Hacettepe University

Department Of Artificial Intelligence

BBM104 Assignment 2 Report

Creating a Smart Home System

Adam Sattout - b2220765061

23/04/2023

Index:

1 – The problem

Here we have to make a dynamic system that can create various devices in a smart home when the user wants and modify them in multiple ways by commands given beforehand. We need to make a system that can receive these commands, evaluate their correctness, and behave accordingly, this system should also use OOP as a main structure shape for better approach.

2 – Solution approach

In this assignment, main approach was creating an abstract class called SmartDevice that has shared attributes that all devices have like name or switchstate, and inheriting multiple subclasses which are the types of smart devices like SmartPlug, SmartLamp, and SmartCamera that each has private attributes, with another subclass that is inherited from SmartLamp called SmartColorLamp. The program will run through the given input line by line in the main method, the line will be given to be evaluated to another method, if a command is valid the program will execute the command with other helper methods. If not, then the method will throw an exception that is dynamic according to the error type like the variable inserted being not positive in a place that It needs to be positive, or assigning a new name to a device that isn't even created, and this error will be caught back in the main method and will print out a suiting error message to the user.

3 – Problems Faced

Some problems that were faced in the commands having too many versions, for example, adding a smart lamp can include or not its switch state, kelvin value and brightness which makes counting for all the possibilities a bit of a problem. However, this problem was solved by setting the given value if provided, and if not setting it to a default value with ternary operators.

Another problem resembled in have a sheer amount of errors and having to account for all and each one of them, this was also solved after some time by creating different exception classes for each type of exception.

One more problem was in finding the best and most logical design in sense by using OOP, but thankfully the design mentioned previously has been implemented, which is going to be further investigated in the next part.

4 – Benefits of the System

This system makes it really efficient to create variables and behaviors for each device, since we have our variables already declared for each smart device created and we just have to set them to the needed value, rather than creating a new variable for each device and set the variable to the new device, it also creates a sense of formality in the structure of the program since all of the devices will have the same attributes and behaviors. This approach is really more abstract than other static programming languages and can be really beneficial for developers.

5 – OOP and its Implementation in This System

OOP has 4 important pillars that it is based on,

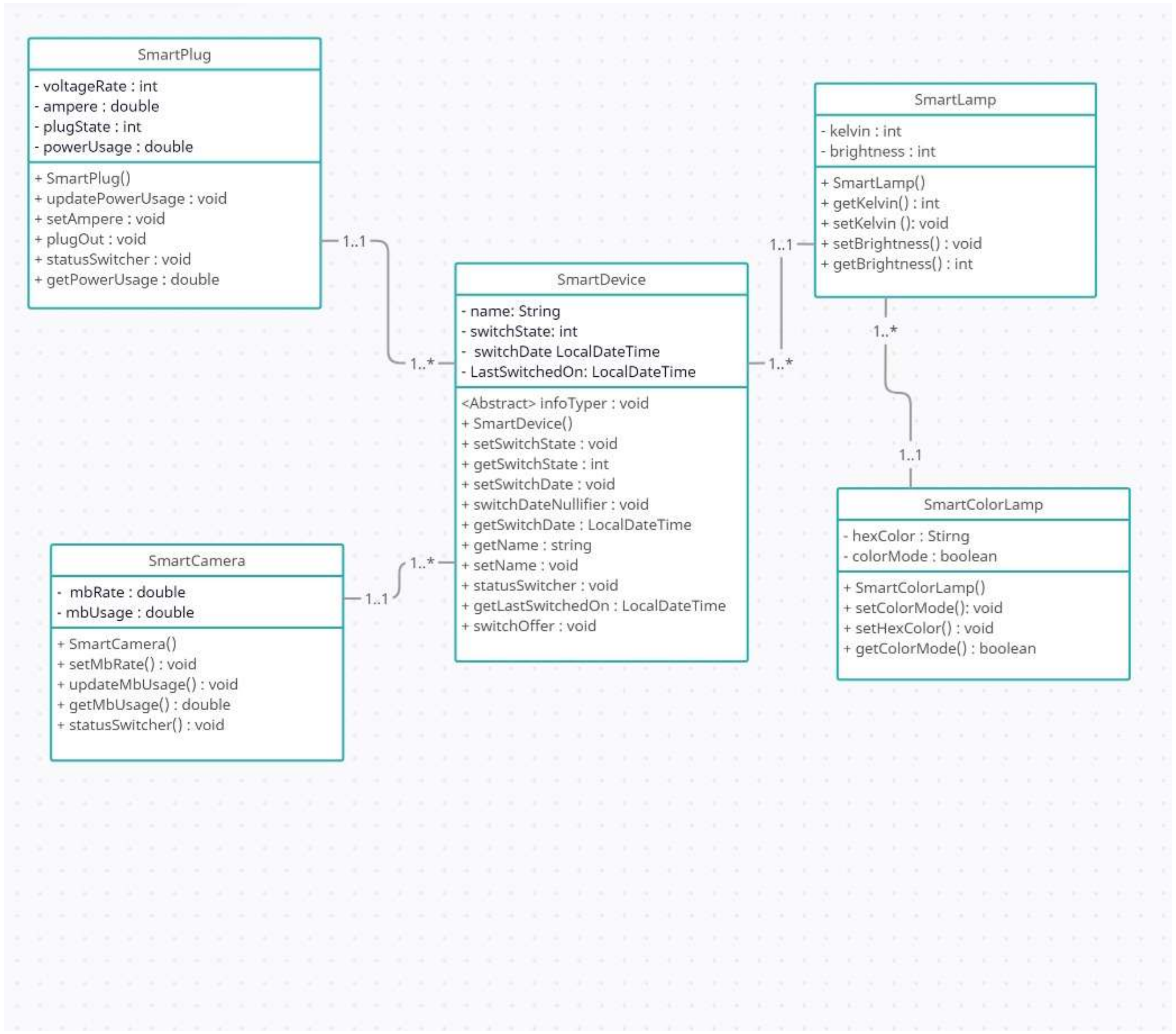
Abstraction : The concept of that all objects in a system will be inherited from one simple object. Like in our program, all types of smart devices are of type smart device, and in fact, our device is as the many different objects in java a modified version of the object type.

