宜宾学院

软件设计模式与体系结构设计报告书

学 院: 人工智能与大数据学部 班 级: 2018级9班

学生姓名: 杨雪 学 号: 200109327

设计地点（单位） 6305

设计题目: 软件设计模式与软件设计-实验7

完成日期： 2021 年 6 月 12 日

# 1 实验-观察者模式

1.1 题干要求

理解观察者模式，根据UML图还原原始代码

1.2 设计思路

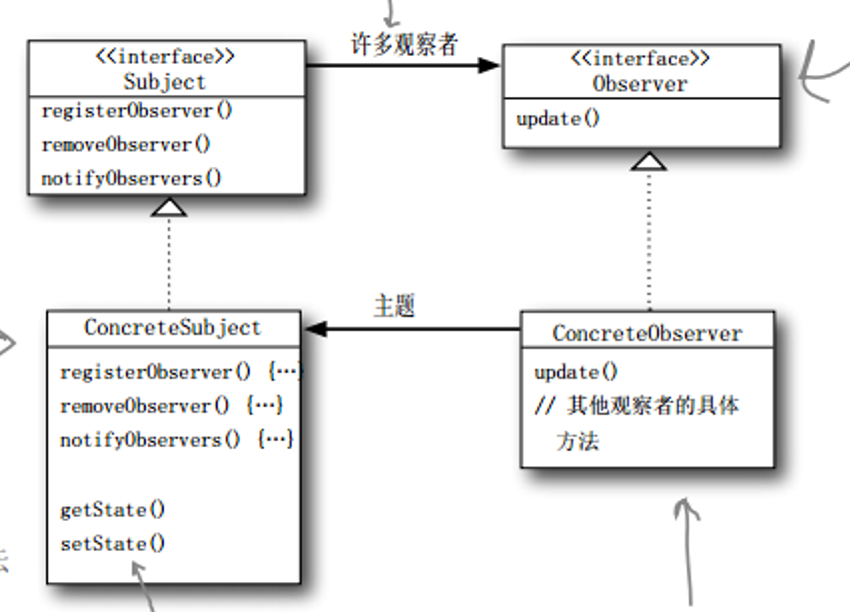


图 1‑1 观察者UML图

1.3 编码与测试

编码：

/\*\*

\* 1、理解观察者模式，根据UML图还原原始代码

\*/

**package** com.test.seven;

**import** java.util.ArrayList;

// 主题

**interface** Subject{

**public** **void** registerObserver(Observer observer);

**public** **void** removeObserver(Observer obserber);

**public** **void** notifyObservers();

}

// 具体主题

**class** ConcreteSubject **implements** Subject{

**private** ArrayList<Observer> list;

**private** String message;

**public** ConcreteSubject() {

list = **new** ArrayList<Observer>();

}

@Override

**public** **void** registerObserver(Observer observer) {

list.add(observer);

}

@Override

**public** **void** removeObserver(Observer obserber) {

**if**(!list.isEmpty()) {

list.remove(obserber);

}

}

@Override

**public** **void** notifyObservers() {

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

Observer obserber = list.get(i);

obserber.update(message);

}

}

**public** **void** setInfomation(String s) {

**this**.message = s;

System.***out***.println("天气服务更新消息： " + s);

//消息更新，通知所有观察者

notifyObservers();

}

}

// 观察者

**interface** Observer{

**public** **void** update(String message);

}

// 具体观察者

**class** ConcreteObserver **implements** Observer{

**private** String name;

**private** String message;

**public** ConcreteObserver(String name) {

**super**();

**this**.name = name;

}

@Override

**public** **void** update(String message) {

**this**.message = message;

read();

}

**public** **void** read() {

System.***out***.println(name + " 收到推送消息： " + message);

}

}

// 测试

**public** **class** ObserverTest01 {

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

ConcreteSubject concreteSubject = **new** ConcreteSubject();

ConcreteObserver zhangObserver = **new** ConcreteObserver("张三");

ConcreteObserver liObserver = **new** ConcreteObserver("李四");

ConcreteObserver wangObserver = **new** ConcreteObserver("王俊");

concreteSubject.registerObserver(zhangObserver);

concreteSubject.registerObserver(liObserver);

concreteSubject.registerObserver(wangObserver);

concreteSubject.setInfomation("今天的太阳真大！！！");

System.***out***.println("======================");

concreteSubject.removeObserver(liObserver);

concreteSubject.setInfomation("今天不适合旅游，容易中暑");

