UNIVERSIDAD NACIONAL DE COLOMBIA

FACULTAD DE INGENIERÍA

OBJECT ORIENTED PROGRAMMING

ENGINERING FACULTY

WORKSHOP 2

AUTHORS:

Eddy Johan Tocancipa Muñoz

Valeria Aranda Pacheco

Luis Santiago Vargas Rodríguez

Bogotá DC.2025

FUNCTIONAL REQUIREMENTS

1. The system must display the project name (“ElecHouse”) on load.

2. It must show an interactive image of a house divided into Bedroom, Living Room, and Kitchen.

3. The user must be able to select a room by clicking on its section.

4. When a room is selected, the system must zoom in smoothly to that area, showing its layout.

5. The right-side panel must display room-specific components:
Bedroom → bulbs, lamps, light fixtures.
Living Room → TV, radio, speakers.
Kitchen → heat, gas, or humidity sensors.

6. The panel must also include common elements, such as outlets or connectors.

7. The user can drag and drop components from the panel to the workspace.

8. The system must allow connecting components with virtual cables that visually represent electric connections.

9. The system must support deleting, rotating, or moving placed components freely within the room.

10. It must be possible to assign electrical values or configuration parameters (voltage, power, resistance, sensitivity, etc.).

11. The system must include a “Test” button to run the circuit simulation.

12. If connections are correct, components should behave as expected:
Bedroom → lights turn on with adjustable brightness.
Living Room → devices display proper visuals and sound.
Kitchen → sensors respond accurately to simulated heat or gas levels.

13. If connections or values are incorrect, the system must display error messages or visual alerts (e.g., red lines, pop-up warnings).

14. The user should be able to reset the current room to remove all components and start again.

15. The user could save a custom-built circuit design for later review.

16. A “Back to House” button must allow users to return to the main house screen.

17. The system must visually highlight active or powered-on components during simulation (glowing lights or blinking LEDs).

18. The program must provide basic feedback messages after simulation (Simulation successful or Incorrect wiring detected).

19. The system may include an educational pop-up or guide explaining basic circuit rules (optional feature)

NON-FUNCTIONAL REQUIREMENTS

1. Usability: The interface must be simple and intuitive, with easily recognizable icons and logical component placement.

2. Accessibility: Texts, buttons, and clickable areas must be large, clear, and suitable for users of different ages and skill levels.

3. Performance:
The initial load time should not exceed 10 seconds.
Simulations should run smoothly, without noticeable delays.

4. Responsiveness: The system must adapt properly to different screen sizes (desktop, tablet, laptop).

5. Scalability: The codebase should allow adding new rooms, components, or sensors with minimal modification.

6. Maintainability: The project structure must be modular and well-documented for future developers.

7. Security: Saved data (user circuits) must be stored locally and securely, without exposing user information.

8. Privacy: No personal data (login, name, or email) should be required for basic use.

9. Interactivity and Realism: Animations for lighting, sound, or alarms must realistically simulate real home-automation behavior.

10. Reliability: The system must function consistently under normal use, without crashes or data loss.

11. Error Handling: The application must provide clear error messages in case of invalid actions (e.g., wrong connections, missing cables).

12. Educational Value: The design must promote learning outcomes, helping users understand how circuits function in everyday home environments.

13. Aesthetics: The color palette and visual design should reflect a modern, clean, and technological style, consistent across all pages.

14. Extensibility: Future integration with databases or sensor APIs should be feasible without major restructuring

CRC CARDS

Class: House	
Responsibilities:	**Collaborators:**
It displays the project title and main menu.	Rooms
Load the general house layout (bedroom, living room, kitchen).	Component Panel
Detect clicks on rooms and zoom into the selected one.	Connection Simulator
Load the corresponding side panel according to the selected room.	

Class: Kitchen	
Responsibilities:	**Collaborators:**
Graphically represent a kitchen.	Kitchen Electronics
Serves as a structure for the classes attached to the kitchen behaviour.	Bug
Contains the furniture of a kitchen.	Temperatures

Class: Bedroom	
Responsibilities:	**Collaborators:**
Graphically represent a Bedroom.	Bedroom electronics
Serves as a structure for the classes attached to the kitchen behaviour.	
Contains the furniture of a Bedroom..	

Class: Living Room	
Responsibilities:	**Collaborators:**
Graphically represent a Living Room.	Living Room Electronics
Serves as a structure for the classes attached to the living room behaviour.	
Contains the furniture of a Livingroom.	

