**import** java.util.\*;

**//Mediator Pattern**

public class Mediator {

public static void main(String[] args) {

IRealtor realtor = new Realtor();

realtor.doMediation();

((Realtor)realtor).displayLog();

}

}

interface IRealtor{

public void doMediation();

}

class Log{

double currentAskingPrice;

double currentOfferingPrice;

String log = "";

public void setOffer(double currentOfferingPrice, String message){

log += "$" + currentOfferingPrice + message + "\n";

}

public void setCounterOffer(double currentAskingPrice, String message){

log += "$" + currentOfferingPrice + message + "\n";

}

public String getLog(){ return log; }

}

class Realtor implements IRealtor { //mediator

Client buyer, seller;

Log log;

public Realtor(){

buyer = new Client(this, "buyer");

seller = new Client(this, "seller");

log = new Log();

}

public void doMediation(){

do{

buyer.offer();

log.setOffer(buyer.getPrice(), " Buyer message: " + buyer.getMessage());

seller.offer();

log.setOffer(seller.getPrice(), " Seller message: " + seller.getMessage());

}while(!buyer.getMessage().equalsIgnoreCase("done")

&& !seller.getMessage().equalsIgnoreCase("done"));

}

public void displayLog() {

System.out.println(log.getLog());

}

}

class Client{

IRealtor realtor;

Scanner scanner;

int offeringPrice;

String message;

String identity;

public Client(IRealtor r, String iden){

realtor = r;

scanner = new Scanner(System.in);

identity = iden;

}

public void offer(){

System.out.println(identity + " Offer price: ");

if(scanner.hasNextInt())

offeringPrice = scanner.nextInt();

scanner.nextLine();

System.out.println("Message: ");

if(scanner.hasNextLine())

message = scanner.nextLine();

}

public int getPrice(){ return offeringPrice; }

public String getMessage(){ return message; }

}

**// Observe Pattern**

**public** **class** ObservePattern {

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

AcademicEventMonitor monitor = **new** AcademicEventMonitor();

IObserver ob1 = **new** RegistrarOffice();

IObserver ob2 = **new** FinancialOffice();

monitor.addEventListener(ob1);

monitor.addEventListener(ob2);

AcademicEvent event = **new** AcademicEvent("withdrawCourse");

monitor.setEvent(event);

//\*\*\*\*\*\*\*\*\*\* Java support of observer pattern \*\*\*\*\*\*\*\*\*

EventSource evSrc = **new** EventSource();

// create an observer

Registration regis = **new** Registration();

FinancialAid finanAid = **new** FinancialAid();

// subscribe the observer to the event source

evSrc.addObserver( regis );

evSrc.addObserver( finanAid );

// starts the event thread

Thread thread = **new** Thread(evSrc);

thread.start();

}

}

**class** AcademicEvent{

String eventName;

CarrollStudent currentStudent;

**public** AcademicEvent(String event){

currentStudent = **new** CarrollStudent();

eventName = event;

}

**public** **void** setStudent(CarrollStudent s){

currentStudent = s;

}

**public** **void** setEvenName(String n){

eventName = n;

}

}

**class** CarrollStudent{

String name = "Evan";

}

**interface** IObserver {

**void** update(AcademicEvent e);

}

**class** RegistrarOffice **implements** IObserver {

**private** CarrollStudent currentStudent;

**public** CarrollStudent getCurrentStudent() {

**return** currentStudent;

}

**public** **void** setCurrentStudent(CarrollStudent s) {

**this**.currentStudent = s;

}

**public** **void** update(AcademicEvent e) {

CarrollStudent s = e.currentStudent;

setCurrentStudent(s);

System.*out*.println("Registrar office received " + e.eventName + " event about student " + s.name + " & updated the student record..");

}

}

**class** FinancialOffice **implements** IObserver {

**private** CarrollStudent currentStudent;

**public** CarrollStudent getCurrentStudent() {

**return** currentStudent;

}

**public** **void** setCurrentStudent(CarrollStudent s) {

**this**.currentStudent = s;

}

**public** **void** update(AcademicEvent e) {

CarrollStudent s = e.currentStudent;

setCurrentStudent(s);

System.*out*.println("Financial office received " + e.eventName + " event about student " + s.name + " & updated the student record..");

}

}

**class** AcademicEventMonitor {

**private** ArrayList<IObserver> observerList;

AcademicEvent event;

**public** AcademicEventMonitor(){ observerList = **new** ArrayList<IObserver>(); }

**public** **void** addEventListener(IObserver observer) {

observerList.add(observer);

}

**public** **void** removeEventListener(IObserver observer) {

observerList.remove(observer);

}

**public** **void** setEvent(AcademicEvent e) {

**this**.event = e;

informListeners();

}

**public** AcademicEvent getEvent(){ **return** event; }

**private** **void** informListeners() {

**for** (IObserver item: observerList) {

item.update(getEvent());

