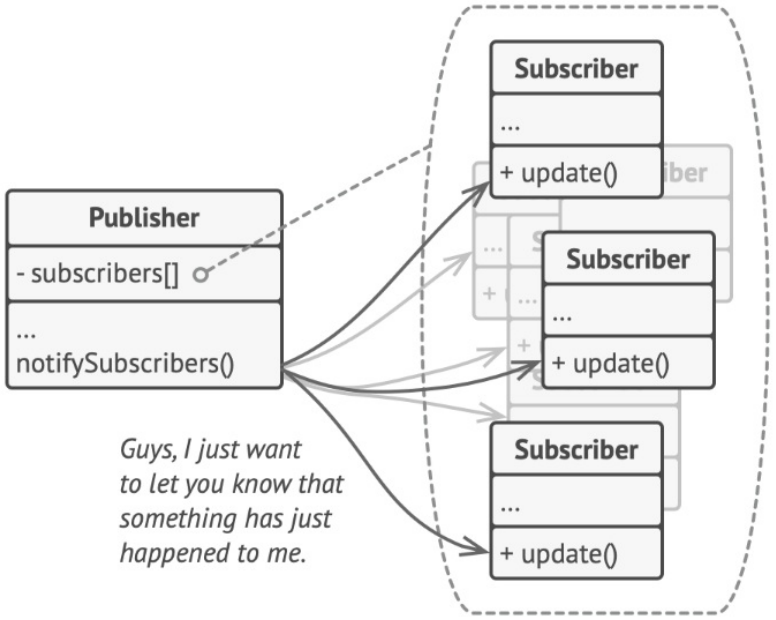
Suppose you go to Amazon App.Now we go to see a product.And we want to buy the product but we see that the product is out of stock .we click on notify me .Now as soon as product is on the stock.Notify me will send me notification that prodct is in stock .Now we need to implement that notify me!!

in Observer design pattern we have 2 object
1.Observable
2.Observer(there can be multiple observer for one observable).

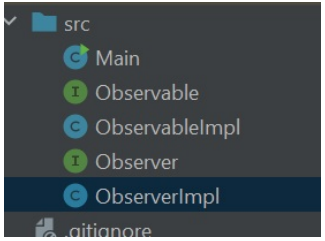
Now whenever there is any state change of observable then it will tell the observer ki observable ki state change ho gyi hai!!

here we have interface Observable having methods like addObserver(Observer observer),removeObserver(Observer observer) and notifyObservers() and one more method that has business logic.
Observer interface has method getUpdate();



Publisher notifies subscribers by calling the specific notification method on their objects.

this is diagram of observable design pattern
this looks good




```
4 usages   1 implementation
public interface Observable {

    3 usages   1 implementation
    public void addObserver(Observer observer);
    no usages   1 implementation
    public void removeObserver(Observer observer);
    1 usage   1 implementation
    public void notifyObserver();
    1 usage   1 implementation
    public void changeState(int val);

    1 usage   1 implementation
    public int getData();
}
```

here getData() is function which tells value of changed state

Interface pe hm data nii le skte as jo bii data lenge interface pe voh final hona chahie and if final kr dia toh chnage nii kr paeenge so we will take data in implementations

```java
public class ObservableImpl implements Observable{
    3 usages
    List<Observer> list=new ArrayList<>();
    2 usages
    int data =0;
    3 usages
    @Override
    public void addObserver(Observer observer) {
        list.add(observer);
    }

    no usages
    @Override
    public void removeObserver(Observer observer) {
            list.remove(observer);
    }

    1 usage
    @Override
    public void notifyObserver() {
        for(Observer observer:list){
            observer.update();
        }
    }
```

<- registering of observer to observable

```java
    1 usage
    @Override
    public void changeState(int val) {
        data=val;
        notifyObserver();
    }

    1 usage
    public int getData() {
        return data;
    }
}
```

here line 27 calls notifyObserver() method which shares the chnaged variable info with all observers

Ek observable ko many observer can observe

```java
7 usages  1 implementation
public interface Observer{

    1 usage  1 implementation
    public void update();
}
```

to get info of data chnaged we passed Observable object to ObserverImpl so that we can fetch data from observable so for communication we can have object in both classes Observer has Observable and Observable has Observers

```java
3 usages
public class ObserverImpl implements Observer{
    2 usages
    int id;
    2 usages
    String name;
    2 usages
    Observable observable;

    3 usages
    public ObserverImpl(int id, String name, Observable observable) {
        this.id = id;
        this.name = name;
        this.observable = observable;
    }

    1 usage
    @Override
    public void update() {
        System.out.println("Observer id : "+this.id+" name : "+this.name);
        System.out.println("the new value is : "+observable.getData());
    }
}
```

