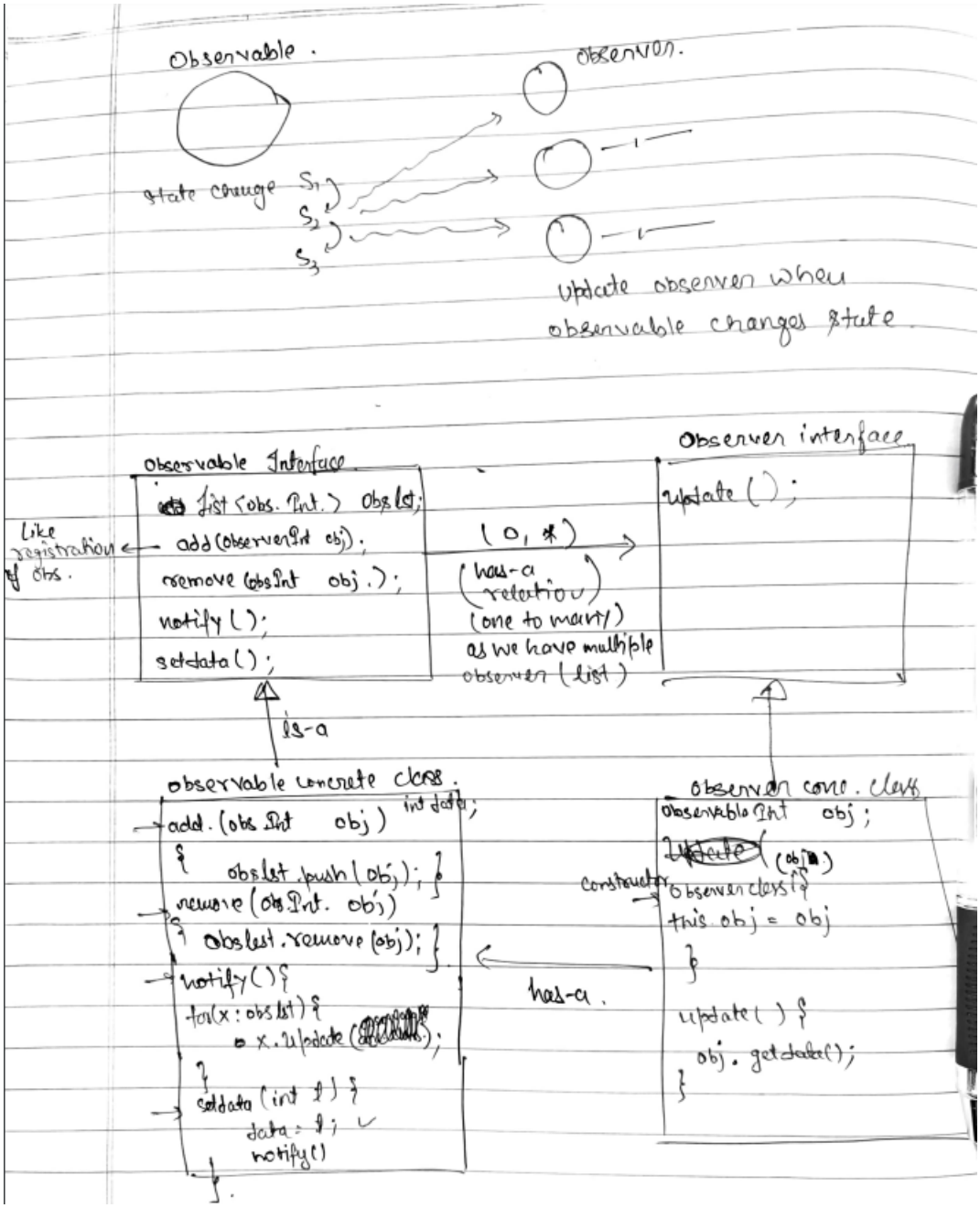
2. Observer - Design Pattern

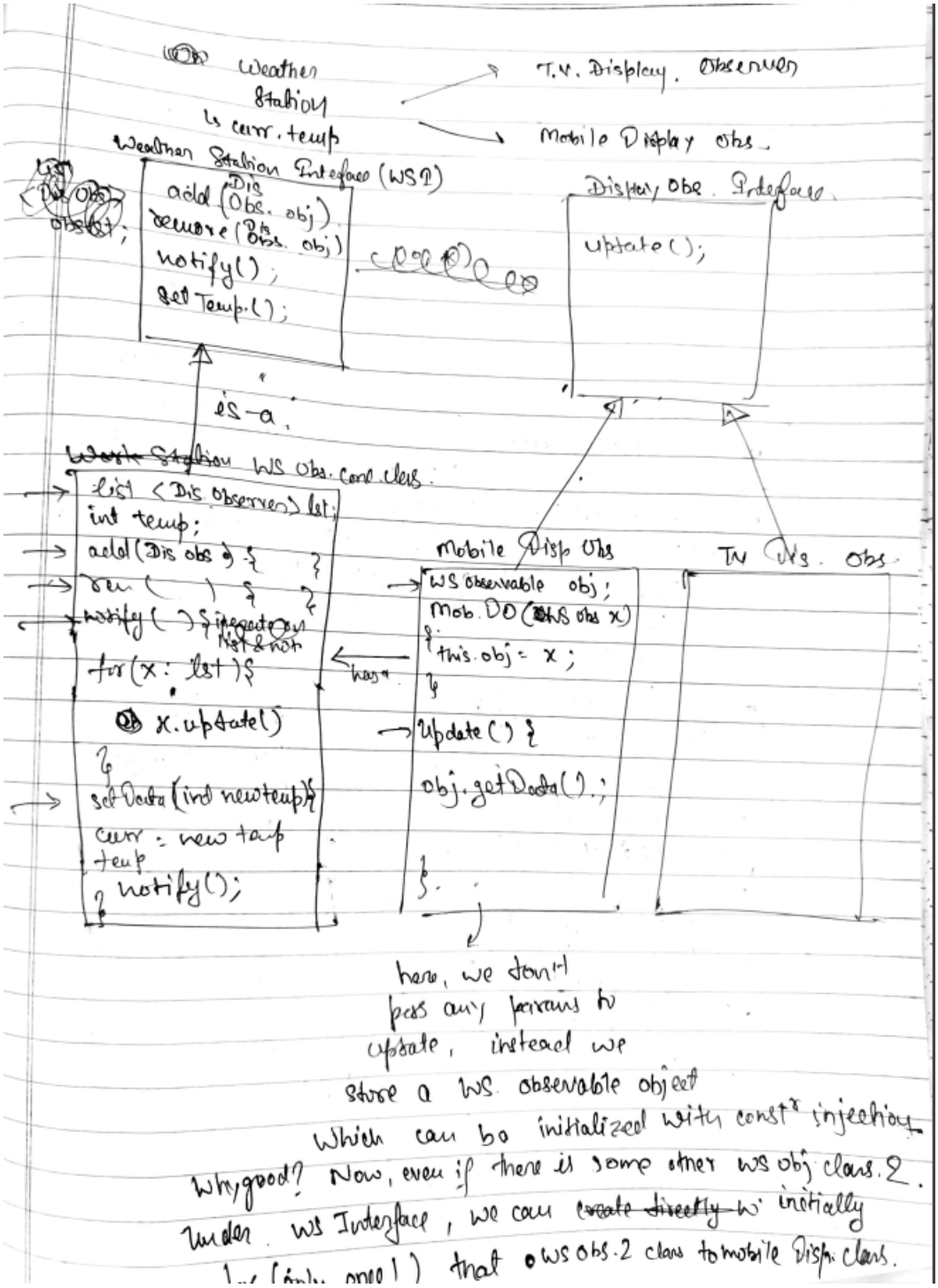
✏️ This design pattern is used when we have a problem related to notifying multiple users when some event occurs. (Observable + Observers) system.

pdf link:

Adobe Scan Jun 14, 2023.pdf



Example: Weather Station



Good example: https://www.geeksforgeeks.org/observer-pattern-set-1-introduction/

Code:

```
#include <bits/stdc++.h>
using namespace std;

// General Observer Interface
class NotificationObserver
{
public:
    virtual void update() = 0;
};

// Interface for stocks managament for any company
class StocksManager
{
public:
    virtual void add(NotificationObserver *obj) = 0;
    virtual void remove(NotificationObserver *obj) = 0;
    virtual void notify() = 0;
    virtual void setStocksPrice(double x) = 0;
    virtual double getStocksPrice() = 0;
};


// ###################################################################################

// Notification Observer on email
class EmailNotificationObserver : public NotificationObserver
{
private:
    StocksManager *generalStocksObservable;
    string email = "";

public:
    EmailNotificationObserver(string Email, StocksManager *obj)
    {
        generalStocksObservable = obj;
        email = Email;
    }

    void sendMail(string emailId, string msg)
    {
        cout << "mail sent to: " << emailId << "\n";
    }

    void update()
    {
        string msg = "hello ok bye";
        sendMail(email, msg);
    }
};

// Notification Observer on mobile phone
class MobileNotificationObserver : public NotificationObserver
{
private:
    StocksManager *generalStocksObservable;
    string phoneNo = "";

