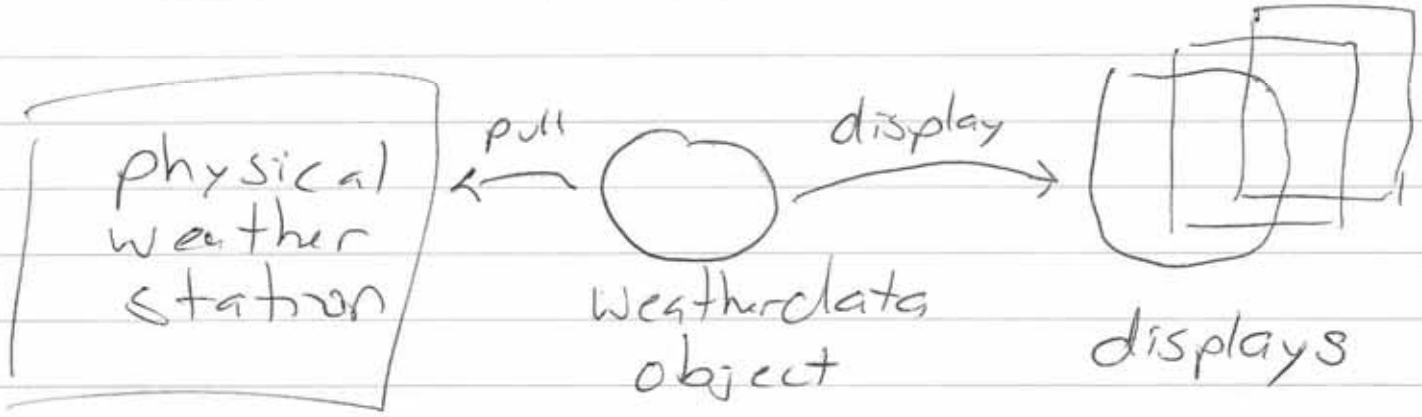
Thursday 9/25/14 4156

another example application -
weather station



initially 3 different displays, will be more
API
current conditions
weather stats
forecast

given WeatherData
getTemperature()
getHumidity()
getPressure()

need to implement
measurementsChanged() - which gets called whenever weather updated

first try - get the 3 measurements, call update on all 3 displays with those parameters ← need to change for each new observer

e.g. currentConditionsDisplay.update(temp, humid, press)

Thursday 9/25/14 4156

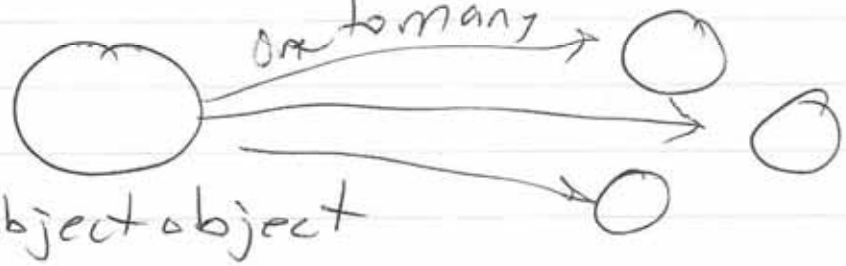
what are the problems?

