Observable/Observer – Programmation événementielle
Introduction à SWING
MVC (Modèle Vue Controleur)

Frédéric Armetta
Frederic.armetta@univ-lyon1.fr

Motif de modélisation :

Observer/Observable

Expression d'un besoin

- Soit un objet (*observer*) souhaite être informé de l'évolution de l'état d'un autre objet (*observable*)
  - *Pull* : l'observer scrute régulièrement l'objet observé
    - coûteux, introduction de latences

Expression d'un besoin

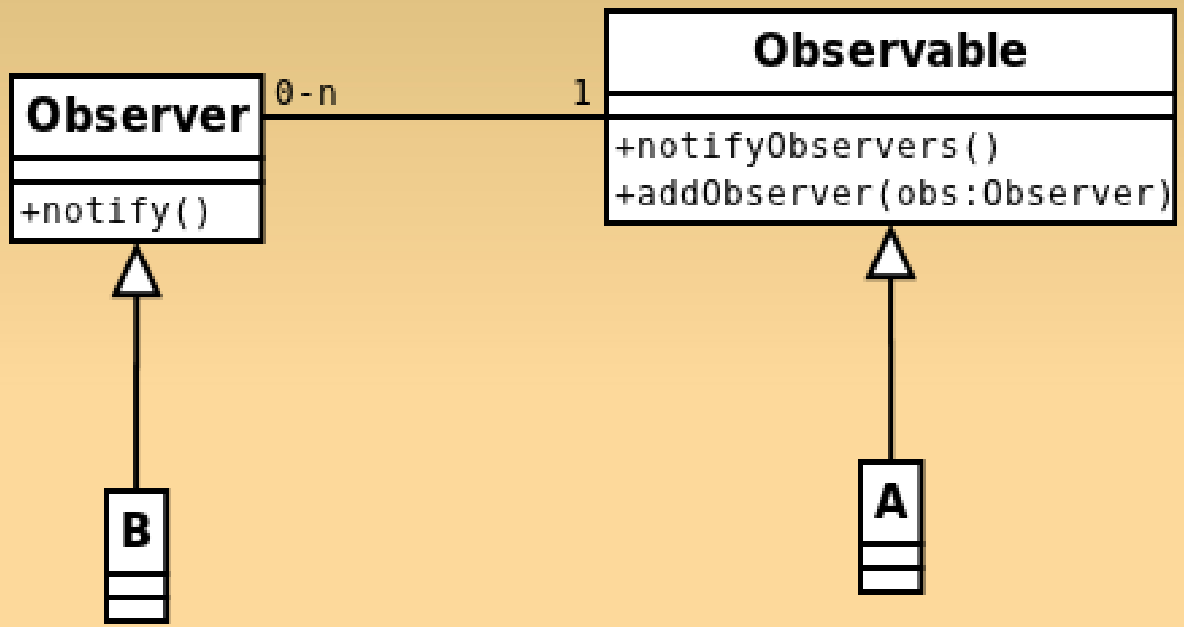
- Soit un objet (*observer*) souhaite être informé de l'évolution de l'état d'un autre objet (*observable*)
  - *Pull* : l'observer scrute régulièrement l'objet observé
    - coûteux, introduction de latences
  - *Push* : l'observable notifie l'observer
    - lorsque cela est opportun

Expression d'un besoin

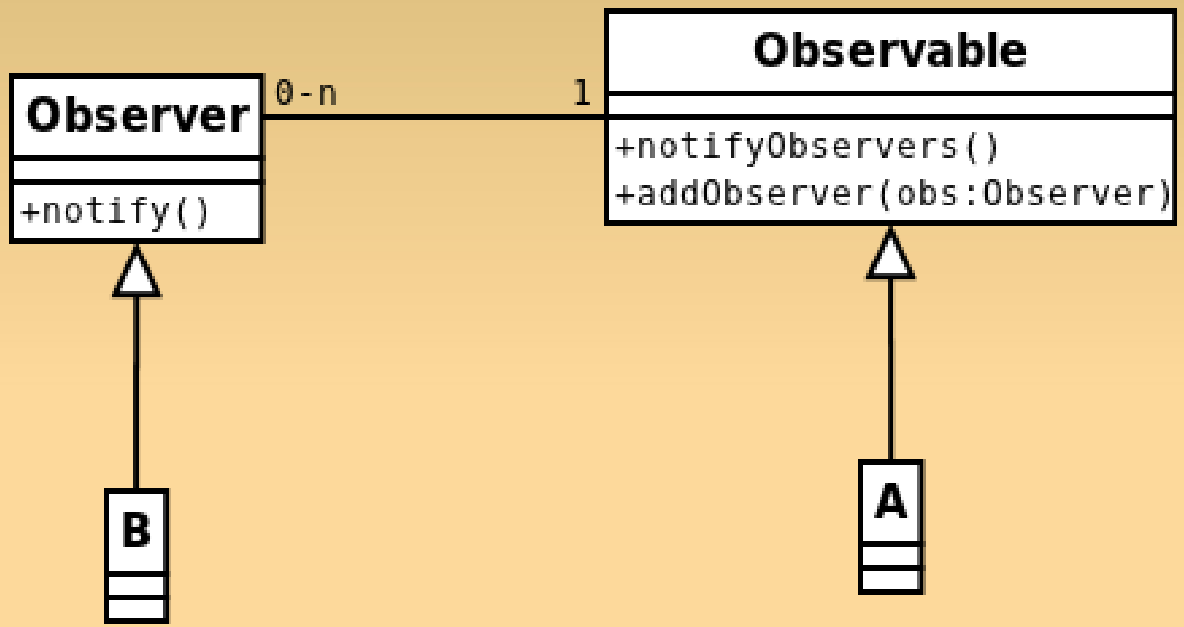
- Soit un objet (*observer*) souhaite être informé de l'évolution l'état d'un autre objet (*observable*)
  - *Pull* : l'observer scrute régulièrement l'objet observé
    - coûteux, introduction de latences
  - *Push* : l'observable notifie l'observer
    - lorsque cela est opportun

Utilité d'un *design pattern* ??