Class: Kitchen Electronics	
Responsibilities:	**Collaborators:**
Represents sensors	Bulb
Store its electrical properties (voltage, polarized)	Heat Sensor
Respond to events (turn on/off, connection error, alarm).	Movement Sensor
	Resistance

Class: Living Room Electronics	
Responsibilities:	**Collaborators:**
Represents Electrodomestics	Resistance
Store its electrical properties (voltage, power)	TV
Respond to events (turn on/off, connection error, display, audio).	Radio

Class: Bedroom Electronics	
Responsibilities:	**Collaborators:**
Represent Lights	Resistance
Store its electrical properties (voltage, Luminosity)	Bulb
Respond to events (turn on/off, connection error, light).	Desk Lamp
	Lamp

Class: Cables	
Responsibilities:	**Collaborators:**
Represent connections between components.	Kitchen, Bedroom, Livingroom Electronics
Verify electrical continuity.	Plug
It Detects connection errors (short circuits, missing ground, etc.).	Test
Conducts current	Control Panel

Class: Test	
Responsibilities:	**Collaborators:**
Analyze user-made connections.	Control Panel, Kitchen, Bedroom, Living Room
Determine if the circuit is correctly assembled.	Kitchen, Bedroom, Livingroom Electronics
Execute the test and display the results (lights on, sensors activated, errors).	Cable
Provide visual or sound feedback.	Result Panel

Class: Result Panel	
Responsibilities:	**Collaborators:**
Display messages about the simulation status.	Connection Simulator
Indicate whether the components work properly or have malfunctions.	Rooms
Show specific alerts (Sensor not calibrated correctly, Short circuit in living room).	

Class: Resistance	
Responsibilities	**Collaborators**
Allows to regulate the current	Electronic Elements
Prevents electronics to burn	Plug
Gives ohms	Cables

Class: Bulb	
Responsibilities	**Collaborators**
Generates light	Resistance
Gives differents intensities of light	Cables
Consume Volts	

Class: Lamp	
Responsibilities	**Collaborators**
Can give differents colors of lights	Resistance
Gives differents intensities of lights	Cables
Consume volts	

Class: Desk Lamp	
Responsibilities	**Collaborators**
Gives differents intensities of lights	Resistance
Consume Volts	Cables

Class: Heat Sensor	
Responsibilities	**Collaborators**
Detects high temperatures	Resistance
Can give an alarm when temperatures are high	Cables
Parameters vary by volts received	Bulb

Class: Motion Sensor	
Responsibilities	**Collaborators**
Detects Movement	Resistance
Can give an alarm when it detects movements.	Cables
Parameters vary by volts received	Bulb

Class: TV	
Responsibilities	**Collaborators**
Gives display	Stereo
Gives audio	Resistance
Consume watts	Cables

Class: Radio	
Responsibilities	**Collaborators**
Gives audio	Stereo
Plays music	Resistance
Plays news	Cables
Plays podcast	
Consume watts	

Class: Plug	
Responsibilities	**Collaborators**
Gives volts	Electronic Components
Give a terminal connection to the cables	Cables
Gives earth	

USER STORIES

Title: Bedroom lighting simulation	*Priority: high*	*Estimate: 5 hours*
User story: As a student learning about home electricity, I want to connect lamps and bulbs to a power source in the bedroom, so that I can understand how lighting circuits work and how brightness depends on power		
Acceptance criteria: Given the bedroom environment is loaded, When I drag a bulb and connect it correctly to the outlet with cables, Then the bulb should turn on when I press “Test” showing adjustable brightness if power values change.		

Title: Living room electronics connection	*Priority: medium*	*Estimate: 4 hours*
User story: As a user exploring how electronic devices use power, I want to connect and test appliances like a TV or a radio, so that I can see how energy distribution and correct wiring affect their proper function.		
Acceptance criteria: Given the living room is selected, when I drag and connect a TV or radio to a power outlet using the “Cables” tool, Then the devices should work correctly (sound and image visible) when I press test, if the connections are wrong then the system, should display an error, message or warning icon		

