6.7 Mẫu thiết kế OBSERVER

6.7.1 Mẫu Observer

Sport Lobby là một trang thể thao tuyệt vời. Nó bao phủ hầu hết mọi kiểu thể thao và cung cấp các tin tức mới nhất, thông tin, lịch các trận đấu, thông tin về từng cầu thủ và đội riêng biệt. Bây giờ họ lập kế hoạch cung cấp bình luận trực tiếp hoặc tỷ số các trận đấu như dịch vụ tin nhắn, nhưng chỉ cho các người sử dụng đặc biệt. Mục đích là tin nhắn SMS tỷ số trực tiếp, tình huống trận đấu và các sự kiện quan trọng sau khoảng thời gian ngắn. Như người sử dụng, bạn cần đăng ký cho gói này và khi có trận đấu diễn ra bạn sẽ nhận được SMS bình luận trực tiếp. Trang này cũng cung cấp lựa chọn hủy đăng ký khỏi gói khi bạn muốn.

Như người phát triển, Sport Lobby yêu cầu bạn cung cấp đặc tính mới này cho họ. Các bình luận viên sẽ ngồi ở buồng bình luận trong trận đấu và họ sẽ cập nhật bình luận trực tiếp cho các đối tượng bình luận. Như người phát triển, công việc của bạn là cung cấp bình luận cho những người sử dụng đã đăng ký bằng cách lấy chúng từ đối tượng bình luận khi nó đã sẵn sàng. Khi có cập nhật, hệ thống cần cập nhật cho các đối tượng đăng ký bằng cách gửi tin nhắn SMS cho họ.

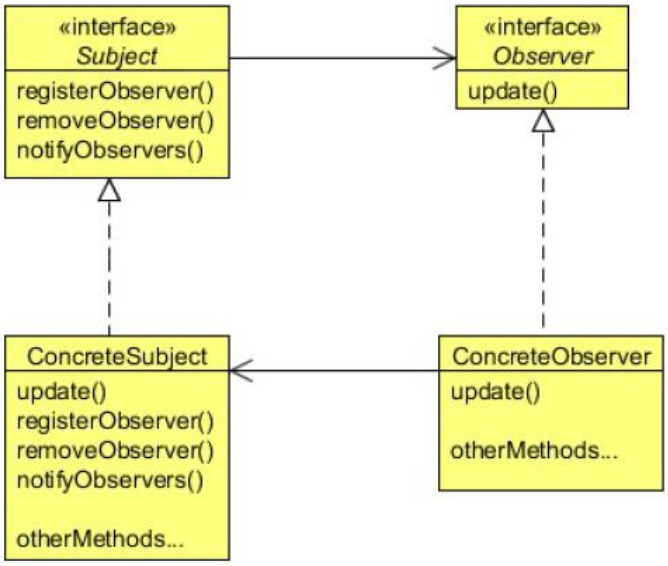
Tình huống này chỉ rõ ánh xạ một nhiều giữa trận đấu và người sử dụng như là có nhiều người sử dụng đăng ký đến một trận đấu. Mẫu thiết kế Observer là rất phù hợp cho tình huống này, ta sẽ xem xét mẫu này và sau đó tạo đặc tính đó cho Sport Lobby.

6.7.2 Mẫu Observer là gì

Mẫu Observer là kiểu mẫu hành vi mà liên quan đến việc giao trách nhiệm giữa các đối tượng. Các mẫu hành vi mô tả luồng điều khiển phức tạp mà khó theo dõi trong thời gian chạy. Chúng tách bạn khỏi luồng điều khiển và cho phép bạn chỉ tập trung vào cách đối tượng kết nối với nhau.

Mẫu Observer định nghĩa phụ thuộc một nhiều giữa các đối tượng sao cho khi một đối tượng thay đổi trạng thái, mọi cái phụ thuộc được thông báo và cập nhật tự động. Mẫu Observer mô tả các phụ thuộc này. Các đối tượng chính trong mẫu này là chủ đề *subject* và người quan sát *obsever*, Chủ đề có thể có nhiều *observer* phụ thuộc. Mọi *observer* được thông báo khi chủ đề có thay đổi trong trạng thái của nó. Ngược lại, mỗi *observer* sẽ truy vấn chủ đề để đồng bộ trạng thái của nó với trạng thái của chủ đề.

