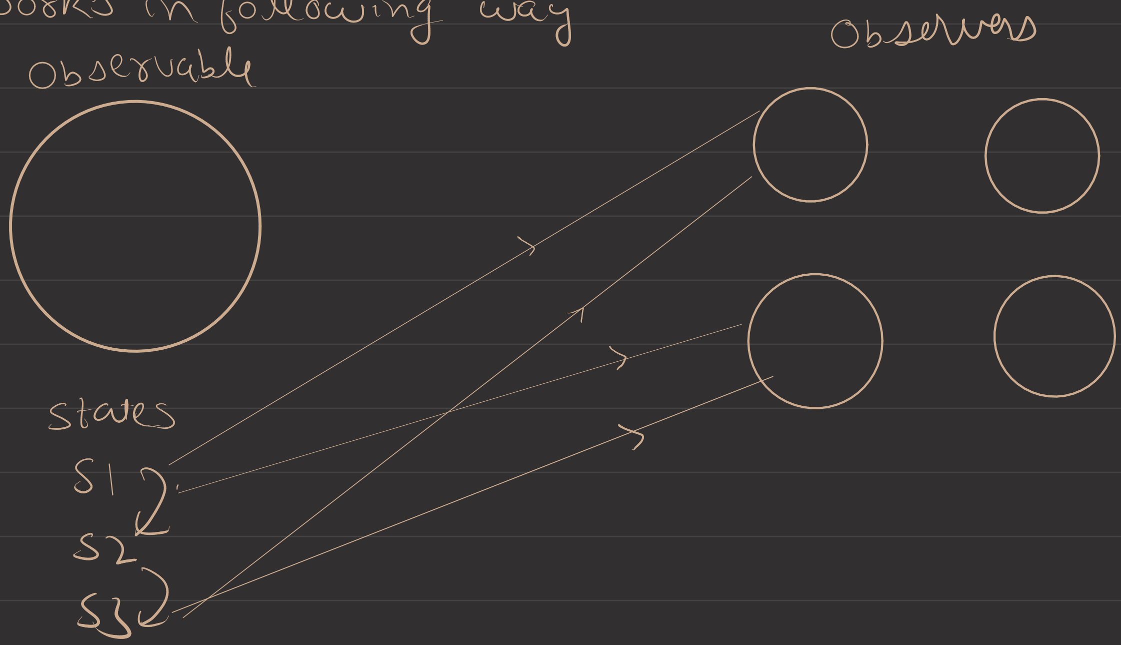
Observer Design Pattern

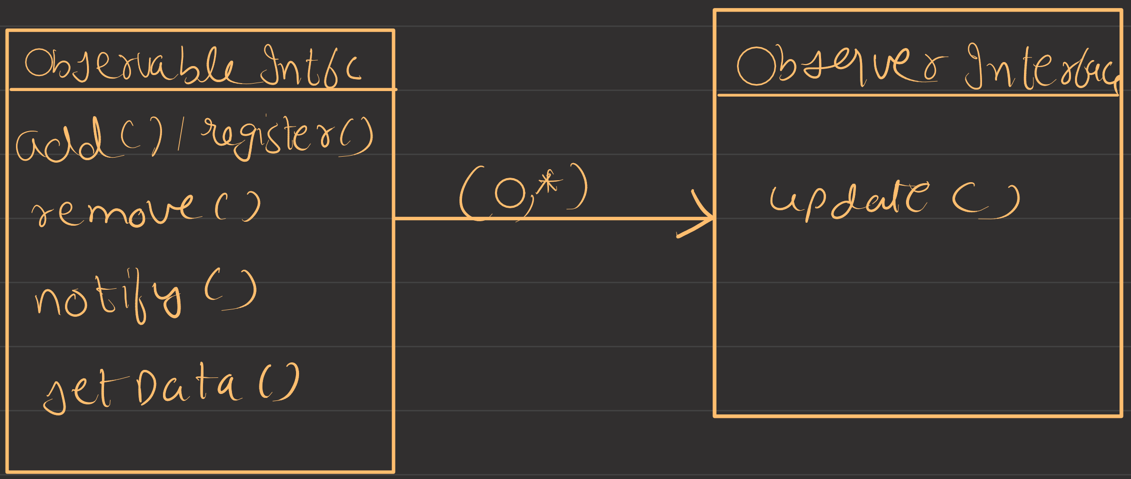
In this pattern we have two interfaces :- Observable & observer

