

Dokumentacja projektu SmartHouse

Spis treści:

- Opis ogólny
- Struktura klas i interfejsów
 - Package abstracts;
 - Package classes.devices;
 - Package classes.house;
 - Package classes;
 - Package enums;
 - Package interfaces;
- Instrukcja uruchomienia i obsługi oraz screenshoty

Opis ogólny

SmartHouse to aplikacja konsolowa (CLI) symulująca zarządzanie domami, pokojami oraz urządzeniami Smart w nich zainstalowanymi. System wspiera rejestrowanie oraz usuwanie nowych urządzeń, rejestrowanie zdarzeń (logger) w formacie tsv w pliku log_<timestamp>.tsv tworzonym przy uruchomieniu programu w miejscu jego uruchomienia, oraz symuluje zachowanie urządzeń w czasie rzeczywistym.

Struktura klas i interfejsów

Package abstracts;

public abstract class SmartDevice

Klasa abstrakcyjna implementująca podstawowe funkcjonalności każdego urządzenia Smart.

```
public SmartDevice(int deviceId, String deviceName, DeviceType deviceType) {
    this.deviceUUID = UUID.randomUUID();
    this.deviceType = deviceType;
    this.deviceId = deviceId;
    this.deviceName = deviceName;
    this.deviceStatus = DeviceStatus.OFF;
}
```

Pola:

- private UUID deviceUUID;
- private int deviceId;
- private String deviceName;
- private DeviceStatus deviceStatus;
- private DeviceType deviceType;
- private Room room; - odpowiada za przechowywanie referencji do pokoju w którym znajduje się urządzenie
- private boolean live; - odpowiada za wypisywanie informacji w konsoli na żywo

Metody:

- public abstract void simulate() throws IllegalAccessException;
- @Override public String toString();
- public void setLive();
- public boolean isLive();
- public void setStatus(DeviceStatus status);
- public DeviceStatus getStatus();
- public int getDeviceId();
- public String getDeviceName();
- public void setDeviceName(String deviceName);
- public DeviceType getDeviceType();
- public void setRoom(Room room);
- public Room getRoom();

Package classes.devices;

public class AirCondition extends SmartDevice implements Switchable, ObservableDevice

Klasa symulująca klimatyzację, wraz z symulacją obniżania temperatury w pokoju.

Switchable – można włączyć/wyłączyć (również za pomocą reguł)

ObservableDevice – dla Loggera, oraz InfoTablet; realizuje wzorzec obserwatora

```
public AirCondition(int deviceId, String deviceName, int chillingPower) {
    super(deviceId, deviceName, DeviceType.AIRCON);
    this.chillingPower = chillingPower;

    running = true;
    thread = new Thread(() -> {
        while (running) {
            try {
                this.simulate();
                Thread.sleep(chillingCycleTime);
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InterruptedException e) {
                running = false;
                Thread.currentThread().interrupt();
            }
        }
    });
    thread.start();
}
```

Pola:

- private int chillingPower;
- private int chillingCycleTime = 1000;
- private Room room;

- private boolean running; - odpowiada sprawdzeniu czy należy dalej symulować w czasie rzeczywistym (wyjście z pętli while)
- private Thread thread; - wątek do symulacji w czasie rzeczywistym

- private List<DeviceObserver> observers = new ArrayList<>(); - odpowiada za listę urządzeń odbierających powiadomienia (Logger, bądź InfoTablet)

Metody:

- Implementowane przez SmartDevice:
 - public void simulate() throws IllegalAccessException - Jeżeli jest włączone urządzenie wysyła powiadomienie do Loggera oraz ochładza pokój

- Implementowane przez Switchable:
 - public void turnOn()
 - public void turnOff()
 - public boolean isOn()

- Implementowane przez ObservableDevice:
 - public void addObserver(DeviceObserver observer)
 - public void removeObserver(DeviceObserver observer)
 - public void notifyObservers(String eventDescription) - do wysyłania zdarzeń z domyślnym rodzajem „STATUS_CHANGED”
 - public void notifyObservers(LogType eventType, String eventDescription) - do wysyłania zdarzeń z wybranym rodzajem (enum LogTypes)

- Klasowe:
 - public int getCoolingPower()
 - public void setRoom(Room room)
 - public Room getRoom()

public class Heater extends SmartDevice implements Switchable, ObservableDevice

Klasa symulująca grzejnik elektryczny, wraz z symulacją ogrzewania pokoju.

