Professorship of Embedded Systems and Internet of Things
Department of Electrical and Computer Engineering
Technical University of Munich

Homework 2

Software Architecture for Distributed Embedded Systems, WS 2021/2022

Prof. Steinhorst, M. Sc. Emanuel Regnath, M. Sc. Jens Ernstberger

Submission Instruction

Please submit your solution as one **zip file** `firstname_lastname.zip`. In the root of the zip file you must place directly all your main python files for each homework task in the notation `hw2yx.py` where y is the number of the exercise and x the question number (see template ZIP on Moodle). Place additional files (e.g. for classes) in the corresponding subfolder `hw2yx` and include your files with `from hw2yx.YOURFILE ↪ import YOURCLASS`. Besides manual inspection, we also want to do some automatic testing, so please stick to the exact naming pattern. **There must not be any extra subfolders besides the module folders of the form "hw2yx" in the root of your zip file.** If you want to include libraries, please use only standard Python libraries: https://docs.python.org/3/library/

Please note that the goal of this homework is to learn the application of the concepts by working independetly on the tasks. Any similarities of your solution to other solutions that indicate plagiarism will be penalized and might be forwarded to the department.

? Exercise 2.1: Python OOP Basics

A subgroup of your *AwesomeLights* startup has come up with a new idea to make conventional lights dimmable: A smart *DimmAdapter* could be installed bwetween the power wires and a standard e27 socket for conventional switch lights. This adapter can output a PWM signal and set different voltage levels via Smartphone App. If it is an LED, the voltage will be set to $5.0\,\mathrm{V}$ and the duty cycle (in %) is set to the brightness level (in %). If it is a conventional bulb, the voltage is set to $230\,\mathrm{V}$ multiplied by the current brightness level. It also checks the temperature before switching a light on. The maximum allowed temperature for an LED is $350\,\mathrm{K}$, while for the bulb it is $500\,\mathrm{K}$.

1. Look at the template code `hw211.py` and implemente the behavior described above using only inheritance. The concrete light (e.g *LEDLight*) should inherit from the *DimmAdapter*, which inherits the abstract *DimmLight*. Please put your solution into `hw211.py`.

2. Implement the same behavior as in 2.1.1 but using composition, that is a *DimmLight* is composed of a *DimmAdapter* and a concrete light (e.g. *LEDLight*). Please put your solution into `hw212.py`.

3. Compare the two approaches 2.1.1 and 2.1.2. What are their advantages/disadvanteges? Please write a short (50 – 150 words) answer into `hw213.md`.

? Exercise 2.2: Observer Pattern

Your company was hired by Alice and Bob to extend their SmartHome with a notification service. The problem is that (i) Alice always forgets to turn the light off when leaving through the garage and (ii) Bob always forgets to close the garage when he comes back home (hence turns on the light before closing the garage). Hence, the notification service that you should develop needs to notify Alice when she forgets to turn off the light and notify Bob when he forgets to close the garage. You can assume that being at home is equivalent to turning on the light, hence there is no sensor to detect whether someone is at home or not. Consider that both of them will get notified for any of the two events, however, they receive custom messages.

You decide to implement the functionality by applying the observer pattern, as you think it might be suitable to solve the task at hand.

1. Given the description of Alice and Bob's Smart Home system, how many unique states can the Smart Home system be in? What *state changes* are relevant for the notification service as described above? Please write a short (50-100 words) response that justifies your answer into `hw221.md`.

2. Now, it's time for you to implement the logic that enables the notification service for Alice and Bob. Whilst brainstorming with your team you come up with a list of requirements for your implementation.

    1. The Observable should keep track of the current system state. The state of the system is necessary to update notifications upon state *change*. When initialized, the light is turned off and the garage is closed. The state should be represented as an integer.
    2. The logic of the SmartHome system should initiate a state change to enable the user to (i) turn the light on & turn the light off as well as (ii) open the garage & close the garage.
    3. The Observers should be separated for Alice and Bob to enable individual notification messages.

    Implement your SmartHome system in `hw222.py`, such that it fulfills the above-stated requirements. Class and function names are given in the python file.

3. Now, your SmartHome notification system should be operational. Let's see if it works as intended by invoking the following storyline:

    1. Bob comes home. He opens the garage and turns on the light. As per usual, he forgets to close the garage. After receiving the notification, he closes the garage.
    2. Alice leaves, opens the garage, and leaves on the light. She receives the notification but chooses to ignore it as Bob is still back home. She doesn't forget to close the garage.
    3. Bob leaves home. He turns off the light, opens the garage, and doesn't forget to close it after leaving.
    4. After a while of using your service, Alice decides to not use it anymore. Hence, she wants to stop observing the observable by detaching her notification service.

    Implement the client-side code in `hw223.py`, such that it follows the above-mentioned storyline. For detaching Alice you need to implement the respective functionality in `hw222.py`. The Log file that your program should produce is shown in hw223.log. There is a function `test()` in `hw223.py` which shows you how the respective function calls should look like.

4. One of your colleagues suggests that you should rather use the 'publish-subscribe' pattern to implement the functionality demanded by Alice and Bob. You disagree with him and decide to stick with the

Observer Pattern. How do you justify your decision? Please write your answers in `hw224.md`. (50-100 words)

Remark: Class and function names are given in the python file. The Observer/Observable classes are implemented and should not be changed. Please only change your code in the dedicated sections. Before submitting your code, check that both `hw222.py` and `hw223.py` are properly formatted with Pylint (https://pylint.org//).