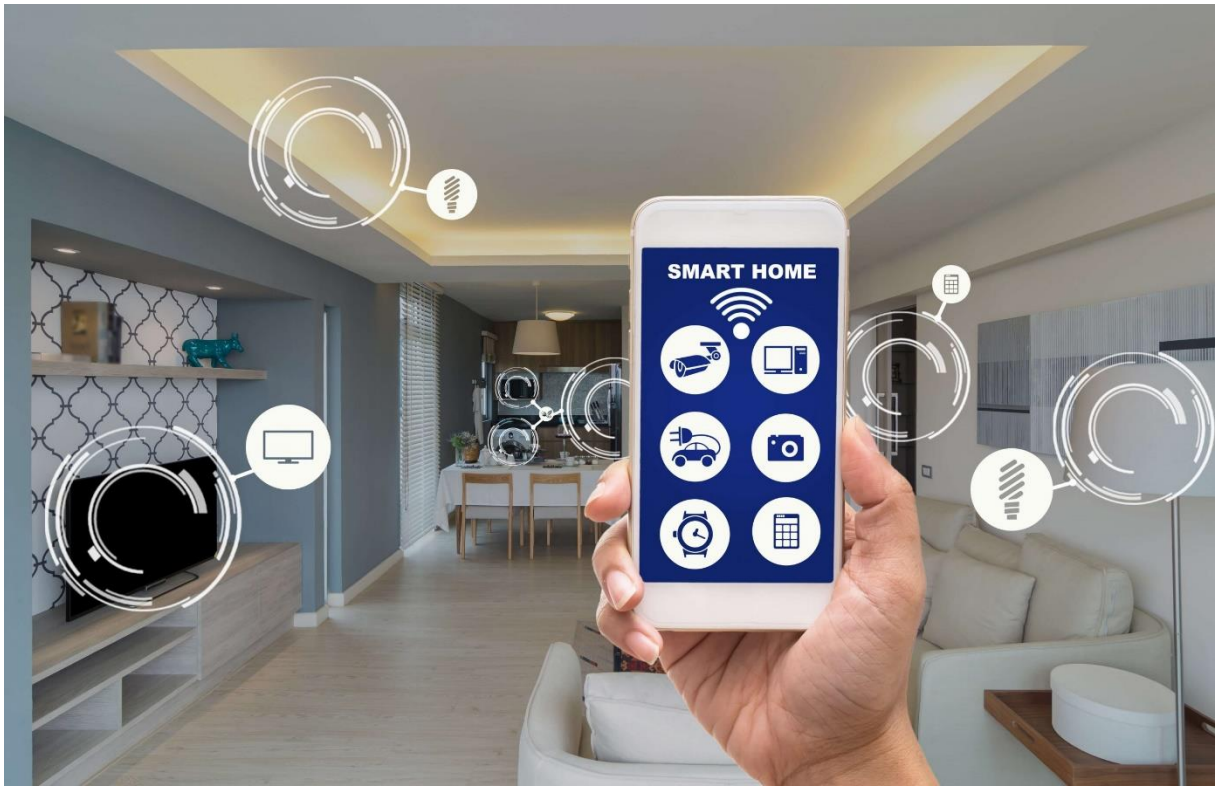
BBM 104
ASSIGNMENT 2

SMART HOME SYSTEM



04/21/2023 Yusuf Küçüköner 2210356092

INDEX:

Problem:

The problem involves creating a program to manage a smart home system that comprises four types of smart devices, namely Smart Lamp, Smart Color Lamp, Smart Plug, and Smart Camera. The program should allow the user to perform various actions on the devices, such as adding new devices, removing existing devices, changing the names of the devices, and switching the status of the devices on or off.

The system also involves time-related problems that the program needs to handle. The program should allow the user to set the switch time of a device, which means the time at which the device should turn on or off. The user should also be able to set the initial time for the system, which is the starting point for time. Additionally, the program should allow the user to change the time based on correct commands. The time-related calculations will be used for energy and storage calculations. The program should also store data for each device for further analysis.

In summary, the problem involves creating a program to manage a smart home system that includes four types of smart devices. The program should handle various actions on the devices, manage exceptions, and perform time-related calculations such as setting switch time of a device, setting initial time, changing time based on correct commands, energy calculations, and storage calculations etc.

