Modèles de conception

DESIGN PATTERN

Principe objet de base

- Classe / Instance
- Classes abstraites
- Interface / Implémentation
- Héritage / Composition
- Héritage de classe / Héritage d'interface
- Type / sous-type / super-type
- Signature
- Polymorphisme
- Couplage fort et faible

Notion de Type

- Un type est le nom utilisé pour caractériser une interface particulière.
- Un objet peut avoir plusieurs types différents
- Des objets très différents peuvent partager le même type.
- Un type est un sous-type d'un autre si son interface contient l'interface de l'autre.
- Un type est un super-type d'un autre si celui-ci est un sous-type du premier.
- On dit qu'un sous-type hérite de l'interface de son super-type

Réutilisation

- Héritage Vs composition
- Délégation
- Couplage faible / fort

Couplage

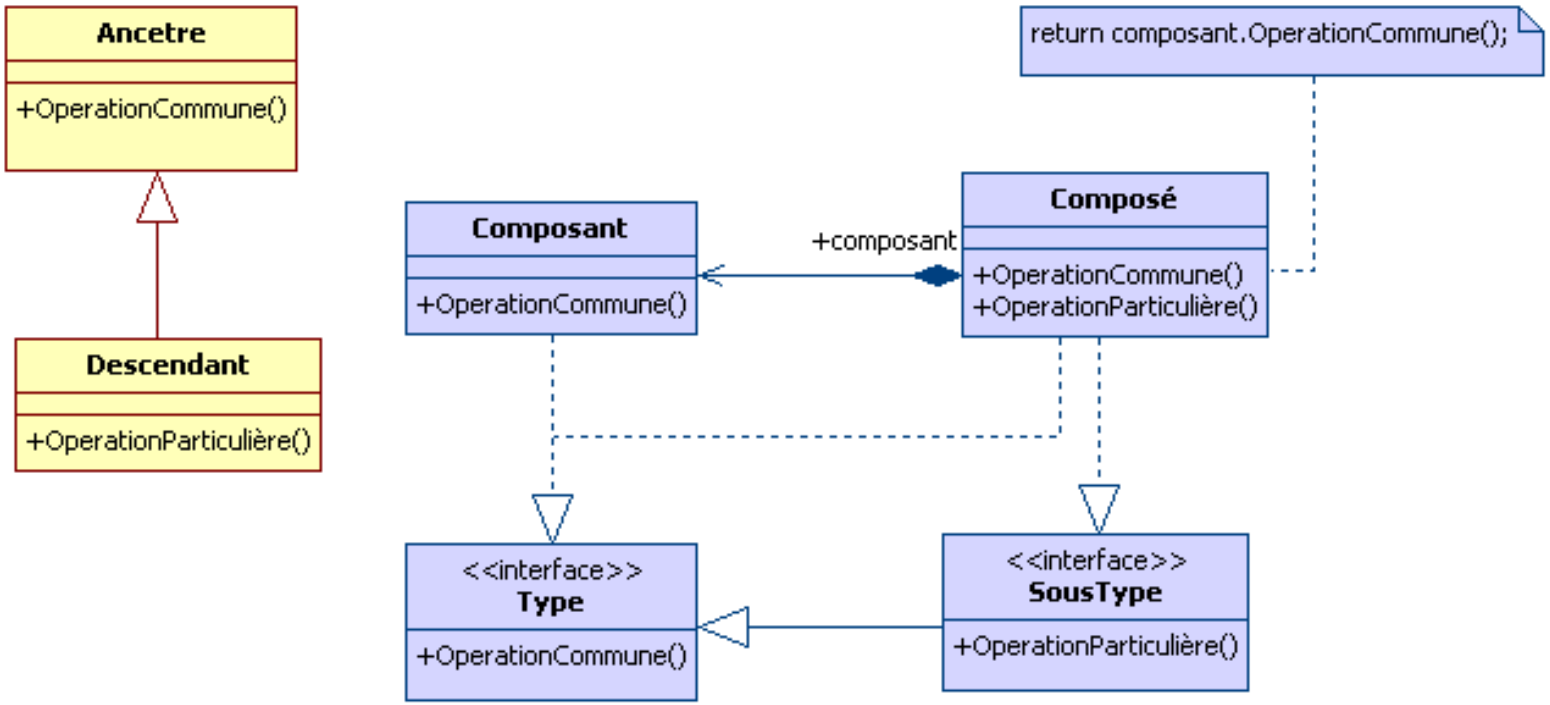
Selon Pressman[1], il existe sept niveaux de couplage, du plus faible au plus fort :

1. **Sans couplage** : les composants n'échangent pas d'information.
2. **Par données** : les composants échangent de l'information par des méthodes utilisant des arguments (paramètres) de type simple (nombre, chaîne de caractères, tableau).
3. **Par paquet** : les composants échangent de l'information par des méthodes utilisant des arguments de type composé (structure, classe).
4. **Par contrôle** : les composants se passent ou modifient leur contrôle par changement d'un drapeau (verrou).
5. **Externe** : les composants échangent de l'information par un moyen de communication externe (fichier, pipeline, lien de communication).
6. **Commun (global)** : les composants échangent de l'information via un ensemble de données (variables) commun.
7. **Par contenu (interne)** : les composants échangent de l'information en lisant et écrivant directement dans leurs espaces de données (variables) respectifs.

Héritage Vs composition

- L'héritage
  - Défini statiquement à la compilation.
  - Le Langage supporte automatiquement l'accès au méthode parente.
  - Grande dépendance entre le descendant et son ancêtre.
  - Héritage de type automatique
- La composition
  - Définie dynamiquement à l'exécution.
  - L'accès au méthode de la classe composante doit être implémentée.
  - La dépendance ce limite au niveau de la définition des interfaces.
  - Pas d'héritage de type. Si nécessaire, il faut la construire.

Héritage Vs composition



Programmer pour une interface, non pour une implémentation

- Déclarer ses variables du type des interfaces et non de celui des classes concrètes.
- Utiliser un des modèles créateurs pour rendre Le système indépendant de l'instanciation des classes concrètes.

Définition

- Les design patterns sont des modèles de conception objet.
- Ils décrivent des solutions standards pour répondre à des problèmes d'architecture et de conception des logiciels.
- L'objectif commun à ses différentes solutions est d'organiser le code d'une application pour qu'ils soit facilement :
  - adaptable
  - réutilisable

Description des modèles de conception

- La description d'un modèle de conception suit un formalisme fixe :
  - **Nom** : représente le principe
  - **Intention** : quel est la raison d'être et le but du modèle.
  - **Description du problème à résoudre** : quelles situations peuvent tirer avantage de l'utilisation du modèle.
  - **Description de la solution** :
    - La structure de la solution (représentée par un schéma UML)
    - les constituants (description du rôle de chaque élément)
    - leurs relations (description des interactions entre les constituants).
  - **Conséquences** : résultats issus de la solution.

Classement des modèles de conception (GoF)

- Classement en fonction de leur rôle
  - Créateur : processus de création d'instance
  - Structurel : composition des classes ou des objets.
  - Comportementaux : interactions et responsabilités entre les classe ou les objets.
- Classement en fonction de leur domaine
  - Les modèles de classe : les relations entre les classes sont établie par héritage et donc statique.
  - Les modèles d'objet ou d'instance : les liaisons entre les instances sont établies au moment de l'exécution.

