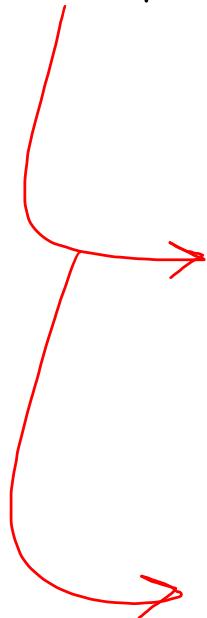


Pre - requisite



OOP's Concepts

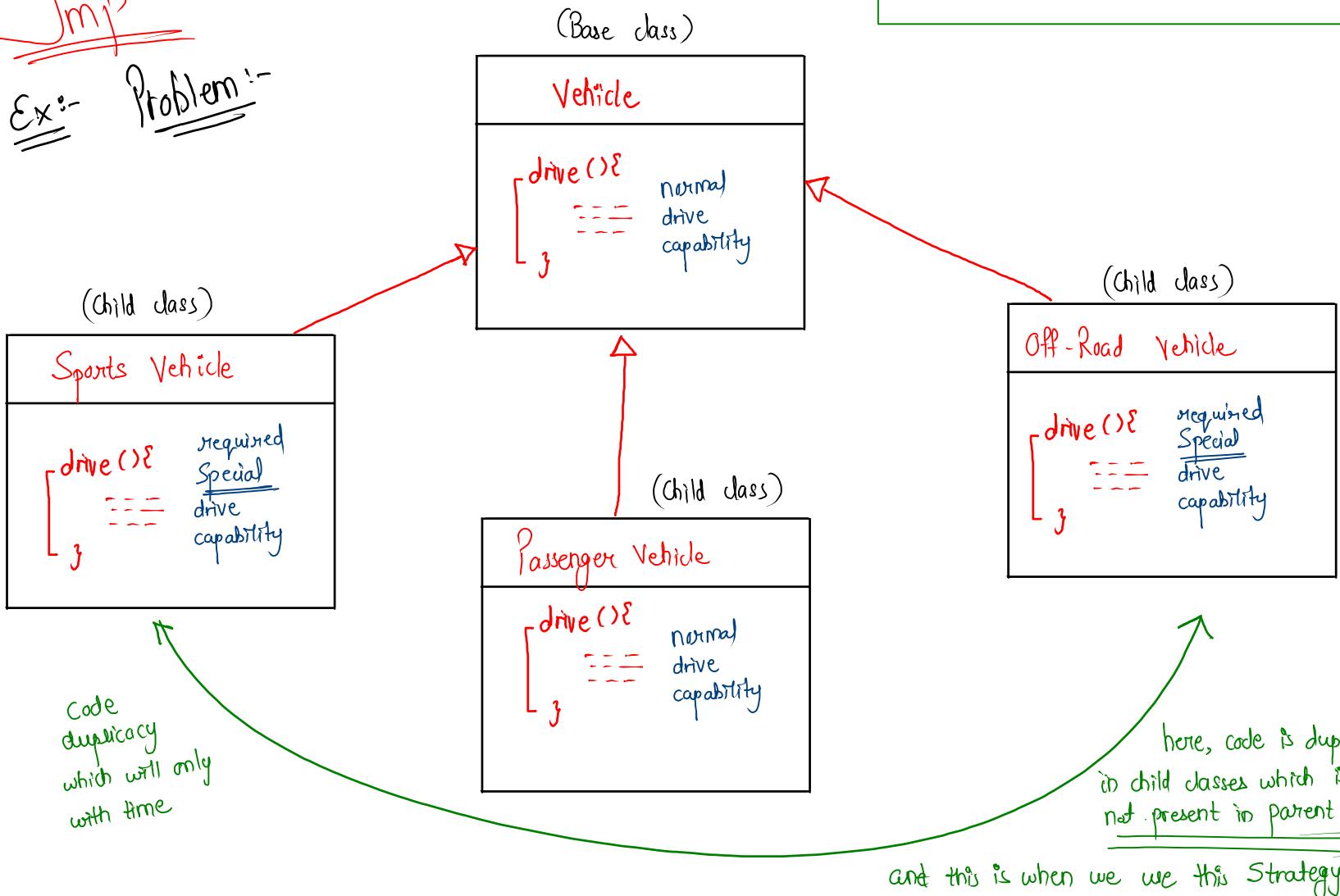
A red hand-drawn arrow pointing from the 'OOP's Concepts' text down towards the 'SOLID Principles' text.

