реалізується шляхом вказування ключового слова `class` після якого

вказується назва підкласу, ключове слово `extends` та назва суперкласу, що розширюється

у новому підкласі. Синтаксис реалізації спадкування:

Код

```
package K1306.Olesko.Lab3;

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

/**
 * This class represents a monitor with various attributes and functionality.
 */
public abstract class monitor {

    private boolean additional;
    private String additional_devices;
    private String type;
    private int price;
    private int count;
    private String version;
    public boolean isavailable;
    public static int n = 0;
    public static int m = 0;

    /**
     * Default constructor initializes an object with default values.
     */
    public monitor() {
        additional = true;
    }
}
```

```

        additional_devices = "mouse";
        type = "AAd27O";
        version = "0.1";
        count = 1;
        price = 100;
        isavailable = true;
    }

    /**
     * Constructor with parameters initializes an object with specified values.
     *
     * @param type      The type of the monitor.
     * @param version    The version of the monitor.
     * @param price      The price of the monitor.
     * @param count      The quantity of available monitors.
     * @param isavailable The availability status of the monitor.
     * @param additional Whether there are additional devices for the monitor.
     * @param additional_devices The additional devices for the monitor.
     */
    public monitor(String type, String version, int price, int count, boolean isavailable, boolean additional, String
additional_devices) {
        this.additional = additional;
        this.additional_devices = additional_devices;
        this.type = type;
        this.version = version;
        this.price = price;
        this.count = count;
        this.isavailable = isavailable;
    }

    /**
     * Checks if the monitor has additional devices and logs the result.
     */
    public void checkAdditional() {
        if (!additional) {
            writeToLogFile("This monitor does not have additional devices.");
        }
        writeToLogFile("This monitor has additional devices.");
    }

    /**
     * Changes the additional devices for the monitor and logs the change.
     *
     * @param devices The new additional devices.
     */
    public void changeAdditionalDevices(String devices) {
        if (!additional) {
            writeToLogFile("This monitor does not have additional devices.");
        }
        writeToLogFile("You changed additional devices from " + additional_devices + " to " + devices);
        this.additional_devices = devices;
    }

    /**
     * Simulates a monitor purchase and logs the result.
     *
     * @param number The number of monitors to purchase.
     */
    public void buy(int number) {
        if (count < number) {
            writeToLogFile("Oops! You want to buy more than we have available.");
        }
        this.count -= number;
        writeToLogFile("You bought " + number + " monitors. Current count: " + count);
    }

```

```

/**
 * Turns off the availability of the monitor and logs the status change.
 */
public void turnOffAvailable() {
    this.isavailable = false;
    writeToLogFile("Monitor is now unavailable.");
}

/**
 * Changes the type of the monitor and logs the change.
 *
 * @param newType The new type for the monitor.
 */
public void changeType(String newType) {
    this.type = newType;
    writeToLogFile("Type changed to: " + type);
}

/**
 * Changes the version of the monitor and logs the change.
 *
 * @param newVersion The new version for the monitor.
 */
public void changeVersion(String newVersion) {
    this.version = newVersion;
    writeToLogFile("Version changed to: " + version);
}

/**
 * Logs the current status of the monitor.
 */
public void status() {
    writeToLogFile("Type: " + type + "\nVersion: " + version + "\nPrice: " + price + "\nCount: " + count + "\nAvailability: " +
isavailable);
}

/**
 * Adds a specified amount to the price of the monitor and logs the new price.
 *
 * @param prices The amount to add to the price.
 */
public void addPrice(int prices) {
    price += prices;
    writeToLogFile("New price: " + price);
}

/**
 * Adds a specified amount to the count of available monitors and logs the new count.
 *
 * @param counts The amount to add to the count.
 */
public void addCount(int counts) {
    count += counts;
    writeToLogFile("New count: " + count);
}

/**
 * Clears the log file.
 */
public void clearLogFile() {
    File logFile = new File("Olesko.txt");
    try {
        PrintWriter writer = new PrintWriter(logFile);
        writer.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}

