Le Décorateur

Le Décorateur (ou Decorator en anglais) est un modèle de conception structurel qui permet d'ajouter des fonctionnalités à un objet sans en changer sa structure et son fonctionnement.

Cette ajout de nouvelles fonctionnalités se fait grâce à la création de classes qui "entourent" un objet cible. Chaque ajout de fonctionnalité nécessite alors l'ajout d'une nouvelle couche à l'objet, celle-ci pouvant être ajoutée dynamiquement durant l'exécution du programme..

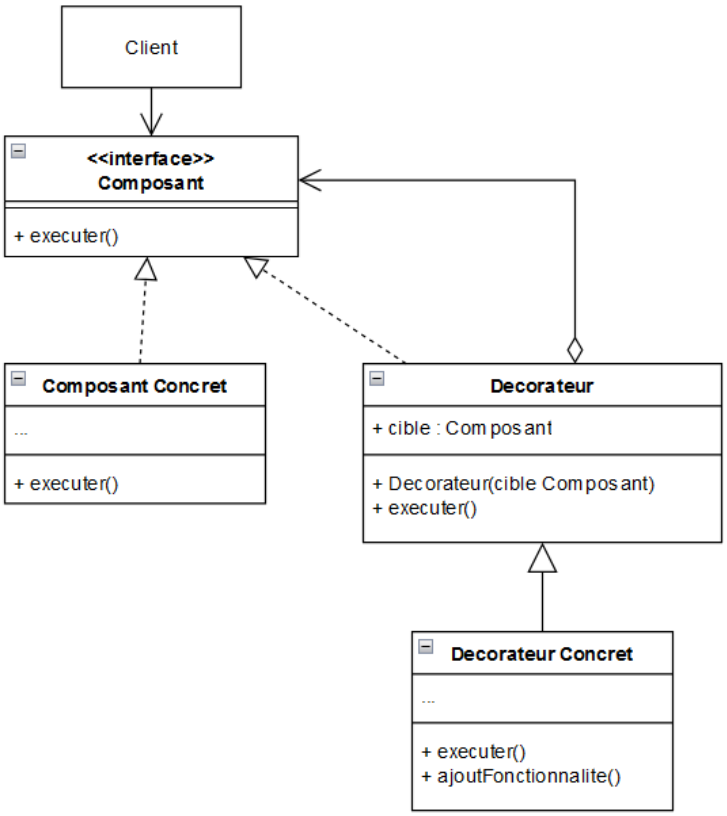
Respect des principes SOLID :

L'implémentation du modèle du Décorateur permet de respecter deux des principes **SOLID** : le **P**rincipe de **R**esponsabilité **U**nique ainsi que le **P**rincipe d'**O**uverture/**F**ermeture.

Principe de **R**esponsabilité **U**nique : Chaque fonctionnalité ajoutée se trouve dans une classe séparée sous la forme d'un décorateur.

Principe d'**O**uverture/**F**ermeture : Possibilité d'ajouter de nouvelles fonctionnalités sans avoir besoin de modifier les classes existantes

Structure :



Ici, l'interface Composant correspond à l'interface qui va définir les fonctionnalités qui vont être communes entre l'objet cible ainsi que les décorateurs qui lui sont associés.

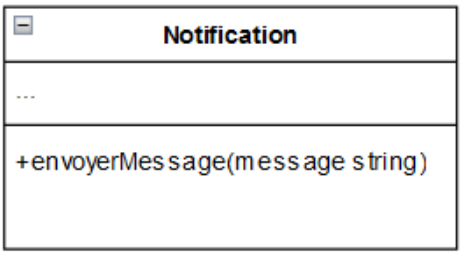
La classe Composant Concret correspond à la classe de l'objet qui va être "entouré". Elle implémente la classe "Composant" et possède l'implémentation du fonctionnement basique de celle- ci.

La classe Décorateur est la classe parent à tous les décorateurs. Elle possède en attribut une référence à l'objet ciblé. L'objet ciblé est de type "Composant" car cela permet au décorateur de posséder comme cible un Composant Concret ou alors un autre Décorateur.

Le Décorateur Concret implémente les fonctionnalités additionnelles qui sont ajoutées à l'objet ciblé. Il fait toujours appel au parent mais cet appel peut être fait après ou avant avoir exécuter les fonctionnalités supplémentaires.

