Міністерство освіти і науки України

Національний університет "Львівська політехніка"

Звіт з лабораторної роботи №3

з курсу “Кросплатформні засоби програмування”

Спадкування та інтерфейси

Виконав: студент гр. KI-306

Шаповал Віталій

Прийняв: к.т.н. Олексів М.В.

Львів 2024 р.

ТЕОРЕТИЧНІ ВІДОМОСТІ

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.

Варіант № 28 Смартлампа

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.

Файл ./ce306/shapoval/lab3/Colorable.java:

```
package ce306.shapoval.lab3;

/**
 * Інтерфейс Colorable для роботи з кольором світла лампи
 */
public interface Colorable {
    /**
     * Метод для встановлення кольору світла лампи за RGB-кодом
     * @param r Червона складова (0-255)
     * @param g Зелена складова (0-255)
     * @param b Синя складова (0-255)
     */
```

```
        void setColor(int r, int g, int b);

        /**
         * Метод для отримання поточного кольору світла лампи у вигляді рядка з
RGB-кодом
         * @return Поточний колір у форматі RGB
         */
        String getColor();
    }
```

Файл ./ce306/shapoval/lab3/Lamp.java:

```
    /**
    * lab 3 package
    */
    package ce306.shapoval.lab3;

    import java.io.*;

    /**
    * Class <code>Lamp</code> implements lamp details
    */
    public abstract class Lamp {
        //private Object battery_;
        protected LampBattery battery;
        protected LampSocket socket;
        protected PrintWriter fout;

        /**
        * Default constructor
        * @throws FileNotFoundException
        */
        public Lamp() throws FileNotFoundException {
            battery = null;
            socket = new LampSocket();
            fout = new PrintWriter(new File("lamp_log.txt"));
        }

        /**
        * Parameterized constructor
        * @param battery <code>LampBattery</code> object
        * @param type <code>LampType</code> object
        * @param socket <code>LampSocket</code> object
        * @throws FileNotFoundException
        */
              public Lamp(LampBattery battery, LampSocket socket) throws
FileNotFoundException {
            this.battery = battery;
            this.socket = socket;
            fout = new PrintWriter(new File("lamp_log.txt"));
        }

        /**
```

```
        * Method returns lamp's information
        * @return Lamp's information
        */
        public String getLampInfo() {
            String info = "Лампа:\n";
            if (battery != null) {
                info += battery.checkStatus() + "\n";
            }
            info += socket.getSocketInfo();

            fout.println(info);
            fout.flush();
            return info;
        }

        /**
        * Method charges the battery
        * @param amount The amount to charge
             *  @return  <code>true</code>  if  the  charge  was  successful,
<code>false</code> otherwise
        */
        public boolean chargeBattery(double amount) {
            if(battery != null) {
                battery.charge(amount);
                fout.printf("Батарея була заряджена на %.2f мАг.\n", amount);
                fout.flush();
                return true;
            }
            fout.println("Батарея не існує");
            fout.flush();
            return false;
        }

        /**
        * Method discharges the battery
        * @param amount The amount to discharge
            *  @return  <code>true</code>  if  the  discharge  was  successful,
<code>false</code> otherwise
        */
        public boolean dischargeBattery(double amount) {
            if (battery != null) {
                battery.discharge(amount);
                fout.printf("Батарея була розряджена на %.2f мАг.\n", amount);
                fout.flush();
                return true;
            }
            fout.println("Батарея не існує");
            fout.flush();
            return false;
        }

        /**
        * Method returns the current battery capacity
```

```
        * @return Current battery capacity
        */
        public double getCurrentBatteryCapacity() {
            if (battery == null) {
                fout.println("Батарея не існує");
                fout.flush();
                return 0;
            }

                            fout.println("Значення потужності батареєю: " +
battery.getCurrentCapacity());
            fout.flush();
            return battery.getCurrentCapacity();
        }

        /**
        * Method checks the status of the lamp
        * @return Current status of the lamp
        */
        public String checkLampStatus() {
            if (battery == null) {
                fout.println("Батарея не існує");
                fout.flush();
                return "Батарея не існує";
            }

            String status = battery.checkStatus();
            fout.println("Стан лампи перевірено: " + status);
            fout.flush();
            return status;
        }

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

    }
```

