

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
Національний аерокосмічний університет ім. М.Є. Жуковського
“Харківський авіаційний інститут”

Кафедра комп’ютерних систем, мереж і кібербезпеки

Лабораторна робота № 7

з дисципліни “Технології програмування”

РОЗРОБЛЕННЯ JAVA-ЗАСТОСУНКІВ З ВИКОРИСТАННЯМ ОБ’ЄКТНО- ОРІЄНТОВАНОГО ПІДХОДУ.

ХАІ.503.525i1.22o.125, ПЗ

Виконав студент гр. 525i1 Проценко Д.І.
(№ групи) (П.І.Б.)

(підпис, дата)

Перевірів ст. викладач каф.503
(науковий ступінь, вчене звання, посада)

Здоровець Ю. В.

(підпис, дата)

(П.І.Б.)

Харків 2022

Тема роботи: Розроблення Java-застосунків з використанням об'єктно орієнтованого підходу.

Мета роботи:

1. Відпрацювати навички об'єктно-орієнтованого аналізу і розробки архітектури програмних систем.
2. Навчитися розробляти програми на мові Java з використанням об'єктно орієнтованого підходу.

Завдання 1.

Частина 1. Умови завдання

1. Порядок виконання роботи

1.1. Розробити архітектуру ПО, що реалізує моделювання системи управління компонентами «розумного будинку». Архітектура повинна бути представлена у вигляді наступних UML діаграм: діаграми способів використання (Use Case), діаграми класів і діаграми пакетів.

1.2. Функціональні вимоги до системи моделювання компонентів "розумного будинку":

- в системі має бути змодельоване не менше 5 типів компонент «розумного будинку», наприклад: світильники, побутова техніка, обігрів, жалюзі, сигналізація та ін. на розсуд студента;
- інтерфейс застосунку - консольний;
- керування застосунку - за допомогою системи консольних команд, які повинні включати наступні класи команд:
 - * команди конфігурації «розумного будинку», наприклад: «додати компонент типу А», «видалити компонент», «відобразити список компонентів» і ін.
 - * команди управління станом компонент «розумного будинку», наприклад, «включити світильник А1», «вимкнути холодильник» тощо.
 - * команди управління планувальником (таймером) для кожного пристрою, наприклад: «включити світильник А1 в xx: xx», «вимкнути пральну машину в xx: xx» і т.п.
 - * інші команди на розсуд студента. після кожної команди програма видає інформацію про успішність виконання команди і поточний стан всіх компонент «розумного будинку»;
- в процесі виконання програми вона видає в консоль інформацію про зміну станів компонент «розумного будинку».

Рис. 1 Умови завдання

1.3. Розробити java-програму, яка моделює роботу системи управління компонентами «розумного будинку».

1.4. Нефункціональні вимоги до програми:

- при розробці програми повинен використовуватися об'єктно-орієнтований підхід з підтримкою всіх можливостей ООП платформи Java;
- в програмі необхідно широко використовувати інтерфейси;
- можна використовувати патерни програмування і/або реалізації застосунку з віконним інтерфейсом.

2. Індивідуальне завдання

Відповідно до варіантів, зазначених нижче, система моделювання (управління) розумним будинком в складі мінімум 5 компонентів (як зазначено в п.1) повинна включати 1 обов'язковий компонент з розширеними можливостями управління. Наприклад, якщо варіант включає обов'язковий елемент - телевізор, то програмна модель компонента телевізор повинна включати всі можливі стани сутності «телевізор» і можливості для керування ним, наприклад: увімкнення / вимкнення, робота по таймеру, отримання списку каналів, перемикання каналів, пошук каналів, налаштування телевізора і т.д.

Перелік компонентів по варіантам:

11. Душова кабіна з гідромасажем

Рис. 2 Персональне завдання

Частина 2. Текст програми

```
import javax.management.InvalidAttributeValueException;
import java.io.*;
import java.security.InvalidKeyException;
import java.util.*;
import java.util.stream.Collectors;

public class Main {
    public static void main(String[] args) throws IOException, InvalidKeyException
    {
        //make a list of starter pack house objects
        List<House> objects = new ArrayList<>();
        Scanner s = new Scanner(System.in);
        int changeable = 0;
        int changeable2 = 0;
        CreateStarterPack(objects);
        Menu1(objects);
    }

    private static void Menu1(List<House> objects) throws IOException,
    InvalidKeyException {

        Scanner s = new Scanner(System.in);
        int changeable = 0;
        int changeable2 = 0;
        System.out.println("\nNext move:\n1.Add new object\n2.Delete object\n" +
            "3.Change condition of object\n4.Working at user
time\nelse.EXIT");
        changeable = s.nextInt();
        try {
            switch (changeable) {
                case 1:
```

```

        CreateNewObject(objects);
        break;
    case 2:
        System.out.print("Which one :");
        changeable2 = s.nextInt();
        if (changeable2 >= objects.size())
            throw new IllegalArgumentException("Illegal argument of
list: " + changeable2);
        objects.remove(changeable2-1);
        break;
    case 3:
        System.out.print("Which one :");
        changeable2 = s.nextInt();
        if (changeable2 > objects.size())
            throw new IllegalArgumentException("Illegal argument of
list: " + changeable2);
        ChangeObject(objects, changeable2 - 1);
        break;
    case 4:
        Time time1 = new Time(0,0);
        System.out.println("Time(int int): ");
        time1.setHours(s.nextInt());
        if (time1.getHours() > 23)
            throw new InvalidAttributeValueException("Wrong hours
value - " + time1.getHours());
        time1.setMinutes(s.nextInt());
        if (time1.getMinutes() > 59)
            throw new InvalidAttributeValueException("Wrong minutes
value - " + time1.getMinutes());

        List<House> Filtered_Objects = objects.stream()
            .filter(house -> house.isWithinWorkingHours(time1))
            .collect(Collectors.toList());

        System.out.println();
        int i = 1;
        for (House object : Filtered_Objects) {
            System.out.println(i++ + ": " + object);
        }
        Filtered_Objects.clear();
        Menu1(objects);
    default:
        System.exit(0);
}
} catch (IOException e) {
    throw new RuntimeException(e);
} catch (InvalidKeyException e) {
    throw new RuntimeException(e);
} catch (IllegalArgumentException e) {
    System.err.println("Error:" + e.getMessage());
} catch (InvalidAttributeValueException e) {
    throw new RuntimeException(e);
}

int i = 1;
System.out.println();
for (House object : objects) {
    System.out.println(i++ + ": " + object);
}
Menu1(objects);
}

private static void ChangeObject(List<House> objects, int num) throws
IOException {

```

```

House object = objects.get(num);
int changeable;
Scanner s = new Scanner(System.in);
Time time1 = new Time(0, 0);
/*
System.out.println("\nWhat
type:\n1.Fridge\n2.Blinds\n3.Lamp\n4.Conditioner\n5.Shower_cabin");

changeable = s.nextInt();
House house = null;
boolean active;
int Temperature;
boolean Shower;
Time time1 = new Time(0,0);
Time time2 = new Time(0,0);

switch (changeable)
{
    case 1:
        System.out.println("Change: 1.Temperature\n2.Change
condition\n3.Time On\n4.Time Off\nelse.Get back");
        changeable = s.nextInt();
        switch (changeable){
            case 1:
                System.out.print("Temperature: ");

            }

        }

    */

    try {
        if (object instanceof Fridge) {
            System.out.println("\nChange: \n1.Change condition\n2.Time
On\n3.Time Off\n4.Temperature\nelse.Get back");
            changeable = s.nextInt();
            switch (changeable) {
                case 1:
                    if (object.isActive())
                        object.ElementOff();
                    else
                        object.ElementOn();
                    System.out.println("***CHANGE COMPLETE***");
                    break;
                case 2:
                    System.out.println("Time(int int): ");
                    time1.setHours(s.nextInt());
                    if (time1.getHours() > 23)
                        throw new InvalidAttributeValueException("Wrong hors
value - " + time1.getHours());
                    time1.setMinutes(s.nextInt());
                    if (time1.getMinutes() > 59)
                        throw new InvalidAttributeValueException("Wrong
minutes value - " + time1.getMinutes());
                    object.onTime(time1);
                    break;
                case 3:
                    System.out.println("Time(int int): ");
                    time1.setHours(s.nextInt());
                    if (time1.getHours() > 23)
                        throw new InvalidAttributeValueException("Wrong hors
value - " + time1.getHours());
                    time1.setMinutes(s.nextInt());

```

```

        if (time1.getMinutes() > 59)
            throw new InvalidAttributeValueException("Wrong
minutes value - " + time1.getMinutes());
        object.offTime(time1);
        break;
    case 4:
        System.out.print("Temperature: ");
        ((Fridge) object).setTemperature(s.nextInt());
        System.out.println("***CHANGE COMPLETE***");
        break;
    default:
        //Menu1(objects);
        break;
    }
} else if (object instanceof Blinds) {
    System.out.println("\nChange: \n1.Change condition\n2.Time
On\n3.Time Off\nelse.Get back");
    changeable = s.nextInt();
    switch (changeable) {
        case 1:
            if (object.isActive())
                object.ElementOff();
            else
                object.ElementOn();
            System.out.println("***CHANGE COMPLETE***");
            break;
        case 2:
            System.out.println("Time(int int): ");
            time1.setHours(s.nextInt());
            if (time1.getHours() > 23)
                throw new InvalidAttributeValueException("Wrong hours
value - " + time1.getHours());
            time1.setMinutes(s.nextInt());
            if (time1.getMinutes() > 59)
                throw new InvalidAttributeValueException("Wrong
minutes value - " + time1.getMinutes());
            object.onTime(time1);
            break;
        case 3:
            System.out.println("Time(int int): ");
            time1.setHours(s.nextInt());
            if (time1.getHours() > 23)
                throw new InvalidAttributeValueException("Wrong hours
value - " + time1.getHours());
            time1.setMinutes(s.nextInt());
            if (time1.getMinutes() > 59)
                throw new InvalidAttributeValueException("Wrong
minutes value - " + time1.getMinutes());
            object.offTime(time1);
            break;
        default:
            //Menu1(objects);
            break;
    }
} else if (object instanceof Lamp) {
    System.out.println("\nChange: \n1.Change condition\n2.Time
On\n3.Time Off\nelse.Get back");
    changeable = s.nextInt();
    switch (changeable) {
        case 1:
            if (object.isActive())
                object.ElementOff();

