Smart Home Simulator

(OOPS Project in C++)

Group 12

B.Tech, 2nd Year

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY VADODARA

International Campus - Diu

October 2025

Contents

Overview

- Modern Smart Home concept and scope.
- Simulator implemented using C++ and OOP principles.
- Dynamic device creation, runtime control, persistent storage.
- Focus on: Inheritance, Polymorphism, Encapsulation, Abstraction.
- Bringing together different Smart Devices features under a single application.

Project Motivation

- Real-world relevance: Internet of Things & Smart Homes
- Learn OOP by simulating realistic device interactions
- Move beyond simple management projects — architectural focus
- Demonstrate modular, extensible design suitable for extension

Core Features

- Add, remove, update, and control devices at runtime
- Group devices by type and by room (location)
- Persistent file-based storage (load / save on startup/exit)
- Menu-driven console interface for quick demos
- Easy extension: new device categories and device-specific props

Comprehensive OOPs Concepts Used

1. **Classes & Inheritance:**
   Appliance, Sensors, Doors, Vehicle, DeviceGroup, SmartHome; each parent class with multiple child classes for real-world modeling.
2. **Polymorphism:**
   Virtual functions (`turnOn()`, `turnOff()`, `status()`), function overriding, and virtual destructors allow uniform handling and dynamic behavior of device types.
3. **Encapsulation:**
   Private/protected attributes, public getters/setters ensure safe data access (e.g. `name`, `isOn`, `currentReading`, `isLocked`).
4. **Friend Classes & Functions:**
   Used for Dishwasher and WashingMachine, also Bicycle and Scooter; enables controlled access to protected members.
5. **Constructors & Destructors:**
   Parameterized constructors and virtual destructors for safe memory management.

Comprehensive OOPs Concepts Used - 2

1. **Inline Functions:**
   Short, frequent methods like `status()`, `lock()`, `unlock()` defined within classes.
2. **Function Overloading:**
   Methods with multiple signatures: `turnOn()`, `turnOn(string mode)`, `unlock()`, `unlock(string key)`.
3. **Operator Overloading:**
   For extra credit, can combine device properties (e.g., total energy).
4. **Dynamic Allocation:**
   Devices allocated on heap using `new`, managed as pointers for flexibility.
5. `this` **Pointer:**
   Used in member functions for attribute access and clarity.
6. **File Handling / Persistence:**
   Device state saved/loaded using `saveToFile()` / `loadFromFile()`.
7. **Energy Tracking:**
   Map of device energy usage tracked/updated in `SmartHome`.

Comprehensive OOPs Concepts Used - 3

1. **Scheduling:**
   Device operations set by time using a schedule vector.
2. **Automation Rules:**
   Sensor-driven automation and rules handled programmatically.
3. **Device Groups / Zones:**
   Logical grouping for bulk actions (`turnOnAll()`, `turnOffAll()`).
4. **CLI / Interactive Menu:**
   Menu-driven operations for all device actions and state queries.
5. **Randomized Sensor Data:**
   Sensors simulate variable readings for dynamic automation.
6. **Advanced OOP Patterns:**
   Multiple inheritance, friend usage, deep encapsulation, and vector-of-pointer management.

Class Hierarchy Blueprint

- Our project features a clear, multi-level class hierarchy utilizing single and multi-level inheritance to organize and manage diverse smart devices in C++.
- Concrete Base Classes: `Appliance`, `Sensor`, `Door`, `Vehicle`
- The devices belonging to each class shall be inherited from its base class.

Sample structure (high-level)

- Appliance → Light, Fan, AC, Refrigerator
- Sensor → Temperature, Humidity, Motion
- Door → SmartDoor, GarageDoor
- Vehicle → Car, Scooter

Custom Device Categories

- Entertainment: TV, Speakers (Examples)
- Pet: Feeders, Cameras (Examples)
- Security: Cameras, Smart Locks (Examples)
- Extensible: can add new device classes with minimal changes
- Freedom for user to add new devices not already mentioned in the simulator.

Room-Based Grouping

- Devices assigned to rooms: Bedroom, Kitchen, Living Room, Garage
- Efficient controls: operate whole room (e.g., "Turn OFF Kitchen")
- Simple FUNCTION CALLS to move devices between rooms or rename
- Freedom to create 'n' rooms, with any names and grouping.

File Handling: Persistent Data

- Devices serialized to file (text file)
- On startup: load file and reconstruct via factory
- Each class implements its own save/load logic

Example:

```
DeviceType:Light
Name:CeilingLamp
Room:LivingRoom
IsOn:1
Brightness:80
---
```

FileManager Class

- Centralized save/load/filter/query by room/type
- Keeps file IO isolated from logic
- Example: saveAll(), loadAll(), queryByRoom(), removeDevice()

Menu-Driven Interface

- Console menu to add / remove / view / control devices
- Control all devices in a room (ON/OFF)
- Display filters: by room, type, or status
- Easy to extend to GUI or IoT integration
- Rename or move devices between rooms
- Device-specific properties (brightness, speed, temperature)
- Advanced filtering and search by attributes
- Scheduling and automation rules

OOP Benefits Realized

- Clean, maintainable, modular codebase
- Extensible design for future features
- Demonstrates core OOP pillars effectively
- Scalable for more complex smart automation

Possible Extensions

- Scheduling / automation (IF-THEN logic)
- Security and access control
- External API integration (MQTT / REST)
- GUI or Web Dashboard (future scope)

Project Summary

- Smart Home Simulator models a connected home using C++ OOP
- Persistent, extensible, and realistic implementation
- Demonstrates strong OOP, design, and modularity skills

Questions?

Thank you!

Questions or feedback?