Smart Device Project

Student Name: Dursun Zahid Korkmaz

Student Id: 2210356020

Date: 22.04.2023

Index

Introduction:

This Project is designed to implement a smart device system using object-oriented programming concepts. In this project, we have focused on creating smart devices that can control different home affairs and provide relevant information about them to the user. The project has been implemented in Java, which is a popular object-oriented programming language known for its simplicity, ease of use, and platform independence. We could use a better option known as C#-Dotnet but this was enough too.

The goal of this project is to demonstrate the use of object-oriented programming concepts in creating a real-world application. It highlights the benefits of OOP, such as encapsulation, inheritance, and polymorphism, which make the code more organized and reusable.

Problem Definition:

The problem that we are trying to solve with the smart device system is the need for a more efficient and convenient way of controlling various household devices. With the increasing complexity and number of electronic devices in our homes, it can become difficult to manage them all manually. Smart devices aim to solve this problem by providing a central platform for controlling all devices in a user-friendly interface. This can help to save time and energy, as well as increase overall comfort for users.

Solution Approach:

Our solution approach is to create a system with different parts all working together according to OOP design principles. We created an abstract base class for the smart devices which holds the common properties and methods among all the smart devices. And then created various smart device classes which inherit from it. Also we created some additional staticly used service-like classes (TimeManager, DeviceManager..). Each code unit will be responsible only for their relevant issues.

Problems Faced and Solutions:

During the development of the project, we faced several challenges, such as organizing, executing and exception handling the commands we read from the input file. However, we were able to overcome these challenges by obeying the rules known as Single Responsability Principle and Open-Closed Principle. Because each class tackles with only the problems related to them, each object checked for their own command exceptions and own organizations.

Benefits of the System:

The implementation of this system has several benefits, such as easy usage and precise time managment. The system is very easy to use because the user only needs to give one file containing all the commands needed. And also the timing of the system is so precise that the user can give timestamps up to minute-second precision.

Benefits of OOP:

Object-Oriented Programming (OOP) has several benefits to talk about. For example it breaks the code into relevant process units and this makes Project files a lot shorter. Hence the Project becomes more easier to read and evaluate, and obviously maintain. Also OOP seperates a Project into different layers which don't communicate to each other directly, instead each layers communicates using interfaces. They are like end-point rules which we can replace the class on the other side whenever it is needed.

Four Pillars of OOP and UML:

The four pillars of OOP are:

Encapsulation - the ability to hide implementation details and expose only necessary information.

Inheritance - the ability to derive new classes from existing ones.

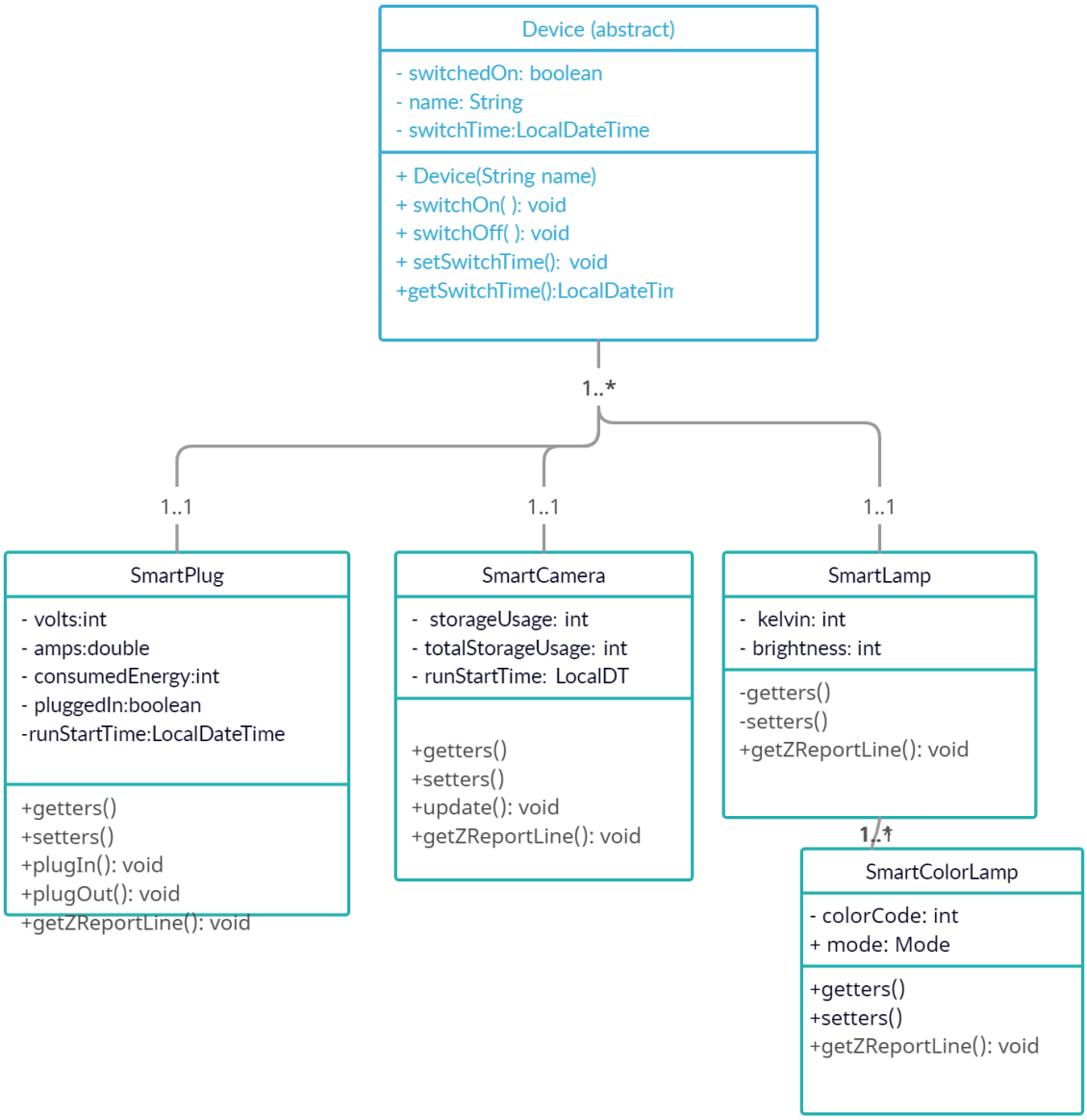
Polymorphism - the ability to use a single method or object to perform different tasks.

Abstraction - the ability to focus on essential features and ignore the implementation details.

UML:

Unified Modeling Language (UML) is a visual language that is used to represent object-oriented designs. It includes various types of diagrams, such as class diagrams, action diagrams etc..

UML Diagram and Explanation:



We have an abstract class called Device, we also have 3 different child classes which inherit from it: SmartPlug, SmartCamera, SmartLamp. We have one more class called SmartColorLamp which inherits from SmartLamp. This hierarchy makes various aspects of the implementation easier. For example each device should have switchOn and switchOff methods. The parent abstract class Device has those methods and necessary fields. If one of the childs has to do a unique operation on switchOff (for example calculating the energy consumption) it can easily override the switchOff method. Also we have a method decleration in Device which all of the childs implemented called getZReportLine(). Abstract classes make using declerations and implementations together possible.

Conclusion:

In summary, we were able to solve “the complex usage of home devices” issue by creating various Java classes. OOP and UML helped us make the project better, and we tackled some difficulties while working on it.