

Practice 4 – Smart Home System

Błażej Drozd

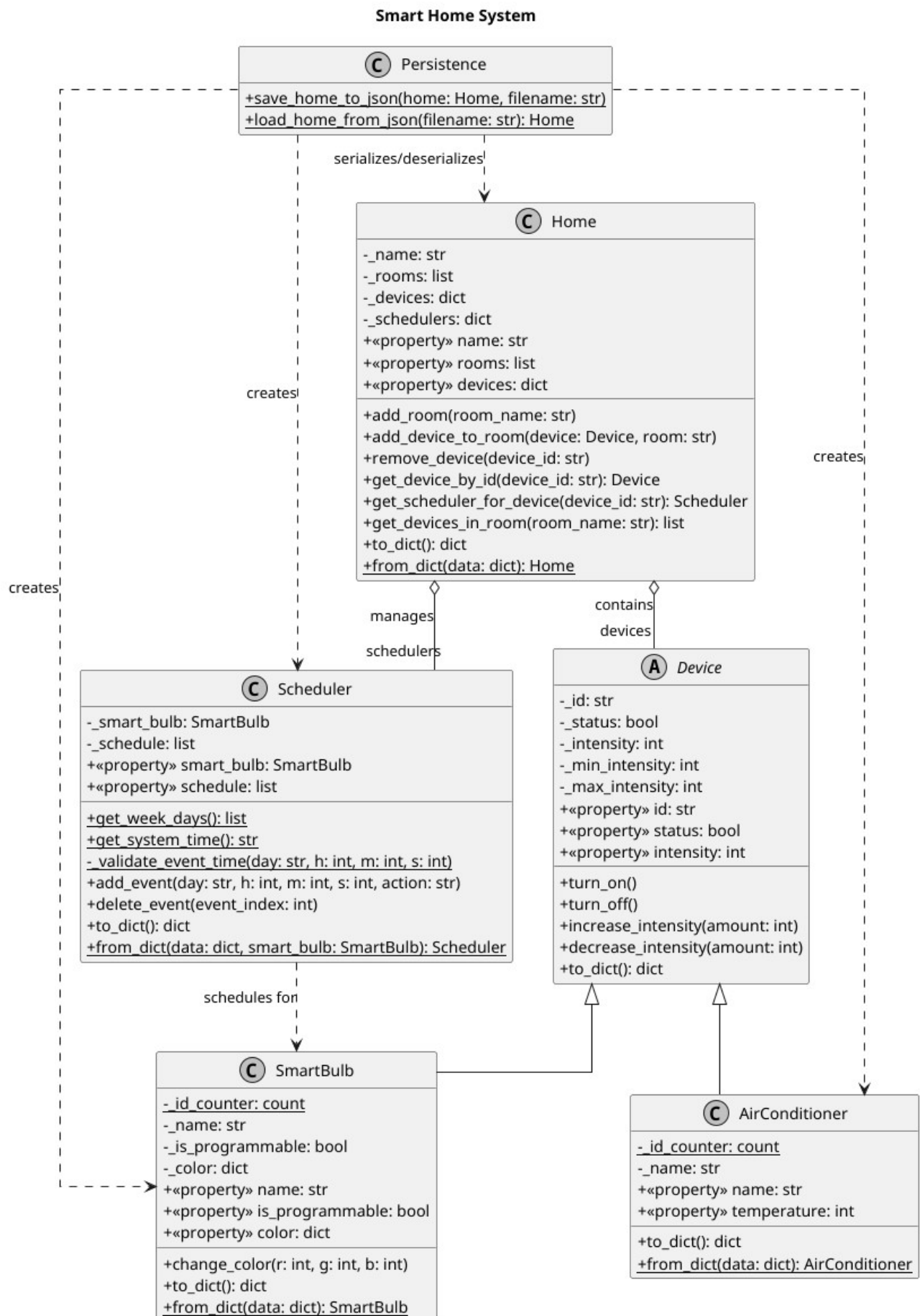
20.11.2025

Project repository: <https://github.com/Tsugumik/programacion-uja>

Changes

- A Device base class was introduced to hold common logic (like status, intensity, and on/off methods). SmartBulb and AirConditioner now inherit from Device, which eliminates code duplication and improves the object-oriented design.
- Implemented a new persistence.py module to save and load the application's state to a JSON file. All core classes (Home, Device subclasses, Scheduler) now support serialization via to_dict() and from_dict() methods. This makes the home's configuration persistent between sessions.
- Based on feedback, I have also reviewed the project. Two methods in the Scheduler class, which don't depend on instance state, have been correctly changed to class methods.

UML Diagram



The repository also includes a `diagram.puml` file containing PlantUML diagram for the entire system, illustrating the relationship between classes.

User Stories Validation

Also I verified that the current implementation of the Smart Home meets all the criteria defined in the user stories.

- HU01: Smart Bulb Management
 - Can be turned on and off:
 - Fulfilled: Implemented via `SmartBulb.turn_on()` and `SmartBulb.turn_off()` methods (inherited from the Device base class).
 - Know its status:
 - Fulfilled: Implemented via the `@property SmartBulb.status` (inherited from Device).
 - Change the intensity:
 - Fulfilled: Implemented via `SmartBulb.increase_intensity()` and `SmartBulb.decrease_intensity()` methods (inherited from Device).
 - Change the color:
 - Fulfilled: Implemented via the `SmartBulb.change_color(r, g, b)` method, which validates and sets the bulb's RGB color. The current color is accessible via the `@property SmartBulb.color`.
- HU02: Air Conditioner Management
 - Be able to know the air temperature:
 - Fulfilled: Implemented via the `@property AirConditioner.temperature`. For this device, 'intensity' represents temperature.