Title: Kitchen sensor calibration	*Priority: high*	*Estimate: 6 hours*
User story: As a learner interested in smart home sensors, I want to connect and calibrate temperature and gas sensors, in the kitchen, so that I can see how sensors react to heat or gas levels and understand the importance of correct calibration.		
Acceptance criteria: Given the kitchen environment is active, when I place and connect sensors to a power outlet, then the system should allow setting calibration values, if it’s correct, the alarm triggers only when high heat or gas levels occur, if its not then the alarm will trigger too early (for example while cooking)		

Title: Save and load circuit design	*Priority: Medium*	*Estimate: 3 hours*
User story: As a user working on my circuit design, I want to save and load my custom connections, so that I can continue my learning or modify previous setups later.		
Acceptance criteria: Given I have created a circuit design When i click “Save” then the system should store my design locally. When i click “Load” then my previous layout and component values should appear exactly as before.		

Title: Error feedback and guidance	*Priority: Low*	*Estimate: 2 hours*
User story: As a beginner user, I want to receive helpful messages when I make mistakes, so that I can understand what went wrong and fix my circuit easily.		
Acceptance criteria: Given a connection or value is incorrect when I press “Test”, then the system should show a clear error message, highlight wrong connections, and suggest possible fixes		

Title: Power consumption indicator	*Priority: high*	*Estimate: 4 hours*
User story: As a user interested in energy efficiency I want to see the total power consumption of connected devices, so that I can learn how much electricity is used by each circuit.		
Acceptance criteria: Given devices are connected in any room, When i press “Show consumption” then the system should display total wattage and energy usage in real time		

ADDITIONS

Class: Bug	
Responsibilities	**Collaborators**
Gives utility to the movement sensor	Movement Sensor
It moves during the “Test” in the kitchen	Temperature
Can die if the temperature is too high	Kitchen

Class: Temperature	
Responsibilities	**Collaborators**
Gives utility to the heat sensor	Heat Sensor
It increases and decreases during the “Test” in the kitchen	Bug
	Kitchen

Class: Control Panel	
Responsibilities	**Collaborators**
Contains several options for the control of the electronics and parameters during the usage of the program	Load
Serves as a structure to the control buttons	Save
	Test
	Cables
	Return
	Parameters

Class: Save	
Responsibilities	**Collaborators**
Saves the parameters and positions of the electronics that were used in the room	Control Panel
	Parameters
	Load

Class: Load	
Responsibilities	**Collaborators**
Changes the parameters and positions of electronics to the ones that were saved	Control Panel
	Parameters
	Save

Class: Parameters	
Responsibilities	**Collaborators**
Changes the Voltage, Resistance and power values, depending of each electronic	Kitchen, Living Room, Bedroom Electronics
Each electronic will have a standart, if the parameters of the electronic dont match up for the standart, it will not work.	Save
	Load
	Test
	Control Panel

Class: Return	
Responsibilities	**Collaborators**
Return the display to the “House” Class	Control Panel
	House

