

Les observables

Problème

Lorsque l'on a des évènements (par exemple nouvel articles) et que l'on a plusieurs terminaux pour recevoir cet évènement (écrans, notifications, emails, etc). Si on fait simplement tout dans une seule classe, cela enfreint le principe d **ouvert-fermé** (c'est à dire qu'un module doit être ouvert au changements sans avoir à le modifier).

Car ici pour ajouter ou supprimer des terminaux (observateurs), on est obligé de modifier tout le module.

Solution

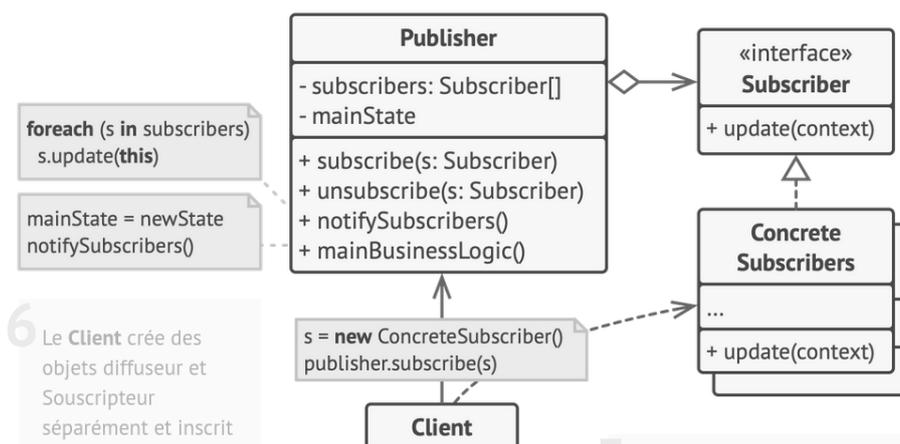
1 Le **Diffuseur** envoie des évènements intéressants à d'autres objets. Ces évènements se produisent quand le diffuseur change d'état ou exécute certains comportements. Le diffuseur possède une infrastructure d'inscription qui permet aux nouveaux souscripteurs de rejoindre la liste et aux souscripteurs actuels de la quitter.

2 Quand un nouvel évènement survient, le diffuseur parcourt la liste d'inscriptions et appelle la méthode de notification déclarée dans l'interface des souscripteurs sur chaque objet souscripteur.

3 L'interface **Souscripteur** déclare les méthodes de notification. Dans la majorité des cas, il n'y a qu'une seule méthode `update`. Elle peut prendre plusieurs paramètres pour que le diffuseur leur envoie plus de détails concernant la modification.

4 Les **Souscripteurs Concrets** exécutent certaines actions en réponse aux notifications envoyées par le diffuseur. Toutes ces classes doivent implémenter la même interface pour ne pas coupler le diffuseur avec leurs classes concrètes.

5 En général, les souscripteurs ont besoin de détails à propos du contexte afin d'exécuter correctement la mise à jour. C'est pour cela que les diffuseurs passent souvent des données du contexte en paramètre de la méthode de notification. Le diffuseur peut même s'envoyer lui-même en paramètre et laisser les souscripteurs récupérer directement les données nécessaires.



6 Le **Client** crée des objets diffuseur et **Souscripteur** séparément et inscrit les souscripteurs aux mises à jour du diffuseur.

La classe qui publie les évènements va stocker une liste d'observateurs et va implémenter des méthodes pour que les observateurs puissent s'abonner ou se désabonner. Les observateurs vont implémenter l'interface "Observateur" qui va simplement contenir une méthode pour recevoir l'évènement. A présent pour ajouter ou supprimer des terminaux il suffit d'abonner ou désabonner un observateur à la classe publisher.

Critique

Ce système a beaucoup d'avantages car il respecte le principe ouvert-fermé, il sépare les préoccupations et permet l'abonnement pendant l'exécution. Cependant, il rends aussi le code plus complexe et plus lent (car si un des observateurs est lent, il ralentit toute l'exécution et c'est généralement assez coûteux de le paralléliser). Il ne faut pas l'utiliser tout de suite ou tout le temps. Il faut seulement le faire lorsque c'est vraiment nécessaire (par exemple ne pas le faire si il n'y qu'un observateur).

Revision #2

Created 19 September 2023 18:08:51 by SnowCode

Updated 3 October 2023 21:07:06 by SnowCode