```

```

        else
            object.ElementOn();
            System.out.println("***CHANGE COMPLETE***");
            break;
        case 2:
            System.out.println("Time(int int): ");
            time1.setHours(s.nextInt());
            if (time1.getHours() > 23)
                throw new InvalidAttributeValueException("Wrong hours
value - " + time1.getHours());
            time1.setMinutes(s.nextInt());
            if (time1.getMinutes() > 59)
                throw new InvalidAttributeValueException("Wrong
minutes value - " + time1.getMinutes());
            object.onTime(time1);
            break;
        case 3:
            System.out.println("Time(int int): ");
            time1.setHours(s.nextInt());
            if (time1.getHours() > 23)
                throw new InvalidAttributeValueException("Wrong hours
value - " + time1.getHours());
            time1.setMinutes(s.nextInt());
            if (time1.getMinutes() > 59)
                throw new InvalidAttributeValueException("Wrong
minutes value - " + time1.getMinutes());
            object.offTime(time1);
            break;
        default:
            //Menu1(objects);
            break;
    }
} else if (object instanceof Conditioner) {
    System.out.println("\nChange: \n1.Change condition\n2.Time
On\n3.Time Off\n4.Temperature\nelse.Get back");
    changeable = s.nextInt();
    switch (changeable) {
        case 1:
            if (object.isActive())
                object.ElementOff();
            else
                object.ElementOn();
            System.out.println("***CHANGE COMPLETE***");
            break;
        case 2:
            System.out.println("Time(int int): ");
            time1.setHours(s.nextInt());
            if (time1.getHours() > 23)
                throw new InvalidAttributeValueException("Wrong hours
value - " + time1.getHours());
            time1.setMinutes(s.nextInt());
            if (time1.getMinutes() > 59)
                throw new InvalidAttributeValueException("Wrong
minutes value - " + time1.getMinutes());
            object.onTime(time1);
            break;
        case 3:
            System.out.println("Time(int int): ");
            time1.setHours(s.nextInt());
            if (time1.getHours() > 23)
                throw new InvalidAttributeValueException("Wrong hours
value - " + time1.getHours());
            time1.setMinutes(s.nextInt());

```

```

        if (time1.getMinutes() > 59)
            throw new InvalidAttributeValueException("Wrong
minutes value - " + time1.getMinutes());
        object.offTime(time1);
        break;
    case 4:
        System.out.print("Temperature: ");
        ((Fridge) object).setTemperature(s.nextInt());
        System.out.println("***CHANGE COMPLETE***");
        break;
    default:
        //Menu1(objects);
        break;
    }
} else if (object instanceof Shower_cabin){
    System.out.println("\nChange: \n1.Change condition\n2.Time
On\n3.Time Off\n" +
        "4.Temperature\n5.Change Hydro Massage\nelse.Get back");
    changeable = s.nextInt();
    switch (changeable) {
        case 1:
            if (object.isActive())
                object.ElementOff();
            else
                object.ElementOn();
            System.out.println("***CHANGE COMPLETE***");
            break;
        case 2:
            System.out.println("Time(int int): ");
            time1.setHours(s.nextInt());
            if (time1.getHours() > 23)
                throw new InvalidAttributeValueException("Wrong hours
value - " + time1.getHours());
            time1.setMinutes(s.nextInt());
            if (time1.getMinutes() > 59)
                throw new InvalidAttributeValueException("Wrong
minutes value - " + time1.getMinutes());
            object.onTime(time1);
            break;
        case 3:
            System.out.println("Time(int int): ");
            time1.setHours(s.nextInt());
            if (time1.getHours() > 23)
                throw new InvalidAttributeValueException("Wrong hours
value - " + time1.getHours());
            time1.setMinutes(s.nextInt());
            if (time1.getMinutes() > 59)
                throw new InvalidAttributeValueException("Wrong
minutes value - " + time1.getMinutes());
            object.offTime(time1);
            break;
        case 4:
            System.out.print("Temperature: ");
            ((Shower_cabin) object).setTemperature(s.nextInt());
            System.out.println("***CHANGE COMPLETE***");
            break;
        case 5:
            if(((Shower_cabin) object).isHydroMassage())
                ((Shower_cabin) object).OffHydroMassage();
            else
                ((Shower_cabin) object).OnHydroMassage();
            break;
        default:

```



```

        //Menu1(objects);
        break;
    }

}

} catch(InvalidAttributeValueException e){
System.err.println("Wrong value of time: " + e.getMessage());
throw new RuntimeException(e);
}
}

private static void CreateNewObject(List<House> objects) throws
IOException, InvalidKeyException {
    int changeable;
    Scanner s = new Scanner(System.in);
    System.out.println("\nWhat
type:\n1.Fridge\n2.Blinds\n3.Lamp\n4.Conditioner\n5.Shower_cabin");

    changeable = s.nextInt();
    House house = null;
    boolean active;
    int Temperature;
    boolean Shower;
    Time time1 = new Time(0, 0);
    Time time2 = new Time(0, 0);

    System.out.print("Is active(true/false): ");
    active = s.nextBoolean();
    System.out.print("Time on(int int): ");
    time1.setHours(s.nextInt());
    time1.setMinutes(s.nextInt());
    System.out.print("Time off(int int): ");
    time2.setHours(s.nextInt());
    time2.setMinutes(s.nextInt());
    try {
        switch (changeable) {
            case 1:
                System.out.print("Temperature(int): ");
                Temperature = s.nextInt();
                house = new Fridge(active, time1, time2, Temperature);
                break;
            case 2:
                house = new Blinds(active, time1, time2);
                break;
            case 3:
                house = new Lamp(active, time1, time2);
                break;
            case 4:
                System.out.print("Temperature(int): ");
                Temperature = s.nextInt();
                house = new Conditioner(active, time1, time2, Temperature);
                break;
            case 5:
                System.out.print("Temperature(int): ");
                Temperature = s.nextInt();
                System.out.print("Hydro Massage(true/false): ");
                Shower = s.nextBoolean();
                house = new Shower_cabin(active, time1, time2, Temperature,
Shower);
                break;
            default:
                throw new InvalidKeyException("Invalid key:" + changeable);
        }
    }
}

```

```

        objects.add(house);
    } catch (InvalidKeyException e) {
        System.err.println("Error:" + e.getMessage());
    }
}

private static void CreateStarterPack(List <House> objects) throws IOException{

    Time onTime2 = new Time(9, 0);
    Time offTime2 = new Time(18, 0);
    Blinds blinds1 = new Blinds(false, onTime2, offTime2);
    objects.add(blinds1);

    Time onTime3 = new Time(5, 0);
    Time offTime3 = new Time(8, 0);
    Lamp lamp1 = new Lamp(false, onTime3, offTime3);
    objects.add(lamp1);

    Time onTime1 = new Time(9, 0);
    Time offTime1 = new Time(22, 0);
    Fridge fridge1 = new Fridge(true, onTime1, offTime1, 15);
    objects.add(fridge1);

    Time onTime4 = new Time(15,30);
    Time offTime4 = new Time(19,30);
    Conditioner conditioner1 = new Conditioner(true, onTime4, offTime4, 18);
    objects.add(conditioner1);

    Time onTime5 = new Time(19,00);
    Time offTime5 = new Time(20,30);
    Shower_cabin shower_cabin1 = new Shower_cabin(true,onTime5,
offTime5,65,true);
    objects.add(shower_cabin1);
    int i = 1;
    System.out.println("\n\t\tSTARTER PACK");
    for (House object : objects) {
        System.out.println(i++ + ": " + object);
    }
}

class Time {
    private int hours;

    private int minutes;

    public Time(int hours, int minutes)
    {
        this.minutes = minutes;
        this.hours = hours;
    }

    public void setHours(int hours) {
        this.hours = hours;
    }

    public void setMinutes(int minutes) {
        this.minutes = minutes;
    }

    public int getHours() {
        return hours;
    }
}

```

```

    }

    public int getMinutes() {
        return minutes;
    }

    public String getTime() {
        return hours + ":" + minutes;
    }
}

public interface Controller{
    public void onTime(Time onTime);

    public void offTime(Time offTime);

    public void ElementOn();

    public void ElementOff();
}

class House implements Controller {
    private boolean IsWork;
    protected Time OnTime;
    protected Time OffTime;

    public House(boolean isWork, Time onTime, Time offTime) {
        IsWork = isWork;
        OnTime = onTime;
        OffTime = offTime;
    }

    @Override
    public void onTime(Time onTime) {
        this.OnTime = onTime;
    }

    @Override
    public void offTime(Time offTime) {
        this.OffTime = offTime;
    }

    @Override
    public void ElementOn() {
        IsWork = true;
    }

    @Override
    public void ElementOff() {
        IsWork = false;
    }

    public boolean isActive() {
        return IsWork;
    }
}