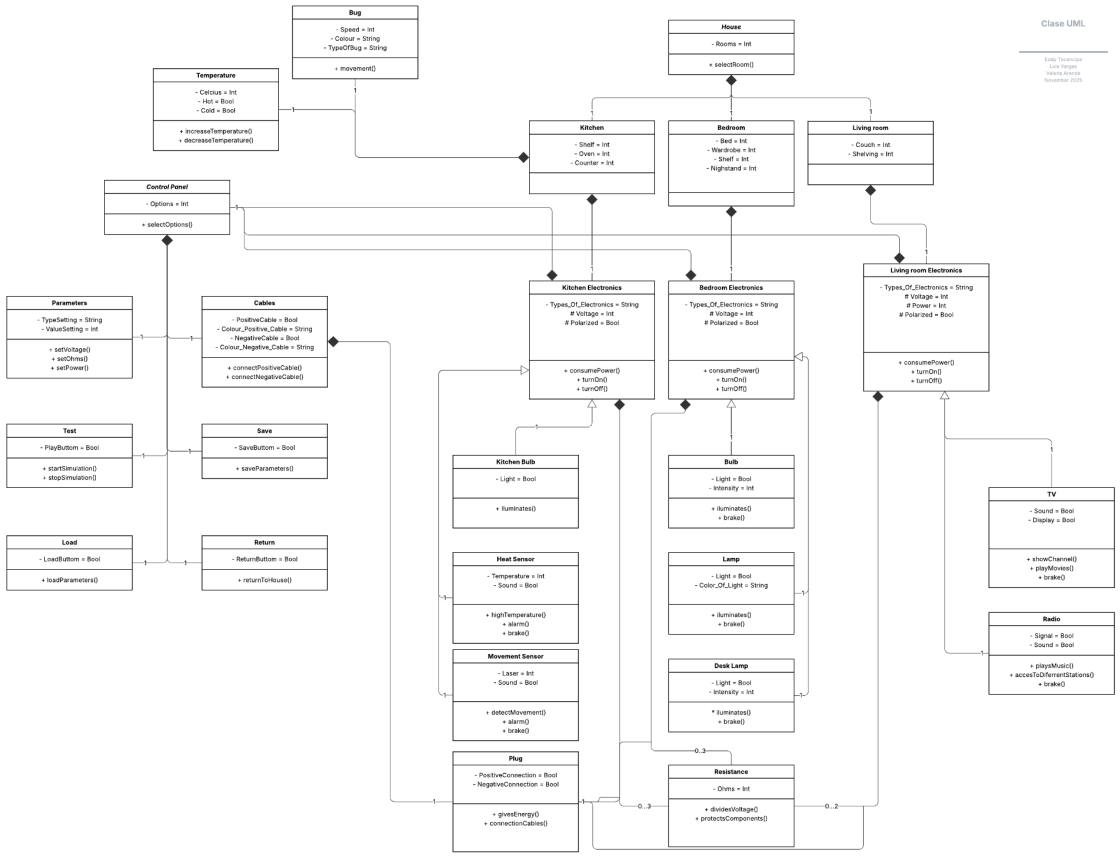
Class: Electronic components	
Responsibilities	**Collaborators**
1. Represent a generic electronic component with properties common to all electrical devices in the system. 2. Store and manage fundamental electrical attributes, such as: voltage intensity resistance connected (ON/OFF state) Encapsulate basic component behavior through methods that control its operational state: turn_on() → Activates the component. turn_off() → Deactivates the component. consumePower() → Simulates power consumption. 4. Provide a common interface that supports inheritance and polymorphism for derived classes (e.g., Bulb, Lamp, Resistance, Sensor). 5. Ensure internal consistency by preventing direct external manipulation of attributes (principle of encapsulation).	Bulb → Inherits from ElectronicComponent and redefines methods such as iluminates() to simulate light emission. Lamp → Extends Bulb, adding specific attributes such as light color or intensity. Resistance → Uses base electrical attributes to simulate voltage division and protection mechanisms. Sensor (e.g., HeatSensor) → Utilizes the common structure to detect variations in temperature or energy. Room, Kitchen, Bedroom, LivingRoom → Incorporate one or more ElectronicComponent objects, simulating a complete smart home environment. In general, the most important attributes of electronic components are those that ensure maintenance and code reuse.

DOCUMENTATION

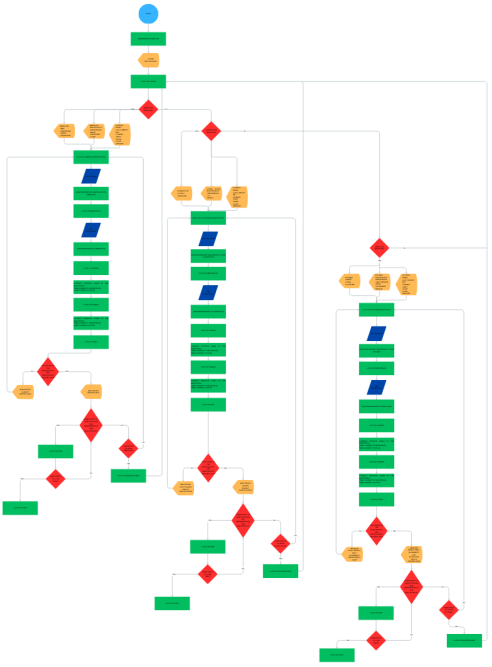
1. We decided to eliminate the “Stereo” Class from the living room because it was redundant.
2. The class “Electronic Element” is now divided into 3 electronics elements, one for each room.
3. The class “Rooms” has been divided into each room intended for the house.

4. Several class names have been changed to match better to their respective buttons showcased at the mockup.
5. Several new classes were added, those were some of the ones that we forgot to add at the first document.
6. Electronic components and its sons that inherit from it like Livingroom, Kitchen and Bedroom electronic components were added.