Switchable – można włączyć/wyłączyć

ObservableDevice – dla Loggera, oraz InfoTablet; realizuje wzorzec obserwatora

```
public Heater(int deviceId, String deviceName, int heatingPower) {
    super(deviceId, deviceName, DeviceType.HEATER);
    this.heatingPower = heatingPower;
    this.setStatus(DeviceStatus.ON);

    running = true;
    thread = new Thread(() -> {
        while (running) {
            try {
                this.simulate();
                Thread.sleep(heatCycleTime);
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InterruptedException e) {
                running = false;
                Thread.currentThread().interrupt();
            }
        }
    });
    thread.start();
}
```

Pola:

- private int heatingPower;

- private int heatCycleTime = 1000;
- private Room room;
- private boolean running;
- private Thread thread;
- private List<DeviceObserver> observers = new ArrayList<>();

Metody:

- Implementowane przez SmartDevice:
 - public void simulate() throws IllegalAccessException - Jeżeli jest włączone urządzenie wysyła powiadomienie do Loggera oraz ogrzewa pokój
- Implementowane przez Switchable:
 - public void turnOn()
 - public void turnOff()
 - public boolean isOn()
- Implementowane przez ObservableDevice:
 - public void addObserver(DeviceObserver observer)
 - public void removeObserver(DeviceObserver observer)
 - public void notifyObservers(String eventDescription)
 - public void notifyObservers(LogType eventType, String eventDescription)
- Klasowe:
 - public int getHeatingPower()
 - public void setRoom(Room room)
 - public Room getRoom()

public class HumiditySensor extends SmartDevice implements SensorDevice<Double>, ObservableDevice

Klasa symulująca czujnik wilgotności, z symulacją pomiarów w czasie rzeczywistym.

SensorDevice<Double> - interfejs dla metod pomiarowych z typem generycznym określającym rodzaj zwracanej wartości

ObservableDevice – dla Loggera, oraz InfoTablet; realizuje wzorzec obserwatora

```
public HumiditySensor(int deviceId, String deviceName) {
    super(deviceId, deviceName, DeviceType.HUMIDITYSENSOR);
    super.setStatus(DeviceStatus.ON);
}
```

Pola:

- private double humidity;
- private Room room;
- private int readCycleTime = 1500;
- private Thread thread;
- private boolean running = false;
- private boolean started = false;
- private List<DeviceObserver> observers = new ArrayList<>();

Metody:

- Implementowane przez SmartDevice:
 - public void simulate() throws IllegalAccessException - Losuje szansę na “FAULT” oraz odczytuje wilgotność pomieszczenia, i przesyła do Loggera oraz na konsolę jeżeli „isLive == true”
- Implementowane przez SensorDevice<T>:
 - public Double readValue()
 - public String getUnit() - g/m³
- Implementowane przez ObservableDevice:
 - public void addObserver(DeviceObserver observer)
 - public void removeObserver(DeviceObserver observer)
 - public void notifyObservers(String eventDescription)
 - public void notifyObservers(LogType eventType, String eventDescription)
- Klasowe:
 - public void fixFault() - symuluje naprawę usterki oraz ponownie uruchamia wątek symulacyjny
 - public void setRoom(Room room) - po ustawieniu pokoju tworzy wątek symulacyjny i go startuje
 - public Room getRoom()

public class InfoTablet extends SmartDevice implements DeviceObserver, ObservableDevice, Switchable

Klasa symulująca tablet otrzymująca informacje od innych urządzeń (realizując wzorzec obserwatora)

DeviceObserver – interfejs dla metod reagujących na otrzymane zdarzenia

ObservableDevice – dla Loggera; realizuje wzorzec obserwatora

Switchable – można włączyć/wyłączyć (również za pomocą reguł)

```
public InfoTablet(int deviceId, String name, SmartDevice device) {
    super(deviceId, name, DeviceType.INFOTABLET);
    super.setStatus(DeviceStatus.ON);
    observe(device);
    this.thread = createThread();
    thread.start();
}
```

Pola:

- private Room room;
- private Thread thread;
- private boolean running = false;
- private List<DeviceObserver> observers = new ArrayList<>();