}

}

测试：

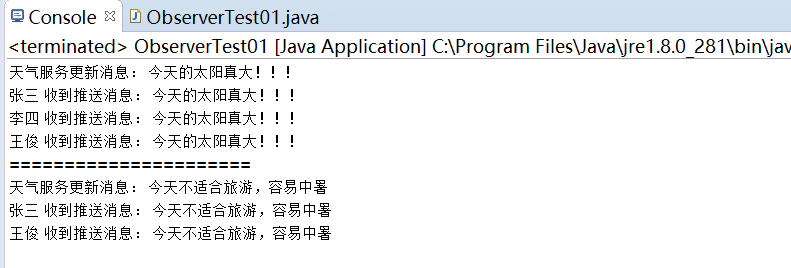
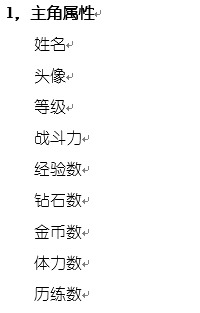
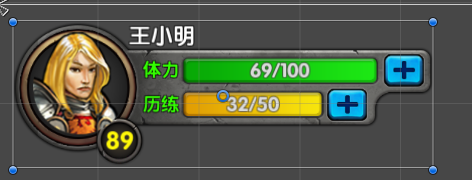


图 1‑2 观察者模式测试截图

# 2 观察者--实战1

2.1 题干要求

现假设要开发一个游戏UI系统，分为2个面板，对应的属性如下所示：





请使用观察者模式，实现更新UI的框架代码，即：主角属性是由服务器获取，然后分发给对应的面板，进行更新。

提示：参考观察者模式将需要观察的属性“所属面板”定义为对应的观察者，从而确定自己需要注册的“主题”。UML如图1-2所示。

2.2 设计思路



图 2‑0‑1 组合模式UML图

2.3 编码与测试

编码：

/\*\*

\* 请使用观察者模式，实现更新UI的框架代码，即：主角属性是由服务器获取，然后分发给对应的面板，进行更新。

\*/

**package** com.test.seven;

**import** java.util.ArrayList;

// 主角属性

**class** Attribute {

// 姓名

**private** String name;

// 头像

**private** String imgurl;

// 等级

**private** **int** grade = 0;

// 战斗力

**private** **int** combateffectiveness = 0;

// 经验数

**private** **int** experience = 0;

// 砖石数

**private** **int** diamonds = 0;

// 金币数

**private** **int** goldcoin = 0;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getImgurl() {

**return** imgurl;

}

**public** **void** setImgurl(String imgurl) {

**this**.imgurl = imgurl;

}

**public** **int** getGrade() {

**return** grade;

}

**public** **void** setGrade(Integer grade) {

**this**.grade = grade;

}

**public** **int** getCombateffectiveness() {

**return** combateffectiveness;

}

**public** **void** setCombateffectiveness(Integer combateffectiveness) {

**this**.combateffectiveness = combateffectiveness;

}

**public** **int** getExperience() {

**return** experience;

}

**public** **void** setExperience(Integer experience) {

**this**.experience = experience;

}

**public** **int** getDiamonds() {

**return** diamonds;

}

**public** **void** setDiamonds(Integer diamonds) {

**this**.diamonds = diamonds;

heroLevel();

}

**public** **int** getGoldcoin() {

**return** goldcoin;

}

**public** **void** setGoldcoin(Integer goldcoin) {

**this**.goldcoin = goldcoin;

}

**public** Attribute(String name, String imgurl, **int** grade, **int** combateffectiveness, **int** experience) {

**super**();

**this**.name = name;

**this**.imgurl = imgurl;

**this**.grade = grade;

**this**.combateffectiveness = combateffectiveness;

**this**.experience = experience;

heroLevel();

}

**public** Attribute(**int** diamonds, **int** goldcoin) {

**super**();

**this**.diamonds = diamonds;

**this**.goldcoin = goldcoin;

heroLevel();

}

@Override

**public** String toString() {

**return** "Attribute [name=" + name + ", imgurl=" + imgurl + ", grade=" + grade + ", combateffectiveness="

+ combateffectiveness + ", experience=" + experience + ", diamonds=" + diamonds + ", goldcoin="

+ goldcoin + "]";

}

**protected** **void** heroLevel() {

**this**.grade = **this**.experience / 1000 + (**this**.experience % 1000 != 0 ? 1 : 0);

}

}

// 主题

**interface** GameSubject {

**public** **void** registerObserver(GameplayerObserver o);

**public** **void** removeObserver(GameplayerObserver o);

**public** **void** notifyObservers();

}

// 具体主题

**class** RoleSubject **implements** GameSubject {

**private** ArrayList<GameplayerObserver> gameList;

**private** Attribute attribute;

**public** RoleSubject(Attribute attribute) {

**super**();

gameList = **new** ArrayList<GameplayerObserver>();

**this**.attribute = attribute;

}

@Override

**public** **void** registerObserver(GameplayerObserver o) {

**int** i = gameList.indexOf(o);

**if** (**null** != o && i < 0) {

gameList.add(o);

