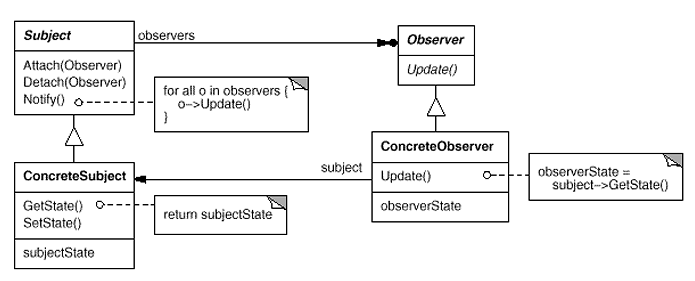
Observer Design Pattern

GOF : **Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically**.

**Structure Diagram**



**Basic Java Code**

**Observer.java**

**public** **abstract** **class** Observer {

**protected** Subject subject;

**public** **abstract** **void** update();

}

**Subject.java**

**import** java.util.ArrayList;

**import** java.util.List;

**class** Subject {

**private** List<Observer> observers = **new** ArrayList<Observer>();

**private** String state;

**public** **void** attach(Observer o) {

observers.add(o);

}

**public** String getState() {

**return** state;

}

**public** **void** setState(String in) {

state = in;

notifyObservers();

}

**private** **void** notifyObservers() {

**for**( Observer observer : observers )

observer.update();

}

}

**ConcreteObserver1.java**

**public** **class** ConcreteObserver1 **extends** Observer {

**public** ConcreteObserver1(Subject s) {

subject = s;

subject.attach(**this**);

}

@Override

**public** **void** update() {

System.***out***.println("ConcreteObserver1 got updated state : "+subject.getState());

}

}

**ConcreteObser2.java**

**public** **class** ConcreteObserver2 **extends** Observer {

**public** ConcreteObserver2(Subject s) {

subject = s;

subject.attach(**this**);

}

@Override

**public** **void** update() {

System.***out***.println("ConcreteObserver2 got updated state : "+subject.getState());

}

}

**Test.java**

**public** **class** Test {

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

Subject subject = **new** Subject();

**new** ConcreteObserver1(subject);

**new** ConcreteObserver2(subject);

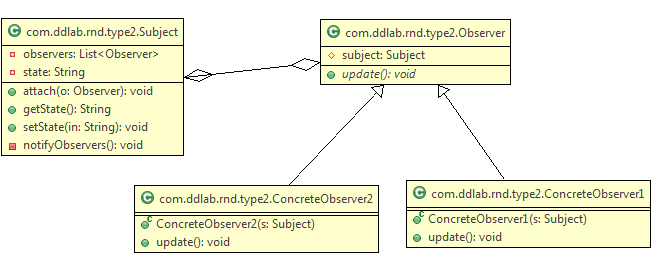
subject.setState("Delhi");

subject.setState("Bangalore");

}

}

**UML Diagram**

****

No let us convert the above example of Aeroplane and AirTrafficControllers.

**AbstractTrafficController.java**

**public** **abstract** **class** AbstractTrafficController {

**protected** Aeroplane plane;

**public** **abstract** **void** update();

}

**AirTrafficController.java**

**public** **class** AirTrafficController **extends** AbstractTrafficController {

**public** AirTrafficController(Aeroplane s) {

plane = s;

plane.attach(**this**);

}

@Override

**public** **void** update() {

System.***out***

.println("AirTrafficController got info that , plane has come to : "

+ plane.getTravelledCity());

}

}

**Aeroplane.java**

**import** java.util.ArrayList;

**import** java.util.List;

**class** Aeroplane {

**private** List<AbstractTrafficController> observers = **new** ArrayList<AbstractTrafficController>();

**private** String travelledCity;

**public** **void** attach(AbstractTrafficController o) {

observers.add(o);

}

**public** String getTravelledCity() {

**return** travelledCity;

}

**public** **void** setTravelledCity(String travelledCity) {

**this**.travelledCity = travelledCity;

notifyObservers();

}

**private** **void** notifyObservers() {

**for** (AbstractTrafficController observer : observers)

observer.update();

}

}

**Test.java**

**public** **class** Test {

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

Aeroplane plane = **new** Aeroplane();

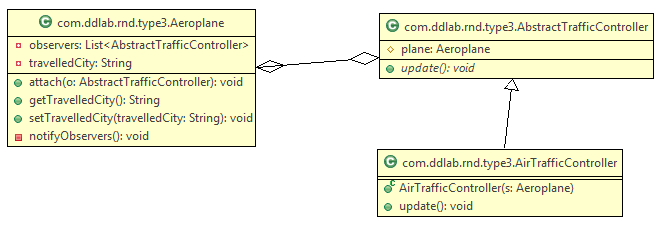
**new** AirTrafficController(plane);

plane.setTravelledCity("Delhi");

}

}

**UML Diagram**

****

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. This pattern helps in maintaining consistency between related objects without making classes tightly coupled. Let us consider an example that an aeroplane is monitored by Air Traffic Controller . Air Traffic contoller will monitor the city the aeroplance has travelled or crossed. Let us have a look at the following example.

In this case Aeroplane extends Observable whereas Air Traffic Contoller implements Observer as Air Traffic contoller wants to observer. So Air Traffic Contoller is an observer. This pattern helps to know the recent update of an object state.

import java.util.Observable;

public class Aeroplane extends Observable

{

private String travelledCity;

public String getTravelledCity() {

return travelledCity;

}

public void setTravelledCity(String travelledCity) {

this.travelledCity = travelledCity;

setChanged();

notifyObservers(travelledCity);

}

}

import java.util.Observable;

import java.util.Observer;

public class AirTrafficController implements Observer

{

private String cityName;

@Override

public void update(Observable o, Object arg)

{

System.out.println("The Aeroplane has crossed the city "+arg);

cityName = (String)arg;

}

public String getCityName() {

return cityName;

}

}

public class MonitorAeroplaneTest {

public static void main(String[] args) {

Aeroplane aeroplane = new Aeroplane();

AirTrafficController atc = new AirTrafficController();

aeroplane.addObserver(atc);

aeroplane.setTravelledCity("India");

System.out.println("Aeroplane is travelling towards : "+atc.getCityName());

aeroplane.setTravelledCity("Moscow");

System.out.println("Aeroplane is travelling towards : "+atc.getCityName());

}

}

Similarly you can think of an examples.

1. Service provider tracks the location of the mobile phone.

2. In case banking customer monitors the changes in the account.

Similarly you can achieve the above using ProertyChangeListener. Let us have a look.

import java.beans.PropertyChangeEvent;

import java.beans.PropertyChangeListener;

import java.beans.PropertyChangeSupport;

public class MyAccount implements PropertyChangeListener

{

private PropertyChangeSupport pcs = new PropertyChangeSupport(this);

private int amount = 5000;

public MyAccount() {

pcs.addPropertyChangeListener(this);

}

public int getAmount() {

return amount;

}

public void setAmount(int amount) {

pcs.firePropertyChange("amount", this.amount, amount);

this.amount = amount;

}

public void withDrawAmount() {

System.out.println("Amount is withdrawn ...");

}

@Override

public void propertyChange(PropertyChangeEvent evt) {

System.out.println("Name = " + evt.getPropertyName());

System.out.println("Old Value = " + evt.getOldValue());

System.out.println("New Value = " + evt.getNewValue());

}

}

public class TestAccount {

public static void main(String[] args) {

MyAccount act = new MyAccount();

act.setAmount(111);

act.withDrawAmount();

act.setAmount(11);

act.withDrawAmount();

}

**}**