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: **Wednesday, 28 May, 2025**

Table of Contents

List of Figures:

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).
  - 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).

- **User Interface:**
  - **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.

➢ 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:

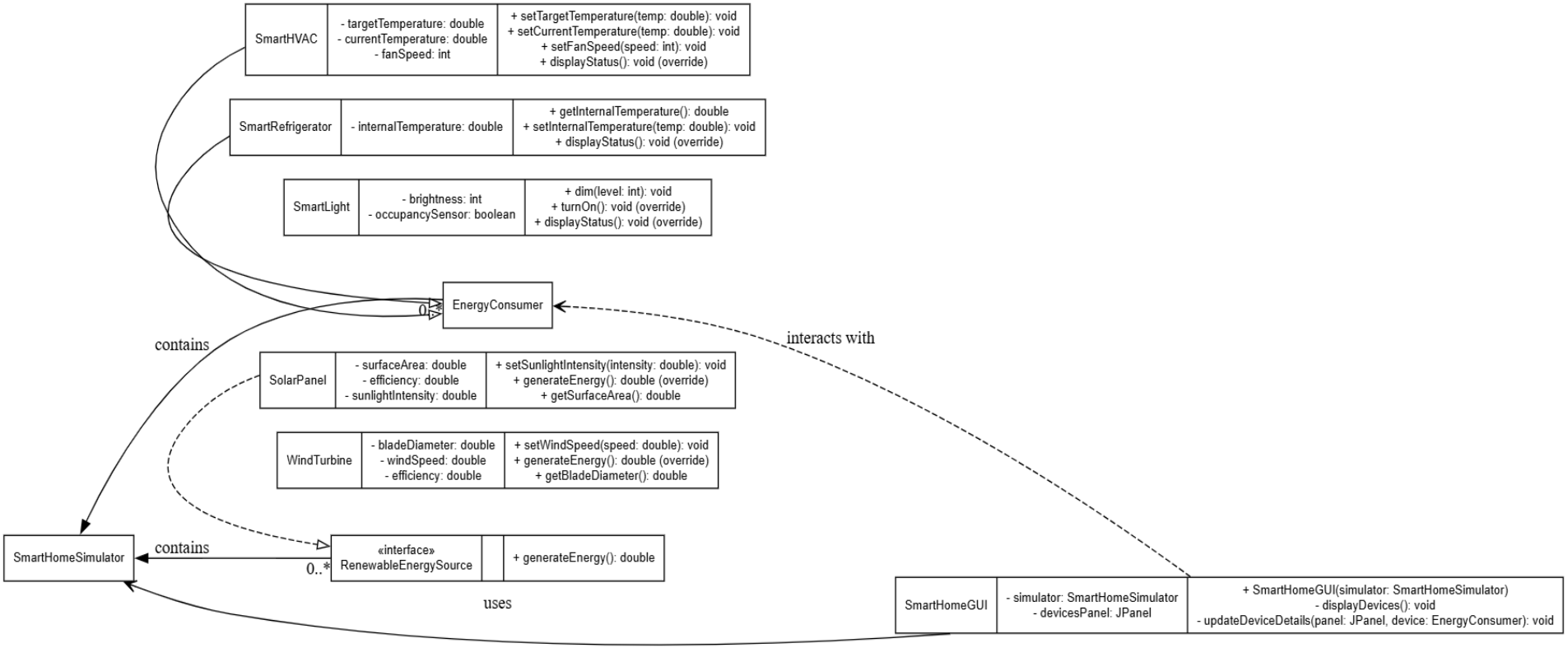


Figure 1: Class Diagram of the Eco-Friendly Smart Home Simulator

Source: https://dreampuf.github.io/GraphvizOnline/

❖ Explanation

This image is a **UML Class Diagram**, which shows the structure of a system by displaying its classes, attributes, operations (methods), and relationships among objects.

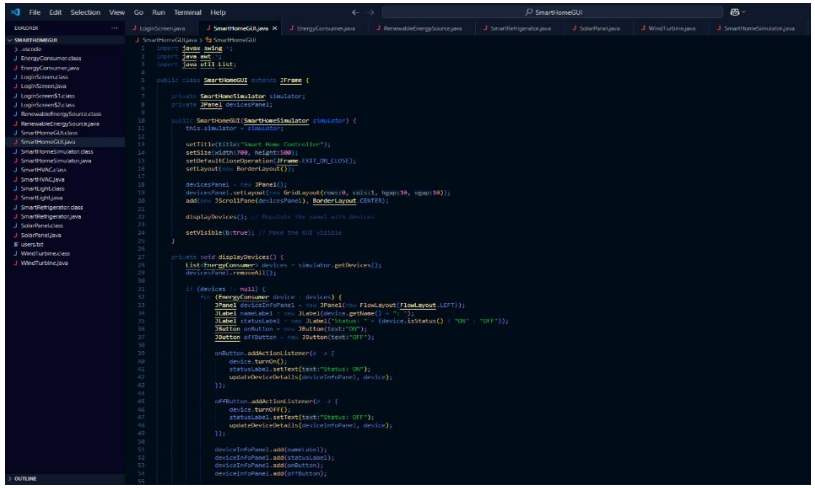
Here's a breakdown of what the diagram illustrates:

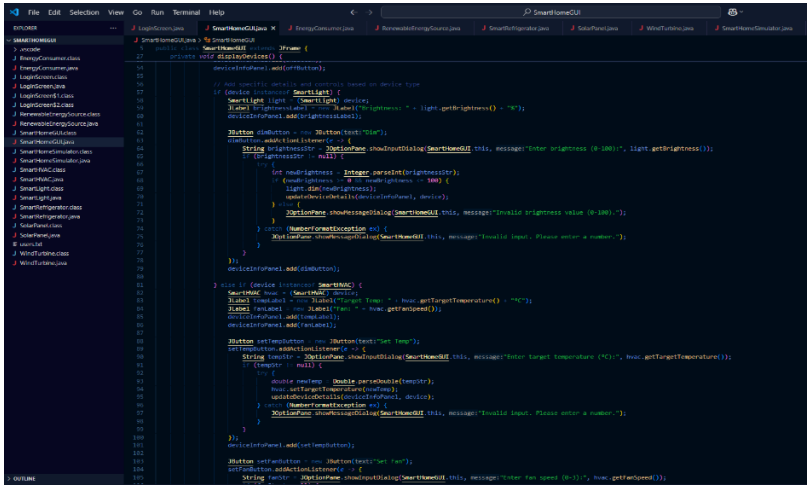
- **Classes and their attributes/operations:**
    - **SmartHVAC:** targetTemperature, currentTemperature, fanSpeed (attributes); setTargetTemperature(), setCurrentTemperature(), setFanSpeed(), displayStatus() (operations).
    - **SmartRefrigerator:** internalTemperature (attribute); getInternalTemperature(), setInternalTemperature(), displayStatus() (operations).
    - **SmartLight:** brightness, occupancySensor (attributes); dim(), turnOn(), displayStatus() (operations).
    - **EnergyConsumer:** This is an interface (indicated by <<interface>>) that has a generateEnergy() operation. This suggests that any class implementing this interface must provide a generateEnergy() method.
    - **SolarPanel:** surfaceArea, efficiency, sunlightIntensity (attributes); setSunlightIntensity(), generateEnergy(), getSurfaceArea() (operations).
    - **WindTurbine:** bladeDiameter, windSpeed, efficiency (attributes); setWindSpeed(), generateEnergy(), getBladeDiameter() (operations).
    - **RenewableEnergySource:** This is also an interface, with a generateEnergy() operation. SolarPanel and WindTurbine implement this interface.
    - **SmartHomeSimulator:** simulator (attribute); SmartHomeSimulator() (constructor).
    - **SmartHomeGUI:** displayDevices(), updateDeviceDetails() (operations).

- **Relationships:**
  - **Association (solid line):**
    - **SmartHomeSimulator** "contains" **SmartHVAC**, **SmartRefrigerator**, and **SmartLight**. This means an instance of **SmartHomeSimulator** will have references to these smart home devices.
    - **SmartHomeSimulator** "contains" **RenewableEnergySource** (with a 0..* multiplicity, meaning zero or more).
    - **EnergyConsumer** "interacts with" **SmartHVAC**, **SmartRefrigerator**, and **SmartLight**. This likely implies that these smart devices consume energy provided by an EnergyConsumer.
  - **Realization/Implementation (dashed line with empty triangle):**
    - **SolarPanel** and **WindTurbine** "uses" (implements) the **RenewableEnergySource** interface. This means they provide the implementation for the generateEnergy() method defined in **RenewableEnergySource**.
    - **EnergyConsumer** is also an interface, and it looks like **SmartHVAC**, **SmartRefrigerator**, and **SmartLight** are somehow related to it, possibly implementing it implicitly or explicitly, or interacting with it in a consumer role.
  - **Dependency (dashed line with arrow):**
    - **SmartHomeGUI** depends on **SmartHomeSimulator**. This means **SmartHomeGUI** uses or interacts with **SmartHomeSimulator** to perform its functions (e.g., to get data for display).

