**Hacettepe University**

**Department of Computer Engineering**

# BBM 104 - Assignment 2 Report

**Ömer Faruk Güler - 2210356084**

**19.04.2023**

## Index

## Problem

The problem in this assignment is to develop a program that controls and manages a set of smart home accessories, namely Smart Lamp, Smart Lamp with Color, Smart Plug, and Smart Camera. Appropriate commands must be provided to control the devices, such as setting the initial time, adding and removing devices, and setting switch time to these devices, etc. The devices must be sorted in ascending order according to their switch times.The order of these devices must be updated after each switch operation, not after a certain time interval. When the z report command is given, these devices should be printed with their information respectively.

## Solution Approach

Four devices are present in this assignment, each with their own special and general features. Various operations will be performed with these features, so the use of OOP(object-oriented programming) will be beneficial. For example, an abstract class named “Smart Device” will be created and it will be inherited by 4 devices, since they all have general properties such as their names and they can all be defined as home devices. To enumerate devices and apply various actions to them, they must first be accessed. The type List<SmartDevice> was used to hold a list of objects of type SmartDevice. Next, a new ArrayList object was created and assigned to a variable named all_devices to hold all objects of type SmartDevice. (ArrayList is a class that implements the List interface in Java, and it provides an implementation of a dynamic array.) This variable includes all Smart cameras, lamps, and plugs, making it a comprehensive list of all the SmartDevices I want to work with. Therefore, the type of the variable all_devices is List<SmartDevice>. Moreover, it is necessary to implement stable sort in this assignment. The relative order of equal elements in the input sequence is preserved by a stable sort, which is a sorting algorithm. In other words, if two elements in the input sequence are equal, then their relative order in the output sequence will be the same as their relative order in the input sequence.

## Challenges and Solutions

While doing this homework, one of the two parts I spent the most time on was calculating the device's on time and executing it with the set and skip commands respectively, and the other was to sort the devices with the same switch time by preserving their order when the set or skip time command comes. To solve the first issue, a variable called switchOnTime was created to keep track of when a device is switched on. When the device is switched off, the duration between the switchOnTime and the time the device was switched off is calculated. After the duration is calculated, it is added to the activeTime variable, which tracks the amount of time that the device has been active. Therefore, the switchOnTime variable is used to calculate the duration of the device's activity, which is then added to the activeTime variable. It is more complicated to get active time in some places(set/skip times and plug in/out) but same logic.
The solution to the other problem was achieved by writing only 6 lines of code, despite the fact that it took hours of thinking to come up with those 6 lines. That code block is designed to

group devices in a list based on their switchTime attribute, while maintaining a specific order. The code first checks whether a device's switchTime attribute is equal to the switchTime attribute of the next device in the list. If so, the code groups the two devices together, and then iterates through the rest of the list to group any additional devices that have the same switchTime property. Once all devices with the same switchTime property have been grouped together, the code moves on to the next device in the list that has a different switchTime property. This ensures that the order of the groups is maintained, with devices grouped together in the order in which they appear in the list. Once the devices with the same switchTime property have been grouped together, a new list called devicesWithSameSwitchTime is created by the code, which contains all of these devices. Afterward, the list is iterated through by the code using a for-each loop, and a method is called for setting or skipping time for that device. Thus, the desired operation is performed.

## Benefits of the System

1. **Energy Efficiency**: The system allows for efficient use of energy by providing features such as energy consumption tracking and remote switching on/off of devices. This can help users save energy and reduce their electricity bills.
2. **Time Saving**: The system saves time for users by automating tasks such as switching on/off devices and adjusting their settings. It makes human life more effortless and comfortable.
3. **Customization**: The system allows for customization of device settings such as brightness, kelvin value, and color code. This allows users to customize their homes to suit the styles they want.
4. **Organization**: The system organizes devices in ascending order according to their switch times, which helps users keep track of their device usage and ensure that they are not wasting energy by leaving devices on unnecessarily.

## Benefits of OOP

OOP has several advantages for the task of implementing a smart home system. I want to mention three main concepts that are very useful in many ways.

OOP allows for encapsulation of data and behavior into objects, which can help to reduce complexity. In this scenario, each smart home accessory can be encapsulated into an object with its own data (such as switch time, kelvin, brightness, color code, etc.) and behavior (such as turning on/off, adjusting kelvin and brightness values, etc.). Encapsulation allows for better organization of code and easier modification of specific functionality without affecting other parts of the code.

Also, OOP allows for inheritance, where objects can inherit properties and methods from a parent object. For example, there is a Smart Lamp and a Smart Lamp with Color. Smart Lamp with Color can inherit Smart Lamp because it contains all the features of Smart Lamp and has a color as an extra.

Finally, OOP allows for polymorphism, where objects can take on multiple forms and behave differently. For example, a smart lamp with color has different settings and color options that can be adjusted when it is turned on or off. Polymorphism allows for code that is more flexible and easier to extend.

## Four Pillars of OOP

1) **Encapsulation**: Encapsulation is used to control access to data. A class can contain variables to store data and methods to act on that data. However, providing direct access to this data can cause problems with the inner workings of a class.