It works in following way



So here whats happening is, whenever there is a state change in observable, it notifies all the notifiers which are attached to it

In terms of classes & Interfaces



lets discuss what's happening here

So the observable Interface has following functions :-

1) add / Register :- The observer is added to a list to notify

2) remove :- Inverse of above

3) notify :- The main function which will notify observer

4) setData :- It is used to change the state of observable

Observable

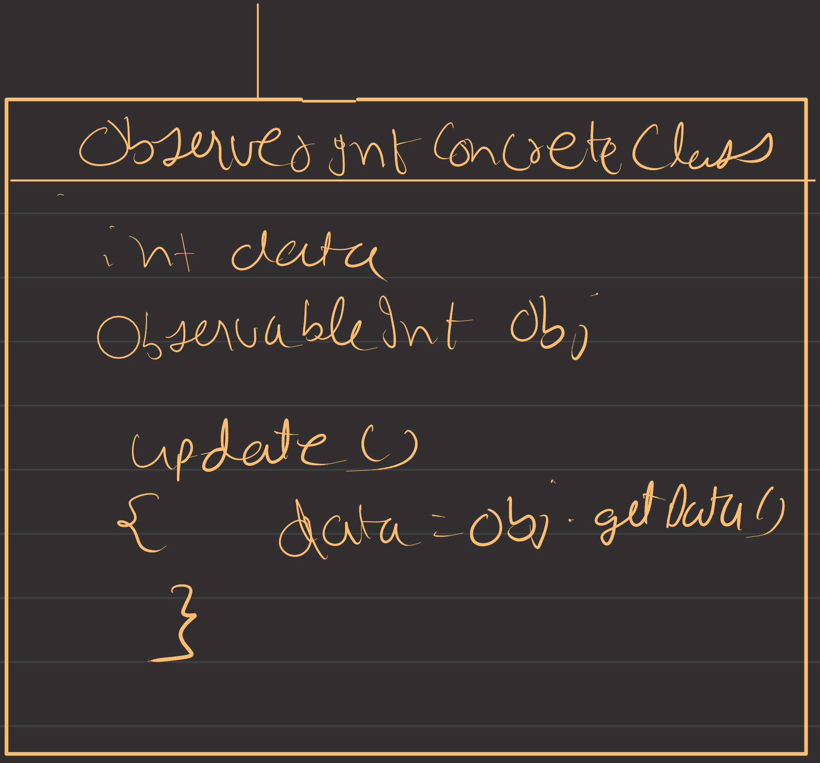
1 update → used to update the observer

ObservableIntConcrete Class

```
lst : observeInt[ ]
 int data
add (observer Int obj) ( lst.push(obj))
remove (observe Int obj) ( lst.remove (obj))

 notify ( )
 {   for (obj in lst)
     {  obj.update()
     }
 setData ( int it)
 {   data = it; notify()}
 getData () {return  data}
```

↑



```
ObserverIntConcreteClass

int data
ObservableInt obj;

update()
{     data = obj.getData()
  }
```

So from above we have following

Observations :-

1 Observer has an object of observable (will discuss)

2 Whenever we setData, we notify observers

<u>Why do we have observable obj in Observer?</u>

Whenever you call an update() method from notify, it will run. But now the problem is which Observable class called it. (Because there can be many)

And then in update we need to check which obj called it by Passing it. This is not a good approach.

So instead we pass in the Object of observable directly into Observer.