Metody:

- Implementowane przez SmartDevice:
 - public void simulate() throws IllegalAccessException - Losuje spośród trzech wiadomości jedną do wysłania, w tym jedna z nich ma szansę na wywołanie „FAULT”
- Implementowane przez DeviceObserver:
 - public void onDeviceEvent(SmartDevice device, String event) - do odbierania zdarzeń z domyślnym typem „STATUS_CHANGED”; dla Loggera
 - public void onDeviceEvent(SmartDevice device, LogType logType, String event) - do odbierania zdarzeń z wybranym rodzajem (enum LogTypes); dla Loggera
- Implementowane przez ObservableDevice:
 - public void addObserver(DeviceObserver observer)
 - public void removeObserver(DeviceObserver observer)
 - public void notifyObservers(String eventDescription)
 - public void notifyObservers(LogType eventType, String eventDescription)
- Implementowane przez Switchable:
 - public void turnOn()
 - public void turnOff()
 - public boolean isOn()
- Klasowe:
 - private Thread createThread() - tworzy wątek symulacyjny
 - private void attemptRestart() - podejmuje próbę stworzenia nowego wątku po ustawieniu statusu „FAULT”
 - public void observe(SmartDevice device) - do realizacji wzorca obserwatora
 - public void setRoom(Room room)
 - public Room getRoom()

public class Lightbulb extends SmartDevice implements Switchable, ObservableDevice

Klasa symulująca żarówkę z możliwością zmiany koloru oświetlenia

Switchable – można włączyć/wyłączyć

ObservableDevice – dla Loggera; realizacja wzorca obserwatora

```
public Lightbulb(int deviceId, String deviceName, int hue, double saturation, double value) {
    this.hue = hue;
    this.saturation = saturation;
    this.value = value;
    super(deviceId, deviceName, DeviceType.LIGHTBULB);
}
```

Pola:

- private int hue;
- private double saturation;
- private double value;
- private DeviceStatus status;
- private Room room;
- private List<DeviceObserver> observers = new ArrayList<>();

Metody:

- Implementowane przez SmartDevice:
 - public void simulate() throws IllegalAccessException - wysyła na konsolę oraz do Loggera informację o tym że żarówka świeci na dany kolor (jeżeli jest włączona)
- Implementowane przez Switchable:
 - public void turnOn()
 - public void turnOff()
 - public boolean isOn()
- Implementowane przez ObservableDevice:
 - public void addObserver(DeviceObserver observer)
 - public void removeObserver(DeviceObserver observer)
 - public void notifyObservers(String eventDescription)
 - public void notifyObservers(LogType eventType, String eventDescription)
- Klasowe:
 - public void changeColor(int hue, double saturation, double value) - pozwala zmienić pola klasowe dotyczące koloru, z walidacją przekazanych wartości
 - public Color getRGBColor() - konwersja HSV na RGB poprzez kalkulację
 - public void setRoom(Room room)
 - public Room getRoom()

public class Outlet extends SmartDevice implements Switchable, ObservableDevice

Klasa symulująca gniazdko elektryczne, z symulacją poboru prądu oraz przeciążenia w czasie rzeczywistym.

Switchable – można włączyć/wyłączyć

ObservableDevice – dla Loggera; realizacja wzorca obserwatora

```
public Outlet(int deviceId, String deviceName, int overloadThreshold) {
    super(deviceId, deviceName, DeviceType.OUTLET);
    this.overloadThreshold = overloadThreshold;
    Thread thread = new Thread(() -> {
        while (running) {
            try {
                simulate();
                Thread.sleep(1000);
            } catch (IllegalAccessException e) {
                setStatus(DeviceStatus.FAULT);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    });
}
```

Pola:

- private boolean powered; - odpowiada za podłączenie gniazdka do prądu
- private double totalEnergyConsumed; - do prowadzenia statystyki zużycia prądu w tym gniazdku
- private int overloadThreshold; - do symulacji przeciążenia
- private boolean running = true;
- private Room room;

- private List<DeviceObserver> observers = new ArrayList<>();

Metody:

- Implementowane przez **SmartDevice**:
 - public void simulate() throws IllegalAccessException - losuje wartości poboru prądu z przedziału, oraz wysyła powiadomienie do Loggera
- Implementowane przez **Switchable**:
 - public void turnOn()
 - public void turnOff()
 - public boolean isOn()
- Implementowane przez **ObservableDevice**:
 - public void addObserver(DeviceObserver observer)
 - public void removeObserver(DeviceObserver observer)
 - public void notifyObservers(String eventDescription)
 - public void notifyObservers(LogType eventType, String eventDescription)
- Klasowe:
 - public void setRoom(Room room)
 - public Room getRoom()

public class TemperatureSensor extends SmartDevice implements SensorDevice<Double>, ObservableDevice

Klasa symulująca czujnik temperatury zasilany wymiennymi bateriami, z symulacją odczytu temperatury w czasie rzeczywistym.

SensorDevice<Double> - interfejs dla metod pomiarowych z typem generycznym określającym rodzaj zwracanej wartości

ObservableDevice – dla Loggera, oraz InfoTablet; realizuje wzorzec obserwatora

```
public TemperatureSensor(int deviceId, String deviceName) {
    super(deviceId, deviceName, DeviceType.TEMPSENSOR);
    super.setStatus(DeviceStatus.ACTIVE);
}
```

Pola:

- private Double temperature;
- private int batteryCycles = 50;
- private Room room;
- private int readCycleTime = 1000;

- private Thread thread;
- private boolean running = false;
- private boolean started = false;

- private List<DeviceObserver> observers = new ArrayList<>();

Metody:

- Implementowane przez **SmartDevice**:
 - public void simulate() throws IllegalAccessException - odczytuje temperaturę pokoju, i przesyła do loggera, oraz na konsolę jeżeli „isLive == true”
- Implementowane przez **SensorDevice<T>**:
 - public Double readValue()
 - public String getUnit() - Celsius
- Implementowane przez **ObservableDevice**:
 - public void addObserver(DeviceObserver observer)
 - public void removeObserver(DeviceObserver observer)
 - public void notifyObservers(String eventDescription)
 - public void notifyObservers(LogType eventType, String eventDescription)

Klasowe:

- `public void changeBattery()` - symulacja wymiany baterii, wraz z zatrzymaniem starego wątku, utworzeniem i wystartowaniem nowego
- `public void setRoom(Room room)` - zmiana pokoju, wraz z zatrzymaniem starego wątku, utworzeniem i wystartowaniem nowego
- `public Room getRoom()`

Package classes.house;

public class House

Klasa grupująca pokoje w domu.

```
public House(String name, int id, float xCoordinate, float yCoordinate) {
    this.name = name;
    this.xCoordinate = xCoordinate;
    this.yCoordinate = yCoordinate;
    this.id = id;
    this.rooms = new ArrayList<>();
}
```

Pola:

- `private String name;`
- `private int id;`
- `private float xCoordinate;`
- `private float yCoordinate;`
- `private ArrayList<Room> rooms;`

Metody:

- **Klasowe:** - tylko gettery/settery
- `public void addRoom(Room room)`
- `public void removeRoom(Room room)`
- `public Boolean hasRooms()`
- `public int getNumOfRooms()`
- `public void sortRoomsByType()`
- `public ArrayList<Room> getRooms()`
- `public String getName()`
- `public void setName(String name)`
- `public float getXCoord()`
- `public float getYCoord()`
- `public int getId()`

public class Room

Klasa zarządzająca pokojami, oraz zarejestrowanymi w nich urządzeniami i regułami automatyzacji, z symulacją zmian temperatury i wilgotności w czasie rzeczywistym.

```
public Room(String name, int id, RoomType type, double ambientTemp) {
    this.name = name;
    this.id = id;
    this.type = type;
    this.devices = new ArrayList<>();
    this.ambientTemp = ambientTemp;
    this.currentTemp = ambientTemp + (Math.random()*10) - 5;
    this.ambientHumidity = (Math.random()*5) + 12;
    this.currentHumidity = ambientHumidity + (Math.random()*2) - 1;
    this.thread = new Thread(() -> {
        while(running) {
            try{
                updateTemperature();
                updateHumidity();
                Thread.sleep(600);
            } catch(Exception e){
                e.printStackTrace();
            }
        }
    });
    thread.start();
}
```

Pola:

- `private String name;`
- `private int id;`
- `private RoomType type;`
- `private List<SmartDevice> devices;`
- `private Thread thread;`
- `private boolean running = true;`