Một cách khác để hiểu mẫu *Observer* là cách quan hệ *Publisher – Subscriber* (đăng lên và đăng ký) làm việc. Giả sử bạn đăng ký tạp chí về môn thể thao mà bạn ưu thích hay tạp chí mẫu. Khi có số mới được in, nó sẽ được phân phối cho bạn. Nếu bạn hủy đăng ký nó, khi bạn không muốn có tạp chí nữa, nó sẽ không được phân phối đến bạn. Nhưng nhà xuất bản vẫn tiếp tục làm việc như trước kia, vì còn nhiều người khác cũng đã đăng ký cho tạp chí đó.

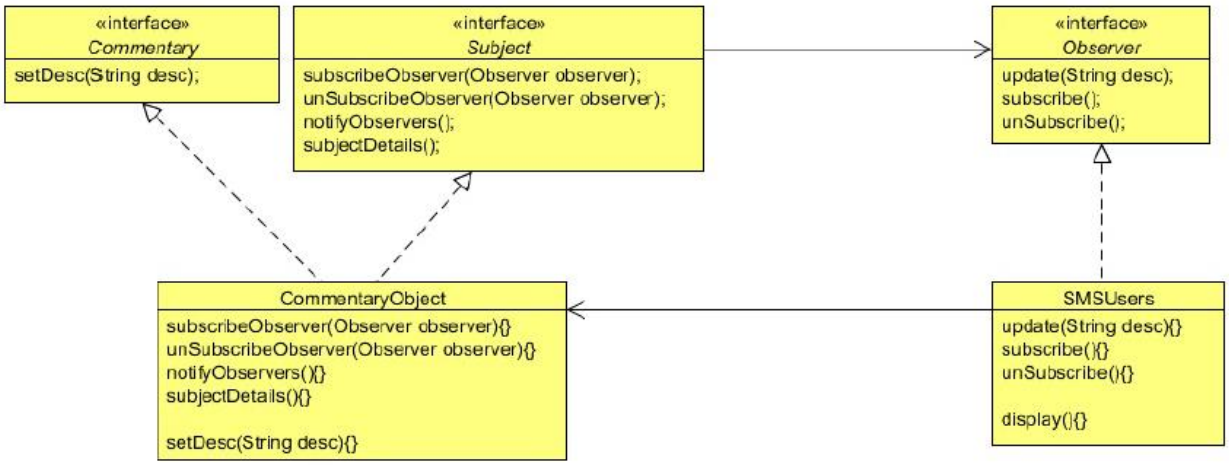


Có bốn thành phần trong mẫu *Observer*:

- Chủ đề, mà được sử dụng để đăng ký *observer*. Các đối tượng sử dụng giao diện này để đăng ký trở thành *observer* và cũng như tự hủy đăng ký khỏi *observer*.
- *Observer* định nghĩa giao diện update cho các đối tượng, mà muốn được thông báo các thay đổi của chủ đề. Mọi *observer* cần cài đặt giao diện *Obsever*. Giao diện này có phương thức *update ()* mà sẽ được gọi khi trạng thái của chủ để thay đổi.
- ConcreteSubject lưu giữ trạng thái quan tâm cho các đối tượng ConcreteObserver. Nó gửi thông báo cho các *observer* của nó khi trạng thái của nó thay đổi. Một chủ đề *Subject* cụ thể luôn cài đặt giao diện *Subject*. Phương thức *notifyObservers ()* được sử dụng để cập nhật mọi observer hiện tại khi trạng thái chủ đề thay đổi.
- ConcreteObserver duy trì tham chiếu đến đối tượng ConcreteSubject và cài đặt giao diện Observer. Mỗi *observer* đăng ký với chủ đề cụ thể để nhận các cập nhật.