SOLID Principles

⇒ Strategy Design Pattern

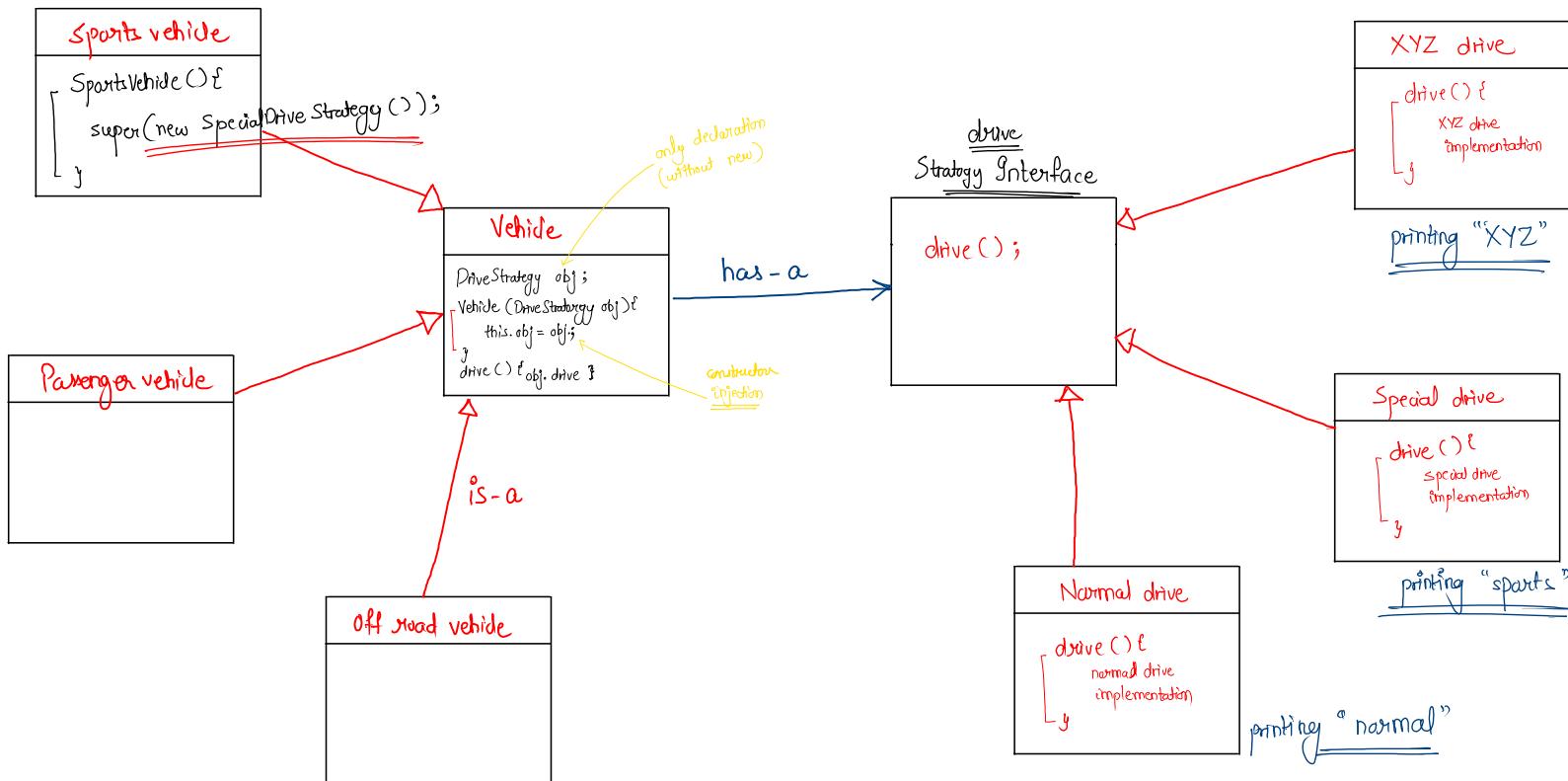
~~JMP~~

Ex:- Problem :-



Solution :-

Now we have DriveStrategy Interface, which is implementation 3 type of drive and we can use any of it while implementation Passenger, Offroad or Sports Vehicle.