Classement des modèles de conception (GoF)

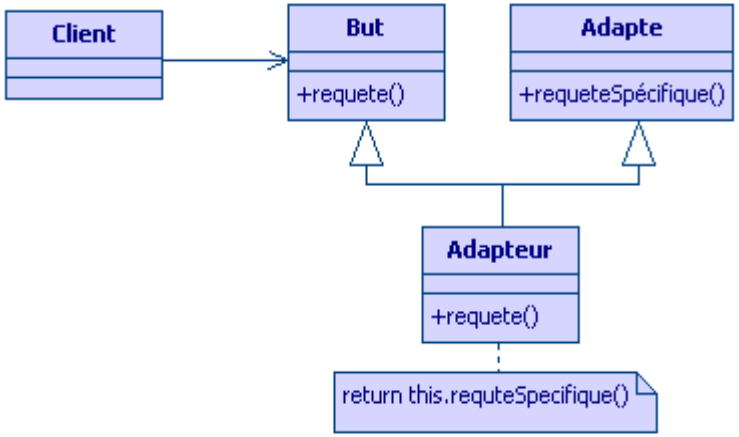
		Rôles		
		Créateur	Structurel	Comportementaux
Domaine	Classe	Fabrication	Adaptateur(classe)	Interprète Patron méthode
	Objet	Fabrique abstraite Monteur Prototype Singleton	Adaptateur(objet) Pont Composite Décorateur Façade Poids Mouche Procuration	Chaîne de responsabilité Commande Itérateur Médiateur Mémento Observateur Etat Stratégie Visiteur

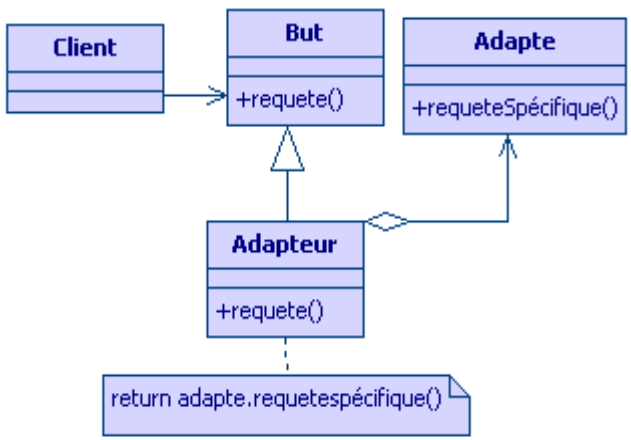
Classement par objectifs

Interfaces	Responsabilité	Construction	Opérations	Extensions
Adaptateur	Singleton	Monteur	Patron méthode	Décorateur
Façade	Observateur	Fabrication	Etat	Itérateur
Composite	Médiateur	Fabrique abstraite	Stratégie	Visiteur
Pont	Procuration	Prototype	Commande	
	Chaine de responsabilité	Mémento	Interprète	
	Poids Mouche			

Adaptateur (Adapter)

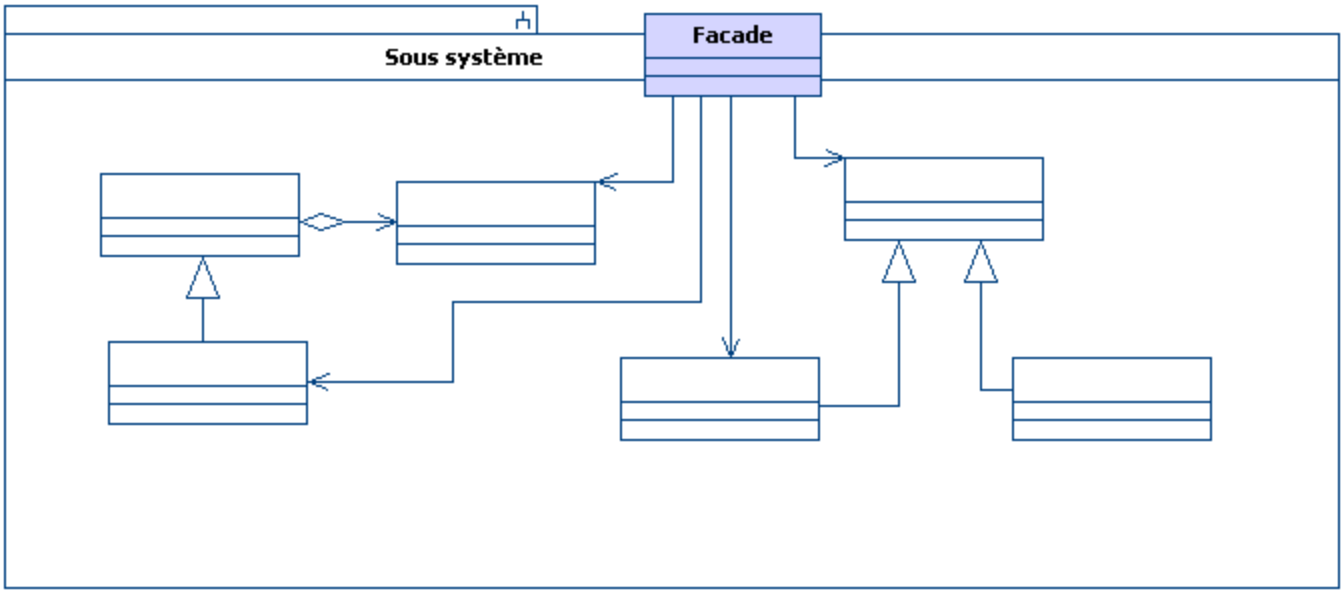
Intention : Convertit l'interface d'une classe en une interface distincte, conforme à l'attente de l'utilisateur. Permet à des classes dont les interfaces sont incompatibles de collaborer entre elles.





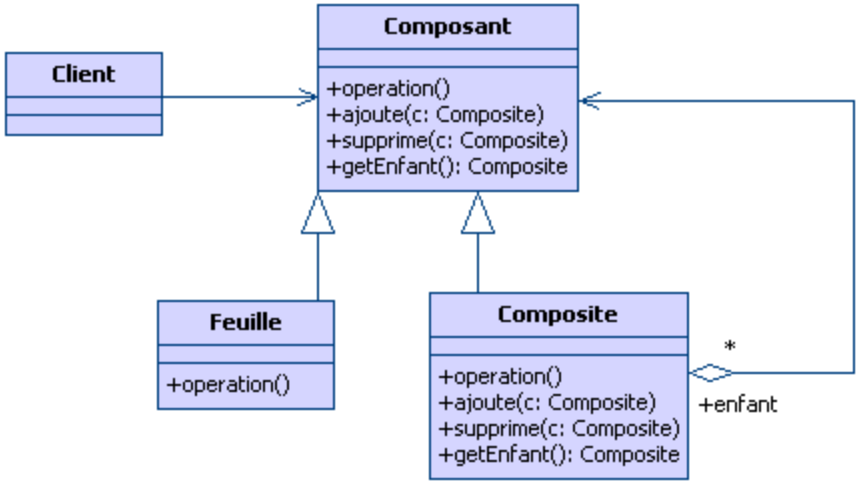
Façade (Facade)

Intention : Fournit une interface unifiée pour un ensemble d'interface d'un sous système. Façade définit une interface de plus haut niveau, qui rend le sous-système plus facile à utiliser



