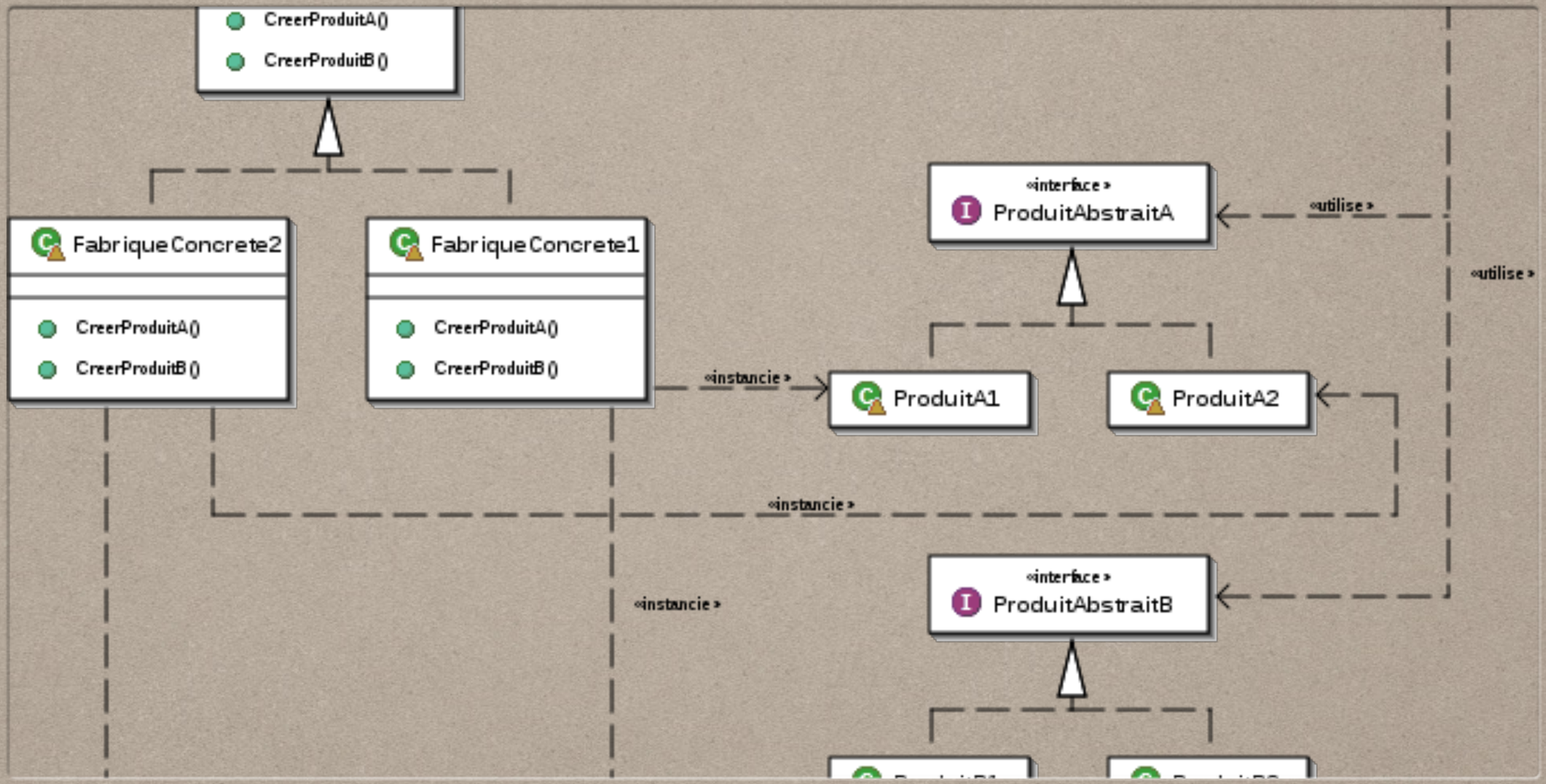


DESIGN PATTERNS

Alexis Ducerf - alexis.ducerf@gmail.com

HISTOIRE

- A Pattern Language - Christopher Alexander en 1977
- Design Patterns - GoF (Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides) en 1995

HISTOIRE

Le GoF apporte 23 patrons de conceptions en 3 grandes familles :

- Patterns de construction : ils définissent comment faire l'instanciation et la configuration des classes et des objets.
- Patterns de structuration : ils définissent comment organiser les classes d'un programme dans une structure plus large (séparant l'interface de l'implémentation).
- Patterns de comportement : ils définissent comment organiser les objets pour que ceux-ci collaborent (distribution des responsabilités) et expliquent le fonctionnement des algorithmes impliqués.