Solution:

The problem involves managing a Smart Home system that contains multiple devices with common and unique features. Each device represents a different object, but they share common properties, which require a common ancestor. To address this, five classes are defined: SmartDevice, SmartLamp, SmartPlug, SmartColorLamp, and SmartCamera. SmartDevice is the main class that contains the shared properties of all objects. Additionally, SmartColorLamp inherits from SmartLamp, as both lamps have many common features.

To operate on these objects, five different classes are defined: EditSmartDevices, EditSmartPlug, EditSmartLamp, EditSmartColorLamp, and EditSmartCamera. While add and remove commands are applicable to all devices, PlugIn and PlugOut are only valid for SmartPlug. Thus, four more classes are created, each with common editing commands and device-specific editing commands. All of these classes work with specific commands, but the command needs to be correct. Due to the possibility of exceptions caused by incorrect commands, five different exception handling classes are created, each of which acts as the exception handler for its corresponding edit class. In some cases, exceptions are handled within the edit classes, as it is more logical and simpler.

In the second part of the problem, time and time-related commands are introduced. For this purpose, a time class is written to manage time. Time is used for various calculations, such as energy and storage, and also for setting switch times for devices. The time class works with specific commands, and exception handling is also considered in case of incorrect input.

In summary, the problem involves designing an OOP-based Smart Home system that can manage multiple devices with common and unique features. The system must handle commands related to device management and time, with appropriate exception handling to address potential errors.

Obstacles on the way to the solution:

The biggest obstacle encountered while setting up the system was the calculation of memory and energy, which could be calculated simply with easy time structures, but it was deemed that obtaining good results in complex time situations required a serious algorithm. For instance, in a code like this, there is 1 Camera and 1 Plug. The Camera is on and uses 10 megabytes of memory per minute, while the Plug is on and plugged in, the amperage of the current is 10 and the time is 14:00. The Plug's switch time is set to 14:20 and the Camera's switch time is set to 14:30. If the time is advanced by 40 minutes with the SkipMinutes command, the energy needed to be calculated in the Plug is 20 minutes, while the energy that needs to be calculated in the Camera is 30 minutes. Therefore, the time needs to be first changed to 14:20, and the Plug and Camera need to be set to 14:20, which requires the calculation of 20 minutes. Then, the time needs to be set to 14:30 and the remaining 10 minutes of the Camera need to be calculated. Finally, the time needs to be assigned to 14:40.

There are 3 tools to change the first given time with SetInitialTime: Nop, SkipMinutes, and SetTime. The normal Nop function provides the time of the nearest switch time. SkipMinutes advances the specified minutes, and SetTime takes the desired date. In this case, the process was performed with SkipMinutes(the same algorithm applies to SetTime too). To prevent any problems when SkipMinutes goes forward 40 minutes: Firstly, the array list of devices was sorted correctly by switch time and old switch time. Then, the switch time of the 1st device was looked at and compared with the new time at which specific minutes were added. If the new time is greater than the switch time, the Nop operation comes and first performs, and it goes to 14:20. The part up to 14:20 is calculated, then the device list is sorted again, if the new time is greater than the switch time in this case it is. Nop operation comes and performs again and time goes to 14:30 and required calculations are handled. This loop keeps going until the first device’s switch time is null or greater than the new time. Then, the loop will stop and the program will continue. In this way, the time calculations that could be given at such endpoints were handled.

The second problem that was encountered was the issue of sorting devices while exporting ZReport. Although the devices were sorted in order of Switch Time and Addition, the old switch times were also included in the sorting. The devices whose switch time had come and gone were sorted on top of the devices without the old switch time, in reverse ascending order between the devices with old switch time. After this was understood, it was decided to use the comparator. With the switch times, in the comparator, the devices were sorted accordingly. If there was no switch time, reverse sorting was performed with the old switch times, and the problem was resolved.

Benefits of this system:

Managing a home is made easier with the use of a smart home system. Various information such as energy consumption, memory usage, number of lamps and plugs in use, can be obtained from a smart home, facilitating home management. A warning can be issued by the smart home system when a device is left on for longer than necessary. This helps to prevent unnecessary energy consumption which provides benefiting the environment and allowing us to save money. The smart home system is capable of performing various tasks automatically, providing security and saving money, which leads to a reduction in stress and an improvement in overall quality of life. A feeling of happiness and comfort can be experienced through the use of a Smart Home system, which is often considered a luxurious lifestyle choice.