 - Be able to change the temperature:
 - Fulfilled: Implemented via `AirConditioner.increase_intensity()` and `AirConditioner.decrease_intensity()`.
 - Be able to know if it is off or on:
 - Fulfilled: Implemented via the `@property AirConditioner.status`.
 - Be able to turn it off or on:
 - Fulfilled: Implemented via `AirConditioner.turn_on()` and `AirConditioner.turn_off()`.
- HU03: Device Distribution in the Home
 - Be able to tell how many rooms the home has and what rooms they are:
 - Fulfilled: Implemented in the Home class via the `@property Home.rooms`.
 - Be able to tell what devices are in the home:
 - Fulfilled: Implemented in the Home class via the `@property Home.devices`.

- Be able to add a device:
 - Fulfilled: Implemented via `Home.add_device_to_room(...)`.
- Be able to remove a device:
 - Fulfilled: Implemented via a `Home.remove_device(...)` method.
- Be able to modify a device:
 - Fulfilled: Implemented indirectly. The user (via `main.py`) can retrieve a device (e.g., `home.get_device_by_id(...)`) and then call its public methods (`.turn_on()`, `.change_color()`, `.increase_intensity()`).
- Be able to know the number of devices in the home and in each room:
 - Fulfilled: Implemented in the `Home` class via `len(home.devices)` and the `get_devices_in_room(...)` method.
- Be able to identify the device that is in each location:
 - Fulfilled: Ensured by the unique device ID and the `Home` data structure, which maps rooms to device lists.
- HU04: Permanent Storage of the Home's Rooms
 - Can be recovered from a file
 - Fulfilled: Implemented by the `persistence.load_home_from_json(filename)` function. This function uses the `@classmethod from_dict()` methods defined in the `Home`, `SmartBulb`, and `AirConditioner` classes to restore the complete object state.
 - Can be saved to a file
 - Fulfilled: Implemented by the `persistence.save_home_to_json(home_object, filename)` function. This function uses the `to_dict()` methods defined in the classes to serialize the object state to JSON.