In summary, this diagram depicts a simulated smart home system where a SmartHomeSimulator manages various smart devices (HVAC, Refrigerator, Light) and renewable energy sources (Solar Panel, Wind Turbine). There's an EnergyConsumer interface that these devices might implement or interact with for energy consumption. A SmartHomeGUI provides a graphical interface to interact with and display information from the SmartHomeSimulator.

➢ Code:





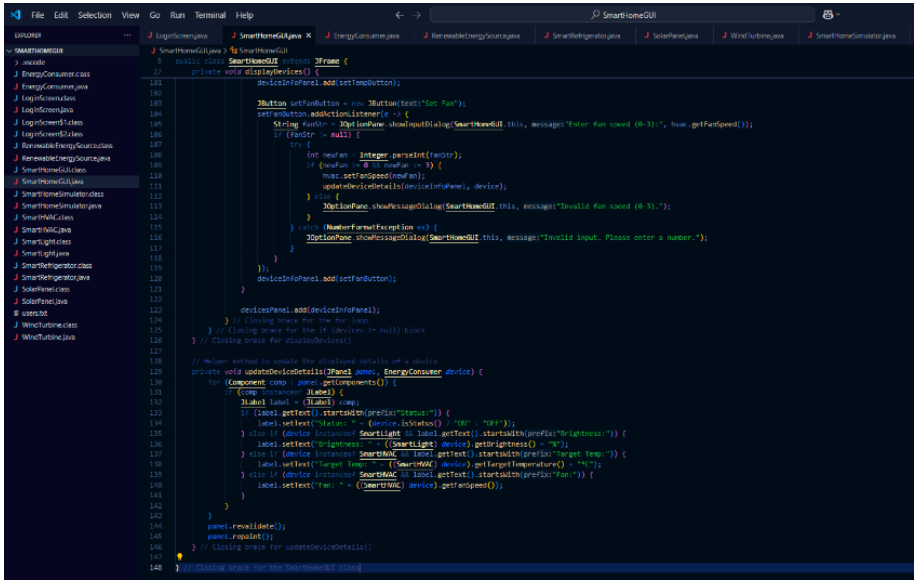


Figure 2: **SmartHomeGUI.java**

❖ Breakdown

The provided Java code defines the **SmartHomeGUI** class, which is a **JFrame** (a top-level window) that serves as the graphical user interface for a **SmartHomeSimulator**.

Here's a detailed breakdown of the code:

1. **Imports:**

- **javax.swing.*:** Imports core Swing classes for GUI components (JFrame, JPanel, JLabel, JButton, JOptionPane, JScrollPane).
- **java.awt.*:** Imports AWT (Abstract Window Toolkit) classes for basic GUI functionalities like layout managers (BorderLayout, GridLayout, FlowLayout) and Components.
- **java.util.List:** Imports the List interface for handling collections of objects.

2. **Class Definition: SmartHomeGUI**

- Extends JFrame: This makes **SmartHomeGUI** a window itself.
- **private SmartHomeSimulator simulator;:** An instance of **SmartHomeSimulator** is held to interact with the backend simulation logic.
- **private JPanel devicesPanel;:** A JPanel to hold and display information and controls for each smart device.

3. **Constructor: SmartHomeGUI**(SmartHomeSimulator simulator)

- **this.simulator = simulator;:** Initializes the simulator instance.
- **setTitle("Smart Home Controller");:** Sets the window title.
- **setSize(700, 500);:** Sets the initial size of the window to 700 pixels wide and 500 pixels high.
- **setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);:** Configures the window to exit the application when the close button is clicked.
- **setLayout(new BorderLayout());:** Sets the layout manager for the JFrame to BorderLayout.
- **devicesPanel = new JPanel();:** Creates a new JPanel.

- **devicesPanel.setLayout(new GridLayout(0, 1, 10, 10));:** Sets the layout manager for devicesPanel to GridLayout.
    - 0: Means any number of rows.
    - 1: Means one column.
    - 10, 10: Specifies horizontal and vertical gaps of 10 pixels between components.
- **add(new JScrollPane(devicesPanel), BorderLayout.CENTER);:** Adds the devicesPanel inside a JScrollPane. This makes the devicesPanel scrollable if its content exceeds its visible area. The scroll pane is placed in the center of the JFrame's BorderLayout.
- **displayDevices();:** Calls a private method to initially populate the devicesPanel with device information.
- **setVisible(true);:** Makes the GUI window visible to the user.

4. **Private Method: displayDevices()**

- This method is responsible for retrieving the list of devices from the simulator and displaying them in the devicesPanel.
- **List<EnergyConsumer> devices = simulator.getDevices();:** Retrieves a list of EnergyConsumer objects from the simulator.
- **devicesPanel.removeAll();:** Clears any existing components from the devicesPanel before re-populating it. This is crucial for refreshing the display.
- **if (devices != null) { ... }:** Checks if the list of devices is not null.
- Looping through devices: for **(EnergyConsumer device : devices):** Iterates over each **EnergyConsumer** (which includes **SmartLight** and **SmartHVAC** as per the UML).
    - JPanel deviceInfoPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));: Creates a new JPanel for each device to hold its specific controls and information. It uses FlowLayout.LEFT to align components to the left.
    - JLabel nameLabel = new JLabel(device.getName() + ": ");: Displays the name of the device. (Assumes EnergyConsumer has a getName() method).
    - JLabel statusLabel = new JLabel("Status: " + (device.isStatus() ? "ON" : "OFF"));: Displays the current ON/OFF status of the device. (Assumes EnergyConsumer has an isStatus() method).

- JButton onButton = new JButton("ON"); and JButton offButton = new JButton("OFF");: Creates ON and OFF buttons for the device.
- Action Listeners for ON/OFF buttons:
  - When "ON" is clicked: device.turnOn();, statusLabel.setText("Status: ON");, and updateDeviceDetails(deviceInfoPanel, device); are called. (Assumes EnergyConsumer has turnOn() and turnOff() methods).
  - When "OFF" is clicked: Similar logic for turning off the device.
- deviceInfoPanel.add(...): Adds the name, status labels, and ON/OFF buttons to the deviceInfoPanel.
- Device-specific controls:
  - **if (device instanceof SmartLight):** If the device is a SmartLight:
    - JLabel brightnessLabel: Displays the current brightness.
    - JButton dimButton: Allows changing the brightness.
    - Dim Button Action Listener:
      - Uses JOptionPane.showInputDialog to get brightness input from the user.
      - Parses the input, validates it (0-100), calls light.dim(), and then updateDeviceDetails() to refresh the display.
      - Includes error handling for NumberFormatException.
  - **else if (device instanceof SmartHVAC):** If the device is a SmartHVAC:
    - JLabel tempLabel: Displays the target temperature.
    - JLabel fanLabel: Displays the fan speed.
    - JButton setTempButton: Allows changing the target temperature.
    - Set Temp Button Action Listener: Similar input, parsing, validation, and update logic as the Dim button.

        - JButton setFanButton: Allows changing the fan speed.
        - Set Fan Button Action Listener: Similar input, parsing, validation (0-3), and update logic as the Set Temp button.
    - **devicesPanel.add(deviceInfoPanel);:** Adds the deviceInfoPanel (containing all controls for a single device) to the main devicesPanel.

5. **Private Helper Method:** updateDeviceDetails(JPanel panel, EnergyConsumer device)

- This method is called to refresh the displayed details of a *specific* device on its deviceInfoPanel. It iterates through the components of the given panel.
- for (Component comp : panel.getComponents()): Iterates through all components within the deviceInfoPanel.
- if (comp instanceof JLabel): Checks if the component is a JLabel.
- Updates labels based on device type and label text:
    - Updates the "Status" label.
    - If SmartLight, updates the "Brightness" label.
    - If SmartHVAC, updates the "Target Temp" and "Fan" labels.
- **panel.revalidate(); and panel.repaint();:** These methods are called to ensure that the panel's layout is re-calculated and it is redrawn to reflect the updated information.

Assumptions about other classes (based on the provided code and UML):

- **SmartHomeSimulator:**
    - Must have a constructor that accepts devices.
    - Must have a getDevices() method that returns List<EnergyConsumer>.
- **EnergyConsumer** (Interface or Abstract Class):
    - Must have getName() method (returns String).

