Hacettepe University
Department of Computer Engineering

BBM104 - Assignment 2 Report

Barış Yıldız - 2210356107

A report of the code written for the second assignment of the BBM104 lecture.

Date: 21.04.2023

Contents

1 The Problem **2**
1.1 Definition 2
1.2 Solution 3
1.3 Difficulties 3
1.4 Benefits of the System 3

2 Object Oriented Programming **4**
2.1 Benefits of OOP in this Problem 4
2.2 The Four Pillars of OOP 4

3 UML Diagram of the Program **5**

4 Resources **7**

Chapter 1

The Problem

1.1 Definition

In this assignment, we are asked to design a computer program that simulates a smart home system. A smart home system is a system that consists of smart devices. Smart devices are devices that aim to help and beautify human life.

The following smart devices can be added to the system:

Smart Lamps: Smart Lamps have kelvin and brightness properties that can be changed. The kelvin value defines a smart lamp's color. Smart lamps can also be switched on or off like other lamps.

Colored Smart Lamps: Colored Smart Lamps have the same properties as smart lamps. Alternatively, a color code instead of a kelvin value can be given to define the color of a colored smart lamp.

Smart Cameras: Smart Cameras have a megabytes-per-minute property. The property helps a smart camera to measure the size of the recordings taken. The measurement is done when the smart camera is switched off, in order to gather the on-time.

Smart Plugs: Smart Plugs can measure the energy consumption by gathering the ampere information of the plugged in device and the on-time. The measurement is done when the plug is switched off or an item is plugged out.

Smart devices also have common properties like name, on/off information and switch time. Switch times show when a certain device will change its status. Switch times are optional values that can be assigned to smart devices.

The smart home system also keeps track of the current time and provides the time information to smart devices that do measurement.

The program that simulates the smart home system provides ways of adding smart devices to the system, changing the values of properties of smart devices, displaying information about them when needed, removing smart devices from the system and setting the initial and intermediate times of the system. The program takes commands from the input file, processes them and outputs them in the output file.

1.2 Solution

Object oriented programming was chosen as the solution approach for this system as smart devices can easily be treated as class objects. Moreover, Inheritance was realized as a very efficient way of reusing methods and attributes.

A smart device abstract class was defined and used for storing all common attributes and methods (e.g. the name property and setName() method). It is also used to store every device in a single arrayList, without the need for multiple lists, each for every smart device type.

All smart devices were defined as a class that stored attributes and methods unique to them. (e.g. mbpm value of smart camera is unique to smart cameras)

Colored Smart Lamp was defined as a subclass of Smart Lamp as it can be treated as an extension of Smart Lamp. (e.g. color code is the only difference between the two)

A Time class was defined to change and maintain the time of the system.

A Device Manager class was defined to add, remove and store smart devices and access their methods. Any given command in the input file has a reflection in the Device Manager class.

HashMaps that mapped the string commands to methods in Device Manager and Time classes were used instead of long switch case blocks.

Because there are many exceptions needed to be handled by the program, exception classes DeviceException, TimeException, TimeFormatException and InitialTimeException were defined.

Lastly, a FileIO class was defined to read the input file and write to the output file.

1.3 Difficulties

Main difficulties that were faced while designing the project are as follows:

Having to take NullPointerException into account when an object has a possibility of being null.

Not being able to concatenate strings because of how Java stores them in the memory, instead using StringBuilder. Note that this is only a memory related concern and not a syntax error.

Comparing strings and other objects with the equals() method instead of using the comparison operator "==" .

The way of mapping strings to methods was unusual.

Checking all potential errors caused by bad commands took a long time, but it made the program explain itself more clearly to the user.

Measuring consumption for smart plugs and cameras was hard to implement and potentially still erroneous. (mentioned in Checklist.pdf)

1.4 Benefits of the System

The system aims to make the user's life better in several ways. Firstly, the smart lamps and colored smart lamps help brighten and color any closed environment. Furthermore, the user can experience different colors and brightness levels by simply changing the properties of these devices via the program. Secondly, the smart cameras and smart plugs help the user be aware of his/her consumption. The smart plugs help the user save money while smart cameras help him/her save storage space and money drained by excessive Internet usage.

Chapter 2

Object Oriented Programming

2.1 Benefits of OOP in this Problem

The first major benefit of using OOP is was realized as the easiest way to code such a problem. Smart devices are all objects in real life and therefore can easily be implemented as objects in Java.

The second major benefit is that it provides readability. The source code is approximately 800 lines long, but it is readable because the code is divided into classes, methods and attributes. Otherwise, there would have been a need for a lot of variables, unrelated-looking methods and confusing code blocks that are all inside the Main class, causing the code to be very hard to maintain.

2.2 The Four Pillars of OOP

The four pillars of OOP that were also used in the program are as follows:

Abstraction: Abstraction is the general idea of OOP and mainly Encapsulation. It is the thought proces of hiding implementation from the user. It aims for the developer to focus on different parts of the code without worrying if the other parts would break as a result.