PATTERNS DE CONSTRUCTION

- Pattern Abstract Factory
- Pattern Builder
- Pattern Factory Method
- Pattern Prototype
- Pattern Singleton

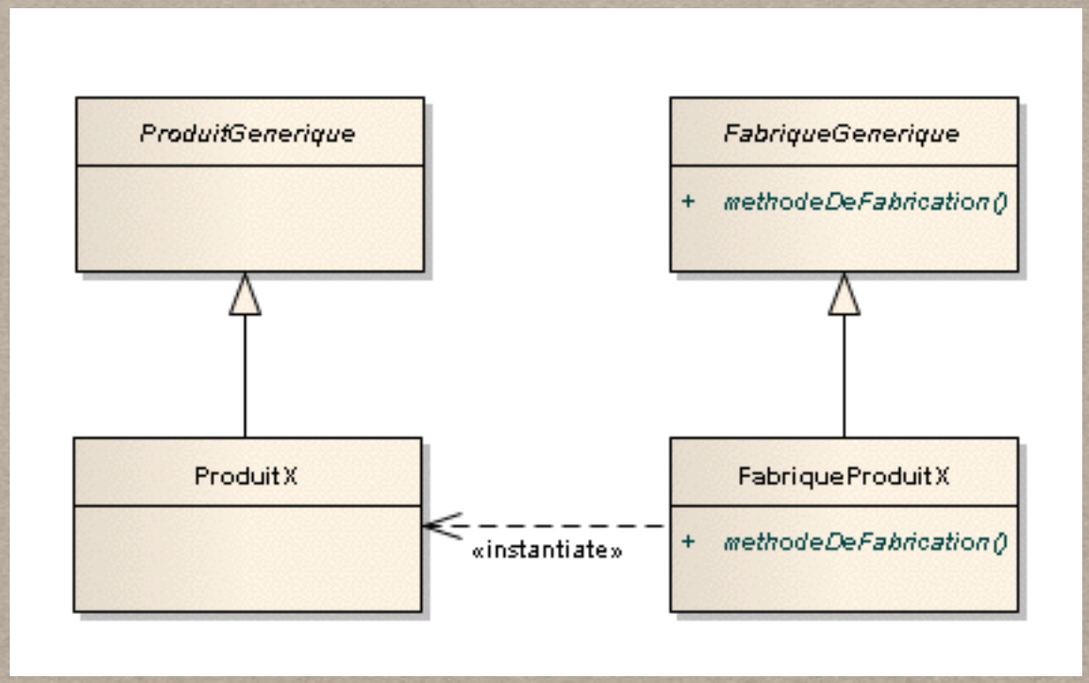
PATTERNS DE STRUCTURATION

- Pattern Adapter
- Pattern Bridge
- Pattern Composite
- Pattern Decorator
- Pattern Facade
- Pattern Flyweight
- Pattern Proxy

PATTERNS DE COMPORTEMENT

- Pattern Chain of Responsibility
- Pattern Command
- Pattern Interpreter
- Pattern Iterator
- Pattern Mediator
- Pattern Memento
- Pattern Observer
- Pattern State
- Pattern Strategy
- Pattern Template
- Pattern Visitor

PATTERN FACTORY METHOD

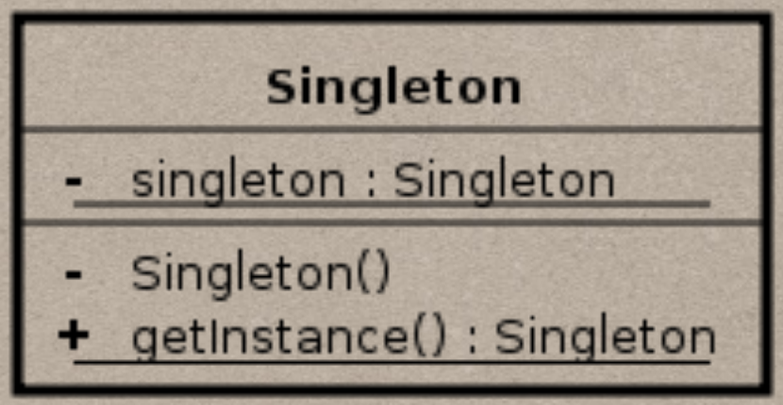


Permet d'instancier des objets dont le type est dérivé d'un type abstrait. La classe exacte de l'objet n'est donc pas connue par l'appelant.