Note:- We just need to pass the object (let's say **Sports vehicle**) to **Vehical** (parent class) using **super()** and it will initialize the **obj** with that type

Client code

```
public class Main {  
    public static void main(String[] args) {  
        Vehide obj = new SportsVehide();  
        obj.drive(); // print sports  
  
        Vehide obj = new NormalDrive();  
        obj.drive(); // print normal  
    }  
}
```

→ Observer Design Pattern :- (Walmart interview) question

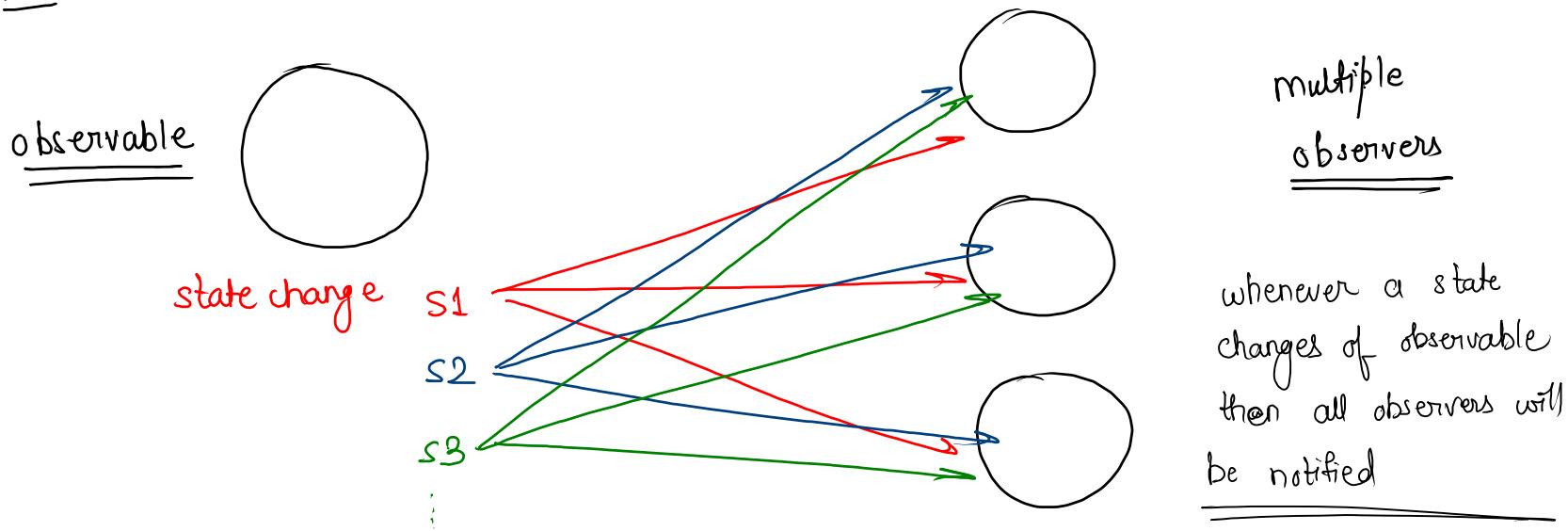
Jue)

