# University Of Science & Technology Chittagong (USTC)

## Faculty of Science and Engineering Technology (FSET)

## Department of Computer Science Engineering (CSE)

Lab Project Documentation: **Eco-Friendly Smart Home Simulator (Java OOP)**

Course Code: **CSE 124**   Course Title: **Object Oriented Programming Language Lab**

Submitted To:

**Debabrata Mallick**

Lecturer,

CSE, FSET, USTC

Submitted By:

Name: **Durjoy Barua**

Roll: **24070134**   Reg: **1185**

Dept: **CSE**   Batch: **43rd**   Semester: **2nd**

Submission Date: **Thursday, 22 May, 2025**

## Table of Contents

**List of Figures:** .......... 3

**Eco-Friendly Smart Home Simulator (Java OOP)** .......... 4

**1. Proposal:** .......... 4

- **Project Goals:** .......... 5
- **Project Scope:** .......... 6
- **Problem Statement:** .......... 14
- **Methodology:** .......... 15

**2. Implementation:** .......... 16

- **Class Diagram (Partial):** .......... 16
- **Code:** .......... 17
- **Output:** .......... 20
- **Technology Stack:** .......... 27
- **System Architecture:** .......... 28
- **Object-Oriented Principles:** .......... 30

**3. Remaining features to implement:** .......... 31

**4. Future Work:** .......... 34

**5. Conclusion:** .......... 35

- **Final Product and its Utility:** .......... 35
- **Project Summary and Impact:** .......... 37

**6. References and Resources:** .......... 38

**7. Appendix:** .......... 44

## List of Figures:

- Figure 1: Full Class Diagram of the Eco-Friendly Smart Home Simulator
- Figure 2: Sample Console Output of the Eco-Friendly Smart Home Simulator
- Figure 3: Terminal Output (GUI) of the Eco-Friendly Smart Home Simulator

# Eco-Friendly Smart Home Simulator (Java OOP)

## 1. Proposal:

The purpose of this project is to develop an **interactive Eco-Friendly Smart Home Simulator** using Java and Object-Oriented Programming (OOP) principles. As energy costs continue to rise and environmental concerns escalate, homeowners increasingly need to understand and optimize their energy consumption. This simulator aims to empower users to design and experiment with virtual smart homes, explore various eco-friendly technologies, and analyze the energy consumption, cost, and overall environmental impact of their choices through a user-friendly interface. OOP is crucial for creating a modular, extensible, and maintainable system that can effectively model complex smart home environments.

## ➢ Project Goals:

**Develop a robust, extensible, and maintainable smart home simulation using Java:** This will involve creating a modular design utilizing Object-Oriented Programming (OOP) principles, ensuring the system is adaptable for future enhancements.

**Simulate the behavior of various smart home devices and systems:** This includes modeling lighting, HVAC, appliances, and renewable energy sources to create a comprehensive virtual environment.

**Model energy consumption, resource usage, and cost:** The simulator will accurately calculate these metrics based on user-defined configurations, device interactions, and dynamic environmental conditions.

**Provide a user-friendly interface for designing, configuring, and interacting with the simulated smart home:** This goal is actively being addressed through the implementation of a Graphical User Interface (GUI), which allows intuitive control and real-time feedback.

**Visualize simulation results through charts, graphs, and reports:** This will enable users to effectively assess the eco-friendliness and cost-effectiveness of their designs.

**Educate users about the benefits of eco-friendly technologies and practices:** The simulator will serve as an interactive learning tool within a smart home environment.

**Enable users to compare different scenarios and evaluate the impact of various design choices:** This will foster informed decision-making regarding energy efficiency and sustainability.

## ➢ Project Scope:

The simulator is designed to include key functionalities by implementing various features, evolving towards a comprehensive tool. The current implementation focuses on the core simulation logic and a foundational Graphical User Interface for interactive control.

**Current Implementation (Features Developed So Far):**

- **Core Simulation:**
  - A modular architecture that allows for easy addition of new devices and systems (demonstrated by extensible EnergyConsumer and RenewableEnergySource classes).
  - Discrete-event simulation to model the passage of time and the behavior of devices (hour-by-hour simulation).
  - Basic modeling of energy consumption for various devices.
  - Simulation of fundamental environmental factors:
    - Sunlight intensity (for solar panels)
    - Wind speed (for wind turbines)
  - Calculation of total energy consumption, cost, and generated renewable energy.

- **Smart Home Devices and Systems (Initial Models):**
    - **Lighting:**
        - Simulates SmartLight with adjustable brightness and occupancy sensor consideration.
    - **HVAC:**
        - Simulates SmartHVAC with target temperature and fan speed control.
    - **Appliances:**
        - Simulates SmartRefrigerator with internal temperature control.
    - **Renewable Energy:**
        - Simulates SolarPanel with user-configurable parameters (size, efficiency, sunlight intensity).
        - Simulates WindTurbine with user-configurable parameters (blade diameter, efficiency, wind speed).

- **<u>User Interface:</u>**
  - **Graphical User Interface (GUI):**
    - An interactive Swing-based GUI for controlling smart devices.
    - Displays a list of added devices and their current status.
    - Provides intuitive controls to turn devices ON/OFF.
    - Allows configuration of specific device settings through input dialogs (e.g., dimming for lights, target temperature and fan speed for HVAC).
    - Offers real-time display of device status updates.
  - **Command-Line Interface (CLI):**
    - Backend console output for detailed logging of simulation events, device status changes, and overall energy calculations per hour.

**Future Scope (Remaining Features to Implement):**

- **Core Simulation Enhancements:**
  - More realistic modeling of energy consumption for various devices, including standby power.
  - Advanced simulation of environmental factors such as:
    - Outdoor temperature (for HVAC, with more complex thermal modeling)
    - Occupancy patterns (automated device response)
    - Integration of seasonal variations or external weather data.
  - Calculation of carbon footprint.
- **Smart Home Devices and Systems Expansion:**
  - **Lighting:**
    - Simulate various types of light bulbs (CFL, incandescent) with different power ratings.
    - Model more advanced smart lighting features such as daylight harvesting.

- **HVAC:**
  - Simulate different HVAC systems (e.g., heat pumps) with varying efficiency ratings (SEER, HSPF).
  - Model thermostat zoning and energy recovery ventilation.
- **Appliances:**
  - Simulate major appliances (washing machines, dishwashers) with Energy Star ratings.
  - Model different usage patterns and energy consumption for each appliance.
- **Renewable Energy:**
  - Model energy storage (batteries) to store excess energy generated by renewable sources.
- **Water Management:**
  - Simulate water usage by fixtures (faucets, showers, toilets).

        - Model rainwater harvesting systems.

- **<u>User Interface Enhancements:</u>**
    - **Graphical User Interface (GUI):**
        - Interactive 2D representation of a home layout for visual device placement.
        - Drag-and-drop functionality to add and arrange devices within the layout.
        - Integration of charts and graphs to visualize simulation results (e.g., energy consumption over time, energy breakdown by device).
    - **Command-Line Interface (CLI):**
        - Further development of a structured text-based interface for programmatic control.
        - Options to load and save simulation configurations via CLI.
        - Batch processing mode to run multiple simulations with different parameters.

- **<u>Reporting and Analysis:</u>**
  - Generate comprehensive and customizable reports summarizing energy consumption, cost, and environmental impact.
  - Compare different simulation scenarios to evaluate the effectiveness of various energy-saving strategies.
  - Provide intelligent recommendations for optimizing energy efficiency based on simulation outcomes.
- **<u>Data Layer:</u>**
  - Handles loading and saving of complete simulation data and configurations (data persistence).

## ➢ Problem Statement:

As global energy costs continue to rise and environmental concerns, particularly regarding carbon emissions, become increasingly pressing, homeowners face a growing challenge in understanding and optimizing their energy consumption. Current methods for evaluating the efficiency and impact of smart home technologies and eco-friendly solutions often lack interactivity and a comprehensive, easy-to-understand visualization of their real-world implications. This makes it difficult for individuals to make informed decisions about sustainable home design and energy management, leading to potentially inefficient choices and missed opportunities for significant energy savings and reduced environmental footprints.

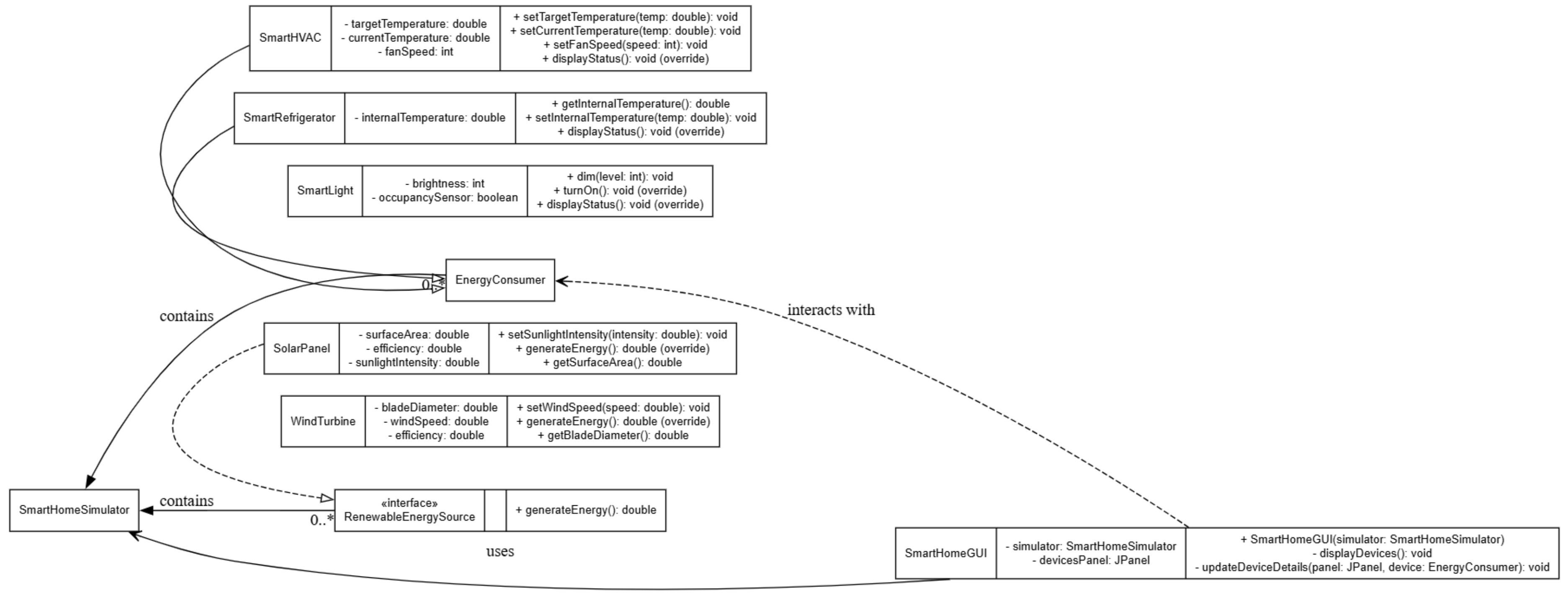
## ➢ Methodology:

The development of the Eco-Friendly Smart Home Simulator adheres to a structured and iterative methodology, emphasizing a modular, object-oriented design to ensure robustness, extensibility, and maintainability.

- Object-Oriented Design (OOD)
- Modular Architecture
- Simulation Approach
- Development Environment and Tools
- Iterative Development & Testing

## 2. Implementation:

### ➢ Class Diagram:

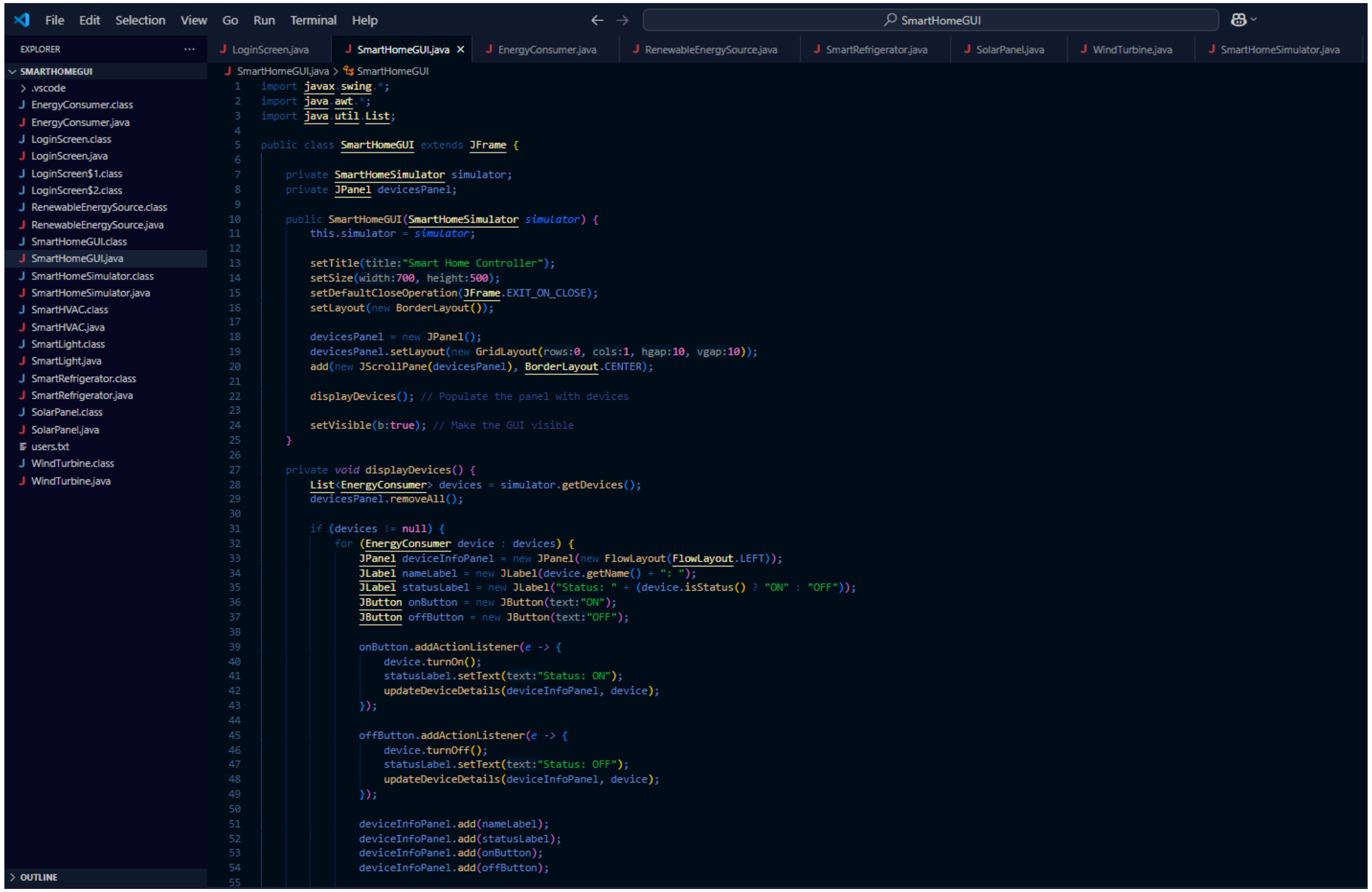


SmartHVAC
- targetTemperature: double
- currentTemperature: double
- fanSpeed: int
+ setTargetTemperature(temp: double): void
+ setCurrentTemperature(temp: double): void
+ setFanSpeed(speed: int): void
+ displayStatus(): void (override)
SmartRefrigerator
- internalTemperature: double
+ getInternalTemperature(): double
+ setInternalTemperature(temp: double): void
+ displayStatus(): void (override)
SmartLight
- brightness: int
- occupancySensor: boolean
+ dim(level: int): void
+ turnOn(): void (override)
+ displayStatus(): void (override)
EnergyConsumer
0..*
contains
interacts with
SolarPanel
- surfaceArea: double
- efficiency: double
- sunlightIntensity: double
+ setSunlightIntensity(intensity: double): void
+ generateEnergy(): double (override)
+ getSurfaceArea(): double
WindTurbine
- bladeDiameter: double
- windSpeed: double
- efficiency: double
+ setWindSpeed(speed: double): void
+ generateEnergy(): double (override)
+ getBladeDiameter(): double
SmartHomeSimulator
contains
0..*
«interface»
RenewableEnergySource
+ generateEnergy(): double
uses
SmartHomeGUI
- simulator: SmartHomeSimulator
- devicesPanel: JPanel
+ SmartHomeGUI(simulator: SmartHomeSimulator)
- displayDevices(): void
- updateDeviceDetails(panel: JPanel, device: EnergyConsumer): void