Design Pattern Observable

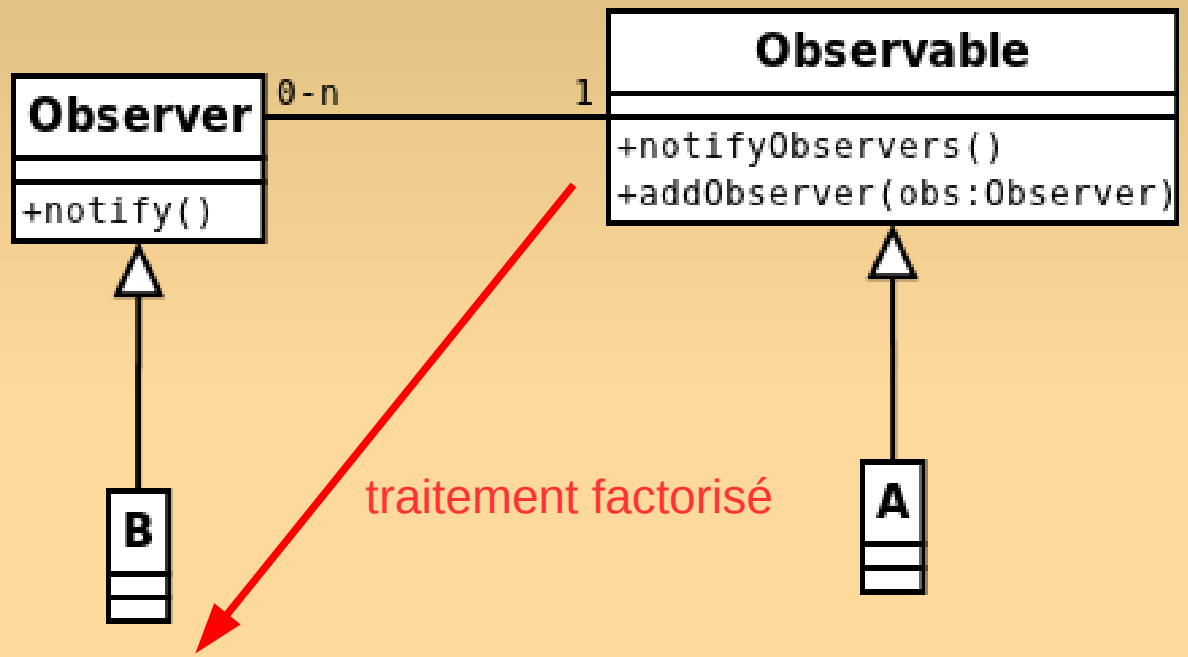


Design Pattern Observable



- A a = new Obervable();
  B b = new Observer();
  a.addObserver(b);
  a.notifyOberver(); (=> b.notify())

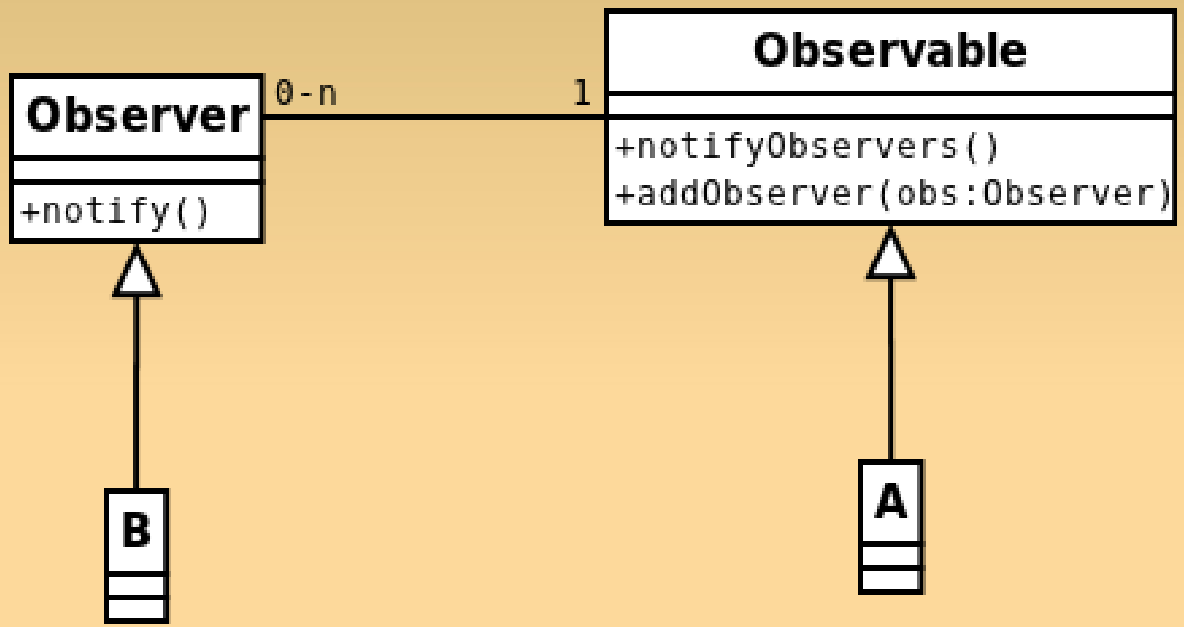
Design Pattern Observer



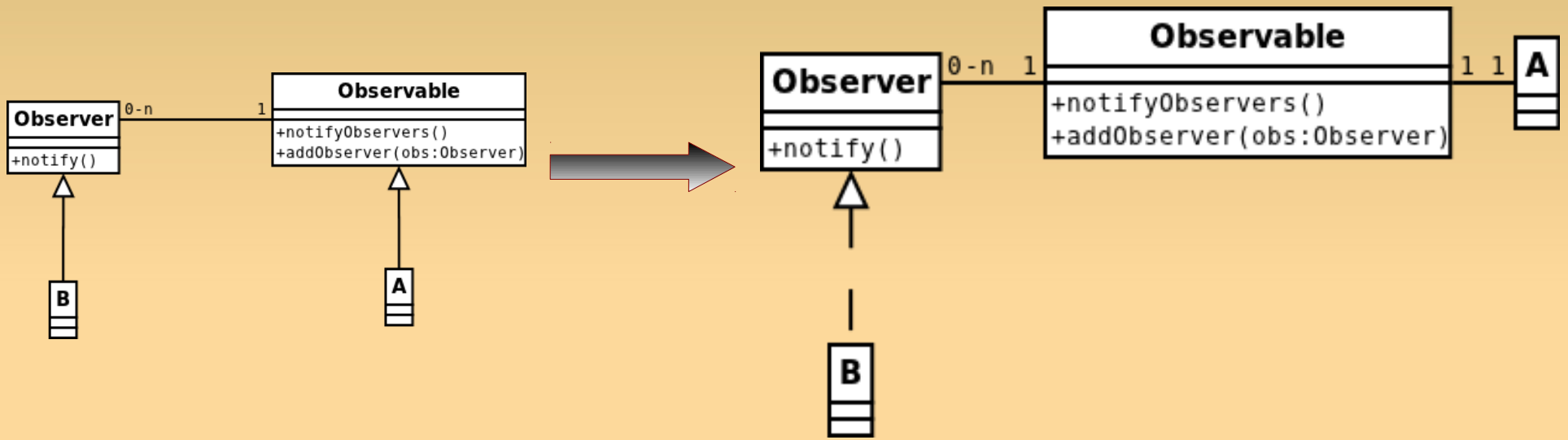
- + notifyObservers() {
  for each o : Observer in lstObserver
  o.notify();
  }

