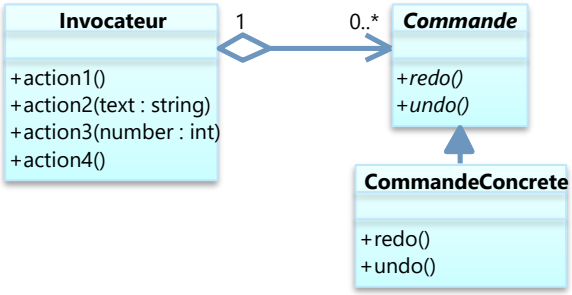
Catalogue des Design Patterns vus

Pattern 1 Commande

1.1 Définition

Le design pattern Commande encapsule la notion d'*invocation*. Il permet de séparer complètement le code initiateur de l'action, du code de l'action elle-même.

1.2 Diagramme de classe



1.3 Exemples

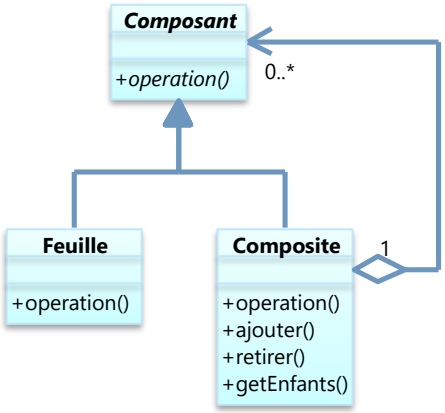
CM 17, TP 28

Pattern 2 Composite

2.1 Définition

Un objet composite est constitué d'un ou de plusieurs objets similaires (ayant des fonctionnalités similaires). L'idée est de manipuler un groupe d'objets de la même façon que s'il s'agissait d'un seul objet. Les objets ainsi regroupés doivent posséder des opérations communes, c'est-à-dire un "dénominateur commun".

2.2 Structure



2.3 Exemple

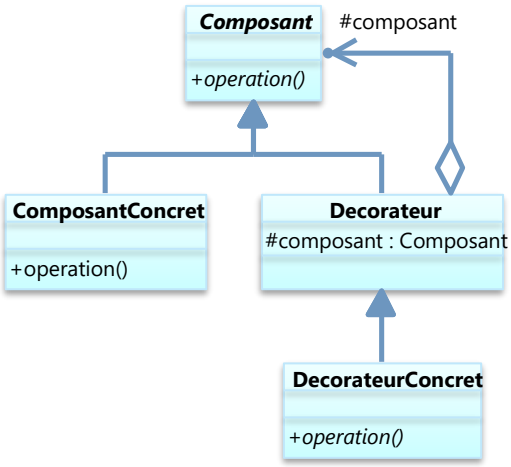
TP 25

Pattern 3 Décorateur

3.1 Définition

Le design pattern Décorateur permet d'attacher dynamiquement des responsabilités supplémentaires à un objet. C'est une alternative souple à l'héritage pour proposer de nouvelles fonctionnalités.

3.2 Diagramme de classe



3.3 Exemples

TD 10, TP Sup.02

Pattern 4 Modèle-Vue-Contrôleur (MVC)

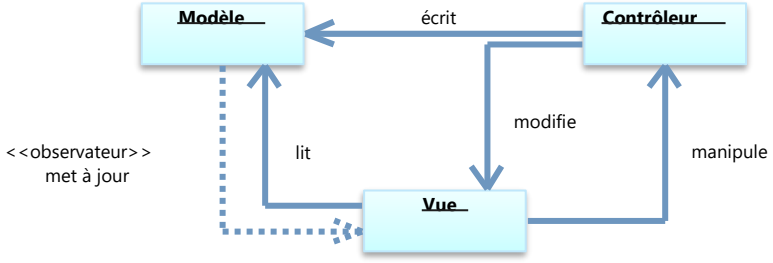
4.1 Définition

Le pattern modèle-vue-contrôleur est un modèle destiné à répondre aux besoins des applications interactives en séparant les problématiques liées aux différents composants au sein de leur architecture respective.

Ce paradigme regroupe les fonctions nécessaires en trois catégories :

1. un modèle (modèle de données) ;
2. une vue (présentation, interface utilisateur) ;
3. un contrôleur (logique de contrôle, gestion des événements, synchronisation).

4.2 Structure



4.3 Exemples

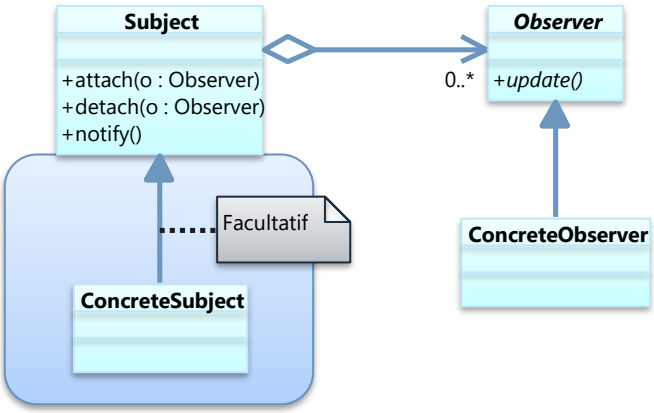
TD 13, CM 17

Pattern 5 Observateur

5.1 Définition

Le pattern Observateur définit une relation entre objets de type un-à-plusieurs, de façon que, lorsqu'un objet change d'état, tous ceux qui en dépendent en soient notifiés et soient mis à jour automatiquement.

