HACETTEPE UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

BBM 104

PROJECT ASSIGNMENT 2 - SMART HOME SYSTEM

Author:
Aykut Alp GÜRLER
B2210356024

Advisor:
R.A. Görkem AKYILDIZ

INDEX

1. Description

It is a well-known fact that humankind has always sought to do things with less effort, as indicated by our evolution progress. As a result, numerous inventions have been made to simplify our lives and allow us to have more leisure time. Automation, which has been developed to satisfy this desire. Smart Home System is one such project that has been created for the purpose of automation. This project requires individuals to attempt to make life easier.

1.1 Problem Definition

Java is based on OOP, which has simple rules to follow, known as the four pillars of OOP: Inheritance, Polymorphism, Abstraction, and Encapsulation. This project assumes that a good OOP design will be implemented for the given scenario, and that time flow will be controlled by the individual.

1.2 Solution Approach

To implement a successful project, it's crucial to start by deeply understanding the problem at hand. This involves spending sufficient time and effort to comprehend the intricacies of the problem before developing an effective and efficient class structure. The class structure should be well-designed, organized, and documented with clear and concise functionality of each class and its methods. It's important to consider how the classes and methods will interact in the main method of the project and include specific variables to facilitate the flow of time and ensure smooth implementation. Once the class structure is developed, it's equally important to establish clear relationships between objects to ensure that they can perform their intended tasks effectively and efficiently. This requires careful consideration of how different objects created in the main method will work together. Finally, the project should be structured abstractly to make it user-friendly and efficient for its intended audience. This involves creating a framework that allows users to perform only necessary operations without getting bogged down in unnecessary details. By following these steps, it's possible to create a successful and well-implemented project.

1.3 Faced Problems and Solutions'

Firstly, a serious abstraction was required for the project. Four objects (SmartPlug, SmartCamera, SmartLamp, SmartColorLamp) were created as desired by the project. Considering the relationship and commands between the classes, it was thought that an upper class was needed for the objects to work together but also to have their unique operations, and in this context, the abstract class "SmartHomeAccessories" was created. When observing the commands, the problem of having a specific relationship between SmartLamp and SmartColorLamp objects was overcome by using the "Inheritance" relationship. The problem of these objects needing to operate with the same commands in the main method and the solution to this problem was solved using the "Abstraction" principles of the previously mentioned SmartHomeAccessories class. The foundation of "Polymorphism" was used by keeping these objects in an ArrayList with this variable in the main method. The user should not be able to access any structure other than the commands related to the Smart Home System, and this problem was solved using the "Encapsulation" principles. This first part was one of two events in the project's submission. The second part was perhaps the more difficult implementation of time flow control. The objects in the system needed to be updated on various topics with the commands entered by the user. The energy spent by the SmartPlug object, the space used by the SmartCamera object for storage, are just two of these updates. The ability of the user to instantly turn on and off the four objects and the ability to specify a time for these objects to turn on and off was again a situation that required attention and these situations were resolved using “LocalDateTime” provided by Java. Finally, the problem encountered was the sorting algorithm. At this point, the “Comparator" class provided by Java was used again, but afterwards, the detail of preserving the order of the two objects in the sorting process when their time was the same was resolved by the written methods.

1.4 Benefits of The System

The benefit of the system is automation and simplification of daily tasks. Specifically, the Smart Home System is designed to make life easier by automating various tasks in the home such as switching on and off devices like lamps and plugs, and controlling the flow of time. The system is based on object-oriented programming principles and utilizes abstraction, inheritance, polymorphism, and encapsulation to create a well-structured and user-friendly system. The system also updates users on various topics such as energy consumption and storage space used. The use of a sorting algorithm helps to maintain order and efficiency within the system. Overall, the benefit of the Smart Home System is to make daily life easier and more efficient through automation and smart technology.

2. OOP

The principles and fundamentals of Object Oriented Programming are perhaps the most important point and purpose of this project. In this part, the benefits and principles of OOP will be explained, as well as the structure used in this project.

2.1 Benefits of OOP

Object-oriented programming (OOP) is a programming paradigm that is used due to its many benefits, including the ability to organize code into reusable and modular structures, reducing the development time and effort required. With OOP, we can create classes that represent objects and encapsulate their data and behavior, making it easier to manage and modify the code. In addition, OOP allows for easy testing and debugging, as objects can be tested and modified independently of one another, without affecting the rest of the program. Overall, OOP is a powerful and efficient way to develop software applications, and its principles and practices continue to shape the way we design and build software today.