Figure 1: Full Class Diagram of the Eco-Friendly Smart Home Simulator

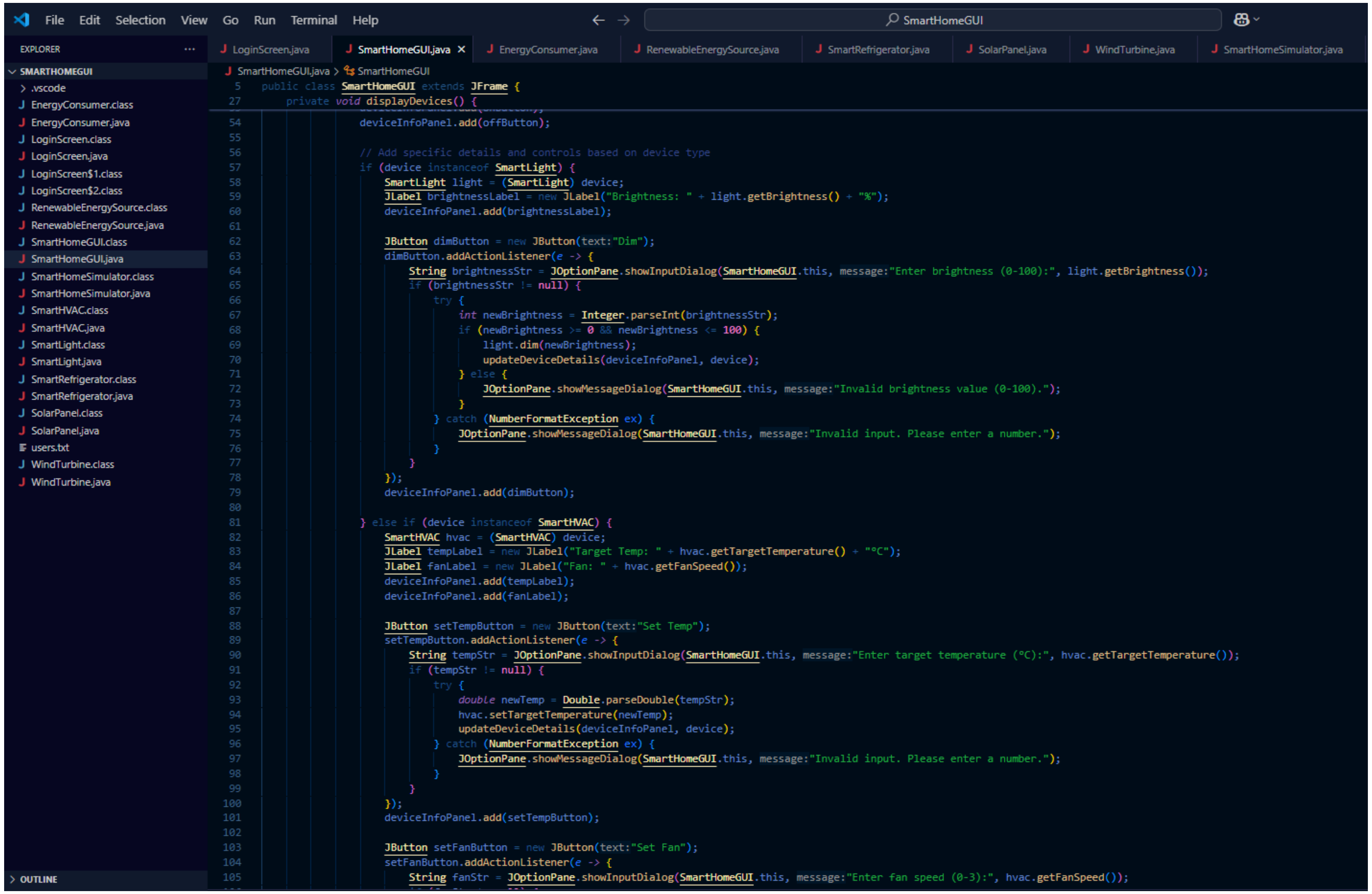
## ➢ Code:

- SmartHomeGUI.java



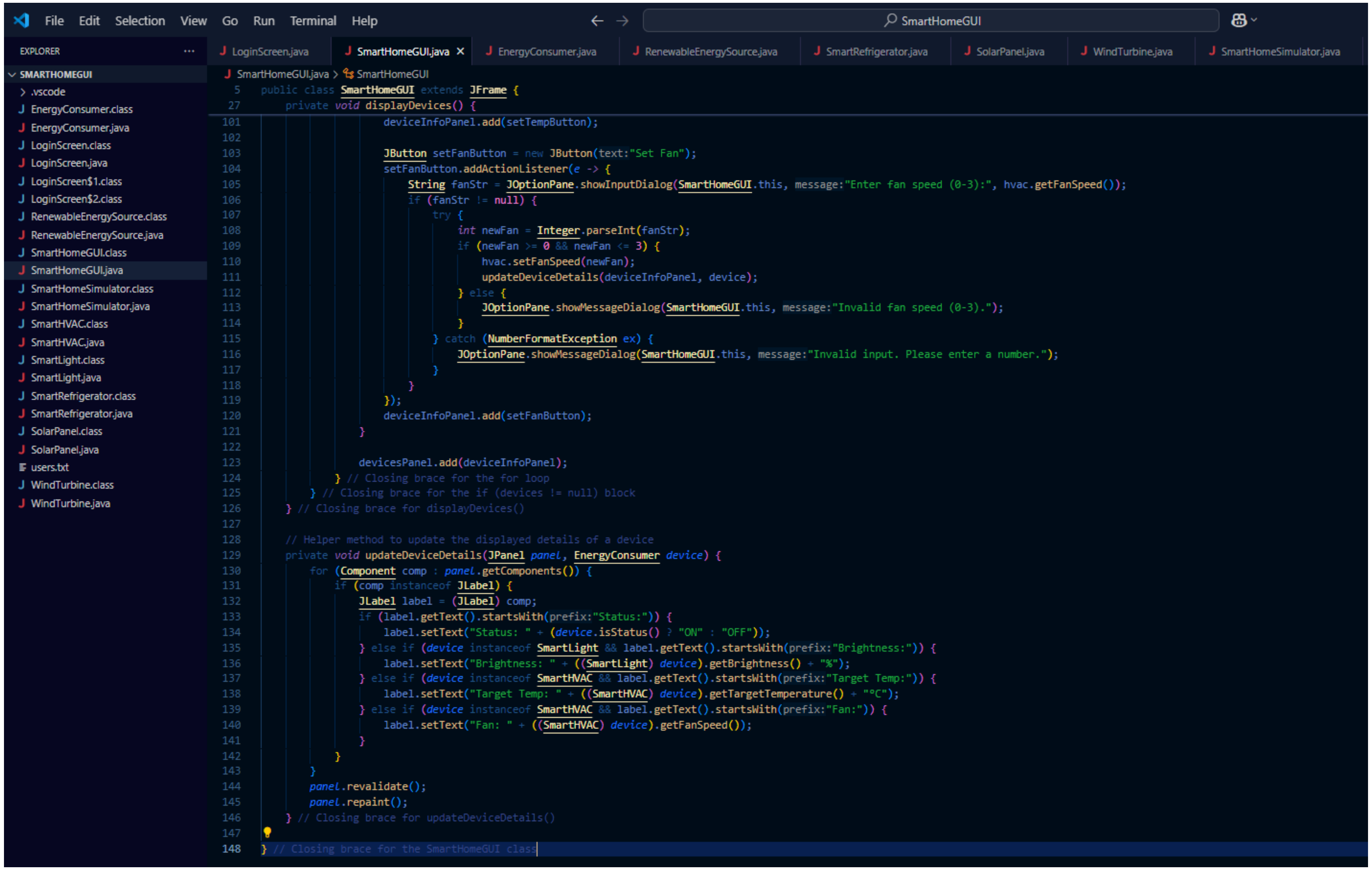
```java
import javax.swing.*;
import java.awt.*;
import java.util.List;

public class SmartHomeGUI extends JFrame {

    private SmartHomeSimulator simulator;
    private JPanel devicesPanel;

    public SmartHomeGUI(SmartHomeSimulator simulator) {
        this.simulator = simulator;

        setTitle(title:"Smart Home Controller");
        setSize(width:700, height:500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        devicesPanel = new JPanel();
        devicesPanel.setLayout(new GridLayout(rows:0, cols:1, hgap:10, vgap:10));
        add(new JScrollPane(devicesPanel), BorderLayout.CENTER);

        displayDevices(); // Populate the panel with devices

        setVisible(b:true); // Make the GUI visible
    }

    private void displayDevices() {
        List<EnergyConsumer> devices = simulator.getDevices();
        devicesPanel.removeAll();

        if (devices != null) {
            for (EnergyConsumer device : devices) {
                JPanel deviceInfoPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
                JLabel nameLabel = new JLabel(device.getName() + ": ");
                JLabel statusLabel = new JLabel("Status: " + (device.isStatus() ? "ON" : "OFF"));
                JButton onButton = new JButton(text:"ON");
                JButton offButton = new JButton(text:"OFF");

                onButton.addActionListener(e -> {
                    device.turnOn();
                    statusLabel.setText(text:"Status: ON");
                    updateDeviceDetails(deviceInfoPanel, device);
                });

                offButton.addActionListener(e -> {
                    device.turnOff();
                    statusLabel.setText(text:"Status: OFF");
                    updateDeviceDetails(deviceInfoPanel, device);
                });

                deviceInfoPanel.add(nameLabel);
                deviceInfoPanel.add(statusLabel);
                deviceInfoPanel.add(onButton);
                deviceInfoPanel.add(offButton);

```

- SmartHomeGUI.java



```java
public class SmartHomeGUI extends JFrame {
    private void displayDevices() {
                deviceInfoPanel.add(offButton);

                // Add specific details and controls based on device type
                if (device instanceof SmartLight) {
                    SmartLight light = (SmartLight) device;
                    JLabel brightnessLabel = new JLabel("Brightness: " + light.getBrightness() + "%");
                    deviceInfoPanel.add(brightnessLabel);

                    JButton dimButton = new JButton(text:"Dim");
                    dimButton.addActionListener(e -> {
                        String brightnessStr = JOptionPane.showInputDialog(SmartHomeGUI.this, message:"Enter brightness (0-100):", light.getBrightness());
                        if (brightnessStr != null) {
                            try {
                                int newBrightness = Integer.parseInt(brightnessStr);
                                if (newBrightness >= 0 && newBrightness <= 100) {
                                    light.dim(newBrightness);
                                    updateDeviceDetails(deviceInfoPanel, device);
                                } else {
                                    JOptionPane.showMessageDialog(SmartHomeGUI.this, message:"Invalid brightness value (0-100).");
                                }
                            } catch (NumberFormatException ex) {
                                JOptionPane.showMessageDialog(SmartHomeGUI.this, message:"Invalid input. Please enter a number.");
                            }
                        }
                    });
                    deviceInfoPanel.add(dimButton);

                } else if (device instanceof SmartHVAC) {
                    SmartHVAC hvac = (SmartHVAC) device;
                    JLabel tempLabel = new JLabel("Target Temp: " + hvac.getTargetTemperature() + "°C");
                    JLabel fanLabel = new JLabel("Fan: " + hvac.getFanSpeed());
                    deviceInfoPanel.add(tempLabel);
                    deviceInfoPanel.add(fanLabel);

                    JButton setTempButton = new JButton(text:"Set Temp");
                    setTempButton.addActionListener(e -> {
                        String tempStr = JOptionPane.showInputDialog(SmartHomeGUI.this, message:"Enter target temperature (°C):", hvac.getTargetTemperature());
                        if (tempStr != null) {
                            try {
                                double newTemp = Double.parseDouble(tempStr);
                                hvac.setTargetTemperature(newTemp);
                                updateDeviceDetails(deviceInfoPanel, device);
                            } catch (NumberFormatException ex) {
                                JOptionPane.showMessageDialog(SmartHomeGUI.this, message:"Invalid input. Please enter a number.");
                            }
                        }
                    });
                    deviceInfoPanel.add(setTempButton);

                    JButton setFanButton = new JButton(text:"Set Fan");
                    setFanButton.addActionListener(e -> {
                        String fanStr = JOptionPane.showInputDialog(SmartHomeGUI.this, message:"Enter fan speed (0-3):", hvac.getFanSpeed());
```

o SmartHomeGUI.java



```java
public class SmartHomeGUI extends JFrame {
    private void displayDevices() {
                    deviceInfoPanel.add(setTempButton);

                    JButton setFanButton = new JButton(text:"Set Fan");
                    setFanButton.addActionListener(e -> {
                        String fanStr = JOptionPane.showInputDialog(SmartHomeGUI.this, message:"Enter fan speed (0-3):", hvac.getFanSpeed());
                        if (fanStr != null) {
                            try {
                                int newFan = Integer.parseInt(fanStr);
                                if (newFan >= 0 && newFan <= 3) {
                                    hvac.setFanSpeed(newFan);
                                    updateDeviceDetails(deviceInfoPanel, device);
                                } else {
                                    JOptionPane.showMessageDialog(SmartHomeGUI.this, message:"Invalid fan speed (0-3).");
                                }
                            } catch (NumberFormatException ex) {
                                JOptionPane.showMessageDialog(SmartHomeGUI.this, message:"Invalid input. Please enter a number.");
                            }
                        }
                    });
                    deviceInfoPanel.add(setFanButton);
                }

                devicesPanel.add(deviceInfoPanel);
            } // Closing brace for the for loop
        } // Closing brace for the if (devices != null) block
    } // Closing brace for displayDevices()

    // Helper method to update the displayed details of a device
    private void updateDeviceDetails(JPanel panel, EnergyConsumer device) {
        for (Component comp : panel.getComponents()) {
            if (comp instanceof JLabel) {
                JLabel label = (JLabel) comp;
                if (label.getText().startsWith(prefix:"Status:")) {
                    label.setText("Status: " + (device.isStatus() ? "ON" : "OFF"));
                } else if (device instanceof SmartLight && label.getText().startsWith(prefix:"Brightness:")) {
                    label.setText("Brightness: " + ((SmartLight) device).getBrightness() + "%");
                } else if (device instanceof SmartHVAC && label.getText().startsWith(prefix:"Target Temp:")) {
                    label.setText("Target Temp: " + ((SmartHVAC) device).getTargetTemperature() + "°C");
                } else if (device instanceof SmartHVAC && label.getText().startsWith(prefix:"Fan:")) {
                    label.setText("Fan: " + ((SmartHVAC) device).getFanSpeed());
                }
            }
        }
        panel.revalidate();
        panel.repaint();
    } // Closing brace for updateDeviceDetails()

} // Closing brace for the SmartHomeGUI class
```

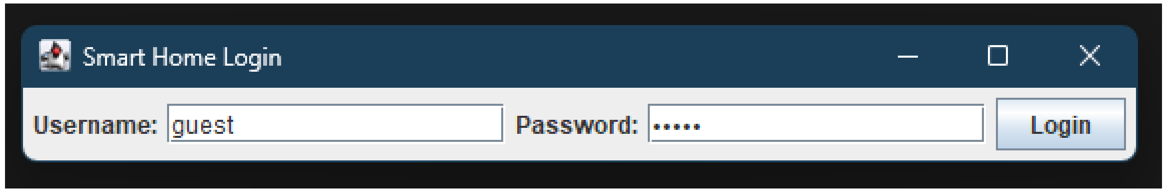
## ➢ Output:

The code now provides a robust foundation for simulating and interactively controlling a smart home. It includes essential classes to represent devices, calculate energy consumption, and model basic environmental interactions. The significant enhancement is the addition of a **user-friendly graphical interface**, which allows for direct manipulation of device states and settings, making the simulation much more accessible and engaging. This GUI integration has shifted the application from a purely programmatic simulation to an interactive control panel.

- **Screenshots:**
  The Smart Home Simulator has been upgraded with a **Graphical User Interface (GUI)**, providing an interactive and user-friendly experience for controlling smart devices and visualizing their status. While the backend still produces **terminal output** for detailed simulation data, the primary interaction is now through the GUI.

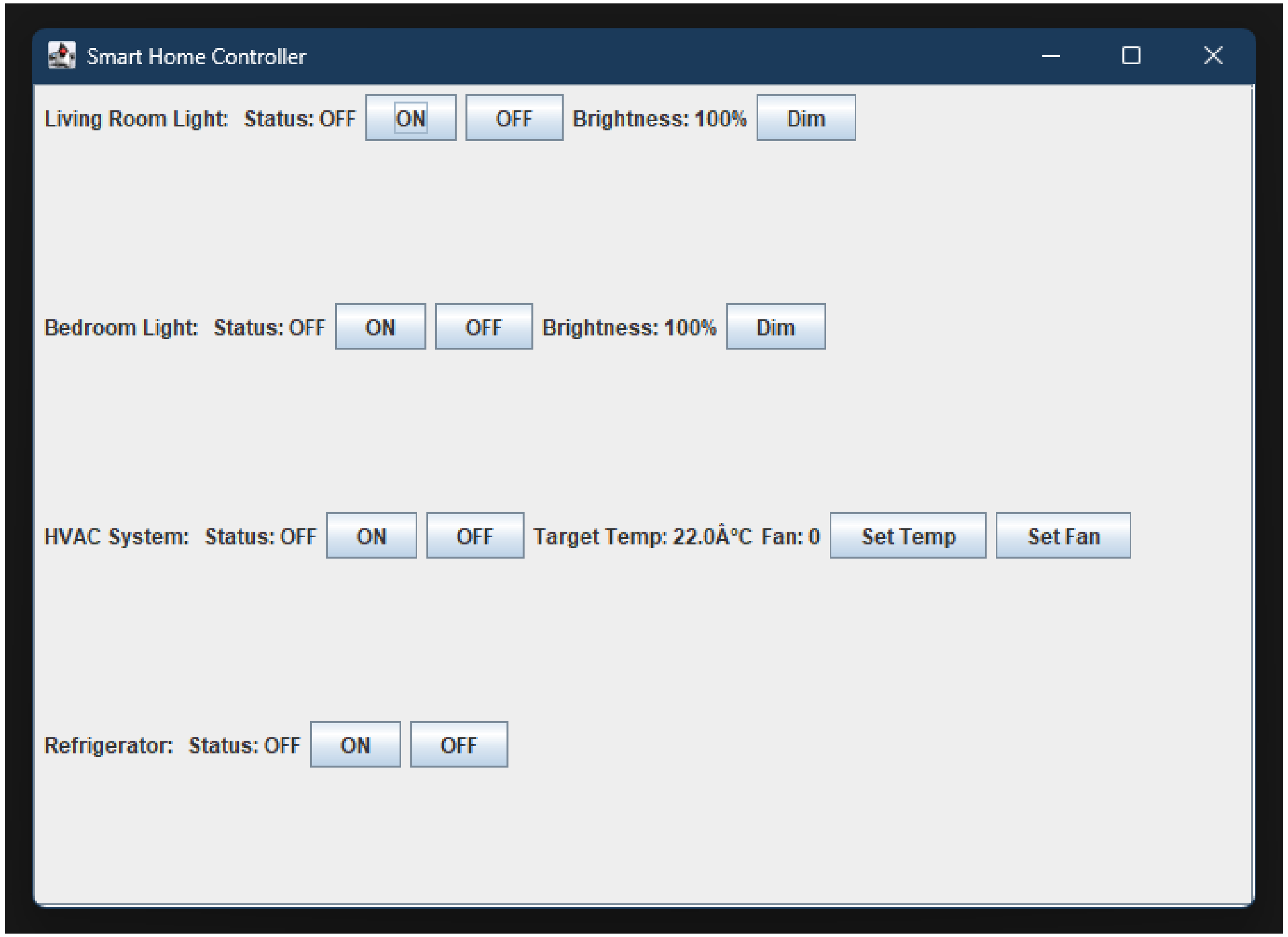
- **GUI Screenshots**

## Login Screen



➢ The initial login screen of the Smart Home Controller.