need to change measurements Changed every time a new display added

instead should use observer design pattern

publishes + subscribers

↑ subject ↑ observers



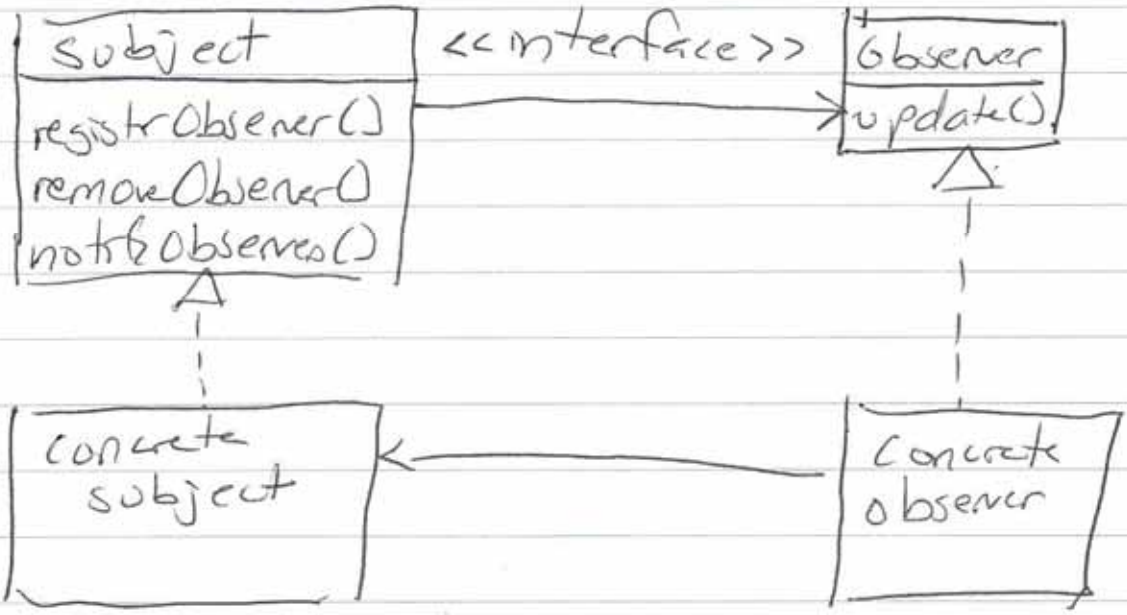
Subject object manages core data

notifies subscribes whenever data changes

↑ registered with subject (can dynamically register + unregister)

observer pattern defines a one to many dependency between objects so that when the object changes state, all of its dependents are notified + updated automatically

Thursday 9/25/14 4156



loose coupling - two objects can interact, but they have very little knowledge of each other

- only thing the subject knows about an observer is that it implements the observer interface

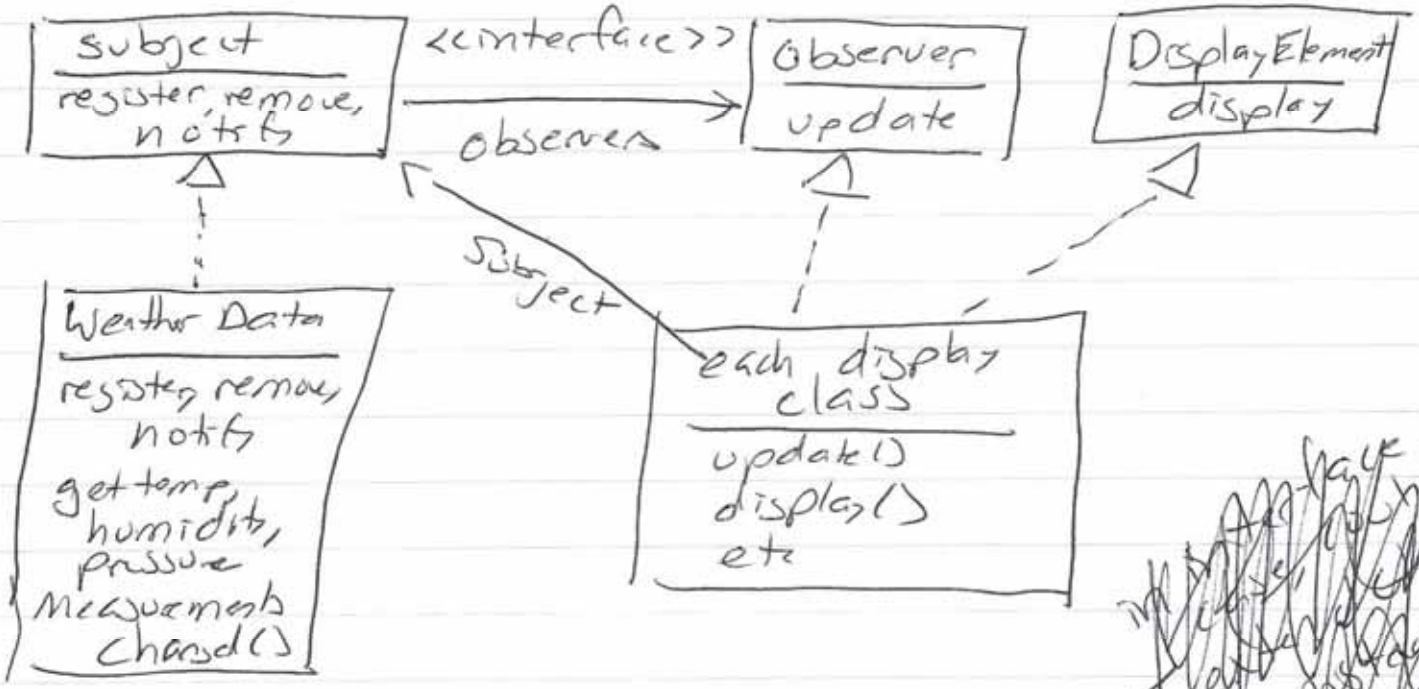
- can add new observers at any time & remove