UML AND FLOWCHART



Link for better visualization:
https://lucid.app/lucidchart/27095361-3fcd-488d-a12f-722610d325cc/edit?invitationId=inv_a4770661-2954-48a7-a42c-7940bfa077f7



Link for better visualization:
https://www.canva.com/design/DAG3wENH1Oo/Ec0g_QpoPjptPbw-pm2Ijg/edit?utm_content=DAG3wENH1Oo&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Summary of the Implementation of Encapsulation, Inheritance, and Polymorphism

Encapsulation:

In the code, each class defines its own attributes and methods to manage its internal behavior, reflecting the principle of encapsulation.

Although access modifiers such as __attribute (private) or _attribute (protected) are not used, the structure demonstrates a proper separation between the internal logic of the objects and their external use.

Methods like turn_on(), turn_off(), and consumePower() are responsible for modifying the state of the object without directly exposing its attributes, ensuring adequate control over data access and modification.

Inheritance:

The project clearly applies the principle of inheritance, allowing code reuse and the creation of hierarchies among different electronic components.

The base class ElectronicComponent serves as the foundation for most devices (Bulb, Lamp, Resistance, HeatSensor, among others), providing common attributes such as voltage, intensity, and resistance, as well as general methods like turn_on() and consumePower().

The code also presents multiple inheritance in classes such as Kitchen, Bedroom, and LivingRoom, which inherit from both Room and their corresponding electronic classes (KitchenElectronics, BedroomElectronics, LivingRoomElectronics).

This approach allows the combination of characteristics from different domains, such as room furniture and the electronic devices present within them.

A clear example of specialization through inheritance is observed in the Lamp class, which extends Bulb and redefines the iluminates() method to include the description of the emitted light color.

Polymorphism:

The principle of polymorphism is demonstrated through method overriding in different classes.

The iluminates() method, present in Bulb, is redefined in Lamp and DeskLamp to modify its behavior according to the specific lamp type.

Thus, each class responds differently to the same method, adapting its execution to the context of the object.

Another example of polymorphism is found in the brake() method, which exists in several classes (Bulb, HeatSensor, Radio, TV, DeskLamp, etc.), each implementing it differently to represent various types of failures.

This approach allows different objects to be handled through a common interface, making the code more flexible and easier to maintain.

Proposed Directory Structure

To keep the project organized and scalable, the following directory structure is proposed:

```
home_automation/
 |---main.py                    # Main program file
 |
 |---components/          # Electronic components
 |---_init_.py
 |--- electronic_component.py
 |--- resistance.py
 |---bulb.py
 |---lamp.py
 |---sensors.py
 |---cables.py
 |---plug.py
 |--- rooms/                          #  Room-related classes
 |      --- room.py
 |      --- kitchen.py
 |      --- bedroom.py
 |      --- living_room.py
 |---environment/          #          Environmental elements
 |      ├── temperature.py
 |      ├── bug.py
 |      └── return_to_house.py
 |--- tests/                          # System tests
 |--- _init_.py
```