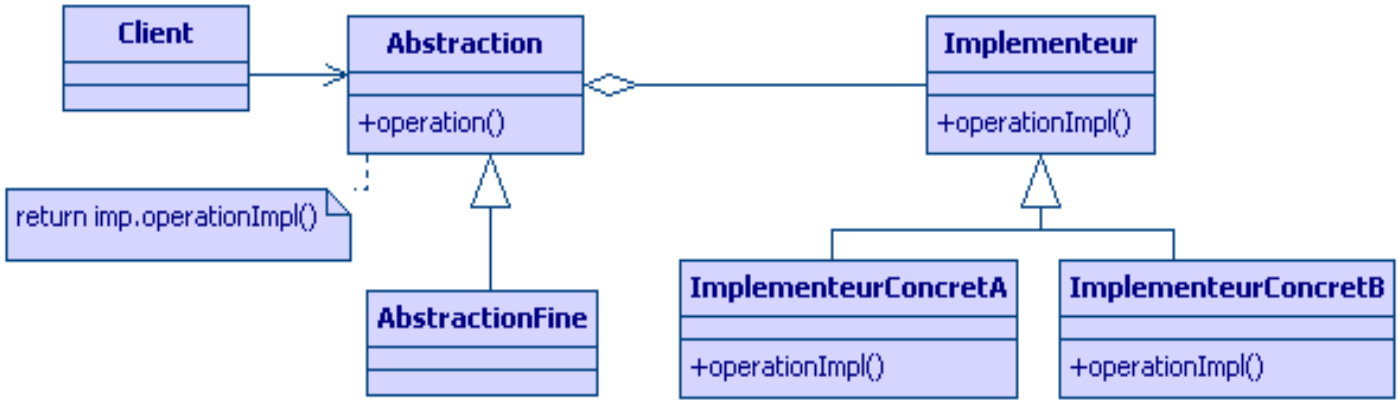
Composite (Composite)

Intention : Organise les objets en structure arborescente représentant le hiérarchie de bas en haut. Le composite permet aux utilisateurs de traiter des objets individuels, et des ensembles organisés de ses objets de la même façon.



Pont (Bridge)

Intention : Découple une abstraction de son implémentation associée afin que les deux puissent être modifiés indépendamment.

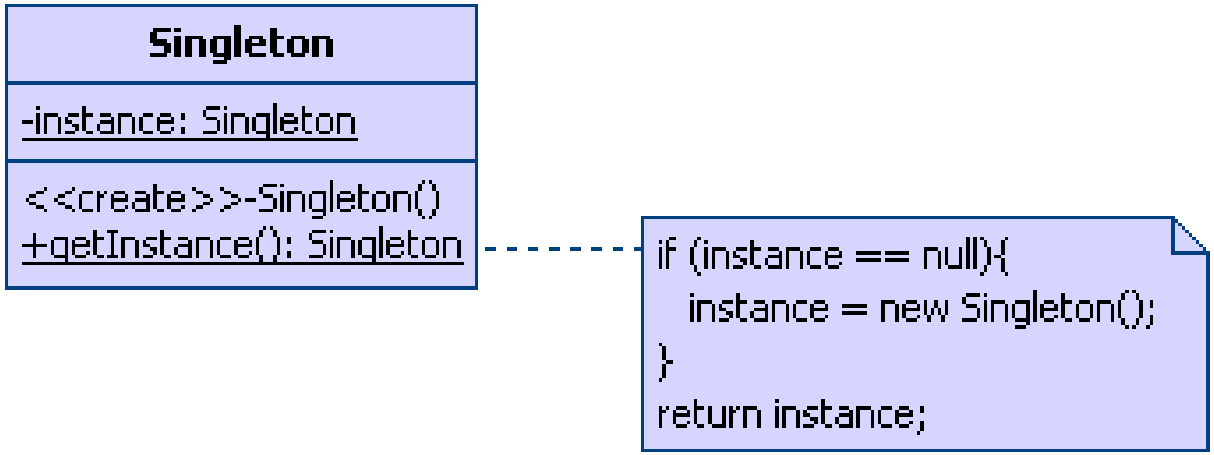


Classement par objectifs

Interfaces	Responsabilité	Construction	Opérations	Extensions
Adaptateur	Singleton	Monteur	Patron méthode	Décorateur
Façade	Observateur	Fabrication	Etat	Itérateur
Composite	Médiateur	Fabrique abstraite	Stratégie	Visiteur
Pont	Procuration	Prototype	Commande	
	Chaine de responsabilité	Mémento	Interprète	
	Poids Mouche			

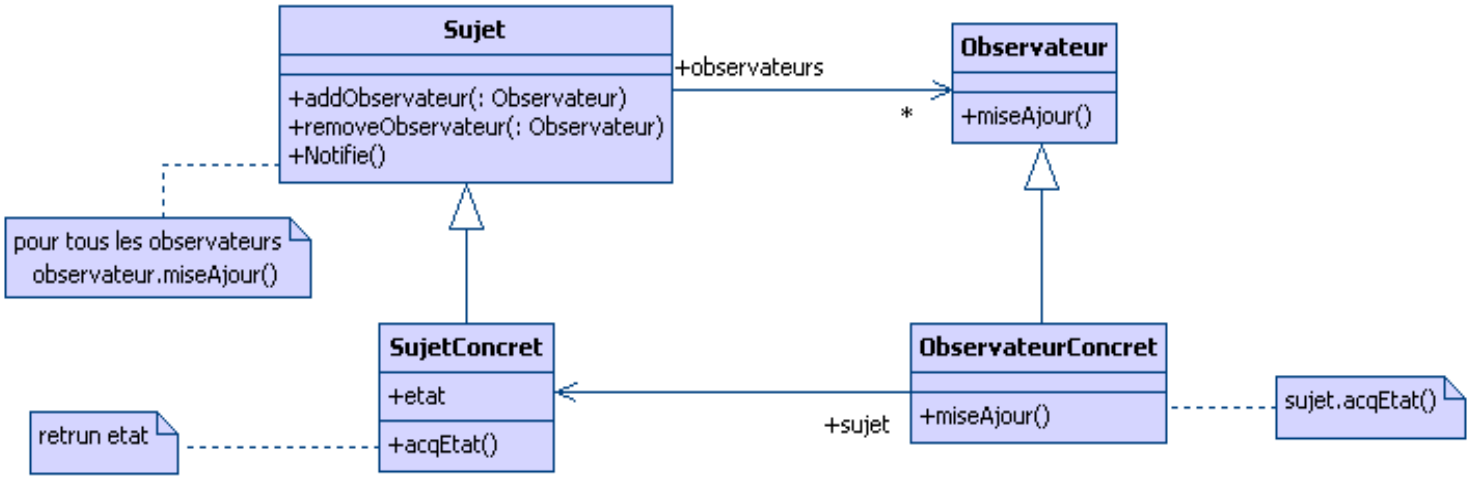
Singleton (Singleton)

Intention : Garantit qu'une classe n'a qu'une seule instance, et fournit à celle-ci, un point d'accès de type global.