## Main Controller



Living Room Light: Status: OFF Brightness: 100%

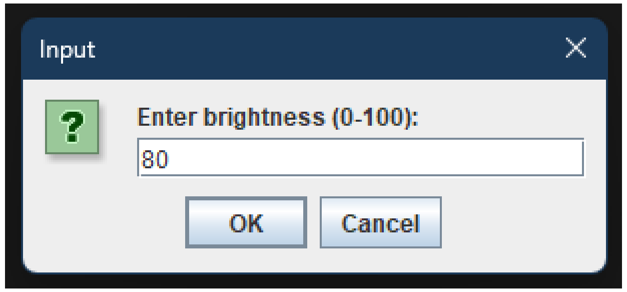
Bedroom Light: Status: OFF Brightness: 100%

HVAC System: Status: OFF Target Temp: 22.0Â°C Fan: 0

Refrigerator: Status: OFF

➢ The main interface for controlling smart devices. Here, I can see the list of added devices with their current status and basic controls.

**Setting Brightness (80%)**

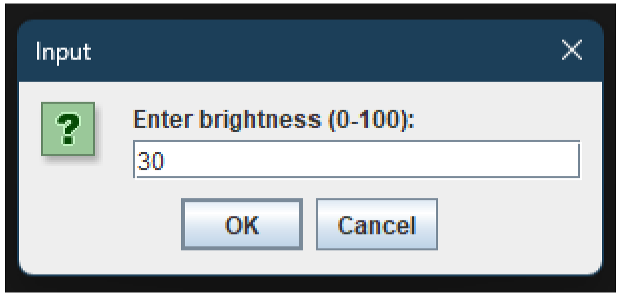

Input

Enter brightness (0-100):

80

- ➢ An input dialog box allows users to precisely set the brightness level for smart lights. In this example, the brightness is being set to 80%.

**Setting Brightness (30%)**

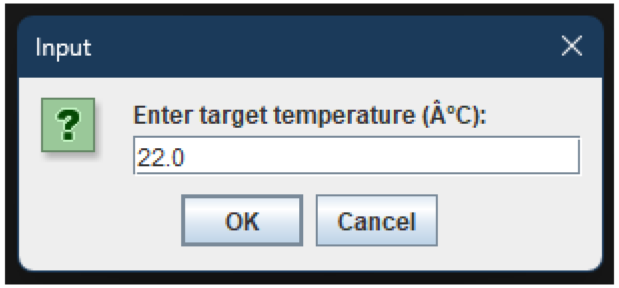

Input

Enter brightness (0-100):

30

- ➢ Another instance of the brightness input dialog, this time showing the light being dimmed to 30%.

## Setting Target Temperature

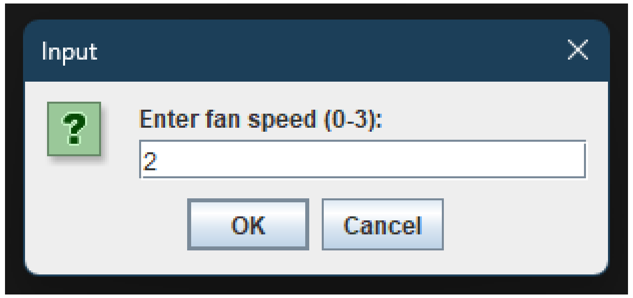

Input

Enter target temperature (Â°C):

22.0

- ➢ This dialog enables users to adjust the desired temperature for the HVAC system, demonstrating fine-grained control over environmental conditions.

## Setting Fan Speed

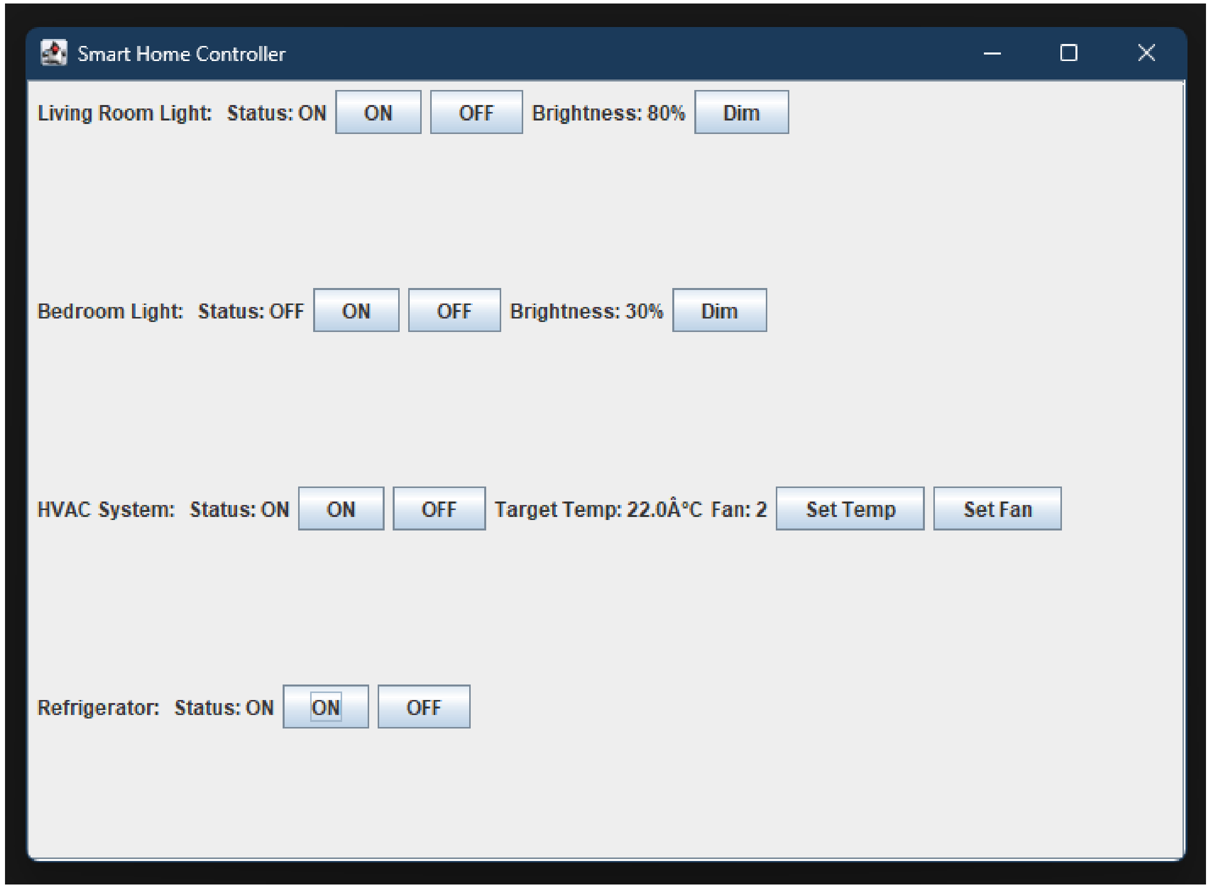

Input

Enter fan speed (0-3):

2

- ➢ Users can control the fan speed of the HVAC system through this input dialog, ranging from 0 (off) to 3 (high).

## Main Controller with Updated Status


Smart Home Controller

Living Room Light: Status: ON ON OFF Brightness: 80% Dim

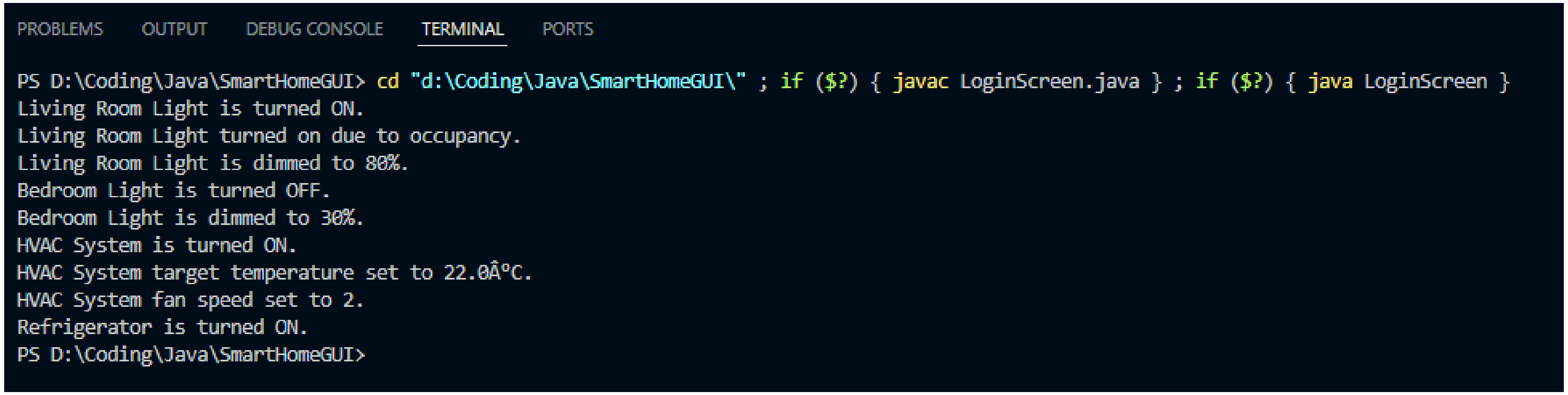
Bedroom Light: Status: OFF ON OFF Brightness: 30% Dim

HVAC System: Status: ON ON OFF Target Temp: 22.0Â°C Fan: 2 Set Temp Set Fan

Refrigerator: Status: ON ON OFF

➢ This screenshot shows the main controller after devices have been interacted with. Notice how the status labels for the devices are updated in real-time to reflect their current state (e.g., ON/OFF, brightness, target temperature, fan speed).

## Terminal Output



```
PS D:\Coding\Java\SmartHomeGUI> cd "d:\Coding\Java\SmartHomeGUI\" ; if ($?) { javac LoginScreen.java } ; if ($?) { java LoginScreen }
Living Room Light is turned ON.
Living Room Light turned on due to occupancy.
Living Room Light is dimmed to 80%.
Bedroom Light is turned OFF.
Bedroom Light is dimmed to 30%.
HVAC System is turned ON.
HVAC System target temperature set to 22.0Â°C.
HVAC System fan speed set to 2.
Refrigerator is turned ON.
PS D:\Coding\Java\SmartHomeGUI>
```

➢ Although the primary interaction is now via the GUI, the backend simulation continues to generate detailed text-based output in the console. This output shows the real-time changes in device status, environmental conditions (like sunlight intensity and wind speed), individual device energy consumption, total energy consumption, generated renewable energy, and the calculated cost for each hour of the simulation. This provides a comprehensive log for analysis.

## ➢ Technology Stack:

The Smart Home Simulator is developed using the following technologies:

**Programming Language:** Java

**GUI Toolkit:** Swing

**IDE:** Visual Studio Code (VS Code)

**Build Tool:** None (manual compilation, facilitated by VS Code Java Extensions)

**Charting Library:** (Future) JFreeChart or similar, to be integrated for data visualization.

## ➢ System Architecture:

The system architecture is based on a **modular, object-oriented design**, allowing for clear separation of concerns and maintainability.

**Core Layer:**

- **Simulation Engine:** This component manages the central simulation loop, time progression, and event handling within the smart home environment.
- **Device Model:** Defines the behavior, attributes, and energy consumption logic for various smart home devices (e.g., lights, HVAC, refrigerators).
- **Environment Model:** Models external environmental factors such as sunlight intensity and wind speed, and their impact on renewable energy generation and device behavior.
- **Resource Management:** Tracks and calculates the usage of resources, primarily energy consumption and generation, to determine costs and environmental impact.

## UI Layer:

- **GUI Components:** This is the primary interface for user interaction. It handles the graphical presentation of devices, controls for device settings, and real-time updates of device status.
- **CLI Components:** While the main interaction is now graphical, command-line interface elements can still be present for backend logging and detailed textual output of simulation events and results.
- **Visualization Components:** (Future) Responsible for generating charts, graphs, and detailed reports to visually represent simulation data for better analysis.

## Data Layer:

- **Data Persistence:** (Future) Handles the loading and saving of simulation configurations (e.g., device setups, initial states) and results, enabling users to resume simulations or review historical data.

## ➢ Object-Oriented Principles:

OOP principles will be used to design and implement the simulator:

- **Abstraction:** Define abstract classes and interfaces to represent common features of devices and systems. For example, an abstract EnergyConsumer class.
- **Encapsulation:** Protect the internal state of objects by using private attributes and public methods.
- **Inheritance:** Create a hierarchy of classes to represent specialized types of devices. For example, LEDLight and CFLight inheriting from SmartLight.
- **Polymorphism:** Use interfaces and abstract classes to allow objects of different classes to be treated in a uniform way. For example, a list of EnergyConsumer objects can be processed to calculate total energy consumption.

## 3. Remaining features to implement:

Based on the project proposal and the current code, the following features remain to be implemented:

**More Complex Device Behaviours:**

- The behavior of some devices can be made more complex. For example, the HVAC system could model the effect of insulation and outside temperature, leading to more realistic energy consumption based on these factors.
- The SmartRefrigerator could incorporate door open/close events or food spoilage alerts.
- SmartLights could have automated schedules or adaptive lighting based on ambient light sensors.

**Advanced Environmental Modelling:**

- More sophisticated modeling of environmental factors, such as varying sunlight intensity throughout different seasons, and more realistic wind patterns that consider geographical data.
- Integration of real-time weather data or historical weather patterns to drive the simulation.

**Data Persistence:**

- The ability to save and load simulation configurations (e.g., device setups, initial states) and results, allowing users to resume previous simulations or analyze historical data. This could involve file I/O or a simple database.

**Reporting and Analysis:**

- Generate more detailed and customizable reports on energy consumption, generation, and cost.

- Provide advanced analysis of the simulation data, potentially including visualizations (charts, graphs) of energy trends, cost breakdowns, and environmental impact.

**Error Handling:**

- Implement robust error handling to make the application more resilient to invalid user inputs (beyond basic number parsing) and unexpected system states, providing meaningful feedback to the user.
- Add logging mechanisms for debugging and tracking application behavior.

## 4. Future Work:

- **Integration with real-world data:** Incorporate real-time weather data and energy prices.
- **Advanced simulation features:** Model more complex phenomena such as heat transfer and airflow.
- **Machine learning:** Implement AI algorithms for energy optimization and predictive control.
- **Home automation integration:** Explore the possibility of connecting the simulator to real-world smart home systems.
- **Online platform:** Develop a web-based version of the simulator with collaborative features.
- **Mobile app:** Create a mobile app for users to monitor and control their simulated smart home remotely.

## 5. Conclusion:

### ➢ Final Product and its Utility:

The **current product is an interactive Eco-Friendly Smart Home Simulator with a Graphical User Interface (GUI)**. This application allows users to interact with various smart home devices and renewable energy sources in a simulated environment. The current version provides real-time control over devices and calculates their energy consumption and generation.

Upon full completion, the final product will evolve into a comprehensive tool that allows users to:

- **Design virtual smart homes** with a variety of energy-consuming devices and renewable energy solutions.
- **Experiment with different eco-friendly technologies** to understand their impact on energy usage.
- **Analyze the energy consumption, cost, and environmental impact** of their

choices through detailed reports and potentially visualizations.

This simulator will be highly useful to users by:

- **Educating them about energy-saving strategies and eco-friendly technologies** in a hands-on, interactive manner.
- **Enabling them to make informed decisions** about their home design, device selection, and energy usage patterns.
- **Providing a practical platform to test and compare different energy management scenarios** before implementing them in a real home, thereby promoting sustainable living and potentially reducing real-world energy costs.

## ➢ Project Summary and Impact:

This project involves the development of an **interactive Eco-Friendly Smart Home Simulator** using Java and Object-Oriented Programming (OOP) principles. The simulator, now featuring a **Graphical User Interface (GUI)**, empowers users to create, control, and analyze virtual smart homes. By providing a hands-on platform to experiment with various energy-consuming devices and renewable energy sources, the project aims to promote sustainable living and energy efficiency. The simulator will have a significant impact by raising awareness about eco-friendly practices and enabling users to make informed decisions to reduce their environmental footprint and energy costs.

## 6. References and Resources:

- **"Head First Java" by Kathy Sierra and Bert Bates**
  - **Publisher:** O'Reilly Media
  - A beginner-friendly book that explains Java and OOP concepts (like encapsulation, inheritance, and polymorphism) with practical examples.
  - Relevant for understanding the class structure and design in the simulator.

