

Міністерство освіти і науки України
Національний університет «Львівська політехніка» Кафедра
«Електронних обчислювальних машин»



Звіт
з лабораторної роботи № 4
з дисципліни: «Кросплатформенні засоби програмування»
на тему: «Спадкування та інтерфейси»

Виконав:
студент групи КІ-36
Музика М
Прийняв:
доцент кафедри ЕОМ
Іванов Ю. С.

Львів – 2022

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

Завдання (варіант №8)

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.
 - Синтаксис реалізації спадкування.
 - Що таке суперклас та підклас?
 - Як звернутися до членів суперкласу з підкласу?
 - Коли використовується статичне зв'язування при виклику методу?
 - Як відбувається динамічне зв'язування при виклику методу?
 - Що таке абстрактний клас та як його реалізувати?
 - Для чого використовується ключове слово instanceof?
 - Як перевірити чи клас є підкласом іншого класу?
 - Що таке інтерфейс?
 - Як оголосити та застосувати інтерфейс?

Індивідуальне завдання: Цифрова відеокамера

Текст програми

```
Camera.java
package KI36.Muzyka.Lab4;

import java.io.*;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList; import
java.util.Date; import
java.util.Scanner;

public abstract class Camera
{
    public boolean isCameraOn = true;
    Scanner scanner = new Scanner(System.in);
    private String name = "unknown";    private
    String firm = "unknown";;    private
    ZoomLever zoom;    private Lens lens;
    private Flash flash;    private Matrix
    matrix;    private ArrayList<Particle>
    photo;    protected Memory memory;
    private PrintWriter fout;
    protected SimpleDateFormat formatter;

    public void turnOffCamera()
    {
        isCameraOn = false;
        fout.write("Camera is turned off");
        fout.flush();    fout.close();
    }

    /**
     * Default constructor to set params from keyboard to camera without name
     * and firm
     * @throws FileNotFoundException
     */
    public Camera() throws FileNotFoundException
    {
        fout = new PrintWriter(new File("MyLog.txt"));
        System.out.println("Enter params for zoom lever: ");

        System.out.print("max zooming >");
        var a = this.scanner.nextInt();
        System.out.print("name >");    var b =
        this.scanner.next();    fout.write("Zoom
        lever:");    zoom = new ZoomLever(a, b);
    }
}
```

```

        System.out.println("Enter params for lens:");
        System.out.println("aperture max, aperture min, focal distance max ,
        focal distance min (for new value separate with space)");
        System.out.print("> ");

        lens = new Lens(scanner.nextInt(), scanner.nextInt(),
        scanner.nextInt(), scanner.nextInt());

        System.out.println("Enter params for matrix:");
        System.out.println("Name matrix and count of megapixels(for
        new value separate with space)");
        System.out.print("> ");
        matrix = new Matrix(scanner.next(), scanner.nextInt());

        photo = new ArrayList<Particle>();

        System.out.println("Enter params for memory:");
        System.out.println("Size of memory and name memory(for new
        value separate with space)");
        System.out.print("> ");
        memory = new Memory(scanner.nextInt(), scanner.next(),
        photo);
        flash = new Flash();
    }

    /**
    * Constructor to read info about camera from txt file
    * @param name The file name
    * @throws IOException
    */
    public Camera(String name) throws
    IOException
    {
        /*this();
        this.name = name;*/

        if(name.lastIndexOf(".txt") != -1 )
        {
            FileReader fr= new FileReader(name);
            Scanner scan = new Scanner(fr);
            ArrayList<String> arrayList = new ArrayList<String>();

            while (scan.hasNextLine()) {arrayList.add(scan.nextLine());}
            fr.close();

            this.firm = arrayList.get(0);
            this.name = arrayList.get(1);

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

            zoom = new ZoomLever(Integer.parseInt(arrayList.get(2)),
            arrayList.get(3));

```

```

        lens = new Lens
            (
                Integer.parseInt(arrayList.get(4)),
                Integer.parseInt(arrayList.get(5)),
                Integer.parseInt (arrayList.get(6)),
                Integer.parseInt(arrayList.get(7))
            );

        matrix = new Matrix(arrayList.get(8),
Integer.parseInt(arrayList.get(9)));

        photo = new ArrayList<Particle>();

        memory = new Memory(Integer.parseInt(arrayList.get(10)),
arrayList.get(11), photo);

        flash = new Flash();
    }else System.out.println("It is no format for txt file");
    }

    /**
    * Construct to set value to camera from keyboard
    * @param name The name of camera
    * @param firm The company that manufactures a camera
    * @throws FileNotFoundException
    */
    public Camera(String name, String firm) throws
FileNotFoundException
    {
        this();
    this.name = name;
    this.firm = firm;
    }

    public String getName() {return name;}

    public void setName(String name) {this.name = name;}
    public String getFirm() {return firm;}

    public void setFirm(String firm) {this.firm = firm;}
    /**
    * Method to change lens on camera
    */
    void
changeLens()
    {
        System.out.println("Enter params for lens " );
        System.out.print("focal distance max >");
        lens.setMaxFocalDistance(scanner.nextInt());
        System.out.print("aperture max >");
        lens.setMaxAperture(scanner.nextInt());
        System.out.print("aperture max >");
        lens.setMinAperture(scanner.nextInt());
        System.out.print("focal

