观察者模式实验

**【实验目的及要求】**

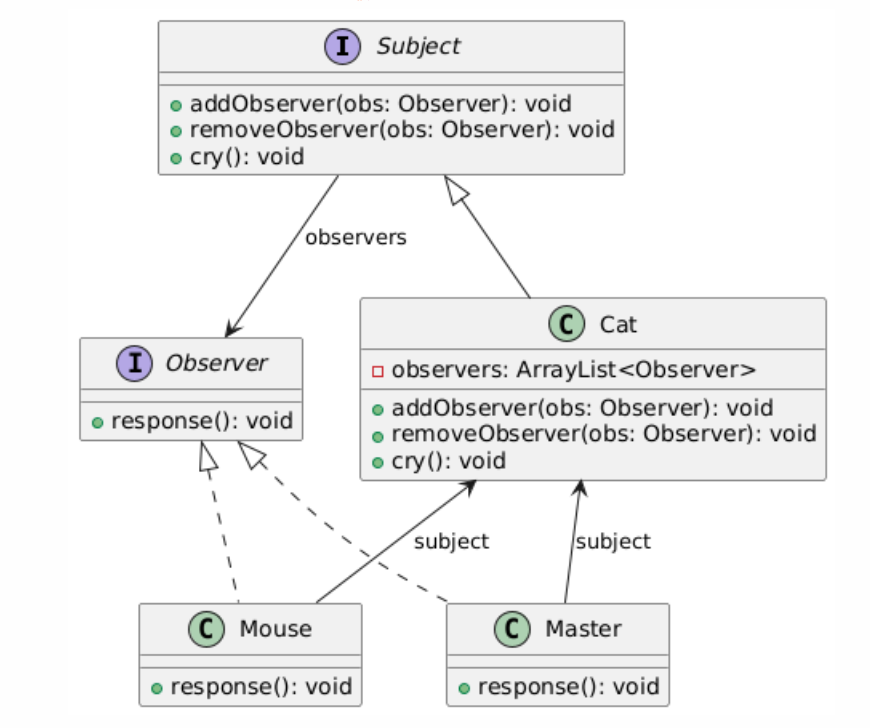
1. 熟悉观察者模式的定义；
2. 掌握观察者模式角色及作用，熟悉观察者模式结构图；
3. 掌握观察者模式编程。

**【实验内容】**

**“猫（****Cat）大叫一声，老鼠（Mouse）开始逃跑，****主人（Master）被惊醒”。请使用观察者模式模拟此过程。**

**【实验步骤】**

1. **类图**

****

1. **实现代码**
2. **抽象主题接口（Subject）**

**package** design.observer;

//抽象主题

**public** **interface** Subject {

**public** **void** addObserver(Observer obs); //添加观察者

**public** **void** removeObserver(Observer obs); //移除观察者

**public** **void** cry();

}

1. **抽象观察者接口（Obsrver）**

**package** design.observer;

//抽象观察者

**public** **interface** Observer {

**public** **void** response();

}

1. **具体主题类**

**public class Cat implements Subject{**

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

**@Override**

**public void addObserver(Observer obs) {**

**observers.add(obs);**

**}**

**@Override**

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

**observers.remove(obs);**

**}**

**// 遍历观察者，调用观察者的响应方法**

**@Override**

**public void cry() {**

**for (Observer obs : observers) {**

**obs.response();**

**}**

**}**

**}**

1. **具体观察者类**

**public class Mouse implements Observer{  
  
 @Override  
 public void response() {  
 System.*out*.println("老鼠（Mouse）开始逃跑");  
 }  
}**

**public class Master implements Observer {  
 @Override  
 public void response() {  
 System.*out*.println("主人（Master）被惊醒");  
 }  
}**

1. **客户端测试类Client**

**package** design.iterator;

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

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

Cat cat = new Cat();

cat.addObserver(new Mouse());

cat.addObserver(new Master());

cat.cry();

}

}

**【思考】**

1. 使用JDK中提供的Observer接口、Observable类简化前面的代码。
2. 查找资料了解发布订阅模式，并将上例改成发布订阅模式。

**（1）**

**public class Cat extends Observable {**

**public void cry() {**

**// 设置状态已改变，通知观察者**

**setChanged();**

**notifyObservers();**

**}**

**}**

**public class Master implements Observer {**

**@Override**

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

**System.out.println("主人（Master）被惊醒");**

**}**

**}**

**public class Mouse implements Observer {**

**@Override**

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

**System.out.println("老鼠（Mouse）开始逃跑");**

**}**

**}**

**public class Test {**

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

**// 创建主题（猫）**

**Cat cat = new Cat();**

**// 创建观察者（老鼠和主人）**

**Mouse mouse = new Mouse();**

**Master master = new Master();**

**// 添加观察者**

**cat.addObserver(mouse);**

**cat.addObserver(master);**

**// 触发猫叫**

**cat.cry();**

**}**

**}**

**（2）添加事件总线，负责管理订阅者，Cat的职责只是触发发布订阅的事件，不负责管理订阅者。下面是有改动的代码，其他的与初始的一致**

**public class EventBus {**

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

**// 添加订阅者**

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

**observers.add(observer);**

**}**

**// 移除订阅者**

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

**observers.remove(observer);**

**}**

**// 发布事件**

**public void publish() {**

**for (Observer observer : observers) {**

**observer.response();**

**}**

**}**

**}**

**public class Cat {**

**private EventBus eventBus;**

**// 通过构造函数传入事件总线**

**public Cat(EventBus eventBus) {**

**this.eventBus = eventBus;**

**}**

**// 触发事件总线发布事件**

**public void cry() {**

**eventBus.publish();**

**}**

**}**

**public class Test {**

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

**EventBus eventBus = new EventBus();**

**Cat cat = new Cat(eventBus);**

**// 创建观察者**

**Mouse mouse = new Mouse();**

**Master master = new Master();**

**// 由事件总线添加观察者**

**eventBus.subscribe(mouse);**

**eventBus.subscribe(master);**

**// 触发猫叫事件，通知所有订阅者**

**cat.cry();**

**}**

**}**

**【总结】**

|  |  |
| --- | --- |
| 实验总结  （不少于100字） | 本次实验通过实现观察者模式，加深了对其定义和应用的理解。观察者模式是一种行为设计模式，其主要作用是实现对象之间的解耦，允许一个对象的状态变化自动通知所有依赖于它的对象。在实验中，我们通过定义主题（Subject）和观察者（Observer）接口，以及具体的实现类，模拟了“猫叫、老鼠逃跑、主人被惊醒”的场景。通过编程实现，我掌握了观察者模式的角色分配和作用，使得主题和观察者之间通过发布和订阅机制实现消息通知，且两者之间不存在直接的依赖关系。此外，我还通过扩展为发布订阅模式，进一步加深了对观察者模式的理解和实际编程能力的提升。 |
| 学号 | 22154A206 |
| 姓名 | 冯艳芳 |
| 成绩 |  |