Principes et patterns

Principes

Principe de substitution de **Liskov** : Les sous-types doivent pouvoir être substitués à leur type de base

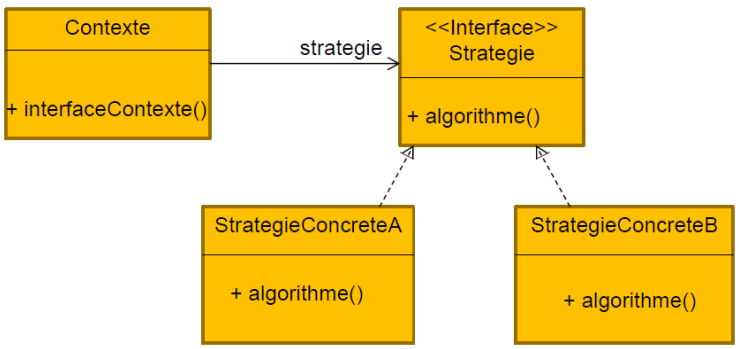
Principe « **ouvert-fermé** » : Les classes doivent être ouvertes à l'extension mais fermées à la modification.

Pattern Stratégie

Objectif : Définir une famille d'algorithmes, encapsuler chacun d'eux et les rendre interchangeables en fonction du contexte.

Solution : Séparer la sélection de l'algorithme de son implémentation. La sélection est basée sur le contexte.

Schéma :



Exemple :

- Un des exemples du pattern Stratégie dans l'API JAVA permet la définition de plusieurs stratégies de comparaison avec l'interface **Comparator<T>** :

```
public Comparator<T>{
    public int compare(T o1, T o2);
}
```

- Utilisation dans la classe **Collections**

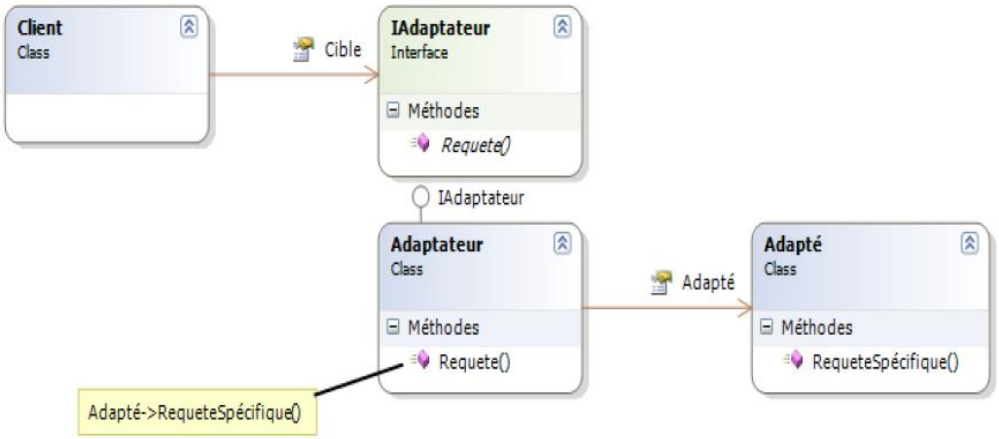
```
public static void sort( List<T> list, Comparator <?super T> c)
```

Pattern Adaptateur

Objectif : Faire correspondre à une interface donnée un objet existant que vous ne contrôlez pas.

Solution : L'adaptateur fournit un encapsuleur avec l'interface voulue.

Schéma :

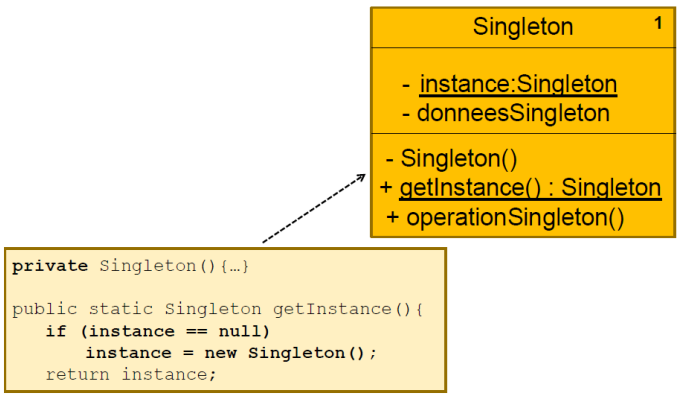


Pattern Singleton

Objectif : Garantir qu'une classe n'a qu'une seule instance et fournir un point d'accès global et unique à cette instance.