    - Must have isStatus() method (returns boolean).
    - Must have turnOn() method (returns void).
    - Must have turnOff() method (returns void).
- **SmartLight** (Concrete class implementing/extending EnergyConsumer):
    - Must have getBrightness() method (returns int).
    - Must have dim(int brightness) method (returns void).
- **SmartHVAC** (Concrete class implementing/extending EnergyConsumer):
    - Must have getTargetTemperature() method (returns double).
    - Must have setTargetTemperature(double temp) method (returns void).
    - Must have getFanSpeed() method (returns int).
    - Must have setFanSpeed(int speed) method (returns void).

This **SmartHomeGUI** provides a functional interface for controlling and monitoring smart devices in a simulated environment.

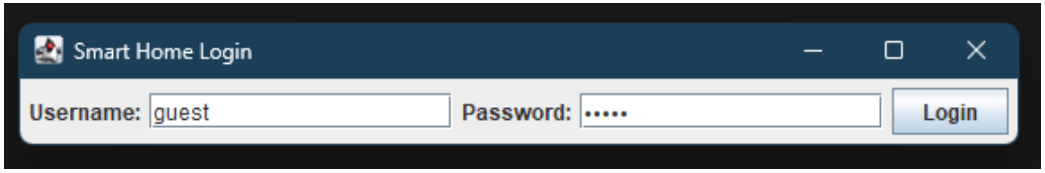
➢ Output:

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.

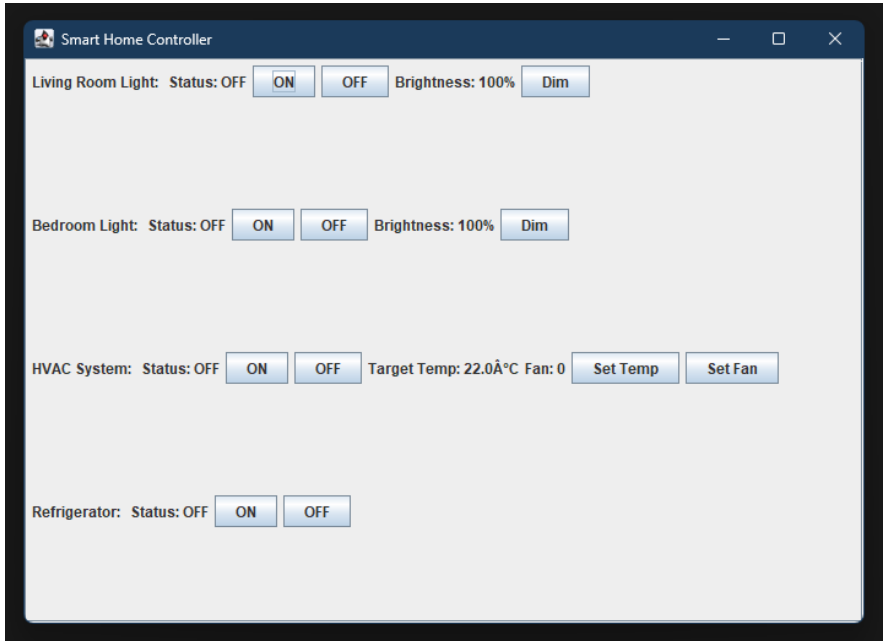
- **GUI Screenshots**

Login Screen



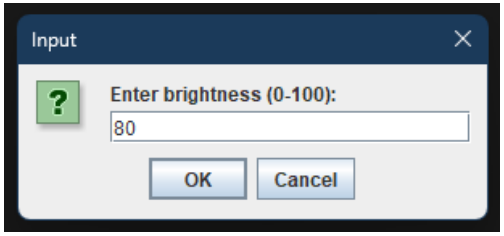
➢ The initial login screen of the Smart Home Controller.

Main Controller



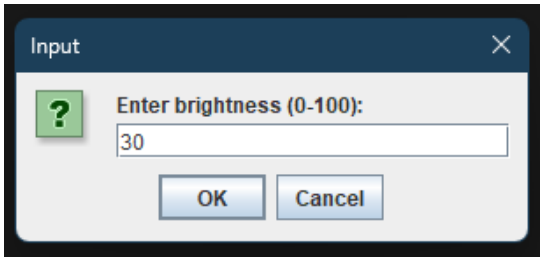
- The main interface for controlling smart devices. Here, I can see the list of added devices with their current status and basic controls.

Setting Brightness (80%)



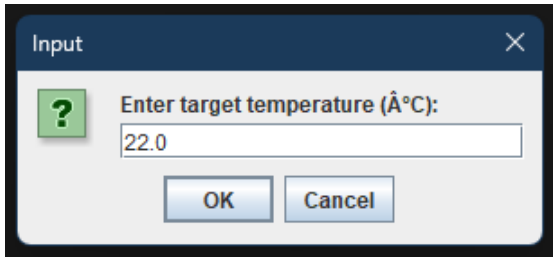
- An input dialog box allows users to precisely set the brightness level for smart lights. In this example, the brightness is being set to 80%.

Setting Brightness (30%)



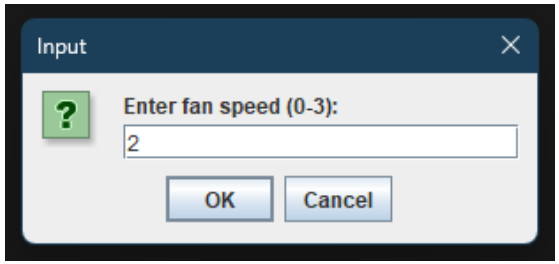
- Another instance of the brightness input dialog, this time showing the light being dimmed to 30%.

Setting Target Temperature



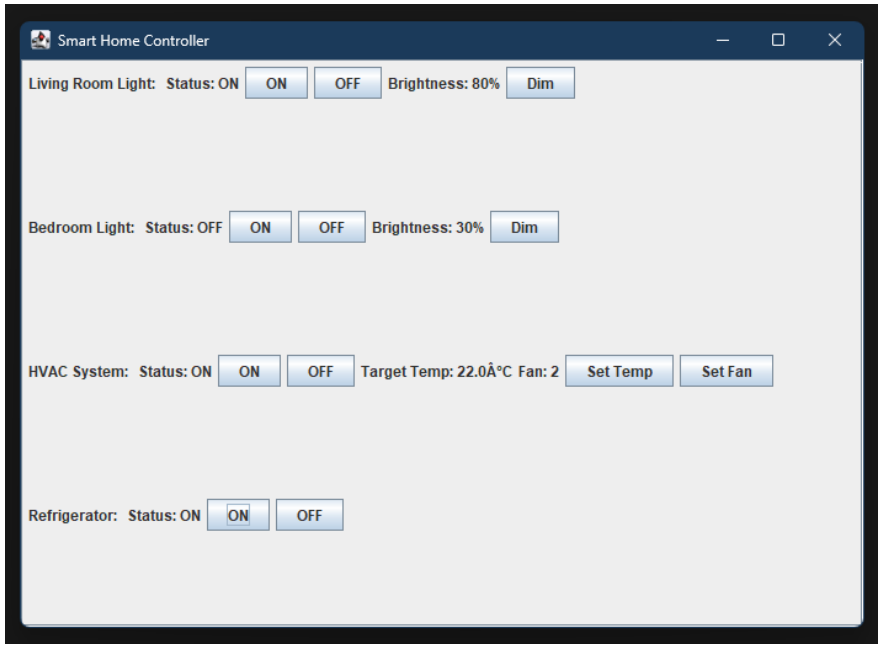
- This dialog enables users to adjust the desired temperature for the HVAC system, demonstrating fine-grained control over environmental conditions.