```

```

distance min >");          lens.setMinFocalDistance(scanner.nextInt());
System.out.println("Lens successfully changed");
    }
    void increaseMemory(int size)
{memory.setSizeOfMemory(memory.getSizeOfMemory() + size);}
    /**
     * Method to take photo
     */
    public void updateMemorySize(ArrayList<Particle> photo)
    {
        memory.updateMemory(photo.get(photo.size()-1));
    }
    public void takePhoto()
    {
        photo.add(new Photo(new Date(), (1+ Math.random() * 15), "jpg"));
        updateMemorySize(photo);
        if (memory.getInUsageMemory() > memory.getSizeOfMemory())
        {
            System.out.println("Can not save photo because of leak
memory");
            fout.write("Can not save photo because of leak memory\n");
            photo.remove(photo.get(photo.size()-1));
        }else
        {
            if(flash.isOff())
            {
                System.out.println("Photo is taken with on flash");
                fout.write("Photo is taken with on flash\n");
            }
            else
            {
                System.out.println("Photo is taken with off flash");
                fout.write("Photo is taken with off flash\n");
            }
        }

        fout.flush();
    }

    /**
     * Method to turn on flash on camera
     */
    public void
turnONFlash()
    {
        flash.setOff(true);
        System.out.println("Flash is set on");
        fout.write("Flash is on\n");          fout.flush();
    }

    /**

```

```

* Method to turn off flash on camera
    */    public void
turnOffFlash()
{
    flash.setOff(false);
    System.out.println("Flash is off");
    fout.write("Flash      is              off\n");
    fout.flush();
}

    /**
* Method to increase zoom on camera
* @param zoom The how much we want to increase zoom
    */    public void
increaseZoom(int zoom)
{
    if(zoom + this.zoom.getZoom() > this.zoom.getMaxZoom() )
    {
        System.out.println("Can not to increase zoom because is out of
bound" + "\nThe max value is " + this.zoom.getMaxZoom());
        fout.write("Can not to increase zoom because is out of bound"
+ "\nThe max value is " + this.zoom.getMaxZoom() + '\n');
    }
else
    {
        this.zoom.increaseZoom(zoom);
        fout.write("Zoom increased on " + zoom + "x\n");
    }
    fout.flush();
}

    /**
* Method to decrease zoom on camera
* @param zoom The how much we want to decrease zoom
    */    public void
decreaseZoom(int zoom)
{
    if(this.zoom.getZoom() - zoom < 0) {
        System.out.println("Can not to decrease zoom because is out of
bound" + "\nThe min value is 1");
        fout.write("Can not to decrease zoom because is out of bound"
+ "\nThe min value is 1\n");
    }
else {
        this.zoom.decreaseZoom(zoom);
        fout.write("Zoom decreased on " + zoom + "x\n");
    }
}

    /**
* Set zoom to default value (1)

```

```

        */        public void
setZoomDefault()
{
    zoom.setZoom(zoom.zoomDefault);
    fout.write("Setting up zoom to default value -> 1x\n");
    fout.flush();
}

    /**
    * Method to get status of flash
    * @return The flash is on/off
        */        public boolean getFlashInfo() {return
flash.isOff();}
    /**
    * Method get info about current zoom
        */        public void
getZoomInfo()
{
    System.out.println("Zoom is " + zoom.getInfoZoom() +
    "and now have " + zoom.getZoom() + "x " + ". Max " +
    zoom.getMaxZoom() + "x" );
    fout.write("Zoom is " + zoom.getInfoZoom() +
    "and now have " + zoom.getZoom() + "x " + ". Max " +
    zoom.getMaxZoom() + "x\n" );          fout.flush();
}

    /**
    * Method to view all list of photos that have been taken by the
camera        */        public void viewListPhotos()
{
    DecimalFormat dec = new DecimalFormat("#0.00");
    formatter = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
    fout.write("Viewing list of photos: \n");
    System.out.println("Name      \t" + "Date                                \t" +
    "Size(Mb)\t" + "      Format\t");
    fout.write("Name      \t" + "Date                                \t" +
    "\tSize(Mb)\t" + "      Format\t\n");
    for (var photo : photo) {
        System.out.println(photo.name + '\t' +
        formatter.format(photo.date) + "\t\t" + dec.format(photo.sizeMb) +
        "\t\t\t" + photo.format);
        fout.write(photo.name + '\t' + formatter.format(photo.date) +
        "\t\t" + dec.format(photo.sizeMb) + "\t\t\t" + photo.format + '\n');
    }
    fout.flush();
}

    /**
    * Method to get status about cameras memory