- `private double currentTemp;`
- `private double ambientTemp;`
- `private double currentHumidity;`
- `private double ambientHumidity;`

- `private List<Rule<? extends SmartDevice>> rules = new ArrayList<>();` - Lista obiektów typu Rule typu dziedziczącego po SmartDevice

Metody:

- `private void updateTemperature()` - okresowo aktualizuje temperaturę dążąc do `ambientTemp`, z uwzględnieniem włączonych grzejników oraz klimatyzacji
- `private void updateHumidity()` - okresowo aktualizuje wilgotność dodając lub odejmując losową wartość w okolicach `ambientHumidity`
- `public void addDevice(SmartDevice device)` - dodaje urządzenie do listy urządzeń oraz wywołuje `setRoom` w urządzeniu
 - `getter`y/`setter`y bez żadnej dodatkowej funkcjonalności
- `public void addRule(Rule<? extends SmartDevice> rule)`
- `public void removeRule(int i)`
- `public List<Rule<? extends SmartDevice>> getRules()`
- `public Boolean hasRules()`
- `public String getName()`
- `public void setName(String name)`
- `public RoomType getRoomType()`
- `public int getId()`
- `public double getTemperature()`
- `public double getHumidity()`
- `public boolean hasDevices()`
- `public List<SmartDevice> getDevices()`
- `public int getNumOfDevices()`

Package classes;

public class Logger implements DeviceLogger

Klasa realizująca wzorzec obserwatora; odbiera zdarzenia od urządzeń przypisanych do Loggera, i zapisuje je w pliku TSV.

DeviceLogger – zapewnia metodę dla logowania

```
public Logger() {
    String timestamp = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyyMMdd_HHmss"));
    this.filePath = Paths.get(System.getProperty("user.dir"), "log_" + timestamp + ".tsv").toString();
    initializeFileWithHeader();
}
```

Pola:

- `private String filePath`; - ścieżka do pliku z logiem
- `private DateTimeFormatter formatter` = `DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")`; - formatowanie `timestamp`’a

Metody:

- Implementowane przez **DeviceLogger**:
 - `public <T extends SmartDevice> void log(SmartDevice device, LogType category, String message)` - formatuje oraz dopisuje linijkę do pliku TSV z wykorzystaniem `BufferedWriter`
- Klasowe**:
 - `private void initializeFileWithHeader()` - próbuje utworzyć plik TSV i umieszcza nagłówek

public class LoggerObserver implements DeviceObserver

Klasa pomocniczna do odbierania zdarzeń od urządzeń.

DeviceObserver – interfejs dla metod reagujących na otrzymane zdarzenia

```
public LoggerObserver(Logger logger) {
    this.logger = logger;
}
```

Pola:

- `private Logger logger`;

Metody:

- Implementowane przez **DeviceObserver**:
 - `public void onDeviceEvent(SmartDevice device, String event)`
 - `public void onDeviceEvent(SmartDevice device, LogType logType, String event)`

public class Rule<T extends SmartDevice>

Klasa realizująca tworzenie reguł automatyzacyjnych z urządzeniami.

```
public Rule(Predicate<T> condition, Consumer<T> action, T actionDevice) {
    this.condition = condition;
    this.action = action;
    this.actionDevice = actionDevice;
}
```

Pola:

- `private final Predicate<T> condition`;
- `private final Consumer<T> action`;
- `private final T actionDevice`;

Metody:

- Klasowe**:
 - `public boolean test()` - sprawdza ustalony warunek
 - `public void execute()` - wykonuje akcję jeżeli ustalony warunek jest prawdą

- `public void forceExecute()` - wykonuje akcję niezależnie od warunku
- `public T getActionDevice()`

public class RuleManager

Klasa odpowiadająca za zarządzanie aktywnymi regułami, oraz ich periodyczne sprawdzanie w czasie rzeczywistym.

```
private RuleManager(List<House> houseList) {
    this.houseList = houseList;

    this.ruleThread = new Thread(() -> {
        while (running) {
            for (House house : houseList) {
                for (Room room : house.getRooms()) {
                    for (Rule<? extends SmartDevice> rule : room.getRules()) {
                        try {
                            rule.execute();
                        } catch (Exception e) {
                            System.err.println("Błąd przy wykonaniu reguły: " + e.getMessage());
                        }
                    }
                }
            }
        }
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            running = false;
            break;
        }
    });
    ruleThread.start();
}
```