- **"Effective Java" by Joshua Bloch**
  - **Publisher:** Addison-Wesley
  - A more advanced book on Java best practices, including how to write reusable and maintainable code (e.g., using the Singleton pattern as in SmartHomeController).

    - Third Edition (2018) is the latest as of my knowledge base.

- **Oracle Java Documentation**
    - **URL:** https://docs.oracle.com/en/java/
    - Official documentation for Java SE, including the API for classes like ArrayList and List used in the code.
    - Great for understanding built-in Java features and syntax.

- **Energy Consumption Data (General Reference)**
    - **Source:** U.S. Energy Information Administration (EIA)
    - URL: https://www.eia.gov/energyexplained/use-of-energy/
    - Provides real-world data on energy usage (e.g., lighting, HVAC), which I approximated in the simulator (e.g., 60W for lights, 1.5 kWh for thermostats).

- **"Carbon Footprint of Electricity Generation" (Simplified Metrics)**
    - **Source:** International Energy Agency (IEA)
    - URL: https://www.iea.org/
    - The simulator uses a simplified 0.5 kg CO2 per kWh for carbon footprint calculations, inspired by average global estimates from such sources.
- **"The Internet of Things" by Samuel Greengard**
    - **Publisher:** MIT Press
    - Explores IoT concepts, including smart devices and energy management, which relate to the simulator’s functionality.
    - Published in 2015, but still relevant for foundational ideas.

- **"Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma et al.**
    - **Publisher:** Addison-Wesley
    - Known as the "Gang of Four" book, it covers patterns like Singleton (used in SmartHomeController) and Factory (suggested as an enhancement).
    - Published in 1994, but timeless for OOP design.

- **JavaPoint Tutorials**
    - **URL:** [https://www.javatpoint.com/java-oops-concepts](https://www.javatpoint.com/java-oops-concepts)
    - A free resource with examples of OOP in Java, including inheritance and abstraction, which are central to the SmartDevice hierarchy.

- **GeeksforGeeks: Java Examples**
  - **URL:** https://www.geeksforgeeks.org/java/
  - Provides practical code snippets for Java concepts like abstract classes and collections, which I used in the simulator.Source Information

- **W3Schools Java Tutorial**
  - **URL:** https://www.w3schools.com/java/
  - Content: Free tutorials on Java programming, including syntax, OOP concepts (classes, objects, inheritance, abstraction), and collections (e.g., ArrayList).

- **Google Gemini**
  - https://gemini.google.com/share/968f7aae37d2

- **X Grok**
  - https://x.com/i/grok/share/i1pbBUaa8kz05cPZu6EOtLWgi

- **Graphviz Online**
  - https://dreampuf.github.io/GraphvizOnline/

## 7. Appendix:

- **GitHub Repository:**

The complete source code for the Eco-Friendly Smart Home Simulator project is publicly available on GitHub. Anyone can access the repository, contribute, or provide feedback via the link below:

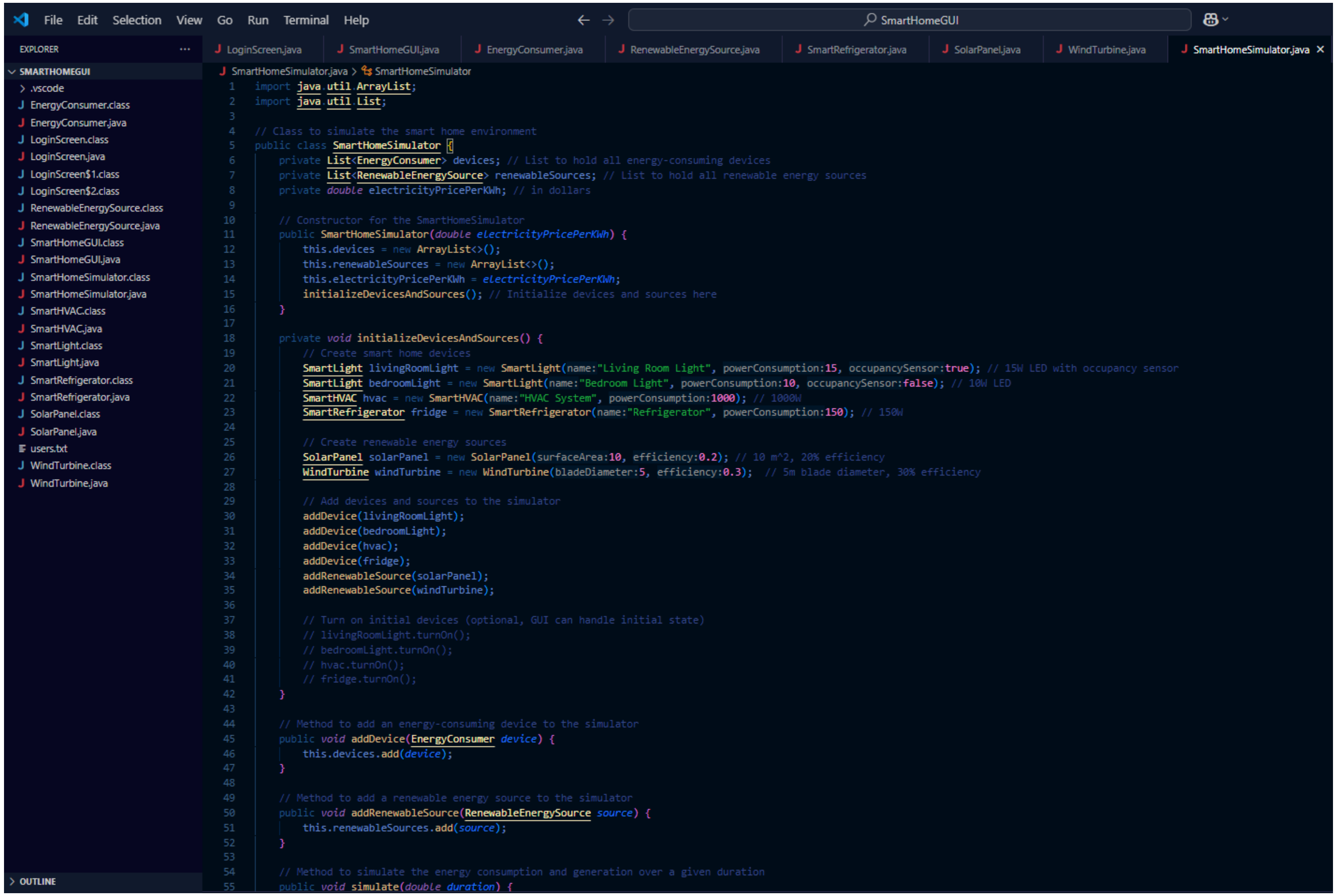
GitHub Repository Link:

https://github.com/iamdurjoybarua/Eco-Friendly-Smart-Home-Simulator

## A. Full Source Code Listing:

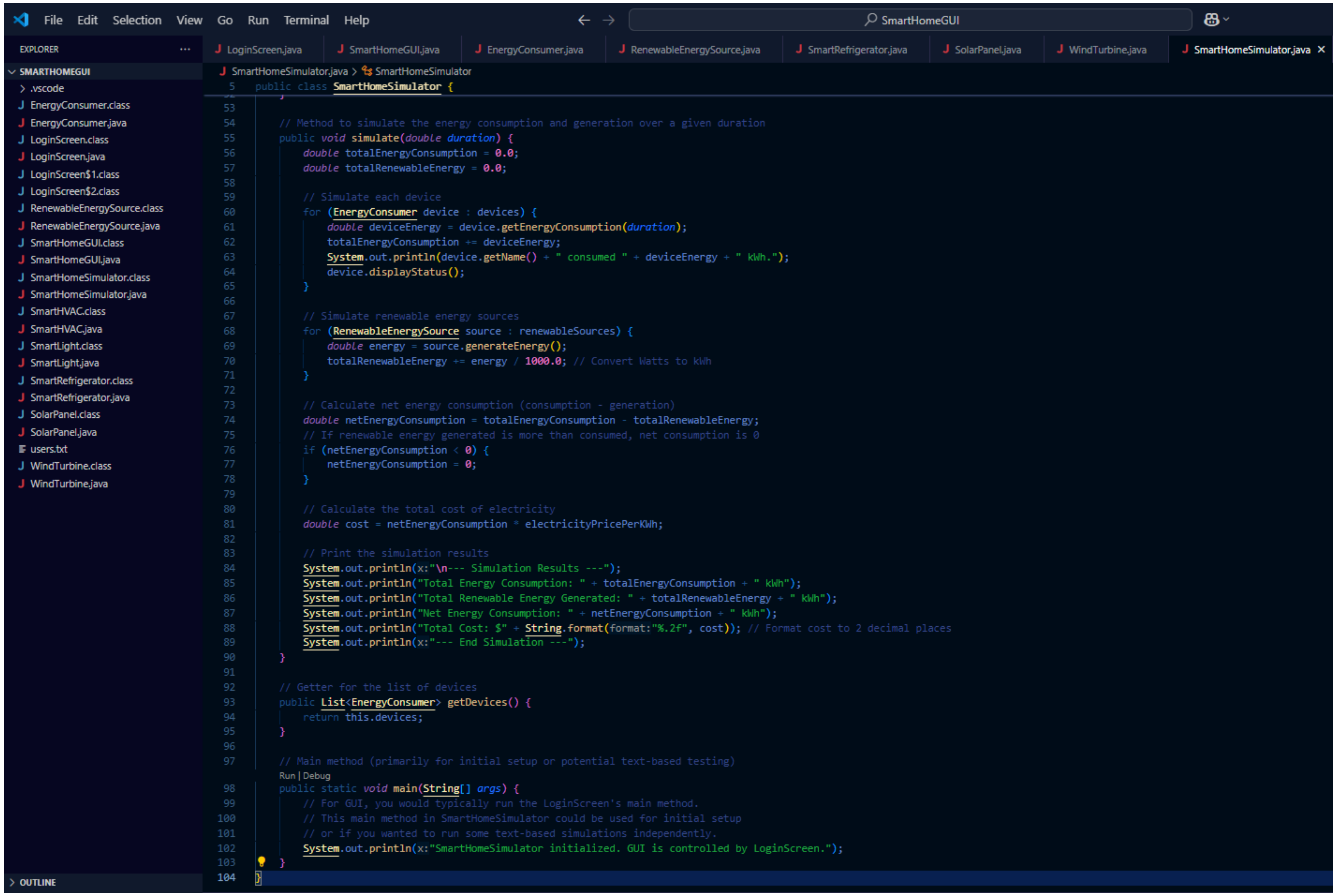
- This section would contain the complete source code for all Java classes developed for the Smart Home Simulator, including:

- SmartHomeSimulator.java



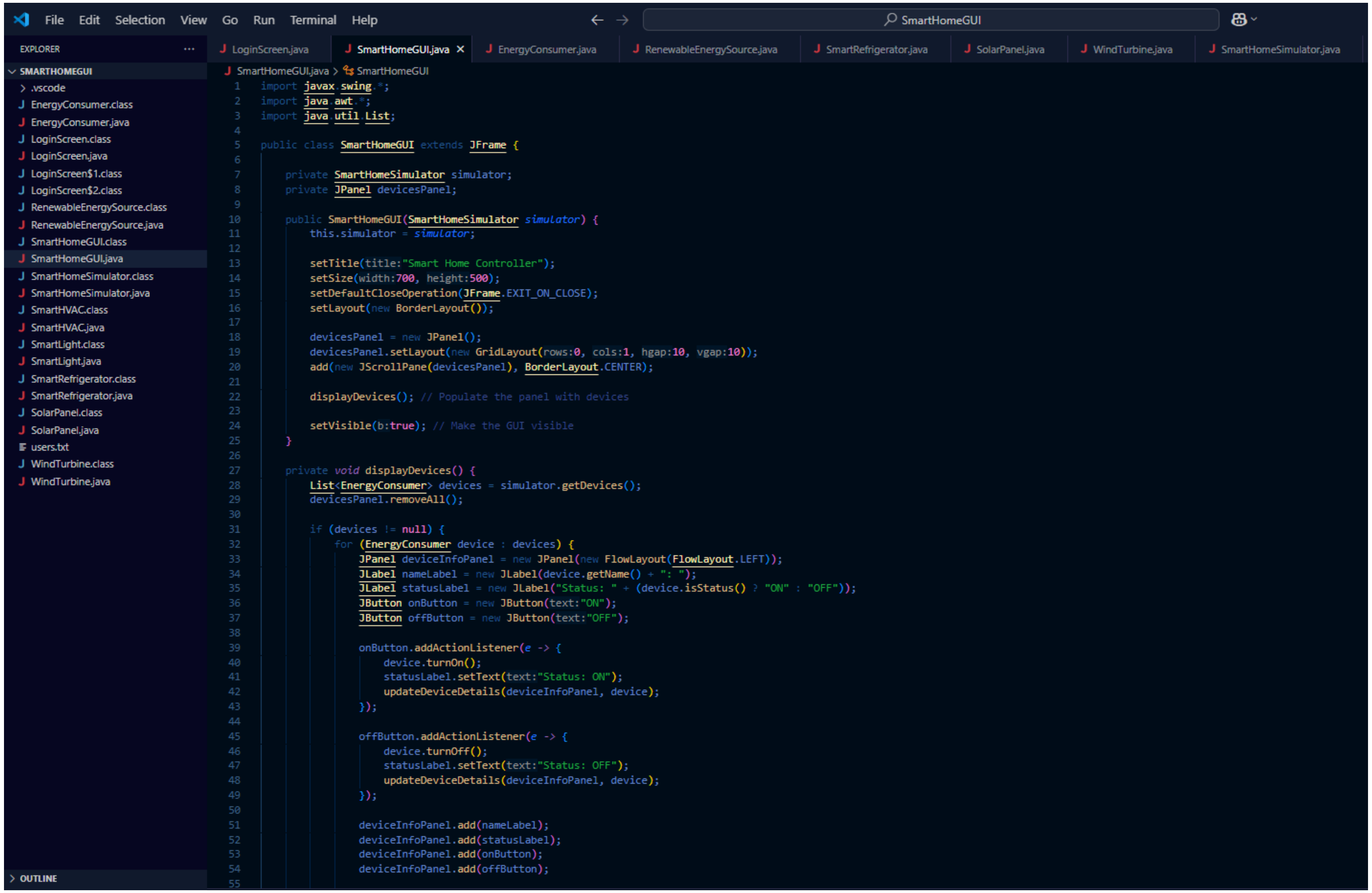
```java
import java.util.ArrayList;
import java.util.List;

// Class to simulate the smart home environment
public class SmartHomeSimulator {
    private List<EnergyConsumer> devices; // List to hold all energy-consuming devices
    private List<RenewableEnergySource> renewableSources; // List to hold all renewable energy sources
    private double electricityPricePerKWh; // in dollars

    // Constructor for the SmartHomeSimulator
    public SmartHomeSimulator(double electricityPricePerKWh) {
        this.devices = new ArrayList<>();
        this.renewableSources = new ArrayList<>();
        this.electricityPricePerKWh = electricityPricePerKWh;
        initializeDevicesAndSources(); // Initialize devices and sources here
    }

    private void initializeDevicesAndSources() {
        // Create smart home devices
        SmartLight livingRoomLight = new SmartLight(name:"Living Room Light", powerConsumption:15, occupancySensor:true); // 15W LED with occupancy sensor
        SmartLight bedroomLight = new SmartLight(name:"Bedroom Light", powerConsumption:10, occupancySensor:false); // 10W LED
        SmartHVAC hvac = new SmartHVAC(name:"HVAC System", powerConsumption:1000); // 1000W
        SmartRefrigerator fridge = new SmartRefrigerator(name:"Refrigerator", powerConsumption:150); // 150W

        // Create renewable energy sources
        SolarPanel solarPanel = new SolarPanel(surfaceArea:10, efficiency:0.2); // 10 m^2, 20% efficiency
        WindTurbine windTurbine = new WindTurbine(bladeDiameter:5, efficiency:0.3);  // 5m blade diameter, 30% efficiency

        // Add devices and sources to the simulator
        addDevice(livingRoomLight);
        addDevice(bedroomLight);
        addDevice(hvac);
        addDevice(fridge);
        addRenewableSource(solarPanel);
        addRenewableSource(windTurbine);

        // Turn on initial devices (optional, GUI can handle initial state)
        // livingRoomLight.turnOn();
        // bedroomLight.turnOn();
        // hvac.turnOn();
        // fridge.turnOn();
    }

    // Method to add an energy-consuming device to the simulator
    public void addDevice(EnergyConsumer device) {
        this.devices.add(device);
    }

    // Method to add a renewable energy source to the simulator
    public void addRenewableSource(RenewableEnergySource source) {
        this.renewableSources.add(source);
    }

    // Method to simulate the energy consumption and generation over a given duration
    public void simulate(double duration) {
```

