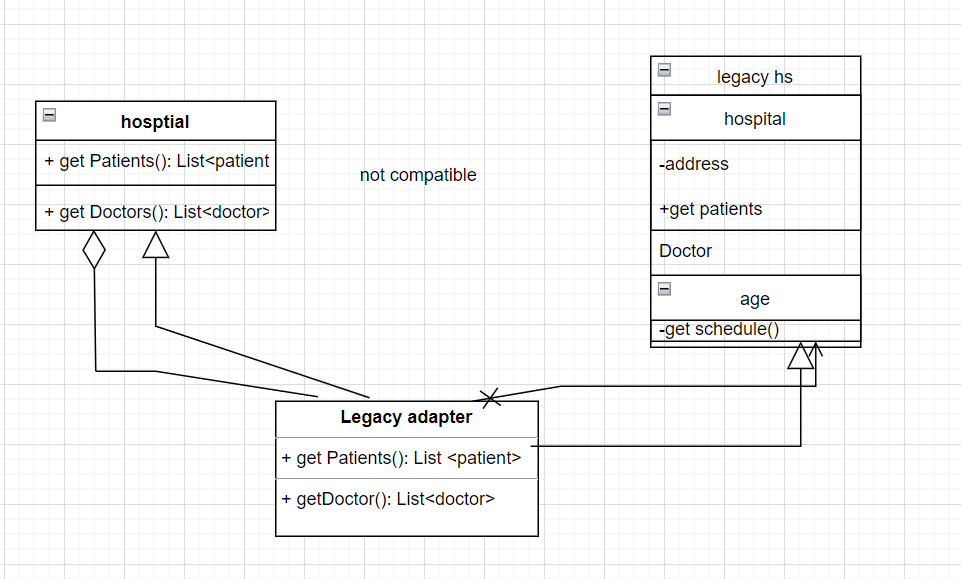
**Design patterns**

What is a design pattern?

A pattern describes a **problem that occurs over and over** again in our environment, and then **describes the core of the solution** to that problem in such a way that **you can use this** This pattern group aims to reduce coupling between classes, enabling future extensions (by introducing abstract class) and encapsulate complex structures**olution a million times** over, without ever doing it the same way, twice.



Consider this scenario:

*A health system has already been running for 100 years with stable in-place system.*

You are a new and modern tech division in the hospital, wants to create mobile app, needing API that consumes the data from legacy system as well as collecting new data from other sources

One way to connect those 2 systems is to create an adapter. In the UML diagram above, Legacy Adapter acts as a translator/connector between 2 systems. This can be done by implementing interfaces or extending parent class (and override some methods) from the new system and then make a delegation call (blue arrow) or calling the superclass method (inheritance green arrow) to get the value.

**Bridge Pattern on Medicine Procurement from Multiple Sources**

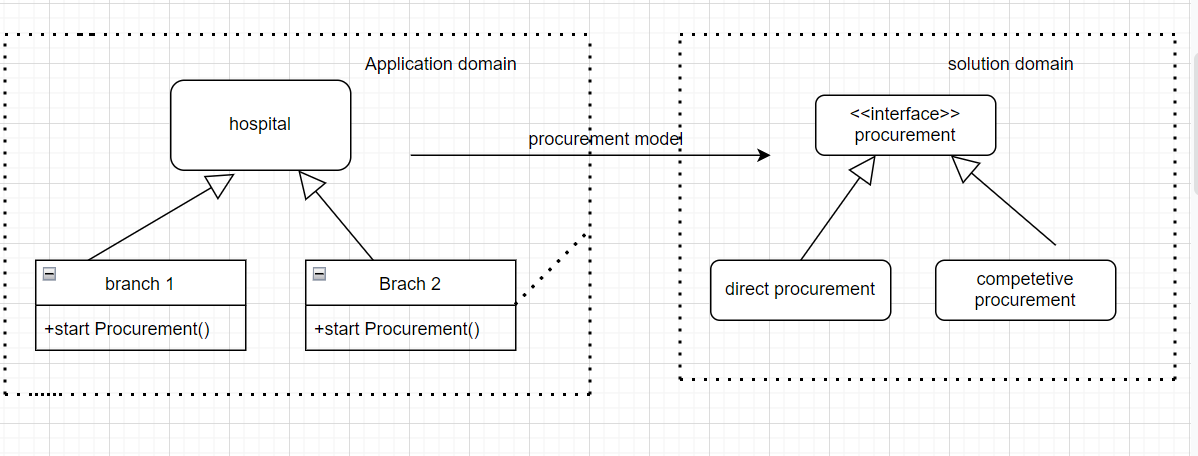
*It was found that the majority of health clinics reported using an expensive direct-procurement model for purchasing medicines.*