```



```

        */        public void
getMemoryStatus()
    {
        System.out.println("Total memory: " + memory.getSizeOfMemory() +
"Mb");
        System.out.println("In Usage memory: " + memory.getInUsageMemory()
+ "Mb");
        System.out.println("Remaining memory: " + memory.getRemainMemory()
+ "Mb");
        fout.write("Total memory: " + memory.getSizeOfMemory() + "Mb\n");
fout.write("In Usage memory: " + memory.getInUsageMemory() +
"Mb\n");
        fout.write("Remaining memory: " + memory.getRemainMemory() +
"Mb\n");
        fout.flush();

    }

    /**
* Method to delete all photo and clear memory
    */        public void
clearMemory()
    {
        memory.setInUsageMemory(0);
photo.clear();
        fout.write("Memory full cleaned up\n");
fout.flush();
    }

    /**
* Method for deleting picture by name
* @param namePicture The picture name that we want to delete
    */        public boolean deletePicture(String
namePicture)
    {
        if (photo.removeIf(photos -> photos.name.equals(namePicture))) {
fout.write("Photo " + namePicture + " was removed\n");
fout.flush();                return true;
        } else
        {
            fout.write("Photo " + namePicture + " was not removed\n");
fout.flush();                return false;
        }

    }

    /**
* Method to view current lens parameters
    */        public void
viewLens()

```

```

        {
            System.out.println("Now the camera has a lens with the following
parameters:");
            fout.write("Now the camera has a lens with the following
parameters:");
            System.out.println(
                "Aperture max -> " + lens.getMaxAperture() +
                "\nAperture min -> " + lens.getMinAperture() +
                "\nFocal distance max" + lens.getMaxFocalDistance() +
                "\nFocal distance min" + lens.getMinFocalDistance());
            fout.write("Aperture max -> " + lens.getMaxAperture() +
                "\nAperture min -> " + lens.getMinAperture() +
                "\nFocal distance max" + lens.getMaxFocalDistance() +
                "\nFocal distance min" + lens.getMinFocalDistance());
            fout.flush();
        }

        /**
         * Method to get full memory size
         * @return Full size of memory in megabytes
         */
        int getFullMemory() {return
memory.getsizeofMemory();}

        /**
         * Method to get memory that used
         * @return Memory that used by photo
         */
        int getMemoryInUsage() {return
memory.getInUsageMemory();}

        /**
         * Method to get how megapixels the camera have
         * @return Num of megapixels the camera
         */
        int getMegapixels() {return
matrix.getMegaPixels();}

        String getMatrixName(){return matrix.getName();}
    }

    class Lens
    {
        private double maxFocalDistance;
        private double minFocalDistance;
        private double minAperture;        private
        double maxAperture;

        /**
         * Default constructor
         */
        public
        Lens()
        {
            System.out.println("Camera have not Lens. Please put on it");

```

```

    }

    /**
    * Constructor to set parameters for lens of camera
    * @param maxAperture
    * @param minAperture
    * @param maxFocalDistance
    * @param minFocalDistance
    */
    public Lens(double maxAperture, double minAperture, double
maxFocalDistance, double minFocalDistance)
    {
        this.maxAperture = maxAperture;
this.minAperture = minAperture;
this.maxFocalDistance = maxFocalDistance;
this.minFocalDistance = minFocalDistance;
    }

    /**
    * Method to get max focal distance
    * @return
    */
    public double getMaxFocalDistance() {
return maxFocalDistance;
    }

    /**
    * Method to set max focal distance
    * @param maxFocalDistance
    */
    public void setMaxFocalDistance(double maxFocalDistance) {
this.maxFocalDistance = maxFocalDistance;
    }

    /**
    * Method to get min focal distance
    * @return
    */
    public double getMinFocalDistance() {
return minFocalDistance;
    }

    /**
    * Method to get min focal distance
    * @param minFocalDistance
    */
    public void setMinFocalDistance(double minFocalDistance) {
this.minFocalDistance = minFocalDistance;
    }

```

```

        public double getMinAperture() {
            return minAperture;
        }
        public void setMinAperture(double minAperture) {
this.minAperture = minAperture;
        }
        public double getMaxAperture() {
return maxAperture;
        }
        public void setMaxAperture(double maxAperture) {
this.maxAperture = maxAperture;
        }
    }

```

```

class ZoomLever
{
    public int zoomDefault = 1;
private int zoom;        private
int maxZoom;        private String
infoZoom;

    /**
 * Constructor to set default value to zoom
 */
    public ZoomLever()
    {
        this.zoom = zoomDefault;
this.maxZoom = zoomDefault;        infoZoom
= "noname";
    }

    /**
 * Constructor to set max zooming level
 * @param zoom max zooming level
 */    public
ZoomLever(int zoom)
    {
        this.maxZoom = zoom;
this.zoom = zoomDefault;
infoZoom = "noname";
    }

    /**
 * @param zoom The max zooming level
 * @param infoZoom The info about zoom
 */
    public ZoomLever(int zoom, String infoZoom)
    {
        this(zoom);
this.infoZoom = infoZoom;
    }
}