```

```

    }
}

/**
 * Writes a message to the log file.
 *
 * @param message The message to be written to the log file.
 */
public void writeToLogFile(String message) {
    try (PrintWriter writer = new PrintWriter(new FileOutputStream(new File("Olesko.txt"), true))) {
        writer.println(message);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
}
}

```

```

package KI306.Olesko.Lab3;

/**
 * monitor Application class implements main method for TouchScreenimplement
 * class possibilities demonstration
 *
 * @author Olesko Bohdan
 * @version 1.0
 * @since version 1.0
 */
public class monitorapp {

    public static void main(String[] args) {
        TouchScreen monitor = new TouchScreen();
        TouchScreen monitor1 = new TouchScreen();
        TouchScreen monitor2 = new TouchScreen();
        phone sam = new phone();
        phone sam1 = new phone();
        phone sam2 = new phone();
        phone sam3 = new phone();
        monitor.clearLogFile();
        sam.turnOffTouchScreen();
        sam1.turnOnTouchScreen();
        sam2.turnOnTouchScreen();
        sam3.turnOffTouchScreen();
        monitor.turnOffTouchScreen();
        monitor1.turnOnTouchScreen();
        monitor2.turnOnTouchScreen();
        monitor2.turnOffTouchScreen();
        System.out.println(monitor.n+" on");
        System.out.println(monitor.m+" off");
    }
}
}

```

```

package KI306.Olesko.Lab3;

public class phone extends monitor implements TouchScreenimplement{
    private boolean Work; // Touch screen operation status

    public phone() {
    }
    @Override
    public void turnOnTouchScreen() {
        // TODO Auto-generated method stub
        setWorks(true);
        n += 1;
        writeToLogFile(" p on "+n);
    }
}

```

```

    }

    @Override
    public void turnOffTouchScreen() {
        setWorks(false);
        m+=1;
        writeToLogFile(" p off ");
    }
    public void setWorks(boolean work) {
        Work = work;
        writeToLogFile("");
    }
}

```

```

package KI306.Olesko.Lab3;

/**
 * The TouchScreen class is a touch screen object that can be used with a monitor.
 * It implements the TouchScreenimplement interface to control the touch screen.
 * @author Olesko Bohdan
 * @version 1.0
 * @since version 1.0
 */

public class TouchScreen extends monitor implements TouchScreenimplement{

    private int Xposition; // The current position of X on the touchscreen
    private int Yposition; // The current position of Y on the touchscreen
    private boolean Work; // Touch screen operation status

    /**
     * Default constructor for the TouchScreen class.
     * Sets the initial values of the X and Y positions to 0 and the touch screen state to "off".
     */

    public TouchScreen(){
        setYposition(0);
        setXposition(0);
        setWork(false);
    }

    /**
     * A method that returns the current X value of the touchscreen position.
     *
     * @return The X value of the touchscreen position.
     */

    public int getXposition() {
        writeToLogFile("TouchScreen Xposition returned "+Xposition);
        return Xposition;
    }

    /**
     * Method that sets the X value of the touchscreen position.
     *
     * @param x The new X value of the touchscreen position.
     */

    public void setXposition(int x) {
        Xposition = x;
        writeToLogFile("TouchScreen Xposition is "+Xposition);
    }

    /**