- SmartHomeSimulator.java



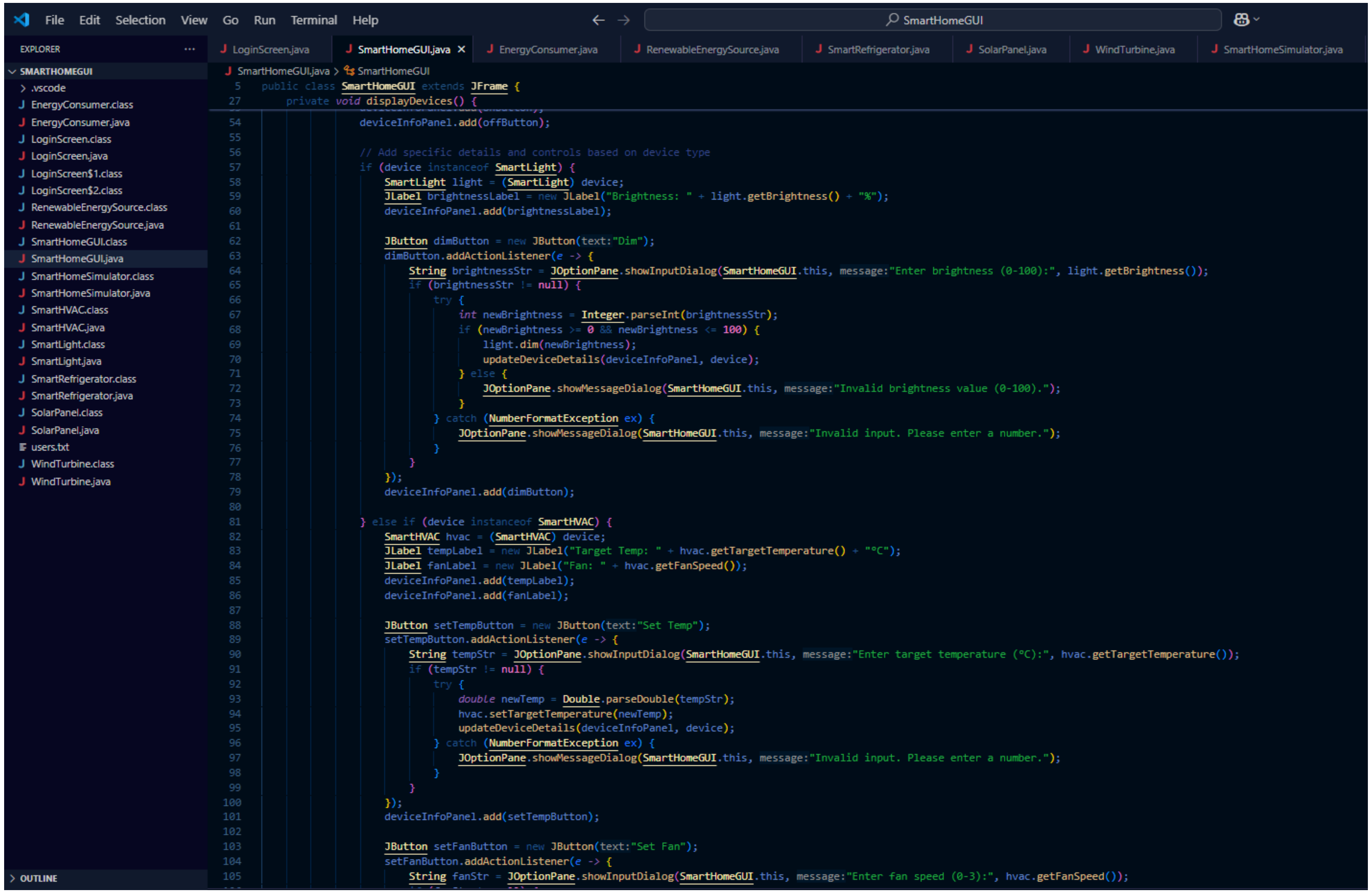
```java
public class SmartHomeSimulator {

    // Method to simulate the energy consumption and generation over a given duration
    public void simulate(double duration) {
        double totalEnergyConsumption = 0.0;
        double totalRenewableEnergy = 0.0;

        // Simulate each device
        for (EnergyConsumer device : devices) {
            double deviceEnergy = device.getEnergyConsumption(duration);
            totalEnergyConsumption += deviceEnergy;
            System.out.println(device.getName() + " consumed " + deviceEnergy + " kWh.");
            device.displayStatus();
        }

        // Simulate renewable energy sources
        for (RenewableEnergySource source : renewableSources) {
            double energy = source.generateEnergy();
            totalRenewableEnergy += energy / 1000.0; // Convert Watts to kWh
        }

        // Calculate net energy consumption (consumption - generation)
        double netEnergyConsumption = totalEnergyConsumption - totalRenewableEnergy;
        // If renewable energy generated is more than consumed, net consumption is 0
        if (netEnergyConsumption < 0) {
            netEnergyConsumption = 0;
        }

        // Calculate the total cost of electricity
        double cost = netEnergyConsumption * electricityPricePerKWh;

        // Print the simulation results
        System.out.println(x:"\n--- Simulation Results ---");
        System.out.println("Total Energy Consumption: " + totalEnergyConsumption + " kWh");
        System.out.println("Total Renewable Energy Generated: " + totalRenewableEnergy + " kWh");
        System.out.println("Net Energy Consumption: " + netEnergyConsumption + " kWh");
        System.out.println("Total Cost: $" + String.format(format:"%.2f", cost)); // Format cost to 2 decimal places
        System.out.println(x:"--- End Simulation ---");
    }

    // Getter for the list of devices
    public List<EnergyConsumer> getDevices() {
        return this.devices;
    }

    // Main method (primarily for initial setup or potential text-based testing)
    Run | Debug
    public static void main(String[] args) {
        // For GUI, you would typically run the LoginScreen's main method.
        // This main method in SmartHomeSimulator could be used for initial setup
        // or if you wanted to run some text-based simulations independently.
        System.out.println(x:"SmartHomeSimulator initialized. GUI is controlled by LoginScreen.");
    }
}
```

- SmartHomeGUI.java



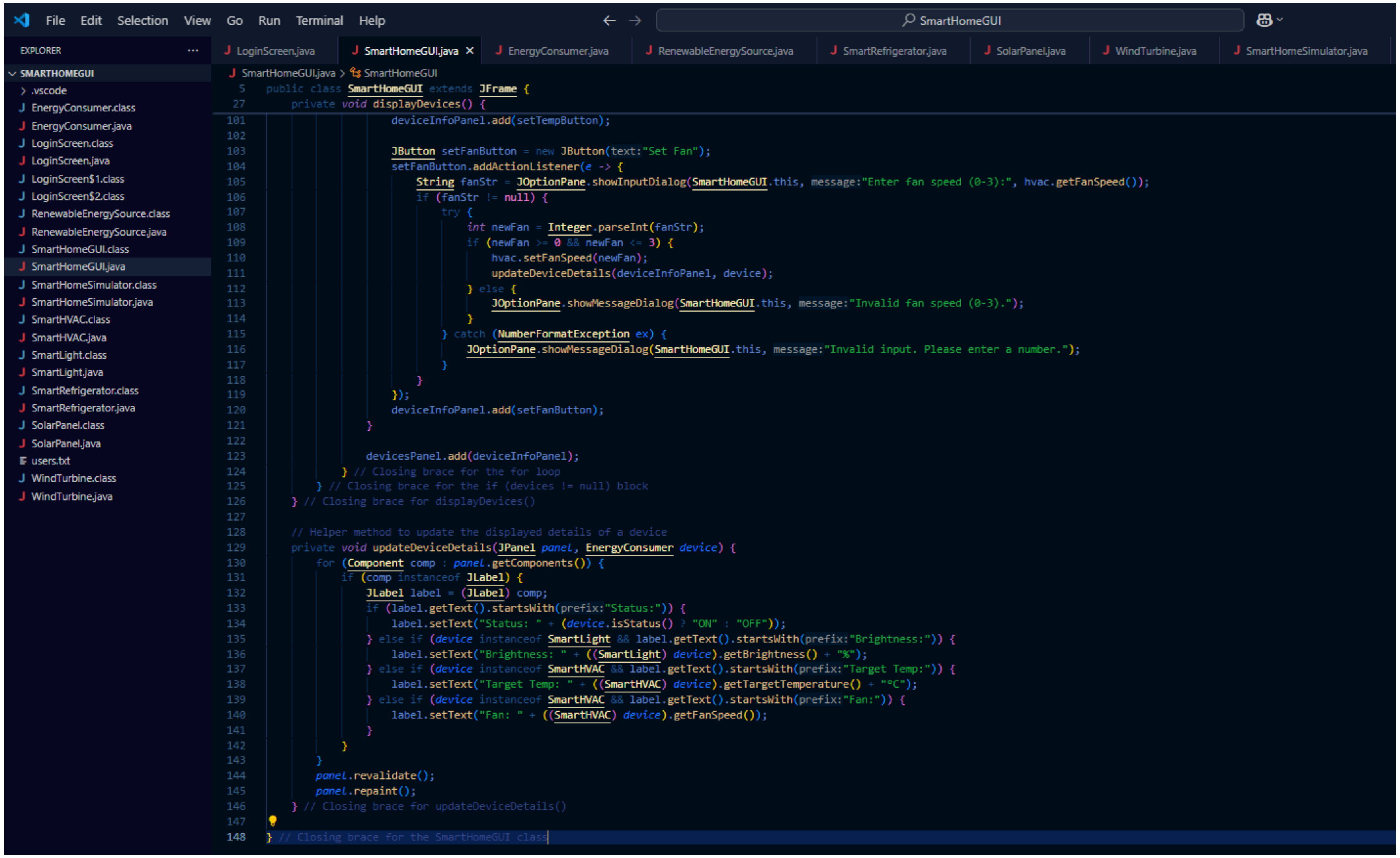
```java
import javax.swing.*;
import java.awt.*;
import java.util.List;

public class SmartHomeGUI extends JFrame {

    private SmartHomeSimulator simulator;
    private JPanel devicesPanel;

    public SmartHomeGUI(SmartHomeSimulator simulator) {
        this.simulator = simulator;

        setTitle(title:"Smart Home Controller");
        setSize(width:700, height:500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        devicesPanel = new JPanel();
        devicesPanel.setLayout(new GridLayout(rows:0, cols:1, hgap:10, vgap:10));
        add(new JScrollPane(devicesPanel), BorderLayout.CENTER);

        displayDevices(); // Populate the panel with devices

        setVisible(b:true); // Make the GUI visible
    }

    private void displayDevices() {
        List<EnergyConsumer> devices = simulator.getDevices();
        devicesPanel.removeAll();

        if (devices != null) {
            for (EnergyConsumer device : devices) {
                JPanel deviceInfoPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
                JLabel nameLabel = new JLabel(device.getName() + ": ");
                JLabel statusLabel = new JLabel("Status: " + (device.isStatus() ? "ON" : "OFF"));
                JButton onButton = new JButton(text:"ON");
                JButton offButton = new JButton(text:"OFF");

                onButton.addActionListener(e -> {
                    device.turnOn();
                    statusLabel.setText(text:"Status: ON");
                    updateDeviceDetails(deviceInfoPanel, device);
                });

                offButton.addActionListener(e -> {
                    device.turnOff();
                    statusLabel.setText(text:"Status: OFF");
                    updateDeviceDetails(deviceInfoPanel, device);
                });

                deviceInfoPanel.add(nameLabel);
                deviceInfoPanel.add(statusLabel);
                deviceInfoPanel.add(onButton);
                deviceInfoPanel.add(offButton);

```

- SmartHomeGUI.java



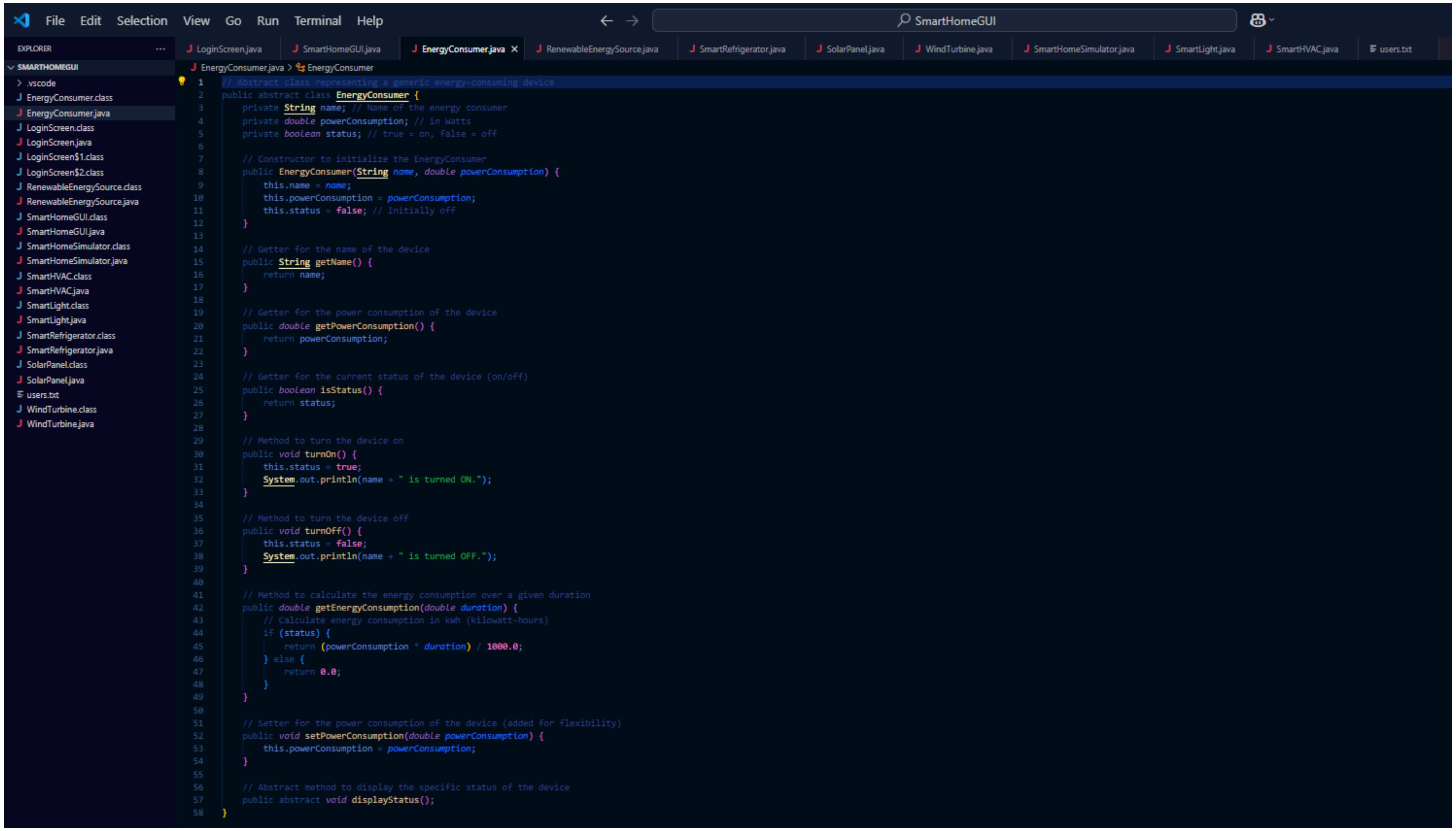
```java
public class SmartHomeGUI extends JFrame {
    private void displayDevices() {
                deviceInfoPanel.add(offButton);

                // Add specific details and controls based on device type
                if (device instanceof SmartLight) {
                    SmartLight light = (SmartLight) device;
                    JLabel brightnessLabel = new JLabel("Brightness: " + light.getBrightness() + "%");
                    deviceInfoPanel.add(brightnessLabel);

                    JButton dimButton = new JButton(text:"Dim");
                    dimButton.addActionListener(e -> {
                        String brightnessStr = JOptionPane.showInputDialog(SmartHomeGUI.this, message:"Enter brightness (0-100):", light.getBrightness());
                        if (brightnessStr != null) {
                            try {
                                int newBrightness = Integer.parseInt(brightnessStr);
                                if (newBrightness >= 0 && newBrightness <= 100) {
                                    light.dim(newBrightness);
                                    updateDeviceDetails(deviceInfoPanel, device);
                                } else {
                                    JOptionPane.showMessageDialog(SmartHomeGUI.this, message:"Invalid brightness value (0-100).");
                                }
                            } catch (NumberFormatException ex) {
                                JOptionPane.showMessageDialog(SmartHomeGUI.this, message:"Invalid input. Please enter a number.");
                            }
                        }
                    });
                    deviceInfoPanel.add(dimButton);

                } else if (device instanceof SmartHVAC) {
                    SmartHVAC hvac = (SmartHVAC) device;
                    JLabel tempLabel = new JLabel("Target Temp: " + hvac.getTargetTemperature() + "°C");
                    JLabel fanLabel = new JLabel("Fan: " + hvac.getFanSpeed());
                    deviceInfoPanel.add(tempLabel);
                    deviceInfoPanel.add(fanLabel);

                    JButton setTempButton = new JButton(text:"Set Temp");
                    setTempButton.addActionListener(e -> {
                        String tempStr = JOptionPane.showInputDialog(SmartHomeGUI.this, message:"Enter target temperature (°C):", hvac.getTargetTemperature());
                        if (tempStr != null) {
                            try {
                                double newTemp = Double.parseDouble(tempStr);
                                hvac.setTargetTemperature(newTemp);
                                updateDeviceDetails(deviceInfoPanel, device);
                            } catch (NumberFormatException ex) {
                                JOptionPane.showMessageDialog(SmartHomeGUI.this, message:"Invalid input. Please enter a number.");
                            }
                        }
                    });
                    deviceInfoPanel.add(setTempButton);

                    JButton setFanButton = new JButton(text:"Set Fan");
                    setFanButton.addActionListener(e -> {
                        String fanStr = JOptionPane.showInputDialog(SmartHomeGUI.this, message:"Enter fan speed (0-3):", hvac.getFanSpeed());
```