```

```

    }

    public int getZoom()
    {
        return zoom;
    }

    public void setZoom(int zoom)
    {
        this.zoom = zoom;
    }

    /**
     * Method to increase zoom level
     * @param zoom      */
    public void
    increaseZoom(int zoom)
    {
        this.zoom += zoom;
    }

    /**
     * Method to decrease zoom level
     * @param zoom      */
    public void
    decreaseZoom(int zoom)
    {
        this.zoom -= zoom;
    }

    public String getInfoZoom()
    {
        return infoZoom;
    }

    public void setInfoZoom(String infoZoom)
    {
        this.infoZoom = infoZoom;
    }

    public int getMaxZoom() {
return maxZoom;
    }

    public void setMaxZoom(int maxZoom) {
this.maxZoom = maxZoom;
    }
}

class Flash
{
    //is flash off ?
    private boolean isOff;

    /**
     * Default constructor
     * By default flash is off
     */

```

```

Flash() {isOff = false;}

/**
 * Constructor
 * @param isOff The status of flashlight
 */
Flash(boolean isOff) {this.isOff = isOff;}

public boolean isOff() {return isOff;}

public void setOff(boolean off) {isOff = off;}

}

class Matrix
{
    //type of cameras matrix
    private String name;        private
    int megaPixels;

    /**
     * Constructor to set default value to matrix
     */
    public Matrix()
    {
        this.megaPixels = 0;
    }
    this.name = null;
    /**
     * Constructor to set type and count of megapixels of camera      * @param
     * name Type of matrix (CCD/CMOS)
     * @param megaPixels
     */
    public Matrix(String name, int
megaPixels)
    {
        this.name = name;
        this.megaPixels = megaPixels;
    }
    public String getName() {return name;}

    public void setName(String name) {this.name = name;}
    public int getMegaPixels() {return megaPixels;}

    public void setMegaPixels(int megaPixels) {this.megaPixels =
megaPixels;}
}

class Photo extends Particle
{
    //store count of taken photo

```

```

        // private static Integer photoCount = 0;
        //Date date;
        // String format;
        //Double sizeMb;
        //String name;

        /**
        * Constructor
        * The default initialization
        */
        Photo()
        {
            date = new Date();
            format = null;           sizeMb
            = (double) 0;           name =
            null;
        }

        /**
        * Constricctor that initialize info about taken photo and increase count
        of photo
        * @param date The data of photo taken
        * @param sizeMb The size of photo
        * @param format The format of photo (png, jpg, etc...) */
        Photo(Date date, Double sizeMb, String format)
        {
            this.format = format;
            this.date = date;           this.sizeMb
            = sizeMb;
            this.name = "untitled" + itemCount.toString();
            itemCount++;
        }
    }

    class Memory
    {
        private int sizeofMemory;

        private int inUsageMemory;

        private String name;

        /**
        * Default constructor
        */
        public Memory()
        {
            this(sizeofMemory = 0;
            this.name = null;           inUsageMemory
            = 0;

```

```

    }

    /**
    * Constructor for filling memory by photos
    * @param sizeOfMemory The size of photo
    * @param name The name of photo
    * @param photoList The list of all photo that taken
    */
    public Memory(int sizeOfMemory, String name, ArrayList<Particle>
photoList) {
        this.name = name;
        this.sizeOfMemory = sizeOfMemory;
        if(!photoList.isEmpty()) for (var
photo : photoList)
            inUsageMemory += photo.sizeMb;
        else
            inUsageMemory = 0;
    }

    public int getSizeOfMemory() {return sizeOfMemory;}

    public void setSizeOfMemory(int sizeOfMemory) {this.sizeOfMemory =
sizeOfMemory;}

    public String getName() {return name;}

    public void setName(String name) {this.name = name;}

    public int getInUsageMemory() {return inUsageMemory;}

    public void setInUsageMemory(int inUsageMemory) {this.inUsageMemory =
inUsageMemory;}

    /**
    * Method to update memory with new photo size
    * @param photo The object of class photo
    */
    public void updateMemory(Particle photo) {inUsageMemory +=
photo.sizeMb;}

    /**
    * Method to get remaining memory in mb
    * @return remain capacity of memory in mb
    */
    public int getRemainMemory() {return this.sizeOfMemory -
this.inUsageMemory;}
}