Pola:

- `private static RuleManager` instance;
- `private final List<House>` houseList;
- `private final Thread` ruleThread;
- `private boolean` running = true;

Metody:

- `public static RuleManager` getInstance(List<House> houseList) - odpowiada za przekazanie referencji do instancji obiektu do wywołującej klasy
- `public void` shutdown() - zatrzymuje wątek
- `public Rule<? extends SmartDevice>` parseRule(SmartDevice condDevice, SmartDevice actionDevice, String condition, String action) - przekształca przekazaną przez użytkownika definicję reguły na nowy obiekt Rule

public class Menu

Klasa realizująca obsługę użytkownika poprzez CLI.

```
public Menu() {}
```

Pola:

- `private static ArrayList<House>` houseArrayList = new ArrayList<>();
- `private int` menu; - wskaźnik menu
- `private int` chosenHouseNumber; - id wybranego domu
- `private House` chosenHouse; - referencja do obiektu
- `private int` chosenRoomNumber; - id wybranego pokoju
- `private Room` chosenRoom; - referencja do obiektu
- `private boolean` running;
- `private Scanner` scanner = new Scanner(System.in);
- `private Logger` logger = new Logger();
- `private DeviceObserver` logObserver = new LoggerObserver(logger);

Metody:

- `public void` run() - ustawia wskaźnik na 0, oraz startuje wątek
- `private void` display() - wyświetla odpowiednie menu w zależności od wskaźnika, oraz czeka na podanie polecenia przez użytkownika
- `private void` execute(String input) - wykonuje polecenie w zależności od podanego polecenia i pozycji wskaźnika
- `private void` newDevice(int id, String name, DeviceType type) - rejestruje nowe urządzenie

Package enums;

public enum DeviceStatus {

- ON,
- OFF,
- ACTIVE,
- FAULT,


```
        • LOW_BATTERY,
        • TAMPERED
    }

    public enum DeviceType {
        • HEATER,
        • AIRCON,
        • TEMPSENSOR,
        • HUMIDITYSENSOR,
        • LIGHTBULB,
        • OUTLET,
        • INFOTABLET
    }

    public enum LogType {
        • DEVICE_ADDED,
        • STATUS_CHANGE,
        • READING,
        • FAULT_DETECTED,
        • FAULT_FIXED,
        • RULE_TRIGGERED
    }

    public enum LogType {

        • KUCHNIA,
        • TOALETA,
        • KORYTARZ,
        • SYPIALNIA,
        • SALON,
        • PODDASZE
    }

}
```

Package interfaces;

```
public interface DeviceLogger {

    <T extends SmartDevice> void log(SmartDevice device, LogType category, String message);

}

public interface DeviceObserver {
    void onDeviceEvent(SmartDevice device, String event);
    void onDeviceEvent(SmartDevice device, LogType logType, String event);
}

public interface ObservableDevice {

    void addObserver(DeviceObserver observer);
    void removeObserver(DeviceObserver observer);
    void notifyObservers(String eventDescription);
    void notifyObservers(LogType eventType, String eventDescription);

}

public interface SensorDevice<T> {

    public T readValue();
    public String getUnit();

}

public interface Switchable {

    public void turnOn();
    public void turnOff();
    public boolean isOn();

}
```

Instrukcja uruchomienia i obsługi

W folderze skompresowanym został przygotowany skompilowany plik .jar, należy go uruchomić przez konsolę – java --enable-preview -jar .\SmartHouse.jar