See observable has many observers but observer can have one observable although one observer can observe many observable and

```java
public class Main {
    public static void main(String[] args) {
        Observable observable=new ObservableImpl();
        observable.addObserver(new ObserverImpl( id: 12, name: "mohit",observable));
        observable.addObserver(new ObserverImpl( id: 11, name: "rohit",observable));
        observable.addObserver(new ObserverImpl( id: 10, name: "aman",observable));
        observable.changeState( val: 232);
    }
}
```

output:-

```
Observer id : 12 name : mohit
the new value is : 232
Observer id : 11 name : rohit
the new value is : 232
Observer id : 10 name : aman
the new value is : 232
```

output:-

```
Observer id : 12 name : mohit
the new value is : 232
Observer id : 11 name : rohit
the new value is : 232
Observer id : 10 name : aman
the new value is : 232
after removing one object ----------------------------
Observer id : 12 name : mohit
the new value is : 111
Observer id : 10 name : aman
the new value is : 111
```

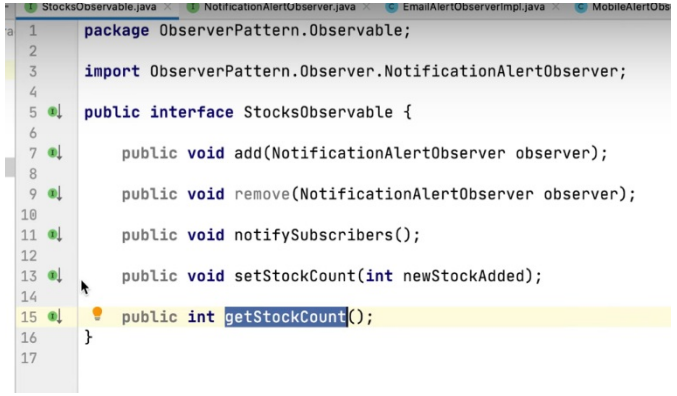
```java
public class Main {
    public static void main(String[] args) {
        Observable observable=new ObservableImpl();
        observable.addObserver(new ObserverImpl( id: 12, name: "mohit",observable));
        Observer obj=new ObserverImpl( id: 11, name: "rohit",observable);
        observable.addObserver(obj);
        observable.addObserver(new ObserverImpl( id: 10, name: "aman",observable));
        observable.changeState( val: 232);
        observable.removeObserver(obj);
        System.out.println("after removing one object ----------------------------");
        observable.changeState( val: 111);
    }
}
```

ex weather station -->sets temp after every hour
temp chnages there are multiple observers like mobile display,tv display
Observable(work station)
Observer(various displays)

Another eg Friend request bheji Accept hui ge nhi !!

now there can be multiple observables suppose cricket match is going on then tv display show that too. So passing observable in parameter help to get data from observabale.CricketmatchObservable we will pass if we need data from that else weather station observable

Now suppose in ObserverImpl in constructor we dont provide observable then we need to provide Observable object in update method as we need to access data what observable has changed.



another eg of observable here we are implementing notify me

see line 33 if stockCount==0 then only notify subscribers yha pe stockCount>0 aana chahie as jaisa hi stock mai aaye toh notify kro

```
1  package ObserverPattern.Observable;
2
3  import ObserverPattern.Observer.NotificationAlertObserver;
4
5  import java.util.ArrayList;
6  import java.util.List;
7
8  public class IphoneObservableImpl implements StocksObservable {
9
10     public List<NotificationAlertObserver> observerList = new ArrayList<>();
11     public int stockCount = 0;
12
13     @Override
14     public void add(NotificationAlertObserver observer) { observerList.add(observer); }
17
18     @Override
19     public void remove(NotificationAlertObserver observer) { observerList.remove(observer); }
22
23     @Override
24     public void notifySubscribers() {
25
26         for(NotificationAlertObserver observer : observerList) {
27             observer.update();
28         }
29     }
30
31     public void setStockCount(int newStockAdded) {
32         if(stockCount == 0) {
33             notifySubscribers();
34         }
35         stockCount = stockCount + newStockAdded;
36     }
37
38     public int getStockCount() { return stockCount; }
41 }
42
```