Exercice : Mettre en place une factory pour une connexion à une BDD.

PATTERN SINGLETON

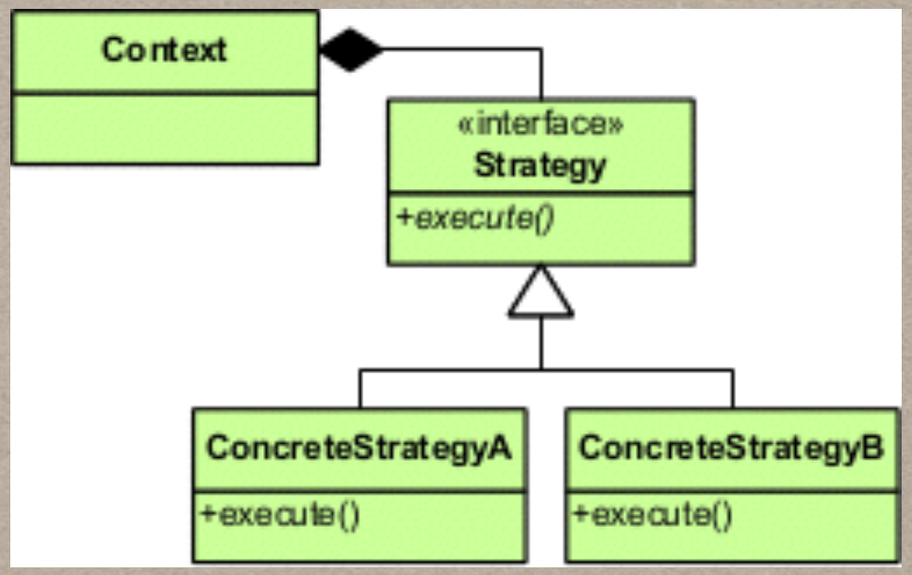
/!\ Antipattern /!



Restreindre l'instanciation d'une classe à un seul objet (ou bien à quelques objets seulement).

Exercice : Mettre en place un singleton pour une connexion à une BDD.

