1、观察器模型

分为事务，观察器，客户。观察器将获取的事务实时传达给客户的模型

(1)创建Affairs类

Affairs.java

import java.util.ArrayList;

import java.util.List;

public class Affairs {

private List<Observer> observers

= new ArrayList<Observer>();

private int state;

public int getState() {

return state;

}

public void setState(int state) {

this.state = state;

notifyAllObservers();

}

public void attach(Observer observer){

observers.add(observer);

}

public void notifyAllObservers(){

for (Observer observer : observers) {

observer.update();

}

}

}

(2)创建Observer类

Observer.java

public abstract class Observer {

protected Affairs affairs;

public abstract void update();

}

(3)创建实体Observer类

BinaryObserver.java

public class BinaryObserver extends Observer{

public BinaryObserver(Affairs affairs){

this. affairs = affairs;

this. affairs.attach(this);

}

@Override

public void update() {

System.out.println( "Binary String: "

+ Integer.toBinaryString( affairs.getState() ) );

}

}

OctalObserver.java

public class OctalObserver extends Observer{

public OctalObserver(Affairs affairs){

this. affairs = affairs;

this. affairs.attach(this);

}

@Override

public void update() {

System.out.println( "Octal String: "

+ Integer.toOctalString( affairs.getState() ) );

}

}

HexaObserver.java

public class HexaObserver extends Observer{

public HexaObserver(Affairs affairs){

this. affairs = affairs;

this. affairs.attach(this);

}

@Override

public void update() {

System.out.println( "Hex String: "

+ Integer.toHexString( affairs.getState() ).toUpperCase() );

}

}

(4)使用affairs和实体Observer类

ObserverPatternDemo.java

public class ObserverPatternDemo {

public static void main(String[] args) {

Affairs affairs = new Affairs ();

new HexaObserver(affairs);

new OctalObserver(affairs);

new BinaryObserver(affairs);

System.out.println("First state change: 15");

affairs.setState(15);

System.out.println("Second state change: 10");

affairs.setState(10);

}

}

2、恢复模型

分为历史记录，发起者，恢复。例如在windows中输入ctrl+z可以恢复原有记录

(1)创建Memory类

Memory.java

public class Memory{

private String state;

public Memory (String state){

this.state = state;

}

public String getState(){

return state;

}

}

(2)创建Origin类

Origin.java

public class Origin{

private String state;

public void setState(String state){

this.state = state;

}

public String getState(){

return state;

}

public Memory saveStateToMemory (){

return new Memory (state);

}

public void getStateFromMemory (Memory Memory){

state = Memory.getState();

}

}

(3)创建Restore类

Restore.java

import java.util.ArrayList;

import java.util.List;

public class Restore{

private List< Memory > memoryList = new ArrayList< Memory >();

public void add(Memory state){

memoryList.add(state);

}

public Memory get(int index){

return memoryList.get(index);

}

}

(4)使用Origin和Restore

MemoryPatternDemo.java

public class MemoryPatternDemo {

public static void main(String[] args) {

Origin origin = new Origin();

Restore restore= new Restore ();

origin.setState("State #1");

origin.setState("State #2");

Restore.add(origin.saveStateToMemory());

origin.setState("State #3");

Restore.add(origin.saveStateToMemory());

origin.setState("State #4");

System.out.println("Current State: " + origin.getState());

origin.getStateFromMemory(Restore.get(0));

System.out.println("First saved State: " + origin.getState());

origin.getStateFromMemory(Restore.get(1));

System.out.println("Second saved State: " + origin.getState());

}

}