- SmartHomeGUI.java



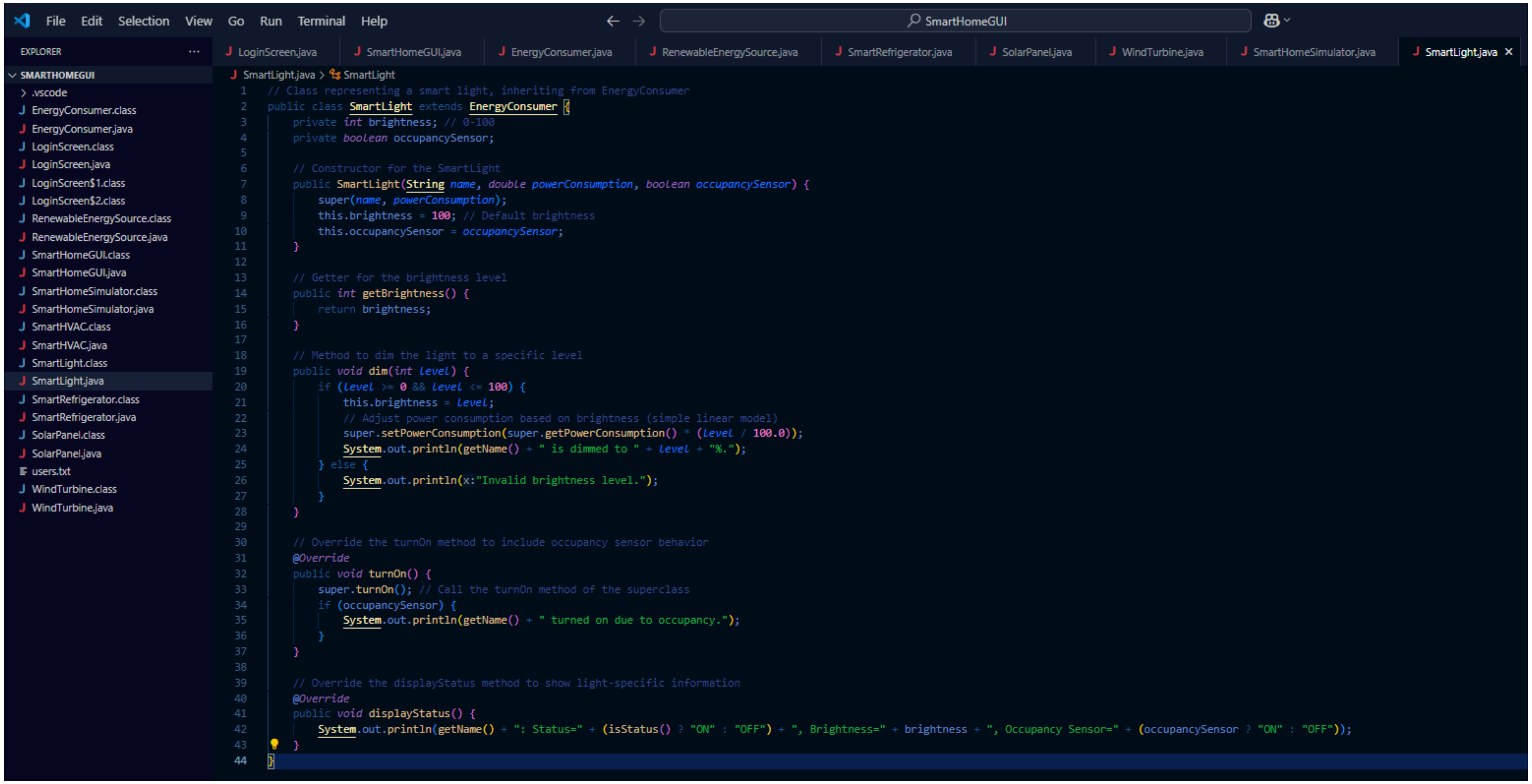
```java
public class SmartHomeGUI extends JFrame {
    private void displayDevices() {
                    deviceInfoPanel.add(setTempButton);

                    JButton setFanButton = new JButton(text:"Set Fan");
                    setFanButton.addActionListener(e -> {
                        String fanStr = JOptionPane.showInputDialog(SmartHomeGUI.this, message:"Enter fan speed (0-3):", hvac.getFanSpeed());
                        if (fanStr != null) {
                            try {
                                int newFan = Integer.parseInt(fanStr);
                                if (newFan >= 0 && newFan <= 3) {
                                    hvac.setFanSpeed(newFan);
                                    updateDeviceDetails(deviceInfoPanel, device);
                                } else {
                                    JOptionPane.showMessageDialog(SmartHomeGUI.this, message:"Invalid fan speed (0-3).");
                                }
                            } catch (NumberFormatException ex) {
                                JOptionPane.showMessageDialog(SmartHomeGUI.this, message:"Invalid input. Please enter a number.");
                            }
                        }
                    });
                    deviceInfoPanel.add(setFanButton);
                }

                devicesPanel.add(deviceInfoPanel);
            } // Closing brace for the for loop
        } // Closing brace for the if (devices != null) block
    } // Closing brace for displayDevices()

    // Helper method to update the displayed details of a device
    private void updateDeviceDetails(JPanel panel, EnergyConsumer device) {
        for (Component comp : panel.getComponents()) {
            if (comp instanceof JLabel) {
                JLabel label = (JLabel) comp;
                if (label.getText().startsWith(prefix:"Status:")) {
                    label.setText("Status: " + (device.isStatus() ? "ON" : "OFF"));
                } else if (device instanceof SmartLight && label.getText().startsWith(prefix:"Brightness:")) {
                    label.setText("Brightness: " + ((SmartLight) device).getBrightness() + "%");
                } else if (device instanceof SmartHVAC && label.getText().startsWith(prefix:"Target Temp:")) {
                    label.setText("Target Temp: " + ((SmartHVAC) device).getTargetTemperature() + "°C");
                } else if (device instanceof SmartHVAC && label.getText().startsWith(prefix:"Fan:")) {
                    label.setText("Fan: " + ((SmartHVAC) device).getFanSpeed());
                }
            }
        }
        panel.revalidate();
        panel.repaint();
    } // Closing brace for updateDeviceDetails()

} // Closing brace for the SmartHomeGUI class
```

- EnergyConsumer.java (and its concrete subclasses like SmartLight.java, SmartHVAC.java, SmartRefrigerator.java)



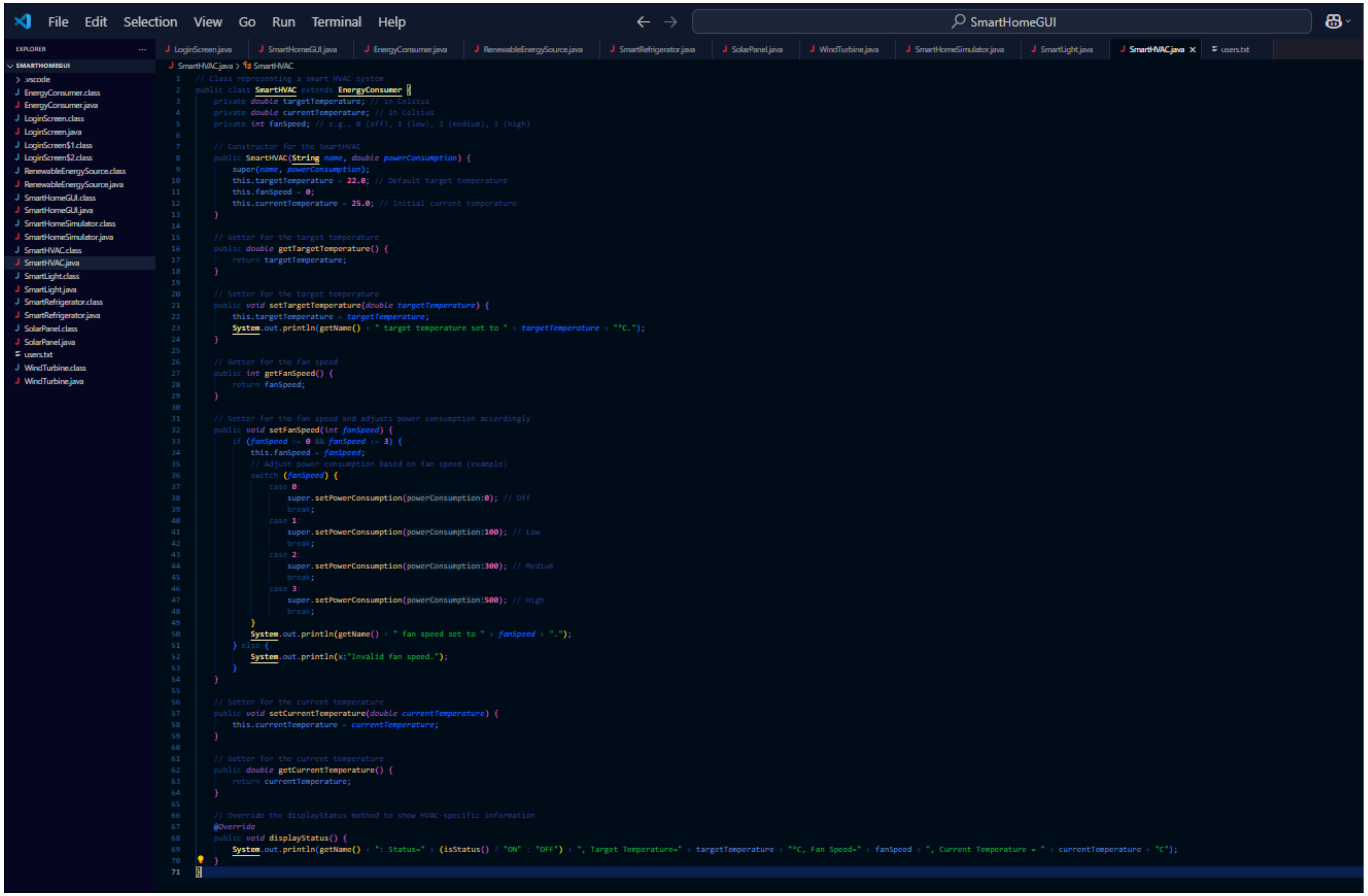
```java
// Abstract class representing a generic energy-consuming device
public abstract class EnergyConsumer {
    private String name; // Name of the energy consumer
    private double powerConsumption; // in Watts
    private boolean status; // true = on, false = off

    // Constructor to initialize the EnergyConsumer
    public EnergyConsumer(String name, double powerConsumption) {
        this.name = name;
        this.powerConsumption = powerConsumption;
        this.status = false; // Initially off
    }

    // Getter for the name of the device
    public String getName() {
        return name;
    }

    // Getter for the power consumption of the device
    public double getPowerConsumption() {
        return powerConsumption;
    }

    // Getter for the current status of the device (on/off)
    public boolean isStatus() {
        return status;
    }

    // Method to turn the device on
    public void turnOn() {
        this.status = true;
        System.out.println(name + " is turned ON.");
    }

    // Method to turn the device off
    public void turnOff() {
        this.status = false;
        System.out.println(name + " is turned OFF.");
    }

    // Method to calculate the energy consumption over a given duration
    public double getEnergyConsumption(double duration) {
        // Calculate energy consumption in kWh (kilowatt-hours)
        if (status) {
            return (powerConsumption * duration) / 1000.0;
        } else {
            return 0.0;
        }
    }

    // Setter for the power consumption of the device (added for flexibility)
    public void setPowerConsumption(double powerConsumption) {
        this.powerConsumption = powerConsumption;
    }

    // Abstract method to display the specific status of the device
    public abstract void displayStatus();
}
```

- SmartLight.java



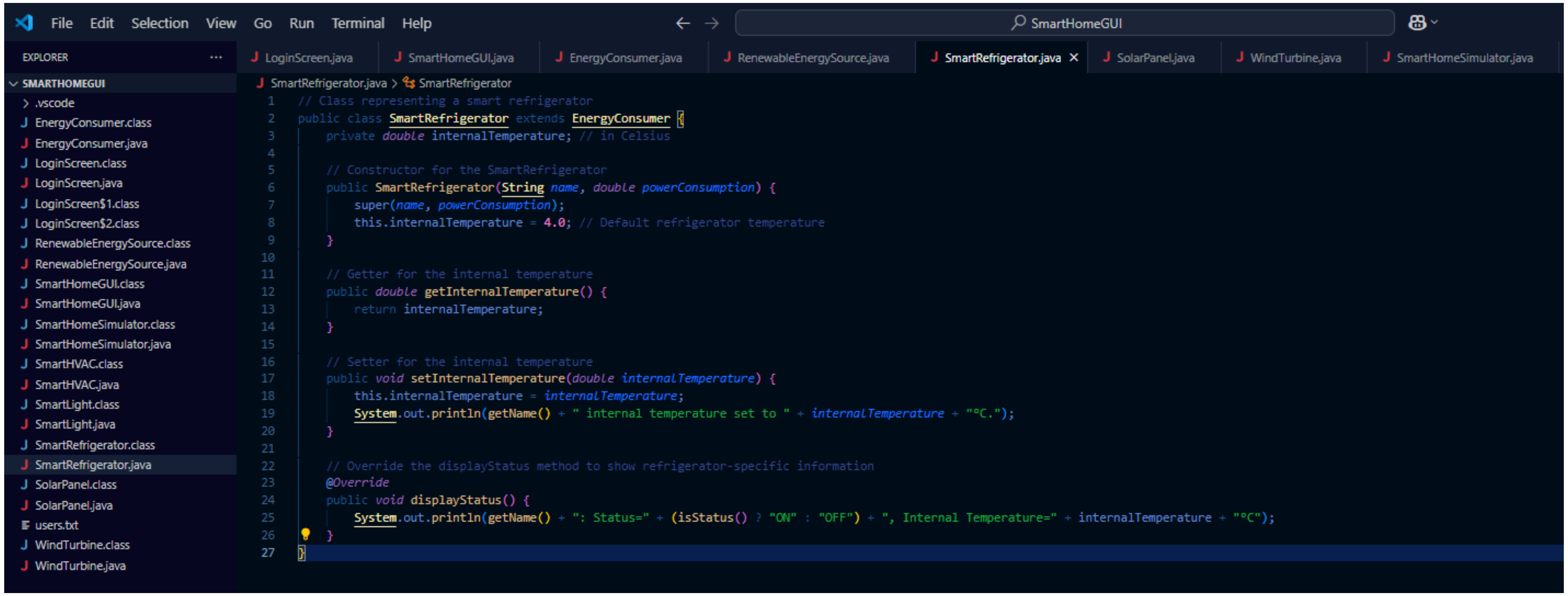
```java
// Class representing a smart light, inheriting from EnergyConsumer
public class SmartLight extends EnergyConsumer {
    private int brightness; // 0-100
    private boolean occupancySensor;

    // Constructor for the SmartLight
    public SmartLight(String name, double powerConsumption, boolean occupancySensor) {
        super(name, powerConsumption);
        this.brightness = 100; // Default brightness
        this.occupancySensor = occupancySensor;
    }

    // Getter for the brightness level
    public int getBrightness() {
        return brightness;
    }

    // Method to dim the light to a specific level
    public void dim(int level) {
        if (level >= 0 && level <= 100) {
            this.brightness = level;
            // Adjust power consumption based on brightness (simple linear model)
            super.setPowerConsumption(super.getPowerConsumption() * (level / 100.0));
            System.out.println(getName() + " is dimmed to " + level + "%.");
        } else {
            System.out.println(x:"Invalid brightness level.");
        }
    }

    // Override the turnOn method to include occupancy sensor behavior
    @Override
    public void turnOn() {
        super.turnOn(); // Call the turnOn method of the superclass
        if (occupancySensor) {
            System.out.println(getName() + " turned on due to occupancy.");
        }
    }

    // Override the displayStatus method to show light-specific information
    @Override
    public void displayStatus() {
        System.out.println(getName() + ": Status=" + (isStatus() ? "ON" : "OFF") + ", Brightness=" + brightness + ", Occupancy Sensor=" + (occupancySensor ? "ON" : "OFF"));
    }
}
```