|---test_components.py

INITIAL CODE

```python
#Base class for electronic components
class ElectronicComponent:
    # Constructor for a general electronic component
     def __init__(self, name:str, voltage:float, intensity:float,
resistance:float, connected:bool=False):
        self.name = name                    # Name of the component
        self.voltage = voltage              # Voltage value
        self.intensity = intensity          # Current intensity
        self.resistance = resistance        # Resistance value
        self.connected = connected          # Connection state

    # Turns the component ON
    def turn_on(self):
        self.connected = True
        print(f"{self.name} is ON")

    # Turns the component OFF
    def turn_off(self):
        self.connected = False
        print(f"{self.name} is OFF")

    # Checks if the component consumes power
    def consumePower(self):
        if self.connected:
            print(f"{self.name} is consuming power")
        else:
            print(f"{self.name} is not consuming power")

class Resistance(ElectronicComponent):
    # Constructor for a resistor
    def __init__(self, resistance):
         self.name = "Resistance"           # Fixed name for this
class
```

```
        self.voltage = None                 # Voltage not defined
here
        self.intensity = None               # Current not defined
here
        self.resistance = resistance      # Sets the ohm value
        self.connected = False
           self.ohms = resistance                # Stores the same
resistance as attribute

    # Simulates voltage division
    def dividesVoltage(self):
        print("Voltage has been divided")
    # Simulates protection of electronic components
    def protectsComponents(self):
        print("Components have been protected")

class Bulb(ElectronicComponent):
    # Constructor for a bulb
    def __init__(self, intensity=None, light:bool=False):
        self.name = "Bulb"
        self.voltage = None
        self.intensity = intensity        # Light intensity
        self.resistance = None
        self.connected = False
        self.light = light                # Light state
    # Checks if bulb illuminates
    def iluminates(self):
        if self.connected:
            print("Bulb is illuminating")
        else:
            print("Bulb cannot illuminate because it's OFF")

    # Simulates a broken bulb
    def brake(self):
        print("Bulb is broken")

class Lamp(Bulb):
    # A lamp that extends bulb
    def __init__(self, color_of_light:str):
        self.name = "Lamp"
        self.voltage = None
        self.intensity = None
        self.resistance = None
```

```
        self.connected = False
        self.light = False
         self.color_of_light = color_of_light   # Color of emitted
light

    # Lamp illumination with color
    def iluminates(self):
        if self.connected:
                            print(f"Lamp  is  illuminating  with
{self.color_of_light} light")
        else:
            print("Lamp cannot illuminate because it's OFF")

class DeskLamp(Lamp):
    # Desk lamp with default white light
    def __init__(self, intensity=None):
        self.name = "Desk Lamp"
        self.voltage = None
        self.intensity = intensity
        self.resistance = None
        self.connected = False
        self.light = False
          self.color_of_light = "white"      # Always white light
for desk lamp

    # Desk lamp illumination
    def iluminates(self):
                print(f"Desk  Lamp  illuminates  with  intensity
{self.intensity}")
    # Simulates desk lamp damage
    def brake(self):
        print("Desk lamp is broken")

class Room:
    # Generic room in the house
    def __init__(self, name:str):
        self.name = name
    # Shows the selected room
    def selectRoom(self):
        print(f"You are in the {self.name}")

class HeatSensor(ElectronicComponent):
    # Heat sensor with temperature detection
```

```
    def __init__(self, temperature=None):
        self.name = "HeatSensor"
        self.voltage = None
        self.intensity = None
        self.resistance = None
        self.connected = False
        self.temperature = temperature
        self.sound = False                  # Alarm sound state
    # Detects high temperature
    def highTemperature(self):
        if self.temperature:
            print("High temperature detected!")
    # Activates alarm
    def alarm(self):
        self.sound = True
        print("HEAT ALARM ACTIVE")
    # Simulates sensor damage
    def brake(self,):
        print("Heat Sensor damaged")


class MovementSensor(ElectronicComponent):
    # Detects motion using a laser
    def __init__(self, laser=None):
        self.name = "MovementSensor"
        self.voltage = None
        self.intensity = None
        self.resistance = None
        self.connected = False
        self.laser = laser                  # Laser state
        self.sound = False
    # Motion detected
    def detectMovement(self):
        print("Movement detected!")
    # Activates alarm
    def alarm(self):
        self.sound = True
        print("MOVEMENT ALARM ACTIVE")
    # Simulates sensor damage
    def brake(self):
        print("Movement Sensor damaged")
```

```
class KitchenElectronics(ElectronicComponent):
    # Electronic appliances inside kitchen
    def __init__(self, type_of_electronics:str):
        self.name = "Kitchen Electronics"
        self.type_of_electronics = type_of_electronics
        self.voltage = None
        self.polarized = False
        self.intensity = None
        self.resistance = None
        self.connected = False


class Kitchen(Room,KitchenElectronics):
    # Represents a kitchen with furniture
    def __init__(self, shelf:int, oven:int, counter:int):
        self.name = "Kitchen"
        self.shelf = shelf
        self.oven = oven
        self.counter = counter


class BedroomElectronics(ElectronicComponent):
    # Electronic devices inside bedroom