}

}

@Override

**public** **void** removeObserver(GameplayerObserver o) {

**int** i = gameList.indexOf(o);

**if** (i > 0) {

gameList.remove(o);

}

}

@Override

**public** **void** notifyObservers() {

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

GameplayerObserver o = gameList.get(i);

o.update(attribute);

}

}

**public** **void** updataRole(String name, String imgurl, **int** combateffectiveness, **int** experience ) {

**this**.attribute.setName(name);

**this**.attribute.setImgurl(imgurl);

**this**.attribute.setCombateffectiveness(combateffectiveness);

**this**.attribute.setExperience(experience);;

notifyObservers();

}

**public** **void** updataRole(**int** diamonds, **int** goldcoin) {

**this**.attribute.setDiamonds(diamonds);

**this**.attribute.setGoldcoin(goldcoin);

notifyObservers();

}

}

// 观察者

**interface** GameplayerObserver {

**void** update(Attribute attribute);

}

// 具体观察者

**class** ColorObserver **implements** GameplayerObserver {

**private** Attribute attribute;

**private** String name;

**public** ColorObserver(String name) {

**super**();

**this**.name = name;

}

@Override

**public** **void** update(Attribute attribute) {

System.***out***.println(name);

System.***out***.println(attribute.getName() + "\t头像" + attribute.getImgurl());

System.***out***.println("等级：" + attribute.getGrade() + "\t体力：" + attribute.getCombateffectiveness());

System.***out***.println("经验数：" + attribute.getExperience());

}

}

**class** PanelObserver **implements** GameplayerObserver {

**private** Attribute attribute;

**private** String name;

**public** PanelObserver(String name) {

**super**();

**this**.name = name;

}

@Override

**public** **void** update(Attribute attribute) {

System.***out***.println(name);

System.***out***.println("砖石数：" + attribute.getDiamonds() + "\t金币数：" + attribute.getGoldcoin());

}

}

// 测试

**public** **class** ObserverTest02 {

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

Attribute attribute1 = **new** Attribute("suave", "https://7.dusays.com/2021/06/10/15050d2a12105.jpg", 1000, 1000, 1000);

RoleSubject roleSubject = **new** RoleSubject(attribute1);

ColorObserver colorObserver = **new** ColorObserver("面板1");

roleSubject.registerObserver(colorObserver);

roleSubject.updataRole("suave","https://7.dusays.com/2021/06/10/15050d2a12105.jpg",2467, 2123);

System.***out***.println("----------------------------------------------------------------");

Attribute attribute2 = **new** Attribute(1000, 1000);

RoleSubject roleSubject2 = **new** RoleSubject(attribute2);

PanelObserver panelObserver = **new** PanelObserver("面板2");

roleSubject2.registerObserver(panelObserver);

roleSubject2.updataRole(2123, 9810);

}

}

测试：

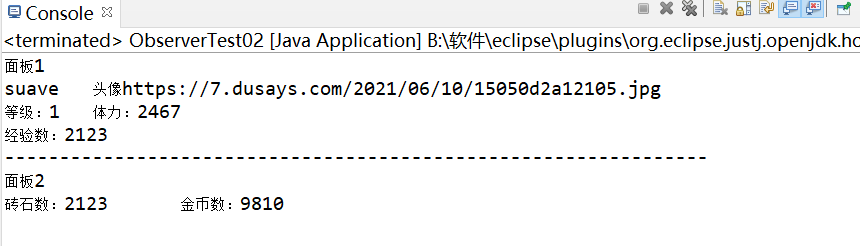


图 2‑2 观察者模式测试截图

# 3 观察者--实战2

3.1 题干要求

假设要实现一个找工作的中介网站，中介网站的职责是让求职者注册，然后当有适合的工作的时候，发送通知给对应的求职者。假设工作类别分为：软件开发人员、硬件开发人员、UI设计人员。请利用观察者模式，实现这个中介网站的框架代码。

提示：参考观察者模式将需要观察的“工作类别和求职者”定义为对应的观察者，从而确定自己需要注册的“主题”。一个求职者和自己关心的工作类别的关系可以通过“用户id-[工作类别，工作类别]”的方式进行绑定处理

UML如图3-1所示。

3.2 设计思路



图 3‑1 组合模式UML图

.3 编码与测试

编码：

/\*\*

\* 假设要实现一个找工作的中介网站，中介网站的职责是让求职者注册，然后当有适合的工作的时候，发送通知给对应的求职者。

\* 假设工作类别分为：软件开发人员、硬件开发人员、UI设计人员。请利用观察者模式，实现这个中介网站的框架代码。

\*/

**package** com.test.seven;

**import** java.util.ArrayList;

//主题

**interface** JobSubject{

**public** **void** registerObserver(UserObserver o);

**public** **void** removeObserver(UserObserver o);

**public** **void** notifyObservers();

}

//软件开发人员主题

**class** SoftwareDevelopers **implements** JobSubject{

**private** ArrayList<UserObserver> softlist;

**private** ArrayList<String> jobs;

**public** SoftwareDevelopers() {

softlist = **new** ArrayList<UserObserver>();

jobs = **new** ArrayList<String>();