Setting Fan Speed



- 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



- 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

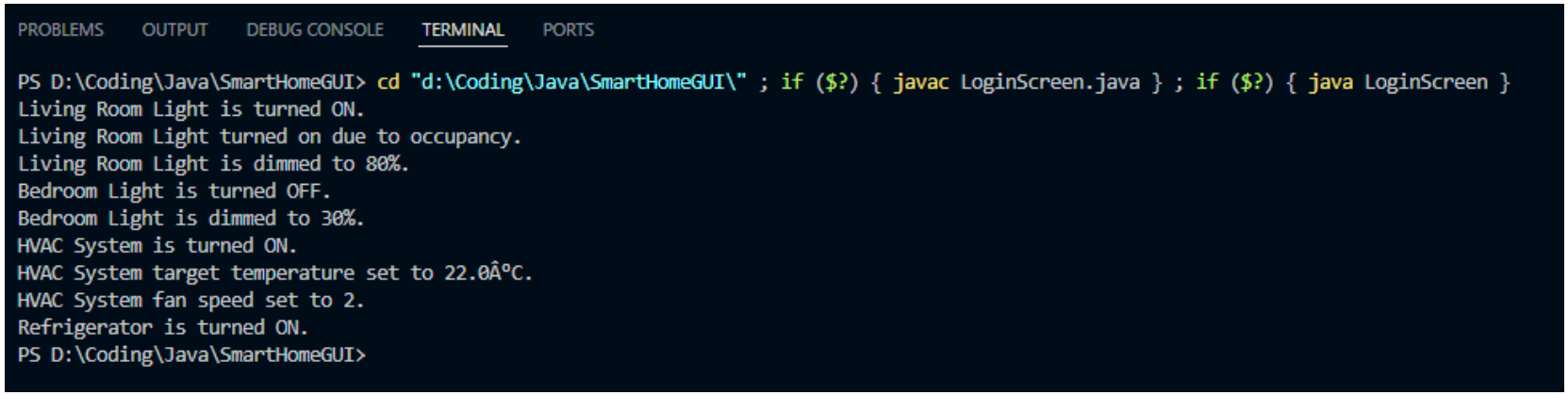


Figure 3: Terminal Output (GUI) of the Eco-Friendly Smart Home Simulator

- 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.

Sample Output

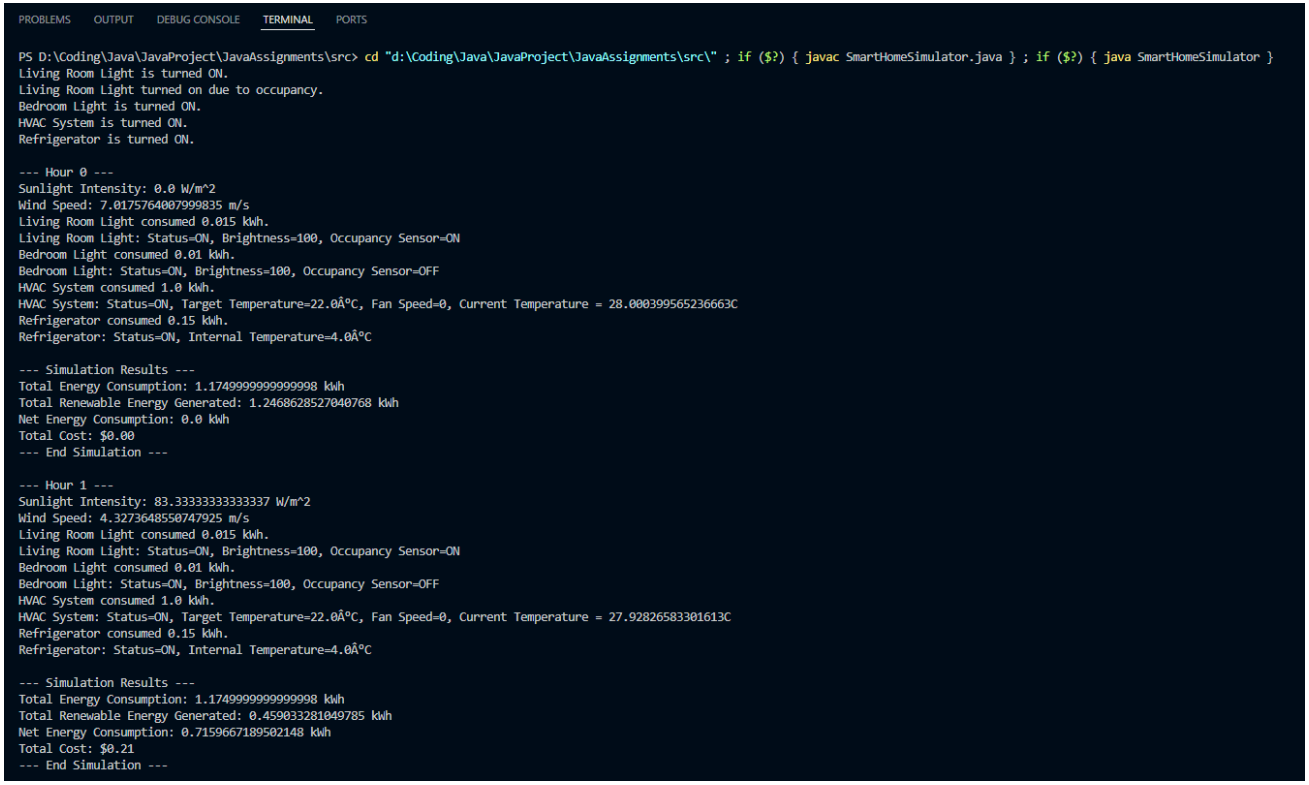


Figure 4: Sample Console Output of the Eco-Friendly Smart Home Simulator

- Even though the main way of using the smart home simulator is now through the graphical interface, the underlying simulation still produces a thorough, text-based log in the console. This log captures every detail, hour by hour. This comprehensive console output serves as a valuable record for in-depth analysis of the simulation's performance.

➢ 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:

This layer forms the brain of the simulator.

- **Simulation Engine:** Manages the main simulation loop, time progression, and event handling.
- **Device Model:** Defines the behavior, attributes, and energy logic for all smart home devices.
- **Environment Model:** Simulates external factors like sunlight and wind, and their impact.
- **Resource Management:** Calculates energy consumption, generation, costs, and environmental impact.

UI Layer:

This layer handles user interaction and display.

- **GUI Components:** Provides the primary interactive graphical interface for device control and real-time status updates.
- **CLI Components:** Used for backend logging and detailed textual output of simulation events.
- **Visualization Components (Future):** Will generate charts and graphs for data analysis.

Data Layer:

This layer manages data storage.

- **Data Persistence (Future):** Will handle saving and loading simulation configurations and results, allowing users to resume or review past simulations.

➢ 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 Behaviors:** Enhancing device realism (e.g., HVAC insulation effects, refrigerator door events, smart light schedules/sensors).
- **Advanced Environmental Modeling:** Integrating seasonal variations, realistic wind patterns, and real-time/historical weather data.
- **Data Persistence:** Implementing save/load functionality for configurations and results (file I/O or database).
- **Reporting and Analysis:** Generating detailed reports and visualizations (charts/graphs) for energy, cost, and environmental impact.
- **Error Handling:** Adding robust error handling for invalid inputs and unexpected states, plus logging for debugging.

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:

Books:

- **"Head First Java" (Sierra & Bates):** Beginner-friendly for Java & OOP basics.
- **"Effective Java" (Bloch):** Advanced Java best practices (e.g., Singleton pattern).
- **"The Internet of Things" (Greengard):** Explores IoT and smart device concepts.
- **"Design Patterns" (Gamma et al.):** "Gang of Four" book covering timeless OOP design patterns (e.g., Singleton, Factory).

Official Documentation & Data:

- **Oracle Java Documentation:** Official Java SE API for built-in features and syntax**. URL:** https://docs.oracle.com/en/java/
- **U.S. Energy Information Administration (EIA):** Real-world energy consumption data for simulator approximations. **URL:** https://www.eia.gov/energyexplained/use-of-energy/
- **International Energy Agency (IEA):** Source for simplified carbon footprint metrics. **URL:** https://www.iea.org/

Online Tutorials & AI Tools:

- **JavaPoint Tutorials:** Free OOP in Java examples. **URL:** https://www.javatpoint.com/java-oops-concepts
- **GeeksforGeeks:** Java Examples: Practical Java code snippets. **URL:** https://www.geeksforgeeks.org/java/
- **W3Schools Java Tutorial:** Free Java tutorials on syntax, OOP, and collections. **URL:** https://www.w3schools.com/java/
- **Google Gemini:** AI tool. **URL:** https://gemini.google.com/share/968f7aae37d2
- **X Grok:** AI tool. **URL:** https://x.com/i/grok/share/i1pbBUaa8kz05cPZu6EOtLWgi
- **Graphviz Online:** Tool for diagram visualization. **URL:** 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 with GUI, including structured into separate Java files for each class:

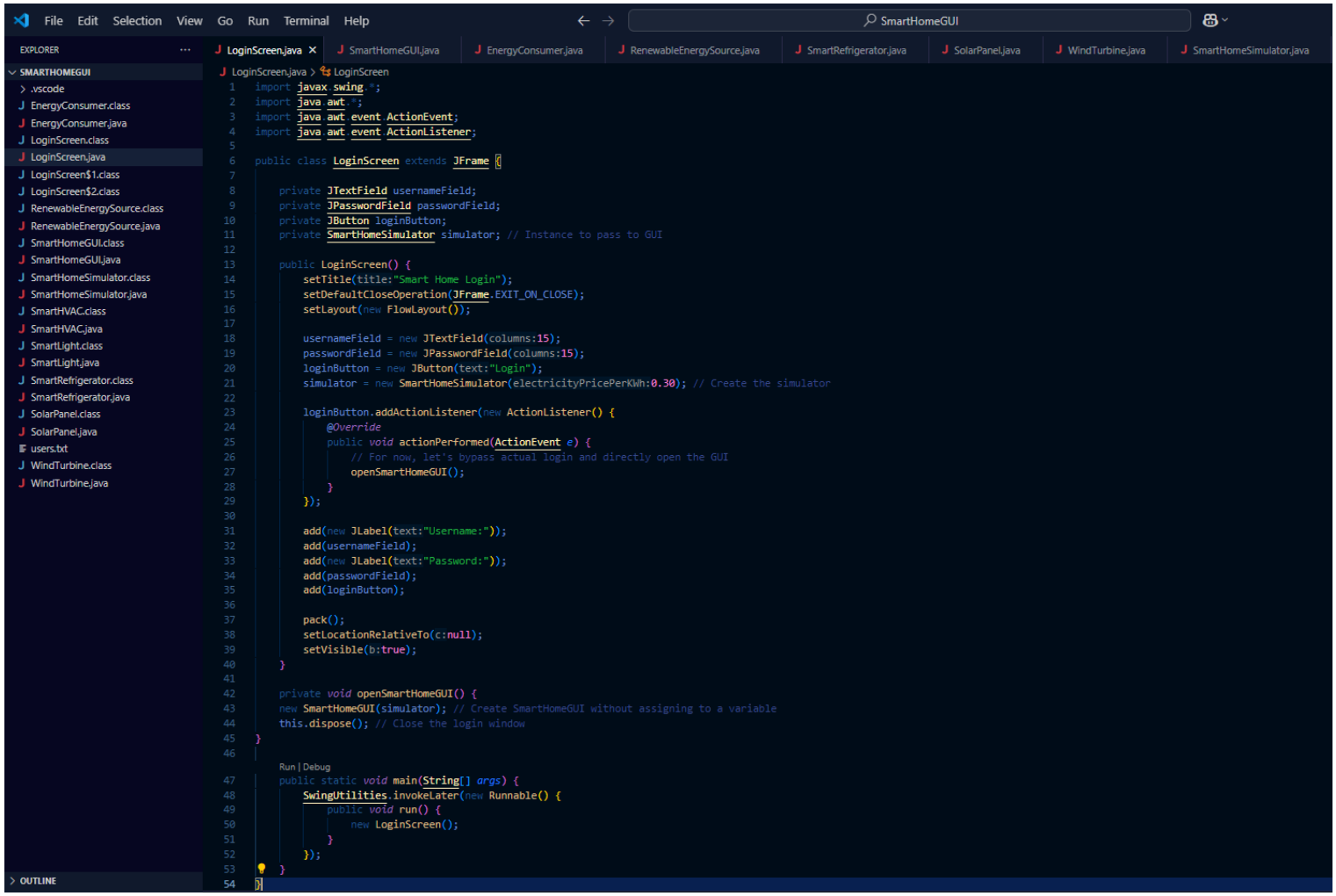


Figure 5: LoginScreen.java

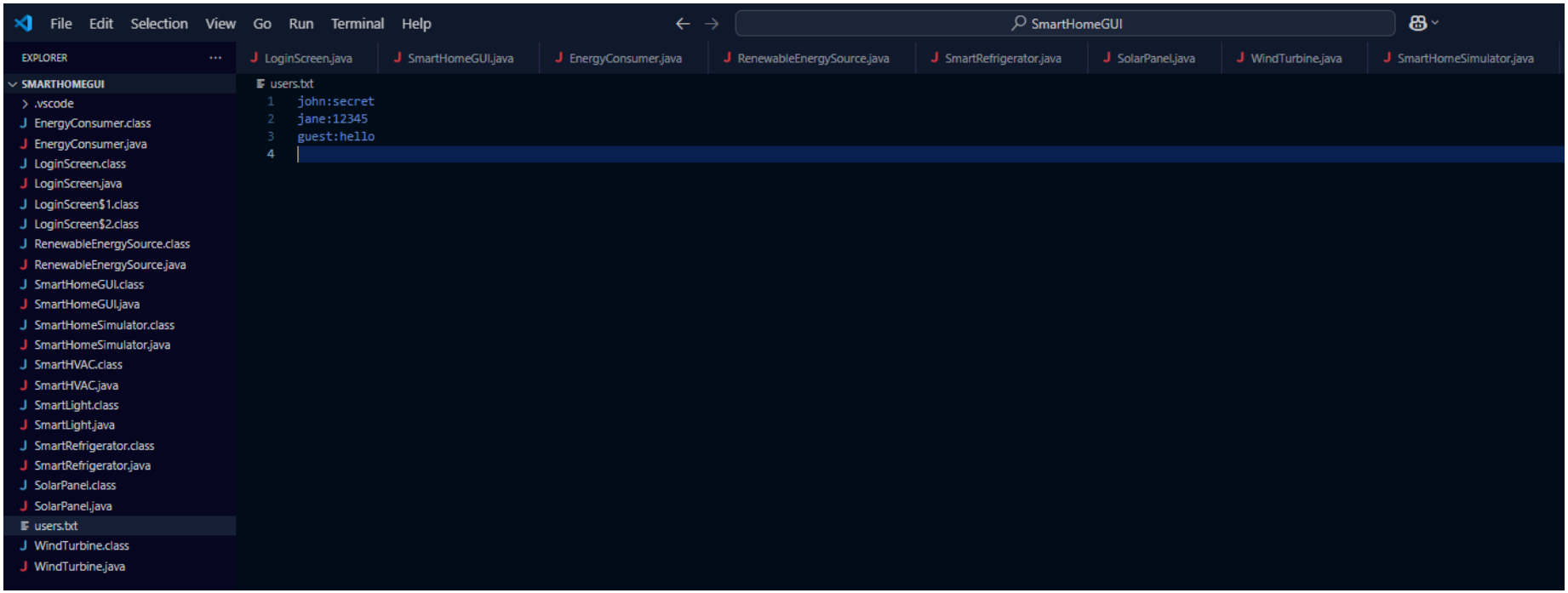


Figure 6: users.txt

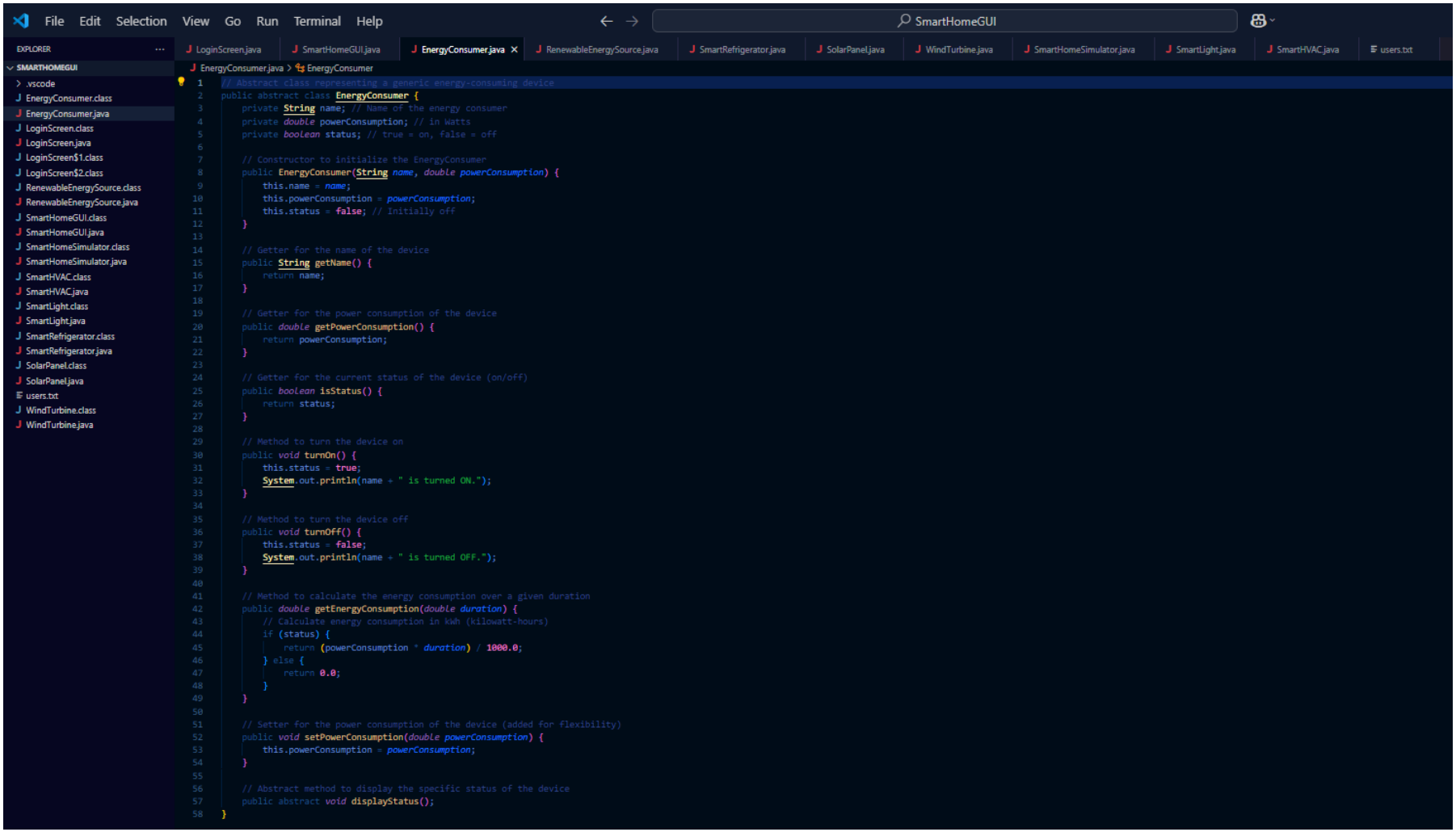


Figure 7: EnergyConsumer.java

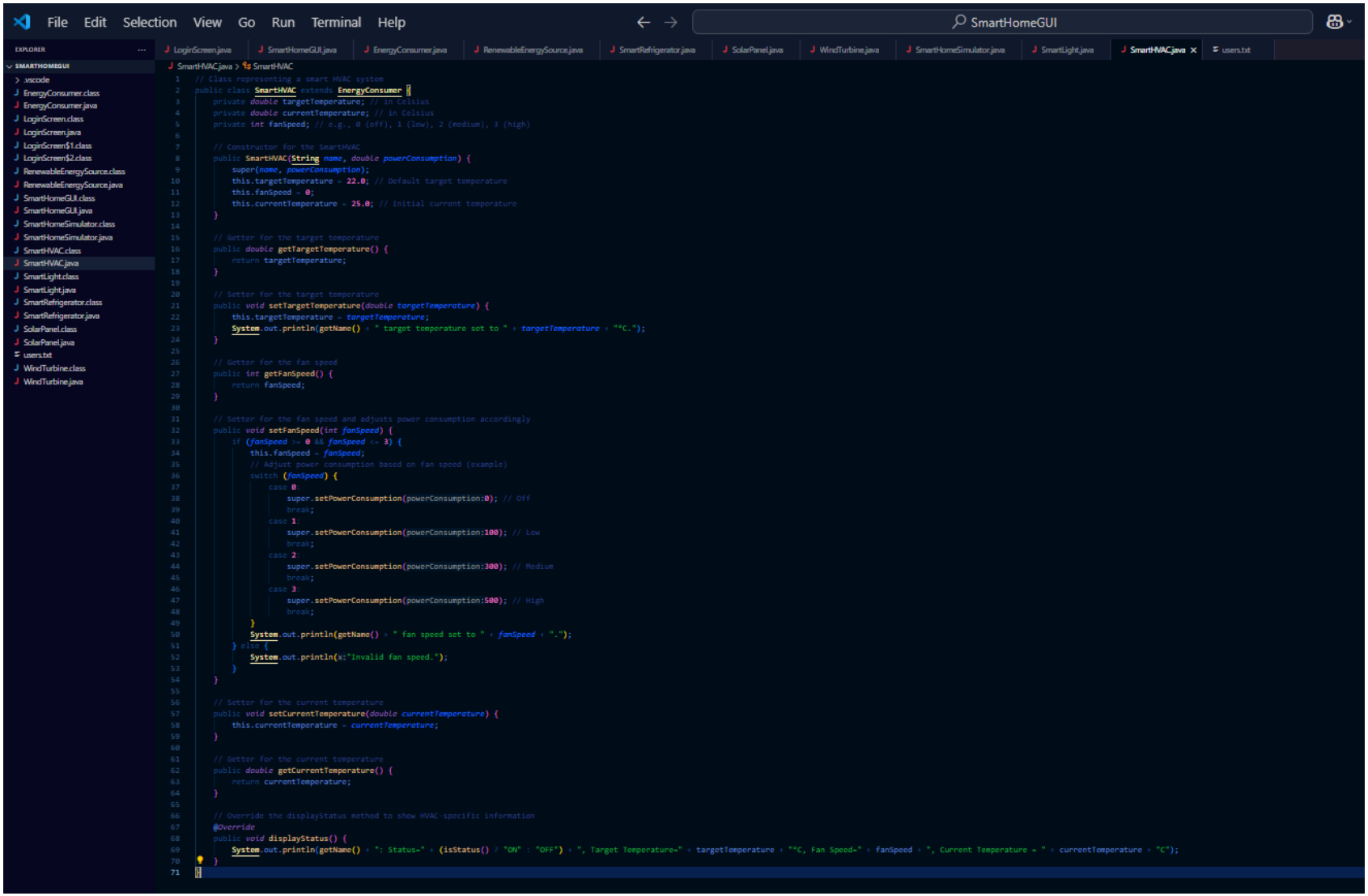


Figure 8: SmartHVAC.java

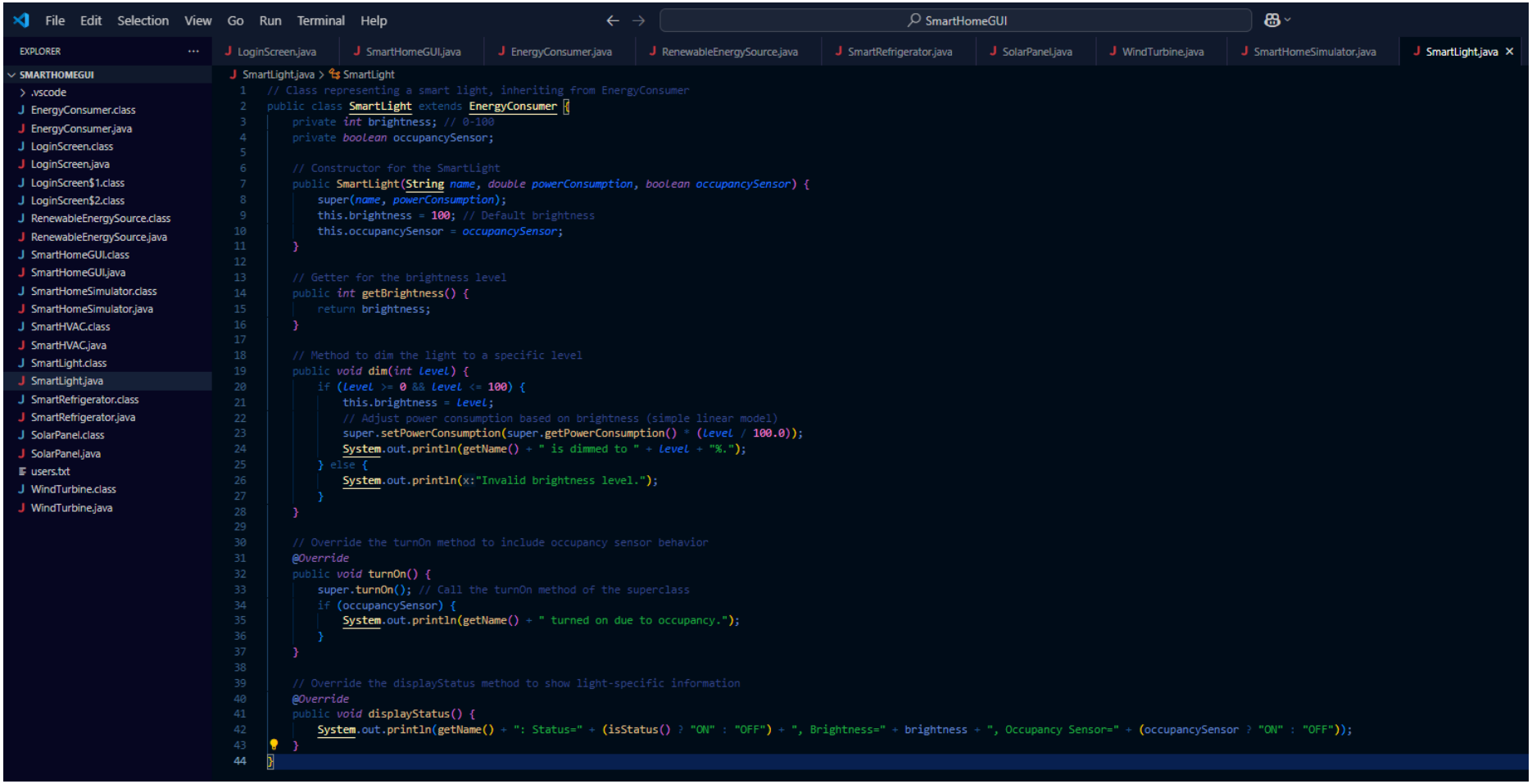


Figure 9: SmartLight.java

