Hacettepe University

Department of Artificial Intelligence Engineering

Project: Smart Home Systems Report

Course Name:

BBM104 – Introduction to Programming Laboratory II

Report Prepared by:

Özge Bülbül

2220765008

April 16, 2023

Table of Contents

1. The Project

1.1 Understanding the Problem

In this project, the main problem was designing a smart home system that involves three main smart home accessories. These accessories have some features that are exclusive to themselves and some that are common with each other. Many commands regarding these devices and their features were present. Apart from these commands and their outputs, we were also expected to take care of various errors and exceptions; regarding the special cases of each command and smart home accessory.

Controlling the smart devices is the base problem. Besides controlling these devices, we were expected to control time flow. Since these devices are autonomous, they may get switched on/off by time and smart plugs and cameras keep account of their energy consumptions and used storage values respectively. We are responsible with keeping record of this process and also making the necessary calculations for plugs and cameras.

1.2 Thinking of a Solution

In the process of designing a solution to the problem, since we are expected to work with java which is an object-oriented language, a UML diagram was designed by me. In this diagram, the links between smart devices are clearly shown by my class hierarchy. By this approach, the devices were each treated like separate objects and dealing with them was quite laborless.

Approaching the time flow, it was necessary to keep record of the first and last time the plugs and cameras were switched on (and plugged in for the plugs) to take the duration between them in order to do the calculations.

1.3 Implementation Process

The way of thinking and approaching the problem was explained in section 1.2. While this approach was being implemented; firstly, three device classes were created: Smart camera, smart lamp and smart plug. Then, in order to have one super class above all, the Smart Devices class was added. The general fields, accessors and mutators that all devices have in common for instance; name, status and their getter-setters were kept by this class. Another device class was created which is the smart color lamp class. The color ambiance variant of smart lamps is represented by this class which is called the smart color lamp. It is a subclass of the smart lamp class and has indigenous features like color code(integer) and color feature(boolean).

Two fields were assigned to smart camera and plug classes: initial time and final time. These were assigned to keep track of their storage and energy consumptions. “Ampere*220*duration(hour)” and “megabytes per record*duration(minute)” equations were used to calculate consumed energy and used storage values respectively. In cases of keeping initial time or updating it, a current time class was created.

Personally, the exceptions were the most challenging part due to the fact that they were quite various and specific. The change name command can be handled to give a brief example. We were expected to check if there were three words to make sure the command was in correct format, if the new name was the same as the old one, if the new name was already taken or if a device of the old name exists; there are specific error messages. Considering that all 17 commands have around 4 error messages, this was not a difficult but a time-consuming section of the problem. While taking care of it, besides the already existing exceptions, four exception classes were created.

1.4 Benefits of This System

The smart home system has many commands and some are specific to one device. With only one command; kelvin, brightness even status of a lamp can be changed. Since every error message is unique, the mistake made by the user can easily be understood. The “ZReport” is especially a very useful and inclusive command. All devices, their types and names, status and switch time, kelvin/brightness/storage/color code etc. features get listed for the user.

There are various benefits of object-oriented programming approach in this project. Firstly, OOP and the use of classes for each device makes this code quite reusable. All the developer needs to do to add a new device is to create that device’s class and make minor modifications to the “Commands” class. Finding any errors that may occur in the code later on can easily be noticed and fixed thanks to this class-separated construction. This is also achieved by modularity which is obtained by OOP. The next benefits are the four pillars of OOP: inheritance, polymorphism, abstraction and encapsulation; which are explained in the next section (2. Object-Oriented Programming).

2. Object-Oriented Programming

2.1 Inheritance

This first pillar of OOP is the relationship between super-classes and subclasses. Subclasses extend their super classes and while obtaining the fields and methods of them, subclasses have specific and private ones.

2.2 Polymorphism

When we have a subclass of another class, the subclass inherits the super class's methods and attributes. If we assign the same method to a different implementation in the subclass, we can use both of them. Polymorphism (many forms) makes us able to use a single action and perform plural acts.

2.3 Abstraction

Abstraction is hiding the implementation code that is unnecessary to the users and only showing the needed, simple section. This is achieved by using abstract classes or interfaces.

2.4 Encapsulation

Encapsulation enables the separation and restriction of the code. We can separate the related code segments such as fields and methods into a class, and hide these from the users.