Observateur (Observator)

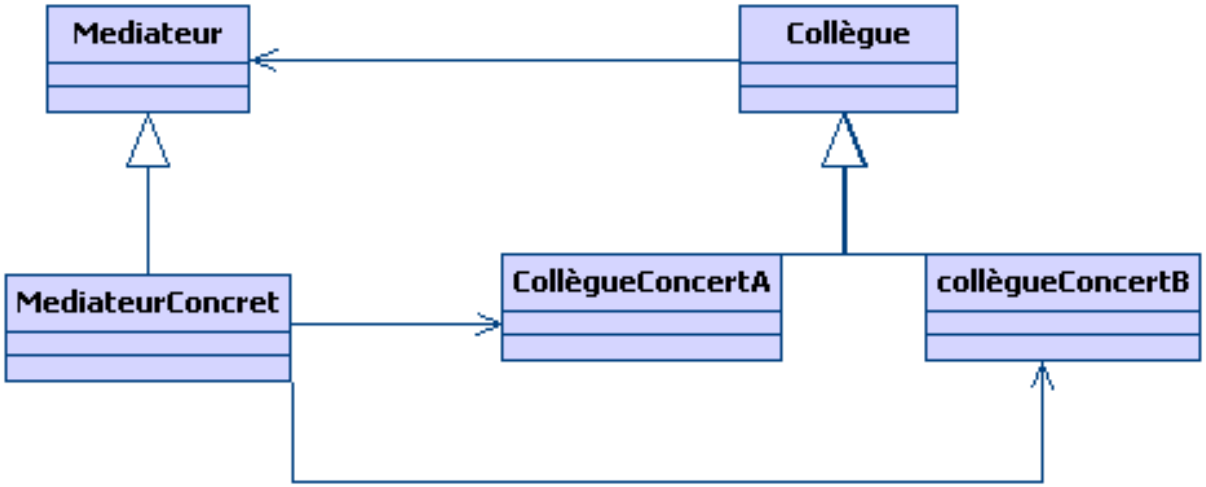
Intention : définit une corrélation entre objets du type un à plusieurs, de façon que, lorsqu'un objet change d'état, tous ceux qui en dépendent, en soient notifiés et mis à jour automatiquement.



Remarque : ce schéma présente un problème de dépendance.

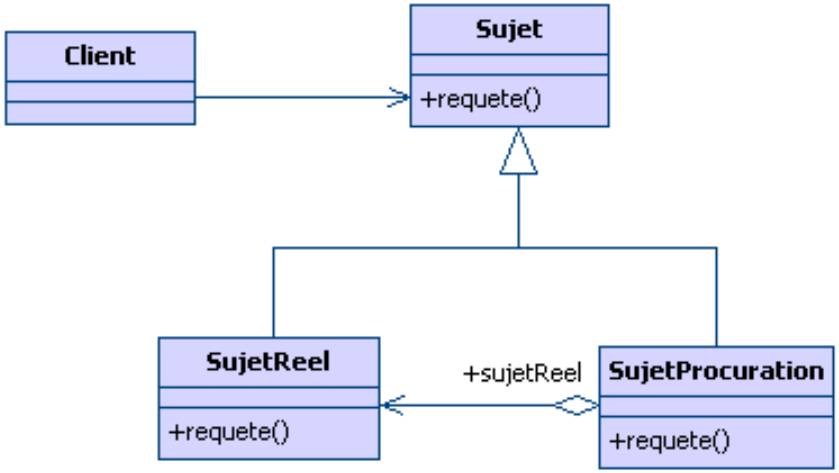
Médiateur (Mediator)

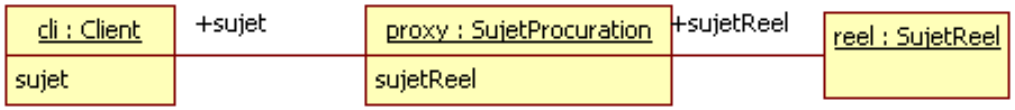
Intention : Définit un objet qui encapsule les modalité d'interaction de divers objet. Le médiateur favorise les couplages faibles, en dispensant les objets d'avoir à faire référence explicite les uns les autres.



Procuration (Proxy)

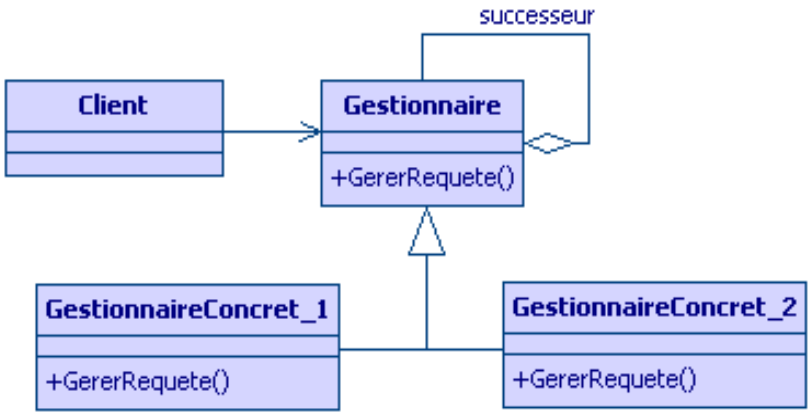
Intention : Fourni un remplaçant d'un autre objet, pour en contrôler l'accès.

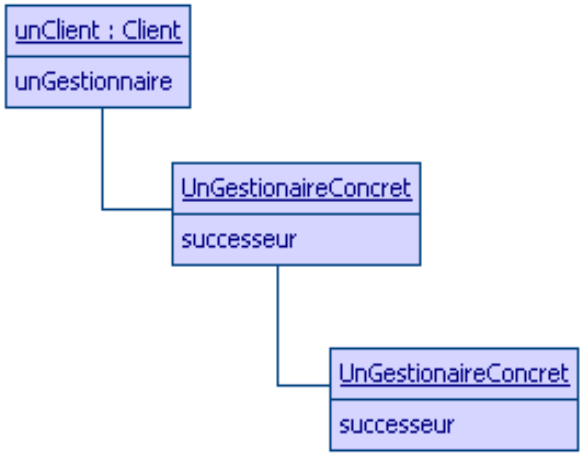




Chaîne de responsabilité (Chain of responsibiliy)

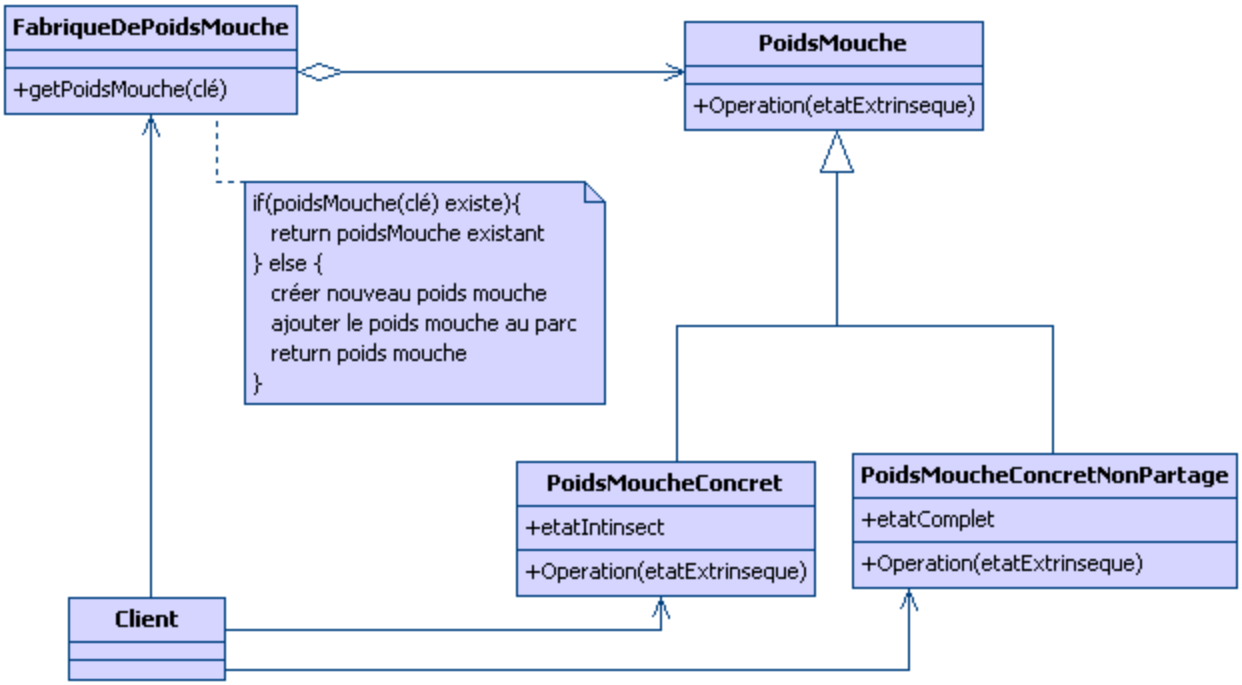
Intention : Permet d'éviter de coupler l'expéditeur d'une requête à son destinataire, en donnant la possibilité à plusieurs objets de prendre en charge la requête.