Since direct-procurement is expensive, we want to provide cheaper option by having an auction based or competitive-procurement. But we don’t want to disrupt an already stable system. There is also a problem that some branches might already have their own customization. This calls for a Bridge Pattern.

In this scenario, our Bridge Pattern consists of

* Application Domain:  
  This is your hospital/hospital system
* Solution Domain:  
  This is where you get your medicines using some procurement model

Procurement can happen with direct or competitive procurement model, and in this case each hospital branch have the the liberty to choose any model they prefer.



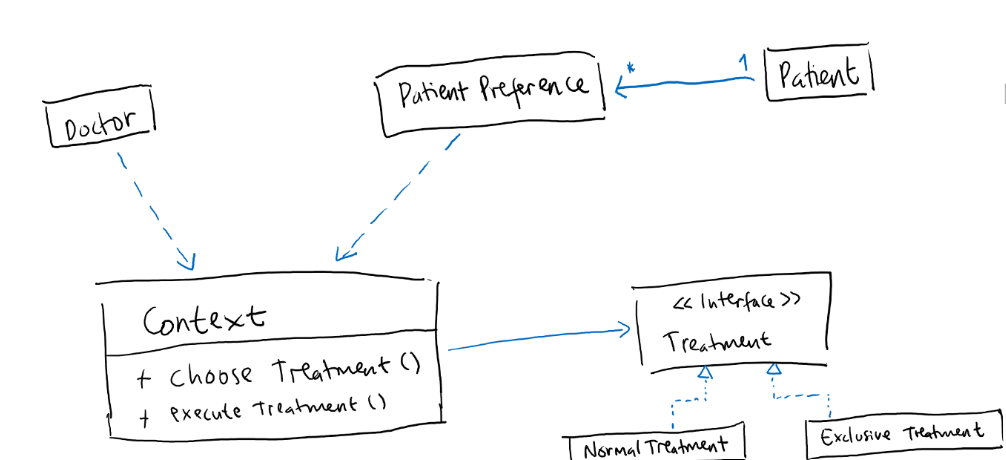
By creating a bridge pattern between abstraction of hospital and procurement, we can strictly define each branch to have procurement model but since the subclass can override the superclass then each branch can choose their preferred model and new branch can also choose their preferred model. Changing model also easy, just add implementation of procurement and change branch that wants to use that new model.

*In a sense, Adapter pattern might seem similar to Bridge pattern. But “Adapter makes things work after they’re designed; Bridge makes them work before they are.*

**Behavioral Patterns**

By applying patterns in this group you can answer “Who does what?” questions, e.g which algorithm is used for some problems, or which object instance to use. This will also simplify control flow and enable a better understanding of your program data flow.

With some preferences given, there will be differences in patient care/treatment. During the hospital system development, this scenario can be handled by utilizing strategy pattern