StocksObservable.java | NotificationAlertObserver.java | EmailAlertObserverImpl

```java
package ObserverPattern.Observer;

public interface NotificationAlertObserver {

    public void update();
}
```

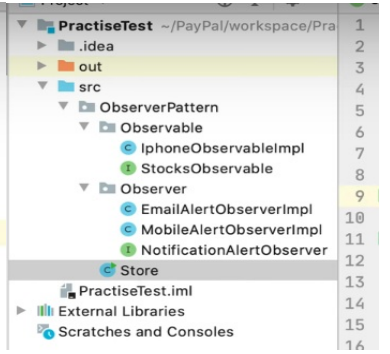
just observers

```java
package ObserverPattern.Observer;

import ObserverPattern.Observable.StocksObservable;

public class EmailAlertObserverImpl implements NotificationAlertObserver {

    String emailId;
    StocksObservable observable;

    public EmailAlertObserverImpl(String emailId, StocksObservable observable){

        this.observable = observable;
        this.emailId = emailId;
    }

    @Override
    public void update() {
        sendMail(emailId, msg: "product is in stock hurry up!");
    }

    private void sendMail(String emailId, String msg){

        System.out.println("mail sent to:" + emailId);
        //send the actual email to the end user
    }
}
```

```java
package ObserverPattern.Observer;

import ObserverPattern.Observable.StocksObservable;

public class MobileAlertObserverImpl implements NotificationAlertObserver{

    String userName;
    StocksObservable observable;

    public MobileAlertObserverImpl(String emailId, StocksObservable observable){

        this.observable = observable;
        this.userName = emailId;
    }

    @Override
    public void update() { sendMsgOnMobile(userName, msg: "product is in stock hurry up!"); }

    private void sendMsgOnMobile(String userName, String msg){

        System.out.println("msg sent to:" + userName);
        //send the actual email to the end user
    }
}
```

another observer implemenatation

```java
package ObserverPattern;

import ObserverPattern.Observable.IphoneObservableImpl;
import ObserverPattern.Observable.StocksObservable;
import ObserverPattern.Observer.EmailAlertObserverImpl;
import ObserverPattern.Observer.MobileAlertObserverImpl;
import ObserverPattern.Observer.NotificationAlertObserver;

public class Store {

    public static void main(String args[]) {

        StocksObservable iphoneStockObservable = new IphoneObservableImpl();

        NotificationAlertObserver observer1 = new EmailAlertObserverImpl( emailId: "xyz1@gmail.com", iphoneStockObservable);
        NotificationAlertObserver observer2 = new EmailAlertObserverImpl( emailId: "xyz2@gmail.com", iphoneStockObservable);
        NotificationAlertObserver observer3 = new MobileAlertObserverImpl( emailId: "xyz_username", iphoneStockObservable);

        iphoneStockObservable.add(observer1);
        iphoneStockObservable.add(observer2);
        iphoneStockObservable.add(observer3);

        iphoneStockObservable.setStockCount(10);


    }
}
```

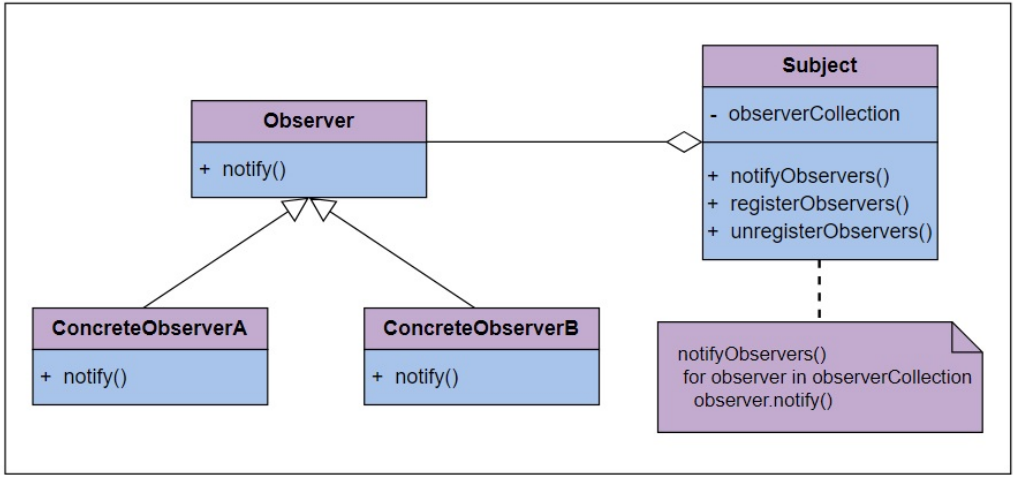



```
/Library/Java/JavaVirtualMachine
mail sent to:xyz1@gmail.com
mail sent to:xyz2@gmail.com
msg sent to:xyz_username
```

Type

The observer is a software design pattern that falls into the behavioral design.