Solution : S'assurer qu'il n'existe qu'une seule instance en définissant une méthode statique de la classe qui retourne le singleton.

Schéma :

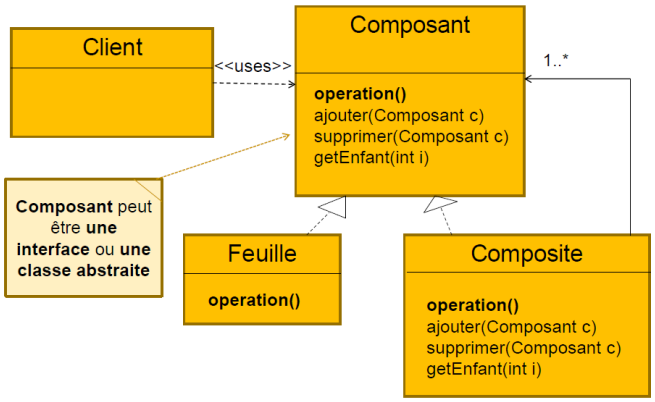


Pattern Composite

Objectif : Effectuer des opérations sur des objets individuels et des combinaisons de ceux-ci sans avoir à les distinguer.

Solution : Composer des objets en des structures arborescentes pour représenter une hiérarchie composant/composé de manière à ce que le client manipule les objets à travers une interface commune.

Schéma :



Exemple :

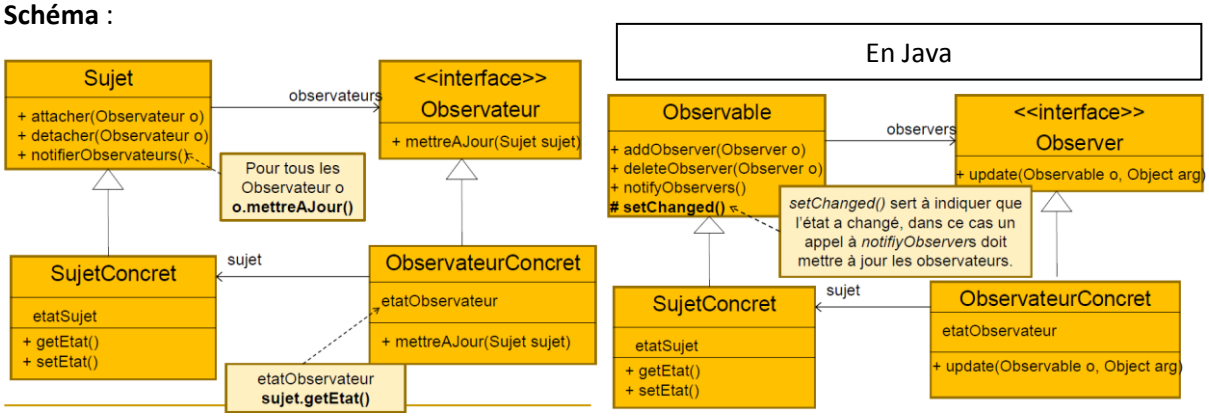
- Une figure simple peut-être un point, une ligne ou un cercle. Une figure peut-être composée d'autres figures, simples ou elles-mêmes composées d'autres figures. Toutes les figures peuvent être dessinées ou translatées.
- Modéliser la hiérarchie d'objets.

Pattern Observateur

Objectif : Définit une dépendance de type un à plusieurs, de façon telle que, quand un objet change d'état, tous ceux qui en dépendent en soient notifiés et automatiquement mis à jour.

Solution : Les observateurs délèguent la responsabilité de contrôle d'un événement à un objet central, le sujet.

Schéma :



Pattern MVC

Fonctionnement :

Il s'agit d'une collection de patterns :

- Le pattern **Observateur** permet au modèle de mettre à jour les vues et les contrôleurs de ses changements d'états. Les vues et les contrôleurs s'enregistrent comme observateurs auprès du modèle.
- Le pattern **stratégie** est mis en œuvre entre la vue et le contrôleur. Le contrôleur encapsule la stratégie liée au comportement de l'interface. La vue gère les aspects visuels et délègue au contrôleur la stratégie de gestion des actions des utilisateurs.
- Le pattern **composite** : Chaque composant fournit un affichage. Un composant pouvant être une feuille comme un bouton ou un composite composé d'autres composants.

Schéma :