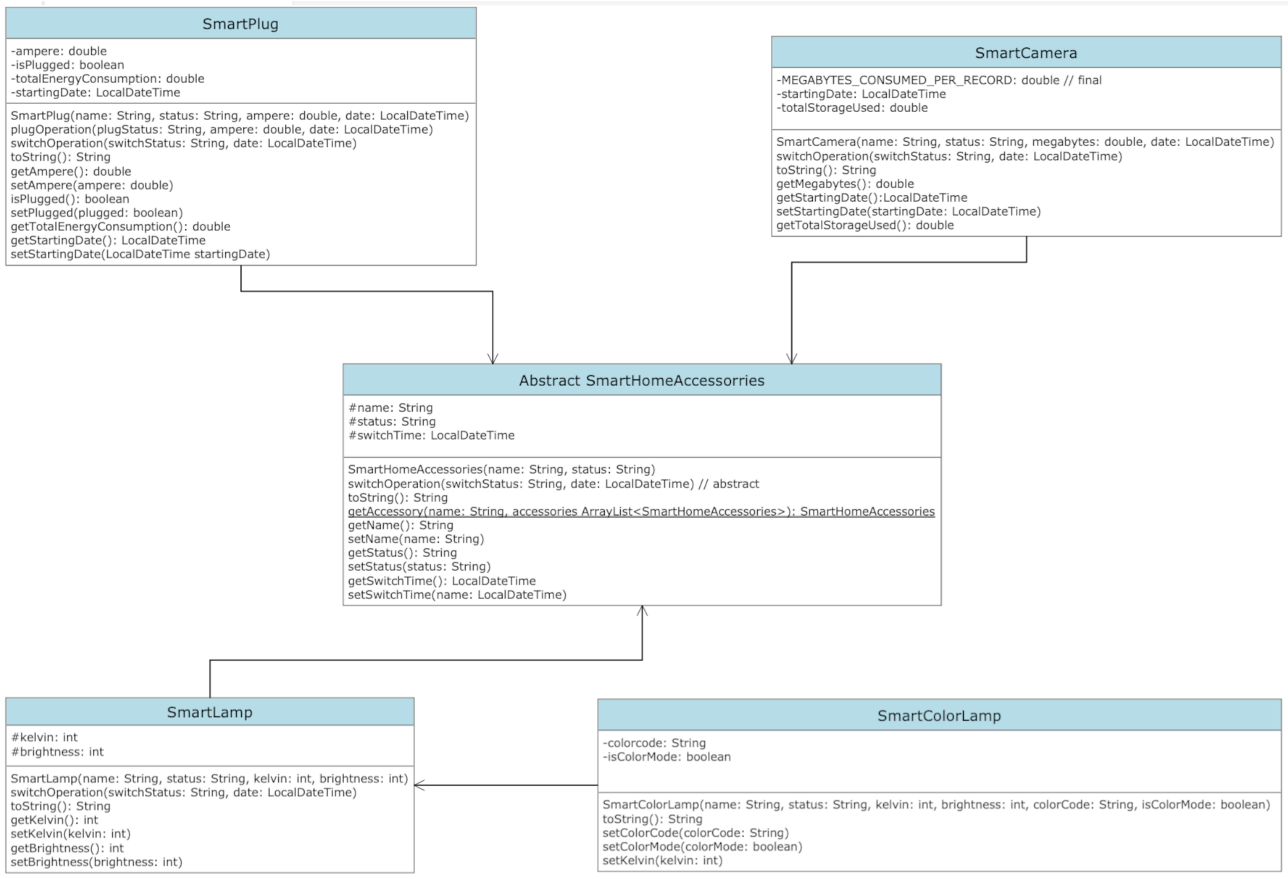
2.2 Four Pillars of OOP

Inheritance allows a class to use the properties of its parent class. To put it simply, it enables dogs to have the common traits of animals. Polymorphism allows objects to perform the same function with different behaviors for different objects that are children of a parent class. Abstraction enables us to adapt the complexity of the real world into the code without delving too deeply into the specifics, in a more general context. Encapsulation provides access to some information while hiding and keeping others private. In this way, it allows us to hide data and increases the sustainability of the program.

