Homework 2: Object-Oriented Programming

Smart Home Automation System

Due Date: February 4, 2025

Objective

The goal of this assignment is to design a simple smart home automation system using object-oriented programming concepts in Python. You will model different smart home devices (lights, thermostats, and speakers) using classes, and allow them to interact with each other through composition and inheritance. You will also implement magic methods (and operator overloading) to simulate a realistic smart home system.

Terminology Review

- **inheritance**: A child class inherits from its parent (class Child(Parent)) to get access to all attributes of the parent. Represents is-a relationships. Generally, the child class specializes the parent class through one or more of:
  - **overloading**: A child class re-defines a method, so instances of the child will use that version of the method instead.
  - **extending**: A child class adds variables or methods not available to the parent.
- **composition**: Objects of another class are added to objects of this class, often during initialization. Represents *has a* relationships.
- **instance** or **bound** variables: Variables that are bound to an instance of this class. Example: variables you add during `__init__`. Accessed via dot notation: `my_obj.var1`.
- **instance** or **bound** methods: Methods that are bound to an instance of this class. Their first parameter is `self`: `my_obj.foo()` calls the bound method `foo()`, passing in `my_obj` as the first parameter, `self`.
- **private** variables or methods: Names that begin with one underscore are "private" - these should not be accessed outside of the class they are defined in (e.g. a user should not call `my_obj._foo()`).
- **magic** or **dunder** methods: Methods that begin and end with double underscores (`__`). Called "magic" because they are called in "magic" ways, e.g. `x + y` instead of `x.__add__(y)`. These are not "private," but you should not call them by their names (e.g. use `len(L)` instead of `L.__len__()`).

Requirements

Part 1: Basic Class Design

1. SmartDevice Class (Base Class)

Attributes:

- `name` (string): The name of the device (e.g., "Living Room Light").
- `status` (boolean): The on/off status of the device (True = on, False = off). Default status should be `False`.

Methods:

- `__init__(self, name)`:
  - Inputs: `name` (string) — The name of the device (e.g., "Living Room Light").
  - Returns: None.
  - Description: Initializes the device with a given name and sets the `status` to `False` (off) by default.
- `turn_on(self)`:
  - Inputs & Returns: None.
  - Description: Turns the device on by setting the `status` attribute to `True`.
- `turn_off(self)`:
  - Inputs & Returns: None.
  - Description: Turns the device off by setting the `status` attribute to `False`.
- `__str__(self)`:
  - Inputs: None.
  - Returns: `str` — A string that includes the device name and its current status (i.e., "Living Room Light: ON" or "Living Room Light: OFF").
  - Description: Returns a string representation of the device, showing the name and its current on/off status.

2. Light Class (Inheriting from SmartDevice)

Attributes:

- `brightness` (integer): Represents the brightness level of the light (1 to 100). Default value should be 100.

Methods:

- `adjust_brightness(self, level)`:
  - Inputs: `level` (integer) — The desired brightness level (between 1 and 100).
  - Returns: None.

- Description: Adjusts the brightness of the light. The brightness is only changed if the level is between 1 and 100 (inclusive). If the level is outside this range, the brightness remains unchanged.

- `__str__(self)`:
  - Inputs: None.
  - Returns: `str` — A string that includes the device name, its on/off status, and the brightness level (e.g., "Living Room Light: ON, Brightness: 75").
  - Description: Returns a string representation of the light, showing the name, its current on/off status, and its brightness level.

3. Thermostat Class (Inheriting from SmartDevice)
Attributes:

- `temperature` (float): The current temperature setting (in degrees Fahrenheit). Default value should be 65.

Methods:

- `adjust_temperature(self, temp)`:
  - Inputs: `temp` (float) — The desired temperature setting (in degrees Fahrenheit).
  - Returns: None.
  - Description: Adjusts the temperature of the thermostat. It uses the private helper method `_check_temperature_limits` to ensure the temperature is within a reasonable range (e.g., 55°F to 80°F). If the temperature is within this range, it updates the `temperature` attribute. If not, it leaves the temperature unchanged.

- `__str__(self)`:
  - Inputs: None.
  - Returns: `str` — A string that includes the device name, its on/off status, and the current temperature (e.g., "Thermostat: ON, Temperature: 68").
  - Description: Returns a string representation of the thermostat, showing the name, its current on/off status, and the current temperature.

- `_check_temperature_limits(self, temp)`:
  - Inputs: `temp` (float) — The desired temperature setting.
  - Returns: `bool` — `True` if the temperature is within the valid range (e.g., between 55°F and 80°F); `False` otherwise.
  - Description: Checks if the given temperature is within the acceptable range. This is a **private method** used internally to validate temperature adjustments.

4. Speaker Class (Inheriting from SmartDevice)
Attributes:

- `volume` (integer): Represents the volume of the speaker (1 to 10). Default value should be 3.