```


CameraApp.java

```
package KI36.Muzyka.Lab4;

import java.io.IOException; import
java.util.Scanner;

import static java.lang.System.out;

public class CameraApp
{
    public static void main(String[] args) throws IOException,
    InterruptedException {
        Human Vasya = new Human();
        //put your path to file info.txt
        VideoCamera videoCamera = new VideoCamera("D:\\University\\3
course\\1
semestr\\Кросплатформні засоби програмування\\Clone_rep\\CPPT_Muzyka_DS_
KI-36_1\\Lab4\\src\\info.txt");
        Vasya.takeItem(videoCamera);
    }
}

class Human
{
    private Scanner scanner;

    public void takeItem(VideoCamera camera) throws InterruptedException
    {
        scanner = new Scanner(System.in);
        while (camera.isCameraOn)
        {
            out.println("Please select what do you want to do");
            out.println("1. Take photo"); out.println("2. Change
            lens"); out.println("3. Turn on flash");
            out.println("4. Turn off flash"); out.println("5.
            Increase zoom"); out.println("6. Decrease zoom");
            out.println("7. Set default zoom"); out.println("8.
            Is flash off?"); out.println("9. What zoom level is
            now? "); out.println("10. View list of photo");
            out.println("11. What is my memory status?");
            out.println("12. Clear memory");
            out.println("13. Delete some picture by name");
            out.println("14. Get full size of memory");
            out.println("15. Get memory in usage");
            out.println("16. Get camera info");
            out.println("17. Get lens info");
```

```

out.println("18. Turn off device");
out.println("19. Record video");
out.println("20. View list of video");

        out.print("Enter you choice > ");
int choice = scanner.nextInt();
switch (choice) {
    case 1 -> {
        camera.takePhoto();
        out.println("\n-----
");
    }
    case 2
-> {
        camera.changeLens();
        out.println("\n-----
");
    }
    case 3 -> {
        camera.turnONFlash();
        out.println("\n-----
");
    }
    case 4 -> {
        camera.turnOffFlash();
        out.println("\n-----
");
    }
    case 5 -> {
        out.print("How much to increase the zoom ? \nEnter
>");
        camera.increaseZoom(scanner.nextInt());
        out.println("\n-----
");
    }
    case 6 -> {
        out.print("How much to decrease the zoom ? \nEnter
>");
        camera.decreaseZoom(scanner.nextInt());
        out.println("\n-----
");
    }
    case 7 -> {
        camera.setZoomDefault();
        out.println("\n-----
");
    }
    case 8 -> {
        if(camera.getFlashInfo()){
out.println("Flash is in");
        }else
{
            out.println("Flash is off");

```

```

        }
        out.println("\n-----
");
    }
    case 9 -> {
        camera.getZoomInfo();
        out.println("\n-----
");
    }
    case 10 -> {
        camera.viewListPhotos();
        out.println("\n-----
");
    }
    case 11 -> {
        camera.getMemoryStatus();
        out.println("\n-----
");
    }
    case 12 -> {
        camera.clearMemory();
        out.println("\n-----
");
    }
    case 13 -> {
        out.print("Picture name that you want to delete ?
\nEnter >");
        var namePhoto = scanner.next();
        if(camera.deletePicture(namePhoto))
            out.println("Photo " + namePhoto + " was successfully deleted");
        else
            out.println("Photo " + namePhoto + " was failed to
delete. Incorrect name");
        out.println("\n-----
");
    }
    case 14 -> {
        int mem = camera.getFullMemory();
        out.println("Full memory is " + mem + " mb");
        out.println("\n-----
");
    }
    case 15 -> {
        int memInUse = camera.getMemoryInUsage();
        out.println("\n-----
");
        out.println(memInUse + " mb");
    }
    case 16 -> {
        String name = camera.getName();
        String firm = camera.getFirm();
        int mgPx

```

```

= camera.getMegapixels();
= camera.getMatrixName();
        out.println("Name is " + name + "\nFirm is " + firm +
"\nMegapixels is " + mgPx + "\nMatrix is " + matrixName);
        out.println("\n-----
");
    }
    case 17 -> {
        camera.viewLens();
    }
    case 18 -> {
        camera.turnOffCamera();
        out.println("\n-----Camera is off-----");
    out.println("\n-----GOODBYE-----");
    }
    case 19 -> {
        camera.startRecord("1080P" , 60);
    }
    case 20 -> {
        camera.printData();
    }
    default -> {
        out.println("Unknown command! Try again! ");
    out.println("\n-----
");
    }
    }
    }
    }
}

```

IPromptableKey

```

package KI36.Muzyka.Lab4;

import java.io.IOException;

/**
 * Interface that have some prompt key
 */
public interface IPromptableKey
{
    static boolean promptEnterKey()
    {
        System.out.println("Press \"ENTER\" to stop recording video...");
    }
    try {
        System.in.read();
    } catch
    return true;
    (IOException e) {

```

```

        e.printStackTrace();
    }
    return false;
}
}

```

IRecorder

```
package KI36.Muzyka.Lab4;
```

```

/**
 * Interface that contain method for recording video
 */
public interface IRecorder extends
IPromptableKey
{
    void startRecord(String resolution, int fps) throws
InterruptedException;    void stopRecord();

    double sizeOfVideo(String resolution, int fps);
}

```

Particle

```
package KI36.Muzyka.Lab4;
```

```
import java.util.Date;
```

```

public abstract class Particle
{
    static Integer itemCount = 0;
    public int fps;    public String
resolution;
    Date date;
    String format;
    Double sizeMb;
    String name;
}

```

Printable

```
package KI36.Muzyka.Lab4;
```

```
/**
```

```
 * Interface that have method to print data from device
```

```
 */ public interface
```

```
Printable
```

```
{
```

```
    void printData();
```

```
}
```

