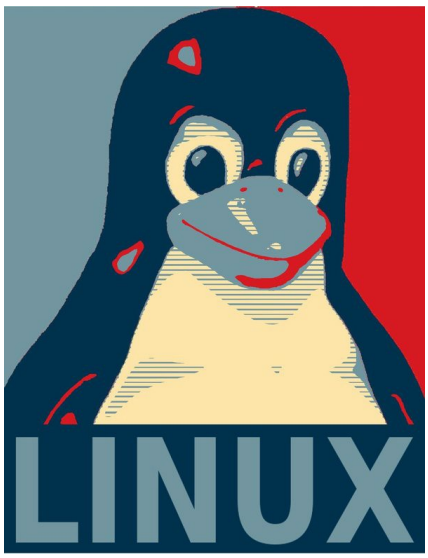
OO Analysis and OO Design and Implementation Assignment

SD Sem3



Part 1 (Static Design): Elevator

A note before you start

This assignment is about designing an elevator system. Software design is not about following rules that will lead you to a "correct" design. What is a "correct" design to begin with? A design is about choices, each choice has advantages and disadvantages. The best way to learning software design is to simply try something and ask feedback.

Please take time to do that: try something and ask your teacher for feedback. Try something else and ask more feedback. That's the best way of learning how to weigh the choices you have!

In this exercise you will design an elevator system based on the following user requirements (taken from (Douglass, 2005)):

A software system must control a set of eight Acme elevators for a building with 20 floors.

Each elevator contains a set of buttons, each corresponding to a desired floor. These are called floor request buttons, since they indicate a request to go to a specific floor. Each elevator also has a current floor indicator above the door.

Each floor has two buttons for requesting elevators called elevator request buttons, because they request an elevator. Each floor has a sliding door for each shaft arranged so that two door halves meet in the center when closed. When the elevator arrives at the floor, the door opens at the same time the door on the elevator opens. The floor does have both pressure and optical sensors to prevent closing when an obstacle is between the two door halves. If an obstruction is detected by either sensor, the door shall open. The door shall automatically close after a timeout period of 5 seconds after the door opens. The detection of an obstruction shall restart the door closure time after an obstruction is removed. There is a speaker on each floor that pings in response to the arrival of an elevator.

On each floor (except the highest and lowest), there are two elevator request buttons, one for UP and one for DOWN. On each floor, above each elevator door, there is an indicator specifying the floor that the elevator is currently at and another indicator for its current direction. The system shall respond to an elevator request by sending the nearest elevator that is either idle or already going in the requested direction. If no elevators are currently available, the request shall pend until an elevator meets the above-mentioned criterion. Once pressed, the request buttons are backlit to indicate that a request is pending. Pressing an elevator request button when a request for that direction is already pending shall have no effect. When an elevator arrives to handle the request, the backlight shall be removed. If the button is pressed when an elevator is on

the floor to handle the request (i.e., it is slated to go in the selected direction), then the door shall stop closing and the door closure timer shall be reset.

To enhance safety, a cable tension sensor monitors the tension on the cable controlling the elevator. In the event of a failure in which the measured tension falls below a critical value, then four external locking clamps connected to running tracks in the shaft stop the elevator and hold it in place.

Part 1.1: use case definition

Read the system description and answer the following questions:

- What is the primary function of the system
- What are the secondary functions of the system
- Why is this system being built? What is it replacing and why?
- What are the actors in this system?

Construct the use case(s) for the primary function(s), taking in account:

- The role the external objects (actors) and system play in each scenario
- The interaction (flows) necessary to complete the scenarios in the use case(s)
- The sequence of events and data to realise the scenario in the use case(s)
- What variations on the scenario are possible?

Draw a use case diagram using the constructed use case(s).

Part 1.2: working towards a class diagram

Please take the following steps:

1. Object discovery: read the system description and make a list of objects of relevance
2. Association discovery: draw an object diagram which lists the objects and their associations, use sequence diagrams where appropriate to find associations (you have to research object diagrams on your own)
3. Use the objects to define the classes you need, draw a first class diagram and define the responsibility of each class (usually defined in a table)

class 1	responsibility of class 1
class 2	responsibility of class 2

4. Complete your first class diagram from step 3. You don't need to use the appropriate relationships yet, however if you use inheritance you must indicate that correctly.
5. Define the required operations for each class.
6. Use sequence diagrams to verify the interactions between classes. Important: do not try to be complete in your sequence diagram: make a sequence diagram for a specific situation. E.g.: sequence diagram for trying to send a file in which the server does not accept the file.

Part 3: finalize your design

A design is more than just a collection of diagrams. A design is a collection of diagrams together with a (short) description of important decisions that were made in the design process.

You put all the diagrams together in a document and add as much text as you need to clarify your design. Your class diagram must be accompanied with a table that explains the responsibility of all classes. Sequence diagrams often have a short explanation of what happens in similar cases. Changes of your design are also documented accompanied with the arguments.

Part 2 (Dynamic Behaviour): Microwave

Part 2.1: implement and unit test the Microwave state diagram

Study the StateBehaviour presentation and design a state diagram for the given microwave oven. A start has been made in the provided code (read further). Your first task is to extend the behavior of the microwave by adding states and describing the added behavior is a document.

The product dir contains code for the given state, please study the implementation and implement the rest of your state diagram. The test dir contains unit tests with mocks for the given state, please unit test your implemented state well. Make sure you really understand:

- how state diagrams work,
- how you can implement them,
- how mocks work, and
- how you can test your states using mocks

Part 2.2: Hand in part 2

You hand in one ZIP file containing:

1. Documentation (in PDF) with:
   1.1. your state diagram(s) + description
   1.2. How you tested your code and what the results are
   1.3. Which known problems your implementation has
2. Make sure your code doesn't have dead code or unnecessary comments.
3. Documentation is delivered in PDF and code is delivered without object files, executables or IDE specific files present. Only source code and Makefile(s).