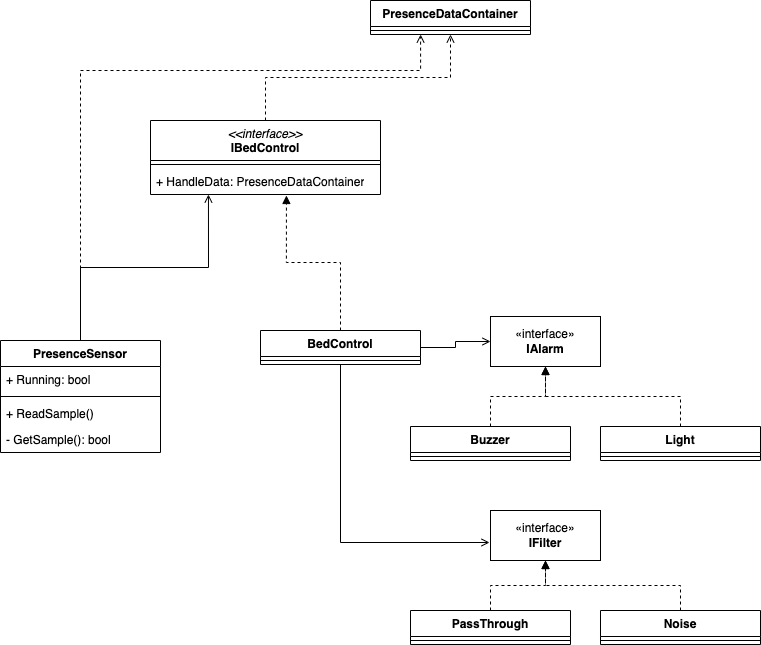
**Exercise: Hospital Bed, GoF Observer**

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 Observer pattern to make the software even more decoupled and extensible.

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 if the patient left the bed. You have implemented this using *GoF Strategy* to enforce OCP and make the filtering and alarm methods very flexible, even run-time configurable.

Today, you are going to refactor and extend your solution so that it becomes even more flexible and extensible by means of the *GoF Observer* pattern.

If you have finished the exercise so far, your design probably looks something like the below:



The objective of this exercise is to lower the coupling between the BedControl and the “alarming”, so that the BedControl does not directly sound or silence the alarm, but instead uses the GoF Observer pattern to notify any listeners that the filter output has changed. One of these listeners will be a new class Alarming that subscribes to the BedControl and is thus notified when the filter output changes. Alarming can then sound (or silence) the alarm.

**Exercise 1:**

Discuss exactly what this exercise requires you to do: How does alarming take place now, and how is it supposed to take place in the future?

**Exercise 2:**

Suggest a design that uses GoF Observer, so that the BedControl can raise a “Patient presence changed” event which Alarming can subscribe to and react upon.

**Exercise 3:**

Implement your design.

**Exercise 4:**

Extend your design and implementation to introduce a Log class. This class shall log (to screen or file) every time the patient leaves his/her bed. Discuss how this new functionality can be introduced into your design. What classes need to change (if any)? Is it easy or hard to introduce this change?