Liens d'héritage occupés

- Si héritage occupé pour les classes A et B ??



Liens d'héritage occupés



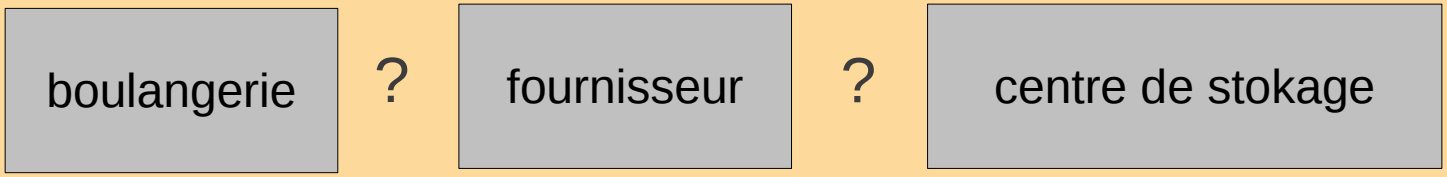
utilisation d'une **interface**
&&
utilisation d'un **objet utilitaire**

Exercice

- Soit la configuration suivante :

  Une boulangerie est approvisionnée par lots de 100 baguettes par le fournisseur.

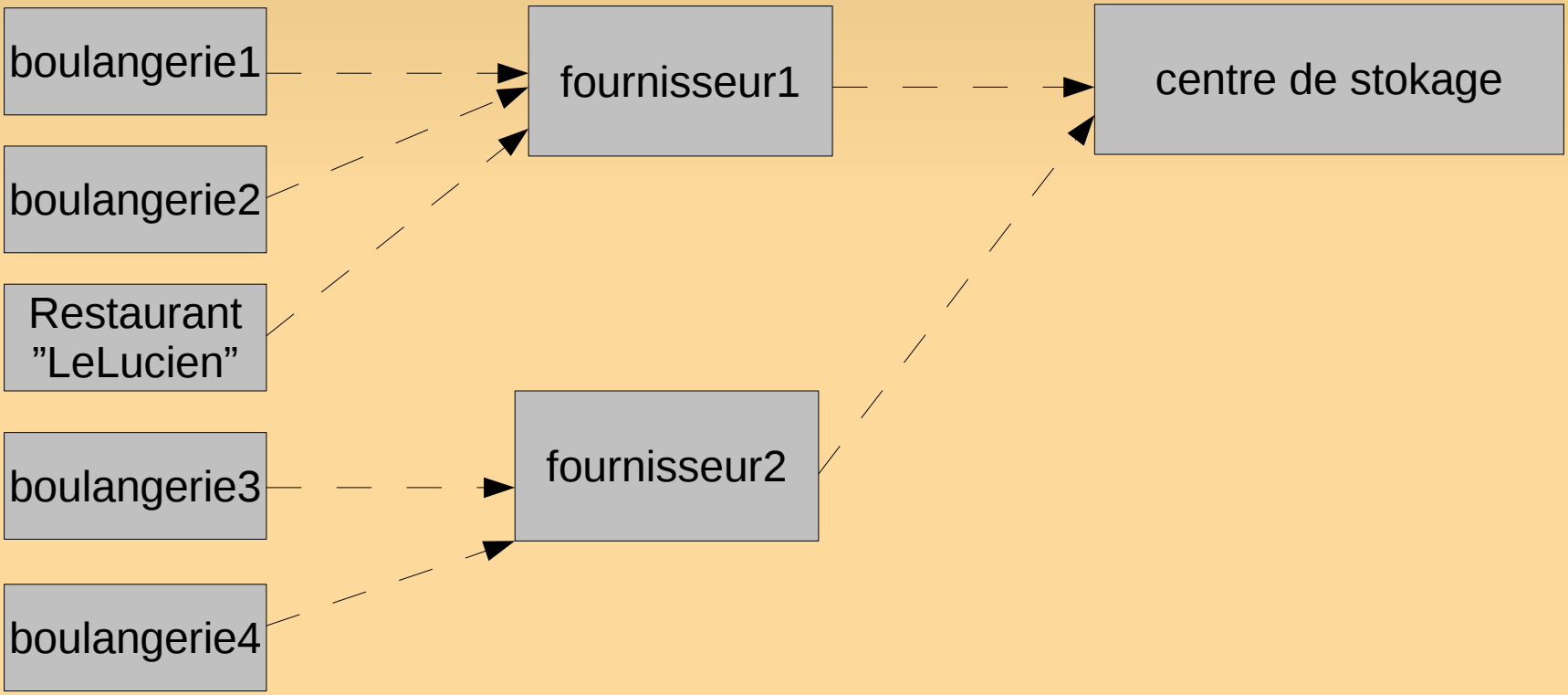
  Le fournisseur est approvisionné par lots de 10000 par le centre de stockage.



  Donner le diagramme de classes, le diagramme d'objets et les principaux traitements

Pour aller plus loin ...