StopWatch

```
package KI36.Muzyka.Lab4;
```

```
 * public class StopWatch extends Thread implements Watcher
```

```
{
```

```
    public boolean isTimerStart()
```

```
    {
```

```
        return isTimerStart;
```

```
    }
```

```
    public void setTimerStart(boolean timerStart)
```

```
    {
```

```
        isTimerStart = timerStart;
```

```
    }
```

```
    private boolean isTimerStart;
```

```
    protected int sec = 0;
```

```
    public String getWriteThis() {return writeThis;}
```

```
    public void setWriteThis(String writeThis) {
```

```
    this.writeThis = writeThis;
```

```
    }
```

```
    private String writeThis;
```

```
    StopWatch() {this.isTimerStart = true;}
```

```
    /**
```

```
 * Override method to start stop watch in new thread
```

```
 */ @Override
```

```
    public void run() {startStopWatch();}
```

```
    /**
```

```
 * Method to start stop watch
```

```

        */
@Override
    public void startStopWatch()
    {
        sec = 0;
        int charsWritten = 0;
        long start = System.currentTimeMillis();
        while (isTimerStart)
        {
            try
            {
                Thread.sleep(msInSec);
                sec++;
            }
            catch (InterruptedException e)
            {
                throw new RuntimeException(e);
            }
            long elapsedTime = System.currentTimeMillis() - start;
            elapsedTime = elapsedTime / 1000;

            String seconds = Integer.toString((int) (elapsedTime %
            secInMinute));
            String minutes = Integer.toString((int) ((elapsedTime %
            secInHours) / secInMinute));
            String hours = Integer.toString((int) (elapsedTime /
            secInHours));

            if (seconds.length() < 2) {
                seconds = "0" + seconds;
            }

            if (minutes.length() < 2) {
                minutes = "0" + minutes;
            }

            if (hours.length() < 2) {
                hours = "0" + hours;
            }

            writeThis = hours + ":" + minutes + ":" + seconds;

            for (int i = 0; i < charsWritten; i++) {
                System.out.print("\b");
            }
            System.out.print(writeThis);
            charsWritten = writeThis.length();

        }
    }

```

```
    }  
}
```

VideoCamera.java

```
package KI36.Muzyka.Lab4;  
  
import java.io.FileNotFoundException;  
import java.io.IOException; import  
java.text.DecimalFormat; import  
java.text.SimpleDateFormat; import  
java.util.ArrayList; import  
java.util.Date; import  
java.util.Objects;  
  
public class VideoCamera extends Camera implements IRecorder, Printable,  
IPromptableKey  
{  
    private Stopwatch watcher;  
  
    private ArrayList<Particle> videoList;  
    /**  
    * Default constructor  
    * @throws FileNotFoundException  
    */  
    public VideoCamera() throws  
FileNotFoundException  
    {  
super();  
        videoList = new ArrayList<>();  
    }  
  
    /**  
    * Constructor to initialize data from file  
    * @param name Name of camera  
    * @throws IOException  
    */  
    public VideoCamera(String name) throws  
IOException  
    {  
super(name);  
        videoList = new ArrayList<>();  
    }  
  
    /**  
    * Consttucor to initialize name and firm camera  
    * @param name Name of camera  
    * @param firm Firm of camera  
    * @throws FileNotFoundException
```



```

        */      public VideoCamera(String name, String
firm) throws
FileNotFoundException
    {
        super(name, firm);
        videoList = new ArrayList<>();
    }

    /**
* Method to recording in new thread
* @param resolution
* @param fps
* @throws InterruptedException
*/
@Override
    public void startRecord(String resolution, int fps) throws
InterruptedException
    {
        watcher = new Stopwatch();
        System.out.print("Recording is starting...\n");
        watcher.start();
        if(IPromptableKey.promptEnterKey())
        {
            stopRecord();
            System.out.println("Recording is stopping...");
        }
        watcher.join();
        var sizeV = sizeOfVideo(resolution, fps);
if(sizeV != -1 )
        {
            videoList.add(new Video(new Date(), sizeV, "mkv", resolution,
fps));
            updateMemorySize(videoList);
        }else
        {
            System.out.println("Size is NULL!");
        }
    }

    /**
* Method to stop recording
*/
@Override
    public void stopRecord() {watcher.setTimerStart(false);}
    /**
* Method to get size of video
* @param resolution

```

```

* @param fps
* @return
*/
@Override
    public double sizeOfVideo(String resolution, int fps)
    {
        for( var a : Video.RESOLUTION.values())
        {
            for(var b: Video.FPS.values())
            {
                if(Objects.equals(a.getValue(), resolution) &&
b.getValue() == fps)
                {
                    for(var c : Video.COEFFICIENT_FPS.values() )
                    {
                        for(var d :
Video.COEFFICIENT_RESOLUTION.values())
                        {
                            if(Objects.equals(a.name(), d.name()) &&
b.name().equals(c.name()))
                            {
                                return watcher.sec * c.getValue() *
d.getValue();
                            }
                        }
                    }
                }
            }
        }
        return -1;
    }

    /**
     *Method to print all video
     */
    @Override
    public void printData()
    {
        DecimalFormat dec = new DecimalFormat("#0.00");
        formatter = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
        System.out.println("Name \t" + " Date \t" +
"\tSize(Mb)\t" + " Format\t" + " \t\t Resolution" + "\t\t FPS");
        for (var photo : videoList) {
            System.out.println(photo.name + '\t' +
formatter.format(photo.date) + "\t\t\t" + dec.format(photo.sizeMb) +
"\t\t\t" + photo.format + "\t\t\t\t" + photo.resolution + "\t\t" +
photo.fps);
        }
    }
}