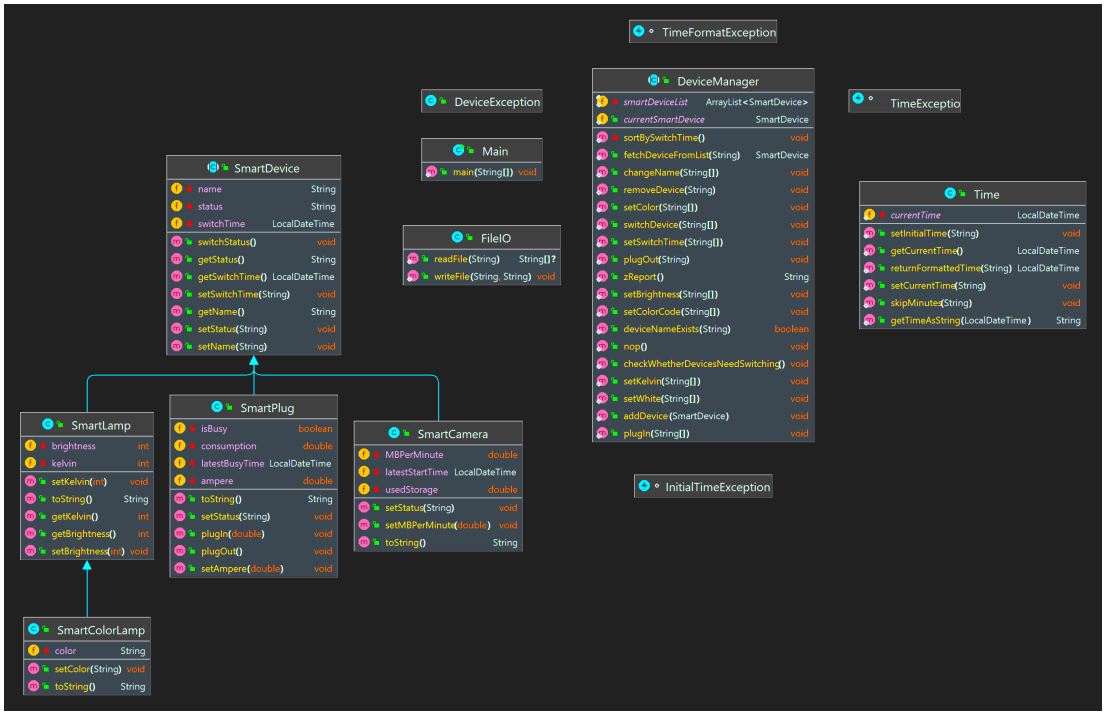
Encapsulation: Encapsulation is the concept of hiding class attributes from the user and providing methods instead to access them (getters) and to modify them (setters). Getters may just return the attribute as is or may be used to fetch the attribute in some specific format. Setters may just set the attribute normally or may be used to put constraints to be checked in order to set the attribute.

Inheritance: Inheritance is the concept of establishing a hierarchy between super and subclasses. Any object that belongs to a subclass also belongs to the superclass of that subclass and therefore may use the methods or attributes of the superclass (except for private ones). Inheritance is important for preventing code copy-pasting.

Polymorphism: Polymorphism is the concept of accessing subclass objects with a reference of the superclass type. Only the methods and attributes declared in the class of the reference type can be accessed, although, if possible, their form in the object's class is called. Polymorphism is the result of Inheritance and class hierarchies.

Chapter 3

UML Diagram of the Program



The **UML Diagram** is a diagram that represents classes as squares. At the top of a square, the name of the class is written. In the middle section, the attributes of the class are written. Finally, at the bottom, the methods defined in the class are written. The lines that connect two classes show that there is an inheritance hierarchy in between them.

All classes in the source code are represented in this UML Diagram.

Exception classes **DeviceException, TimeException, TimeFormatException and InitialTimeException** are exception classes and are not connected to any other class apart from RuntimeException.

The **Time** class is not connected to any other class. It contains an attribute representing the current time and useful methods to set and format time values.

The **DeviceManager** class is not connected to any other class. It contains the **smartDeviceList** attribute representing the arrayList where devices are stored. It also has the attribute **currentSmartDevice** that represents the current device being added, removed, having its values changed etc. The methods in the class provide ways of adding, removing devices; sorting devices, changing attributes of devices etc.

The **FileIO** class is not connected to any other class and is only used for two purposes: reading the input file and writing to the output file.

The **SmartDevice** abstract class is the super class of all smart device classes. It contains attributes that are common among all smart devices and methods related to these attributes. It is also the type of the **smartDeviceList** arrayList of **DeviceManager**.

The **SmartLamp, SmartColorLamp, SmartPlug, SmartCamera**, classes are classes that represent all types of smart devices. **SmartColorLamp** is the subclass of **SmartLamp** because it is a kind of smart lamp.

Chapter 4

Resources

The methods in the FileIO class were adopted from our research assistants.

Method **readFile()** was adopted from *RA. Nebi YILMAZ.*

Method **writeFile()** was adopted from *RA. Görkem AKYILDIZ.*