Assignment 3

INFSCI 0201 - Intermediate Programming with Python. Fall 2023

Due Date: December 8, 2023, before 11:59 PM

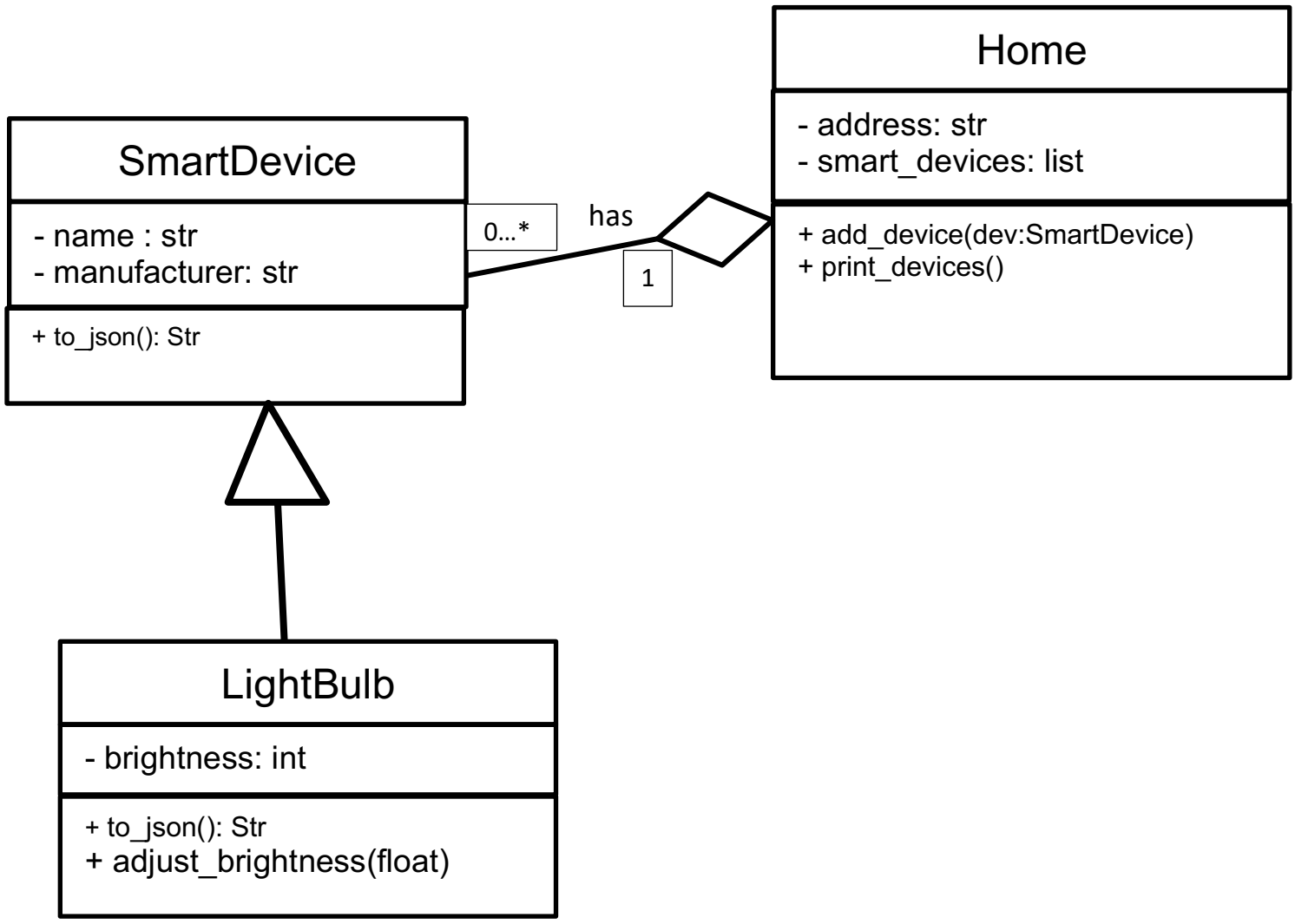
This assignment has 2 parts. First part covers a slight modified version of one midterm question. Second part includes implementation of a Flask application.

This assignment is 150 points, meaning that it will be weighted more compared to the previous 2 assignments.

Part #1: Implement Classes given Class Diagram (40 points)

Instructions:

Consider the following UML class diagram representing smart devices that a home might have.



