**SOLID est le terme donné par Oncle Bob, chaque caractère représente un principe de conception :**

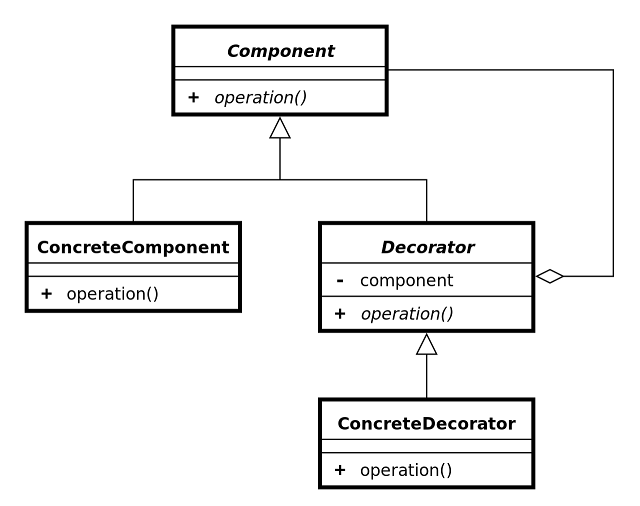
* **S pour le principe Single Responsibility: qu'on doit avoir qu'une seul raison métier pour modifier un bloc de code, une méthode, ou une classe.**
* **pour le principe Open closed design: si je veux faire évoluer les fonctionnalité de mon application, la seul chose que je dois faire c'est étendre certaines classe bien choisies, via l'utilisation des designe pattern (stratégie, bridge).**
* **L pour le principe Liskov substitution principle: si B se comporte comme b, alors B étends A.**
* **I pour le principe Interface segregation: les méthodes d'une interface doivent être minimal par rapport au méthodes dont le client a besoin.**
* **D pour le principe Dependency inversion: au lieu de dépendre d'une classe on dépend d'un interface.**

**6. Quelle est la différence entre les patterns Decorator, Proxy et Adapter en Java ?**

**Ces patterns ressemblent beaucoup parce que leur structure ou leur diagramme de classe est très similaire, mais leur intention est très différente. Decorator ajoute des fonctionnalités supplémentaires sans toucher à la classe, Proxy fournit un contrôle d’accès et Adapter est utilisé pour faire fonctionner deux interfaces incompatibles.**

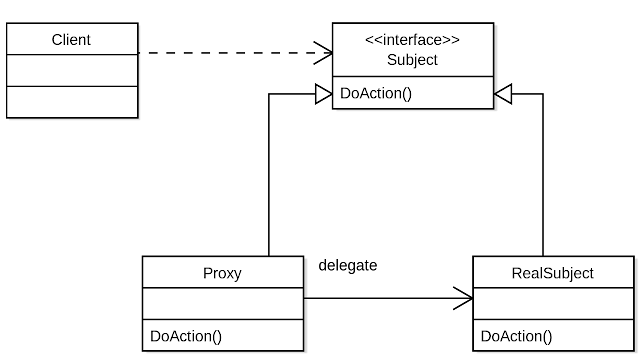
Design pattern Decorator

Le pattern Decorator permet à un utilisateur d’ajouter de nouvelles fonctionnalités à un objet existant sans modifier sa structure. Ce type de pattern est inclus dans le pattern structurel, car il agit comme une enveloppe pour la classe existante. Ce pattern crée une classe décorateur qui enveloppe la classe d’origine et fournit des fonctionnalités supplémentaires en préservant la signature des méthodes de classe.



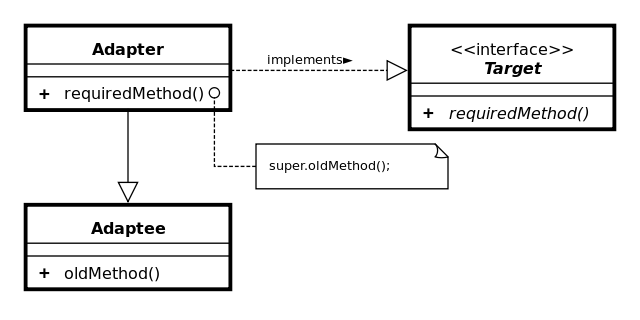
Design pattern Proxy

Dans le pattern Proxy, une classe représente la fonctionnalité d’une autre classe. Ce type de pattern appartient du pattern structurel. Dans le pattern Proxy, nous créons un objet ayant un objet original pour interfacer ses fonctionnalités avec le monde extérieur.



Design pattern Adapter

Le pattern Adapter fonctionne comme un pont entre deux interfaces incompatibles. Ce type de pattern appartien au pattern structurel car ce pattern combine la capacité de deux interfaces indépendantes. Un exemple concret pourrait être un lecteur de mémoire servant d’adaptateur entre une carte mémoire et un ordinateur portable.

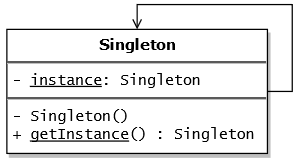


**7. Quand utiliser le pattern de conception Singleton en Java?**

Réponse

Lorsque nous n’avons besoin que d’une instance d’une classe et que nous souhaitons qu’elle soit disponible globalement, nous pouvons utiliser ce pattern. Ce n’est pas gratuit car cela augmente le couplage entre les classes et les rend difficiles à tester.

Design pattern Singleton



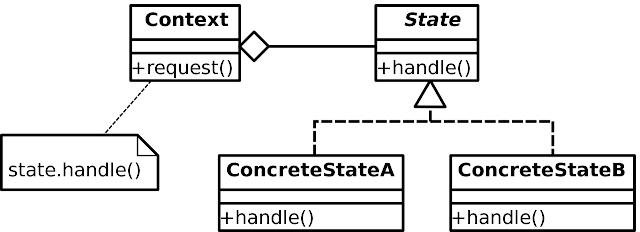
8. Quelle est la différence entre State et Strategy Pattern?

Réponse

Bien que la structure ou le diagramme de classes du pattern State et Strategy soit identique, leur intention est complètement différente. Le pattern State est utilisé pour faire quelque chose de spécifique en fonction de l’état tandis que le pattern Strategy vous permet de basculer entre les algorithmes sans changer le code qui l’utilise.

Design pattern State

Dans le pattern State, le comportement d’une classe change en fonction de son état. Ce type de pattern est associé au pattern de comportement. Dans le pattern State, nous créons des objets qui représentent différents états et un objet de contexte dont le comportement varie lorsque son objet d’état change.

  
**source : wikimedia.org**

Design pattern Strategy

Dans le pattern Strategy, un comportement de classe ou son algorithme peuvent être modifiés au moment de l’exécution. Ce type de pattern est associé au pattern de comportement. Dans le pattern Stratégie, nous créons des objets représentant différentes stratégies et un objet de contexte dont le comportement varie en fonction de son objet de stratégie. L’objet de stratégie modifie l’algorithme d’exécution de l’objet de contexte.