Inheritance : It’s the act of creating subclasses that are a modified and more specific version of a superclass, like having a dog be a subclass to the superclass animal, or in our case having a SmartPlug be a subclass for the superclass SmartDevice, where SmartPlug has all the attributes and behaviors of SmartDevice with some of its own.

Encapsulation : This can be known as hiding some specific variables to an object making them private, and showing some to everyone in the program making the public, which can make mistaking a value of an object of a class with another object really hard since they both won't be shown to each other. In the design implemented all attributes of classes were set to private with public behaviors that called getters and setters that access these attributes.

Polymorphism : Which can be in the form of making an object behave in different ways when needed, like casting an animal to behave like a dog and have its attributes and behaviors, or casting a SmartDevice to behave like a subclass of its own like SmartCamera.

Here we can see how OOP was implemented in the system by this UML diagram.


SmartPlug
- voltageRate : int
- ampere : double
- plugState : int
- powerUsage : double
+ SmartPlug()
+ updatePowerUsage : void
+ setAmpere : void
+ plugOut : void
+ statusSwitcher : void
+ getPowerUsage : double
1..1
1..*
SmartLamp
- kelvin : int
- brightness : int
+ SmartLamp()
+ getKelvin() : int
+ setKelvin (): void
+ setBrightness() : void
+ getBrightness() : int
1..1
1..*
SmartDevice
- name: String
- switchState: int
- switchDate LocalDateTime
- LastSwitchedOn: LocalDateTime
<Abstract> infoTyper : void
+ SmartDevice()
+ setSwitchState : void
+ getSwitchState : int
+ setSwitchDate : void
+ switchDateNullifier : void
+ getSwitchDate : LocalDateTime
+ getName : string
+ setName : void
+ statusSwitcher : void
+ getLastSwitchedOn : LocalDateTime
+ switchOffer : void
1..*
1..1
SmartColorLamp
- hexColor : Stirng
- colorMode : boolean
+ SmartColorLamp()
+ setColorMode(): void
+ setHexColor() : void
+ getColorMode() : boolean
SmartCamera
- mbRate : double
- mbUsage : double
+ SmartCamera()
+ setMbRate() : void
+ updateMbUsage() : void
+ getMbUsage() : double
+ statusSwitcher() : void
1..*
1..1