```

```

}

class Video extends Particle
{
    enum COEFFICIENT_FPS
    {
        FPS30(1) , FPS60(2), FPS120(4);

        private final int value;
        COEFFICIENT_FPS(int i)
        {
            this.value = i;
        }
        public int getValue()
        {
            return value;
        }
    }

    enum COEFFICIENT_RESOLUTION
    {
        P1080(4), P720(2), P480(1), K2(6), K4(10);
    private final int value;
        COEFFICIENT_RESOLUTION(int i)
        {
            this.value = i;
        }
        public int getValue()
        {
            return value;
        }
    }

    enum FPS
    {
        FPS30(30) , FPS60(60), FPS120(120);

        private final int value;

        FPS(int value) {
this.value = value;
        }
        public int getValue()
        {
            return value;
        }
    }

    enum RESOLUTION
    {
        P1080("1080P"), P720("720P"), P480("480P"), K2("2K"), K4("4K");
    private final String value;
        RESOLUTION(String i)

```

```

        {
            this.value = i;
        }
        public String getValue()
        {
            return value;
        }
    }

    /**
    * Constructor
    * The default initialization
    */
    Video()
    {
        date = new Date();
        format = null;           sizeMb
        = (double) 0;           name =
        null;                   resolution =
        null;                   fps = 0;
    }

    /**
    * Constructor for initialize data of video
    * @param date
    * @param sizeMb
    * @param format
    * @param resolution
    * @param fps
    */
    Video(Date date, Double sizeMb, String format, String resolution, int
    fps)
    {
        this.format = format;
        this.date = date;           this.sizeMb
        = sizeMb;
        this.name = "untitledVideo" + itemCount.toString();
        this.resolution = resolution;    this.fps = fps;
        itemCount++;
    }
}

```

Watcher

```
package KI36.Muzyka.Lab4;
```

```
/**
```

```
 * Interface to realize stop watch / timer
```

```
 */ public interface
```

```
Watcher
```

```
{
```

```
    int msInSec = 1000;
```

```
int secInMinute = 60;
```

```
int secInHours = 3600;
```

```
void startStopWatch();
```

```
}
```

