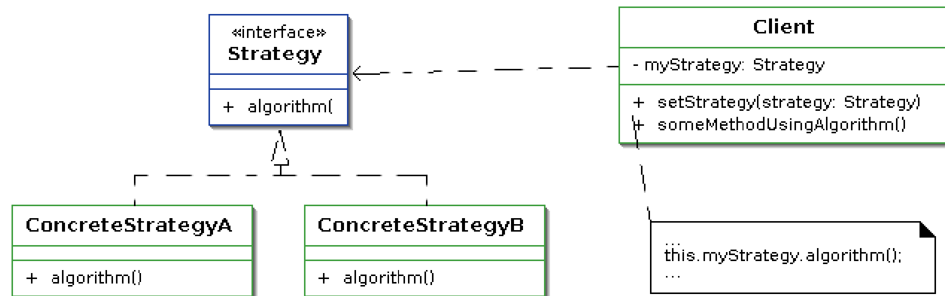
Design Pattern : strategy

Intent

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Structure



Eléments caractéristiques

- un attribut représentant la stratégie qui est d'un type abstrait
- l'invocation des méthodes de la stratégie, en faisant varier le type concret de la stratégie on change le comportement apparent de la classe cliente.

On remplace la variation du comportement que l'on peut obtenir par héritage par une variation du comportement par composition via la stratégie.
De plus, cette variation peut apparaître dynamiquement pendant « l'existence » de l'objet puisqu'il suffit de modifier la stratégie utilisée via `setStrategy()`. Il en résulte également que les comportements de tous les objets d'une même classe `Client` peuvent s'exprimer différemment, via leurs différentes stratégies.

Exemples rencontrés

API Java : Layout Manager

Dans les api awt/swing, un objet `Container` gère la disposition des différents objets graphiques qu'il contient grâce à un `LayoutManager`. Celui-ci modélise les *stratégies* de placement du *container*.

- `public interface LayoutManager`
  Defines the interface for classes that know how to lay out Containers.

  et ses implémentations via les classes : `java.awt.FlowLayout`, `java.awt.GridLayout`, `java.awt.BorderLayout`, ...

- Dans `java.awt.Container` :

  `public void setLayout(LayoutManager mgr)` *Sets the layout manager for this container.*

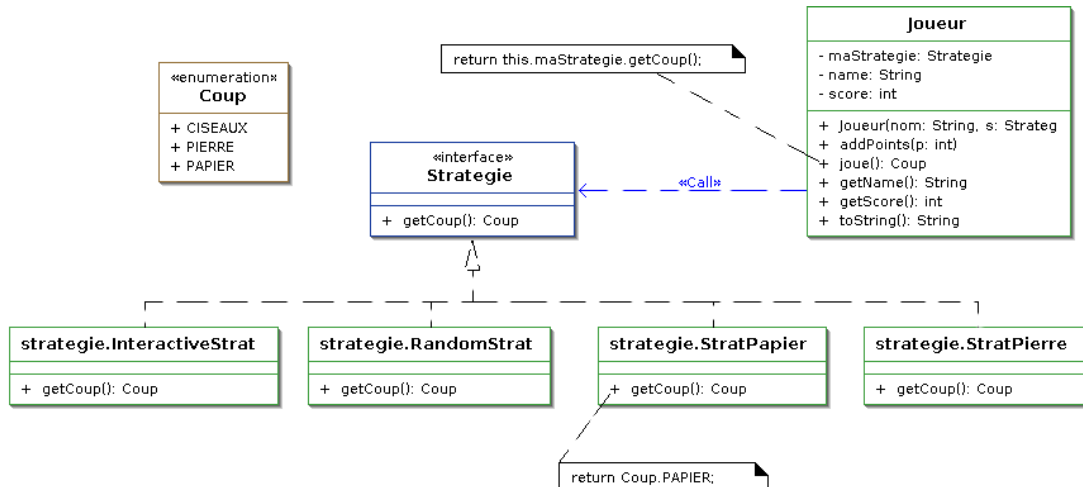
  Cette méthode permet de fixer/changer la stratégie de placement adoptée par le *container*.