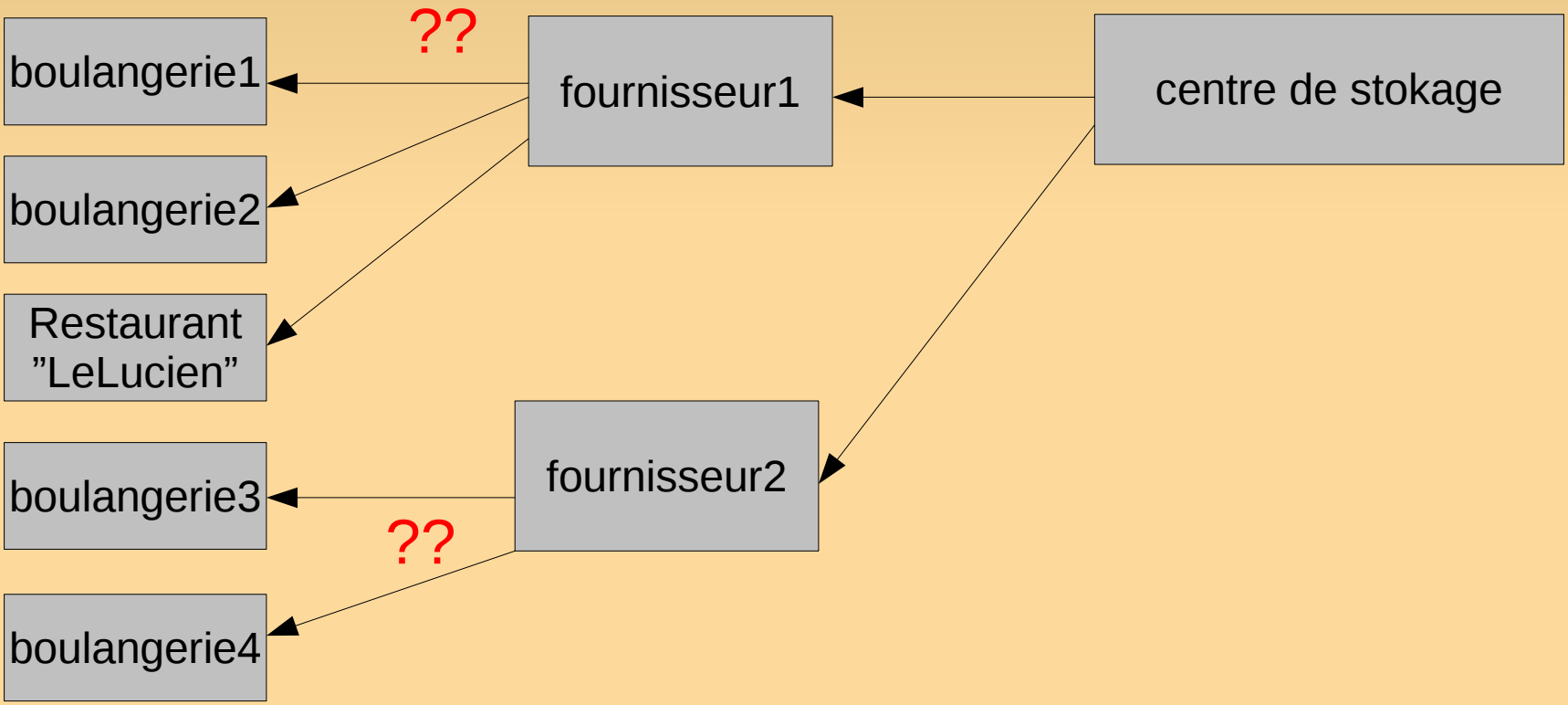
- Soit la configuration suivante :



Le design pattern initial est-il suffisant pour représenter cette configurattion ?

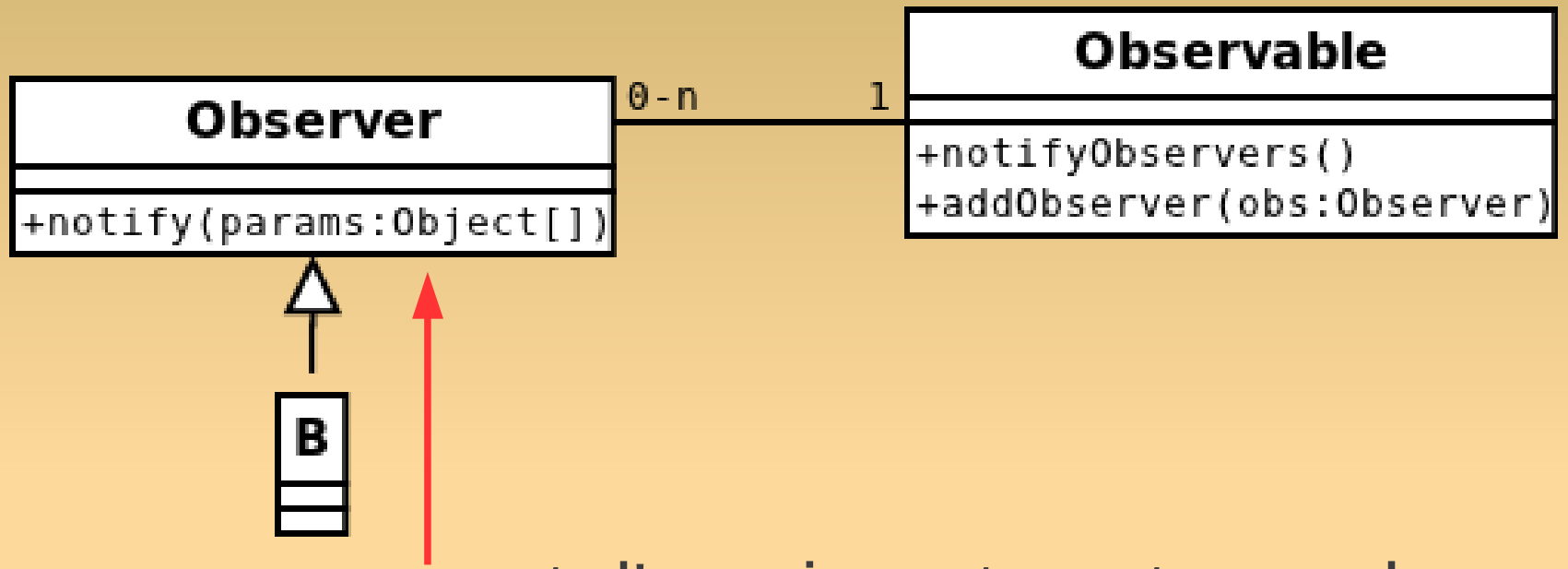
Pour aller plus loin ...