}

@Override

**public** **void** registerObserver(UserObserver o) {

softlist.add(o);

}

@Override

**public** **void** removeObserver(UserObserver o) {

**if**(!softlist.isEmpty()) {

softlist.remove(o);

}

}

@Override

**public** **void** notifyObservers() {

**if** (**null** != softlist && softlist.size() > 0) {

**for** (UserObserver o : softlist) {

o.update(**this**);

}

}

}

**public** **void** addJob(String job) {

**this**.jobs.add(job);

notifyObservers();

}

**public** ArrayList<String> getJobs() {

**return** jobs;

}

**public** String toString(){

StringBuffer jobsString = **new** StringBuffer("软件开发新岗位： ");

**for**(String j: jobs){

jobsString.append(j + ", ");

}

**return** jobsString.toString();

}

}

//硬件开发人员主题

**class** HardwareDevelopers **implements** JobSubject{

**private** ArrayList<UserObserver> softlist;

**private** ArrayList<String> jobs;

**public** HardwareDevelopers() {

softlist = **new** ArrayList<UserObserver>();

jobs = **new** ArrayList<String>();

}

@Override

**public** **void** registerObserver(UserObserver o) {

softlist.add(o);

}

@Override

**public** **void** removeObserver(UserObserver o) {

**if**(!softlist.isEmpty()) {

softlist.remove(o);

}

}

@Override

**public** **void** notifyObservers() {

**if** (**null** != softlist && softlist.size() > 0) {

**for** (UserObserver o : softlist) {

o.update(**this**);

}

}

}

**public** **void** addJob(String job) {

**this**.jobs.add(job);

notifyObservers();

}

**public** ArrayList<String> getJobs() {

**return** jobs;

}

**public** String toString(){

StringBuffer jobsString = **new** StringBuffer("硬件开发新岗位： ");

**for**(String j: jobs){

jobsString.append(j + ", ");

}

**return** jobsString.toString();

}

}

//UI设计人员主题

**class** UIDesigners **implements** JobSubject{

**private** ArrayList<UserObserver> softlist;

**private** ArrayList<String> jobs;

**public** UIDesigners() {

softlist = **new** ArrayList<UserObserver>();

jobs = **new** ArrayList<String>();

}

@Override

**public** **void** registerObserver(UserObserver o) {

softlist.add(o);

}

@Override

**public** **void** removeObserver(UserObserver o) {

**if**(!softlist.isEmpty()) {

softlist.remove(o);

}

}

@Override

**public** **void** notifyObservers() {

**if** (**null** != softlist && softlist.size() > 0) {

**for** (UserObserver o : softlist) {

o.update(**this**);

}

}

}

**public** **void** addJob(String job) {

**this**.jobs.add(job);

notifyObservers();

}

**public** ArrayList<String> getJobs() {

**return** jobs;

}

**public** String toString(){

StringBuffer jobsString = **new** StringBuffer("UI新岗位： ");

**for**(String j: jobs){

jobsString.append(j + ", ");

}

**return** jobsString.toString();

}

}

//观察者

**interface** UserObserver{

**public** **void** update(JobSubject s);

}

//具体观察者

**class** User **implements** UserObserver{

**private** String name;

**public** User(String name) {

**super**();

**this**.name = name;

}

@Override

**public** **void** update(JobSubject s) {

System.***out***.println(name + "收到消息：" + s.toString());

}

}

// 测试

**public** **class** ObserverTest03 {

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

SoftwareDevelopers soft = **new** SoftwareDevelopers();

HardwareDevelopers hard = **new** HardwareDevelopers();

UIDesigners ui = **new** UIDesigners();

soft.registerObserver(**new** User("张三"));

hard.registerObserver(**new** User("李四"));

ui.registerObserver(**new** User("王三"));

soft.addJob("软件开发人员");

hard.addJob("硬件开发人员");

ui.addJob("UI设计人员");

}

}

测试：

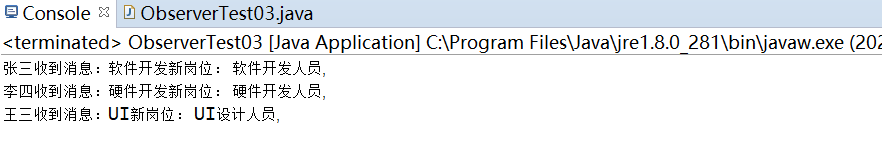


图 3‑2 观察者模式测试截图