```

```

    }

    public boolean isWithinWorkingHours(Time inputTime) {
        if(this.isActive() && inputTime.getHours() > OffTime.getHours()){
            return true;
        } else if (inputTime.getHours() > OnTime.getHours() &&
inputTime.getHours() < OffTime.getHours()) {
            return true;
        } else if (inputTime.getHours() == OnTime.getHours() &&
inputTime.getMinutes() >= OnTime.getMinutes()) {
            return true;
        } else if (inputTime.getHours() == OffTime.getHours() &&
inputTime.getMinutes() < OffTime.getMinutes()) {
            return true;
        }
        return false;
    }
}

class Blinds extends House{
    public Blinds(boolean isWork, Time onTime, Time offTime) {
        super(isWork, onTime, offTime);
    }

    @Override
    public String toString() {
        return "Blinds {" +
            "Condition=" + (isActive() == true? "working":"no working") +
            ", OnTime=" + OnTime.getTime() +
            ", OffTime=" + OffTime.getTime() +
            "}";
    }
}

class Conditioner extends House{
    private int Temperature;
    public Conditioner(boolean isWork, Time onTime, Time offTime, int temperature)
{
        super(isWork, onTime, offTime);
        Temperature = temperature;
    }

    @Override
    public String toString() {
        return "Conditioner {" +
            (isActive() == true? "Condition=" + (isActive() == true?
"working":"no working") +
            ", OnTime=" + OnTime.getTime() +
            ", OffTime=" + OffTime.getTime() +
            ", Temperature=" + Temperature:
"Condition=" + (isActive() == true? "working":"no
working") +
            ", OnTime=" + OnTime.getTime() +
            ", OffTime=" + OffTime.getTime() +
            '}';
    }
}

```

```

    }
}

class Fridge extends House{
    private int Temperature;
    public Fridge(boolean isWork, Time onTime, Time offTime, int temperature) {
        super(isWork, onTime, offTime);
        Temperature = temperature;
    }

    public void setTemperature(int temperature) {
        Temperature = temperature;
    }

    @Override
    public String toString() {
        return "Fridge {" +
            (isActive() == true? "Condition=" + (isActive() == true?
"working":"no working") +
                ", OnTime=" + OnTime.getTime() +
                ", OffTime=" + OffTime.getTime() +
                ", Temperature=" + Temperature :
                "Condition=" + (isActive() == true? "working":"no
working") +
                    ", OnTime=" + OnTime.getTime() +
                    ", OffTime=" + OffTime.getTime()
                ) +
            '}';
    }
}

class Lamp extends House{
    public Lamp(boolean isWork, Time onTime, Time offTime) {
        super(isWork, onTime, offTime);
    }

    @Override
    public String toString() {
        return "Lamp {" +
            "Condition=" + (isActive() == true? "working":"no working") +
            ", OnTime=" + OnTime.getTime() +
            ", OffTime=" + OffTime.getTime() +
            "}";
    }
}

class Shower_cabin extends House{
    private int Temperature;

    private boolean HydroMassage;

```

```

    public Shower_cabin(boolean isWork, Time onTime, Time offTime, int
temperature, boolean hydroMassage) {
        super(isWork, onTime, offTime);
        Temperature = temperature;
        HydroMassage = hydroMassage;
    }

    public int getTemperature() {
        return Temperature;
    }

    public void setTemperature(int temperature) {
        Temperature = temperature;
    }

    public boolean isHydroMassage() {
        return HydroMassage;
    }

    public void OnHydroMassage() {
        HydroMassage = true;
    }

    public void OffHydroMassage() {
        HydroMassage = false;
    }

    @Override
    public String toString() {
        return "Shower_cabin{" +
            (isActive() == true? "Condition=" + (isActive() == true?
"working":"no working") +
                ", OnTime=" + OnTime.getTime() +
                ", OffTime=" + OffTime.getTime() +
                ", Temperature=" + Temperature +
                ", HydroMassage=" + HydroMassage:
                "Condition=" + (isActive() == true? "working":"no
working") +
                    ", OnTime=" + OnTime.getTime() +
                    ", OffTime=" + OffTime.getTime()
                ) +
            '}';
    }
}