- Soit la configuration suivante :



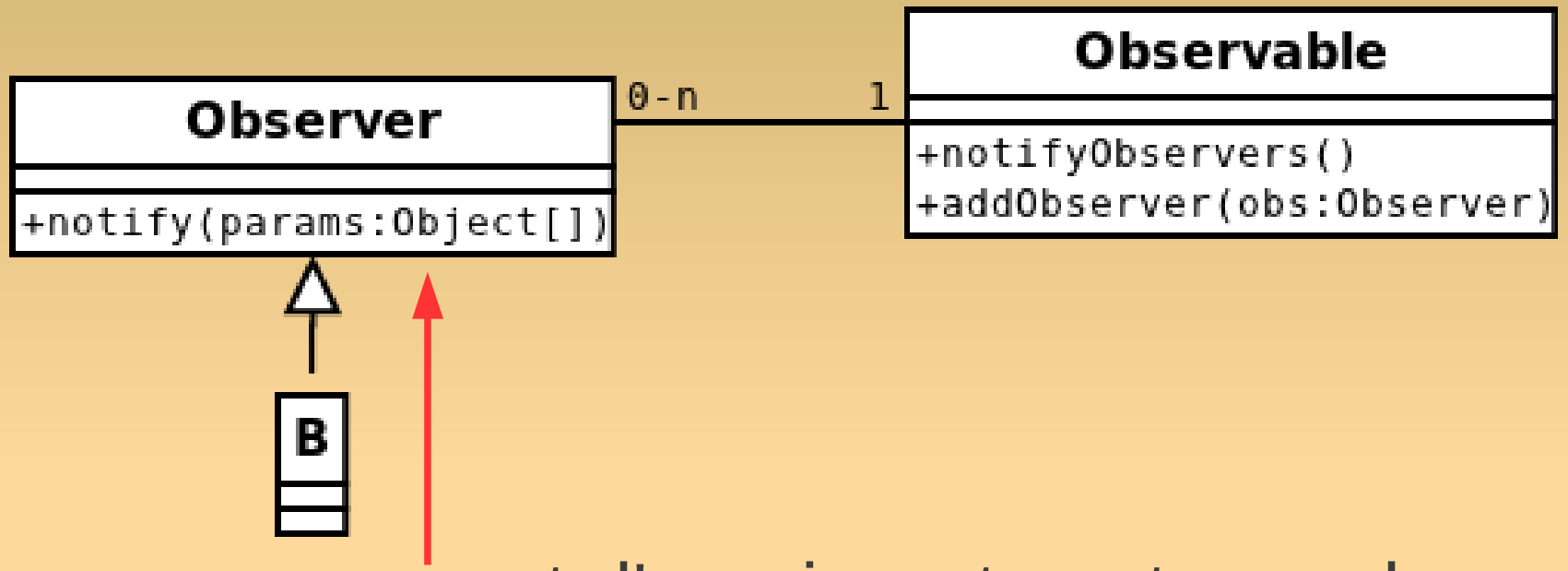
On ne connait pas l'émetteur du message …
=> prévoir un ”protocole” d'écoute plus riche

Design Pattern Observer++



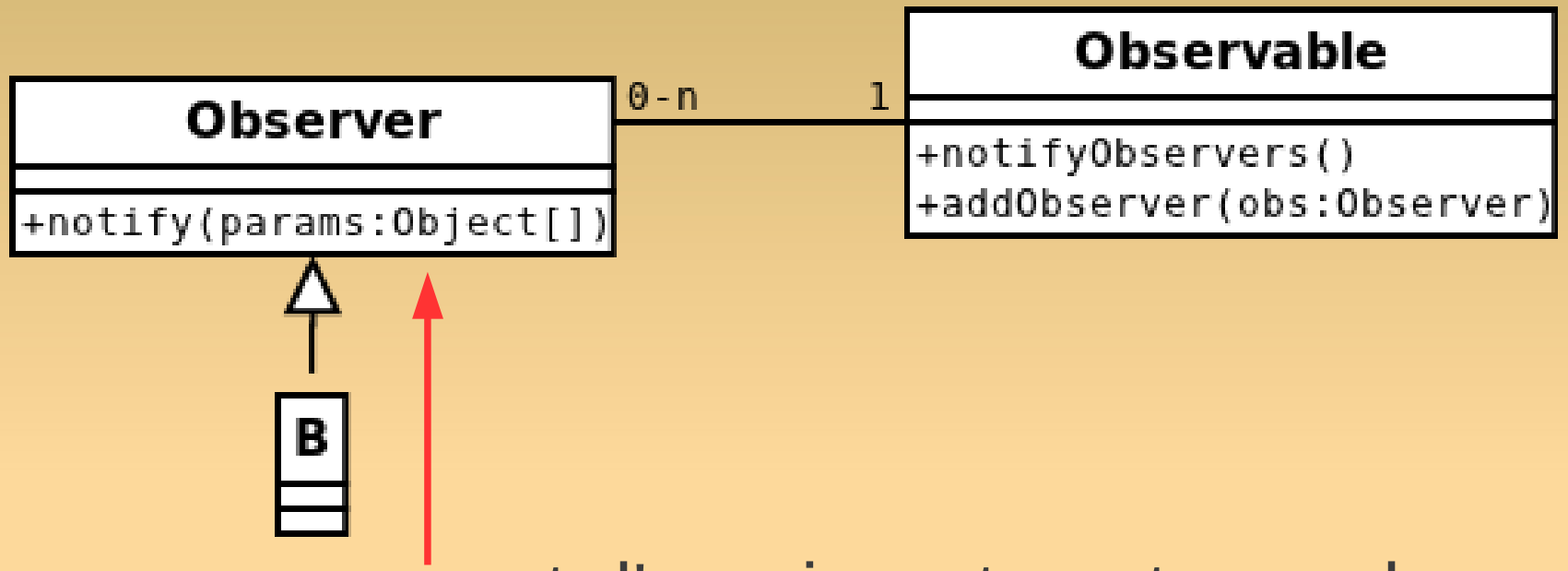
- params permet d'exprimer tous types de configurations

