Hacettepe University
Department of Computer Engineering
BBM104

PROJECT ASSIGNMENT 2: SMART HOME SYSTEM

Abdussamet Tekin - 2220356042

23.04.2023

Definition of the Project

This is a project for making people's lives easier by creating a Smart Home System in Java programming language. In this system, there are many Smart Home Devices such as Smart Plug, Smart Camera, Smart Lamp and Smart Color Lamp.

Smart Device

Each smart device has capability of being switchable, has a name, has current time, has switch time which is initially null and has a string representation. When time of the system is set to or after switch time, devices switch themselves.

Smart Plug

Smart Plug is a plug which user can plug in and plug out a device to. Users may also switch the plug on and off. Smart Plug can also calculate total energy consumption of the plugged device. This is done by calculating the elapsed time in between two switch times and multiplying this value to power of the plugged device.

```
W = P * t = V * I * t
watt-hour = watt * hour = volts * ampere * hour
(W: Work) (P: Power) (V: Voltage) (I: Current) (t: Elapsed time)
```

Smart Camera

Smart Camera is a camera which user can switch on and switch off in order to record videos. Each Smart Camera has a value "Megabyte Per Minutes", which denotes the amount of storage used by the camera each passing minute when it is on. The camera calculates the elapsed time between two switch times and uses this time to calculate the storage used by the device.

```
MB = MB per minute * minute
```

Smart Lamp

Smart Lamp is a lamp which user can switch on or off, set its kelvin value and its brightness value.

Smart Color Lamp

Smart Color Lamp is a modified version of the Smart Lamp and additional to Smart Lamp you can set its color code. When the color code is set, it is in color mode, if kelvin value is set it is in the normal mode.

Solution to This Project

For design of this project, Smart Device can be set as the abstract superclass of all the devices that were defined. Every device has a name and status, so name and status can be declared as the private fields of this class. Same approach can be used for switch time and for the current time that holds the time of the system. As switch operation's implementation may vary device to device, it can be declared as an abstract method.

To operate over Smart Devices, a Smart System class can be created. This class will hold the devices on two data structures, HashMap and ArrayList. The reason behind using two data structures to operate on the object is for two reasons:

1. Querying the HashMap for device name when you have to operate on a specific object in which system specifies with its name,
2. Sorting the ArrayList according to switch times of objects and when system has to do the same operation on every device such as setting the new time and printing the String representation etc.

As in Java the reference variables are just addresses that point to the objects somewhere on the heap, having two data structures to hold these reference variables has no efficiency cost on the program whatsoever since system does not create the same device twice and hold them separately.

Problems that were faced during the development of the project

Problem 1 – Input and Output

The input files can contain blank lines and these blank lines should be ignored and not put into the String input array which will hold the input lines. Overcoming this problem was relatively easy since project used the FileInput and FileOutput classes that were previously written by the TA Görkem Akyıldız. The FileInput class have an option discardEmptyLines which will ignore the blank lines and won't put them into the string array.

Problem 2 – Adding devices to the system

This problem can be solved by creating a SmartDeviceFactory class that will create the object and add it to both HashMap and ArrayList. Since the system should not create a faulty object that contains illegal arguments i.e., a Smart Lamp having -1 brightness value; there should be another class that will check for the validity of the arguments provided by the command.

This class will be CheckArgumentValidity. The CheckArgumentValidity class will check for the validity of the arguments, and if the arguments are valid, it will let the SmartDeviceFactory class to instantiate the device and add it to the system. If all the arguments provided by the command are valid, then CheckArgumentValidity will instantiate ValidArgs class which will hold the valid arguments. Then according to the argument number provided by the validArgs object, different constructors of the Smart Devices will be called and created devices are put into the HashMap and ArrayList in SmartDeviceFactory. It should be noted that CheckArgumentValidity just provides the address to the validArgs object which is a superclass of device type specific argument classes such as LampValidArgs. The CheckArgumentValidity typecasts this object to LampValidArgs in order to reach device type specific arguments.

Problem 3 – Operating Over Devices

Smart System has methods that will do the operations specified by the command. A method selector will call relevant method within the class in order to do a certain operation. For example, setting the status of a device off is done by querying this device from HashMap with its name then doing the operation on the returned value. But when you have to write Z Report, it will iterate over the ArrayList and write the String representation of the devices.

Problem 4 – Time

Smart System has a current time and every other device should synchronize their times with this time. While a device synchronizes its time, it also checks if current time is same or after the switch time. If so, device switches itself.

Problem 5 – Sorting Devices

This problem can be solved by writing a comparator class which will sort devices according to their switch times. This comparator class has a method compare which compares two devices. Sorting is done according to the comparator class.

Benefits Of This System

This system helps humans to operate with devices without knowing the inner structure of the system itself. This is the perfect example of abstraction.