2.5 UML Diagram

UML diagram is a very useful way of picturizing the code implementation as a whole. In the diagram all classes and their inheritors can be shown. Also, the methods and fields these classes have and if they are private, protected or public (-, #, + respectively) are given to be visually represented. In this project's UML diagram (figure 1), all class hierarchy can be viewed. Static methods are given underlined and also distinguished by the "usesStatistically" writing next to the specific arrow. Finally, it must be mentioned that UML diagrams may include exception classes, too. However, my diagram became quite large and as the diagram gets bigger the image quality drops, therefore; it was chosen by me to leave these out of the diagram.

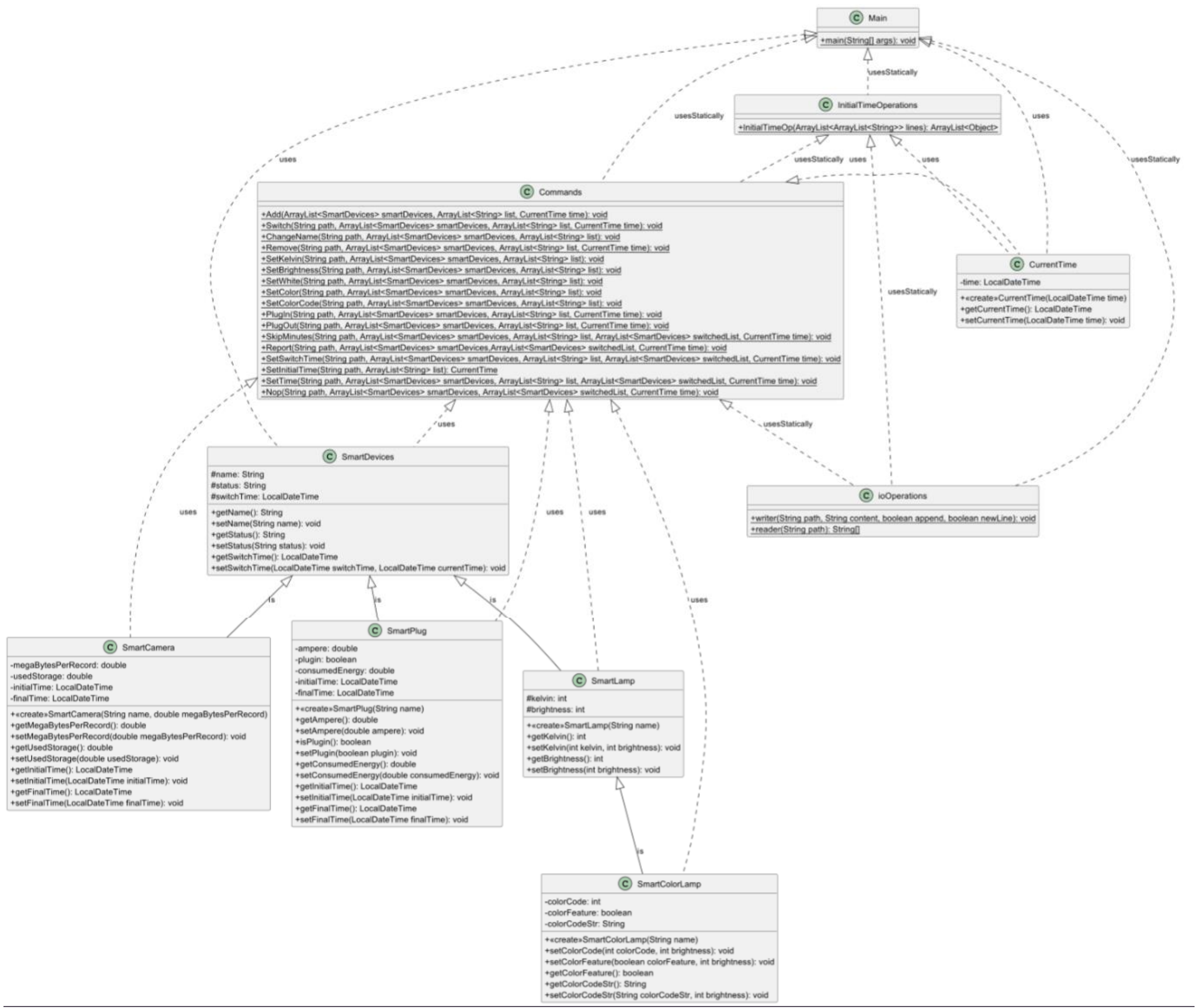
Main
+main(String[] args): void
usesStatically
InitialTimeOperations
+InitialTimeOp(ArrayList<ArrayList<String>> lines): ArrayList<Object>
usesStatically
uses
usesStatically
uses
uses
uses
usesStatically
Commands
+Add(ArrayList<SmartDevices> smartDevices, ArrayList<String> list, CurrentTime time): void
+Switch(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list, CurrentTime time): void
+ChangeName(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list): void
+Remove(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list, CurrentTime time): void
+SetKelvin(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list): void
+SetBrightness(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list): void
+SetWhite(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list): void
+SetColor(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list): void
+SetColorCode(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list): void
+PlugIn(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list, CurrentTime time): void
+PlugOut(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list, CurrentTime time): void
+SkipMinutes(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list, ArrayList<SmartDevices> switchedList, CurrentTime time): void
+Report(String path, ArrayList<SmartDevices> smartDevices,ArrayList<SmartDevices> switchedList, CurrentTime time): void
+SetSwitchTime(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list, ArrayList<SmartDevices> switchedList, CurrentTime time): void
+SetInitialTime(String path, ArrayList<String> list): CurrentTime
