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 2020/2021

M. Sc. Regnath, Dr.-Ing. Hamad, and Prof. Steinhorst

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/`

? 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 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 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 K, while for the bulb it is 500 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`.

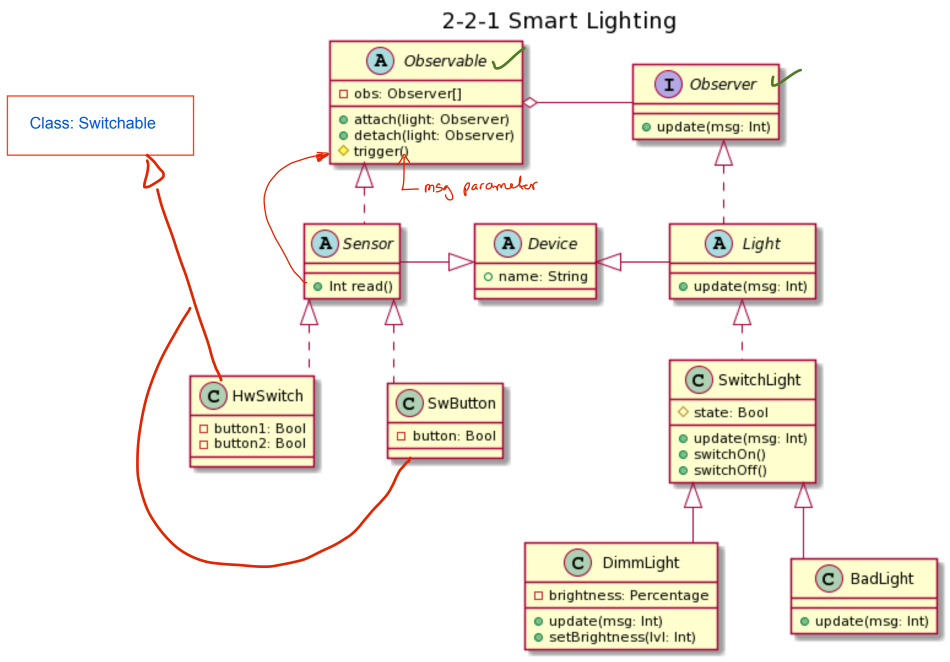
2. Implement the same behavior as in 2.1.1 but using composition, that is a *DimmLight* is composed of a *DimmAdapter* and a concrete light (e.g. *LEDLight*). Please put your solution into `hw212.py`.

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 and Publish-Subscribe Patterns

In the last homework, we have a group of lights in the system that must react to a command issued by the different controls. Using *observer* pattern seems very reasonable for modeling this part of the system as shown in the figure below.

write main() !



1. Extend the solution of tutorial 4 question 1 (`code_ex_4_1_3_sol.py`), which is in the hw22x folder, to implement the "2-2-1" UML class diagram shown above. Submit `hw221-code.py`
   Note: using LightConfig instead of msg for the update function, as in the UML diagram of Exercise 2.3, will be considered correct too.

2. Let us consider a case where the *BadLight* was implemented badly; the update function was performing an endless loop. Update your program to implement such a case. Try to run your code and explain what happening and why? discuss whether the issue can be solved by running each observer with a separate thread? Submit `hw222-code.py` which contains the issue and proposed solution, and submit `hw222-solution.md` or `.txt` to explain the issue and your solution (50-100 words).
   Note: deleting the loop from the Badlight is not acceptable solution.

3. Another way to model the relations between the group of lights and the different controller of the system is by using the so-called Publisher-Subscriber pattern (read about it here). Usually, this pattern is used when the publisher and subscriber components do not belong to the same address space, and they communicate over a network. However, we will implement the publisher-subscriber pattern in this home-work where these components belong to the same address space. The file `hw223-pub-sub.py` contains this implementation. Your tasks are:

   1. Update the code of `hw223-pub-sub.py` by adding one publisher `publisher2` which publishs `message3` under topic `topic3` and one subscriber `subscriber3` which subscribes to the topic `topic3`. Submit `hw2231.log` which is generated by the program.