Poids Mouche (Flyweight)

Intention : Assure en mode partagé le support efficace d'un grand nombre d'objets à fine granularité.

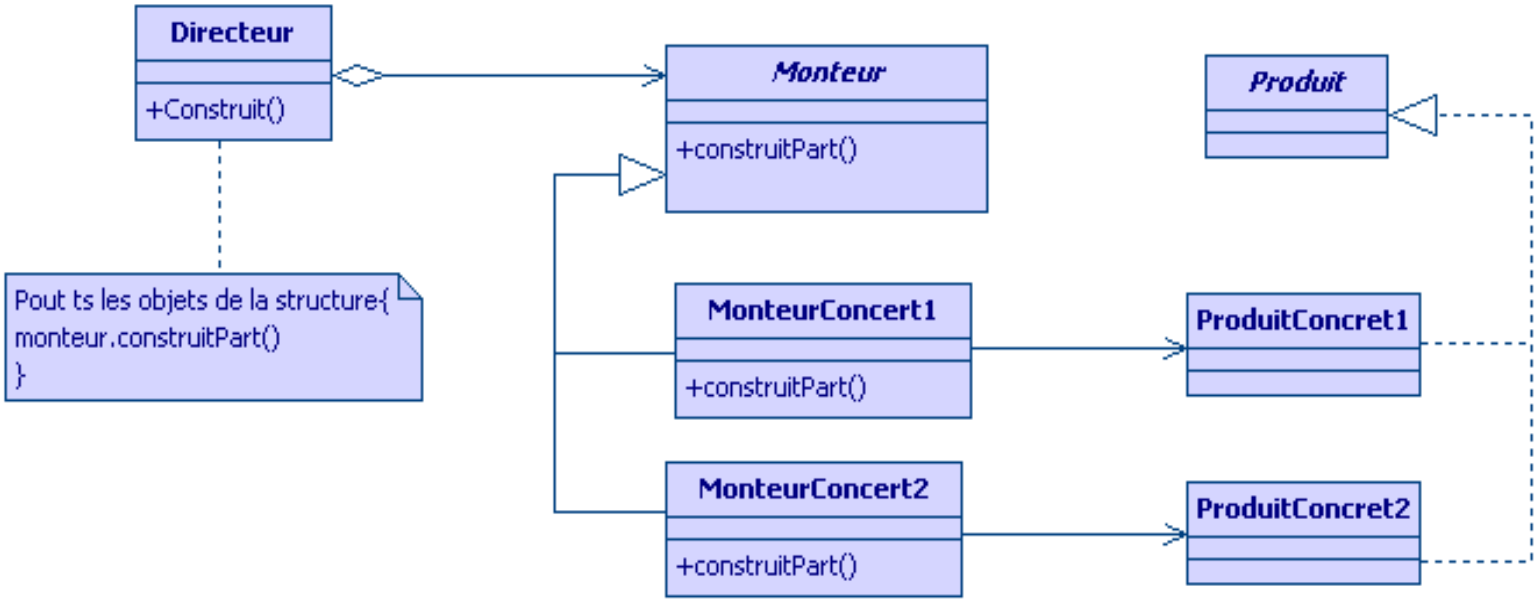


Classement par objectifs

Interfaces	Responsabilité	Construction	Opérations	Extensions
Adaptateur	Singleton	Monteur	Patron méthode	Décorateur
Façade	Observateur	Fabrication	Etat	Itérateur
Composite	Médiateur	Fabrique abstraite	Stratégie	Visiteur
Pont	Procuration	Prototype	Commande	
	Chaine de responsabilité	Mémento	Interprète	
	Poids Mouche			

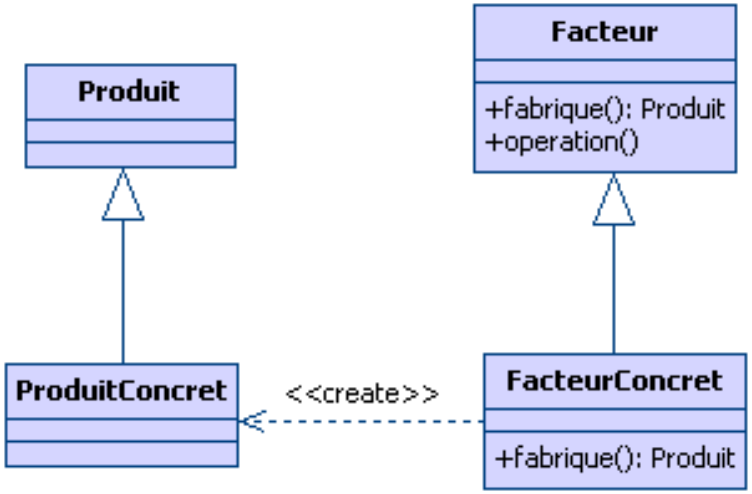
Monteur (Builder)

Intention : Dans un objet complexe, dissocie sa construction de sa représentation, de sorte que, le même procédé de construction puisse engendrer des représentation différentes



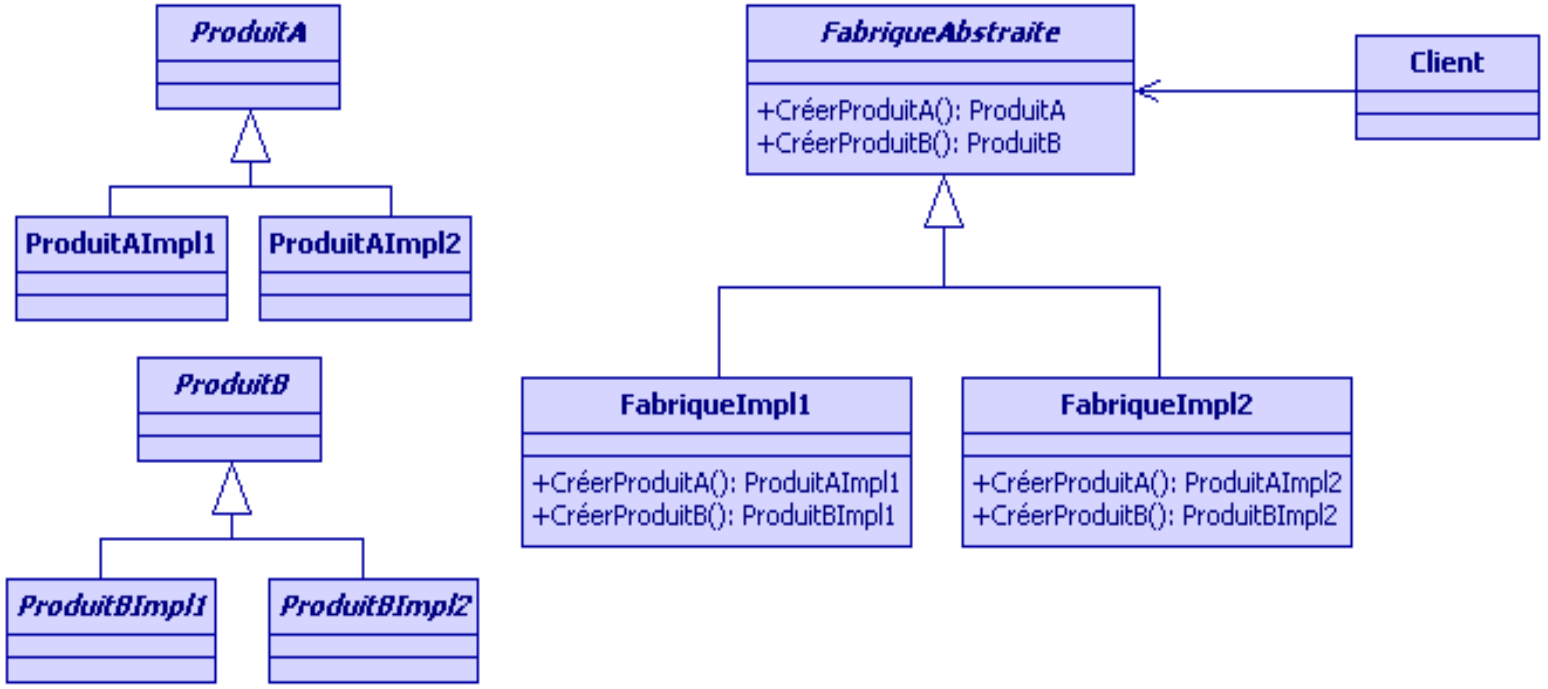
Fabrication (Factory method)