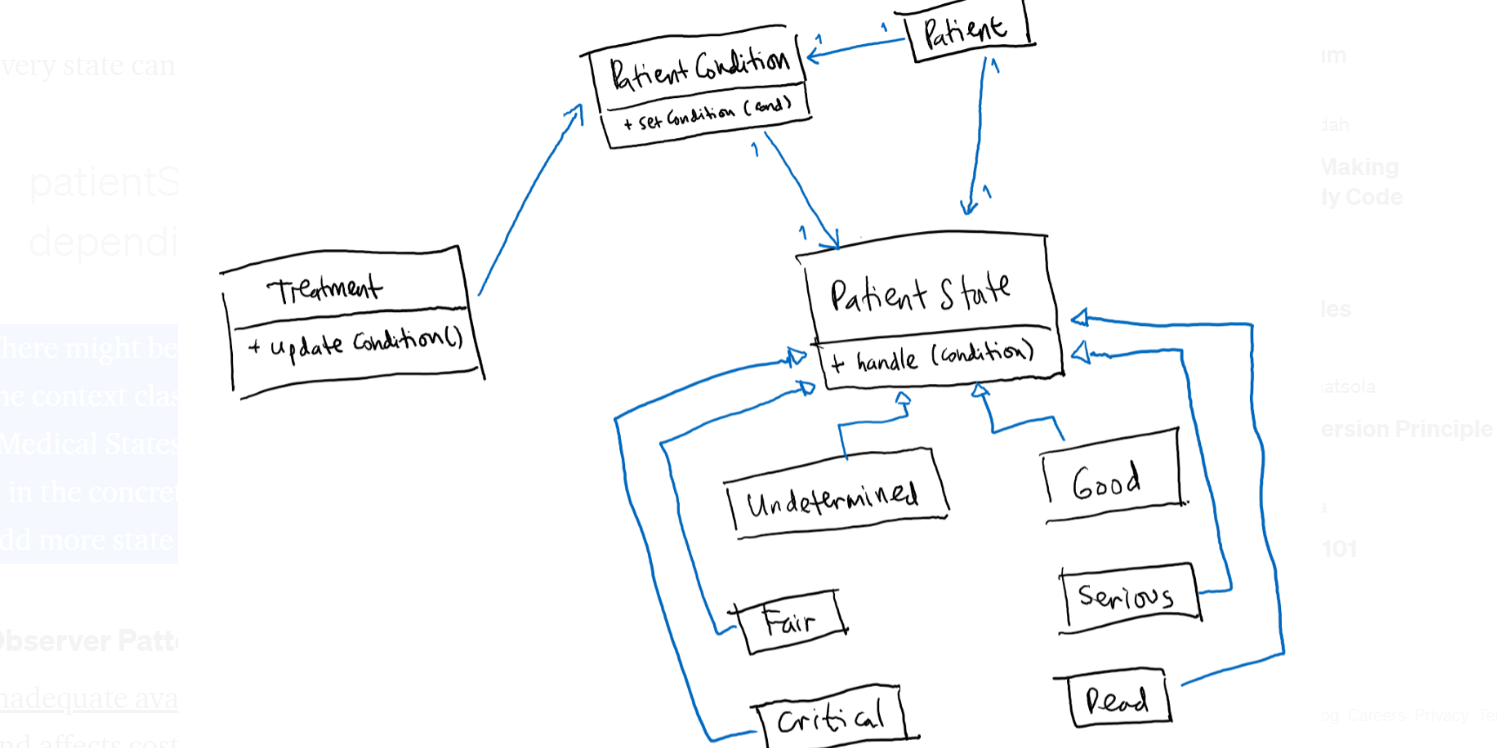
The difference with Bridge Pattern is that if you use a bridge pattern, the patient (depending on the preference) will receive the type of treatment that he/she wants before system runtime while in strategy pattern, the treatment will be decided from patient preferences and can be changed at runtime if the preferences of the patient changed.

**State Pattern on Medical State representation**

Sometimes, hospital describe patient’s condition in a medical state jargon. Even though this is not commonly used clinical term, it will serve as a nice example for the needs of using State Pattern. State Pattern is used to help object control their behavior by changing its internal state.

**Undetermined**  
Patient awaiting physician and/or assessment.  
**Good**

**Vital signs** are stable and within normal limits. Patient is conscious and comfortable. Indicators are excellent.  
**Fair**  
Vital signs are stable and within normal limits. Patient is conscious, but may be uncomfortable. Indicators are favorable.  
**Serious**  
Vital signs may be unstable and not within normal limits. Patient is seriously ill. Indicators are questionable.  
**Critical**  
Vital signs are unstable and not within normal limits. Patient may be unconscious. Indicators are unfavorable.  
**Dead**  
Vital signs have ceased. Patient has died.



Every state can move to another state depending on the patient’s condition.

There might be a question on where to handle state change, should it happen in the context class (TreatmentContext) or should it happen in concrete class (Medical States). I would say that this is up to personal preferences, but handling it in the concrete class will reduce the conditional block and making it simpler to add more state and/or see the limitation on each state.