```

```

* A method that returns the current Y value of the touchscreen position.
*
* @return The Y value of the touchscreen position.
*/

    public int getYposition() {
        writeToLogFile("TouchScreen Yposition returned "+Yposition);
        return Yposition;
    }

    /**
    * Method that sets the Y value of the touchscreen position.
    *
    * @param x The new Y value of the touchscreen position.
    */

    public void setYposition(int y) {
        Yposition = y;
        writeToLogFile("TouchScreen Yposition is "+Yposition);
    }

    /**
    * A method that returns the operating status of the touch screen.
    *
    * @return Touch screen operation status (true - enabled, false - disabled).
    */

    public boolean getWork() {
        writeToLogFile("TouchScreen returned "+Work);
        return Work;
    }

    /**
    * A method that sets the operating status of the touch screen.
    *
    * @param work New status of touch screen operation (true - enabled, false - disabled).
    */

    public void setWork(boolean work) {
        Work = work;
        writeToLogFile("TouchScreen now is "+Work);
    }

    /**
    * Method to enable the touch screen.
    * Sets the operating status of the touch screen to "on".
    */

    @Override
    public void turnOnTouchScreen() {
        setWork(true);
        n+=1;
        writeToLogFile("m"+n);
    }

    @Override
    public void turnOffTouchScreen() {
        setWork(false);
        m+=1;
        writeToLogFile(" m ");
    }
}

```

```

package KI306.Olesko.Lab3;

/**
 * The TouchScreenImplement interface defines methods for controlling the touch screen.
 * Classes that implement this interface must provide functionality
 * to enable and disable the touch screen.
 */

public abstract interface TouchScreenimplement {
    /**
     * Method to enable the touch screen.
     */

    void turnOnTouchScreen();

    /**
     * Method to disable touch screen.
     */

    void turnOffTouchScreen();
}

```

Результат виконання програми з додатковим завданням

```

p off
p on 1
p on 2
p off
TouchScreen now is false
m
TouchScreen now is true
m3
TouchScreen now is true
m4
TouchScreen now is false
m

```


Документація

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [INDEX](#) [HELP](#)

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH:

Package K1306.Olesko.Lab3

Class monitorapp

java.lang.Object[Ⓔ]
K1306.Olesko.Lab3.monitorapp

public class **monitorapp**
extends Object[Ⓔ]

monitor Application class implements main method for TouchScreenimplement class possibilities demonstration

Since:
version 1.0

Version:
1.0

Author:
Olesko Bohdan

Constructor Summary

Constructors

Constructor	Description
monitorapp()	

Method Summary

[All Methods](#) [Static Methods](#) [Concrete Methods](#)

Контрольні запитання

1. Синтаксис реалізації спадкування: ``class ChildClass extends ParentClass{...}``.
2. Суперклас - це клас, від якого інший клас (підклас) успадковує властивості та методи. Підклас - це клас, який успадковує властивості та методи від суперкласу.
3. Для звернення до членів суперкласу з підкласу можна використовувати ключове слово ``super``. Наприклад, ``super.method()`` викличе метод з суперкласу.
4. Статичне зв'язування відбувається при компіляції і визначається типом посилання на об'єкт. Виклик методу визначається під час компіляції і не залежить від типу об'єкта в рантаймі.
5. Динамічне зв'язування відбувається в рантаймі і визначається типом об'єкта. Виклик методу залежить від типу об'єкта, який виконує код в рантаймі.
6. Абстрактний клас - це клас, який не може бути інстанційованим і містить один або більше абстрактних методів. Для його створення використовується ключове слово ``abstract``.
7. Ключове слово ``instanceof`` використовується для перевірки, чи об'єкт належить певному класу або інтерфейсу.

8. Для перевірки, чи клас є підкласом іншого класу, використовується ключове слово ``extends``. Наприклад, ``class ChildClass extends ParentClass``.
9. Інтерфейс - це контракт, який визначає набір методів, які клас повинен реалізувати. Він не містить реалізації методів, а лише їхні сигнатури.
10. Для оголошення інтерфейсу використовується ключове слово ``interface``, наприклад: ``interface MyInterface { ... }``. Щоб застосувати інтерфейс, клас повинен використовувати ключове слово ``implements`` і реалізувати всі методи інтерфейсу.

Висновок

На даній лабораторній роботі я навчився працювати з успадкуванням класів та можливістю створювати інтерфейси і працювати з статичними змінними.