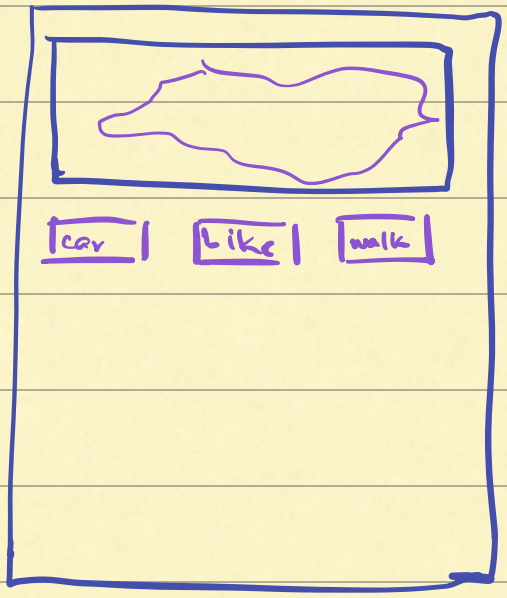
Agenda:

→ Strategy design pattern

→ Observer " "

Google Maps



Google Maps {

findPath (from, to, mode) {

if (mode == car) {

=

}

else if (mode == bike) {

=

}

else if (mode == walk) {

=

}

Similar set of code.

1. SRP
2. OCP

1st Solution → ① CarPathCalculator
- findPath / calculatePath {
}

② BikePathCalculator
- findPath { }

③ WalkPathCalculator
- findPath { }

googleMaps {

findPath (from, to, mode) {

if (mode == car) {

CarPathCalc carpathC = new CarPathCalc();

carpathC.findPath();

}

:

1. OCP
2. DI

Dependency Inversion

2nd Solution

<<Path Calculator Strategy>>

findPath(Source, Destination);

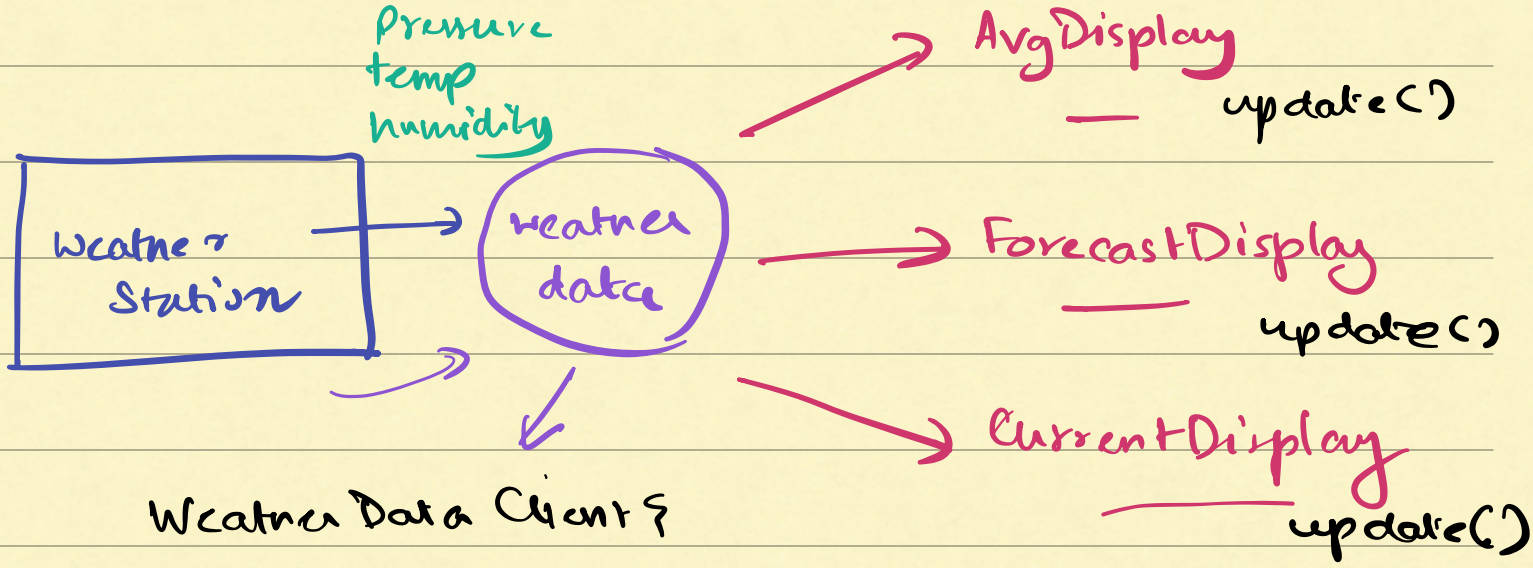
↙ ↓ ↘

CarPath Calculator	BikePath Calc	WalkPath Calculator
findPath(–,–){ ≡ }	findPath(–,–){ = }	

```
googleMaps{
    findPath(from, to, mode){        <- String Type mode. / Enum Type
        if (mode == car){
            PathCalculatorStrategy p = new CarPathCalc();
            p.findPath();
        }
        if ( ——> ){
            =
        }
    }
        ↓
    findPath2(from, to, PathCalcStrategy p){     <- Strategy Object.
        p.findPath(from, to);
    }
}
```

OCP
↓
to remove OCP add factory method.

Observer Design Pattern :



WeatherDataClient {

updateAlldisplays() {

avgDisplay.update();

forecastdisplay.update();

currentDisplay.update();

}

OCP

Publisher

<<Subject>>

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObserver();

Subscriber

<<Observer>>

void update(-,-);

implements

```
WeatherData :implements Subject {
    List<Observer> observerList;
    notifyObservers() {
        for (Observer o : observerList) {
            o.update();
        }
    }
}

weatherData.registerObserver(new
                    argDisplay());
```