Cała obsługa programu następuje poprzez podanie komendy w formie pojedynczej litery, lub cyfry reprezentującej wybór zarejestrowanego domu, pokoju, urządzenia, lub reguły, a następnie wciśnięcie klawisza Enter by zatwierdzić polecenie. Po wykonaniu jednego polecenia program pokazuje dostępne polecenia które jest w stanie wykonać w aktualnej chwili.

```
Windows PowerShell
Dostępne polecenia: (D)odaj dom
Podaj polecenie: d
ID domu: [1]
Nazwa domu: Wakacyjny domek
Szerokość geograficzna (12,34567): 51,11142
Wysokość geograficzna (12,34567): 67,55439

Zarejestrowane domy:
[1]      (51,11142; 67,55439)      Wakacyjny domek

Dostępne polecenia: (D)odaj dom, (U)suń dom, (Z)mień nazwę domu
lub podaj numer domu
Podaj polecenie: 1

Zarządzasz teraz domem: [1] Wakacyjny domek
Brak zarejestrowanych pokoi w tym domu.
Dostępne polecenia: (W)róż, (D)odaj pokój
Podaj polecenie: d
ID pokoju: [1]
Nazwa pokoju: Kuchnia na parterze
Dostępne kategorie: KUCHNIA TOALETA KORYTARZ SYPIALNIA SALON PODDASZE
Kategoria pokoju: kuchnia
Temperatura pokojowa dla symulacji ogrzewania (zmiennoprzecinkowa): 21

Zarządzasz teraz domem: [1] Wakacyjny domek
Zarejestrowane pokoje:
[1|KUCHNIA] Kuchnia na parterze
Dostępne polecenia: (W)róż, (D)odaj pokój, (U)suń pokój, (Z)mień nazwę pokoju
lub podaj numer pokoju
Podaj polecenie: |
```

Plik log_<timestamp>.tsv tworzy się w miejscu uruchomienia programu, zaraz po zarejestrowaniu pierwszego pokoju.

```
log_20250527_215950.tsv
Plik  Edytuj  Wyświetl

# Log start: 2025-05-27 21:59:50
2025-05-27 22:01:38 temp001 TEMPERATURE Kuchnia na parterze DEVICE_ADDED Dodano nowe urządzenie do pokoju.
2025-05-27 22:01:38 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 27.670719751112014
2025-05-27 22:01:39 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 27.320538329594108
2025-05-27 22:01:40 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 26.574725411748148
2025-05-27 22:01:41 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 25.789351087017025
2025-05-27 22:01:42 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 25.114034349529216
2025-05-27 22:01:43 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 25.289711286266655
2025-05-27 22:01:44 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 24.743264076151124
2025-05-27 22:01:45 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 23.410847309656944
2025-05-27 22:01:46 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 24.611143135320763
2025-05-27 22:01:47 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 25.751273734547212
2025-05-27 22:01:48 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 25.973452731396087
2025-05-27 22:01:49 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 26.22531867124041
2025-05-27 22:01:50 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 25.384682001370027
2025-05-27 22:01:51 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 24.208050710360908
2025-05-27 22:01:52 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 24.476523909572343
2025-05-27 22:01:53 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 26.10305694641997
2025-05-27 22:01:54 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 25.488184295532356
2025-05-27 22:01:55 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 24.20070875370269
2025-05-27 22:01:56 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 21.774388437625348
2025-05-27 22:01:57 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 22.742730086408
2025-05-27 22:01:58 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 21.63491647949864
2025-05-27 22:01:59 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 20.604255785070094
2025-05-27 22:02:00 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 19.300542711408205
2025-05-27 22:02:01 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 21.822254030611106
2025-05-27 22:02:02 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 23.724573298216935
2025-05-27 22:02:03 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 23.035694091171234
2025-05-27 22:02:04 hum001 HUMIDITYSENSOR Sypialnia na pi@trze DEVICE_ADDED Dodano nowe urządzenie do pokoju.
2025-05-27 22:02:04 hum001 HUMIDITYSENSOR Sypialnia na pi@trze READING Humidity: 14.577972156413095
2025-05-27 22:02:04 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 24.502261713320255
2025-05-27 22:02:05 hum001 HUMIDITYSENSOR Sypialnia na pi@trze READING Humidity: 13.574672498803924
2025-05-27 22:02:05 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 24.892144217721913
2025-05-27 22:02:06 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 25.024639036872433
2025-05-27 22:02:07 hum001 HUMIDITYSENSOR Sypialnia na pi@trze READING Humidity: 14.554620430190438
2025-05-27 22:02:07 temp001 TEMPERATURE Kuchnia na parterze READING Temperature: 25.339557870909584
2025-05-27 22:02:08 hum001 HUMIDITYSENSOR Sypialnia na pi@trze READING Humidity: 14.327000057779214

Wiersz 8, kolumna 26  9 802 znaki  100%  Windows (CRLF)  UTF-8
```