6.7.3 Cài đặt mẫu Obsever

Chúng ta xem sử dụng mẫu Observer như thế nào trong việc phát triển đặc tính yêu cầu của Sport Lobby. Ai đó sẽ cập nhật đối tượng chủ đề và công việc của bạn là cập nhật trạng thái đối tượng đã đăng ký với đối tượng chủ đề cụ thể đó. Do đó, khi có thay đổi trạng thái của đối tượng chủ đề cụ thể, mọi đối tượng phụ thuộc nó sẽ nhận được thông báo và sẽ cập nhật.



Dựa trên điều đó, trước hết ta tạo giao diện Subject. Trong giao diện Subject có ba phương thức chính và tùy chọn, nếu yêu cầu bạn có thể bổ sung một vài phương thức theo yêu cầu.

```
package com.javacodegeeks.patterns.observerpattern;

public interface Subject {

        public void subscribeObserver(Observer observer);
        public void unSubscribeObserver(Observer observer);
        public void notifyObservers();
        public String subjectDetails();
}
```

Ba phương thức chính trong giao diện Subject là:

- *subscriberObserver*, mà được sử dụng để đăng ký observer hoặc ta có thể nói đăng ký observer sao cho nếu có thay đổi trong trạng thái của chủ đề đó, mọi observer này sẽ nhận được thông báo.
- *unsubscriberObserver*, mà được sử dụng để hủy đăng ký observer sao cho nếu có thay đổi trong trạng thái của chủ đề đó, observer hủy đăng ký này sẽ không được nhận thông báo nữa.
- *notifyObservers*, phương thức này thông báo các observer đã đăng ký khi có thay đổi trong trạng thái của chủ đề đó.

Và tùy chọn, ở đây có thêm một phương thức *subjectDetails ()*, nó là phương thức hiển nhiên và cần thiết. Ở đây công việc là trả về chi tiết của chủ đề đó.

Bây giờ chúng ta sẽ xét giao diện Observer.

```
package com.javacodegeeks.patterns.observerpattern;

public interface Observer {

        public void update(String desc);
        public void subscribe();
        public void unSubscribe();
}
```

- *update (String desc)* là phương thức được gọi bởi chủ đề trên observer để thông báo mô tả desc đó, khi có thay đổi trong trạng thái của chủ đề.
- *subscriber ()* là phương thức được sử dụng để đăng ký bản thân nó với chủ đề đó.
- *unsubscriber ()* là phương thức được sử dụng để hủy đăng ký bản thân nó với chủ đề đó.

```
package com.javacodegeeks.patterns.observerpattern;

public interface Commentary {

        public void setDesc(String desc);
}
```

Giao diện trên được sử dụng bởi các bình luận viên để cập nhật các bình luận trực tiếp về đối tượng bình luận. Đây là giao diện tùy chọn chỉ để code tuân thủ theo nguyên tắc giao diện, chứ không liên quan gì đến mẫu Observer. Bạn cần áp dụng các nguyên lý hướng đối tượng với mẫu thiết kế mỗi khi áp dụng. Giao diện chỉ chứa một phương thức mà được sử dụng để thay đổi trạng thái của đối tượng chủ đề cụ thể.

```
package com.javacodegeeks.patterns.observerpattern;

import java.util.List;

public class CommentaryObject implements Subject,Commentary{

        private final List<Observer>observers;
        private String desc;
        private final String subjectDetails;

        public CommentaryObject(List<Observer>observers,String subjectDetails){
                this.observers = observers;
                this.subjectDetails = subjectDetails;
        }
        @Override
        public void subscribeObserver(Observer observer) {
                observers.add(observer);
        }

        @Override
        public void unSubscribeObserver(Observer observer) {
                int index = observers.indexOf(observer);
                observers.remove(index);

        }

        @Override
        public void notifyObservers() {
                System.out.println();
                for(Observer observer : observers){
                        observer.update(desc);
                }

        }

        @Override
        public void setDesc(String desc) {
                this.desc = desc;
                notifyObservers();
        }
        @Override
        public String subjectDetails() {
                return subjectDetails;
        }

}
```