amazon.com
product is unavailable
notify me button

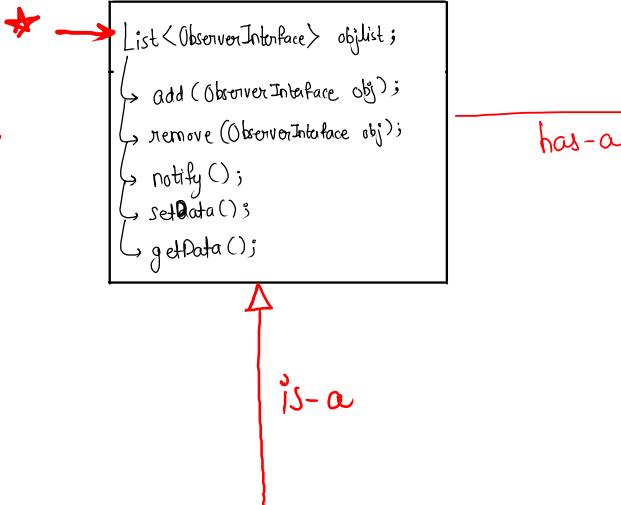
In amazon.com, we are looking for a product which is unavailable and there is a button of notify me?, so send notifications to customers when product is available.

Implement this button (LLD questions)

Note:- There are 2 states



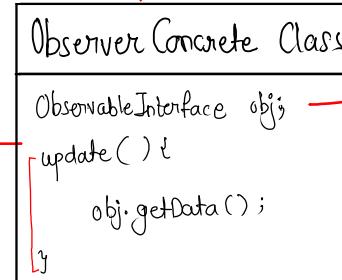
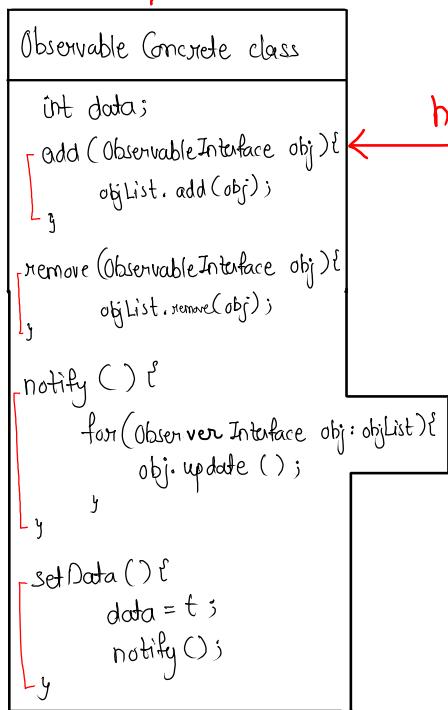
Observable Interface



Observer Interface

```
update();
```

there can
be multiple
concrete
classes



here this obj is used
to know that which
concrete class we are
referencing currently.

Note:- here task of notify() method is to
notify all the observers to call the
update method according the current
changes.