- never need to modify subject to add new kinds of observers

- can reuse subjects & observers independently of each other, & changes to either will not affect the other

Thursday 9/25/14 4156

back to weather station




```
public class WeatherData implements Subject {
    private ArrayList observers;
    private float temperature;
    private float humidity;
    private float pressure;

    public WeatherData() {
        observers = new ArrayList();
    }

    methods to add, remove & notify
    public void measurementsChanged() {
        notifyObservers();
    }
}
```

Do not need to change any code for new observers

Thursday 9/25/14 4156

push vs. pull

```
public class SomeDisplay
      implements Observer, DisplayElement {

    private float temp;
    private float hum;
    private Subject weatherData;

    public SomeDisplay(Subject weatherData) {
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }

    public void update(float temp,
        float hum, float press) {
        this.temp = temp;
        this.hum = hum;
        display();
    }

    public void update() {
        this.temp = getTemp(this.weatherData);
        this.hum = getHum(this.weatherData);
        display();
    }
```

↑ pushes same parameters to all display elements

↑ pulls only data needed from subject

Thursday 9/25/14 4156

another example - Starbuzz coffee

has lots of different kinds of coffee
House Blend, Dark Roast, Decaf, Espresso

has lots of condiments
milk, Soy, Mocha, Whip
- maybe dable helpings

how to compute cost of a given beverage w/ its condiments

Separate subclass per combination?
- class explosion

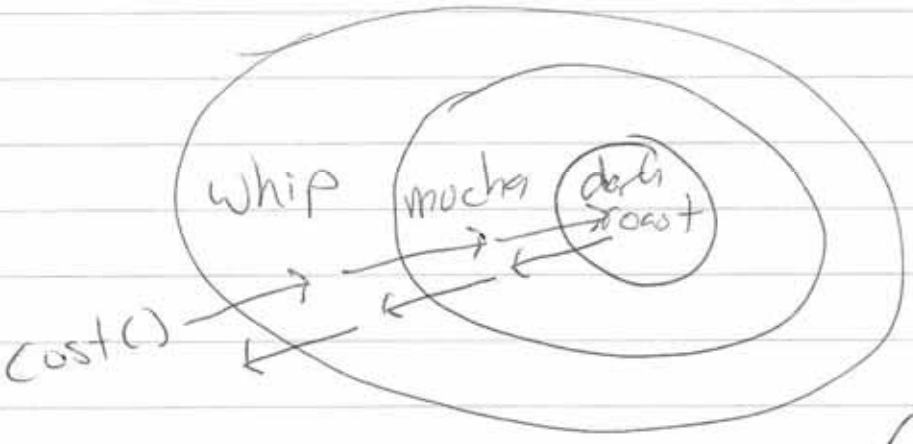
instance variable w/ getter & setter for each kind of condiment?
(& subclass per kind of drink)
- need to implement for each new kind of beverage

appl, open-closed principle
classes shald be open for extension but closed for modification
- add new behavior w/o modifying existing code

Thursday 9/25/14 4156

decorator pattern - attach additional responsibilities to an object dynamically

- decorators have the same supertype as the object they decorate
- can wrap an object w/ one or more decorators at runtime
- can pass decorated object in place of original object
- adds own behavior before or after delegating request to original object



adds each decorator has instance variable for the object it decorates (can for any other needs specific to behavior)

do not need to modify original beverage classes & superclass