Design Pattern Observer++



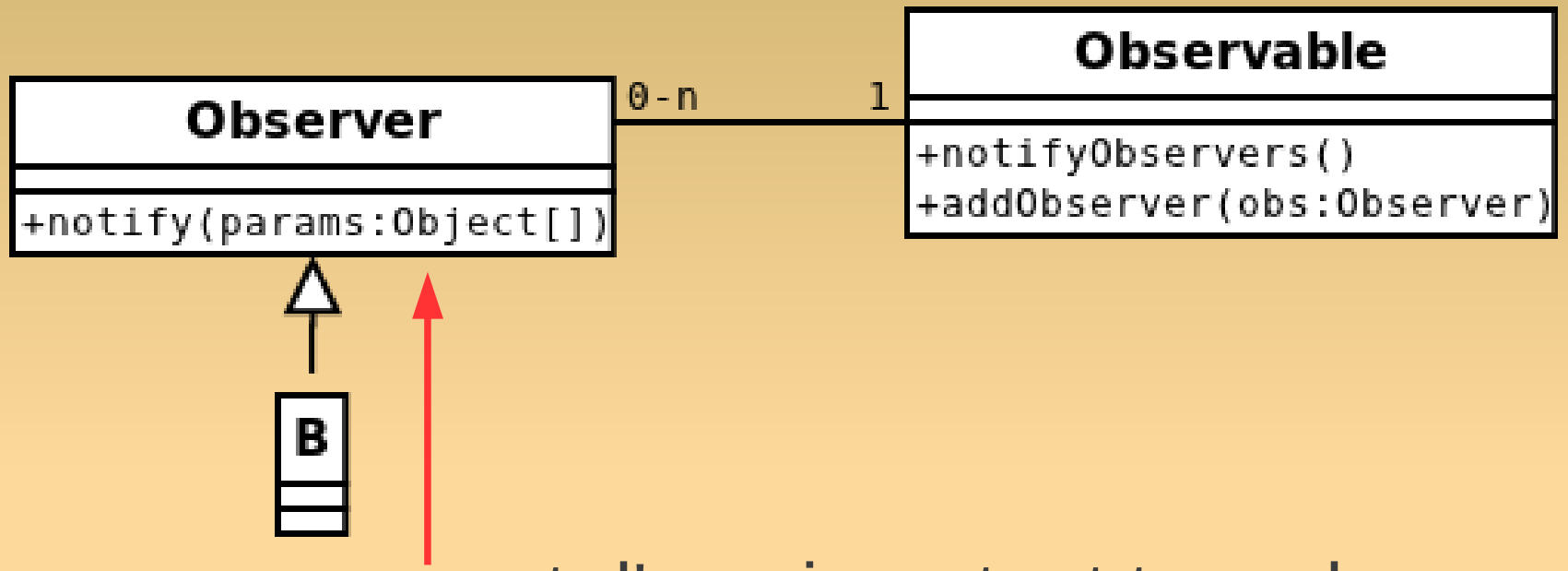
- params permet d'exprimer tous types de configurations mais ...

Design Pattern Observer++



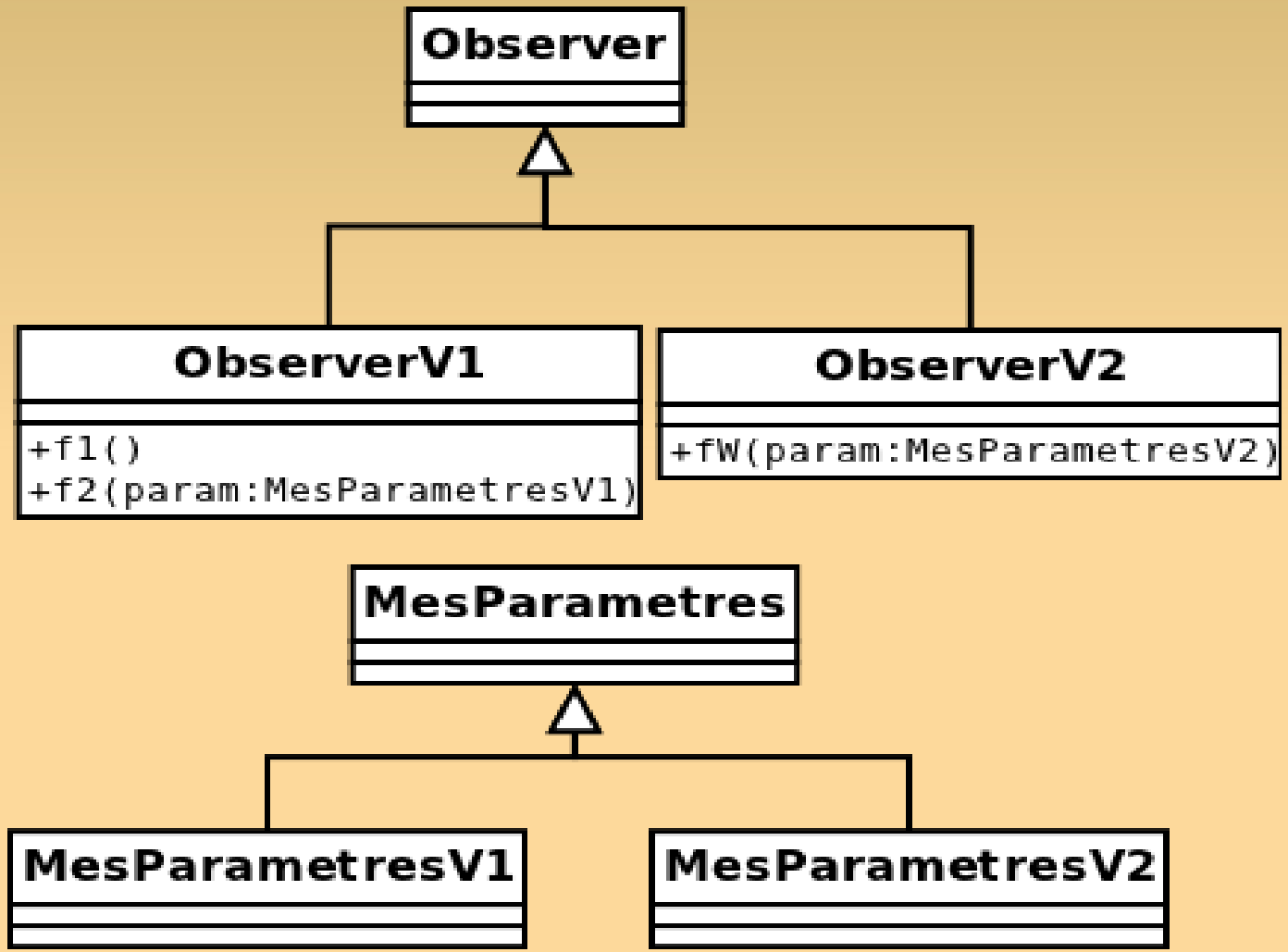
- params permet d'exprimer tous types de configurations mais …
- Possibilité d'utiliser une(/plusieurs) classe(s) dédié(es), mais aussi …