```

Частина 3. Діаграма структури класів

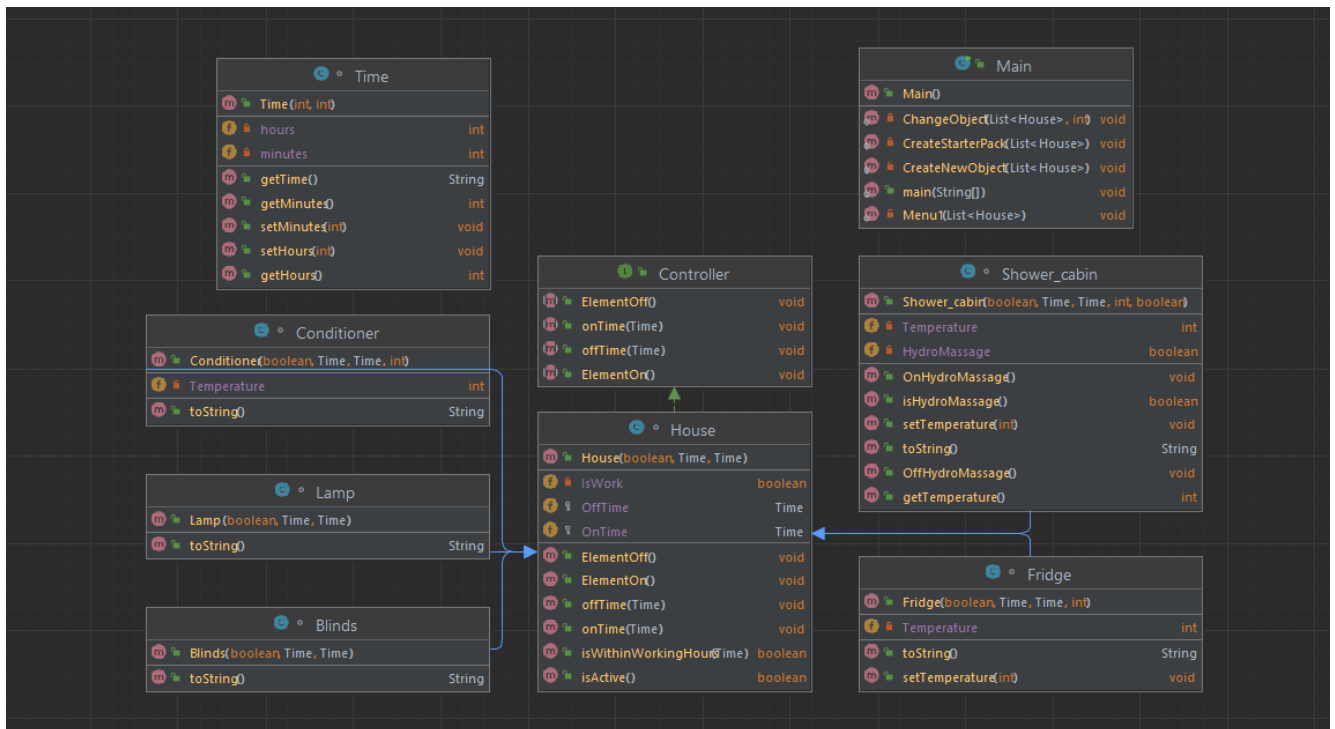


Рис. 3 Діаграма класів

Частина 4. Скріншоти виконання завдання

```
STARTER PACK
1: Blinds      {Condition=no working, OnTime=9:0, OffTime=18:0}
2: Lamp        {Condition=no working, OnTime=5:0, OffTime=8:0}
3: Fridge       {Condition=working, OnTime=9:0, OffTime=22:0, Temperature=15}
4: Conditioner {Condition=working, OnTime=15:30, OffTime=19:30, Temperature=18}
5: Shower_cabin {Condition=working, OnTime=19:0, OffTime=20:30, Temperature=65, HydroMassage=true}

Next move:
1.Add new object
2.Delete object
3.Change condition of object
4.Working at user time
else.EXIT
1

What type:
1.Fridge
2.Blinds
3.Lamp
4.Conditioner
5.Shower_cabin
5

Is active(true/false): true
Time on(int int): 12 11
Time off(int int): 23 50
Temperature(int): 87
Hydro Massage(true/false): false
```

Рис. 4 Скріншот виконання завдання (1 частина)

```

1: Blinds      {Condition=no working, OnTime=9:0, OffTime=18:0}
2: Lamp        {Condition=no working, OnTime=5:0, OffTime=8:0}
3: Fridge      {Condition=working, OnTime=9:0, OffTime=22:0, Temperature=15}
4: Conditioner {Condition=working, OnTime=15:30, OffTime=19:30, Temperature=18}
5: Shower_cabin{Condition=working, OnTime=19:0, OffTime=20:30, Temperature=65, HydroMassage=true}
6: Shower_cabin{Condition=working, OnTime=12:11, OffTime=23:50, Temperature=87, HydroMassage=false}

Next move:
1.Add new object
2.Delete object
3.Change condition of object
4.Working at user time
else.EXIT
2
Which one :5

1: Blinds      {Condition=no working, OnTime=9:0, OffTime=18:0}
2: Lamp        {Condition=no working, OnTime=5:0, OffTime=8:0}
3: Fridge      {Condition=working, OnTime=9:0, OffTime=22:0, Temperature=15}
4: Conditioner {Condition=working, OnTime=15:30, OffTime=19:30, Temperature=18}
5: Shower_cabin{Condition=working, OnTime=12:11, OffTime=23:50, Temperature=87, HydroMassage=false}

```

Рис. 5 Скріншот виконання завдання (2 частина)

```

Next move:
1.Add new object
2.Delete object
3.Change condition of object
4.Working at user time
else.EXIT
3
Which one :1

Change:
1.Change condition
2.Time On
3.Time Off
else.Get back
1
***CHANGE COMPLETE***

1: Blinds      {Condition=working, OnTime=9:0, OffTime=18:0}
2: Lamp        {Condition=no working, OnTime=5:0, OffTime=8:0}
3: Fridge      {Condition=working, OnTime=9:0, OffTime=22:0, Temperature=15}
4: Conditioner {Condition=working, OnTime=15:30, OffTime=19:30, Temperature=18}
5: Shower_cabin{Condition=working, OnTime=12:11, OffTime=23:50, Temperature=87, HydroMassage=false}