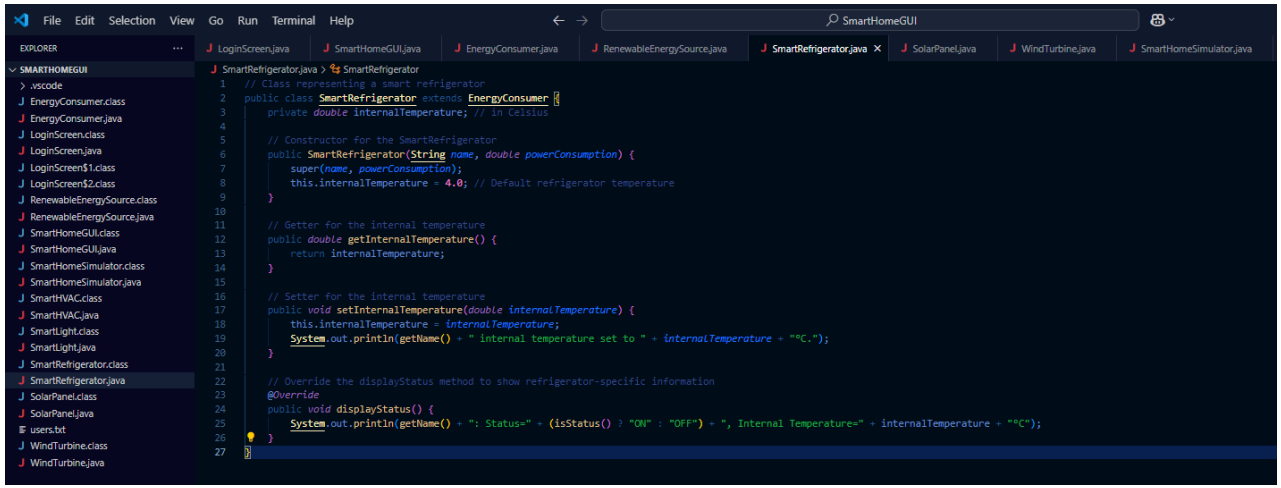


Figure 10: SmartRefrigerator.java

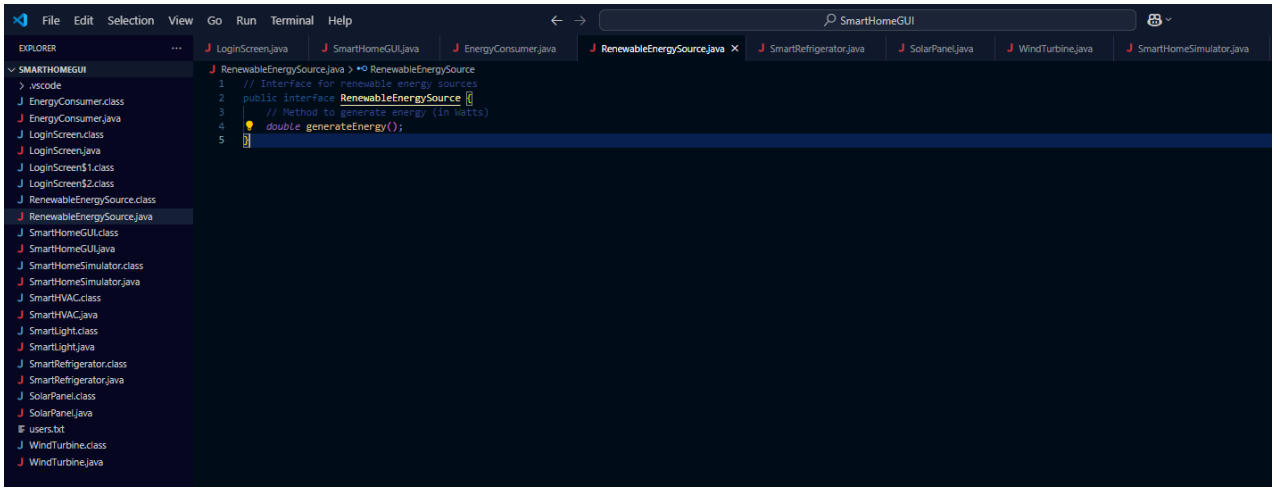


Figure 11: RenewableEnergySource.java

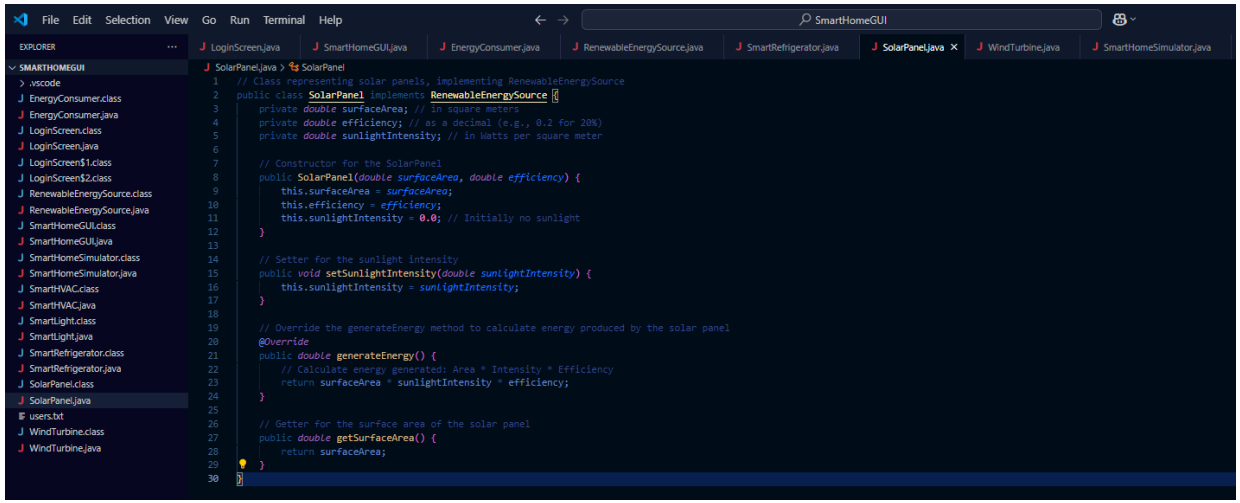


Figure 12: SolarPanel.java

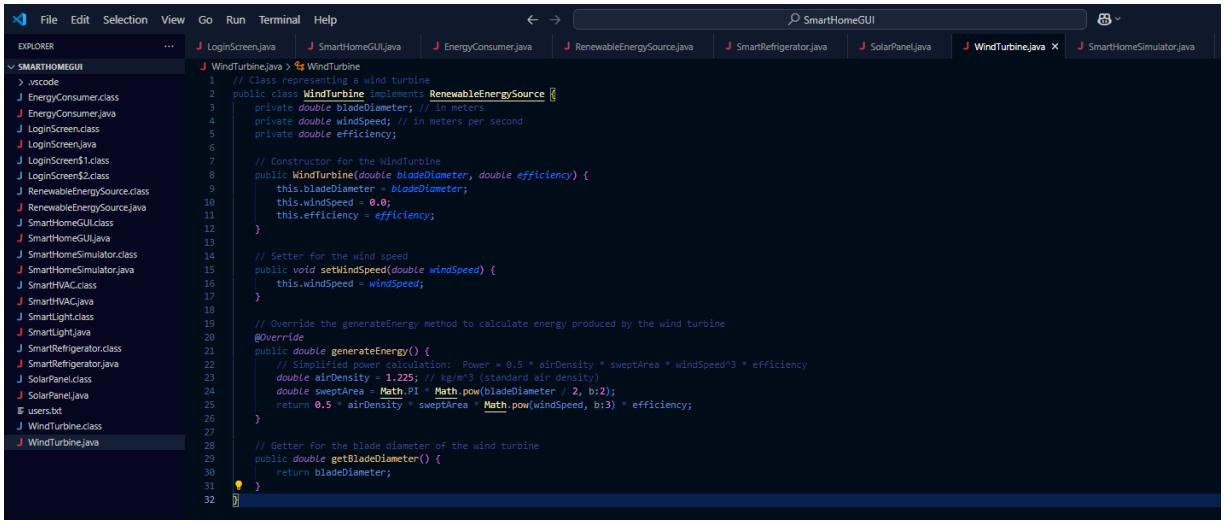


Figure 13: WindTurbine.java

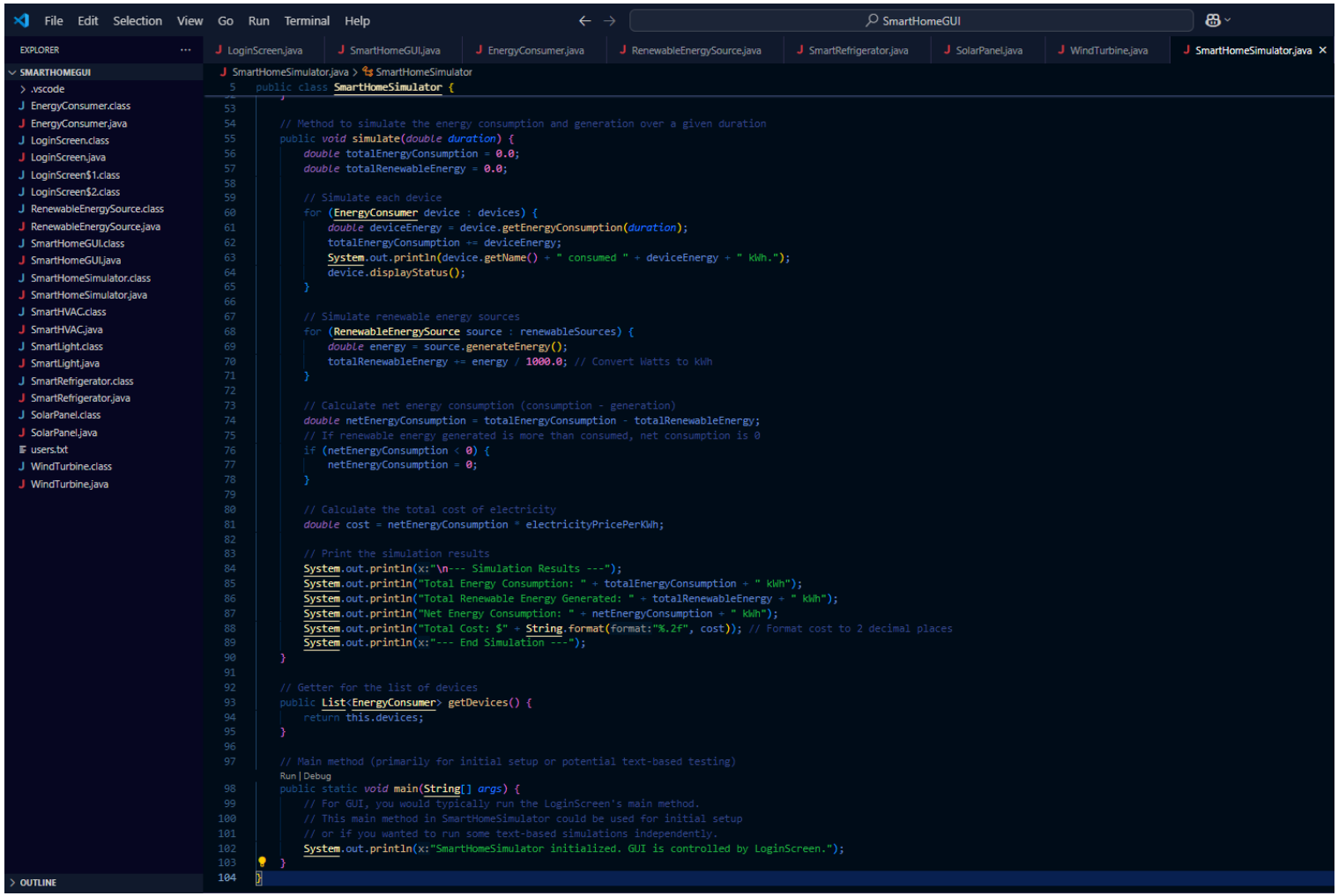