Файл ./ce306/shapoval/lab3/LampBattery.java:

```
    package ce306.shapoval.lab3;

    /**
     * The <code>LampBattery</code> class represents the battery used in a lamp.
     * It includes fields for nominal voltage, nominal capacity, and current
capacity,
     * and provides methods to charge and discharge the battery, as well as to
check its status.
     */
    public class LampBattery {
```

```
        private double nominalVoltage;   // The nominal voltage of the battery (in
volts)
        private double nominalCapacity;   // The nominal capacity of the battery
(in mAh)
        private double currentCapacity;   // The current capacity of the battery
(in mAh)

        /**
         * Default constructor that initializes the battery with default values.
         * Nominal voltage is set to 3.7V and capacity to 1500mAh.
         * The current capacity is set to be fully charged.
         */
        public LampBattery() {
            nominalVoltage = 3.7;
            nominalCapacity = 1500;
            currentCapacity = nominalCapacity;
        }

        /**
         * Constructor that initializes the battery with specified nominal voltage
and capacity.
           * The current capacity is set to match the nominal capacity (fully
charged).
         *
         * @param nominalVoltage The nominal voltage of the battery (in volts)
         * @param nominalCapacity The nominal capacity of the battery (in mAh)
         */
        public LampBattery(double nominalVoltage, double nominalCapacity) {
            this.nominalVoltage = nominalVoltage;
            this.nominalCapacity = nominalCapacity;
            this.currentCapacity = nominalCapacity;
        }

        /**
         * Constructor that initializes the battery with specified nominal voltage,
         * nominal capacity, and current capacity.
         *
         * @param nominalVoltage The nominal voltage of the battery (in volts)
         * @param nominalCapacity The nominal capacity of the battery (in mAh)
         * @param currentCapacity The current capacity of the battery (in mAh)
         */
        public LampBattery(double nominalVoltage, double nominalCapacity, double
currentCapacity) {
            this.nominalVoltage = nominalVoltage;
            this.nominalCapacity = nominalCapacity;
            this.currentCapacity = currentCapacity;
        }

        /**
         * Charges the battery by the specified amount.
         * The battery cannot be overcharged beyond its nominal capacity.
         *
         * @param amount The amount to charge the battery by (in mAh)
```

```
         */
        public void charge(double amount) {
            if (amount < 0) {
                  System.out.println("Cannot charge the battery with a negative
value.");
                return;
            }
            currentCapacity += amount;
            if (currentCapacity > nominalCapacity) {
              currentCapacity = nominalCapacity; // Limit the battery capacity to
the nominal value
            }
        }

        /**
         * Discharges the battery by the specified amount.
         * The battery cannot have a negative capacity.
         *
         * @param amount The amount to discharge the battery by (in mAh)
         */
        public void discharge(double amount) {
            if (amount < 0) {
                System.out.println("Cannot discharge the battery with a negative
value.");
                return;
            }
            currentCapacity -= amount;
            if (currentCapacity < 0) {
                currentCapacity = 0; // Prevent the capacity from becoming negative
            }
        }

        /**
         * Returns the nominal voltage of the battery.
         *
         * @return The nominal voltage (in volts)
         */
        public double getNominalVoltage() {
            return nominalVoltage;
        }

        /**
         * Returns the nominal capacity of the battery.
         *
         * @return The nominal capacity (in mAh)
         */
        public double getNominalCapacity() {
            return nominalCapacity;
        }

        /**
         * Returns the current capacity of the battery.
         *
```

```
         * @return The current capacity (in mAh)
         */
        public double getCurrentCapacity() {
            return currentCapacity;
        }

        /**
         * Checks the status of the battery based on its current capacity.
         *
         * @return A string indicating the current status of the battery:
         *         - "Battery is discharged" if the capacity is 0
        *        - "Battery is almost discharged" if the capacity is less than 20%
of nominal
         *         - "Battery is in normal condition" otherwise
         */
        public String checkStatus() {
            if (currentCapacity == 0) {
                return "Battery is discharged.";
            } else if (currentCapacity < nominalCapacity * 0.2) {
                return "Battery is almost discharged.";
            } else {
                return "Battery is in normal condition.";
            }
        }
    }
```

