More Design Patterns

October 3rd, 2022

Modularity

What makes good modularity?

- Each module should have one job
- Information hiding
    - Anything that is hidden can be changed later on
- Generality (we <3 polymorphism)

Benefits:

- Changeability! We can make adjustments to one module without needing to change the guts of other pieces of software.
- Easier debugging, since code is grouped in a sensable way and we can test modules independantly.
- Modules can be undestood in isolation
    - A system can be undestood as a collection of modules.
- Modules can hide implementation details
- Reusability
- Modules can be developed in parallel

Challenges:

- We are actually not very good at selecting good ways to modularize problems
- If you have to refactor the entire thing then you can get into real trouble
- Performance challenges :(
- There is no perfect modularity

Observer

The problem we are trying to solve is that one object's state must me kept consistent with another object's state. The subject holds the state. The observer wants to know about state changes to the subject. When the subject changes, it notifies the observer.

Consider the following exampile using the bad standard Java UI library:

```
class MyListener implements ActionListener {
    void actionPerformed(Action e) {
        System.out.println("Button Clicked!");
    }
}

JButton b = new JButton("Click me1");
b.addActionListener(new MyListener());
```

`MyListener` is the observer, `JButton` is the subject.

In a more general sense, the subject has a list of its observers, and has a public methods that can add or remove an observer to that list. It will also have a method that notifies every observer in that list.

The observer will have some sort of update method.

Wrappers

We have Adapters, Proxies, and Decorators. Debating the difference between these three subtypes is a fool's errand.

Adapter

When we want to convert code to a new interface without rewriting the code from scratch. Why would we do this instead of just rewriting it?

- Techincal risk in redoing the old code if it works correctly
- Maybe we can't change the old code!
  - We might not maintain it
  - Other code might rely on the old code

Proxy

Could the adapter be made a subclass of the adaptee?

```
class NetworkConnection implements NetworkInterface {
    String getPage(URL u) { ... }
}
class SafeNetworkConnection implements NetworkInterface {
    String getPage(URL u) {
        if (Safe.check(u)) { return c.getPage(u); }
        else { return new SuspiciousPageWarning(); }
    }
}
```