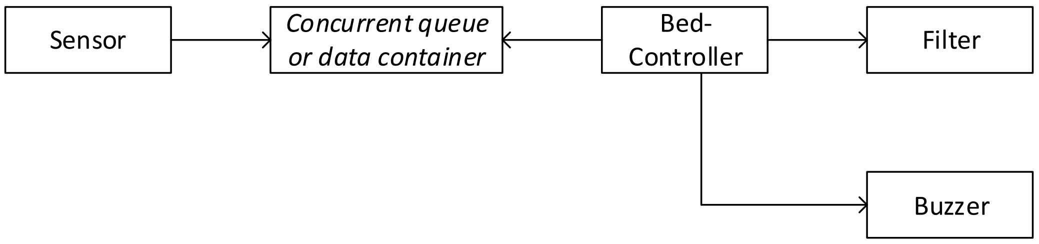
Exercise: Hospital Bed – GoF Strategy

In this exercise, you will continue to work with the Hospital bed, the exercise you have already started. We will extend the solution to use the GoF Strategy pattern to make the software more flexible, even run-time configurable.

Last week you worked on the hospital bed system which could detect the presence or absence of a patient in a hospital bed and sound an alarm (a buzzer) if the patient left the bed. You also implemented filtering of the sensor input so that you could be more confident that a patient was actually out of the bed before the alarm was sounded.

Today, you are going to refactor and extend your solution so that it becomes much more flexible. You are also going to implement new ways to alarm caretakers, and finally to make it very flexible to select how the filtering and alarming will take place. This is going to happen by means of the GoF Strategy pattern.

If you have finished the exercise, and if you have considered the Single Responsibility Principle in your solution, it probably looks something like the below[1]:



Exercise 1:
From our discussion in class you have learned that *filtering* is a candidate for application of the GoF Strategy pattern. By using GoF Strategy, your software can contain two types of filtering:

- "Raw",i.e. no, filtering
- Noise filtering, i.e. 3 consecutive, identical samples before the output is trusted.

Design and refactor your software so that filtering uses GoF Strategy, i.e. that it contains both ways of filtering. How hard would it be to implement and use a third type of filter?

Exercise 2:
Similarly, *alarming* is a candidate for application of GoF Strategy, so again: Design and refactor your application so that alarming uses GoF Strategy. Implement alarming in these two ways:

- A buzzer
- A light

How hard would it be to implement a third way of alarming, i.e. text/emailing caretakers?

Exercise 3:
Redesign your software so that the filtering and alarming methods ("strategies") are both configurable run-time, i.e. so that you can set the strategy for filtering/alarming runtime in the `BedController`.

Exercise 4:
Implement unit tests of `BedController` and your filters. Is it easy or hard to test these software components? Would it be easier or harder if filtering and alarming was an integral part of `BedController`?

---

[1] Maybe your classes are called something else. Maybe you have the filter before, not after the queue/data container. Both are just fine – as you know, there is no "right" solution here!