Figure 14: **SmartHomeSimulator.java**

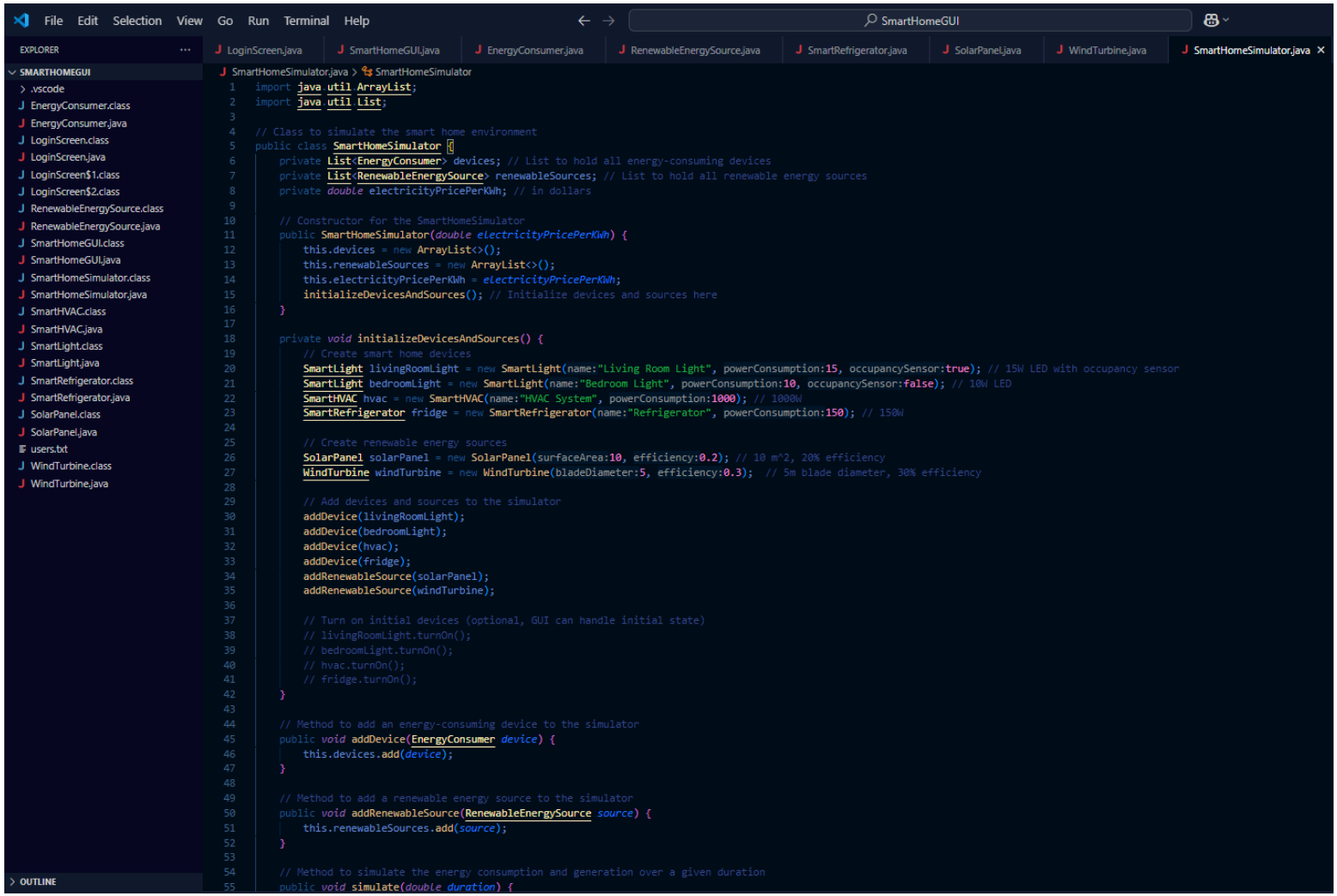


Figure 14: **SmartHomeSimulator.java**

B. 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.

i. 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).

ii. 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).

iii. 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.

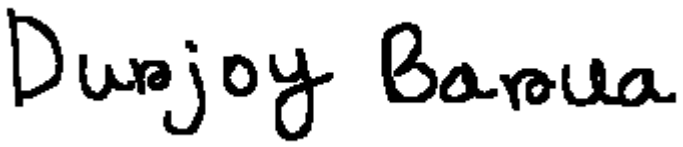
c. Controlling Devices:

- **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.

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

Digital Signature:



Durjoy Barua

Roll: 24070134

Reg: 1185

University of Science & Technology Chittagong (USTC)

Date of Submission: Wednesday, 28 May, 2025
