Hacettepe University

Project Assignment 2: Smart Home System

İbrahim Emek 2210356032

Instructor: Görkem Akyıldız

b2210356032@cs.hacettepe.edu.tr

Problem :

The problem is to implement a system for controlling smart home accessories, including Smart Lamp (with white-ambiance and color-white-ambiance variants), Smart Plug, and Smart Camera, based on commands received from the user. The system must adhere to the principles of Object-Oriented Programming (OOP) and the four pillars of OOP: Inheritance, Polymorphism, Abstraction, and Encapsulation. The devices must be sorted based on their switch times in ascending order, with the exception of devices with the same switch time, which must maintain their initial order relative to each other. The system must also handle exceptional situations, such as devices without switch times. The commands received from the user must be validated for errors, such as values out of range or non-integer values, and appropriate error messages must be displayed. The initial time for the system must be set using a specific command format.

My solution:

I used OOP to solve the problem. I used classes, I created objects, and I used them. In the uml part, I talked more about my solution.

Problems I faced and solutions:

I used Calendar in the beginning, but it didn't help me. Only it helped me in finding the maximum day of a month. It didn't add time, I couldn't find the minutes between two Calendar, it was hard to reach, and use the Calendar, so I deleted the part I used calendar. I only kept finding maximum day part. I didn't go step by step, so when I printed zreport first time, almost nothing was right, so I restarted the project. I used static parts a lot in my first project, but I think using static this much is not appropriate to OOP, I changed them in my next try. I only used them if the method is not about a specific object.

Benefits of this system:

Convenience: The system offers an easy and efficient way to control multiple smart home accessories through commands and instructions, saving time and effort for users.

Flexibility: The system is designed to accommodate various types of smart home devices, such as Smart Lamps, Smart Plugs, and Smart Cameras, allowing for easy expansion or modification as needed. Customization: Users can customize the lighting settings of Smart Lamps, including white ambiance and color-white ambiance, to suit their preferences and mood, adding a personalized touch to their smart home experience.

Organization: The system organizes devices based on their switch times, helping users keep track of schedules and switch times in an organized manner for efficient device management.

Error Handling: The system includes error validation to detect and handle issues such as out-of-range values or non-integer inputs in user commands, enhancing reliability and robustness by preventing incorrect commands from being executed and providing appropriate error messages.

OOP Principles: The system adheres to Object-Oriented Programming (OOP) principles like Inheritance, Polymorphism, Abstraction, and Encapsulation, making it modular, maintainable, and extensible for improved code reusability, reduced duplication, and easier maintenance and enhancements.

Overall, the system offers convenience, flexibility, customization, organization, error handling, and follows OOP principles, making it a valuable solution for effectively controlling smart home accessories.

Benefits of this OOP in this system:

Modularity: OOP allows for the system to be divided into separate and independent objects, such as Smart Lamps, Smart Plugs, and Smart Cameras, that can be easily maintained, modified, and tested individually without impacting the entire system. This promotes organized code and promotes reusability, making it simpler to manage and update the system over time.

Extensibility: OOP enables the addition of new features or devices to the system without having to modify the existing code. New objects can be added through inheritance or polymorphism, without disrupting the existing functionality of the system. This makes it easier to expand the system with new devices or functionalities in the future, without making major changes to the code.

Polymorphism: OOP allows objects of different types to be treated interchangeably through polymorphism. For instance, different types of Smart Lamps with varying features can be handled using a common interface or base class. This promotes flexibility and scalability, allowing for seamless integration of new devices or variations of existing devices in the system.

Abstraction: OOP enables abstraction, where complex functionalities can be encapsulated into classes or objects, hiding the internal details and exposing only the essential features to the users of the system. This promotes a simplified and user-friendly system interface, reducing complexity and enhancing usability.

Encapsulation: OOP allows for encapsulation, where data and methods related to an object are encapsulated within the object, and access to them is controlled through well-defined interfaces. This promotes data privacy and security, as well as maintainability of the code, by preventing unauthorized access and minimizing dependencies between objects.

Code Reusability: OOP promotes code reusability through inheritance and composition, where common functionalities can be inherited or composed into multiple objects. This reduces code duplication, promotes consistency, and makes it easier to maintain and update the system. Overall, using OOP principles in the smart home control system enhances modularity, extensibility, polymorphism, abstraction, encapsulation, and code reusability, resulting in a more robust, scalable, and maintainable system

Four Pillars of OOP:

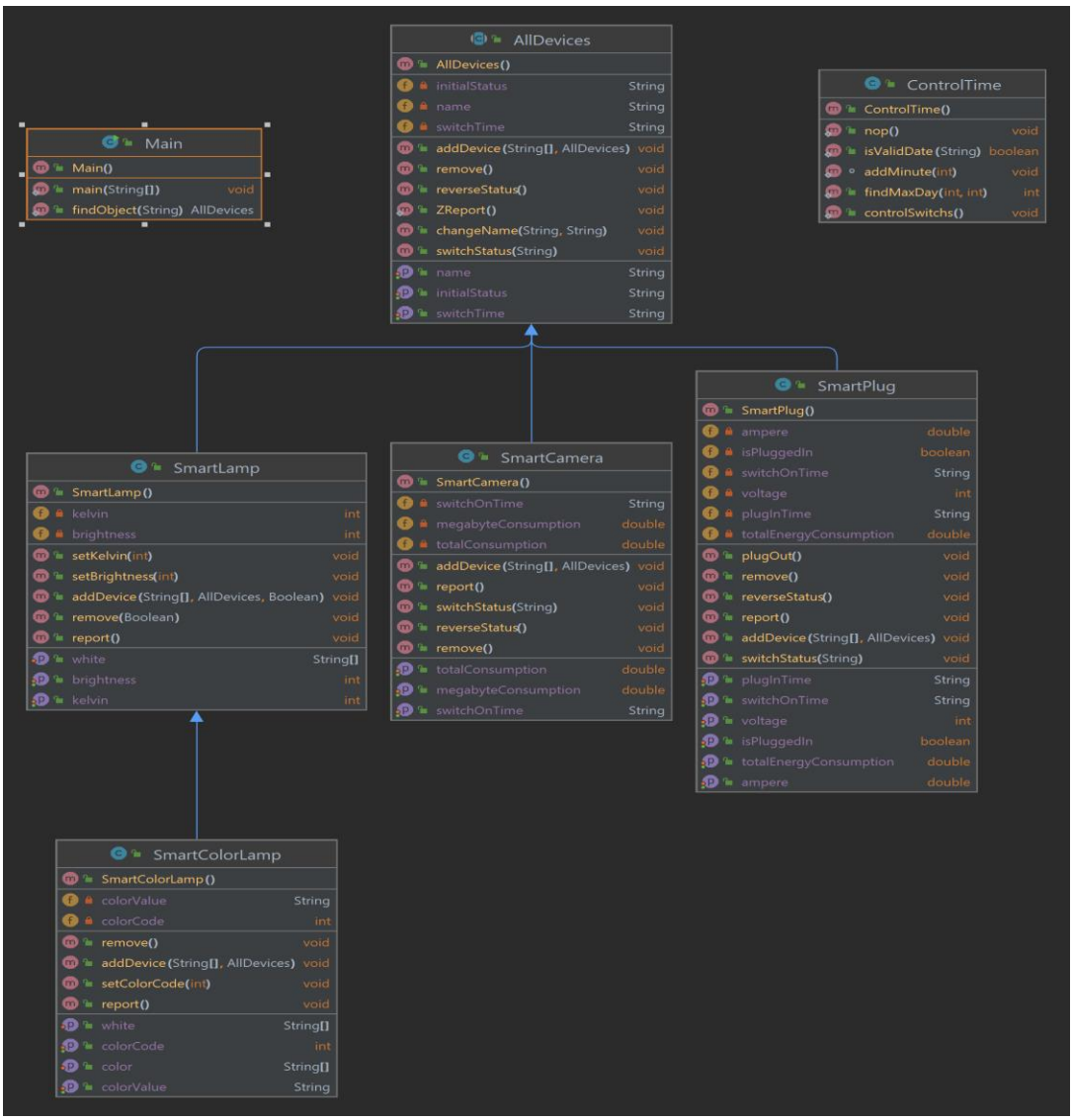
Encapsulation: Bundling data and methods into a class or object, while hiding internal details and exposing essential features. Promotes data privacy, security, and maintainability.

Inheritance: Ability of a class to inherit properties and methods from its parent class. Facilitates code reusability and customization through hierarchical relationships between classes.

Polymorphism: Ability of objects of different classes to be treated interchangeably, based on a common interface. Promotes flexibility and scalability in the system.

Abstraction: Simplifying complex systems by encapsulating essential features into abstract classes, interfaces, or methods. Promotes code reusability and reduces complexity.

UML (Unified Modeling Language): Standardized graphical notation for representing the design and structure of object-oriented systems. Provides visual representation for better understanding and communication among stakeholders. Includes various types of diagrams for modeling different aspects of OOP systems.



AllDevices is superclass of other classes which indicates smart devices. In this class I defined common things such as name, switchTime, initialStatus, and in order to store all devices' name, I used static string list. I used methods to reach the fields. Also, I used methods which affect all smart device. In other classes' which indicates smart devices, I used fields according to their feature, and according to commands, I added methods. In the ControlTime class I controlled time. I added time, I changed time, I checked if

I should switch status of smart devices according to time, and I checked if the date is valid. In the main part, I checked the type of object, and I called the other classes according to type.