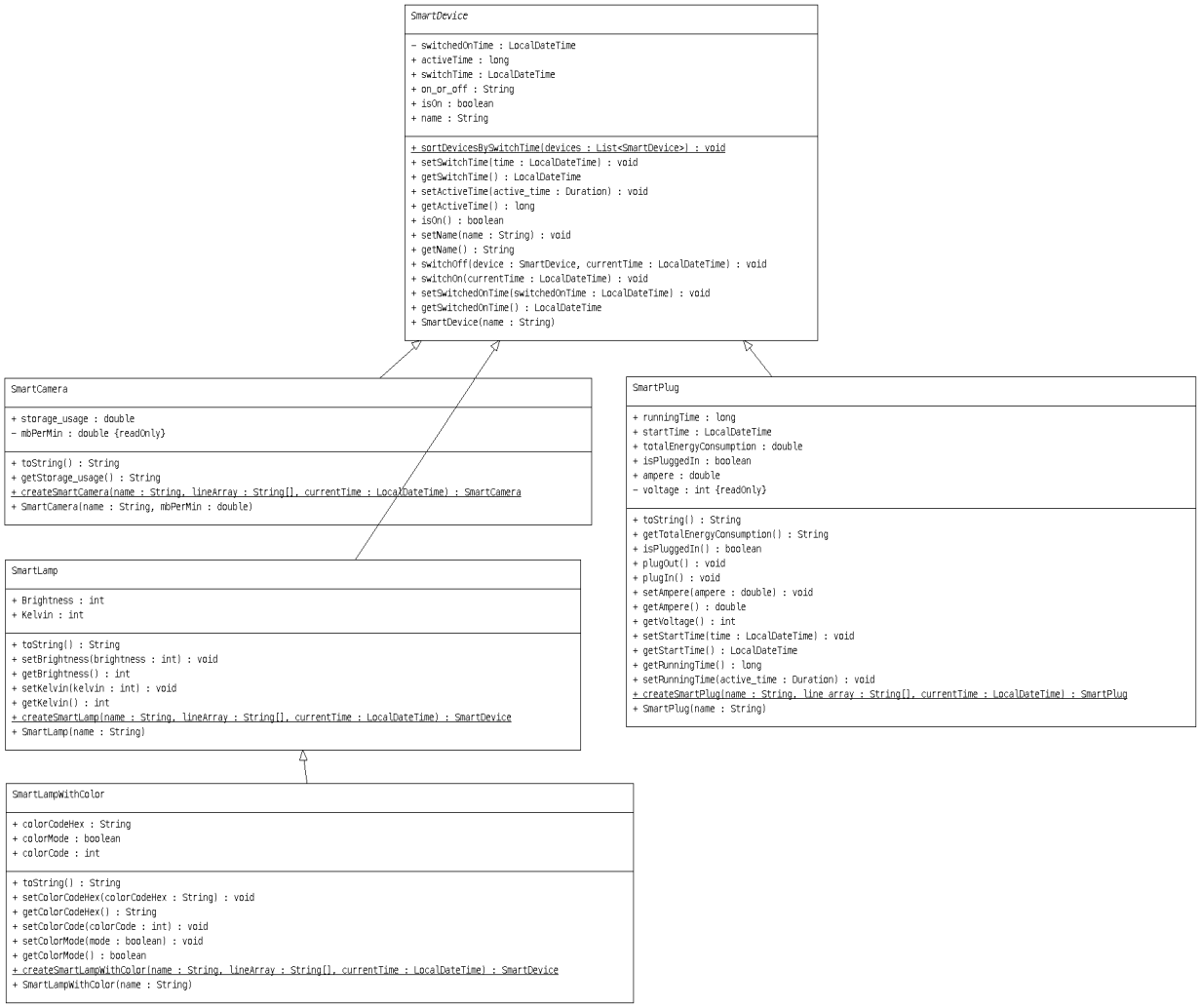
2) **Inheritance**: Inheritance is the process of creating a new class from an existing class by inheriting the properties and behaviors of it. Inheritance helps to reduce code duplication and make the code more readable.

3) **Polymorphism**: Polymorphism is the ability of an object to behave in different ways. We can produce different results from a method when we call it with a reference of different subclasses of an object. This feature is achieved by the ability of subclasses to override the same method defined by the superclass. Polymorphism provides flexibility and extensibility to the code.

4) **Abstraction**: Abstraction is a programming principle that involves exposing necessary information while hiding unnecessary details. It is important for both security and design purposes, as it helps to reduce complexity and make it easier to manage and maintain code. Complex details can be abstracted away, allowing us to focus on the essential components of a system and making it easier to read and modify in the future.

## UML

UML diagrams are illustrations of the design of the code. Since people tend to draw and understand the graphics they draw, these diagrams make the code easy to understand.




As can be seen from the uml diagram, 5 classes are used. Since the abstract one contains the attributes and methods of all devices, all other classes are its instances. Also, SmartLampWithColor inherits the SmartLamp because it is also a SmartLamp. There are attributes in the upper half of the classes shown in the diagram, and methods in the lower half. These allow the properties of the class to be manipulated and accessed. Access modifiers are keywords used to set the accessibility of classes, variables, methods, and constructors. By using access modifiers, the visibility and accessibility of different parts of the class can be controlled. This helps to keep our code organized and maintainable. There are some attributes that are public(+) and private(-) in this UML. They could also be protected or default. The public access modifier is the most permissive and the private access modifier is the most restrictive. It allows access to the class, method, variable, or constructor only within the same class. For example, the voltage value is private because it does not have to be known by the user.