- SmartHVAC.java



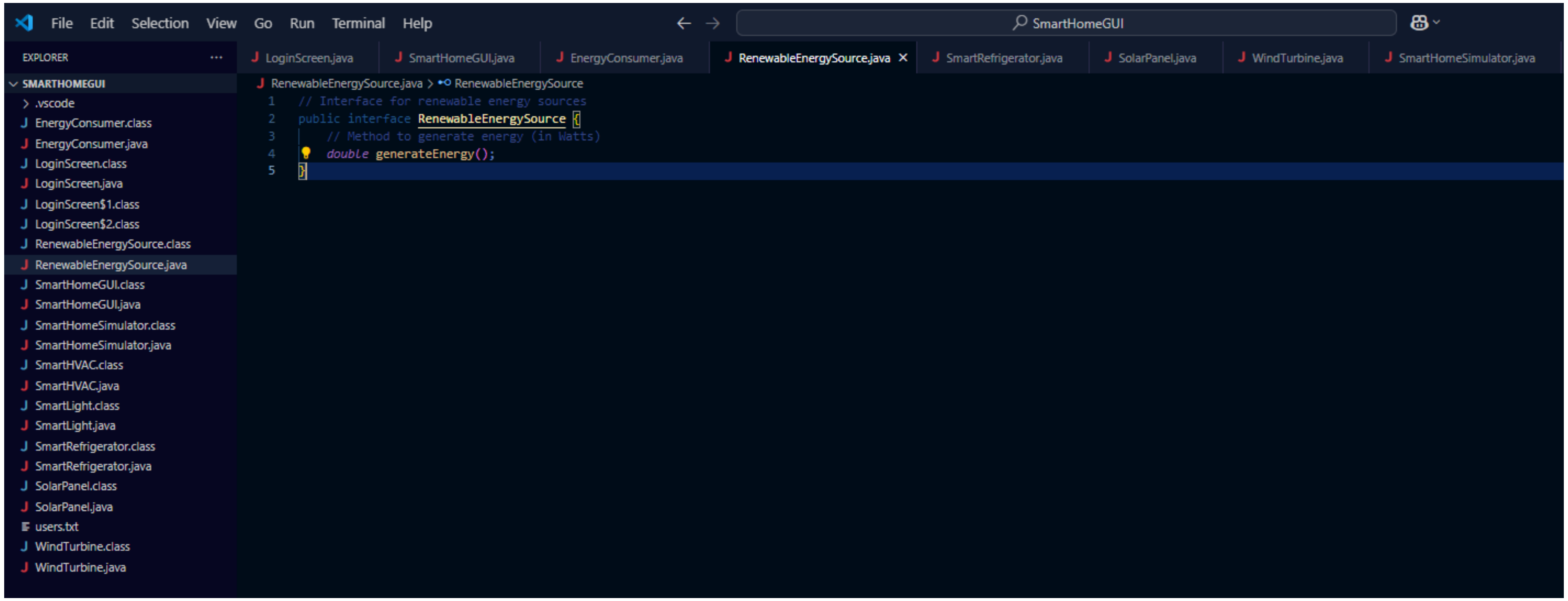
```java
// Class representing a smart HVAC system
public class SmartHVAC extends EnergyConsumer {
    private double targetTemperature; // in Celsius
    private double currentTemperature; // in Celsius
    private int fanSpeed; // e.g., 0 (off), 1 (low), 2 (medium), 3 (high)

    // Constructor for the SmartHVAC
    public SmartHVAC(String name, double powerConsumption) {
        super(name, powerConsumption);
        this.targetTemperature = 22.0; // Default target temperature
        this.fanSpeed = 0;
        this.currentTemperature = 25.0; // Initial current temperature
    }

    // Getter for the target temperature
    public double getTargetTemperature() {
        return targetTemperature;
    }

    // Setter for the target temperature
    public void setTargetTemperature(double targetTemperature) {
        this.targetTemperature = targetTemperature;
        System.out.println(getName() + " target temperature set to " + targetTemperature + "°C.");
    }

    // Getter for the fan speed
    public int getFanSpeed() {
        return fanSpeed;
    }

    // Setter for the fan speed and adjusts power consumption accordingly
    public void setFanSpeed(int fanSpeed) {
        if (fanSpeed >= 0 && fanSpeed <= 3) {
            this.fanSpeed = fanSpeed;
            // Adjust power consumption based on fan speed (example)
            switch (fanSpeed) {
                case 0:
                    super.setPowerConsumption(powerConsumption:0); // Off
                    break;
                case 1:
                    super.setPowerConsumption(powerConsumption:100); // Low
                    break;
                case 2:
                    super.setPowerConsumption(powerConsumption:300); // Medium
                    break;
                case 3:
                    super.setPowerConsumption(powerConsumption:500); // High
                    break;
            }
            System.out.println(getName() + " fan speed set to " + fanSpeed + ".");
        } else {
            System.out.println(x:"Invalid fan speed.");
        }
    }

    // Setter for the current temperature
    public void setCurrentTemperature(double currentTemperature) {
        this.currentTemperature = currentTemperature;
    }

    // Getter for the current temperature
    public double getCurrentTemperature() {
        return currentTemperature;
    }

    // Override the displayStatus method to show HVAC-specific information
    @Override
    public void displayStatus() {
        System.out.println(getName() + ": Status=" + (isStatus() ? "ON" : "OFF") + ", Target Temperature=" + targetTemperature + "°C, Fan Speed=" + fanSpeed + ", Current Temperature = " + currentTemperature + "C");
    }
}
```

- SmartRefrigerator.java



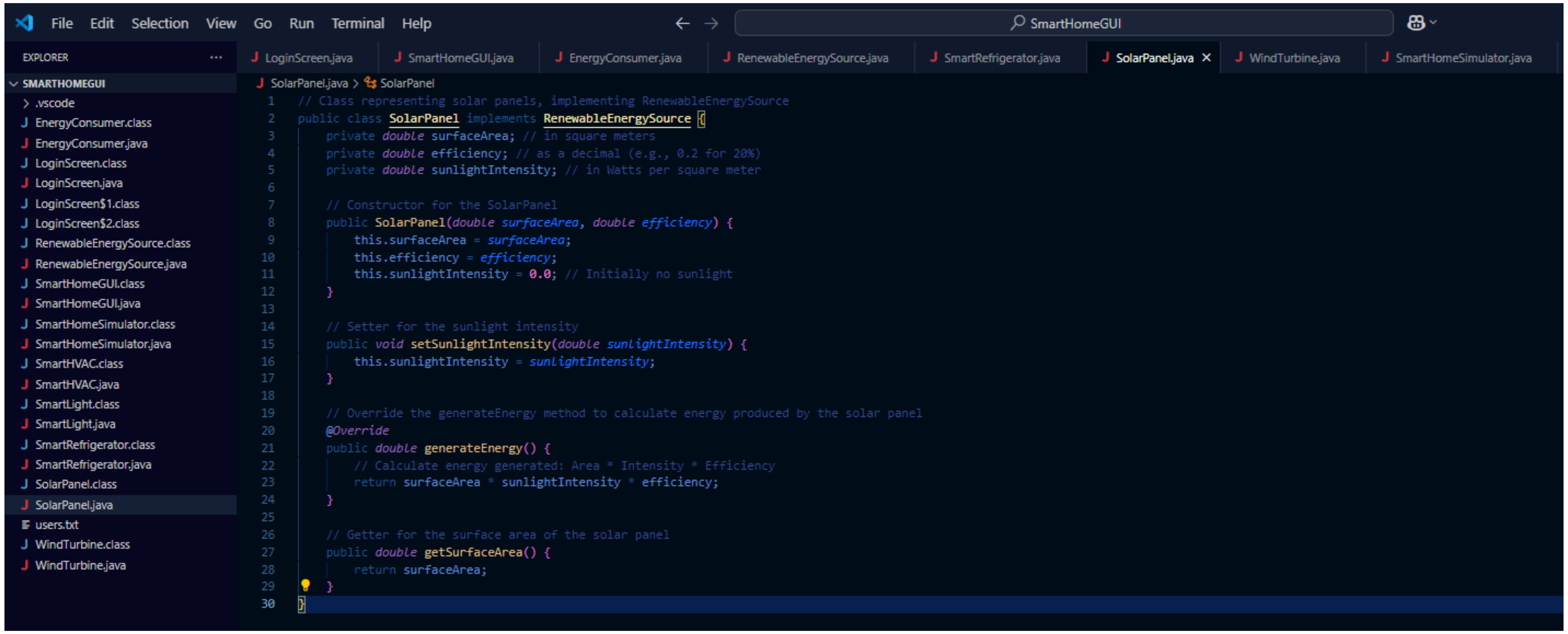
```java
// Class representing a smart refrigerator
public class SmartRefrigerator extends EnergyConsumer {
    private double internalTemperature; // in Celsius

    // Constructor for the SmartRefrigerator
    public SmartRefrigerator(String name, double powerConsumption) {
        super(name, powerConsumption);
        this.internalTemperature = 4.0; // Default refrigerator temperature
    }

    // Getter for the internal temperature
    public double getInternalTemperature() {
        return internalTemperature;
    }

    // Setter for the internal temperature
    public void setInternalTemperature(double internalTemperature) {
        this.internalTemperature = internalTemperature;
        System.out.println(getName() + " internal temperature set to " + internalTemperature + "°C.");
    }

    // Override the displayStatus method to show refrigerator-specific information
    @Override
    public void displayStatus() {
        System.out.println(getName() + ": Status=" + (isStatus() ? "ON" : "OFF") + ", Internal Temperature=" + internalTemperature + "°C");
    }
}
```

- RenewableEnergySource.java (interface and its implementing classes like SolarPanel.java, WindTurbine.java)



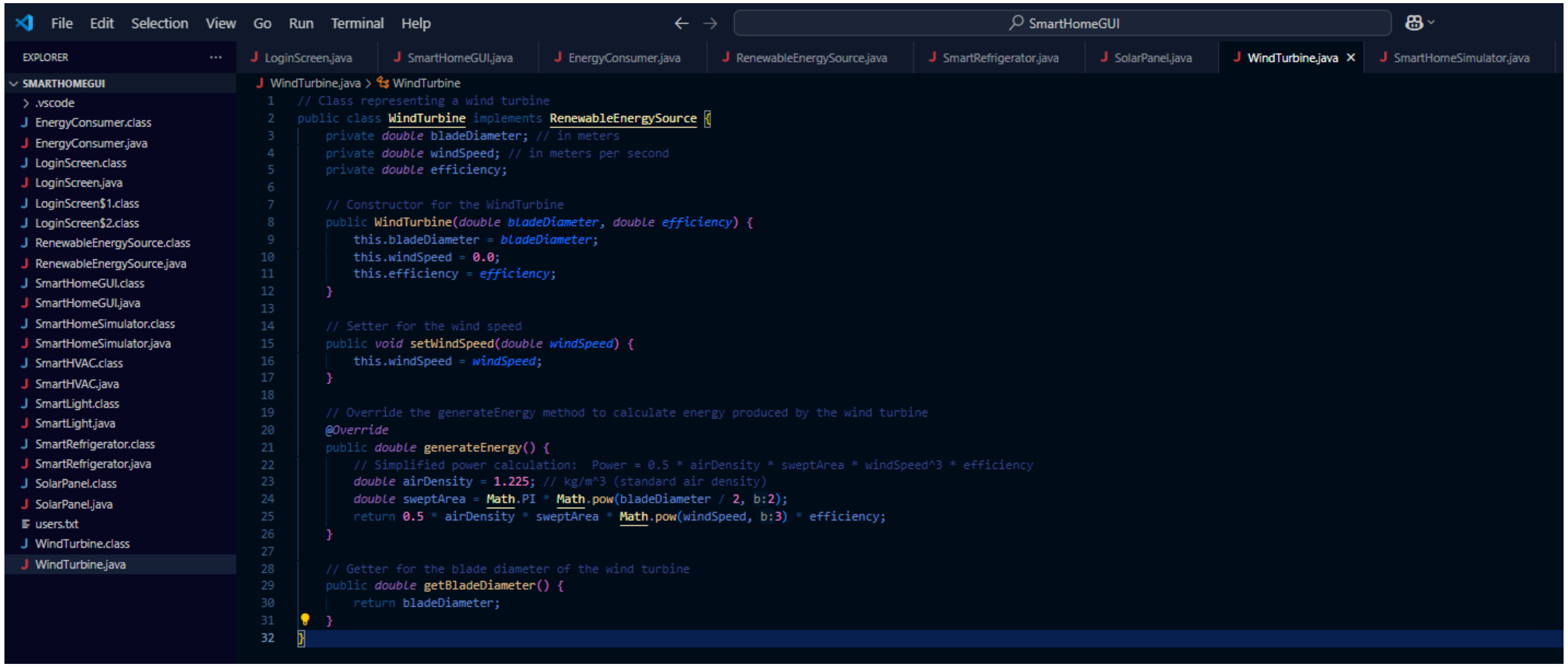
```java
// Interface for renewable energy sources
public interface RenewableEnergySource {
    // Method to generate energy (in Watts)
    double generateEnergy();
}
```

- SolarPanel.java



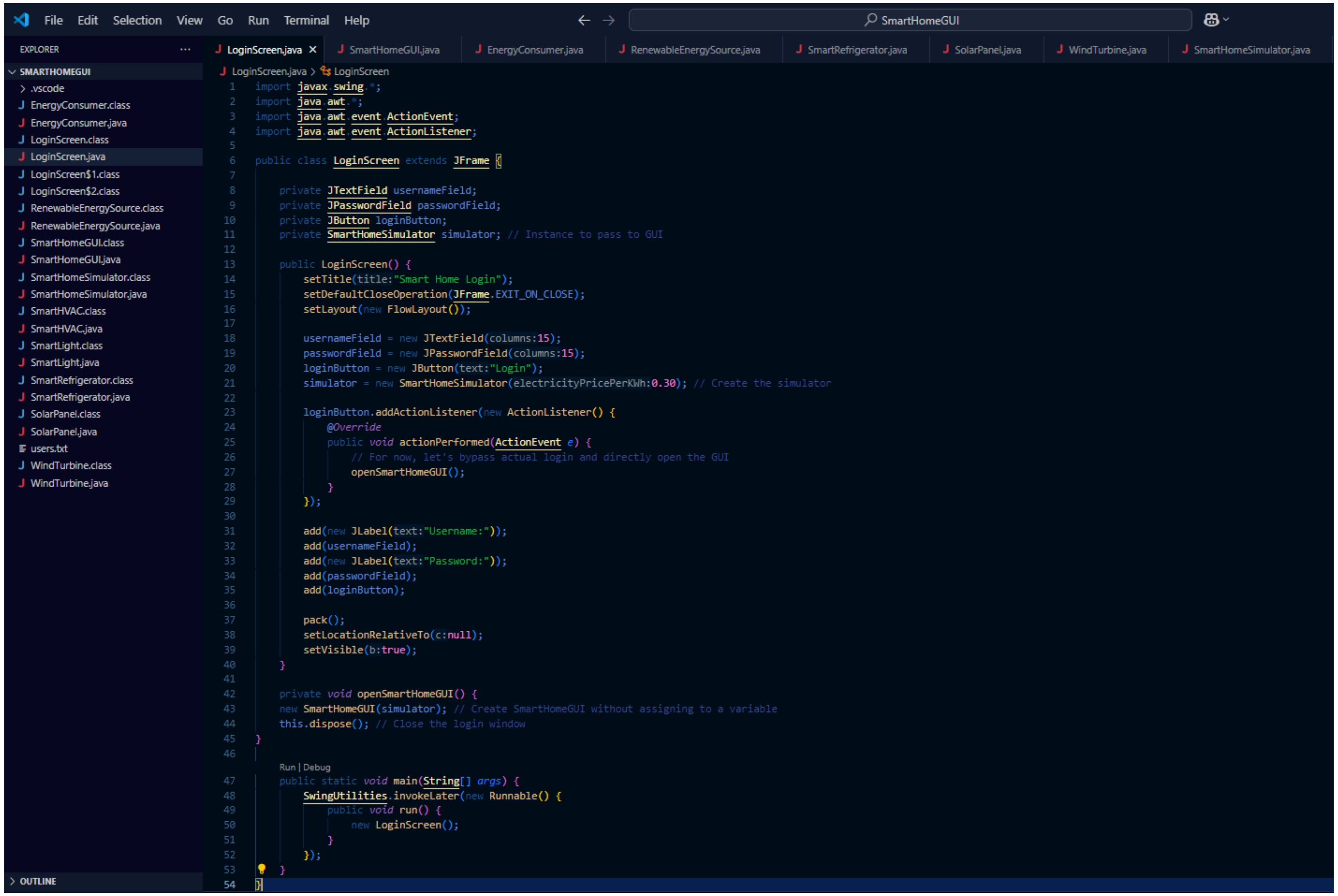
```java
// Class representing solar panels, implementing RenewableEnergySource
public class SolarPanel implements RenewableEnergySource {
    private double surfaceArea; // in square meters
    private double efficiency; // as a decimal (e.g., 0.2 for 20%)
    private double sunlightIntensity; // in Watts per square meter

    // Constructor for the SolarPanel
    public SolarPanel(double surfaceArea, double efficiency) {
        this.surfaceArea = surfaceArea;
        this.efficiency = efficiency;
        this.sunlightIntensity = 0.0; // Initially no sunlight
    }

    // Setter for the sunlight intensity
    public void setSunlightIntensity(double sunlightIntensity) {
        this.sunlightIntensity = sunlightIntensity;
    }

    // Override the generateEnergy method to calculate energy produced by the solar panel
    @Override
    public double generateEnergy() {
        // Calculate energy generated: Area * Intensity * Efficiency
        return surfaceArea * sunlightIntensity * efficiency;
    }

    // Getter for the surface area of the solar panel
    public double getSurfaceArea() {
        return surfaceArea;
    }
}
```

