Université Claude Bernard Lyon 1
Polytech Lyon - Département Informatique

4A , 2019–2020

ISI-3

TD3. Design Patterns(correction)

Laëtitia Matignon

---

1 Design Pattern Observateur/Observé

Soit le code suivant utilisant les classes représentées dans le diagramme de la figure 1 :

```
SujetConcret s = new SujetConcret("PopCorn", 1.29f);
NameObserver nameObs = new NameObserver();
PriceObserver priceObs = new PriceObserver();
s.addObserver(nameObs);
s.addObserver(priceObs);
s.setName("Frosties");
s.setPrice(4.57f);
s.setPrice(9.22f);
s.setName("Smacks");
```

On souhaite obtenir l'affichage suivant lors de l'exécution du code précédent :

```
Name changed to Frosties
Price changed to 4.57
Price changed to 9.22
Name changed to Smacks
```

Question 1 *Proposer une implémentation de sorte à obtenir l'affichage souhaité pour :*
- *les méthodes* `setName` *et* `setPrice` *de la classe* `SujetConcret`
- *les méthodes* `update` *pour chaque* `Observer`,

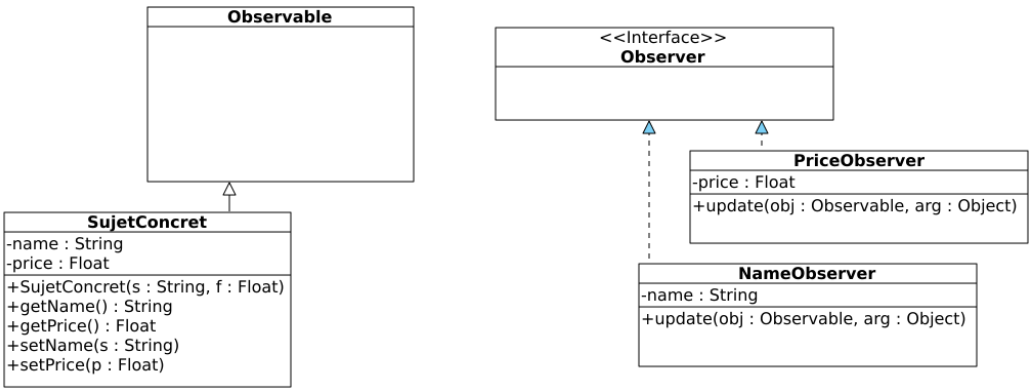

FIGURE 1 – Design Pattern Observateur Observé

Lorsque le sujet change de nom ou de prix (`setName` ou `setPrice`), il faut notifier l'observateur correspondant (méthode `notifyObservers` d'`Observable` qui appelle `update` sur chaque `Observer` (cf. slide 98 du CM2)). Les observateurs lorsqu'il sont notifiés vont afficher la phrase correspondante.
La difficulté ici est que vous avez 2 observateurs. A partir du moment où vous appelez `notifyObservers` dans `setName` et `setPrice`, **les 2 observateurs sont notifiés** et exécutent leur méthode `update`. Il faut donc trouver un moyen pour que seul `NameObserver` affiche quelque chose lorsque `setName` notifie, et seul `PriceObserver` affiche quelque chose lorsque `setPrice` notifie.

V1 : L'observé (`SujetConcret`) donne des informations (via la méthode `setName` et `setPrice`) qui sont de types différents (Float et String). On peut donc **pousser** des arguments via la méthode `notifyObservers` et son paramètre `Object arg`. Les arguments **poussés** sont récupérés dans la méthode `update` de l'observateur. Chaque observateur vérifie le type de l'argument poussé pour savoir s'il doit afficher quelque chose :

```
public void setName(String name) {
this.name = name;
setChanged();//NE PAS OUBLIER: TOUJOURS APPELER setChanged AVANT notify (cf.
    slide 98 du CM2): notify appele update si booleen changed mis a true ...
notifyObservers(name);// pousse argument recupere dans parametre arg de
    update
}
public void setPrice(float price) {
this.price = price;
setChanged();
notifyObservers(new Float(price));//pousse argument recupere dans parametre
    arg de update
}
public class NameObserver implements Observer {
 public void update(Observable obj, Object arg) {
  if (arg instanceof String) {//cet observateur verifie le type de l'argument
      pousse: si c'est un String, il affiche, sinon, il n efait rien
   name = (String)arg;
   System.out.println("Name changed to " + name);}}
public class PriceObserver implements Observer {
 public void update(Observable arg0, Object arg) {
   if (arg instanceof Float) {
    Float prix = (Float)arg; //cet observateur verifie le type de l'argument
        pousse: si c'est un Float, il affiche, sinon, il n efait rien
    System.out.println("Price change to "+ prix);}}}
```

V2 : la solution v1 fonctionne car les types des éléments poussés sont différents. Une autre solution consiste à ce que l'observateur tire des informations de l'observé, et vérifie si cette information a changé. Si oui, il devra afficher le nouveau prix ou nom. il faut donc que les observateurs :

- mémorisent des informations (ajout d'un attribut)
- récupérent l'instance qui les a notifié : cela se fait avec le paramètre `Observable o` dans la méthode `update`.

```
public static void main(String[] args) {
      SujetConcret s = new SujetConcret("PopCorn", 1.29f);
      //modification pour utiliser le constructeur avec parametre
      NameObserver nameObs = new NameObserver(s.getName());
      PriceObserver priceObs = new PriceObserver(s.getPrice());
      s.addObserver(nameObs);
      s.addObserver(priceObs);
      s.setName("Frosties");
      s.setPrice(4.57f);
      s.setPrice(9.22f);
      s.setName("Smacks");
   }

public void setName(String name) {
this.name = name;
setChanged();
notifyObservers();//on ne pousse plus d'information ici contrairement a la V1
}
public void setPrice(float price) {
this.price = price;
setChanged();
notifyObservers();
}


public class NameObserver implements Observer {
   String name ;
   public NameObserver(String name) {
      super();
      this.name = name;
   }
   @Override
   public void update(Observable o, Object arg) {
      if (o instanceof SujetConcret){//recupere dans o celui qui l'a notifie,
           verifie si c'est SujetConcret
         String n = ((SujetConcret)(o)).getName(); //l'observateur tire des
             informations sur l'observable
         if (!n.equals(name) ){
            name = n;
            System.out.println("Name changed to "+n);
         }
      }
   }
}
```

```
public class PriceObserver implements Observer {
   Float price;
      public PriceObserver(Float price) {
      super();
      this.price = price;
   }
      @Override
      public void update(Observable o, Object arg) {
         if (o instanceof SujetConcret){//recupere dans o celui qui l'a
             notifie, verifie si c'est SujetConcret
            Float p = ((SujetConcret)(o)).getPrice(); //l'observateur tire des
                informations sur l'observable
            if (p != price ){
               price = p;
               System.out.println("Price changed to "+p);
            }
         }
      }
}
```