Intention : Définit une interface pour la création d'un objet, tout en laissant aux sous classes le choix de la classe à instancier.



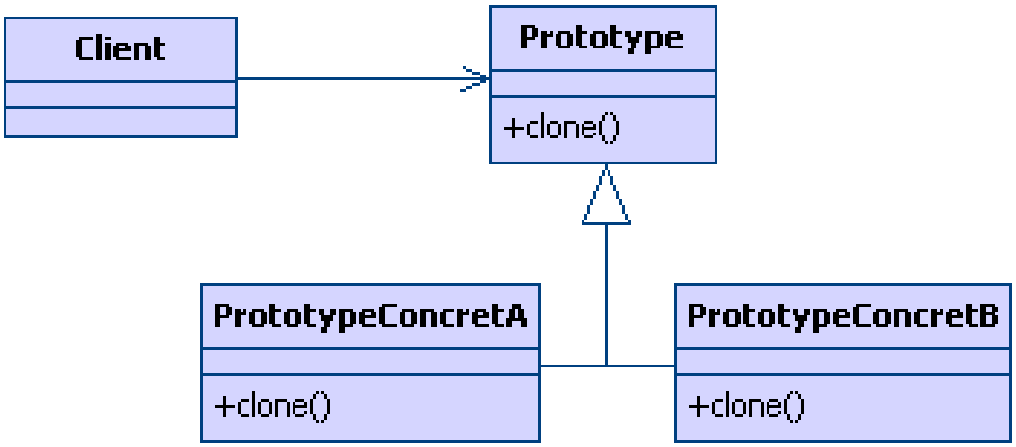
Fabrique Abstraite (Abstract Factory)

Intention : Fournit une interface, pour créer des familles d'objets apparentés ou dépendants sans avoir à spécifier leurs classes concrètes



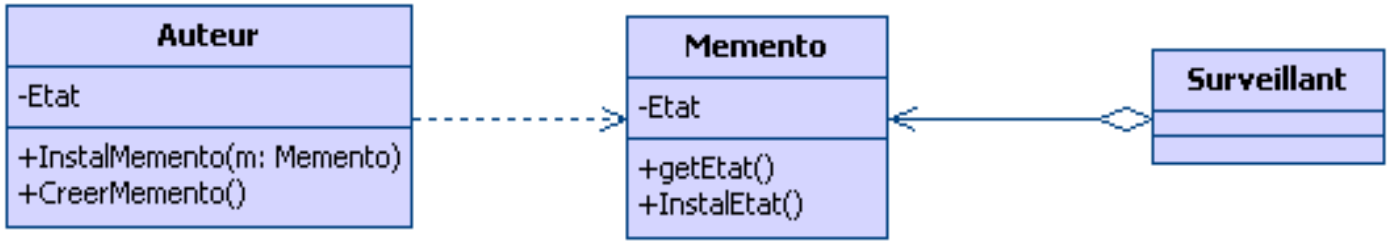
Prototype (Prototype)

Intention : Spécifie les espèces d'objets à créer, en utilisant une instance de type prototype, et crée de nouveau objets par copies de ce prototype.



Mémento (Memento)

Intention : Sans violer l'encapsulation, acquiert et délivre à l'extérieur une information sur l'état interne d'un objet, afin que celui-ci puisse être rétabli ultérieurement dans cet état.

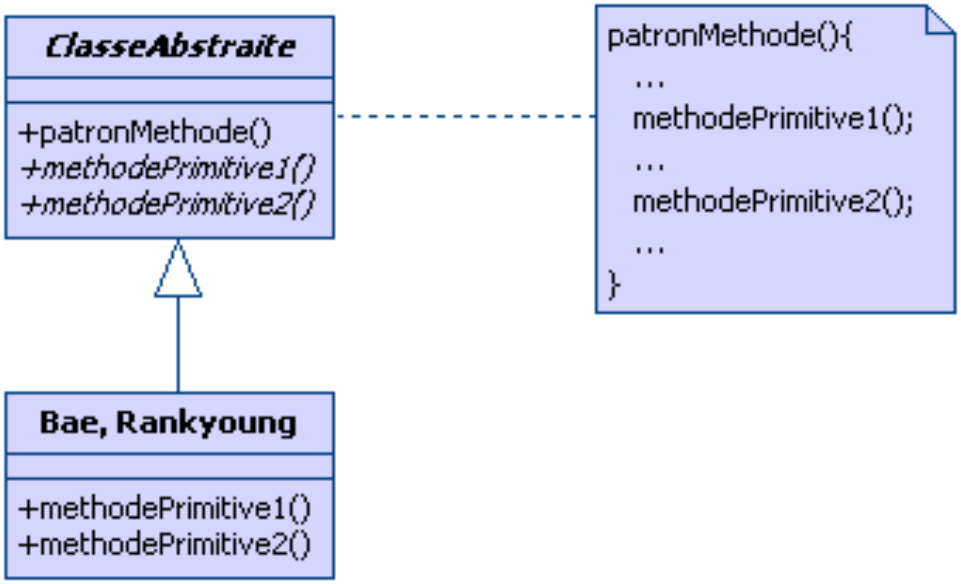


Classement par objectifs

Interfaces	Responsabilité	Construction	Opérations	Extensions
Adaptateur	Singleton	Monteur	Patron méthode	Décorateur
Façade	Observateur	Fabrication	Etat	Itérateur
Composite	Médiateur	Fabrique abstraite	Stratégie	Visiteur
Pont	Procuration	Prototype	Commande	
	Chaine de responsabilité	Mémento	Interprète	
	Poids Mouche			