Example

A Weather station is updating current tempo every hour

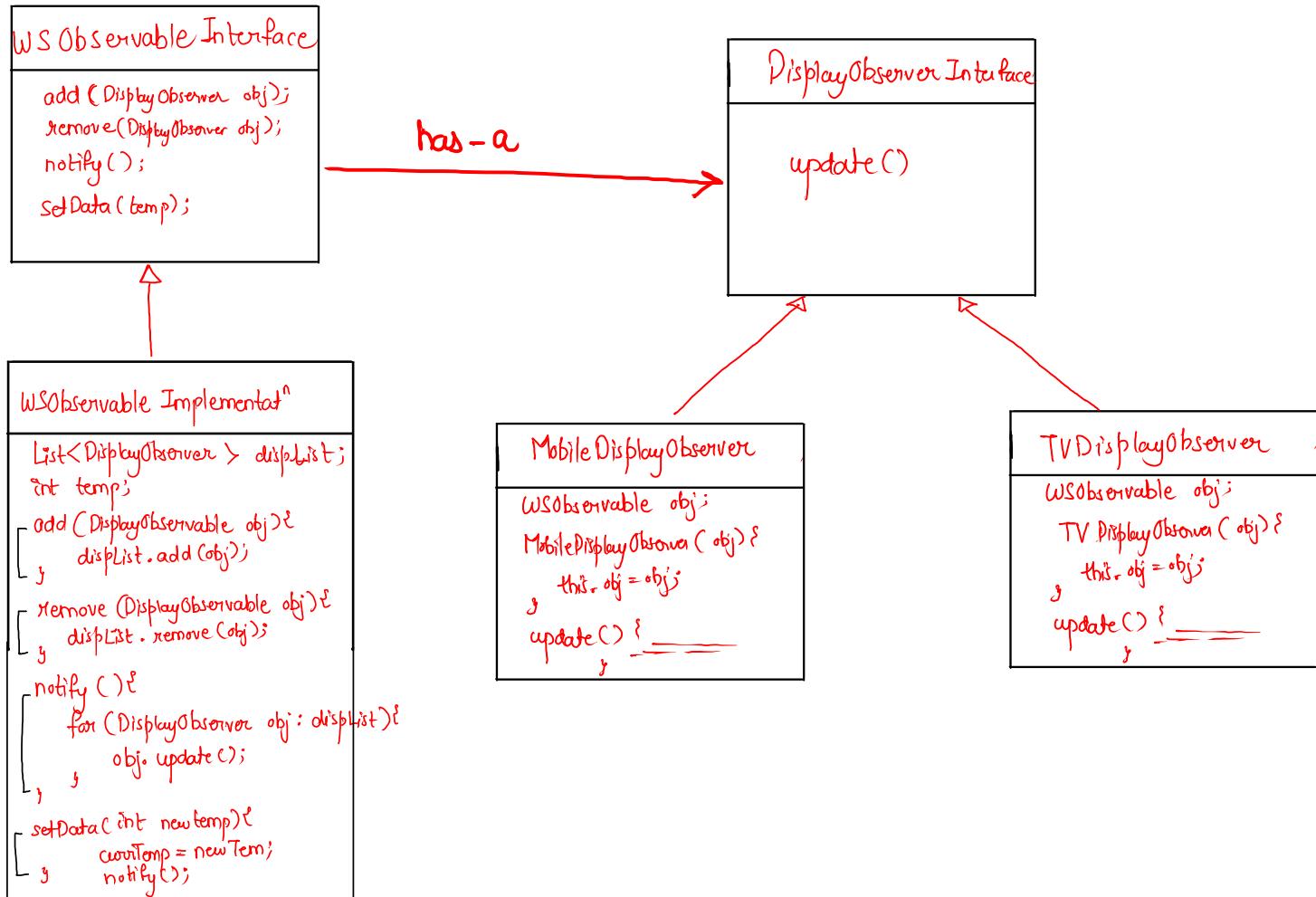
observed by

TV display observer
X

Mobile display observer

Solution

Note :- WS Observable = weather station observable



Solution of Walmart Interview Question :-

```
public interface StocksObservable {  
    public void add(NotificationAlertObserver observer);  
    public void remove(NotificationAlertObserver observer);  
    public void notifySubscribers();  
    public void setStockCount(int newStockAdded);  
    public int getStockCount();  
}
```

has-a

```
public interface NotificationAlertObserver {  
    public void update();  
}
```

```
public class IphoneObservableImpl implements StocksObservable{  
  
    public List<NotificationAlertObserver> observerList = new ArrayList<>();  
    public int stockCount = 0;  
  
    @Override  
    public void add(NotificationAlertObserver observer) { observerList.add(observer); }  
  
    @Override  
    public void remove(NotificationAlertObserver observer) { observerList.remove(observer); }  
  
    @Override  
    public void notifySubscribers() {  
        for(NotificationAlertObserver observer : observerList) {  
            observer.update();  
        }  
    }  
  
    public void setStockCount(int newStockAdded) {  
        if(stockCount == 0) {  
            notifySubscribers();  
        }  
        stockCount = stockCount + newStockAdded;  
    }  
  
    public int getStockCount() { return stockCount; }  
}
```

```
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, "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  
    }  
}
```

```
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, "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  
    }  
}
```

Note:- here, we have created a stockobservable which we are implementing using IphoneObservableImpl, and now we want to notify the update to all the required customer, for which we have 2 type :- either we can sent it through mobile phone or we can send it through email.

client code

```
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);  
    }  
}
```

☞ this will notify all through email or mobile
and update stock count by +10