

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

Миценко О. 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

Файл House.java

```
package K1306MytsenkoLab2;

import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDateTime;

/**
 * The <code>House</code> class represents a house and its operations.
 * It includes functionality for managing the number of floors, addresses,
 * gardens and gives information about house .
 * This class also logs events to a file named "Log.txt".
 *
 * @author Oleksandr Mytsenko
 * @version 1.0
 */

public class House {
    private FileWriter writer; // Поле для зберігання посилання на потік запису в файл

    private String address;
    private int numberOfFloors;
    private boolean hasGarden;

    /**
     * Default constructor for the house.
     *
     */
    // Конструктори
```

```

public House(){
    address = "No information";
    numberOfFloors = 0;
    hasGarden = false;
}

/**
 * Parameterized constructor for the house.
 *
 * @param address Specifies initial address.
 * @param numberOfFloors The initial number of floors.
 * @param hasGarden Specifies if the garden initially exists.
 */
public House(String address, int numberOfFloors, boolean hasGarden) {
    this.address = address;
    this.numberOfFloors = numberOfFloors;
    this.hasGarden = hasGarden;
}

/**
 * Constructs a house with the specified address and number of floors, defaulting to no garden.
 *
 * @param address The address of the house.
 * @param numberOfFloors The number of floors in the house.
 */
public House(String address, int numberOfFloors) {
    this(address, numberOfFloors, false);
}

/**
 * Constructs a house with the specified address and defaults to one floor and no garden.
 *
 * @param address The address of the house.
 */
public House(String address) {
    this(address, 1);
}

// Методи

// Метод для відкриття файлу для запису
/**
 * Open the file for writing
 */
public void openLogFile() {
    try {
        writer = new FileWriter("log.txt", true);
    } catch (IOException e) {
        System.err.println("Помилка при відкритті файлу для запису: " + e.getMessage());
    }
}

```

```
}
```

```
// Метод для закриття файлу після закінчення запису
```

```
/**
```

```
 * Close the file after ending of writing
```

```
 */
```

```
public void closeLogFile() {
```

```
    try {
```

```
        if (writer != null) {
```

```
            writer.close();
```

```
        }
```

```
    } catch (IOException e) {
```

```
        System.err.println("Помилка при закритті файлу: " + e.getMessage());
```

```
    }
```

```
}
```

```
/**
```

```
 * Display details of the house
```

```
 */
```

```
public void displayDetails() {
```

```
    System.out.println("Адреса будинку: " + address);
```

```
    System.out.println("Кількість поверхів: " + numberOfFloors);
```

```
    System.out.println("Наявність саду: " + (hasGarden ? "Так" : "Ні"));
```

```
}
```

```
/**
```

```
 * Log a message to the file.
```

```
 *
```

```
 * @param message The message to log.
```

```
 */
```

```
private void logMessage(String message) {
```

```
    try (FileWriter writer = new FileWriter("log.txt", true)) {
```

```
        LocalDateTime timestamp = LocalDateTime.now();
```

```
        writer.write("[ " + timestamp + " ] " + message + " - " + this + "\n");
```

```
    } catch (IOException e) {
```

```
        System.err.println("Помилка при записі до файлу: " + e.getMessage());
```

```
    }
```

```
}
```

```
/**
```

```
 * Set the number of floors for the house.
```

```
 *
```

```
 * @param numberOfFloors The number of floors to set.
```

```
 */
```

```
public void setNumberOfFloors(int numberOfFloors) {
```

```
    this.numberOfFloors = numberOfFloors;
```

```
    logMessage("Встановлено кількість поверхів: " + numberOfFloors);
```

```

}
/**
 * Set the address for the house.
 *
 * @param address The address to set.
 */
public void setAddress(String address) {
    this.address = address;
    logMessage("Оновлено адресу: " + address);
}
/**
 * Set whether the house has a garden or not.
 *
 * @param hasGarden True if the house has a garden, false otherwise.
 */
public void setHasGarden(boolean hasGarden) {

    this.hasGarden = hasGarden;
    logMessage("Оновлено інформацію про сад.");
}
/**
 * Add a floor to the house.
 */
public void addFloor() {

    numberOfFloors++;
    logMessage("Додано поверх.");
}
/**
 * Remove a floor from the house.
 * If the number of floors is already at the minimum, log a message accordingly.
 */
public void removeFloor() {
    if (numberOfFloors > 0) {
        numberOfFloors--;
        logMessage("Видалено поверх.");
    } else {
        logMessage("Не можна видалити поверх. Кількість поверхів вже мінімальна.");
    }
}
/**
 * Get the number of floors for the house.
 *
 * @return The number of floors.
 */
public int getNumberOfFloors() {
    logMessage("Дана інформація про кількість поверхів.");
    return numberOfFloors;
}