public:
    MobileNotificationObserver(string PhoneNumber, StocksManager *obj)
    {
        generalStocksObservable = obj;
        phoneNo = PhoneNumber;
    }

    void sendMessage(string pnum, string msg)
    {
```

```
        cout << "text message sent to: " << pnum << "\n";
    }

    void update()
    {
        string msg = "hello ok bye";
        sendMessage(phoneNo, msg);
    }
};

// ###############################################################################


// Samsung Stocks manager Observable class
class SamsungStocksManager : public StocksManager
{
private:
    set<NotificationObserver *> observersList;
    double stockesPrice = 23;

public:
    void add(NotificationObserver *obj)
    {
        observersList.insert(obj);
    }
    void remove(NotificationObserver *obj)
    {
        observersList.erase(obj);
    }

    void notify()
    {
        for (auto &x : observersList)
        {
            x->update();
        }
    }
    void setStocksPrice(double newPrice)
    {
        stockesPrice = newPrice;
        notify();
    }
    double getStocksPrice()
    {
        return stockesPrice;
    }
};

int main()
{
    StocksManager *samsungStocks = new SamsungStocksManager();

    NotificationObserver *observer1 = new EmailNotificationObserver("abc#gmail.com", samsungStocks);
    NotificationObserver *observer2 = new EmailNotificationObserver("xyz#gmail.com", samsungStocks);
    NotificationObserver *observer3 = new MobileNotificationObserver("9164684364", samsungStocks);

    samsungStocks->add(observer1);
    samsungStocks->add(observer2);
    samsungStocks->add(observer3);

    samsungStocks->setStocksPrice(123.165);



    return 0;
}
```