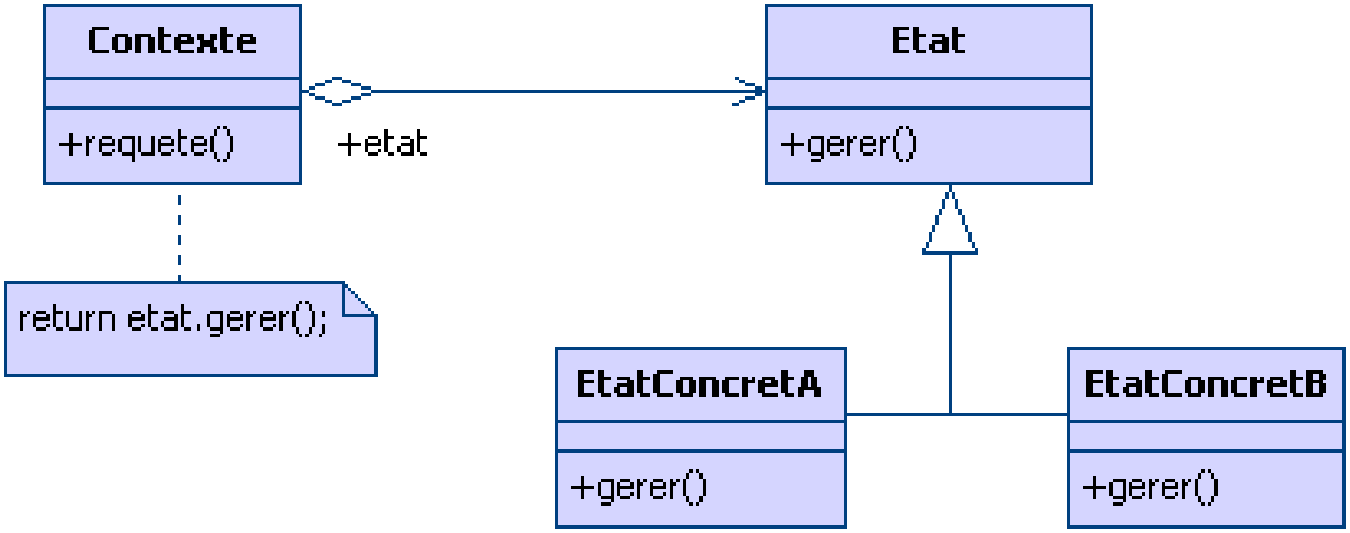
Patron méthode (Template Methode)

Intention : Définit le squelette de l'algorithme d'une opération, en déléguant le traitement de certaines étapes à des sous-classes. Le patron de méthode permet aux sous-classes de redéfinir certaines étapes d'un algorithme sans modifier la structure de l'algorithme.



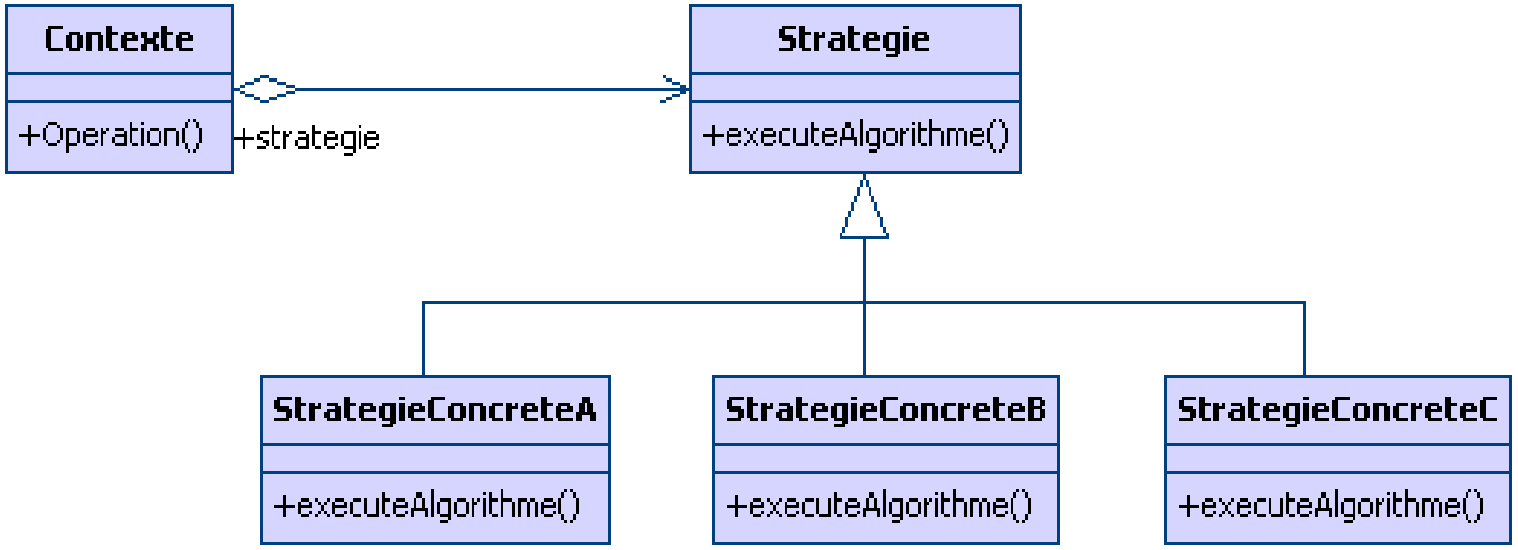
Etat (State)

Intention : permet à un objet de modifier son comportement lorsque son état interne change. L'objet paraîtra changer de classe.



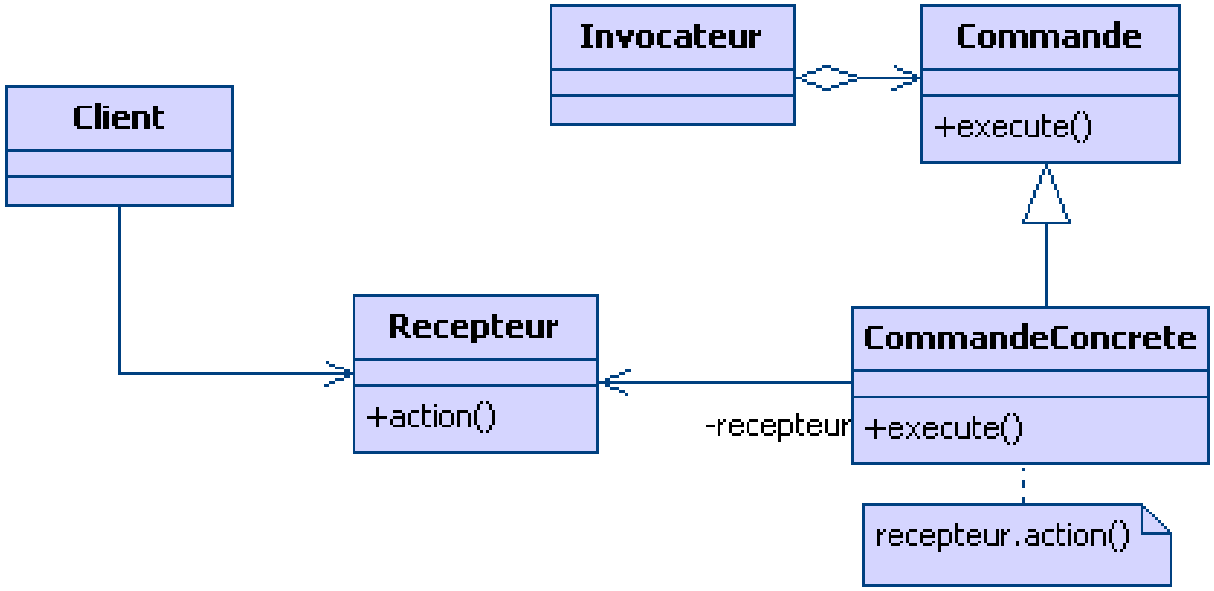
Stratégie (Strategy)

Intention : Définit une famille d'algorithmes, encapsule chacun d'entre eux, et les rend interchangeables. Une stratégie permet de modifier un algorithme indépendamment de ses clients.