```

```

    }
    /**
     * Get the address of the house.
     *
     * @return The address.
     */
    public String getAddress() {
        logMessage("Дана інформація про адресу.");
        return address;
    }
    /**
     * Check if the house has a garden.
     *
     * @return True if the house has a garden, false otherwise.
     */
    public boolean hasGarden() {
        logMessage("Дана інформація про наявність саду.");
        return hasGarden;
    }
}

// Додаткові методи

}

```

Файл HouseDrive.java

```

package KI306MytsenkoLab2;

public class HouseDrive {
    public static void main(String[] args) {
        House house1 = new House("Вулиця Лінкольна, 123", 3, true);
        House house2 = new House("Вулиця Індепенденс, 456");
        House house3 = new House("Вулиця Кеннеді, 789", 2);

        house1.openLogFile();
        house2.openLogFile();
        house3.openLogFile();

        house1.displayDetails();
        house2.displayDetails();
        house3.displayDetails();

        house1.setAddress("Вулиця Нова, 555");
        house1.setNumberOfFloors(4);
        house1.addFloor();
        house1.setHasGarden(false);
        house1.removeFloor();

        house2.setNumberOfFloors(6);
    }
}

```

```

        house2.addFloor();
        house2.setHasGarden(true);
        house2.removeFloor();

        house3.setHasGarden(true);
        house3.addFloor();

        house1.displayDetails();
        house2.displayDetails();
        house3.displayDetails();

        house1.closeLogFile();
        house2.closeLogFile();
        house3.closeLogFile();
    }
}

```

Файл OfficeCenter.java

```
package K1306.Mytsenko.Lab3;
```

```
import K1306.Mytsenko.Lab3.*;
import K1306.Mytsenko.Lab3.OfficeInterface.*;
```

```

/**
 * The {@code OfficeCenter} class represents an office center, which extends the {@code House}
 * abstract class and implements the {@code OfficeFunctionality} interface. It adds specific
 * functionality for an office center, including office space allocation and office equipment management.
 *
 * @author Oleksandr Mytsenko
 * @version 1.0
 */
public class OfficeCenter extends House implements OfficeInterface {
    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;
        // Additional logic related to adding a whiteboard
    }

}

```

Файл OfficeInterface.java

```
package KI306.Mytsenko.Lab3;

public interface OfficeInterface {
    public int getOfficeSpace();
    public void setOfficeSpace(int officeSpace);
    public void allocateOfficeSpace(int squareMeters);
    public void equipMeetingRoom(boolean projector, boolean whiteboard);
    public void addDesks(int desksToAdd);
    public void removeDesks(int desksToRemove);
    public void addProjector(boolean hasProjector);
    public void addWhiteboard(boolean hasWhiteboard);
}
```

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
```

Фрагмент згенерованої документації

The screenshot shows a web browser window displaying the Java documentation for the `KI306.Mytsenko.Lab3` package. The browser's address bar shows the path `C:\Users\Oleksandr Mytsenko\Documents\CPPT\CPPT Labs\CPPT_Mytsenko_OS_KI-306_1\Lab3\Doc\KI306\Mytsenko\Lab3\package-summary.html`. The page title is "Package KI306.Mytsenko.Lab3". Below the title, there is a section for "All Classes and Interfaces" with tabs for "Interfaces" and "Classes". The "Classes" tab is selected, showing a list of classes: `House`, `HouseApp`, `OfficeCenter`, and `OfficeInterface`. Each class has a brief description. The `House` class is described as representing a house and its operations. The `HouseApp` class is described as providing a simple driver program to test the `House` and `OfficeCenter` classes. The `OfficeCenter` class is described as representing an office center, which extends the `House` abstract class and implements the `OfficeFunctionality` interface. The `OfficeInterface` class is listed but has no description. The browser window also shows a search bar and a "JavaScript is disabled on your browser" message. The Windows taskbar at the bottom shows the time as 11:30 PM on 9/22/2023, and the weather as 18°C Mostly cloudy.

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.