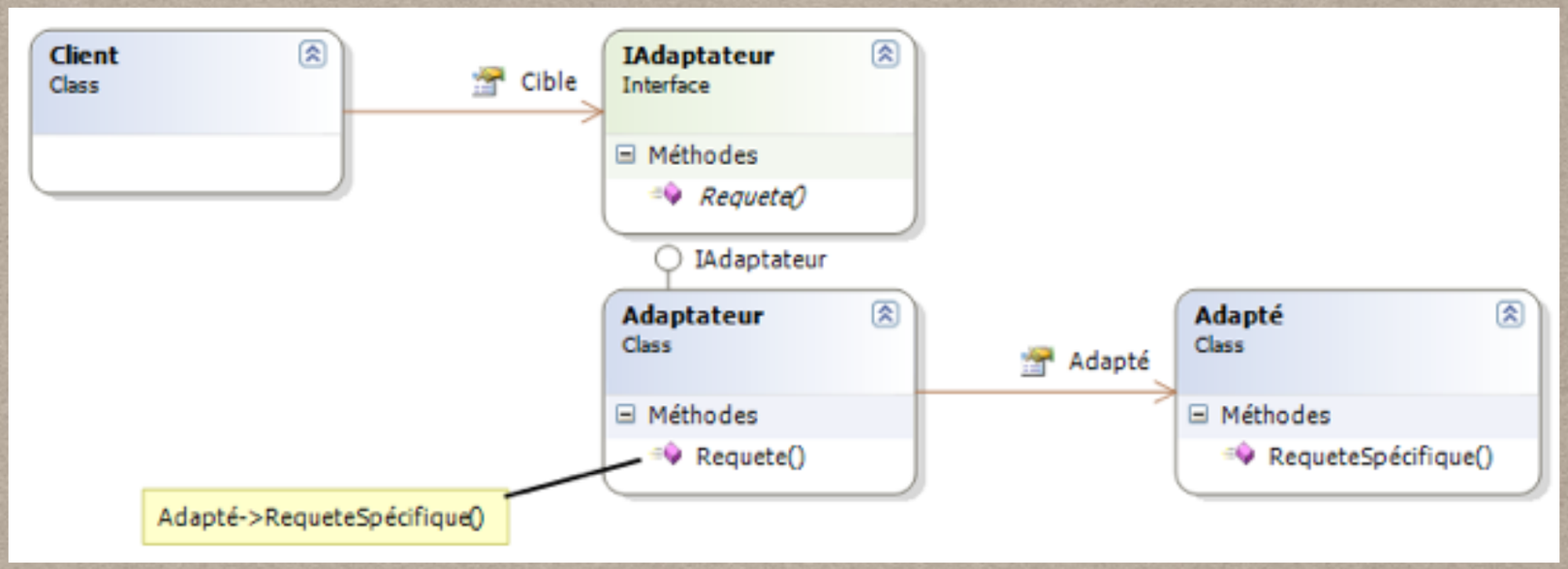
PATTERN STRATEGY



Le patron de conception stratégie est utile pour des situations où il est nécessaire de permuter dynamiquement les algorithmes utilisés dans une application.

Exercice : Mettre en place un système qui permet de fournir un prix HT, avec TVA à 5,5% et à 20%.

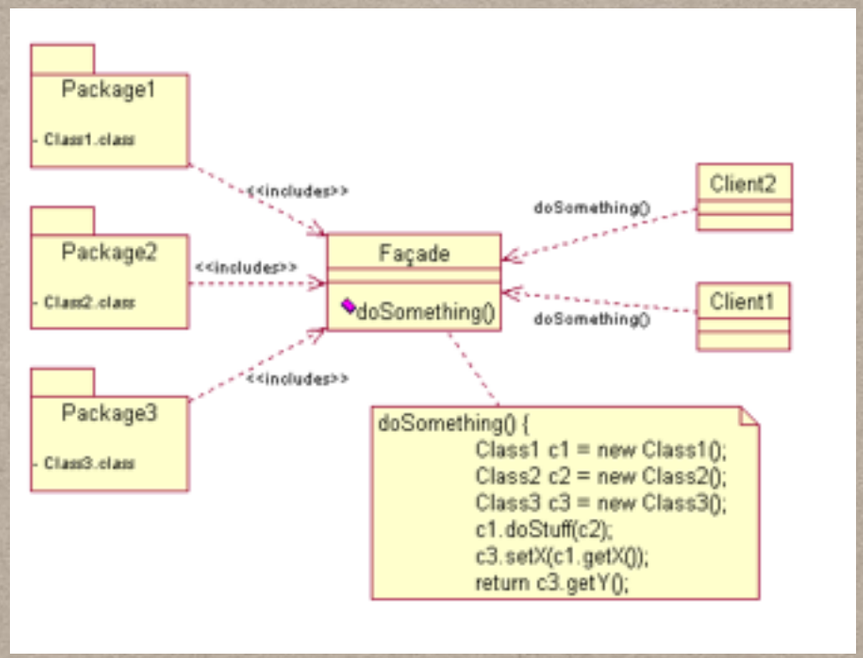
PATTERN ADAPTER



L'Adaptateur fait fonctionner ensemble des classes qui n'auraient pas pu fonctionner sans lui, à cause d'une incompatibilité d'interfaces.

Exercice : l'API d'envoi de mail a changé, comment mettre en place un adaptateur pour ne pas tout changer ?

