Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ

Звіт

З лабораторної роботи №3

Варіант – 10

З дисципліни: «Кросплатформні засоби програмування»

На тему: «Спадкування та інтерфейси»

Виконав: ст. гр. КІ-306

Згурський Т.С.

Прийняв:

доцент кафедри ЕОМ

Іванов Ю. С.

Львів 2023

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

Завдання (варіант № 10)

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. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації.
4. Дати відповідь на контрольні запитання

Вихідний код програми

houseapp:

```
package org.example;


/**
 * The {@code HouseApp} class serves as a simple driver program to test the House and OfficeCenter classes.
 *
 * @version 1.0
 */
public class HouseApp {
    public static void main(String[] args) {
        // Create an office center
        OfficeCenter officeCenter = new OfficeCenter("789 Oak St.", 5, 500, true, 50, true, true) {
            @Override
            public void addWhiteboard2(boolean hasWhiteboard) {
                // An overridden method, but currently, it does nothing (empty body).
            }
        };

        // Display office center details
        officeCenter.displayDetails();

        // Update office center details
```

```
        officeCenter.setNumberOfFloors(6);
        officeCenter.setAddress("101 Pine St.");
        officeCenter.setOfficeSpace(600);
        officeCenter.equipMeetingRoom(false, true);
        officeCenter.addDesks(10);

        // Display updated office center details
        officeCenter.displayDetails();

        // The next line calls the toString method, but its result is not used or displayed.
        officeCenter.toString();
    }
}
```

officecenter:

```
package org.example;

import java.io.*;

/**
 * The {@code OfficeCenter} class represents an office center, extending the {@code House}
 * abstract class and implementing the {@code OfficeInterface} and {@code OfficeInterface2} interfaces.
 * It adds specific functionality for an office center, including office space allocation and office equipment management.
 *
 * @version 1.0
 */
public abstract class OfficeCenter extends House implements OfficeInterface, OfficeInterface2 {

    // Fields representing the state of the office center
    private int officeSpace;  // Represents the amount of office space in square meters
    private boolean hasMeetingRoom;
```

```
    private int numberOfDesks;
    private boolean hasProjector;
    private boolean hasWhiteboard;

    /**
     * Constructor for creating an OfficeCenter object with specified parameters.
     *
     * @param address         The address of the office center.
     * @param numberOfFloors  The number of floors in the office center.
     * @param officeSpace     The amount of office space in square meters.
     * @param hasMeetingRoom  Indicates whether the office center has a meeting room.
     * @param numberOfDesks   The number of desks in the office center.
     * @param hasProjector    Indicates whether the office center has a projector.
     * @param hasWhiteboard   Indicates whether the office center has a whiteboard.
     */
    public OfficeCenter(String address, int numberOfFloors, int officeSpace,
                        boolean hasMeetingRoom, int numberOfDesks, boolean hasProjector, boolean
hasWhiteboard) {
        super(address, numberOfFloors);
        this.officeSpace = officeSpace;
        this.hasMeetingRoom = hasMeetingRoom;
        this.numberOfDesks = numberOfDesks;
        this.hasProjector = hasProjector;
        this.hasWhiteboard = hasWhiteboard;
    }

    // Getter and setter methods for the additional fields

    /**
     * Gets the office space in square meters.
     *
     * @return The office space in square meters.
     */
    public int getOfficeSpace() {
        return officeSpace;
    }
```

```
/**
 * Sets the office space in square meters.
 *
 * @param officeSpace The office space to set in square meters.
 */
public void setOfficeSpace(int officeSpace) {
    this.officeSpace = officeSpace;
}

/**
 * Allocates additional office space.
 *
 * @param squareMeters The additional office space to allocate in square meters.
 */
public void allocateOfficeSpace(int squareMeters) {
    this.officeSpace += squareMeters;
}

/**
 * Equips the meeting room with a projector and/or whiteboard.
 *
 * @param projector  Indicates whether to equip a projector.
 * @param whiteboard Indicates whether to equip a whiteboard.
 */
public void equipMeetingRoom(boolean projector, boolean whiteboard) {
    this.hasMeetingRoom = true;
    this.hasProjector = projector;
    this.hasWhiteboard = whiteboard;
}

/**
 * Adds desks to the office center.
 *
 * @param desksToAdd The number of desks to add.
 */
public void addDesks(int desksToAdd) {
    this.numberOfDesks += desksToAdd;
```

```
}

/**
 * Removes desks from the office center.
 *
 * @param desksToRemove The number of desks to remove.
 */
public void removeDesks(int desksToRemove) {
    if (desksToRemove > this.numberOfDesks) {
        System.out.println("Cannot remove more desks than available.");
    } else {
        this.numberOfDesks -= desksToRemove;
    }
}

/**
 * Adds a projector to the office center.
 *
 * @param hasProjector Indicates whether to add a projector.
 */
public void addProjector(boolean hasProjector) {
    this.hasProjector = hasProjector;
    // Additional logic related to adding a projector
}

/**
 * Adds a whiteboard to the office center.
 *
 * @param hasWhiteboard Indicates whether to add a whiteboard.
 */
public void addWhiteboard(boolean hasWhiteboard) {
    this.hasWhiteboard = hasWhiteboard;
}

/**
 * Adds a projector to the office center and prints a message.
 *
```

```
     * @param hasProjector Indicates whether to add a projector.
     */
    public void addProjector2(boolean hasProjector) {
        System.out.println("Projector was added");
        this.hasProjector = true;
    }

    /**
     * Writes the address to a file named "Address.txt" and returns the address.
     *
     * @return The address of the office center.
     */
    public String writeAddressToFile() {
        String address = getAddress();

        try (FileWriter writer = new FileWriter("Address.txt")) {
            writer.write(address);
        } catch (IOException e) {
            System.err.println("Error writing to file: " + e.getMessage());
        }

        return address;
    }

    /**
     * Overrides the toString method to write the address to a file.
     *
     * @return The address of the office center.
     */
    @Override
    public String toString() {
        return writeAddressToFile();
    }
}
```