2.3 UML Diagram

At the top of the hierarchy is an abstract class called "SmartHomeAccessories", which has three instance variables: name (String), status (String), and lastSwitchTime (LocalDateTime), as well as getter and setter methods for each variable. It also has an abstract method called "switchOperation" which takes a String parameter called "switchStatus" and a LocalDateTime parameter called "switchTime".
Two classes inherit from the "SmartHomeAccessories" class: "SmartPlug" and "SmartCamera".
The "SmartPlug" class has additional instance variables: ampere (double), isPlugged (boolean), totalEnergyConsumption (double), and startingDate (LocalDateTime), as well as getter and setter methods for each variable. The class also has a constructor that takes the name, status, ampere, and starting date as parameters, and two methods: "plugOperation" and "switchOperation". The "plugOperation" method takes a String parameter called "plugStatus" and a LocalDateTime parameter called "plugTime", and updates the "isPlugged" variable and energy consumption accordingly. The "switchOperation" method takes a String

parameter called "switchStatus" and a LocalDateTime parameter called "switchTime", and updates the "status" variable and energy consumption accordingly.
The "SmartCamera" class also has additional instance variables: MEGABYTES_CONSUMED_PER_RECORD (double), startingDate (LocalDateTime), and totalStorageUsed (double), as well as getter and setter methods for each variable. The class has a constructor that takes the name, status, megabytes, and starting date as parameters, and overrides the "switchOperation" method of the superclass. The "switchOperation" method takes a String parameter called "switchStatus" and a LocalDateTime parameter called "switchTime", and updates the "status" variable and calculates the amount of storage used by the camera since the startingDate using the MEGABYTES_CONSUMED_PER_RECORD variable. The calculated value is added to the totalStorageUsed variable.
Finally, there are two more classes that inherit from the "SmartHomeAccessories" class: "SmartLamp" and "SmartColorLamp". The "SmartLamp" class has instance variables for kelvin (int) and brightness (int), as well as getter and setter methods for each variable. It has a constructor that takes the name, status, kelvin, and brightness as parameters, and overrides the "switchOperation" and "toString" methods of the superclass. The "switchOperation" method takes a String parameter called "switchStatus" and updates the "status" variable, while the "toString" method returns a string representation of the object.
The “SmartColorLamp” class, inherits from the class “SmartLamp”, has additional instance variables: colorCode (String) and isColorMode (boolean), as well as getter and setter methods for each variable. It has a constructor that takes the name, status, kelvin, brightness, colorCode, and isColorMode as parameters, and calls the constructor of its superclass. The "toString" method of "SmartColorLamp" overrides the method of its superclass and returns a string representation of the object that includes the colorCode value if isColorMode is true, or the kelvin value if isColorMode is false.


SmartPlug
-ampere: double
-isPlugged: boolean
-totalEnergyConsumption: double
-startingDate: LocalDateTime
SmartPlug(name: String, status: String, ampere: double, date: LocalDateTime)
plugOperation(plugStatus: String, ampere: double, date: LocalDateTime)
switchOperation(switchStatus: String, date: LocalDateTime)
toString(): String
getAmpere(): double
setAmpere(ampere: double)
isPlugged(): boolean
setPlugged(plugged: boolean)
getTotalEnergyConsumption(): double
getStartingDate(): LocalDateTime
setStartingDate(LocalDateTime startingDate)
SmartCamera
-MEGABYTES_CONSUMED_PER_RECORD: double // final
-startingDate: LocalDateTime
-totalStorageUsed: double
SmartCamera(name: String, status: String, megabytes: double, date: LocalDateTime)
switchOperation(switchStatus: String, date: LocalDateTime)
toString(): String
getMegabytes(): double
getStartingDate():LocalDateTime
setStartingDate(startingDate: LocalDateTime)
getTotalStorageUsed(): double
Abstract SmartHomeAccessorries
#name: String
#status: String
#switchTime: LocalDateTime
SmartHomeAccessories(name: String, status: String)
switchOperation(switchStatus: String, date: LocalDateTime) // abstract
toString(): String
getAccessory(name: String, accessories ArrayList<SmartHomeAccessories>): SmartHomeAccessories
getName(): String
setName(name: String)
getStatus(): String
setStatus(status: String)
getSwitchTime(): LocalDateTime
setSwitchTime(name: LocalDateTime)
SmartLamp
#kelvin: int
#brightness: int
SmartLamp(name: String, status: String, kelvin: int, brightness: int)
switchOperation(switchStatus: String, date: LocalDateTime)
toString(): String
getKelvin(): int
setKelvin(kelvin: int)
getBrightness(): int
setBrightness(brightness: int)
SmartColorLamp
-colorcode: String
-isColorMode: boolean
SmartColorLamp(name: String, status: String, kelvin: int, brightness: int, colorCode: String, isColorMode: boolean)
toString(): String
setColorCode(colorCode: String)
setColorMode(colorMode: boolean)
setKelvin(kelvin: int)