Файл ./ce306/shapoval/lab3/LampSocket.java:

```
    package ce306.shapoval.lab3;

    /**
     * The <code>LampSocket</code> class represents the socket (or base) of a
lamp,
     * which connects the lamp to the electrical supply.
     * It includes attributes such as the type, shape, and diameter of the socket.
     */
    public class LampSocket {

        // Fields to store the properties of the socket
        private String type;      // The type of the socket (e.g., E5, G24, B22d)
       private String shape;    // The shape or connection type (e.g., threaded,
bi-pin)
        private double diameter;  // The diameter of the socket in millimeters

        /**
        * Default constructor initializing with default values (type E5, threaded
connection).
         * The diameter is set to 5mm.
         */
        public LampSocket() {
            this.type = "E5";
            this.diameter = 5;
            this.shape = "threaded connection";
        }
```

```
        /**
         * Constructor to initialize the socket with specific type, shape, and
diameter.
         *
         * @param type The type of the socket (e.g., E27, G24)
        * @param shape The connection type or shape of the socket (e.g., threaded,
bi-pin)
         * @param diameter The diameter of the socket in millimeters
         */
        public LampSocket(String type, String shape, double diameter) {
            this.type = type;
            this.diameter = diameter;
            this.shape = shape;
        }

        /**
         * Returns the type of the socket.
         *
         * @return A string representing the socket type
         */
        public String getType() {
            return type;
        }

        /**
         * Returns the diameter of the socket in millimeters.
         *
         * @return The diameter of the socket
         */
        public double getDiameter() {
            return diameter;
        }

        /**
         * Returns the shape or connection type of the socket.
         *
         * @return A string representing the socket's shape
         */
        public String getShape() {
            return shape;
        }

        /**
         * Sets the type of the socket.
         *
         * @param type The new type of the socket
         */
        public void setType(String type) {
            this.type = type;
        }

        /**
```

```
         * Sets the diameter of the socket in millimeters.
         *
         * @param diameter The new diameter of the socket
         */
        public void setDiameter(double diameter) {
            this.diameter = diameter;
        }

        /**
         * Sets the shape or connection type of the socket.
         *
         * @param shape The new shape of the socket
         */
        public void setShape(String shape) {
            this.shape = shape;
        }

        /**
          * Displays detailed information about the socket, including its type,
shape, and diameter.
         *
         * @return A formatted string containing socket details
         */
        public String getSocketInfo() {
            return "Socket Type: " + type + "\n" +
                    "Socket Shape: " + shape + "\n" +
                    "Socket Diameter: " + diameter + " mm\n";
        }
    }
```

Файл ./ce306/shapoval/lab3/SmartLamp.java:

```
    package ce306.shapoval.lab3;

    import java.io.FileNotFoundException;

    /**
       *   The   <code>SmartLamp</code>   class   extends   the   abstract   class
<code>Lamp</code> and implements the <code>Colorable</code> interface.
     */
    public final class SmartLamp extends Lamp implements Colorable {
        // Fields to store color components
        protected int red, green, blue;

        /**
         * Constructor that initializes the lamp with a given battery and socket.
         */
            public  SmartLamp(LampBattery  battery,  LampSocket  socket)  throws
FileNotFoundException {
            super(battery, socket);
            // Default light color is white (255, 255, 255)
            this.red = 255;
            this.green = 255;
            this.blue = 255;
```

```
        }

        /**
         * Implementation of the method to change the light color.
         * @param r Red component (0-255)
         * @param g Green component (0-255)
         * @param b Blue component (0-255)
         */
        @Override
        public void setColor(int r, int g, int b) {
            this.red = r;
            this.green = g;
            this.blue = b;
            System.out.printf("The lamp color has been changed to RGB(%d, %d, %d)\
n", r, g, b);
        }

        /**
         * Implementation of the method to retrieve the current color.
         * @return The current color in RGB format.
         */
        @Override
        public String getColor() {
          return String.format("The current lamp color is: RGB(%d, %d, %d)", red,
green, blue);
        }

        /**
         * Implementation of the abstract method <code>getLampInfo</code>.
         * @return Information about the lamp and its color.
         */
        @Override
        public String getLampInfo() {
            String info = super.getLampInfo();
            info += getColor();

            fout.println(info);
            fout.flush();
            return info;
        }
    }
```

Файл ./SmartLampApp.java:

```
    import ce306.shapoval.lab3.*;
    import java.io.FileNotFoundException;
    import java.util.Scanner;

    /**
     * The <code>SmartLampApp</code> class is a driver class that tests the
functionality
     * of the <code>SmartLamp</code> class. It provides a console menu for users
to interact
```

```
     * with a smart lamp object, allowing them to change the color, charge and
discharge the battery,
     * check the status, and retrieve lamp information.
     */
    public class SmartLampApp {
        /**
             * The entry point of the application. It initializes the
<code>SmartLamp</code> object
         * and displays a menu for user interaction. The user can choose from
various options
        * such as getting lamp information, changing the lamp color, charging the
battery,
         * discharging the battery, checking the battery status, and exiting the
application.
         *
         * @param args Command line arguments (not used in this application)
         */
        public static void main(String[] args) {
            // Scanner to read user input
            Scanner scanner = new Scanner(System.in);

            // Creating SmartLamp object
            SmartLamp smartLamp = null;
            try {
                smartLamp = new SmartLamp(new LampBattery(), new LampSocket());
            } catch (FileNotFoundException e) {
                   System.out.println("Error: Unable to create file for logging
data.");
                return;
            }

            boolean exit = false;

            while (!exit) {
                // Displaying menu options
                System.out.println("===== Smart Lamp Menu =====");
                System.out.println("1. Get lamp information");
                System.out.println("2. Change lamp color");
                System.out.println("3. Charge battery");
                System.out.println("4. Discharge battery");
                System.out.println("5. Check battery status");
                System.out.println("6. Check current battery capacity");
                System.out.println("7. Exit");
                System.out.print("Enter your choice: ");

                int choice = scanner.nextInt();
                switch (choice) {
                    case 1:
                        System.out.println(smartLamp.getLampInfo());
                        break;
                    case 2:
                        System.out.print("Enter RGB values (r g b): ");
                        int r = scanner.nextInt();
```

```
                        int g = scanner.nextInt();
                        int b = scanner.nextInt();
                        smartLamp.setColor(r, g, b);
                        break;
                    case 3:
                        System.out.print("Enter charge amount: ");
                        double chargeAmount = scanner.nextDouble();
                        if (smartLamp.chargeBattery(chargeAmount)) {
                            System.out.println("Battery charged.");
                        } else {
                            System.out.println("Failed to charge battery.");
                        }
                        break;
                    case 4:
                        System.out.print("Enter discharge amount: ");
                        double dischargeAmount = scanner.nextDouble();
                        if (smartLamp.dischargeBattery(dischargeAmount)) {
                            System.out.println("Battery discharged.");
                        } else {
                            System.out.println("Failed to discharge battery.");
                        }
                        break;
                    case 5:
                                        System.out.println("Battery  status:  "  +
smartLamp.checkLampStatus());
                        break;
                    case 6:
                                 System.out.println("Current battery capacity: " +
smartLamp.getCurrentBatteryCapacity());
                        break;
                    case 7:
                        System.out.println("Exiting.");
                        exit = true;
                        break;
                    default:
                        System.out.println("Invalid choice.");
                }
            }

            // Releasing resources
            smartLamp.dispose();
            scanner.close();
        }
    }
```