```

Рис. 6 Скріншот виконання завдання (3 частина)


```

Next move:
1.Add new object
2.Delete object
3.Change condition of object
4.Working at user time
else.EXIT
3
Which one :2

Change:
1.Change condition
2.Time On
3.Time Off
else.Get back
2
Time(int int):
4 50

1: Blinds      {Condition=working, OnTime=9:0, OffTime=18:0}
2: Lamp        {Condition=no working, OnTime=4:50, OffTime=8:0}
3: Fridge      {Condition=working, OnTime=9:0, OffTime=22:0, Temperature=15}
4: Conditioner {Condition=working, OnTime=15:30, OffTime=19:30, Temperature=18}
5: Shower_cabin{Condition=working, OnTime=12:11, OffTime=23:50, Temperature=87, HydroMassage=false}

```

Рис. 7 Скріншот виконання завдання (4 частина)

```

Next move:
1.Add new object
2.Delete object
3.Change condition of object
4.Working at user time
else.EXIT
3
Which one :3

Change:
1.Change condition
2.Time On
3.Time Off
4.Temperature
else.Get back
3
Time(int int):
23 15

1: Blinds      {Condition=working, OnTime=9:0, OffTime=18:0}
2: Lamp        {Condition=no working, OnTime=4:50, OffTime=8:0}
3: Fridge      {Condition=working, OnTime=9:0, OffTime=23:15, Temperature=15}
4: Conditioner {Condition=working, OnTime=15:30, OffTime=19:30, Temperature=18}
5: Shower_cabin{Condition=working, OnTime=12:11, OffTime=23:50, Temperature=87, HydroMassage=false}

```

Рис. 8 Скріншот виконання завдання (5 частина)

```

Next move:
1.Add new object
2.Delete object
3.Change condition of object
4.Working at user time
else.EXIT
3
Which one :4

Change:
1.Change condition
2.Time On
3.Time Off
4.Temperature
else.Get back
4
Temperature: 32
***CHANGE COMPLETE***

1: Blinds      {Condition=working, OnTime=9:0, OffTime=18:0}
2: Lamp        {Condition=no working, OnTime=4:50, OffTime=8:0}
3: Fridge      {Condition=working, OnTime=9:0, OffTime=23:15, Temperature=15}
4: Conditioner {Condition=working, OnTime=15:30, OffTime=19:30, Temperature=32}
5: Shower_cabin{Condition=working, OnTime=19:0, OffTime=20:30, Temperature=65, HydroMassage=true}

```

Рис. 9 Скріншот виконання завдання (6 частина)

```

Next move:
1.Add new object
2.Delete object
3.Change condition of object
4.Working at user time
else.EXIT
3
Which one :5

Change:
1.Change condition
2.Time On
3.Time Off
4.Temperature
5.Change Hydro Massage
else.Get back
5

1: Blinds      {Condition=working, OnTime=9:0, OffTime=18:0}
2: Lamp        {Condition=no working, OnTime=4:50, OffTime=8:0}
3: Fridge      {Condition=working, OnTime=9:0, OffTime=23:15, Temperature=15}
4: Conditioner {Condition=working, OnTime=15:30, OffTime=19:30, Temperature=32}
5: Shower_cabin{Condition=working, OnTime=19:0, OffTime=20:30, Temperature=65, HydroMassage=false}

```

Рис. 10 Скріншот виконання завдання (7 частина)

```

Next move:
1.Add new object
2.Delete object
3.Change condition of object
4.Working at user time
else.EXIT
4
Time(int int):
10 00

1: Blinds      {Condition=working, OnTime=9:0, OffTime=18:0}
2: Fridge      {Condition=working, OnTime=9:0, OffTime=23:15, Temperature=15}

Next move:
1.Add new object
2.Delete object
3.Change condition of object
4.Working at user time
else.EXIT
4
Time(int int):
20 00

1: Blinds      {Condition=working, OnTime=9:0, OffTime=18:0}
2: Fridge      {Condition=working, OnTime=9:0, OffTime=23:15, Temperature=15}
3: Conditioner {Condition=working, OnTime=15:30, OffTime=19:30, Temperature=32}
4: Shower_cabin{Condition=working, OnTime=19:0, OffTime=20:30, Temperature=65, HydroMassage=false}

```

Рис.11 Скріншот виконання завдання (8 частина)

```

Next move:
1.Add new object
2.Delete object
3.Change condition of object
4.Working at user time
else.EXIT
7
Process finished with exit code 0
|

```

Рис. 12 Скріншот виконання завдання (9 частина)

Завдання 2. (перероблене 5.1)

Частина 1. Умови завдання

ЗАВДАННЯ 2.

Розробити програму з використанням абстрактних класів та інтерфейсів відповідно до завдання.

- використовувати при розробці успадкування і поліморфізм;
- зобразити UML діаграму класів програми.
- зробити висновки.

Рис. 13 Умови завдання

11. interface Фільм ← abstract class Український фільм ← class Комедія.