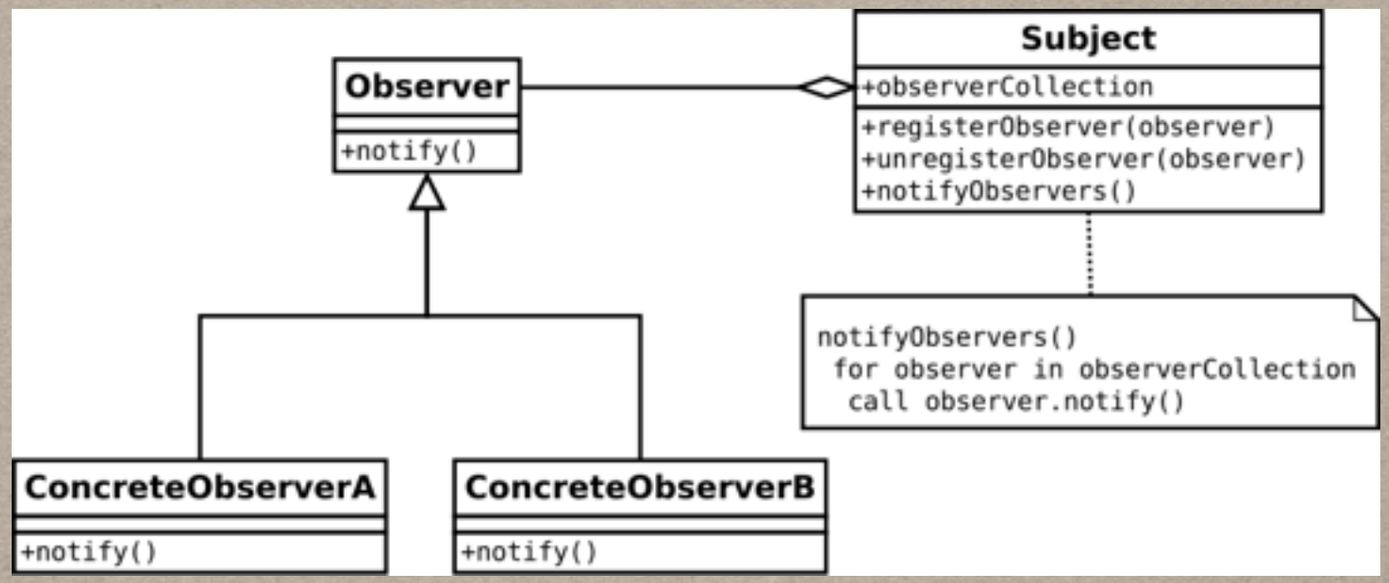
PATTERN FACADE



Le pattern façade a pour but de cacher une conception et une interface complexe difficile à comprendre (cette complexité étant apparue « naturellement » avec l'évolution du sous-système en question).

Exercice : Mettre en place une façade sur une calculatrice.

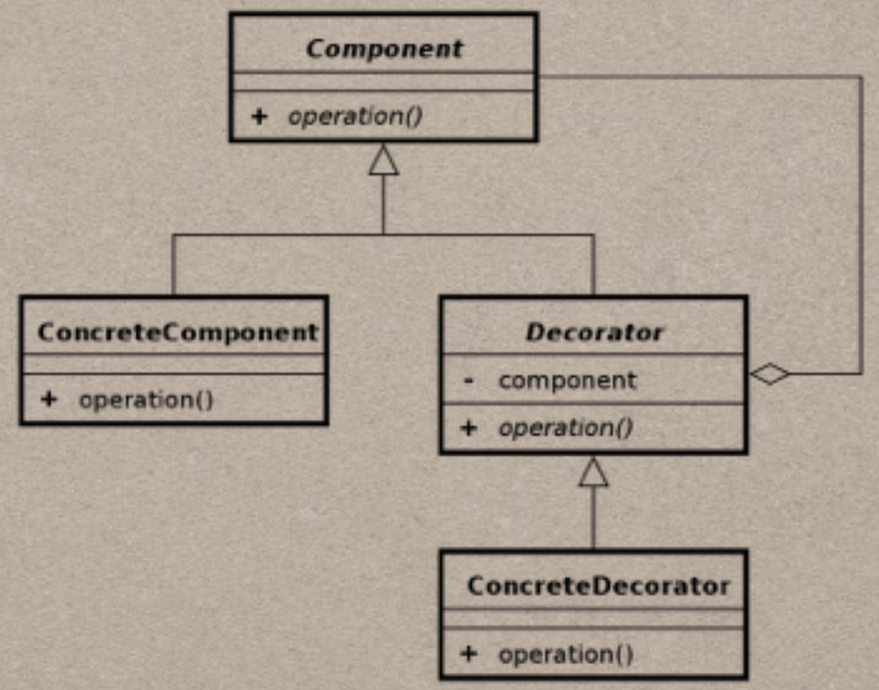
PATTERN OBSERVER



Ce pattern permet d'envoyer un signal à des modules qui jouent le rôle d'observateur. En cas de notification, les observateurs effectuent alors l'action adéquate en fonction des informations qui parviennent depuis les modules qu'ils observent (les « observables »).

