Question No 1:

Modify pattern use ObserverPatternDemo such that you change observer and then update the state. Only active observers are.

import java.util.ArrayList;

import java.util.List;

class Subject {

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

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 detach(Observer observer) {

observers.remove(observer);

}

public void notifyAllObservers() {

for (Observer observer : observers) {

observer.update();

}

}

}

abstract class Observer {

protected Subject subject;

public abstract void update();

}

class BinaryObserver extends Observer {

public BinaryObserver(Subject subject) {

this.subject = subject;

this.subject.attach(this);

}

@Override

public void update() {

System.out.println("Binary String: " + Integer.toBinaryString(subject.getState()));

}

}

class OctalObserver extends Observer {

public OctalObserver(Subject subject) {

this.subject = subject;

this.subject.attach(this);

}

@Override

public void update() {

System.out.println("Octal String: " + Integer.toOctalString(subject.getState()));

}

}

class HexaObserver extends Observer {

public HexaObserver(Subject subject) {

this.subject = subject;

this.subject.attach(this);

}

@Override

public void update() {

System.out.println("Hex String: " + Integer.toHexString(subject.getState()).toUpperCase());

}

}

public class ObserverPatternDemo {

public static void main(String[] args) {

Subject subject = new Subject();

HexaObserver hexaObserver = new HexaObserver(subject);

OctalObserver octalObserver = new OctalObserver(subject);

BinaryObserver binaryObserver = new BinaryObserver(subject);

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

subject.setState(15);

// Change active observers

subject.detach(octalObserver);

subject.attach(new BinaryObserver(subject));

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

subject.setState(10);

}

}

Complete code.

package observer.eventsmanagement;

import java.io.File;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

interface EventListener {

void update(String eventType, File file);

}

class EmailNotificationListener implements EventListener {

private String email;

public EmailNotificationListener(String email) {

this.email = email;

}

@Override

public void update(String eventType, File file) {

System.out.println("Email to " + email + ": Someone has performed " + eventType + " operation with the following file: " + file.getName());

}

}

class LogOpenListener implements EventListener {

private File log;

public LogOpenListener(String fileName) {

this.log = new File(fileName);

}

@Override

public void update(String eventType, File file) {

System.out.println("Save to log " + log + ": Someone has performed " + eventType + " operation with the following file: " + file.getName());

}

}

class SMSSupportListener implements EventListener {

private String phoneNumber;

private String defaultSMS = "This is a notification about a file operation.";

public SMSSupportListener(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

@Override

public void update(String eventType, File file) {

String message = "SMS to " + phoneNumber + ": " + eventType + " operation performed on " + file.getName();

if (defaultSMS.length() > 160) {

System.out.println("Warning: Default SMS length exceeds 160 characters. Please define a valid SMS.");

} else {

System.out.println(message);

}

}

}

class EventManager {

private Map<String, List<EventListener>> listeners = new HashMap<>();

public EventManager(String... operations) {

for (String operation : operations) {

this.listeners.put(operation, new ArrayList<>());

}

}

public void subscribe(String eventType, EventListener listener) {

List<EventListener> users = listeners.get(eventType);

if (users != null) {

users.add(listener);

}

}

public void unsubscribe(String eventType, EventListener listener) {

List<EventListener> users = listeners.get(eventType);

if (users != null) {

users.remove(listener);

}

}

public void notify(String eventType, File file) {

List<EventListener> users = listeners.get(eventType);

if (users != null) {

for (EventListener listener : users) {

listener.update(eventType, file);

}

}

}

}

class Editor {

public EventManager events;

private File file;

public Editor() {

this.events = new EventManager("open", "save");

}

public void openFile(String filePath) {

this.file = new File(filePath);

events.notify("open", file);

}

public void saveFile() throws Exception {

if (this.file != null) {

events.notify("save", file);

} else {

throw new Exception("Please open a file first.");

}

}

}

public class ObserverPatternDemo {

public static void main(String[] args) {

Editor editor = new Editor();

editor.events.subscribe("open", new LogOpenListener("log.txt"));

editor.events.subscribe("save", new EmailNotificationListener("admin@example.com"));

editor.events.subscribe("save", new SMSSupportListener("+123456789"));

try {

editor.openFile("test.txt");

editor.saveFile();

} catch (Exception e) {

e.printStackTrace();

}

}

}