Benefits of OOP:

OOP focuses on the objects to use represent data and behaviours. OOP provides flexibility, building primitive blocks first then reunite them(modularity), in this way we can change code more easily because blocks isn't bond each other. We can call this 'discrete systems'. This process boosts the abstraction, reusability. Flexibility was utilized in the Smart Home project, as separate classes were prepared and utilized to handle exceptions without affecting the class that performed. This event serves as an example of abstraction, where different classes are possible to preserve the cleanliness and simplicity of the code. Abstraction can be achieved by using interfaces that act as bridges with the outside world. Another important contribution of OOP is encapsulation, which is obtained by hiding implementation details (abstraction) and data due to the dangers that may come from the outside world. In this way, data that we do not want to be accessed privately can be protected.

Encapsulation also helps in maintaining an object's state. In the Smart Home project, encapsulation was provided by using private variables and interfaces.Inheritance allows for the reuse of code and prevents code from repeating itself. Common features in the Smart Home project were collected in superclasses, while subclasses were given specific features, thus avoiding code duplication.

Polymorphism helps to program in general rather than specifically. Polymorphism allows for explicit casting of the superclass to the subclass type, which was fundamental in the smart home project. In the smart home project, all devices were thrown into the same arraylist, as they all have the same superclass. By using their specific features instead of the superclass features of the class that was thrown, whenever the same arraylist entries were pulled, I was able to access their specific features by doing type casting whenever I wanted. Thanks to polymorphism, more flexible and extensible systems can be established.

Four Pillars of OOP and UML:

Encapsulation

Provides security and hides implementation details. Provides security because it helps to protect the object's data to be reached, in other words, saves the integrity of data.

Inheritance

Inheritance allows to reuse of a code and prevents code from repeating itself.

Polymorphism

Enables us to process objects that share the same superclass so we can process their common properties which inherit from the superclass together. Also helps objects to respond to the same method differently. This can be achieved through method overriding. Lastly, provides overloading, which methods can have same name but different signature will make them separate methods.

Abstraction

Under the name of 'Abstract', it tries to present the project to the outside world in a simple way for the user to use. It can be achieved with Interface and abstract classes. The basis is to hide the programming details and show the part that the user will use, and it has a connection with encapsulation in this way.

UML

Graphical Language used to visualize, design software n b n n systems

UML Diagram

https://www.hizliresim.com/gg7b0oo

Explanation of the UML Diagram

The Main class is where the necessary input is received from the IO, the output file is named, and the selectMethod of the Method Resolver class is sent to the method for processing. In selectMethod, various class objects are created and used. EditSmartDevice is the first class to be created, which is then linked to EditSmartPlug, EditSmartCamera, EditSmartLamp, and EditSmartColorLamp to organize devices. EditSmartDevice implements functionality common to all devices, while the other classes implement functionality specific to each device type. The interfaces of these classes, namely EditSmartDeviceInterface, EditSmartCameraInterface, EditSmartPlugInterface, EditSmartLampInterface, and EditSmartColorLampInterface, facilitate communication between the user and the program. EditSmartDeviceErrorHandler, EditSmartPlugErrorHandler, EditSmartCameraErrorHandler, EditSmartLampErrorHandler, and EditSmartColorLampErrorHandler catch exceptions generated by erroneous commands in the Edit classes without disrupting the program's normal operation. SmartDevice, SmartCamera, SmartLamp, SmartPlug, and SmartColorLamp are five classes that represent the properties of devices. SmartDevice is the superclass, while SmartCamera, SmartLamp, and SmartPlug are its subclasses. SmartColorLamp is a subclass of SmartLamp. All classes have valid constructors that accept specific properties for each object (device) of their respective classes. The SmartSystemMethods class includes methods that do not directly impact devices. Finally, the Time class is responsible for managing the flow of time.

References

1. https://www.geeksforgeeks.org/comparator-interface-java/
2. https://www.geeksforgeeks.org/class-type-casting-in-java/
3. https://www.geeksforgeeks.org/calendar-class-in-java-with-examples/
4. https://www.geeksforgeeks.org/date-class-java-examples/
5. https://www.logicbig.com/tutorials/core-java-tutorial/java-regular-expressions/regex-lookahead.html
6. https://www.digitalocean.com/community/tutorials/java-simpledateformat-java-date-format
7. https://www.grammarly.com/
8. https://www.quillbot.com/grammar-check