    def __init__(self, type_of_electronics:str):
        self.name = "Bedroom Electronics"
        self.type_of_electronics = type_of_electronics
        self.voltage = None
        self.polarized = False
        self.intensity = None
        self.resistance = None
        self.connected = False


class Bedroom(Room,BedroomElectronics):
    # Represents bedroom furniture
        def __init__(self, bed:int, wardrobe:int, shelf:int,
nightstand:int):
        self.name = "Bedroom"
        self.bed = bed
        self.wardrobe = wardrobe
        self.shelf = shelf
        self.nightstand = nightstand
```

```
class LivingRoomElectronics(ElectronicComponent):
    # Electronic devices in living room
    def __init__(self, type_of_electronics:str, power=None):
        self.name = "Living Room Electronics"
        self.type_of_electronics = type_of_electronics
        self.power = power                  # Power consumption
        self.voltage = None
        self.polarized = False
        self.intensity = None
        self.resistance = None
        self.connected = False


class LivingRoom(Room,LivingRoomElectronics):
    # Living room furniture
    def __init__(self, couch:int, shelving:int):
        self.name = "Living Room"
        self.couch = couch
        self.shelving = shelving


class TV(LivingRoomElectronics):
    # Television object
    def __init__(self):
        self.name = "TV"
        self.type_of_electronics = "TV"
        self.power = None
        self.voltage = None
        self.polarized = False
        self.intensity = None
        self.resistance = None
        self.connected = False
        self.display = False                # Display state
        self.sound = False                  # Sound state

    # Shows a TV channel
    def showChannel(self):
        if self.connected:
            print("Showing channel")
        else:
            print("TV is OFF")
    # Plays a movie
```

```
    def playMovies(self):
        print("Playing movie")
    # Damaged TV
    def brake(self):
        print("TV broken")


class Radio(LivingRoomElectronics):
    # Radio object
    def __init__(self):
        self.name = "Radio"
        self.type_of_electronics = "Radio"
        self.power = None
        self.voltage = None
        self.polarized = False
        self.intensity = None
        self.resistance = None
        self.connected = False
        self.signal = False
        self.sound = False
    # Plays music
    def playsMusic(self):
        print("Music is playing")
    # Changes station
    def accessToDifferentStations(self):
        print("Changing radio station")
    # Damaged radio
    def brake(self):
        print("Radio broken")


class Wire(ElectronicComponent):
    # Electrical wire
    def __init__(self, length=None, material=None):
        self.name = "Wire"
        self.voltage = None
        self.intensity = None
        self.resistance = None
        self.connected = False
        self.length = length
        self.material = material
```

```python
class Cables:
    # Cable connections and colors
    def __init__(self, positive_color:str, negative_color:str):
        self.positive = False
        self.negative = False
        self.positive_color = positive_color
        self.negative_color = negative_color
    # Connects the positive cable
    def connectPositiveCable(self):
        print("Positive cable connected")
    # Connects the negative cable
    def connectNegativeCable(self):
        print("Negative cable connected")


class Temperature:
    # Manages temperature values
    def __init__(self,celsius:float,hot:bool,cold:bool):
        self.celsius = celsius
        self.hot = hot
        self.cold = cold
    # Increases temperature and triggers AC
    def increaseTemperature(self,celsius):
        if celsius>20:
            print("its warm")
            print("turning on the air conditioning")
    # Decreases temperature and triggers heating
    def decreaseTemperature(self,celsius):
        if celsius<5:
            print("its cold")
            print("turning on the heating")


class Bug:
    # Small entity with movement
             def    __init__(self,    speed=None,    color:str=None,
bugType:str=None):
        self.speed = speed
        self.color = color
        self.type = bugType
    # Bug movement
    def movement(self):
        print("Bug is moving")
```

```python
class Plug:
    # Power plug that supplies energy
    def __init__(self):
        self.positive_connection = False
        self.negative_connection = False
    # Supplies energy
    def givesEnergy(self):
        print("Energy supplied")
    # Cables connected to the plug
    def connectionCables(self):
        print("Cables connected to plug")


class return_to_house:
    # Class to return system to home
    def __init__(self):
        self.returnButton=False
    # Executes return action
    def returnToHouse(self):
        print("Returning to house")
```

DOCUMENTATION

This code implements a basic object-oriented simulation of a smart home system. Each class models a real component inside a house, such as electronic devices, lamps, sensors, rooms, and wiring. The main parent class, ElectronicComponent, defines common properties like voltage, intensity, resistance, and the ability to turn on, turn off, and consume energy. Other classes inherit and extend this functionality to represent specific devices including bulbs, resistances, televisions, radios, heat sensors, and movement sensors. Additional classes model house environments such as the kitchen, bedroom, and living room, combining physical attributes with electronics. The purpose of this system is to abstract how home automation works: rooms hold devices, devices perform actions, and sensors trigger alarms or responses. This structure can be used as the foundation for a larger interactive simulation, a smart-home controller, or a learning tool to demonstrate inheritance, abstraction, and behavior modeling in object-oriented programming.