LE PATTERN OBSERVATEUR

Ludovic Liétard

1

Le pattern OBSERVATEUR

- L 'objectif du pattern observateur :
- « Définir une interdépendance de type un à plusieurs, de façon telle que, quand un objet change d 'état, tous ceux qui en dépendent en soient notifiés et automatiquement mis à jour ».

2

Le pattern OBSERVATEUR

- Role : Comportemental
- Problème : Un objet doit pouvoir faire une notification à d 'autres objets, dont le nombre est inconnu a priori, sans faire d 'hypothèse sur leur nature

3

Le pattern OBSERVATEUR

- Solution :
  - Les objets à avertir sont qualifiés d '**observateurs** (puisqu 'ils attendent d 'être notifié).
  - Un objet à l 'origine de l 'événement est un **sujet**. C 'est lui qui avertit les observateurs.

4

Le pattern OBSERVATEUR

- Solution :
  - Le sujet connaît ses observateurs.
  - Un nombre quelconque d 'observateurs peut observer un objet.

5

Le pattern OBSERVATEUR

- Solution :
  - Un sujet est donc muni de deux méthodes :
    - `void attacher(Observateur o)` : pour ajouter un observateur à sa liste d 'observateurs
    - `void detacher(Observateur o)` : pour supprimer un observateur de sa liste d 'observateurs

6

Le pattern OBSERVATEUR

- Solution :
  - Lorsque l'événement survient, le sujet avertit ses observateurs ⇒ les observateurs sont munis de la méthode `averti(Sujet s)` pour être averti par le Sujet.

7

Le pattern OBSERVATEUR

- Solution :
  - `averti(Sujet s)` : pour signaler qu'un événement vient de survenir. L'observateur peut instaurer un dialogue avec le sujet pour avoir des précisions supplémentaires si besoin est. Le paramètre Sujet permet à l'observateur de dépendre de plusieurs sujets.

8

Le pattern OBSERVATEUR

Implémentation (Java) :

- `Observateur` et `Sujet` peuvent être des interfaces
- mais également des superclasses abstraites
- `Observateur` est une interface et `Sujet` est une superclasse abstraite

9

- Implémentation :

```
public interface Observateur {
public void averti(Sujet c);
}
```

10

- Implémentation :

```
abstract class Sujet {

        ArrayList<Observateur> mesObs;
        public Sujet(){
              mesObs = new ArrayList<Observateur>();
        }
        public void attache(Observateur o){
                        obs.add(o);
        }
        public void detache(Observateur o){
                        obs.remove(obs.indexOf(o));
        }
```

11

- Implémentation :

```
public void alerte(){
            Iterator it;
            Observateur o;

                  it=mesObs.iterator();
                  while (it.hasNext()){
                        o=(Observateur)(it.next());
                        o.averti(this);
                  }
        }
    }/ / Fin Sujet
```

12

Le pattern OBSERVATEUR

- Exemple :
  - Un système de Clients et de Représentants.
  - Un Client fait des achats… et est suivi par des Représentants.
  - Si l'un de ses achats dépasse un montant de 2000 euros, ses Représentants sont alertés et lui envoient une lettre (de promotion).

13

Le pattern OBSERVATEUR

- Exemple :
  - Un Représentant est averti lorsqu'un événement arrive ⇒ `Observateur`
  - Un Client est à l'origine de l'événement ⇒ `Sujet`

14

- Implémentation :

```
public class Commercial implements Observateur{
String nom;
public Commercial(String n){
        nom=n;
}

public void averti(Sujet c){
        System.out.println("Commercial averti : "+nom);
        ((Client) c).lettre(nom);
}
}
```

15

Implémentation :

```
public class Client extends Sujet {
    String nom;
        int somme;

        public Client(String n){
          super();
          somme = 0; nom = n;
          }
```

16

Implémentation : // Class Client suite

```
    public void achat(int montant){
            somme = somme + montant;
            if (montant > 2000) {
              this.alerte();
             }
       }
        public void lettre(String n){
          System.out.println("reçu une lettre de : "+n);
        }
}
```

17

- Implémentation :

```
public static void main(String[] args) {
        Client c = new Client("Client 1");
        c.attache(new Commercial("Commercial 1"));
        c.attache(new Commercial("Commercial 2"));
        c.achat(90);
        c.achat(19000);
}
```

18