Professorship of Embedded Systems and Internet of Things
Department of Electrical and Computer Engineering
Technical University of Munich

Homework 1

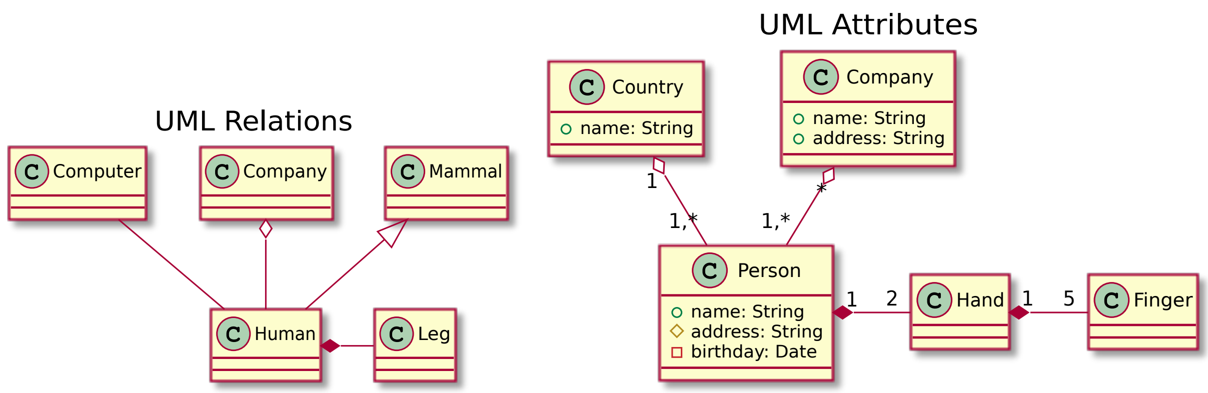
Software Architecture for Distributed Embedded Systems, WS 2021/22

Prof. Steinhorst, M. Sc. Emanuel Regnath, M. Sc. Jens Ernstberger

Please submit your UML solutions as PlantUML code (`.pu`) *and* SVG image. All textual answers should be placed in additional `.md`, or `.txt` file. Please use our PlantUML template `plantuml-template.pu` as starting point.

? Exercise 1.1: UML Class Diagram Basics

UML class diagrams are useful for designing a system architecture. To better understand UML class relations, attributes, and multiplicity, we have the following examples:



1. Similar to the figure above left, create a UML class diagram with the four relation types (*association, aggregation, composition, inheritance*) that share a common class. Your task is to find another example, so do not use `Human`, `Person`, or similar as the common class name. Please submit `hw111-relations.pu` and `hw111-relations.svg`.
2. Similar to the figure above right, create an UML class diagram with different multiplicities (*—*, x—*, x—y) and have one class that has all three attribute types (public, protected, private). Please submit `hw112-attributes.pu` and `hw112-attributes.svg`.

? Exercise 1.2: Designing a Smart Light System

Smart lighting is an essential system within smart homes that enable the user to save energy and have fine-grained control of various intelligent lights. A smartphone app can the define how the lighting system should behave based on switches, sensors or other events. Your task is to design the system architecture and provide a UML class diagram that is clear and readable, open for extension, and covers the functionality that is described in following requirements.

System Description

We have two types of smart lights: A `SwitchLight`, which can be either on or off and a `DimmLight`, which can also set a brightness level between 0 and 100 percent when it is switched on. A DimmLight will also remember the last used brightness level the next time it is switched on.

Individual lights can be grouped together such that any command affects the entire group. For example, the user might want to create a group “Dinning Table” including three DimmLights. When the user sets a brightness level for this group, each light in the group will be set to the same brightness level. Groups must contain individual lights and cannot be nested (include other groups). In addition to `groups`, the user should be able to create `scenes`. A scene is basically a group of lights with individual light settings. For example, the user might create a scene “Cooking”, which switches on the kitchen light, sets it to maximum brightness and switches off the three lights at the dinner table (or sets their brightness individually). Scenes can only contain individual lights and their setting. Scenes cannot be nested meaning they cannot include sub-scenes or groups. Scenes can only be activated but not deactivated or reversed (you do not need to remember previous states).

Besides lights, we have one type of sensor in our system: A `DayLightSensor`, which reports the outside ambient brightness in Lux to a smartphone app.

Smart lights, groups, and scenes can be controlled in three different ways: 1. by using hardware switches, 2. by automated events (e.g. ambient brightness, time of day), 3. by clicking a button in a smartphone application `ESI-App`. Hardware switches always have two buttons that are pressed and trigger an event, e.g. one button will switch a light or group on while the other switches the light or group off. In general, the user can define in the `ESI-App` which input events will trigger which light, group, or scene.

The user can create groups and scenes in `ESI-App` and see the different states of each light. As mentioned before, the user can define triggers, which consist of a simple integer condition ($=, <, >$) and an action. The condition simply compares an input value (e.g. sensor) to a numerical literal. The action can set the state of light or group or activate a scene.

Note: The names and numbers given in the example show a potential use case and do not reflect actual names or numbers of the system architecture. So you should not create a class `Cooking` but a scene should have a `name` attribute.

General Working Advice

Your UML diagrams should cover the high-level system architecture focusing on the entities mentioned above. The three core aspects are 1. modeling the different real world objects, 2. the relations and interactions between individual lights, groups, and scenes, 3. mapping events (switch on) to actions (group on). Provide all attributes and functions that are necessary to understand what a class can do (function) and remember (attribute). You may also introduce any classes that are related to the core classes and help to abstract or manage other classes.

There are two aspects that you do not need (and should not) include in your design:

- **GUI:** assume that once your class has a attribute it can in principle be displayed. You do not need to add any functions or create classes like `Display` or `WindowManager`.
- **Network:** We assume devices can identify and talk to each other by an ID (e.g. name/number). You can give any class a `send(id, msg)` and `recv(id, msg)` function. The send function can send any attribute or class of the owning class to any device. Also any class with the `recv` function can receive any state of any device. Authentication is handled automatically. So do not create classes like `Connection`, or `NetworkManager`.

Note: While in general you can think of the `Light` class as the device itself, you should prefer to think of it as a representation from the viewpoint of the smartphone app `ESI-App`. That means the `on()` function of the `Light` class might change a state or send a message to the real light but not toggle a GPIO pin. Essentially, the functions will be executed on the smartphone.

Your Tasks

1. List the core entities that should become classes from the description above. Submit `hw121-classes.txt` or `hw121-classes.md`.
2. Which of the core entities are real world objects? Draw a UML class diagram that shows the relation and possible abstraction of all real world objects. Submit `hw122-objects.pu` and `hw122-objects.svg`.
3. How could you model the relation between individual lights, groups, and scenes? Draw a UML class diagram that models only lights, groups, and scenes. Submit `hw123-groups.pu` and `hw123-groups.svg`.
4. Draw a UML class diagram that models the full system architecture. Your final diagram should have between 8 and 24 classes. Submit `hw124-system.pu` and `hw124-system.svg`.
5. Explain shortly your design decisions, e.g. why you chose certain relations or patterns. Please also submit `hw125-explanation.txt` or `hw125-explanation.md` with max. 300 words.