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\* Java Support of Observer Pattern \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* References:

\* http://docs.oracle.com/javase/7/docs/api/java/util/Observable.html

\* http://docs.oracle.com/javase/7/docs/api/java/util/Observer.html

\*/

**class** EventSource **extends** Observable **implements** Runnable{

**public** **void** run(){

**final** Scanner inReader;

inReader = **new** Scanner(System.*in*);

String evenName = "start";

**while**( !evenName.equals("stop") ){

evenName = inReader.nextLine();

setChanged();

notifyObservers( **new** AcademicEvent(evenName) );

}

}

}

**class** Registration **implements** Observer{

**private** String eventN;

**public** **void** update (Observable obj, Object arg){

/\* obj would be an EvenSource object if used, and arg is the parameter object provided to “notifyObservers(Object arg)” \*/

**if** (arg **instanceof** AcademicEvent){

eventN = (String) ((AcademicEvent)arg).eventName;

System.*out*.println("\nRegistrar received event: "+ eventN + " for processing...");

}

}

}

**class** FinancialAid **implements** Observer{

**private** String eventN;

**public** **void** update (Observable obj, Object arg){

System.*out*.println(obj.getClass());

**if** (arg **instanceof** AcademicEvent){

eventN = (String) ((AcademicEvent)arg).eventName;

System.*out*.println("\nFinancial Aid received event: " + eventN + " for processing..." );

}

}

}

**// State Pattern**

**public** **class** StatePattern {

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

PriorityMailHandler handler = **new** PriorityMailHandler();

handler.addMail(**new** PriorityMail());

handler.addMail(**new** PriorityMail());

handler.checkStatus();

handler.clearUp();

}

}

**enum** MState{

*INITIAL*, *InPROCESS*, *FINAL*, *NULL*

}

**abstract** **class** MailingState{

**protected** MState s;

**public** MState getStateConstant() { **return** s; }

**public** **void** setStateConstant(MState s) { **this**.s = s; }

**public** **abstract** **void** informCustomer();

**public** **abstract** **void** transition(PriorityMail mail);

}

**class** InitialState **extends** MailingState{

**public** InitialState(){ **this**.s = MState.*INITIAL*; }

@Override

**public** **void** informCustomer() {

System.*out*.println("Your priority mail has been processed...");

}

@Override

**public** **void** transition(PriorityMail mail) {

mail.setState( **new** InProcessState() );

mail.setStateConstant(MState.*InPROCESS*);

}

}

**class** InProcessState **extends** MailingState{

@Override

**public** **void** informCustomer() {

System.*out*.println("Your priority mail is being delivered...");

}

@Override

**public** **void** transition(PriorityMail mail) {

mail.setState( **new** FinalsState() );

mail.setStateConstant(MState.*FINAL*);

}

}

**class** FinalsState **extends** MailingState{

@Override

**public** **void** informCustomer() {

System.*out*.println("Your priority mail has arrived its destination...");

}

@Override

**public** **void** transition(PriorityMail mail) {

mail.setState( **new** NullState() );

mail.setStateConstant(MState.*NULL*);

}

}

**class** NullState **extends** MailingState{

@Override

**public** **void** informCustomer() {

System.*out*.println("It's null state...");

}

@Override

**public** **void** transition(PriorityMail mail) {

mail.setState( **new** InitialState() );

mail.setStateConstant(MState.*NULL*);

}

}

**class** PriorityMail{

MailingState state;

**public** MailingState getState() { **return** state; }

**public** **void** setState(MailingState state) { **this**.state = state; }

**public** **void** setStateConstant(MState s){ state.setStateConstant(s); }

**public** MState getStateConstant(){ **return** state.getStateConstant(); }

**public** **void** informCustomer(){ state.informCustomer(); }

**public** **void** transitionToNextState(){

state.transition(**this**);

}

**public** **double** getMailingCharge(String zip){ **return** 0.0; }

}

**class** PriorityMailHandler{

ArrayList<PriorityMail> list;

**public** PriorityMailHandler(){ list = **new** ArrayList<PriorityMail>(); }

**public** **void** addMail(PriorityMail mail){

mail.setState(**new** InitialState());

mail.informCustomer();

list.add(mail);

}

**public** **void** checkStatus(){

**for**(**int** i = 0; i < list.size(); i++){

PriorityMail mail = list.get(i);

**if**(mail.getStateConstant() == MState.*INITIAL*){

mail.transitionToNextState();

mail.informCustomer();

}

**if**( hasArrived( mail ) ) {

mail.transitionToNextState();

mail.informCustomer();

}

}

}

**public** **void** clearUp(){

**for**(**int** i = 0; i < list.size(); i++){

PriorityMail mail = list.get(i);

**if**(mail.getStateConstant() == MState.*FINAL*){

mail.transitionToNextState();

}

}

}

**public** **boolean** hasArrived(PriorityMail mail){ **return** **true**; }

}