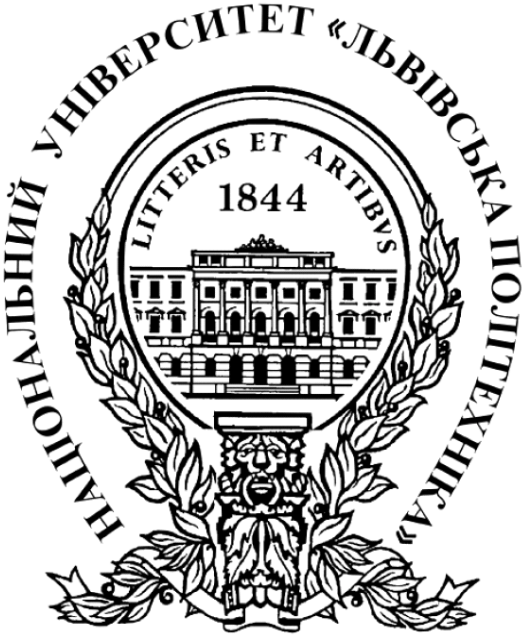
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Інститут комп'ютерних технологій, автоматики та метрології

Кафедра ЕОМ



Звіт
До лабораторної роботи №4
З дисципліни: «Кросплатформні засоби програмування»
На тему «Спадкування та інтерфейси»
Варіант №26

Виконав: ст. гр. КІ-36
Бродський Б.А.
Перевірив: доцент кафедри ЕОМ
Іванов Ю.С.

Львів – 2022

Мета: 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 №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.

26. Моторний човен

Виконання:

Лістинг програми:

```
public interface GetInOut
{
    void GetIn(int count);
    void GetOut(int count);
}
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * Class Logger. Was created to log information, errors and warnings. Also there
was implemented Singelton
 * @author
 * @version 1.0
 */
public class Logger
{
    private static Logger logger;
    private final String fileName;

    protected final String infoFlag = new String("[INFO] ");
    protected final String errorFlag = new String("[ERROR] ");
    protected final String warningFlag = new String("[WARNING] ");

    /**
     * Constructor
     * @param fileName
     */
    private Logger(String fileName)
    {
        this.fileName = fileName;
        File loggerFile = null;
```

```
        FileWriter fout = null;
        try
        {
            loggerFile = new File(fileName);
            fout = new FileWriter(loggerFile, true);
            SimpleDateFormat formatter= new SimpleDateFormat("yyyy-MM-dd 'at'
HH:mm:ss z");
            Date date = new Date(System.currentTimeMillis());
            fout.write("[" + formatter.format(date) + "] " + "Logger start to
work\n");
        }
        catch (IOException e)
        {
            System.err.println("Something wrong with log file" +
e.getMessage());
            System.exit(1);
        }
        finally
        {
            try
            {
                fout.flush();
                fout.close();
            }
            catch (IOException e)
            {
                System.out.println(e.getMessage());
            }
        }
    }

    /**
     * Method to do logging
     * @param massege
     */
    public void log(String massege)
    {
        File loggerFile = null;
        FileWriter fout = null;
        try
        {
            loggerFile = new File(this.fileName);
            fout = new FileWriter(loggerFile, true);
            SimpleDateFormat formatter= new SimpleDateFormat("yyyy-MM-dd 'at'
HH:mm:ss z");
            Date date = new Date(System.currentTimeMillis());
            fout.write("[" + formatter.format(date) + "] " + massege +  "\n");
        }
        catch (IOException e)
        {
            System.err.println("Something wrong with log file" +
e.getMessage());
            System.exit(1);
        }
        finally
        {
            try
            {
                fout.flush();
```

```
                fout.close();
            }
            catch (IOException | NullPointerException e)
            {
                System.out.println(e.getMessage());
            }
        }
    }

    /**
     * Singleton implementation
     * @param fileName
     * @return
     */
    public static Logger getLogger(String fileName)
    {
        if (logger == null)
        {
            logger = new Logger(fileName);
        }
        return logger;
    }

    /**
     * Getter for logger
     * @return logger
     */
    public static Logger getLogger()
    {
        return logger;
    }

}
public class Main {
    public static void main(String[] args) {
        MotorBoat motorBoat = new MotorBoat(
                    new Oar(34.6, "plastic"),
                    new Pump("Electric", 60),
                    new Seat(6, "plastic"),
                    "black", 900
        );

        motorBoat.PumpBoat();
        motorBoat.PumpBoat();

        motorBoat.GetIn(4);
        motorBoat.GetIn(4);
        motorBoat.GetOut(4);

        System.out.println(motorBoat);
    }
}
public class MotorBoat extends RowingBoat implements GetInOut
{
    /**
     * Constructor
     *
     * @param oar
     * @param pump
```

```
     * @param seat
     * @param color
     * @param volume
     */
    public MotorBoat(Oar oar, Pump pump, Seat seat, String color, double volume)
{
        super(oar, pump, seat, color, volume);
        logger.log(logger.infoFlag + "Motor constructor called");
    }

    @Override
    public void GetIn(int count) {
        if(seat.getCapacity() >= seat.getBookedSeats() + count)
        {
            System.out.println("In boat seat " + count + " persons");
            seat.setBookedSeats(seat.getBookedSeats() + count);
            System.out.println("Now in boat " + seat.getBookedSeats() + " booked
seats");
        }
        else
        {
            System.out.println("Boat did not have enough seats");
        }
        logger.log(logger.infoFlag + "Motorboat GetIn method called");
    }

    @Override
    public void GetOut(int count) {
        if(seat.getBookedSeats() - count >= 0)
        {
            System.out.println("From boat get out  " + count + " persons");
            seat.setBookedSeats(seat.getBookedSeats() - count);
            System.out.println("Now in boat " + seat.getBookedSeats() + " booked
seats");
        }
        else
        {
            System.out.println("Boat did not have " + count + " persons");
        }
        logger.log(logger.infoFlag + "Motorboat GetOut method called");
    }

    /**
     * Overrided method to pump boat
     */
    @Override
    public void PumpBoat() {
        if (!isPumped)
        {
            System.out.println("Boat will be pumped for " + volume /
pump.getPower() + " minutes");
            isPumped = true;
        }
        else
        {
            System.out.println("Boat is already pumped");
        }
        logger.log(logger.infoFlag + "RowingBoat method PumpBoat called");
    }
```

```
    @Override
    public String toString() {
        return "MotorBoat:" +
                "\noar=" + oar +
                "\npump=" + pump +
                "\nseat=" + seat +
                "\ncolor='" + color + '\'' +
                "\nvolume=" + volume +
                "\nisPumped=" + isPumped;
    }
}
public class Oar
{
    private double length;
    private String material;

    public Oar(double length, String material) {
        this.length = length;
        this.material = material;
    }

    public double getLength() {
        return length;
    }

    public void setLength(double length) {
        this.length = length;
    }

    public String getMaterial() {
        return material;
    }

    public void setMaterial(String material) {
        this.material = material;
    }

    @Override
    public String toString() {
        return "Oar{ " +
                "length = " + length +
                ", material = '" + material + '\'' +
                " }";
    }
}
public class Pump
{
    private String type;
    private double power;

    public Pump(String type, double power) {
        this.type = type;
        this.power = power;
    }

    public String getType() {
        return type;
    }
```

```
    public void setType(String type) {
        this.type = type;
    }

    public double getPower() {
        return power;
    }

    public void setPower(double power) {
        this.power = power;
    }

    @Override
    public String toString() {
        return "Pump{ " +
                "type = '" + type + '\'' +
                ", power = " + power +
                " }";
    }
}
/**
 * Class RowingBoar
 * @author
 * @version 1.0
 */
public abstract class RowingBoat
{
    protected Oar oar;
    protected Pump pump;
    protected Seat seat;
    protected String color;
    protected double volume;
    protected boolean isPumped = false;
    protected Logger logger = Logger.getLogger("logs.txt");

    /**
     * Constructor
     * @param oar
     * @param pump
     * @param seat
     * @param color
     * @param volume
     */
    public RowingBoat(Oar oar, Pump pump, Seat seat, String color, double
volume) {
        this.oar = oar;
        this.pump = pump;
        this.seat = seat;
        this.color = color;
        this.volume = volume;
        logger.log(logger.infoFlag + "RowingBoat constructor called");
    }

    /**
     * Pump boat method
     */
    public abstract void PumpBoat();
```

```
    public Oar getOar() {
        return oar;
    }

    public void setOar(Oar oar) {
        this.oar = oar;
    }

    public Pump getPump() {
        return pump;
    }

    public void setPump(Pump pump) {
        this.pump = pump;
    }

    public Seat getSeat() {
        return seat;
    }

    public void setSeat(Seat seat) {
        this.seat = seat;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public double getVolume() {
        return volume;
    }

    public void setVolume(double volume) {
        this.volume = volume;
    }

    public boolean isPumped() {
        return isPumped;
    }

    public void setPumped(boolean pumped) {
        isPumped = pumped;
    }

    @Override
    public String toString() {
        return "RowingBoat:" +
                "\noar = " + oar +
                "\npump = " + pump +
                "\nseat = " + seat +
                "\ncolor = '" + color + '\'' +
                "\nvolume = " + volume +
                "\nisPumped = " + isPumped;
    }
}
```

```java
public class Seat
{
    private int capacity;
    private String material;
    private int bookedSeats = 0;

    public Seat(int capacity, String material) {
        this.capacity = capacity;
        this.material = material;
    }

    public int getCapacity() {
        return capacity;
    }

    public void setCapacity(int capacity) {
        this.capacity = capacity;
    }

    public String getMaterial() {
        return material;
    }

    public void setMaterial(String material) {
        this.material = material;
    }

    public int getBookedSeats() {
        return bookedSeats;
    }

    public void setBookedSeats(int bookedSeats) {
        this.bookedSeats = bookedSeats;
    }

    @Override
    public String toString() {
        return "Seat{ " +
                "capacity = " + capacity +
                ", material = '" + material + '\'' +
                ", bookedSeats = " + bookedSeats +
                " }";
    }
}
```

Результати:

```
Boat will be pumped for 15.0 minutes
Boat is already pumped
In boat seat 4 persons
Now in boat 4 booked seats
Boat did not have enough seats
From boat get out  4 persons
Now in boat 0 booked seats
MotorBoat:
oar=Oar{ length = 34.6, material = 'plastic' }
pump=Pump{ type = 'Electric', power = 60.0 }
seat=Seat{ capacity = 6, material = 'plastic', bookedSeats = 0 }
color='black'
volume=900.0
isPumped=true

Process finished with exit code 0
```

Висновок: ознайомився з спадкуванням та інтерфейсами у мові Java.