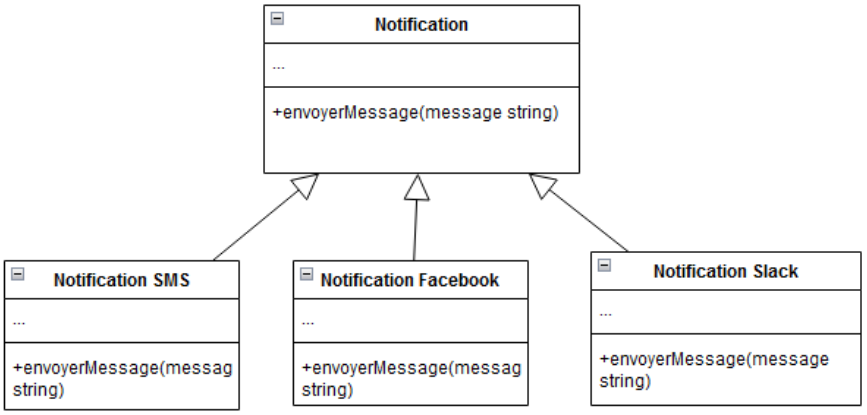
Exemple d'utilisation :

Prenons comme exemple la création d'une bibliothèque qui permet à un programme de notifier ses utilisateurs en cas évènements importants.
On crée alors dans un premier temps la classe Notification :

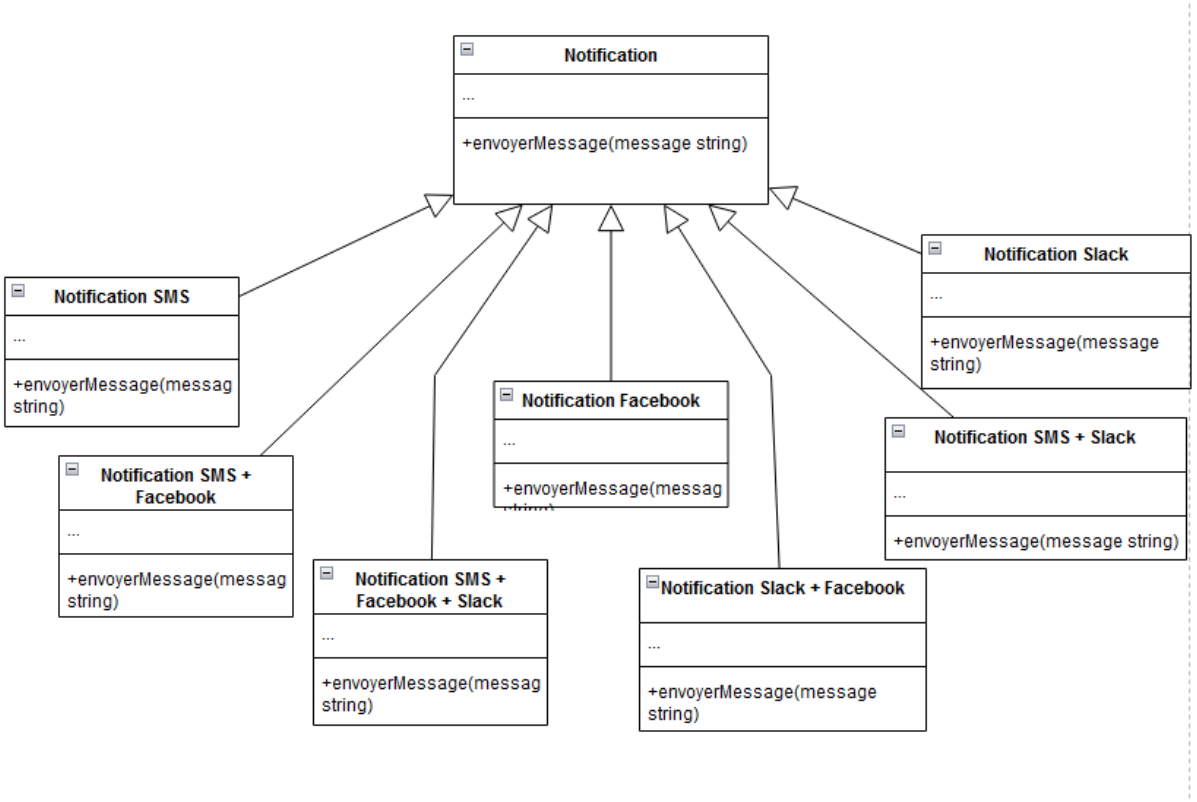


Par défaut, cette classe envoie les notifications à l'utilisateur seulement par Mails. Le problème étant que les utilisateurs de la bibliothèque souhaitent utiliser la bibliothèque pour envoyer des notifications via d'autres moyens comme SMS, Facebook ou encore Slack.

