Chapter 2 : Observer Pattern

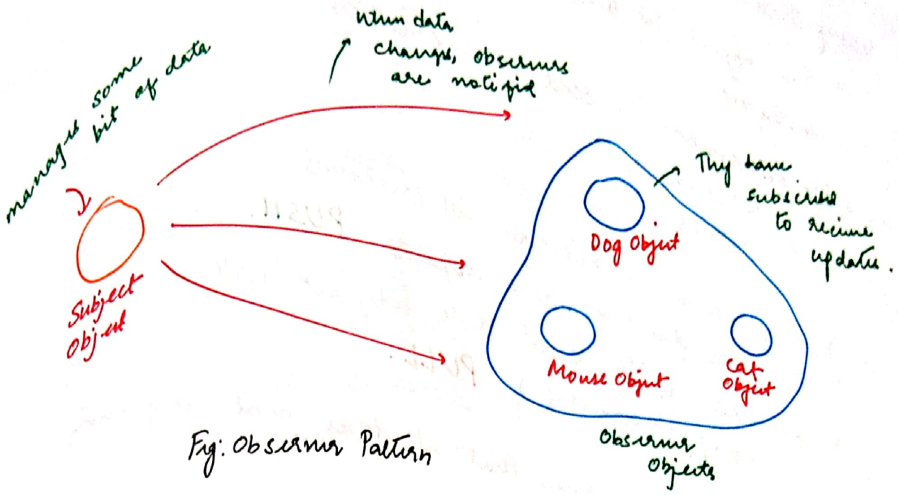

Fig: Observer Pattern

OBSERVER

- Observer pattern defines a one to many dependency between objects so that when one object changes state, all of its dependents are notified & updated automatically.
- we can reuse subjects or observers independently of each other
- The power of loose coupling.
  - loosely coupled means can interact but have very little knowledge.
  - object design where subject & observer are loosely coupled. Only thing subject knows about observer is that it implements a certain interface.

DP: Strive for loosely coupled designs between objects that interact. They allow us to build flexible OO systems that can handle change because they minimize interdependency between objects.

* Weather station -

* Observer - push notifications

  Subject - pass state to observers

  let observer PULL my state

} Push vs Pull.

* Java's built in Observable, extend it and tell it to notify. Observer interface and Observable class.

* Using Java's built in Observer:
  - For object to become observer. implement observer interface.
  - For observable to send notifications.
    - setChanged
    - notify
  - For observer to recieve notifications.
    - pass data as data objects. PUSH.
    - we don't send data object with notifyObservers(). PULL.

* Write your code such that it does not depend on notifications.

* DARK SIDE of java.util.observable.
  - Observable is a class, not interface. Can't add to existing class that extends another superclass.
  - can't even create own implementation.
  - violates second DP. (It doesn't favor composition over inheritance)

* Java swing heavily uses Observer pattern.

* How Java's built in Observer pattern works:- (Block diagram)

Pg 122 (Design Pattern challenge solution).