Design Pattern Observer++

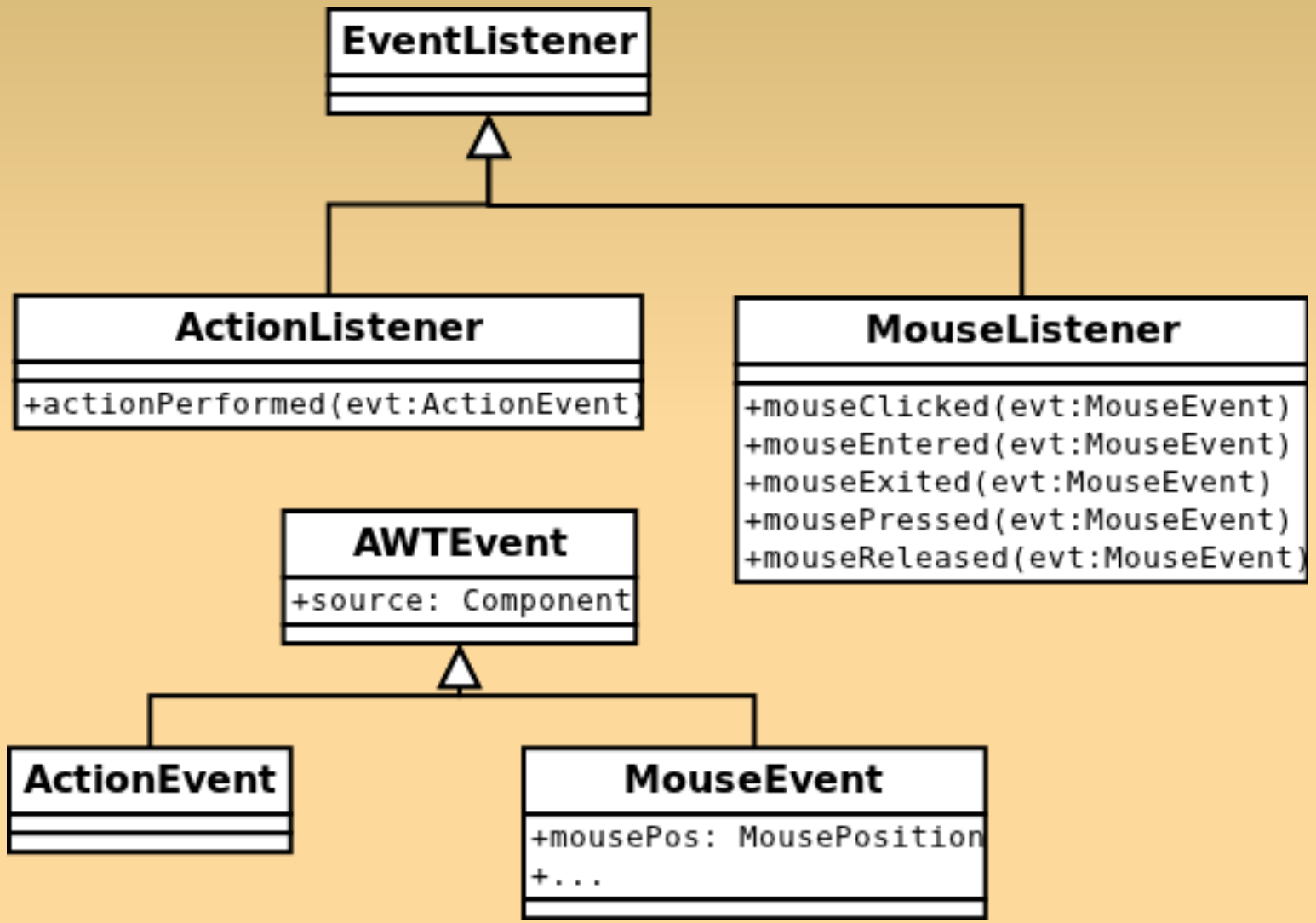


- params permet d'exprimer tout type de configuration mais …
- Possibilité d'utiliser une(/plusieurs) classe(s) dédié(es), mais aussi …
- Possibilité de spécialiser les observers

