软件体系结构

《软件体系结构作业十六》

学　号 22920212204396

姓　名 黄子安

2024 年 5 月 8 日

一、利用 JDK 的 java.util 包中提供的 Observable 类以及 Observer 接口实现课堂的例子（对随机数的观察输出）

在本例中 NumberGenerator 是被观察的对象，而 GraphObserver 和 DigitObserver 是观察者对象，Generator 中的随机数发生变化时会影响 Observer

具体代码如下，让 NumberGenerator 继承 Observable 对象，当发生改变时会调用 notifyObservers()方法，进而通知所有的观察者

根据对应接口的源码可以知道 Obserable 中有一个成员变量 changed 用于标志被观察者有发生变化，在 notifyObservers()中如果发现 changed 是 false 会直接返回不会响应，所以需要在通知前使用 setChanged()将 changed 标志置为 true，这样设置可以保证调用通知的时候被观察者一定是发生变化了，可以减少不必要的调用通知，保证程序的性能，同时也可以更加灵活编程，比如可以实现多次改变对象的状态，但只在所有修改完成后通知观察者。

```
public class NumberGenerator extends Observable {
    private Random random = new Random();
    private Integer number;

    public Integer getNumber() {
        return this.number;
    }

    public void execute() {
        for(int i = 0; i < 10; ++i) {
            number = random.nextInt(50);
            setChanged();
            notifyObservers();
        }
    }
}
```

之后定义两个观察者，需要实现 Observer 接口并重写对应的 update 函数，在被观察者发生变化之后观察者做出对应的响应

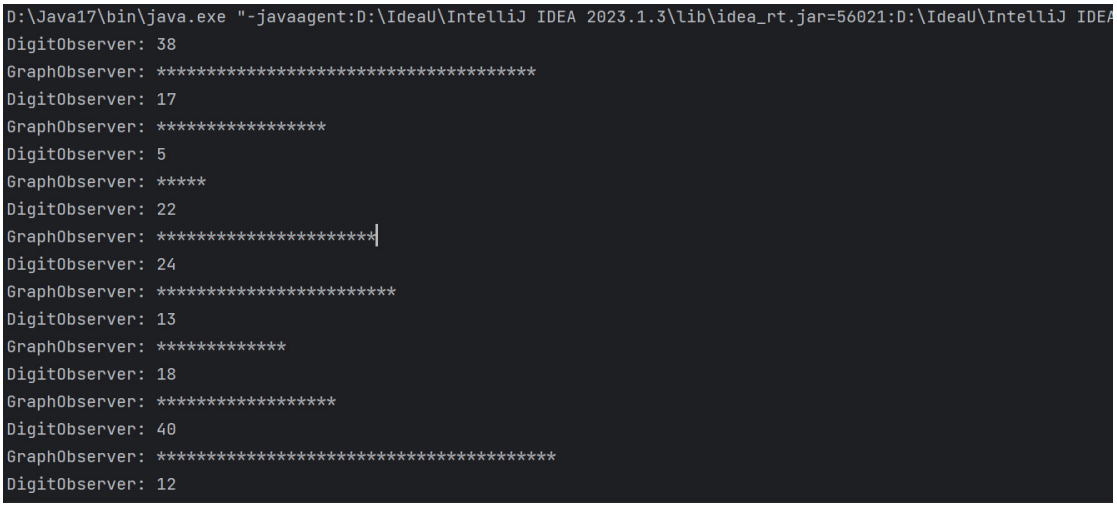
```
public class GraphObserver implements Observer {

    @Override
    public void update(Observable o, Object arg) {
        System.out.print("GraphObserver: ");

        int count = ((NumberGenerator) o).getNumber();
        for(int i = 0; i < count; ++i)
        {
            System.out.print("*");
        }
        System.out.println();

        try{
            Thread.sleep(100);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
public class DigitObserver implements Observer {

    @Override
    public void update(Observable o, Object arg) {
        System.out.println("DigitObserver: " + ((NumberGenerator) o).getNumber());

        try{
            Thread.sleep(100);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
```

最后编写客户端用于测试，通过使用 addObserver 方法将两个观察者加入到被观察者对象当中

```
public class Main {
    public static void main(String[] args) {
        NumberGenerator generator = new NumberGenerator();
        GraphObserver graphObserver = new GraphObserver();
        DigitObserver digitObserver = new DigitObserver();
        generator.addObserver(graphObserver);
        generator.addObserver(digitObserver);
        generator.execute();
    }
}
```

最后输出如下，与课堂代码效果一致：



不过从 JDK 9 后这两个类和接口已经被废除了，一个原因是不可序列化，因为 Observable 没有实现可序列化，因此不能将 Observable 及其子类序列化，这会带来很多限制；另外也不能保证线程安全，这些方法可以被其子类覆盖，事件通知可以以不同的顺序发生，并且可能发生在不同的线程上，状态更新与通知不能一一对应，这足以破坏线程安全。

后续 java.beans 包提供的 PropertyChangeEvent 和 PropertyChangeListener 可以更有效实现观察者模式