Chain ofResp

```
public static void main(String args[]) {

    LogProcessor logObject = new InfoLogProcessor(new DebugLogProcessor(new ErrorLogProcessor(null)));

    logObject.log(LogProcessor.ERROR, "exception happens");
    logObject.log(LogProcessor.DEBUG, "need to debug this ");
    logObject.log(LogProcessor.INFO, "just for info ");

}
public abstract class LogProcessor{
    public static int INFO=1;
    public static int DEBUG=2;
    public static int ERROR=3;
    LogProcessor nextLogProcessor;
    LogProcessor(LogProcessor logProcessor){
        this.nextLogProcessor= logProcessor;
    }

    public void log(int logLevel,String message){
        if(nextLogProcessor!=null){
            nextLogProcessor.log(logLevel,message);
        }
    }
}

public class InfoLogProcessor extends LogProcessor{

    InfoLogProcessor(LogProcessor nexLogProcessor){
        super(nexLogProcessor);
    }


    public void log(int logLevel,String message){

        if(logLevel == INFO) {
            System.out.println("INFO: " + message);
        } else{

            super.log(logLevel, message);
        }

    }
}
public class DebugLogProcessor extends LogProcessor{

    DebugLogProcessor(LogProcessor nexLogProcessor){
        super(nexLogProcessor);
    }

    public void log(int logLevel,String message){

        if(logLevel == DEBUG) {
            System.out.println("DEBUG: " + message);
        } else{

            super.log(logLevel, message);
        }

    }
}
public class ErrorLogProcessor extends LogProcessor{

    ErrorLogProcessor(LogProcessor nexLogProcessor){
        super(nexLogProcessor);
    }

    public void log(int logLevel,String message){

        if(logLevel == ERROR) {
            System.out.println("ERROR: " + message);
        } else{

            super.log(logLevel, message);
        }

    }
}
```

Observer

```
    public static void main(String[] args) {
        StocksObservable iphone = new IphoneStocksObservable();
        NotificationAlertObserver o1 = new EmailAlertObserverImpl("abc@gmai.com",iphone);
        NotificationAlertObserver o2 = new EmailAlertObserverImpl("abc@gmai.com",iphone);
        NotificationAlertObserver o3 = new EmailAlertObserverImpl("abc@gmai.com",iphone);

        iphone.add(o1);
        iphone.add(o2);
        iphone.add(o3);
        iphone.setStockCount(10);
    }


public class EmailAlertObserverImpl implements NotificationAlertObserver{
    String emailId;
    StocksObservable so;

    public EmailAlertObserverImpl(String emailId, StocksObservable so){
        this.emailId = emailId;
        this.so=so;
    }

    @Override
    public void update(){
        sendEmail(emailId,"product is in stock. Please hurry");
    }

    private void sendEmail(String emailId, String msg){
        System.out.println("Mail sent  to "+emailId);
    }
}

public interface StocksObservable{
    public void add(NotificationAlertObserver observer);
    public void remove(NotificationAlertObserver observer);

    public void notifySubscribers();
    public void setStockCount(int newStock);
    public int getStockCount();
}

public interface NotificationAlertObserver{
    public void update();
}

public class IphoneStocksObservable implements StocksObservable{
    public List<NotificationAlertObserver> observerList = new ArrayList<>();

    public int stockCount = 0;
    public void add(NotificationAlertObserver observer) {
        observerList.add(observer);
    }
    public void remove(NotificationAlertObserver observer){
        observerList.remove(observer);
    }

    public void notifySubscribers(){
        for(NotificationAlertObserver observer: observerList){
            observer.update();
        }
    }
    public void setStockCount(int newStock){
        if(stockCount==0){
            notifySubscribers();
        }
        stockCount +=newStock;

    }
    public int getStockCount(){
        return stockCount;
    }
}
```

Stratergy

```
DriveStratergy(){
//Interface
drive()
}

NormalDrive() implements DriveStratergy{
drive("Normal Drive")
}

SpecialDrive() implements DriveStratergy {
drive("Special Drive")
}
----
Vehicle{
    DriveStratergy ds;
    Vehicle(DriveStratergy ds){
        this.ds = ds;
    }
    drive(){
        ds.drive();
    }

}
SportsCar extends Vehicle{

    SportsCar(){
        super(new SpecialDrive());
    }
```