- WindTurbine.java



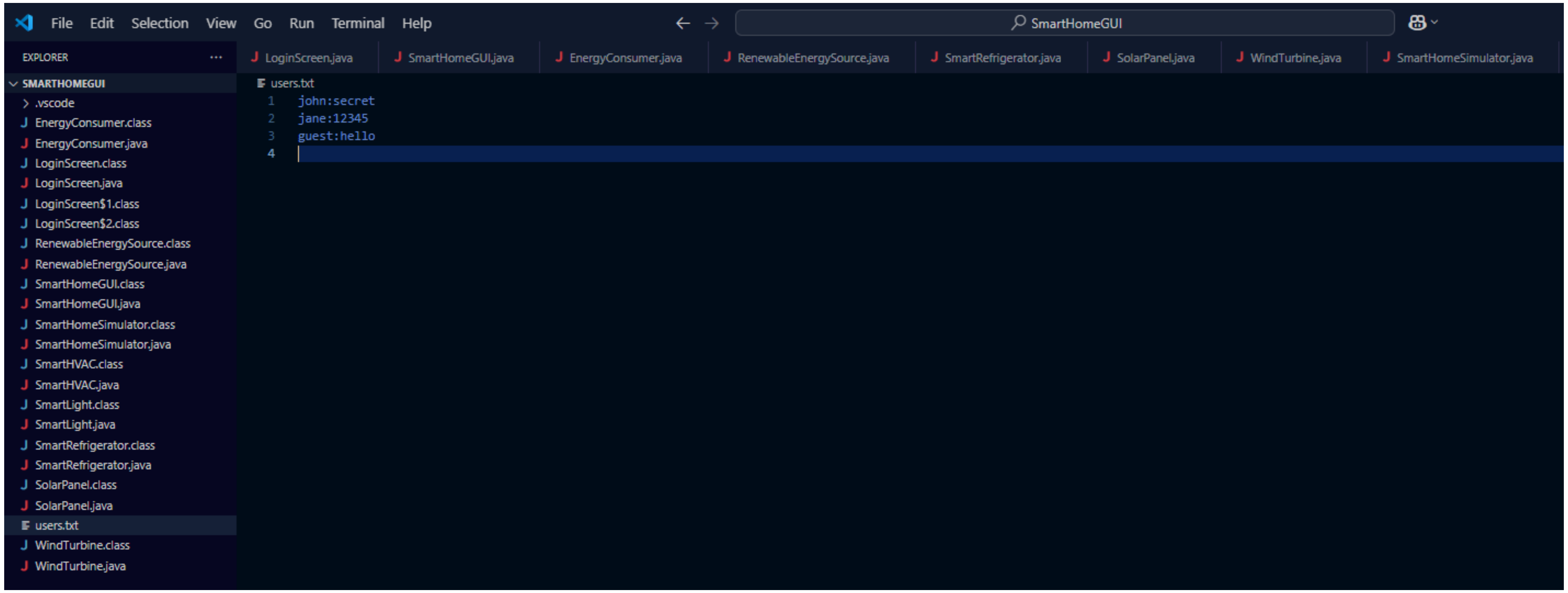
```java
// Class representing a wind turbine
public class WindTurbine implements RenewableEnergySource {
    private double bladeDiameter; // in meters
    private double windSpeed; // in meters per second
    private double efficiency;

    // Constructor for the WindTurbine
    public WindTurbine(double bladeDiameter, double efficiency) {
        this.bladeDiameter = bladeDiameter;
        this.windSpeed = 0.0;
        this.efficiency = efficiency;
    }

    // Setter for the wind speed
    public void setWindSpeed(double windSpeed) {
        this.windSpeed = windSpeed;
    }

    // Override the generateEnergy method to calculate energy produced by the wind turbine
    @Override
    public double generateEnergy() {
        // Simplified power calculation:  Power = 0.5 * airDensity * sweptArea * windSpeed^3 * efficiency
        double airDensity = 1.225; // kg/m^3 (standard air density)
        double sweptArea = Math.PI * Math.pow(bladeDiameter / 2, b:2);
        return 0.5 * airDensity * sweptArea * Math.pow(windSpeed, b:3) * efficiency;
    }

    // Getter for the blade diameter of the wind turbine
    public double getBladeDiameter() {
        return bladeDiameter;
    }
}
```

- Any other supporting classes or utilities (interface and its implementing classes like LoginScreen.java, users.txt)



```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class LoginScreen extends JFrame {

    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton;
    private SmartHomeSimulator simulator; // Instance to pass to GUI

    public LoginScreen() {
        setTitle(title:"Smart Home Login");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        usernameField = new JTextField(columns:15);
        passwordField = new JPasswordField(columns:15);
        loginButton = new JButton(text:"Login");
        simulator = new SmartHomeSimulator(electricityPricePerKWh:0.30); // Create the simulator

        loginButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // For now, let's bypass actual login and directly open the GUI
                openSmartHomeGUI();
            }
        });

        add(new JLabel(text:"Username:"));
        add(usernameField);
        add(new JLabel(text:"Password:"));
        add(passwordField);
        add(loginButton);

        pack();
        setLocationRelativeTo(c:null);
        setVisible(b:true);
    }

    private void openSmartHomeGUI() {
    new SmartHomeGUI(simulator); // Create SmartHomeGUI without assigning to a variable
    this.dispose(); // Close the login window
}

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new LoginScreen();
            }
        });
    }
}
```

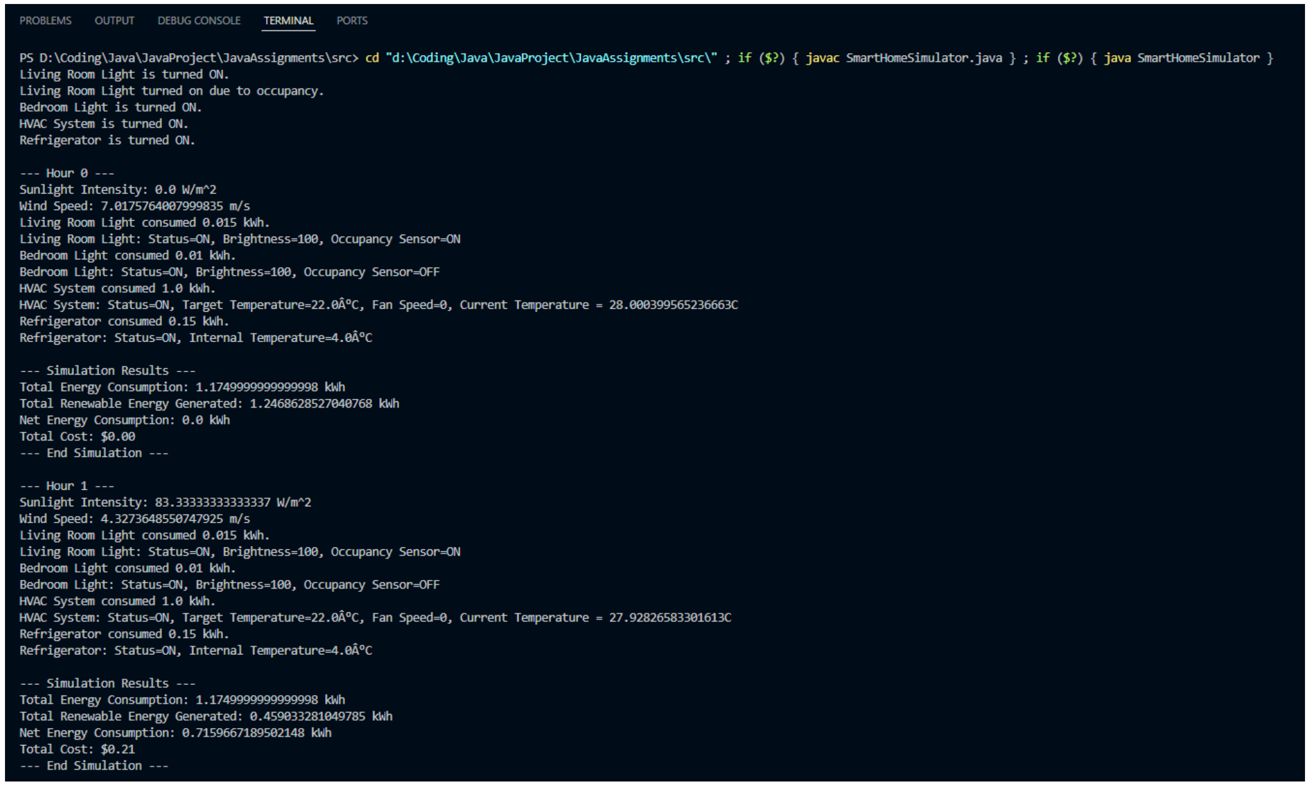
- users.txt



```
john:secret
jane:12345
guest:hello

```

## B. Sample Input/Output Data:

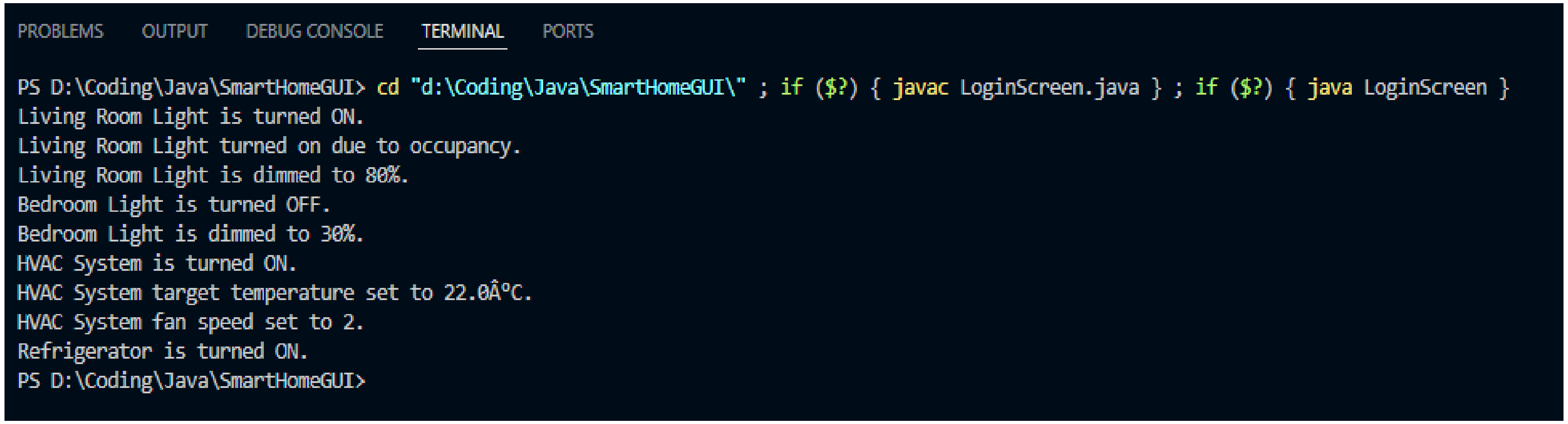
- **Example Console Output:** A complete log of the terminal output generated during a typical simulation run, showcasing device status changes, energy consumption, generation, and cost calculations.



```
PS D:\Coding\Java\JavaProject\JavaAssignments\src> cd "d:\Coding\Java\JavaProject\JavaAssignments\src\" ; if ($?) { javac SmartHomeSimulator.java } ; if ($?) { java SmartHomeSimulator }
Living Room Light is turned ON.
Living Room Light turned on due to occupancy.
Bedroom Light is turned ON.
HVAC System is turned ON.
Refrigerator is turned ON.

--- Hour 0 ---
Sunlight Intensity: 0.0 W/m^2
Wind Speed: 7.0175764007999835 m/s
Living Room Light consumed 0.015 kWh.
Living Room Light: Status=ON, Brightness=100, Occupancy Sensor=ON
Bedroom Light consumed 0.01 kWh.
Bedroom Light: Status=ON, Brightness=100, Occupancy Sensor=OFF
HVAC System consumed 1.0 kWh.
HVAC System: Status=ON, Target Temperature=22.0Â°C, Fan Speed=0, Current Temperature = 28.000399565236663C
Refrigerator consumed 0.15 kWh.
Refrigerator: Status=ON, Internal Temperature=4.0Â°C

--- Simulation Results ---
Total Energy Consumption: 1.1749999999999998 kWh
Total Renewable Energy Generated: 1.2468628527040768 kWh
Net Energy Consumption: 0.0 kWh
Total Cost: $0.00
--- End Simulation ---

--- Hour 1 ---
Sunlight Intensity: 83.33333333333337 W/m^2
Wind Speed: 4.3273648550747925 m/s
Living Room Light consumed 0.015 kWh.
Living Room Light: Status=ON, Brightness=100, Occupancy Sensor=ON
Bedroom Light consumed 0.01 kWh.
Bedroom Light: Status=ON, Brightness=100, Occupancy Sensor=OFF
HVAC System consumed 1.0 kWh.
HVAC System: Status=ON, Target Temperature=22.0Â°C, Fan Speed=0, Current Temperature = 27.92826583301613C
Refrigerator consumed 0.15 kWh.
Refrigerator: Status=ON, Internal Temperature=4.0Â°C

--- Simulation Results ---
Total Energy Consumption: 1.1749999999999998 kWh
Total Renewable Energy Generated: 0.459033281049785 kWh
Net Energy Consumption: 0.7159667189502148 kWh
Total Cost: $0.21
--- End Simulation ---
```

Figure 2: Sample Console Output of the Eco-Friendly Smart Home Simulator

- **GUI Interaction Flow:** Screenshots or a sequence of screenshots demonstrating typical user interactions with the GUI, showing how devices are controlled and how the status updates.



```
PS D:\Coding\Java\SmartHomeGUI> cd "d:\Coding\Java\SmartHomeGUI\" ; if ($?) { javac LoginScreen.java } ; if ($?) { java LoginScreen }
Living Room Light is turned ON.
Living Room Light turned on due to occupancy.
Living Room Light is dimmed to 80%.
Bedroom Light is turned OFF.
Bedroom Light is dimmed to 30%.
HVAC System is turned ON.
HVAC System target temperature set to 22.0Â°C.
HVAC System fan speed set to 2.
Refrigerator is turned ON.
PS D:\Coding\Java\SmartHomeGUI>
```

Figure 3: Terminal Output (GUI) of the Eco-Friendly Smart Home Simulator

## D. User Manual/Instructions:

- This section provides a brief guide on how to set up, compile, run, and interact with the Eco-Friendly Smart Home Simulator application.

### 1. System Requirements:

- **Java Development Kit (JDK):** Version 8 or higher.
- **Operating System:** Windows, macOS, or Linux.
- **Integrated Development Environment (IDE):** VS Code (recommended) or any other Java-compatible IDE (e.g., IntelliJ IDEA, Eclipse).

### 2. How to Compile and Run the Application:

- The application can be compiled and run using either VS Code (recommended for ease of use) or the command line.

**A. Using VS Code (Recommended):**

- **Open Project:** Open the project folder (containing all my .java files) in VS Code.
- **Install Java Extensions:** Ensure I have the "Extension Pack for Java" installed from the VS Code Marketplace. This provides language support, debugging, and build automation.
- **Run LoginScreen.java:**
- Navigate to the LoginScreen.java file in the VS Code Explorer.
- Look for the green "Run" arrow button that appears next to the main method (or at the top right of the editor).
- Click this "Run" button.
- **Alternatively, Run from Terminal:**
- Open the integrated terminal in VS Code (Ctrl+`` orView > Terminal`).
- The Java extensions typically handle compilation automatically. I can usually run the main class directly if my project is set up: java -cp . LoginScreen (This

assumes LoginScreen.java is my main class with the main method and all compiled .class files are in the current directory or specified classpath).

**B. Using Command Line:**

- **Navigate to Source Directory:** Open my terminal or command prompt and navigate to the directory where all my .java source files are located (e.g., cd path/to/my/project/src).
- **Compile:** Compile all Java files. Ensure all necessary .java files (like EnergyConsumer.java, SmartLight.java, SmartHomeSimulator.java, SmartHomeGUI.java, etc.) are in the same directory or correctly referenced.
- Bash
- javac *.java
- If I encounter issues with class dependencies, we might need to compile specific files in order, or use a tool like Maven/Gradle (which I am not currently using).
- **Run:** Execute the main class (LoginScreen.java is my entry point).

- Bash
- java LoginScreen
- **3. Instructions on How to Interact with the GUI:**
- Upon successful execution, the graphical user interface (GUI) will appear, allowing interactive control of the simulated smart home.

**a. Login Screen:**

- **Credentials:** Enter the username and password in the provided fields. (Based on common login screens, default might be user/pass or similar if not specified).
- **Login:** Click the "Login" button. If credentials are correct, the main "Smart Home Controller" window will appear. An incorrect login will show an error message.

**b. Main Smart Home Controller Interface:**

- The main window displays a list of all simulated smart home devices. Each device is presented with its name, current status, and specific control buttons.
- **Device List:** Devices are listed vertically.
- **Status Display:** For each device, a label shows its current status (e.g., "Status: ON" or "Status: OFF"). For specific devices, additional details like brightness, target temperature, or fan speed are displayed.

**c. Controlling Devices:**

- **Turning Devices ON/OFF:**
- Click the **"ON" button** next to a device to turn it on.
- Click the **"OFF" button** next to a device to turn it off.
- The "Status" label for the device will update immediately.
- **Setting Parameters for SmartLight:**
- Locate a SmartLight device in the list.

- Click the **"Dim" button**.
- A small input dialog box will appear.
- Enter the desired **brightness level** as a number between **0 and 100** (0 for off, 100 for full brightness).
- Click "OK". The brightness label for the light will update, and its power consumption will be adjusted in the simulation.
- **Setting Parameters for SmartHVAC:**
- Locate a SmartHVAC device.
- Click the **"Set Temp" button**.
- An input dialog will appear.
- Enter the **target temperature** in Celsius (e.g., 22.0).
- Click "OK". The "Target Temp" label will update.
- Click the **"Set Fan" button**.
- An input dialog will appear.

- Enter the **fan speed** as a number between **0 and 3** (0 for off, 1 for low, 2 for medium, 3 for high).
- Click "OK". The "Fan" label will update, and the HVAC's power consumption will adjust based on the fan speed.

**d. Interpreting Displayed Information:**

- **GUI Updates:** All changes made through the GUI (turning devices on/off, setting parameters) are immediately reflected in the labels next to the respective devices.
- **Console Output (Backend Simulation Log):**
- While I interact with the GUI, keep an eye on my **terminal or command prompt (where I ran the application)**.
- The simulator continuously prints detailed information about the simulation's state, including:
- When a device is turned ON/OFF.
- When parameters (brightness, temperature, fan speed) are changed.

- Hourly updates on:
- Individual device energy consumption (in kWh).
- Total energy consumption of all devices (in kWh).
- Total renewable energy generated (in kWh).
- Net energy consumption (total consumption - total renewable energy).
- Total cost incurred (based on net consumption and electricity price).
- This console output serves as a comprehensive log of the simulation's progress and results.

By following these instructions, users can effectively operate and gain insights from the Eco-Friendly Smart Home Simulator.

## Digital Signature:

**(This section is for the formal acknowledgment and signing of the document.)**

Durjoy Barua

---

**Durjoy Barua**

**Roll: 24070134**

**University Of Science & Technology Chittagong (USTC)**

**Date of Submission: 22 May 2025**

---