Lớp trên làm việc như chủ đề cụ thể mà cài đặt giao diện Subject và cung cấp cài đặt của nó. Nó cũng lưu giữ tham chiếu đến các *observer* đã đăng ký chủ đề đó.

```
package com.javacodegeeks.patterns.observerpattern;

public class SMSUsers implements Observer{

        private final Subject subject;
        private String desc;
        private String userInfo;

        public SMSUsers(Subject subject,String userInfo){
                if(subject==null){
                        throw new IllegalArgumentException("No Publisher found.");
                }
                this.subject = subject;
                this.userInfo = userInfo;
        }

        @Override
        public void update(String desc) {
                this.desc = desc;
                display();
        }

        private void display(){
                System.out.println("["+userInfo+"]: "+desc);
        }

        @Override
        public void subscribe() {
                System.out.println("Subscribing "+userInfo+" to "+subject.subjectDetails()+ ↩
                    " ...");
                this.subject.subscribeObserver(this);
                System.out.println("Subscribed successfully.");
        }

        @Override
        public void unSubscribe() {
                System.out.println("Unsubscribing "+userInfo+" to "+subject.subjectDetails ↩
                    ()+" ...");
                this.subject.unSubscribeObserver(this);
                System.out.println("Unsubscribed successfully.");
        }

}
```

Lớp trên là *observer* cụ thể mà cài đặt giao diện *Observer*. Nó cũng lưu giữ tham chiếu đến chủ đề mà nó đăng ký vào và biến *userInfo* tùy chọn mà được sử dụng để hiện thông tin người sử dụng.

```
package com.javacodegeeks.patterns.observerpattern;

import java.util.ArrayList;

public class TestObserver {

        public static void main(String[] args) {
                Subject subject = new CommentaryObject(new ArrayList<Observer>(), "Soccer  ↩
                    Match [2014AUG24]");
                Observer observer = new SMSUsers(subject, "Adam Warner [New York]");
                observer.subscribe();
```

```
                System.out.println();

                Observer observer2 = new SMSUsers(subject, "Tim Ronney [London]");
                observer2.subscribe();

                Commentary cObject = ((Commentary)subject);
                cObject.setDesc("Welcome to live Soccer match");
                cObject.setDesc("Current score 0-0");

                System.out.println();

                observer2.unSubscribe();

                System.out.println();

                cObject.setDesc("It's a goal!!");
                cObject.setDesc("Current score 1-0");

                System.out.println();

                Observer observer3 = new SMSUsers(subject, "Marrie [Paris]");
                observer3.subscribe();

                System.out.println();

                cObject.setDesc("It's another goal!!");
                cObject.setDesc("Half-time score 2-0");

        }
}
```

Ví dụ trên sẽ in ra:

```
Subscribing Adam Warner [New York] to Soccer Match [2014AUG24] ...
Subscribed successfully.

Subscribing Tim Ronney [London] to Soccer Match [2014AUG24] ...
Subscribed successfully.

[Adam Warner [New York]]: Welcome to live Soccer match
[Tim Ronney [London]]: Welcome to live Soccer match

[Adam Warner [New York]]: Current score 0-0
[Tim Ronney [London]]: Current score 0-0

Unsubscribing Tim Ronney [London] to Soccer Match [2014AUG24] ...
Unsubscribed successfully.

[Adam Warner [New York]]: It's a goal!!

[Adam Warner [New York]]: Current score 1-0

Subscribing Marrie [Paris] to Soccer Match [2014AUG24] ...
Subscribed successfully.

[Adam Warner [New York]]: It's another goal!!
[Marrie [Paris]]: It's another goal!!

[Adam Warner [New York]]: Half-time score 2-0
[Marrie [Paris]]: Half-time score 2-0
```

Như bạn thấy, đầu tiên hai người sử dụng đăng ký họ cho trận bóng và bắt đầu nhận được bình luận. Nhưng sau đó một người hủy đăng ký nó, nên người này không nhận được bình luận nữa. Sau đó người sử dụng khác đăng ký và bắt đầu nhận được bình luận.