Tasks:

1. **[15 points]** Implement each class fully. This includes constructors, properties and methods listed on UML class diagram, and getters&setters that are not listed. For the child class, you have to call the parent class' constructor. Please check your lecture slides to find out how to call parent class constructor properly. Adhere to field types noted in the diagram. For example, do not change a list variable to a tuple. Constructors should take parameters to initialize instance variables, including name, manufacturer, brightness, address. Do not implement a constructor that does not take any parameter.
2. **[10 points]** to_json() method is left unimplemented in SmartDevice class. Leave it unimplemented and make SmartDevice class an abstract class. Check your slides to see how you can do it. LightBulb overrides this method and returns a proper JSON string that includes its fields: name, manufacturer, and brightness. Reuse your code from Assignment 2. (5 points)
3. **[5 points]** adjust_brightness(float) of LightBulb: Takes a float number and sets its brightness and prints out an info message: "Brightness is set to <brightness_value>". Replace <brightness_value> with actual float number. (5 points)
4. **[5 points]** add_device(dev:SmartDevice) of Home class: Gets a SmartDevice object and adds it to its smart devices list. Also, it asks the smart device object to connect to a network (call its function). (5 points)
5. **[5 points]** print_devices() of Home class: Prints out info about all devices added to home. (5 points)

Part #2: Create a Flask App for Home Devices in Part#1 (110 points)

In the second part, you will create a Flask application for home devices.

Tasks:

1- Create a new folder for your Flask application. Create a virtual environment and install required packages (e.g., Flask)
2- Take one of the simple Flask application codes from course's GitHub repository, and copy. (Do not implement your Flask app in a Jupyter notebook). Rename the example Flask application file as assignment_3.py. Your application code should be under your main application folder that you created in Step 1.
3- Create two subfolders under your Flask application folder created in Step 1. First one is called templates, and the second one is model.
4- **[10 points]** Save your code from Part #1 to smart_devices.py that includes the implementation for SmartDevice, LightBulb and Home classes, all in the same file. Save/move smart_devices.py under the folder called model.
5- **[10 points]** Inside assignment_3.py, import smart devices that you have implemented in Part 1. If you put smart_devices.py in model folder properly, you should be able to use the following code to import these classes:

```
from model.smart_devices import Home
from model.smart_devices import SmartDevice
from model.smart_devices import LightBulb
```

This way, you will be able to access code written in another Python file as we were doing with other packages.

6- **[15 points]** In your Flask app, implement a function called index() that would be called when the default URL is called, i.e., 127.0.0.1 or localhost. In your index function, create a new home object and a light bulb object. Add lightbulb object to home.
7- **[15 points]** The index function in step 6 should render a template that shows information about the home object: Its address, and its smart devices. Create a template called home.html under templates folder you

have created in step 3. Following the examples we had in the class, render home.html. Do not assume that the home only has a certain number of light bulbs. Your code should work with any number of smart devices added to the home. For example, your template should work for a Home with 2 light bulbs or 10 light bulbs, etc.

8- **[45 points]**Add a post endpoint to your application that gets all required information to create a new LightBulb object (15 points). Reusing your implementation from Assignment 2/Part 3, save the LighbBulb object to a JSON file called light.json (10 points). Return a proper message to the client (caller of this endpoint) (5 points).
9- **[15 points]** Test your endpoint in Step 8 with a curl command and take a screenshot of your terminal showing what you did. Include this screenshot to your repository while submitting your code.

Submission Guidelines:

Push your application folder in Part 2 as your assignment submission to your repository. Since you are going to use Part 1 in Part 2, you do not need to separate part 1 from part 2. But, if you wish, you can still do it that way.

Your repository should have your application folder that includes:

1- Your application code (assignment_3.py)
2- templates folder and home.html
3- model folder and smart_devices.py in it. I will grade Part 1 based on smart_devices.py file.
4- light.json that is created while you are testing your application.
5- The curl command screenshot (as described in Step 9 of Part 2).
6- If you get help from any AI-powered tool, submit a separate txt or docx file indicating how you used them.

Commit your code to your own repository. Make sure you have pushed your changes so that I can see them for grading. Make sure that you can see your code on GitHub.com before you think you are done.

Submit your repository link through Canvas. Even if you use the same repository for other assignments, you must perform this step.

Grading Policy:
Grades corresponding to each step shared above. Your submission will be graded based on each step and your implementation.