API Java : FilenameFilter

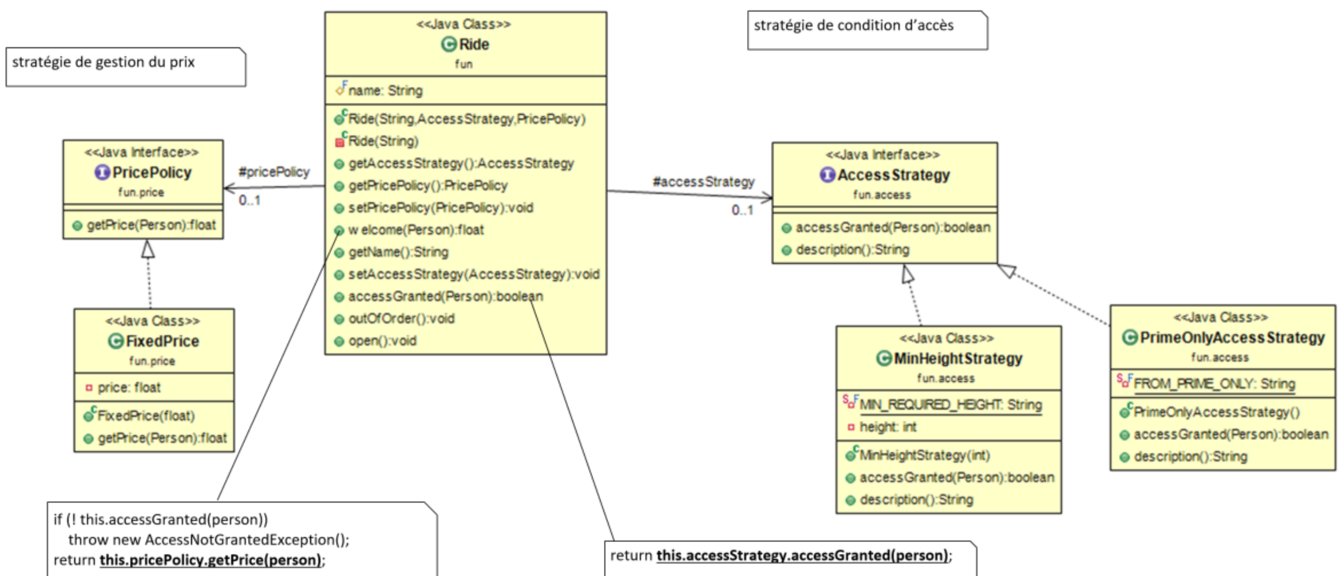
L'interface `FilenameFilter` représente le type des stratégies de sélection des fichiers par la méthode `list` de la classe `File`. Il y a ici une petite différence car la stratégie n'est pas attachée à la classe mais passée en paramètre lors de l'appel de la méthode mais le principe reste bien le même.

cc BY NC SA

S4 – Pierre-feuille-ciseaux : joueurs et stratégie
Une seule classe de joueur et plusieurs classes de stratégie. Lorsque le joueur joue c'est en fait la stratégie qui fait le choix du coup à jouer. En changeant la stratégie d'un joueur à l'autre on obtient des joueurs qui jouent différemment



TD – Attractions : condition d'accès et politique de prix
Pour une attraction (*ride*), la condition d'accès est une stratégie : la méthode `accessGranted()` délègue le travail à la méthode `accessGranted()` de son attribut `accessStrategy`.
Il en est de même pour la politique de prix : le calcul du prix de l'attraction retournée par la méthode `welcome()` de `Ride` est géré par la méthode `getPrice()` de l'attribut `pricePolicy`.
La mise en place de ces deux stratégies permet une multiplicité des combinaisons condition d'accès/politique de prix lors de la création des différentes attractions sans avoir à décliner des sous-types de `Ride`.
De plus, il est possible de dynamiquement faire évoluer le comportement des attractions par ue évolution de leurs stratégies. On peut en effet modifier la contrainte d'accès (par exemple pour les « journée spéciales enfants ») ou la politique de prix (par exemple pour les « journée discount »).



TD – Plugins : PluginChecker et FileNameFilter Pour les objets `PluginChecker`, le `FileNameFilter` est une stratégie qui détermine les fichiers qui seront surveillés par le `PluginChecker`.