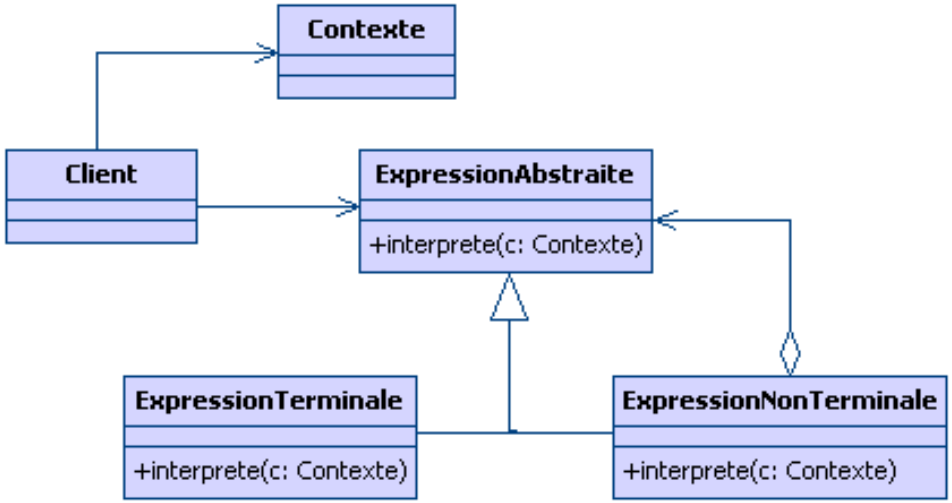
Commande (Command)

Intention : Encapsule une requête comme un objet, ce qui permet de faire un paramétrage des clients avec différentes requêtes, file d'attente, ou historique de requête et d'assurer le traitement des opérations réversible.



Interprète (Interpreter)

Intention : Dans un langage donné, il définit une représentation de sa grammaire, ainsi d'un interprète qui utilise cette représentation pour analyser la syntaxe du langage.

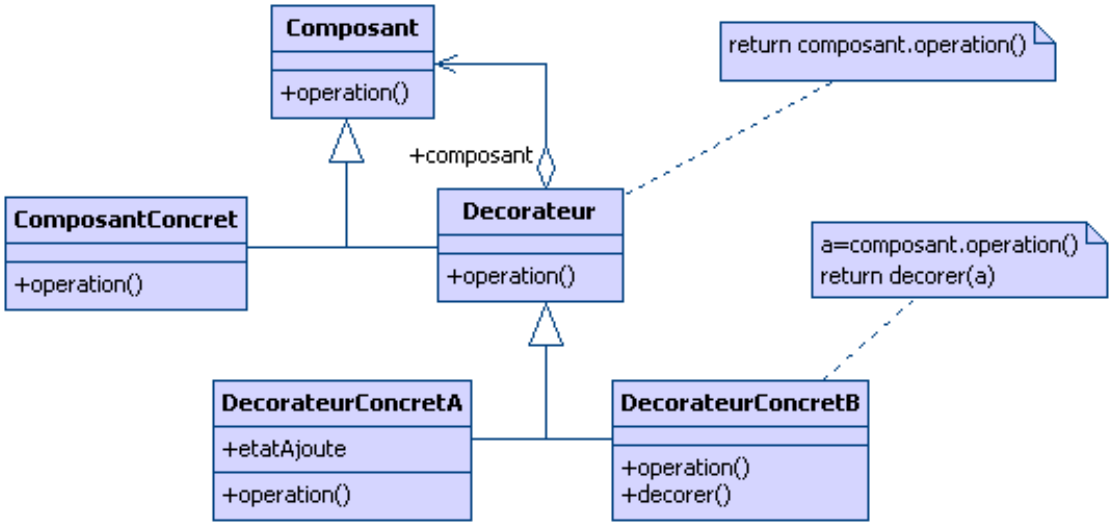


Classement par objectifs

Interfaces	Responsabilité	Construction	Opérations	Extensions
Adaptateur	Singleton	Monteur	Patron méthode	Décorateur
Façade	Observateur	Fabrication	Etat	Itérateur
Composite	Médiateur	Fabrique abstraite	Stratégie	Visiteur
Pont	Procuration	Prototype	Commande	
	Chaine de responsabilité	Mémento	Interprète	
	Poids Mouche			

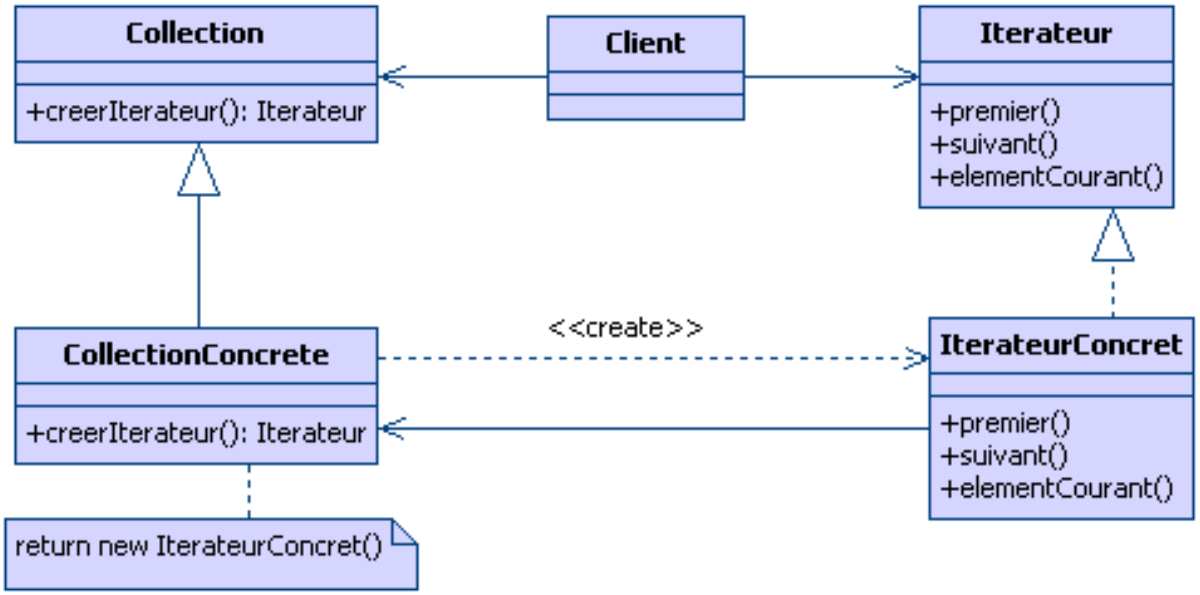
Décorateur (Decorator)

Intention : Attache des responsabilités supplémentaires à un objet de façon dynamique. Il permet une solution alternative pratique pour l'extension de fonctionnalités, à celle de dérivation de classes



Itérateur (Iterator)

Intention : Fournit un moyen pour accéder en séquence aux éléments d'un objet de type agrégat sans révéler sa représentation sous-jacente.



Visiteur (Visitor)

Intention : Représente une opération à effectuer sur les éléments d'une structure d'objet. Le visiteur permet de définir une nouvelle opération sans modifier les classes des éléments sur lesquels il opère.