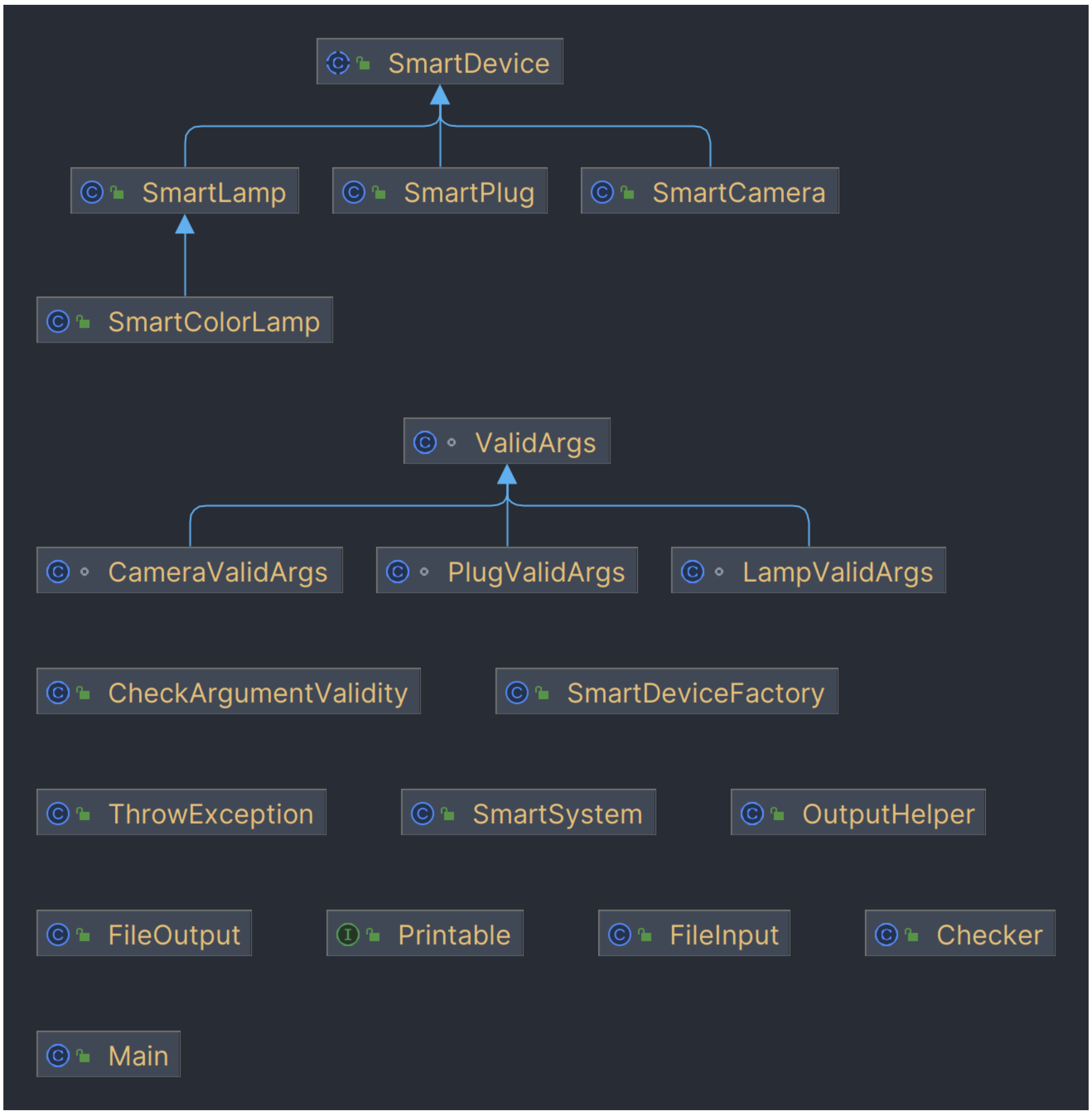
Benefits of OOP

Object Oriented Programming provides encapsulation, inheritance and polymorphism which are very crucial to systems that are complicated as this. It allows code reuse with inheritance, flexibility within the program with polymorphism, and disallows data to be reached from anywhere with encapsulation.

Four Pillars of OOP

1. Encapsulation: This principle says data of a class should not be reached without any restrictions.
2. Inheritance: This principle saves programmers from writing the same procedures over and over again and allows code reuse. And allows linking classes together if they are related inherently.
3. Polymorphism: This principle allows objects of different types act differently when they are asked to do the same task. For example, making noise behaviour is existent in all animals, but dogs “bark” and cats “meow”. They do the same task on their own ways.
4. Abstraction: This principle says to users, implementation and inner workings of the program should not be visible. For example, a basic driver doesn’t need to know how changing the gears work inside of the car. All they have to do is just use the gearshift lever. Or to me a cat just meows, it doesn’t make noise by vibrating its vocal cords while exhaling.

UML Diagram



1. It can be seen from the diagram that all four devices are subclasses of Smart Device. And Smart Color Lamp is subclass of Smart Lamp.
2. CheckArgumentValidity checks for the validity of the arguments.
3. And every valid argument class are subclasses of ValidArgs.
4. Main just instantiates Smart System and calls startSystem() method on the object.
5. Printable is a functional interface for adding prefixes to Strings.
6. When the system has to throw and exception, it calls the static methods of ThrowException class.
7. Checker is comparator class for devices.
8. OutputHelper outputs to file using FileOutput.
9. FileInput lets program to input from the input file.