Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

ATL – Ateliers Logiciels

Design pattern Observateur/Observé

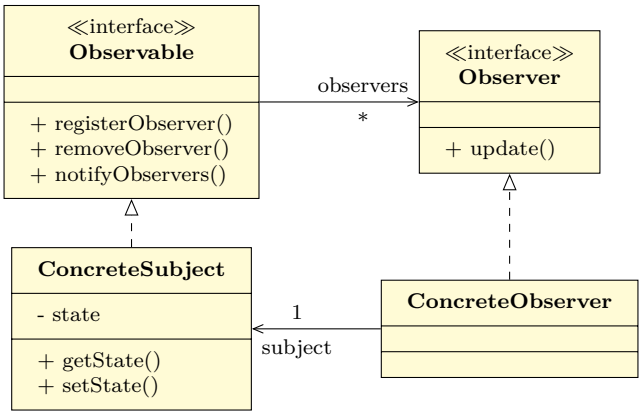
Introduction

Consignes

Ce document introduit le *design pattern* (patron de conception) *Observer*.
Vous pouvez obtenir les ressources de cette fiche via le lien `https://git.esi-bru.be/ATL/designpatterncards`.
N'oubliez pas de vous **connecter** au serveur `git.esi-bru.be` pour avoir accès au téléchargement.

1 Observer

Établit une relation un (*Observé*) à plusieurs (*Observateurs*) telle que lorsque l'état d'un objet change, tous les objets dépendants sont avertis qu'ils peuvent modifier leurs états. De cette manière, tous les objets *observateurs* peuvent être synchronisés avec l'objet qu'ils observent.



Pour qu'un *observateur* soit averti du changement d'état de l'*observé*, il faut que ce dernier

- maintienne une liste de ses *observateurs* et que la possibilité soit donnée de s'enregistrer/se retirer en tant qu'*observateur* ;
- qu'à chaque modification de son état, tous ses *observateurs* soient notifiés. C'est l'*observé* qui notifie les *observateurs*.

Lorsqu'un *observateur* est notifié, il doit pouvoir connaître le nouvel état de l'*observé* pour se mettre à jour en synchronisation avec celui-ci. Soit l'*observé* doit offrir la ou les méthodes qui permettent de l'interroger, soit il envoie son état au moment de la notification.

Version du document
17 octobre 2022

2 Exemple

À titre d'illustration, un exemple basique est disponible sur le serveur *gitlab* de l'école : `https://git.esi-bru.be/ATL/designpatterncards`.

Dans cet exemple, le modèle est un simple compteur, un entier, avec 3 opérations : incrémentation, décrémentation et ajout d'un nombre au hasard.

La fenêtre principale permet de contrôler le modèle. Trois boutons permettent d'effectuer les 3 opérations du modèle. De plus 3 autres boutons permettent chacun d'ouvrir une vue du modèle, une fenêtre affichant la valeur :

- ▷ sous forme décimale du compteur pour la première vue ;
- ▷ sous forme hexadécimale, et ;
- ▷ sous forme binaire pour la troisième vue.

Le modèle est `Observable`, il permet aux observateurs de *s'inscrire* et de se *désinscrire*. Il notifie les observateurs chaque fois que la valeur du compteur change en appelant leur méthode `update`.

Les observateurs sont les trois vues, elles se mettent à jour chaque fois que le modèle les notifie. Chaque vue s'inscrit lors de leur création auprès du modèle en tant qu'observateur et se désinscrit lorsque la fenêtre est fermée.

3 Alternatives

Il existe beaucoup d'alternatives et de variantes à l'implémentation de ce patron de conception.

Généralement la méthode `update` reçoit en paramètre l'observable ainsi qu'un tableau de paramètres : `update (Object observable, Object[] params)`. Cela permet de réagir différemment en fonction de notifications différentes, ou de réagir de manière plus précise grâce à l'information supplémentaire fournie par les paramètres.

Notez que les classes `java.util.Observer` et `java.util.Observable` sont *dépréciées* et ne doivent donc plus être utilisées.

Vous trouverez trois implémentations différentes ici :
`https://www.baeldung.com/java-observer-pattern`.

La dernière implémentation utilise l'interface `PropertyChangeListener` qui fait office d'interface `Observeur`, et la classe `PropertyChangeSupport`, une classe utilitaire dédiée à gérer la liste des observateurs (*listeners*).

4 Loose coupling

Dans le *Design Pattern Observer*, on parle de *Loose coupling* entre le sujet d'observation et l'observateur car ils peuvent interagir sans connaître grand chose l'un de l'autre.

Loose coupling entre l'observateur et l'observé :

- La seule chose que l'*observé* connaît à propos de l'*observateur* c'est que celui-ci implémente l'interface `Observer` ;
- Un *observateur* de n'importe quel type peut être ajouté/retiré n'importe quand ;
- Il ne faut pas modifier `Observable` si un nouveau type d'*observateur* est ajouté.