Результат виконання програми

```
Адреса будинку: 789 Oak St.
Кількість поверхів: 5
Наявність саду: Ні
Адреса будинку: 101 Pine St.
Кількість поверхів: 6
Наявність саду: Ні
Кількість офіс центрів з більше 5 поверхами: 1
```

Відповіді на контрольні запитання

1. Синтаксис реалізації спадкування.
- class МійКлас implements Інтерфейс { // тіло класу }

2. Що таке суперклас та підклас?
- суперклас - це клас, від якого інший клас успадковує властивості та методи.
Підклас - це клас, який успадковує властивості та методи від суперкласу.

3. Як звернутися до членів суперкласу з підкласу?
- super.назваМетоду([параметри]); // виклик методу суперкласу
-super.назваПоля; // звернення до поля суперкласу

4. Коли використовується статичне зв’язування при виклику методу?
- Статичне зв’язування використовується, коли метод є приватним, статичним, фінальним або конструктором. В таких випадках вибір методу відбувається на етапі компіляції.

5. Як відбувається динамічне зв’язування при виклику методу?
- вибір методу для виклику відбувається під час виконання програми на основі фактичного типу об’єкта. 6. Що таке абстрактний клас та як його реалізувати?
- це клас, який має один або більше абстрактних методів (методів без реалізації).
Щоб створити абстрактний клас, використовується ключове слово abstract.
Приклад: abstract class АбстрактнийКлас { abstract void абстрактнийМетод(); }

7. Для чого використовується ключове слово instanceof?
- для перевірки, чи об'єкт належить до певного класу або інтерфейсу.
Синтаксис: if (об'єкт instanceof Клас) { // код, який виконується, якщо об'єкт належить до класу }

8. Як перевірити чи клас є підкласом іншого класу?
- В Java використовується ключове слово extends, щоб вказати, що клас є підкласом іншого класу. Перевірити, чи один клас є підкласом іншого класу можна шляхом аналізу ієрархії успадкування.

9. Що таке інтерфейс?
- це абстрактний тип даних, який визначає набір методів, але не надає їх реалізацію. Всі методи інтерфейсу є загальнодоступними та автоматично є public. Інтерфейси використовуються для створення контрактів, які класи повинні реалізувати.

10. Як оголосити та застосувати інтерфейс?

- Для оголошення інтерфейсу використовується ключове слово interface.
Синтаксис: interface Інтерфейс { // оголошення методів та констант }
- Для застосування інтерфейсу в класі використовується ключове слово implements.
Синтаксис: class МійКлас implements Інтерфейс { // реалізація методів інтерфейсу }

Висновок: на даній лабораторній роботі ознайомитися з спадкуванням та інтерфейсами у мові Java.