Methods:

- `increase_volume(self)`:
  - Inputs & Returns: None.
  - Description: Increases the volume by 1, with a maximum volume of 10.
- `decrease_volume(self)`:
  - Inputs & Returns: None.
  - Description: Decreases the volume by 1, with a minimum volume of 1.
- `__str__(self)`:
  - Inputs: None.
  - Returns: `str` — A string that includes the device name, its on/off status, and the volume setting (e.g., "Outdoor Speaker: OFF, Volume: 5").
  - Description: Returns a string representation of the speaker, showing the name, its current on/off status, and its volume setting.

Part 2: Composition

5. SmartHome Class

Attributes:

- `devices` (list): A list that stores instances of `SmartDevice` (lights, thermostats, cameras).

Methods:

- `__add__(self, other)`:
  - Inputs: `other` (SmartDevice) — An instance of a `SmartDevice` (e.g., `Light`, `Thermostat`, or `Speaker`).
  - Returns: None.
  - Description: Overloads the `+` operator to allow devices to be added to the `SmartHome` instance. This method appends the device to the `devices` list in the `SmartHome`.
  - Example: `my_house + hall_light` should add `hall_light` to the list of devices associated with `my_house`.
- `turn_off_all(self)`:
  - Inputs & Returns: None.
  - Description: Turns off all devices listed for the smart home instance.
- `__str__(self)`:
  - Inputs: None.
  - Returns: `str` — A string representation of the statuses of all devices in the smart home, showing the name and current status of each device (e.g., "Living Room Light: ON, Bedroom Thermostat: OFF").
  - Description: Returns a string listing the name and status of each device in the `SmartHome` instance.

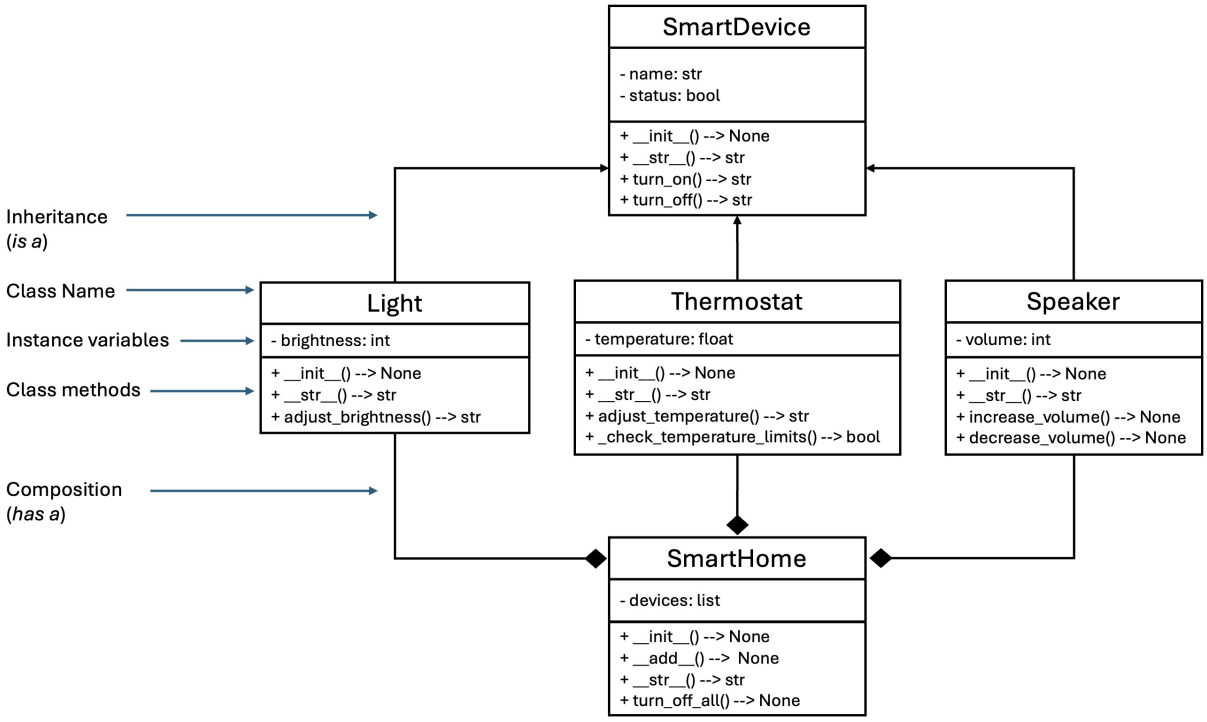

Figure 1: Class diagram including SmartDevice, Light, Thermostat, Speaker, and SmartHome. Note the inheritance and composition relationships.

Requirements and Submission

- Submit a file called `smart_home.py`. Students must submit to Gradescope individually within 24 hours of the due date to receive credit.
- In addition to automatically testing feature specifications on Gradescope, we will manually grade *readability* and *structure* on this assignment. Here are some hints:
  - Every method should have a docstring.
  - Use appropriate names for variables and functions.
  - Use white space—both within lines and between lines.
  - Include an *appropriate* number of comments. You probably won't need too many if you follow the previous three items.
  - Don't repeat yourself—use variables, functions, inheritance, etc. when appropriate.