Điều này xảy ra động không thay đổi code đã có và không chỉ vậy, giả sử công ty muốn truyền thông bình luận trên thư điện tử hoặc hãng khác muốn hợp tác với công ty này để truyền thông bình luận. Mọi việc bạn cần làm là tạo ra hai lớp mới UserEmail và ColCompany và bạn tạo họ là observer của chủ đề bằng cách cài đặt giao diện Obsever. Cứ như vậy Subject đã biết observer của nó, nên sẽ cung cấp cập nhật.

6.7.4 Mẫu Observer có sẵn trong Java

Java có hỗ trợ xây sẵn cho mẫu Observer. Tổng quát nhất là mẫu giao diện Observer và lớp quan sát được Observable class trong gói Java.util. Nó hoàn toàn tương tự như giao diện Subject và Observer của chúng ta, nhưng cho bạn nhiều chức năng vượt qua giới hạn.

Giả sử thử cài đặt bài toán trên sử dụng mẫu Observer có sẵn trong Java:

```
package com.javacodegeeks.patterns.observerpattern;

import java.util.Observable;

public class CommentaryObjectObservable extends Observable implements Commentary {
        private String desc;
        private final String subjectDetails;

        public CommentaryObjectObservable(String subjectDetails){
                this.subjectDetails = subjectDetails;
        }

        @Override
        public void setDesc(String desc) {
                this.desc = desc;
                setChanged();
                notifyObservers(desc);
        }

        public String subjectDetails() {
                return subjectDetails;
        }
}
```

Trên đây chúng ta mở rộng lớp *Observable* để làm cho lớp của chúng ta như chủ đề và lưu ý rằng lớp trên đây không giữ tham chiếu đến *observer*, nó được điều khiển bởi lớp cha, chính là lớp *Observable*. Tuy nhiên, chúng ta khai báo phương thức *setDesc* để thay đổi trạng thái của đối tượng như đã làm trong Ví dụ trước. Phương thức *setChanged* là phương thức từ lớp cha mà được sử dụng để đặt cờ thay đổi là true. Phương thức notifyObservers thông báo cho mọi obsever và sau đó gọi phương thức *clearChanged* để chỉ ra rằng đối tượng này sẽ không thay đổi lâu hơn nữa. Mỗi *observer* có phương thức *update* của nó với hai đối số: đối tượng kiểu *observable* và đối tượng *agr*.

```
package com.javacodegeeks.patterns.observerpattern;

import java.util.Observable;

public class SMSUsersObserver implements java.util.Observer{

        private String desc;
        private final String userInfo;
        private final Observable observable;

        public SMSUsersObserver(Observable observable,String userInfo){
                this.observable = observable;

                this.userInfo = userInfo;
        }

        public void subscribe() {
                System.out.println("Subscribing "+userInfo+" to "+(( ↩
                    CommentaryObjectObservable)(observable)).subjectDetails()+" ...");
                this.observable.addObserver(this);
                System.out.println("Subscribed successfully.");
        }

        public void unSubscribe() {
                System.out.println("Unsubscribing "+userInfo+" to "+(( ↩
                    CommentaryObjectObservable)(observable)).subjectDetails()+" ...");
                this.observable.deleteObserver(this);
                System.out.println("Unsubscribed successfully.");
        }

        @Override
        public void update(Observable o, Object arg) {
                        desc = (String)arg;
                        display();
        }

        private void display(){
                System.out.println("["+userInfo+"]: "+desc);
        }

}
```

Ta bàn luận về một số phương thức chính.

Lớp trên đây cài đặt giao diện *Observer* mà có một phương thức chính *update*, mà được gọi khi chủ đề gọi phương thức *notifyObservers*. Phương thức *update* dùng đối tượng kiểu *Observable* và đối tượng *arg* như các tham số.