Виконання програми:

Термінал:

===== Smart Lamp Menu =====

1. Get lamp information

2. Change lamp color

3. Charge battery

4. Discharge battery

5. Check battery status

6. Check current battery capacity

7. Exit

Enter your choice: 1

Лампа:

Battery is in normal condition.

Socket Type: E5

Socket Shape: threaded connection

Socket Diameter: 5.0 mm

The current lamp color is: RGB(255, 255, 255)

===== Smart Lamp Menu =====

1. Get lamp information

2. Change lamp color

3. Charge battery

4. Discharge battery

5. Check battery status

6. Check current battery capacity

7. Exit

Enter your choice: 2

Enter RGB values (r g b): 0 0 0

The lamp color has been changed to RGB(0, 0, 0)

===== Smart Lamp Menu =====

1. Get lamp information

2. Change lamp color

3. Charge battery

4. Discharge battery

5. Check battery status

6. Check current battery capacity

7. Exit

Enter your choice: 1

Лампа:

Battery is in normal condition.

Socket Type: E5

Socket Shape: threaded connection

Socket Diameter: 5.0 mm

The current lamp color is: RGB(0, 0, 0)

===== Smart Lamp Menu =====

1. Get lamp information

2. Change lamp color

3. Charge battery

4. Discharge battery

5. Check battery status

6. Check current battery capacity

7. Exit

Enter your choice: 7

Exiting.

Вміст Лог файлу:

Лампа:

Battery is in normal condition.

Socket Type: E5

Socket Shape: threaded connection

Socket Diameter: 5.0 mm

Лампа:

Battery is in normal condition.

Socket Type: E5

Socket Shape: threaded connection

Socket Diameter: 5.0 mm

The current lamp color is: RGB(255, 255, 255)

Лампа:

Battery is in normal condition.

Socket Type: E5

Socket Shape: threaded connection

Socket Diameter: 5.0 mm

Лампа:

Battery is in normal condition.

Socket Type: E5

Socket Shape: threaded connection

Socket Diameter: 5.0 mm

The current lamp color is: RGB(0, 0, 0)

Висновок

На цій лабораторній роботі я поглибив свої знання з об'єктно-орієнтованого програмування на мові Java, зокрема навчився розширювати класи, що вже були реалізовані у попередніх лабораторних роботах, шляхом додавання нових функцій та реалізації інтерфейсів. Я зробив суперклас з лабораторної роботи №2 абстрактним, що дозволило створити більш гнучку та розширювану архітектуру. Розроблений підклас забезпечив коректне функціонування лампи з можливістю зміни кольору та роботи з батареєю, а також реалізував інтерфейс для зміни кольору лампи.

Особливу увагу було приділено організації коду в окремому пакеті, відповідно до структури пакунків Група.Прізвище.Lab3, а також доданню коментарів, які дали змогу автоматично згенерувати документацію до пакету. Це дало змогу практично закріпити навички написання самодокументованого коду та роботи з інструментами генерації документації в Java.

Окрім того, я вдосконалив свої навички роботи з системами контролю версій Git, завантаживши проект на платформу GitHub згідно методичних вказівок. Це допомогло покращити вміння співпраці над проектами та ведення спільної розробки з використанням сучасних інструментів для контролю версій.