+SetTime(String path, ArrayList<SmartDevices> smartDevices, ArrayList<String> list, ArrayList<SmartDevices> switchedList, CurrentTime time): void
+Nop(String path, ArrayList<SmartDevices> smartDevices, ArrayList<SmartDevices> switchedList, CurrentTime time): void
CurrentTime
-time: LocalDateTime
+«create»CurrentTime(LocalDateTime time)
+getCurrentTime(): LocalDateTime
+setCurrentTime(LocalDateTime time): void
usesStatically
uses
usesStatically
ioOperations
+writer(String path, String content, boolean append, boolean newLine): void
+reader(String path): String[]
SmartDevices
#name: String
#status: String
#switchTime: LocalDateTime
+getName(): String
+setName(String name): void
+getStatus(): String
+setStatus(String status): void
+getSwitchTime(): LocalDateTime
+setSwitchTime(LocalDateTime switchTime, LocalDateTime currentTime): void
uses
uses
uses
uses
is
is
is
SmartCamera
-megaBytesPerRecord: double
-usedStorage: double
-initialTime: LocalDateTime
-finalTime: LocalDateTime
+«create»SmartCamera(String name, double megaBytesPerRecord)
+getMegaBytesPerRecord(): double
+setMegaBytesPerRecord(double megaBytesPerRecord): void
+getUsedStorage(): double
+setUsedStorage(double usedStorage): void
+getInitialTime(): LocalDateTime
+setInitialTime(LocalDateTime initialTime): void
+getFinalTime(): LocalDateTime
+setFinalTime(LocalDateTime finalTime): void
SmartPlug
-ampere: double
-plugin: boolean
-consumedEnergy: double
-initialTime: LocalDateTime
-finalTime: LocalDateTime
+«create»SmartPlug(String name)
+getAmpere(): double
+setAmpere(double ampere): void
+isPlugin(): boolean
+setPlugin(boolean plugin): void
+getConsumedEnergy(): double
+setConsumedEnergy(double consumedEnergy): void
+getInitialTime(): LocalDateTime
+setInitialTime(LocalDateTime initialTime): void
+getFinalTime(): LocalDateTime
+setFinalTime(LocalDateTime finalTime): void
SmartLamp
#kelvin: int
#brightness: int
+«create»SmartLamp(String name)
+getKelvin(): int
+setKelvin(int kelvin, int brightness): void
+getBrightness(): int
+setBrightness(int brightness): void
is
SmartColorLamp
-colorCode: int
-colorFeature: boolean
-colorCodeStr: String
+«create»SmartColorLamp(String name)
+setColorCode(int colorCode, int brightness): void
+setColorFeature(boolean colorFeature, int brightness): void
+getColorFeature(): boolean
+getColorCodeStr(): String
+setColorCodeStr(String colorCodeStr, int brightness): void

Fig. 1. Smart Home System UML Diagram