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

Exercise Sheet 1 – Solution

Software Architecture for Distributed Embedded Systems, WS 2020/21

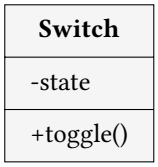
Prof. Steinhorst, Dr.-Ing. Mohammad Hamad, M. Sc. Emanuel Regnath

? Exercise 1.1: Unified Modeling Language (UML)

Together with some friends you have founded "AwesomeLights", a Startup that wants to provide a decentralized indoor lighting solution using smart IoT devices. In contrast to conventional indoor lighting, your system offers to plug-in new switches and lights on the fly and allows dynamic reconfiguration using a smartphone app. Your first task in this new company is to create an UML model of the whole system. In order to learn UML, you decide to solve the following problem questions.

1. You realize that a UML class diagram is about drawing boxes and connecting them with lines. Which information is specified in the 3 sections of a class box? Give an example for each section when modeling a light switch.

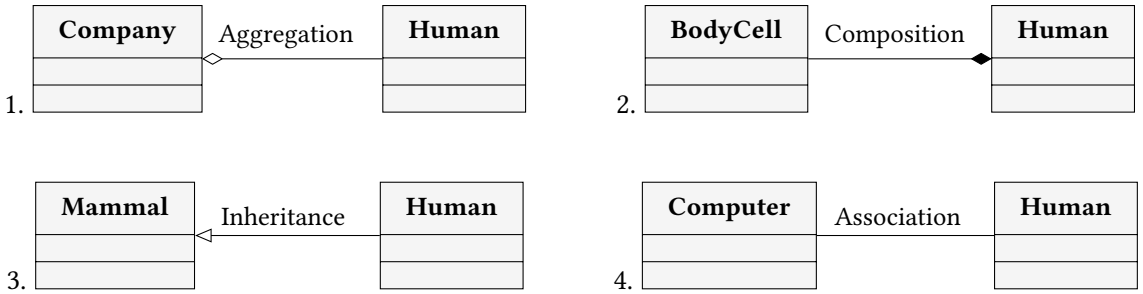
Solution:



1. Class name: e.g. **Switch**
2. attributes: e.g `state [bool]`
3. methods: e.g. `toggle()`

2. Which four types of object relations are shown in the following? Name each relation and give an example for the classes A and B that would have this relation. Use the class `Human` as A or B in each example. What is the difference between the first two relations?

Solution:



Aggregation and Composition are specific cases of an Association. In an aggregation the child C can exist independently of the parent P ("P owns a C"). In a composition, the child only exists when the parent exists ("Every P is made of C"). If you remove P, you automatically remove C.

Summary from weakest to strongest relation:

- Association: “P may use a C. C may belong to P”
- Aggregation: “P owns a C. C is a part of P”
- Composition: “P is made of C. C is part of every P”
- Inheritance: “P is the class of C. Every C *is a* P.”

3. How are private, public and protected attributes represented in a class diagram?

Solution:

- Public: using a `'+'` e.g. `+name`
- Protected: using a `'#'` e.g. `#address`
- Private: using a `'-'` e.g. `-secret`

4. Which notations specify valid ranges of multiplicity?

◯ 0..	◯ 3-5*	◯ 3...*
◯ -3	√ **1..2**	√ **3..3**
◯ *4	◯ 3,*	√ **3,4,6**

? Exercise 1.2: Object Oriented Programming

As a first prototype, you try to program a centralized IoT controller, which should control the smart light bulbs. You start with one light bulb and two buttons that can only be temporarily pressed but not switch between two states. In order to avoid spaghetti code, you decide to learn Object Oriented Programming (OOP).

1. What are the four main concepts of object oriented programming?

> **Solution:** Remember: **A Pie**, but more relevant is this order:
>
> - **A**bstraction: extract common features, remove details, identify objects
> - **E**ncapsulation: pack attributes and methods into objects, access methods via interfaces
> - **I**nheritance: Express "is a" relationship between objects
> - **P**olymorphism: overwrite inherited methods to provide different behaviors

2. What is the difference between the terms: *class*, *data structure*, and *type*? Give a short definition.

> **Solution:**
>
> - Class: object description with attributes and methods that can be public or private
> - Data structure: object description using only public attributes (data fields)
> - Type: general description of a data format. Could describe an object.

3. What is the difference between *class*, *object*, *instance*, and *reference*? Give a short description and assign these (maybe several) terms to each identifier in the following code:

```
class LightSource()
class LED(LightSource) # inherit from LightSource
green_led = LED()
red_led  = LED()
power_led = red_led
```

> **Solution:**
>
> - Class: the description of a group of things you want to model, e.g. "LED: Any diode that can produce light". In programming: Description of the data format.
> - Object: a semantic group of existing things that represent one specific class, e.g. "every LED that exists". In programming:
> - Instance: one specific existing thing, which is an object, e.g. "this green LED on that Raspberry". In programming: a memory region that holds the data in the layout described by a class.
> - Reference: an identifier that refers to an instance, e.g. "Status LED". In programming: memory address of an instance.

```
LightSource    # class
LED            # class
green_led      # object, instance of LED and LightSource
red_led        # object, instance of LED and LightSource
power_led      # reference to red_led instance
```

Object is a semantic concept, *instance* is an existing thing. The `green_led` and `red_led` are two separate instances of LED-objects described by the class `LED` and they are also instances of the class `LightSource`. However, they are not `LightSource` objects.

4. Unfortunately, while you were studying OOP concepts, your co-worker (who wants to stay anonymous) has already implemented the first draft over night and pushed it to the git repository. Look at the following bad code and identify how it could be improved with OOP concepts.

   Rewrite the code and declare the 3 classes `Controller`, `Light` and `Button`. Either give a high-level code on paper or refactor the full code file, which is uploaded on Moodle (**recommended**).

```
def main():
    print("Starting controller...")
    init() # controller
    init_light_and_buttons()
    light_state = OFF  # state of the light

    print("Entering main loop:")
    while True:
        (b1, b2) = receive_button_states(2)
        first_button_state = b1
        second_button_state = b2

        if first_button_state == PRESSED:
            if light_state == OFF:
                kitchen_light_on() # power light bulb
                light_state = ON
            else:
                kitchen_light_off()
                light_state = OFF

        if second_button_state == PRESSED:
            if light_state == OFF:
                kitchen_light_on()
                light_state = ON
            else:
                kitchen_light_off()
                light_state = OFF
```

Solution:

Use classes for light and buttons, encapsulate states in classes. Add lights and buttons to the controller class.

```
class Button;
class Light;

class Controller:
    def run(self):
        print("Running control loop:")
        while True:

            # read button presses
```

```
            bstates = receive_button_states(len(self.buttons))
            for idx, button in enumerate(self.buttons):
                button.read_state(bstates[idx])

            # switch lights
            for idx, button in enumerate(self.buttons):
                if button.is_pressed():
                    self.__switch_all_lights()


def main():

    # create lights
    l1 = Light("Kitchen")
    l2 = Light("Living Room")

    # create buttons
    b1 = Button("Door")
    b2 = Button("Window")
    b3 = Button("Desk")

    # create controller
    ctrl = Controller()
    ctrl.init()

    # add
    ctrl.add_light(l1)
    ctrl.add_light(l2)

    ctrl.add_button(b1)
    ctrl.add_button(b2)
    ctrl.add_button(b3)

    # run
    ctrl.run()
}
```

Python

Since we will use python for our code examples and tutorials, it might be the perfect opportunity for you to learn Python or improve your Python skills. Take a look at the Python in 10 minutes tutorial for the basics or Python Class and Objects for the OOP concepts of Python.