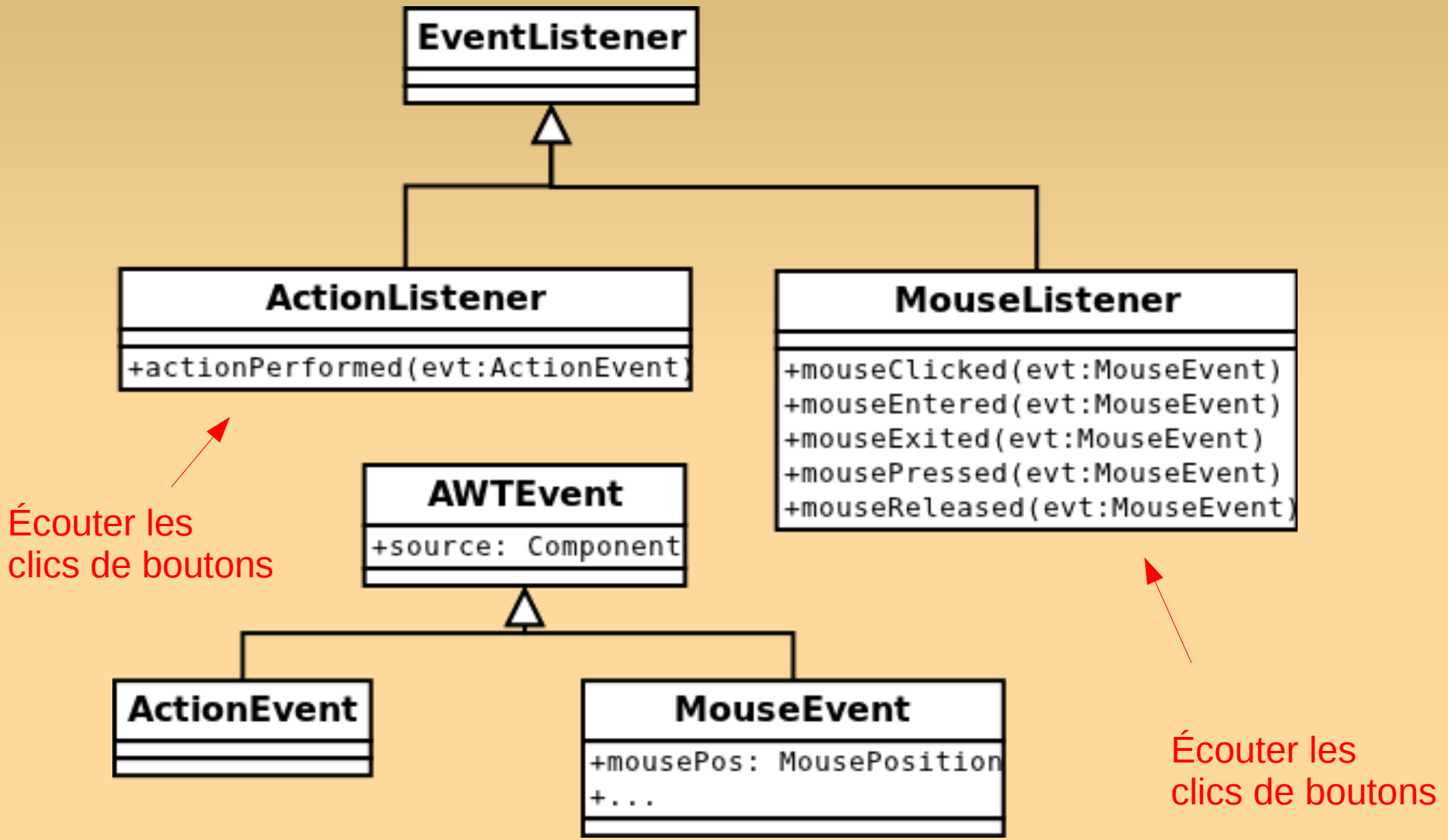
Design Pattern Observer++



Programmation événementielle Swing

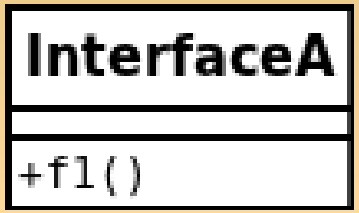


Programmation événementielle Swing



Exercice

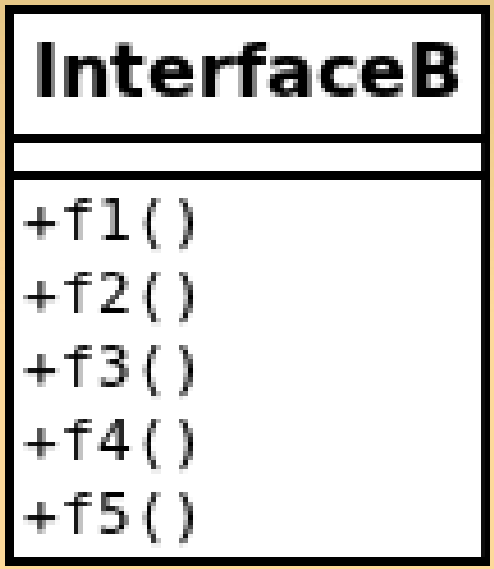
- Soit l'interface suivante :



En utilisant une classe anonyme, donner une définition simple de f1(afficher ”hello”)

Exercice

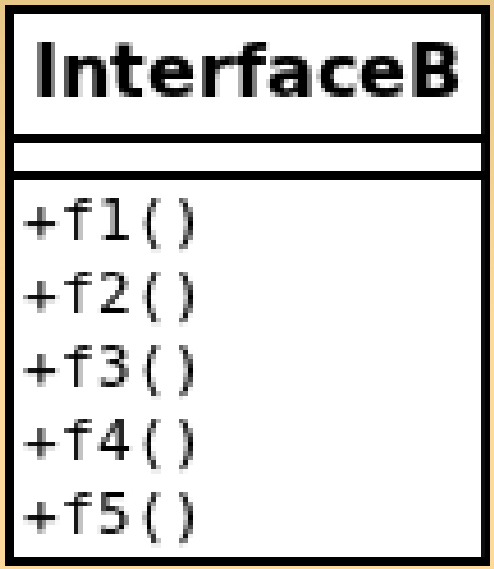
- Soit l'interface suivante :



En utilisant une classe anonyme, donner une définition simple de f1(afficher ”hello”)

Exercice

- Soit l'interface suivante :

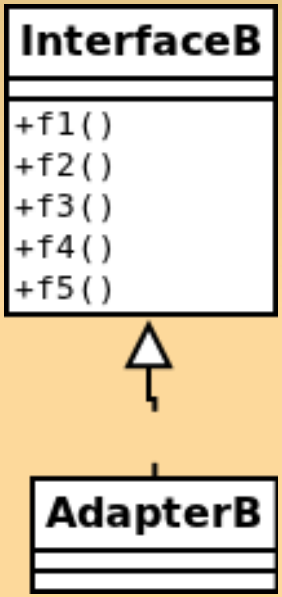



```
new InterfaceB() {
    public void f1() {
       afficher(”hello”);
    }

public void f2() {}
public void f3() {}

  ...

}
```

Comment pourrait-on simplifier ce type d'utilisation (on a souvent besoin de redéfinir en ligne une seule fonction) ?

Design pattern Adapter

- Soit l'interface suivante :




```
new AdapterB() {
    public void f1() {
        afficher(”hello”);
    }
}
```

AdapterB donne une définition vide de toutes les fonctionalités de InterfaceB