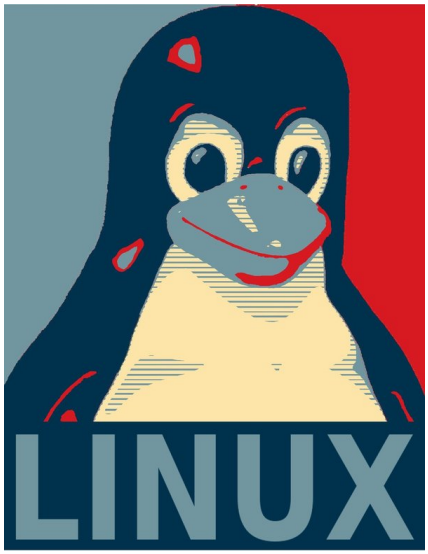
Design Challenge
Pound Lock



Design Challenge

A note before you start

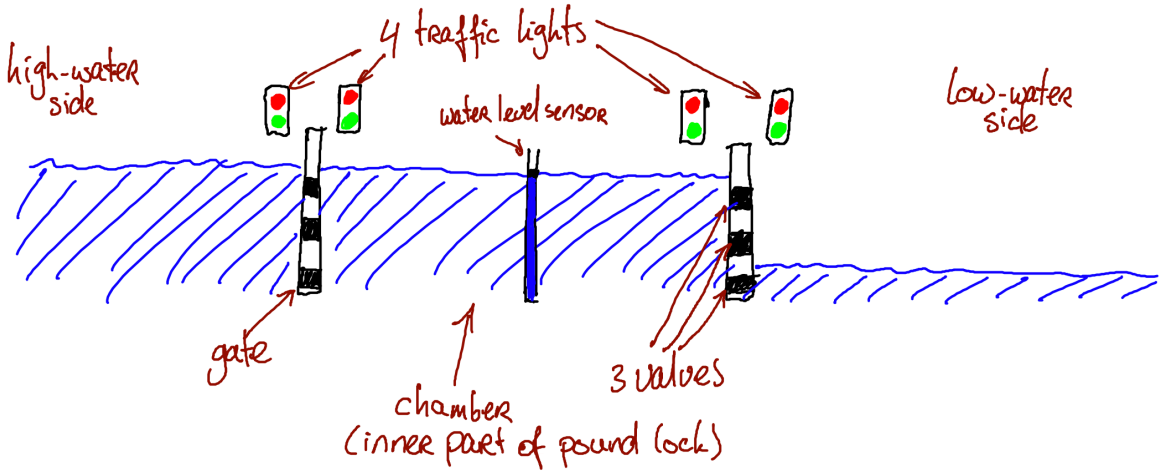
This challenge is about designing a pound lock system. Software design is not about following rules that will lead you to a "correct" design. What is a "correct" design to begin with? A design is about choices, each choice has advantages and disadvantages. The best way to learning software design is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

That is the reason for this challenge, a challenge in 2 parts that should both be ready for feedback in after one day of work. Don't worry if your design is not entirely finished, as long as you have enough to receive feedback on.

Part 1

In this challenge you will design a lock system based on the user requirements below. Please read this Wikipedia article first to get more information about pound locks in general.

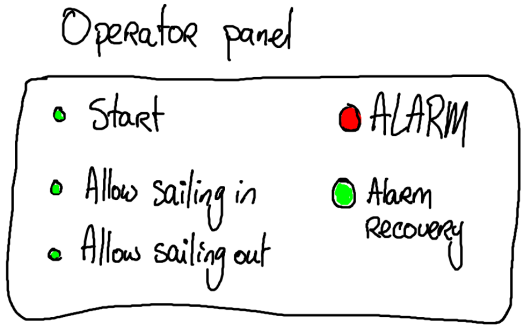
Functional Description



1. The operator decides whether or not to rise or lower the chamber water level, to rise or lower the water he presses the "start" button.
2. After pressing the “start” button, the opened gate (on the side where the water level is equal) will be closed. The gate is made of 2 watertight doors.
   2.1. If the boats came from the high water side, valves in the low water doors are opened so the chamber is drained downstream. Then the low-water side doors are opened.
   2.2. If the boats came from the low water side, the valves in the high water doors are opened so the chamber is filled using upstream water.
3. In order to prevent damage to the doors, there are traffic lights at the pound lock. The boats may only leave the lock when their traffic light is green. The traffic light will only turn green when the doors are fully opened and the operator has released the lock for sailing out. For this purpose, the lock operator has an "allow sailing out" button.

4. Similarly, there is a traffic light at each entrance to the pound lock. Boats may only enter the lock if their traffic light is green. For this, the lock operator has an "allow sailing in" button.
5. Traffic lights are red or green. The cycle is: red → green → red + green → red.
6. The gates are operated by a hydraulic system. This installation may only be activated to open the doors when the water level on both sides of the door is at the same level. The chamber has a water level sensor.
7. To regulate the water level in the chamber, three valves are installed in each door. The bottom valves are always below the chamber water level.
    7.1. If the water in the lock must rise, the bottom row on the high-water side is opened first, when the second row is under water, this row is opened too, and likewise the third one when it is under water. You must never open a valve when it is above the chamber water level!
    7.2. If the water in the chamber must be lowered, the lower valve is opened on the low-water side. The middle and top valves on the lower water side must never be opened.
8. In an emergency, e.g. if someone has fallen into the water, the lock operator can intervene by pressing an "alarm" button. If the doors are closed on both sides, the (possibly) opened valves in the doors are immediately closed in order to stabilise the falling or rising water level. If doors are opening or closing, the doors are stopped immediately.
9. The emergency situation is lifted by the lock operator by pressing an “alarm recovery" button. Normal operation is then resumed.
10. Gates should never be opened on two sides at the same time. Gates will fail if they receive an open or close command during opening or closing. In other words: doors may only receive an open or close command when they are not moving.
11. There is no detection of the number of vessels in the lock.
12. The water level sensor has 5 values, the sensor will always return the highest valid level (i.e.: if the water is in between the middle and higher gate, the sensor will return Above_middle_valve, even though Above_low_water is also true):
    12.1. High: water level is equal to the high-water side
    12.2. Above_high_valve: water level is above the highest door valve
    12.3. Above_middle_valve: water level is above the middle door valve
    12.4. Above_low_water: water level is higher than the low-water side, but lower than the middle valve
    12.5. Low: water level is equal to the low-water side

Operator user interface



Use case definition

Read the user requirements 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).

Working towards a class diagram

Please take the following steps:

1. Object discovery: read the user requirements and list the 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
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 2: State behaviour

In part 1 you have created a design for your pound lock. That part doesn't describe the state behaviour of your system. The pound lock system has various events: the operator panel buttons trigger an event and you can raise an event whenever the water level changes. Depending on the state you are in, you may want to react to some events or you may want to ignore certain events.

Your task is to determine which classes have state behaviour and to create the state diagram for that state machine. Beware: you will probably get multiple state diagrams that interact with each other: the interaction between state diagrams is always based on events. So the state machine in class A sends an event to a state machine in class B, etc.