Python Assignment

Class Descriptions:

Device Class

Description: Abstract base class for representing a generic device.

Attribute*:

- `device_id` (int): A static variable to keep track of device IDs.
- `mode` (Mode): The current operating mode of the device (ON or OFF).

Methods:

- `__init__(self)`: Initializes a new Device instance with a unique ID and OFF mode.
- `__str__(self) -> str`: Returns a string representation of the Device in the format: "(ID:device_id): mode".
- `turn_on(self)`: Turns the device ON, changing its mode to 'ON'.
- `turn_off(self)`: Turns the device OFF, changing its mode to 'OFF'.
- `get_mode(self) -> str`: Returns the current mode of the device as a string ('ON' or 'OFF').
- `toggle_status(self)`: Switches the device's mode.
- `@abstractmethod handle_events(self)`: An abstract method that should be implemented by subclasses to handle device-specific events.

Enums:

- `Mode (Enum)`: An enumeration representing the possible operating modes of the device.
  - `Mode.ON`: Represents the ON mode.
  - `Mode.OFF`: Represents the OFF mode.

SmartLight Class

Description: A subclass of the Device class, representing smart light devices.

Attributes:

- `brightness` (int): The current brightness level of the smart light.

Methods:

- `__init__(self)`: Initializes a new SmartLight instance with a unique ID, random brightness value, and turns it ON.

- `handle_events(self)`: Handles events for the smart light, adjusting its brightness based on the current mode. If the brightness falls below 40, it automatically turns ON; if it goes above 80, it turns OFF.

- `__str__(self) -> str`: Returns a string representation of the SmartLights in the format: "Smart light name (ID:device_id): mode, current brightness: brightness".

- `get_brightness(self) -> int`: Returns the current brightness level of the smart light.

- `set_brightness(self, value:int)`: Sets the brightness value to the number given.

Inherited Attributes and Methods: All the methods, attributes of the Device class.

Thermostat Class

Description: A subclass of the Device class, representing thermostat devices.

Attributes:

- `temperature` (float): The current temperature set on the thermostat.

Methods:

- `__init__(self)`: Initializes a new Thermostat instance with a unique ID, and a random initial temperature value.

- `handle_events(self)`: Handles events for the thermostat, adjusting the temperature based on the current mode and random factors. If the temperature falls below 18°C, it automatically turns ON; if it goes above 24°C, it turns OFF.

- `__str__(self) -> str`: Returns a string representation of the Thermostat in the format: "Thermostat (ID:device_id): mode, current temperature: temperature".

- `get_temperature(self) -> float`: Returns the current temperature set on the thermostat.

- `set_temperature(self, value:int)`: Sets the temperature value to the number given.

Inherited Attributes and Methods: All the methods, attributes of the Device class.

SecurityCamera Class

Description: A subclass of the Device class, representing security camera devices.

Attributes:

- `motion_detected` (bool): A flag indicating whether motion has been detected.

Methods:

- `__init__(self)`: Initializes a new Security camera instance with a unique ID and turns it on.

- `handle_events(self)`: Handles events for the security camera, simulating motion detection and triggering alarms.

- `__str__(self) -> str`: Returns a string representation of the Security Camera in the format: "Security camera (ID:device_id): mode, motion: motion_detected".

- `send_alarm(self)`: Simulates sending a notification when motion is detected.

- `get_motion_detected (self) -> bool`: Returns whether motion has been detected by the security camera.

- `detect_motion(self)`: Turns the motion_detected value to True.

Inherited Attributes and Methods: All the methods, attributes of the Device class.

SmartWindow Class

Description: A subclass of the Device class, representing smart window devices.

Attributes:

- `air_purity` (int): The level of how polluted the air is.

Methods:

- `__init__(self)`: Initializes a new SmartWindow instance with a unique ID, and a random initial for the air condition.

- `handle_events(self)`: Handles events for the smart window by checking the air quality. If the quality of the air exceeds a threshold, the smart window opens and lets fresh air in.

- `__str__(self) -> str`: Returns a string representation of the SmartWindow in the format: "Smart Window (ID:device_id): mode, air purity: air_purity".

- `polluting_air(self, num: int)`: Simulates the introduction of bacteria into the air and decreases the quality of the air. If the level exceeds 0, it's capped at 0.

- `letting_fresh_air_in(self)`: Simulates the open smart window. The increase is random but capped at 100.

- `get_air_purity(self) -> int`: Returns the current level of air quality.

- `set_purity(self, num:int)`: Sets the air_purity value to the number given.

Inherited Attributes and Methods: All the methods, attributes of the Device class.

CoffeeMachine Class

Description: A subclass of the Device class, representing coffee machine devices.

Attributes:

- `order_recognized` (bool): A flag indicating whether the voice command has been recognized.

Methods:

- `__init__(self)`: Initializes a new Coffee Machine instance with a unique ID and turns it on.

- `handle_events(self)`: Handles events for the coffee machine, simulating order detection and triggering coffee brew.

- `__str__(self) -> str`: Returns a string representation of the Coffee Machine in the format: "Coffee Machine (ID:device_id): mode, brew: order_recognized".

- `send_feedback(self)`: Simulates sending a notification when an order is received.

- `get_order_recognized (self) -> bool`: Returns whether an order has been processed by the coffee machine.

- `recognize_order(self)`: Turns order_recognized value to True.

Inherited Attributes and Methods: All the methods, attributes of the Device class.

How to Use the Dashboard + Automation Rules

After starting the application, there are already one of each device implemented. By clicking the buttons and adjusting the sliders, you can change the devices' values. Automation rules are included in SmartLight, Thermostat, and SmartWindows objects' `handle_events` functions.

Test Cases

Written in test.py