Phương thức *addObservable* được dùng để đăng ký *observer* cho chủ đề và phương thức *deleteObserver* được dùng để xóa *observer* khỏi danh sách của chủ đề.

Thử kiểm tra ví dụ này.

```
package com.javacodegeeks.patterns.observerpattern;

public class Test {

        public static void main(String[] args) {
                CommentaryObjectObservable obj = new CommentaryObjectObservable("Soccer ↩
                    Match [2014AUG24]");
                SMSUsersObserver observer = new SMSUsersObserver(obj, "Adam Warner [New ↩
                    York]");
                SMSUsersObserver observer2 = new SMSUsersObserver(obj,"Tim Ronney [London]" ↩
                    );
                observer.subscribe();
                observer2.subscribe();
                obj.setDesc("Welcome to live Soccer match");
                obj.setDesc("Current score 0-0");

                observer.unSubscribe();

                obj.setDesc("It's a goal!!");
                obj.setDesc("Current score 1-0");
        }
}
```

Nó in ra kết quả sau:

```
Subscribing Adam Warner [New York] to Soccer Match [2014AUG24] ...
Subscribed successfully.
Subscribing Tim Ronney [London] to Soccer Match [2014AUG24] ...
Subscribed successfully.

[Tim Ronney [London]]: Welcome to live Soccer match
[Adam Warner [New York]]: Welcome to live Soccer match
[Tim Ronney [London]]: Current score 0-0
[Adam Warner [New York]]: Current score 0-0
Unsubscribing Adam Warner [New York] to Soccer Match [2014AUG24] ...
Unsubscribed successfully.
[Tim Ronney [London]]: It's a goal!!
[Tim Ronney [London]]: Current score 1-0
```

Lớp trên tạo chủ đề và hai *observer*. Phương thức *subscribe* của *observer* bổ sung nó vào danh sách obsever của chủ đề. Sau đó *setDesc* thay đổi trạng thái của chủ đề mà gọi phương thức *setChanged* để đặt cờ thay đổi là *true* và thông báo cho các *observer*. Kết quả thu được, phương thức *update* của *observer* được gọi mà phương thức trong lớp d*isplay* sẽ hiển thị kết quả. Sau này, một *observer* rút khỏi danh sách *observer*. Theo đó các bình luận sau này không được cập nhật cho người đó.

Java cung cấp tiện ích xây sẵn cho mẫu *Observer*, nhưng nó có nhược điểm của nó. *Observable* là một lớp mà bạn kế thừa nó. Điều này nghĩa là bạn không thể bổ sung thêm hành vi *Observable* vào lớp có sẵn vì nó đã mở rộng một lớp cha khác. Điều này hạn chế khả năng tái sử dụng. Bạn ngay cả không thể tạo cài đặt của riêng bạn mà vẫn làm việc tốt với *API Obsever* xây sẵn trong Java

Một số phương thức trong *API Observable* là *Protected.* Điều đó có nghĩa là bạn không thể gọi các phương thức như *setChange* trừ khi bạn có lớp con *Observable.* Và bạn không thể tạo khởi tạo của lớp *Observable* và kết hợp nó với các đối tượng của riêng bạn, mà bạn có lớp con. Thiết kế này vi phạm nguyên tắc thiết kế "kết hợp ưa thích trên kế thừa".

6.7.5 Khi nào sử dụng mẫu Observer

Sử dụng mẫu Observer trong một các tình huống sau:

- Khi trừu tượng có hai khía cạnh, một phụ thuộc vào cái kia. Đóng gói các khía cạnh này trong các đối tượng tách bạch nhau cho phép bạn đa dạng và tái sử dụng chúng độc lập.
- Khi một thay đổi của đối tượng này yêu cầu thay đổi các đối tượng khác và bạn không biết có bao nhiêu đối tượng cần phải được thay đổi.
- Khi một đối tượng có thể thông báo cho các đối tượng khác mà không cần giả thiết về ai là các đối tượng này. Nói cách khác, bạn không muốn các đối tượng này quá gắn chặt nhau.