info.txt

SONY

FX3 Body

20

Optical

1

5

18

55

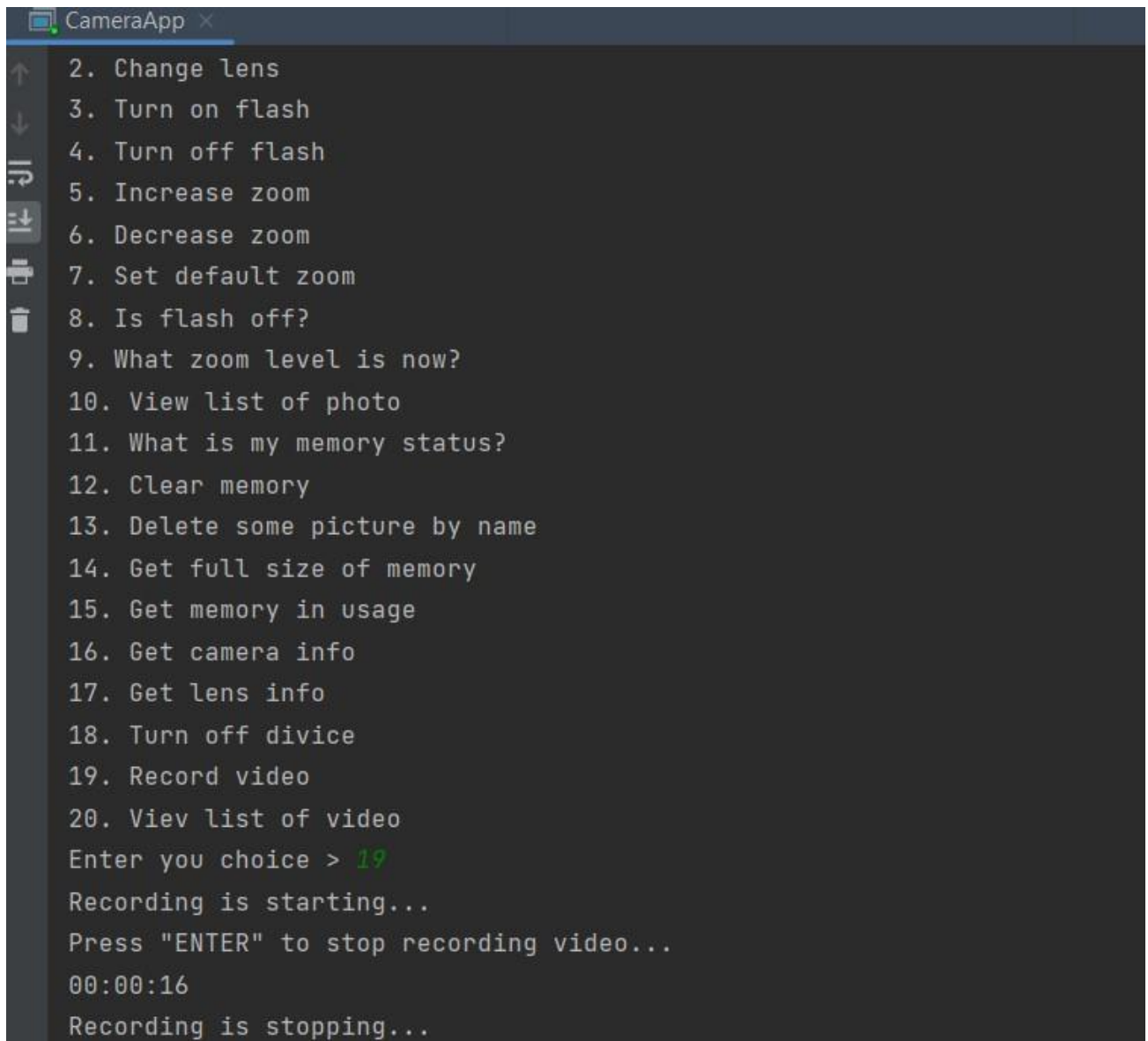
CMOS

60

10000

MicroSD

Результат виконання програми



```
CameraApp x
2. Change lens
3. Turn on flash
4. Turn off flash
5. Increase zoom
6. Decrease zoom
7. Set default zoom
8. Is flash off?
9. What zoom level is now?
10. View list of photo
11. What is my memory status?
12. Clear memory
13. Delete some picture by name
14. Get full size of memory
15. Get memory in usage
16. Get camera info
17. Get lens info
18. Turn off device
19. Record video
20. View list of video
Enter you choice > 19
Recording is starting...
Press "ENTER" to stop recording video...
00:00:16
Recording is stopping...
```

Рис.1. Фрагмент із результату виконання програми

Відповіді на контрольні запитання

1. Синтаксис реалізації спадкування.

```
class Підклас extends Суперклас {  
    Додаткові поля і методи }
```

2. Що таке суперклас та підклас?

Найчастіше супер-клас – це базовий клас, а підклас – це похідний клас від суперкласу.

3. Як звернутися до членів суперкласу з підкласу?

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. Коли використовується статичне зв'язування при виклику методу?

Механізм статичного зв'язування передбачає визначення методу, який необхідно викликати, на етапі компіляції.

5. Як відбувається динамічне зв'язування при виклику методу?

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. Що таке абстрактний клас та як його реалізувати?

Абстрактні класи призначені бути основою для розробки ієрархій класів та не дозволяють створювати об'єкти свого класу. Вони реалізуються за допомогою ключового слова `abstract`. На відміну від звичайних класів абстрактні класи можуть містити абстрактні методи (а можуть і не містити).

7. Для чого використовується ключове слово `instanceof`?

Оператор `instanceof` дозволяє перевірити приналежність об'єкта до зазначеного класу з урахуванням успадкування.

8. Як перевірити чи клас є підкласом іншого класу?

1) Можна перевірити за допомогою ключового слова **`instanceof`**

2) Перевірити за допомогою метода `isAssignableFrom()`, наприклад

```
public class Test {  
  
    public class A {}  
  
    public class B extends A {}  
  
    public class C {}  
  
    public static void main(String[] args) {  
        System.out.println("B extends A : " + A.class.isAssignableFrom(B.class));  
        System.out.println("C  
        extends A : " + A.class.isAssignableFrom(C.class));  
    }  
}
```

Результат:

```
B extends A : true  
C  
extends A : false
```

9. Що таке інтерфейс?

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. Як оголосити та застосувати інтерфейс?

Синтаксис оголошення інтерфейсів:

```
[public] interface НазваІнтерфейсу {
```

```
    Прототипи методів та оголошення констант інтерфейсу
```

```
}
```

Для застосування інтерфейсу потрібно оголосити за допомогою ключового слова `implements`, що клас реалізує інтерфейс. Якщо клас реалізує кілька інтерфейсів, то вони перелічуються через кому після ключового слова `implements`.

Висновок

Під час виконання даної лабораторної роботи я ознайомився з процесом процесом спадкування, поліморфізму та створення абстрактного класу. Розробив предметну область згідно свого варіанту (відеокамера), ознайомився з усіма принципами ООП – інкапсуляція даних, поліморфізм, наслідування. Зрозумів наскільки ООП є корисним інструментом при підтримці проєкту та його розширенні.