20 Observer pattern

Observer pattern

Intent

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

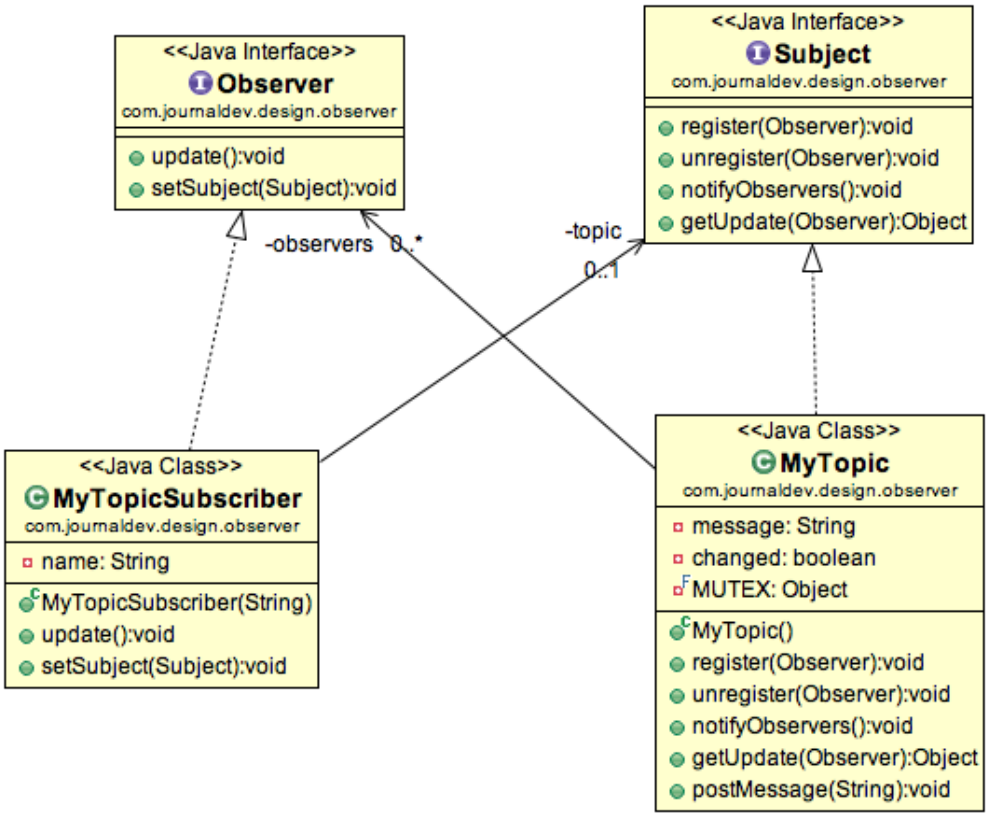
Note

- Useful when you are interested in the state of an object and want to get notified whenever there is any change.
- The object that ***watch*** on the state of another object are called **Observer**.
- The object that is ***being watched*** is called **Subject.**

>> ผู้ถูกเฝ้าดู (Subject) notify ไปให้หลายผู้เฝ้า (Observer) เมื่อตัวเองเปลี่ยน state

Reference

[1] Observer Design Pattern in Java
https://www.journaldev.com/1739/observer-design-pattern-in-java




```
1. public interface Subject {
2.
3.     //methods to register and unregister observers
4.     public void register(Observer obj);
5.     public void unregister(Observer obj);
6.
7.     //method to notify observers of change
8.     public void notifyObservers();
9.
10.    //method to get updates from subject
11.    public Object getUpdate(Observer obj);
12.}
```

```
1. public class MyTopic implements Subject {
2.
3.     private List<Observer> observers;
4.     private String message;
5.     private boolean changed;
6.     private final Object MUTEX = new Object();
7.
8.     public MyTopic() {
9.         this.observers = new ArrayList<>();
10.    }
11.
12.    @Override
13.    public void register(Observer obj) {
14.        if (obj == null) {
15.            throw new NullPointerException("Null Observer");
16.        }
17./* Java synchronized : allowing only one thread to execute at any given time. */
18.        synchronized (MUTEX) {
19.            if (!observers.contains(obj)) {
20.                observers.add(obj);
21.            }
22.        }
```

```
    }

    @Override
    public void unregister(Observer obj) {
        synchronized (MUTEX) {
            observers.remove(obj);
        }
    }

    @Override
    public void notifyObservers() {
        List<Observer> observersLocal = null;
        //synchronization is used to make sure any observer registered after message is received is not notified
        synchronized (MUTEX) {
            if (!changed) {
                return;
            }
            observersLocal = new ArrayList<>(this.observers);
            this.changed = false;
        }

        for (Observer obj : observersLocal) {
            obj.update();
        }

    }

    @Override
    public Object getUpdate(Observer obj) {
        return this.message;
    }

    //method to post message to the topic
    public void postMessage(String msg) {
        System.out.println("Message Posted to Topic:" + msg);
        this.message = msg;
        this.changed = true;
        notifyObservers();
    }
}
```

```
public interface Observer {

    //method to update the observer, used by subject
    public void update();

    //attach with subject to observe
    public void setSubject(Subject sub);
}
```

```
public class MyTopicSubscriber implements Observer {

    private String name;
    private Subject topic;

    public MyTopicSubscriber(String nm) {
        this.name = nm;
    }

    @Override
    public void update() {
        String msg = (String) topic.getUpdate(this);
        if (msg == null) {
            System.out.println(name + ":: No new message");
```

```
        } else {
            System.out.println(name + ":: Consuming message::" + msg);
        }
    }

    @Override
    public void setSubject(Subject sub) {
        this.topic = sub;
    }

}
```

```
public class DemoObserver {

    public static void main(String[] args) {

        //create subject
        MyTopic topic = new MyTopic();

        //create observers
        Observer obj1 = new MyTopicSubscriber("Obj1");
        Observer obj2 = new MyTopicSubscriber("Obj2");
        Observer obj3 = new MyTopicSubscriber("Obj3");

        //register observers to the subject
        topic.register(obj1);
        topic.register(obj2);
        topic.register(obj3);

        //attach observer to subject
        obj1.setSubject(topic);
        obj2.setSubject(topic);
        obj3.setSubject(topic);

        //check if any update is available
        obj1.update();

        //now send message to subject
        topic.postMessage("sawasdee");
    }
}
```