Structure

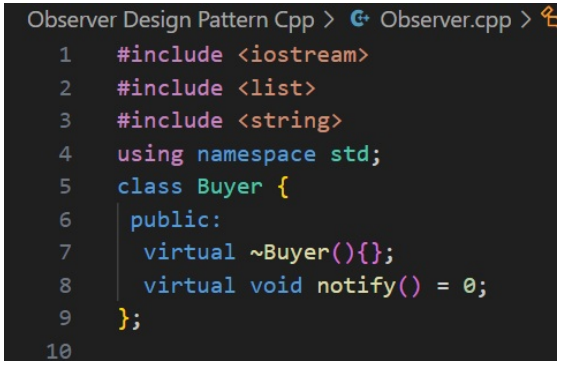


UML for the observer design pattern

Participants

- `Subject`:
  - Manages its collection of observers.
  - Offers the possibility for the observers to log in and log out. In the above example, the shop/store was our subject.
- `Observer`:
  - Defines an interface through which the observers can be notified.
- `ConcreteObserver`:
  - Will be notified by the `Subject`. In our example, buyers were our concrete observers.

see c++ implementation with proper logic of notify me app



in c++ we need to make sure of our implementation as buyer needs to be defined 1st and then shop as shop using buyer

Here buyer is pure virtual class as no implementation of methods so it is working like an interface

```
class Shop {
public:
 void registerBuyer(Buyer* observer) {
   observers.push_back(observer);
 }
 void unregisterBuyer(Buyer* observer) {
   observers.remove(observer);
 }
 void updatedata(int val){
   int prevdata=this->data;
   this->data+=val;
   if(prevdata<0 && this->data >0){
       notifyBuyers();
   }else if(this->data <0){
       cout<<"Not is stock Please try later"<<endl;
   }else {
       cout<<"Now total stock of data is "<<getData();
   }
 }
int getData(){
   return this->data;
}
```

```
 }
 private:
  list<Buyer *> observers;
  int data=-123;
void notifyBuyers() {
    for (auto observer: observers) observer->notify();
  }
};
```

see updateData() we want customer to be only notified when previously it was not in stock we dont want to update customer everytime our stock gets updated

```
44
45  class Directcustomer : public Buyer {
46  private:
47   Shop * shop;
48   public:
49      Directcustomer(Shop *shop){
50          this->shop=shop;
51      };
52      void notify() override {
53          cout << "Direct customer Notified\n";
54          cout<<"data in direct customer is :"<<shop->getData()<<endl;
55      }
56  };
57
```

```
58   class Partnercustomer : public Buyer {
59   private:
60   Shop* shop;
61    public:
62       Partnercustomer(Shop * shop){
63           this->shop=shop;
64       };
65       void notify() override {
66           cout << "Partners Notified \n";
67           cout<<"data in partner customer is :"<<shop->getData()<<endl;
68       }
69   };
70
71
```

these both are inherited from buyer class

```
71
72  int main() {
73
74      Shop* Subject = new Shop;
75      Buyer* George = new Directcustomer(Subject);
76      Buyer* Xyz_company = new Partnercustomer(Subject);
77
78      Subject->registerBuyer(George);
79      Subject->registerBuyer(Xyz_company);
80
81      Subject->updatedata(-12);
82      Subject->updatedata(1248);
83      std::cout <<  "Unrigester: George " << "\n";
84      Subject->unregisterBuyer(George);
85      Subject->updatedata(1026);
86
87      delete George;
88      delete Xyz_company;
89
90      delete Subject;
91
92  }
```

```
[Running] cd "c:\Users\user\Documents\pr
Observer && "c:\Users\user\Documents\pro
Not is stock Please try later
Direct customer Notified
data in direct customer is :1113
Partners Notified
data in partner customer is :1113
Unrigester: George
Now total stock of data is 2139
```