5.2 Diagramme de classes



5.3 Exemple

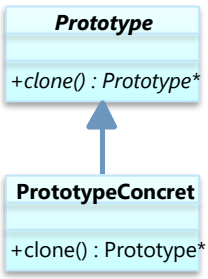
TD 13, TP 28, TP Sup.05

Pattern 6 Prototype

6.1 Définition

Le pattern Prototype permet la construction polymorphe d'objets. En d'autres termes, il est possible de copier un objet sans en connaitre le type. Ce pattern est utile lorsque la création d'un objet est coûteuse en temps ou ressources (décompression / téléchargement de données par exemple) en permettant la création de l'objet par recopie d'un objet déjà initialisé. Une seconde utilité de ce pattern est de rendre copiable des objets en ne connaissant que leur interface abstraite.

6.2 Diagramme de classes



6.3 Exemples

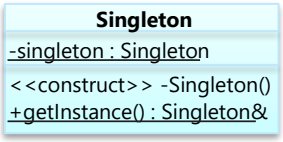
TD 12, TP Sup.04

Pattern 7 Singleton

7.1 Définition

Le pattern singleton permet de restreindre l'instanciation d'une classe à un seul objet (ou bien à quelques objets seulement). Il est utilisé lorsque l'on a besoin d'exactement un objet pour coordonner des opérations dans un système.

7.2 Diagramme de classes



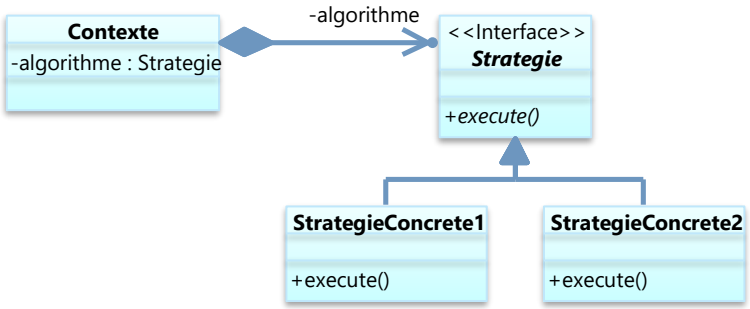
NB : un membre souligné est un membre statique

Pattern 8 Stratégie

8.1 Définition

Le patron de conception stratégie est utile pour des situations où il est nécessaire de permuter dynamiquement les algorithmes utilisés dans une application. Le patron stratégie est prévu pour fournir le moyen de définir une famille d'algorithmes, encapsuler chacun d'eux en tant qu'objet, et les rendre interchangeables. Ce patron laisse les algorithmes changer indépendamment des clients qui les emploient.

8.2 Diagramme de classes



8.3 Exemple

TD 12 :

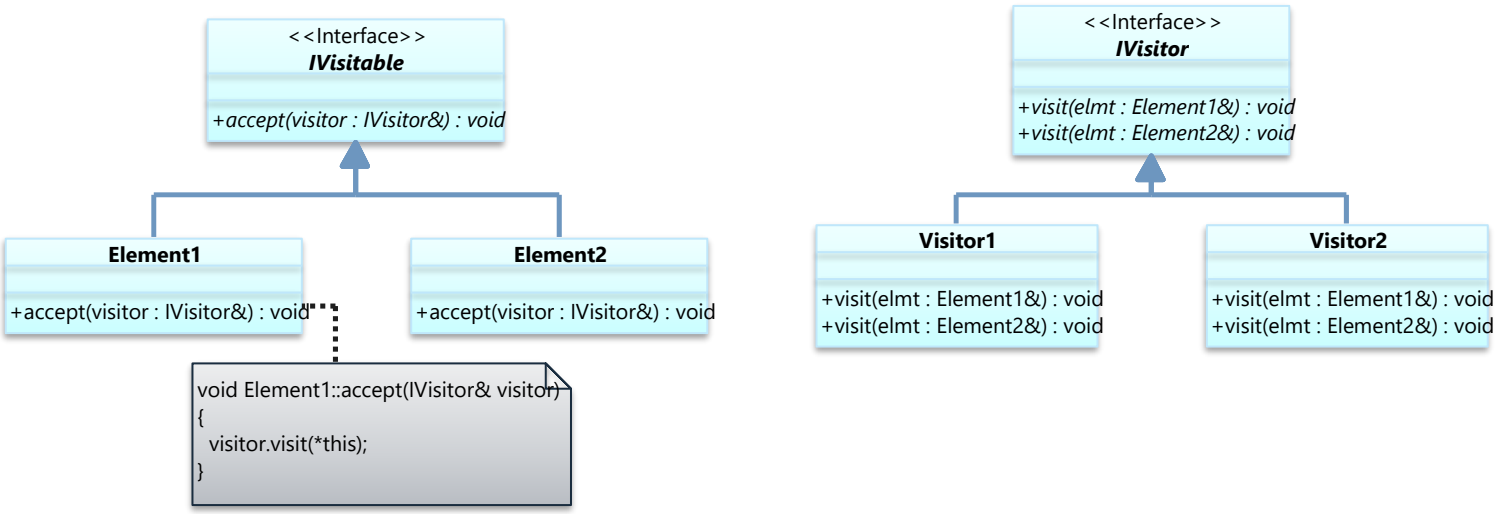
Applicable pour le choix de l'algorithme d'intelligence artificielle – méthode « `agir` ». La classe `IA` est similaire à la classe `Context` du pattern. L'interface `Strategy` peut alors encapsuler la méthode `agir`, similaire à la méthode `execute` du pattern.

Pattern 9 Visiteur

9.1 Définition

Le pattern visiteur permet de séparer un algorithme d'une structure de données. Nous l'avons utilisé pour mettre en place la sérialisation d'une structure de donnée supportant différents formats d'enregistrement.

9.2 Diagramme de classe



9.3 Exemple

CM 17