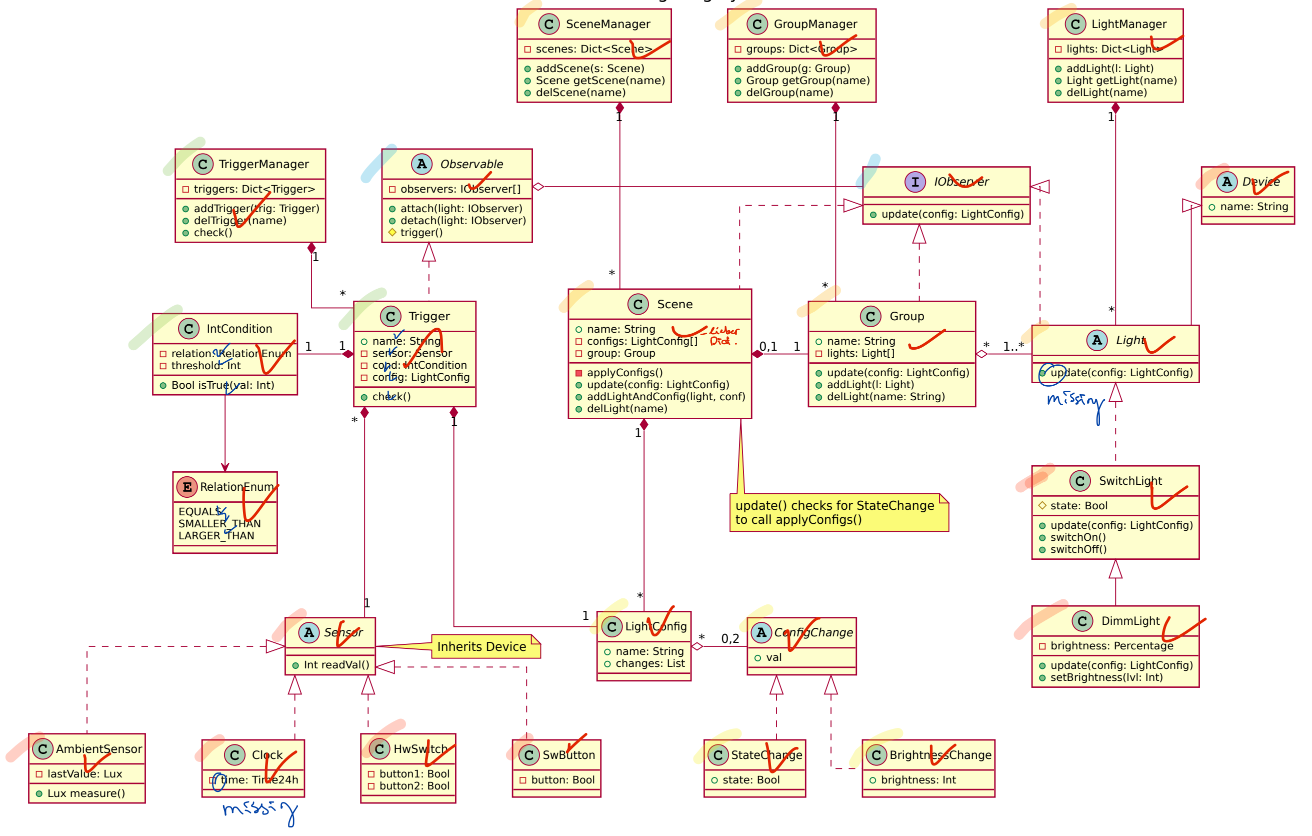
2. Update your UML diagram shown in the figure above to use the Publisher-Subscriber pattern instead of Observer pattern. Submit `hw2232.svg`.
3. Implement your proposed class diagram `hw2232.svg` by using the the code in `hw223-pub-sub.py`. Submit `hw2233-code.py` and `hw2233.log` of your program.

4. Compare the Observer pattern and the Publisher-Subscriber pattern. Please consider the case when the Publisher-Subscriber pattern is implemented over the network. Discuss if we will face the same issue as in 2-2-2. Submit `hw224-comparison.md` or `.txt` (max 50-200 words).

? Exercise 2.3: Implementing a Smart Light System

Now, we will implement the Smart Light System from Homework 1. Below (and in the template ZIP) is the final version of the UML. Note that we have changed some names, so please stick to this version and do not use the UML from the HW1 solution.

2-3-1 Smart Lighting System



Your Task

1. Implement the Smart Light System in Python 3. Name your classes, functions and attributes exactly as shown in the UML, except for private attributes, where you should prepend a underscore (_). In general, please follow our updated Software Design Guide. Your final code should fulfill the requirements from Homework 1 and not crash when calling your public interfaces in arbitrary order. Examples how we will test your code are shown in the template `hw231.py`. Use the formatter `black`[1] for code layout. Please submit `hw231.py` and all module files in `hw23x`.

[1] Install here: https://black.readthedocs.io/en/stable/installation_and_usage.html. If you have problems with the installation, use the online version: https://black.now.sh/