Рис. 14 Персональне завдання

Частина 2. Текст програми

```
/*                                Film.java                                */
public                                interface                                Film                                {

    public                                String                                getName ();
    public                                void                                setName (String                                name);
    public                                int                                getBudget ();
    public                                void                                setBudget (int                                budget);
    public                                void                                setLanguage (String                                language);
    public                                String                                getLanguage ();
}

/*                                UkrainianFilm.java                                */
abstract class UkrainianFilm implements Film{
    protected String language;
    protected String Name;
    protected int Budget;

    @Override
    public String getName() {
        return this.Name;
    }

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

    @Override
    public int getBudget() {
        return this.Budget;
    }

    @Override
    public void setBudget(int budget) {
        this.Budget = budget;
    }
}
```

```

@Override
public void setLanguage(String language) {
    this.language = language;
}

@Override
public String getLanguage() {
    return this.language;
}
}

/*      Comedy.java      */

class Comedy extends UkrainianFilm
{
    private boolean IsFunny;

    public boolean getisFunny() {
        return IsFunny;
    }

    public void setFunny(boolean funny) {
        IsFunny = funny;
    }
}

```

Частина 3. Діаграма структури класів

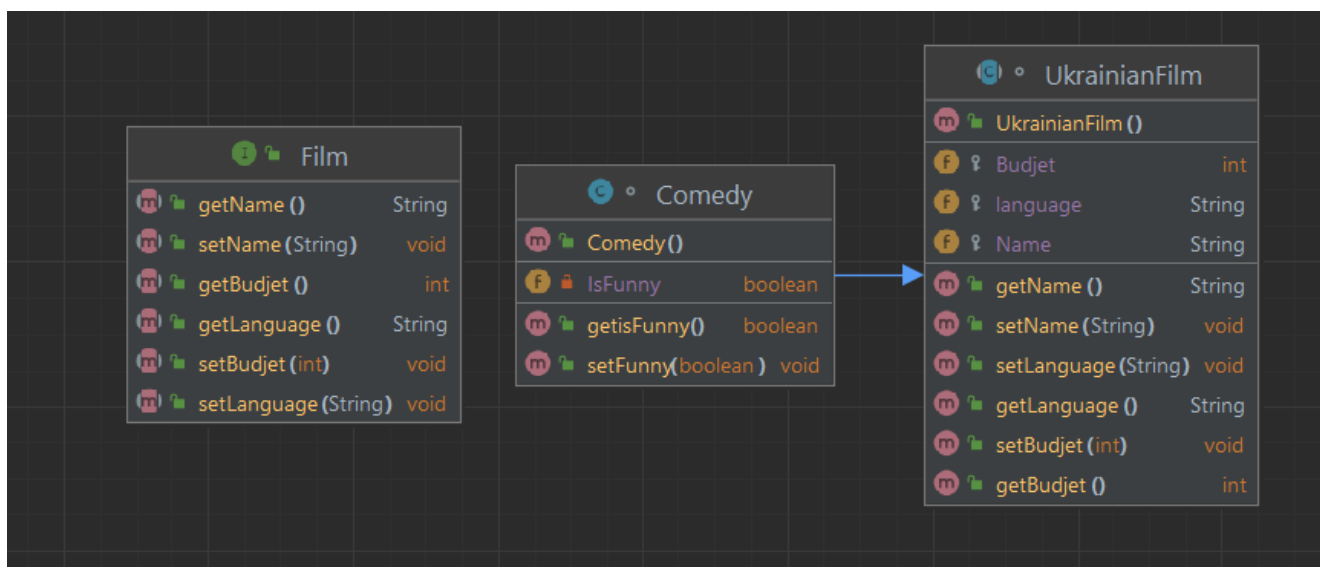
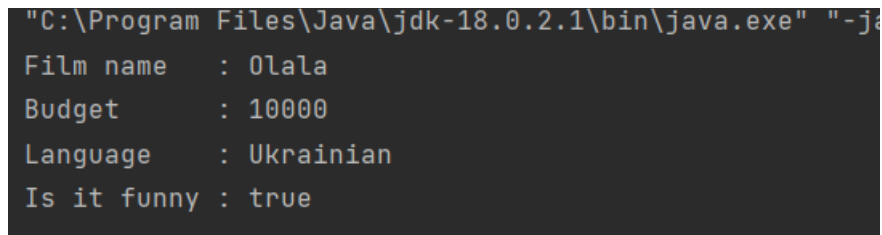


Рис. 15 Діаграма класів

Частина 4. Скріншоти виконання завдання



```
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-ja
Film name   : Olala
Budget      : 10000
Language    : Ukrainian
Is it funny : true
```

Рис. 16 Скріншот виконання завдання (1 частина)

Висновки

1. Відпрацював навички об'єктно-орієнтованого аналізу і розробки архітектури програмних систем.
2. Навчився розробляти програми на мові Java з використанням об'єктно орієнтованого підходу.

Використані джерела

1. Лекції з “Технології програмування”в