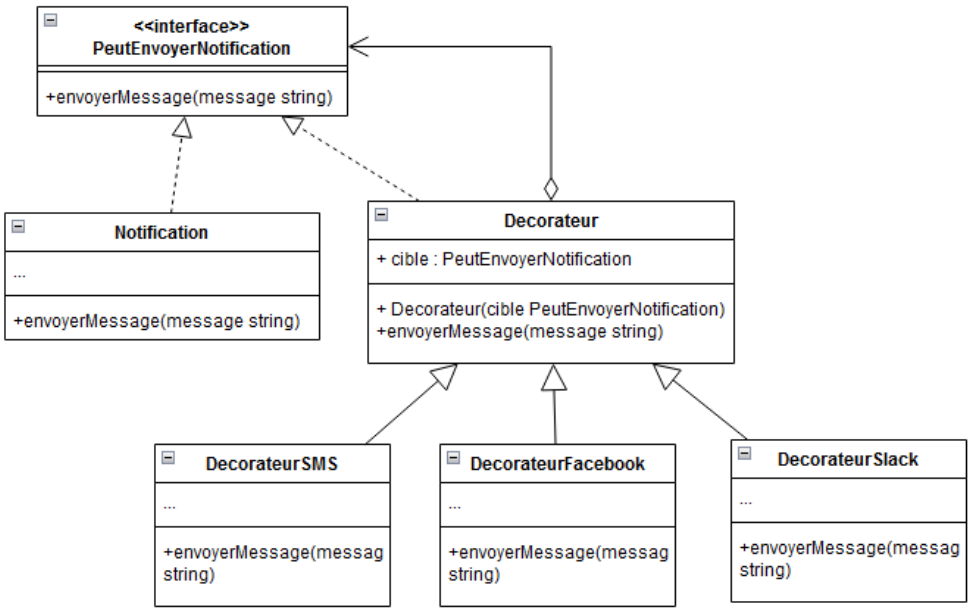
Avec la mise en place actuelle, il n'est possible d'ajouter de nouveaux moyens de notifications que via la création de classes filles.



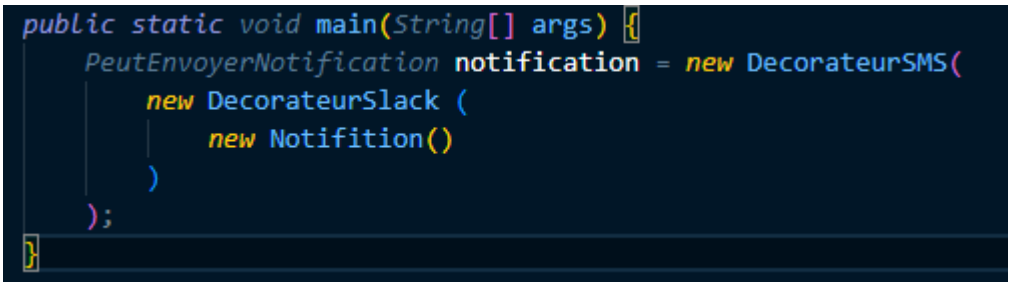
Le problème avec cette implémentation est qu'il est impossible pour l'utilisateur de la bibliothèque d'envoyer des notifications sur plusieurs service spécifique sans avoir une implémentation de ce type :



On peut alors régler cette implémentation en utilisant la structure du Décorateur ainsi que le principe de composition. L'utilisation du principe de composition au lieu de l'inheritance permet de modifier le comportement de l'objet pendant l'exécution.



Pour créer un objet de type Notification qui permet d'envoyer des notification via Mail, SMS et Slack, on devra alors utiliser la syntaxe suivante :



```
public static void main(String[] args) {
    PeutEnvoyerNotification notification = new DecorateurSMS(
        new DecorateurSlack (
            new Notifition()
        )
    );
}
```

On peut alors ajouter autant de Décorateur que l'on souhaite.

Conclusion :

Le pattern Décorateur est un modèle de conception structurel qui favorise l'extensibilité et la modularité des applications. Il permet d'ajouter des fonctionnalités à un objet de manière flexible tout en évitant de multiplier les sous-classes (grâce au principe de composition). Cependant, il doit être utilisé avec prudence pour éviter une trop grande complexité dans la gestion des objets "entourés".