Exercice : J'ai un mur sur lequel on attache un post-it . Je peux l'attacher et le détacher. Je veux recevoir un mail à chaque modification et connaitre le contenu du post-it et la personne qui interagit avec.

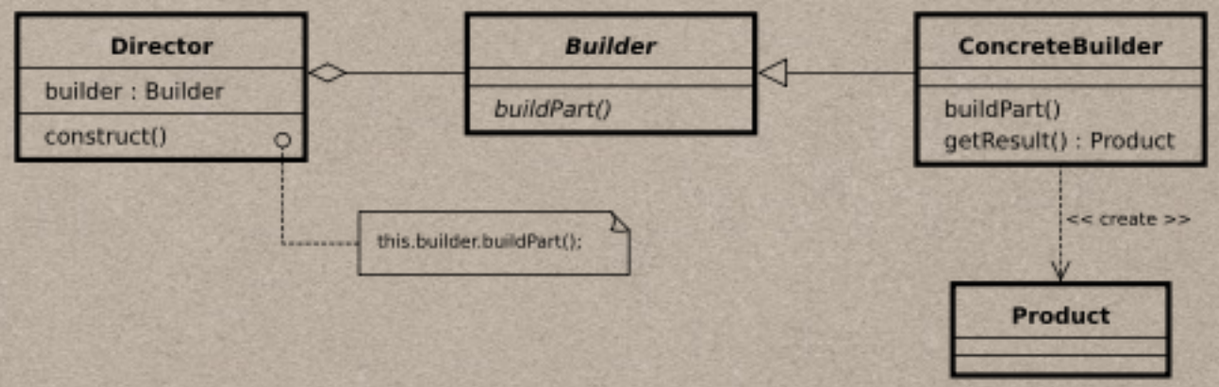
PATTERN DECORATOR



Un décorateur permet d'attacher dynamiquement de nouvelles responsabilités à un objet. Les décorateurs offrent une alternative assez souple à l'héritage pour composer de nouvelles fonctionnalités.

Exercice : Créer le processus d'assemblage d'une voiture.

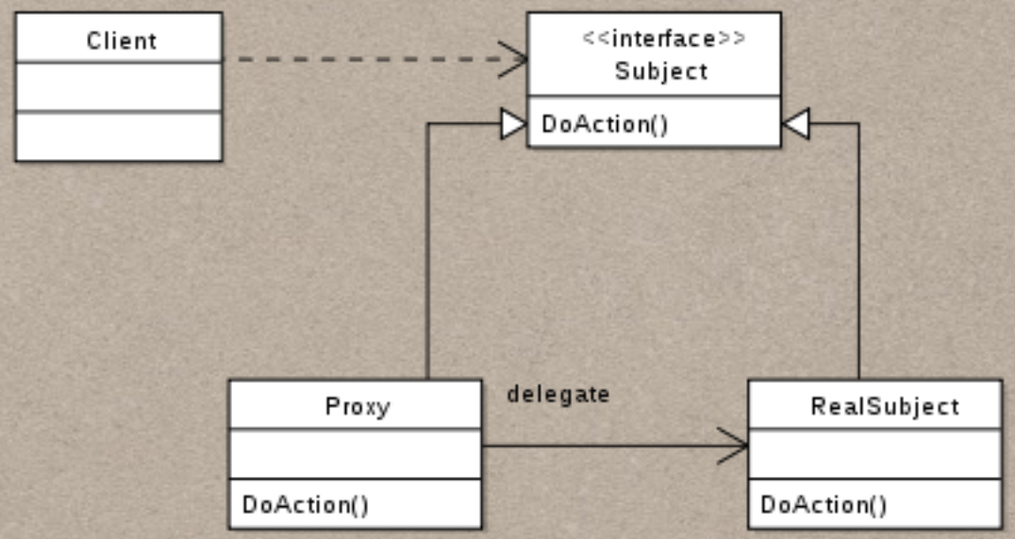
PATTERN BUILDER



Ce pattern est utilisé pour la création d'une variété d'objets complexes à partir d'un objet source. L'objet source peut consister en une variété de parties contribuant individuellement à la création de chaque objet complet grâce à un ensemble d'appels à l'interface commune de la classe abstraite Builder.

Exercice : Créer le processus de création d'une facture de restaurant.

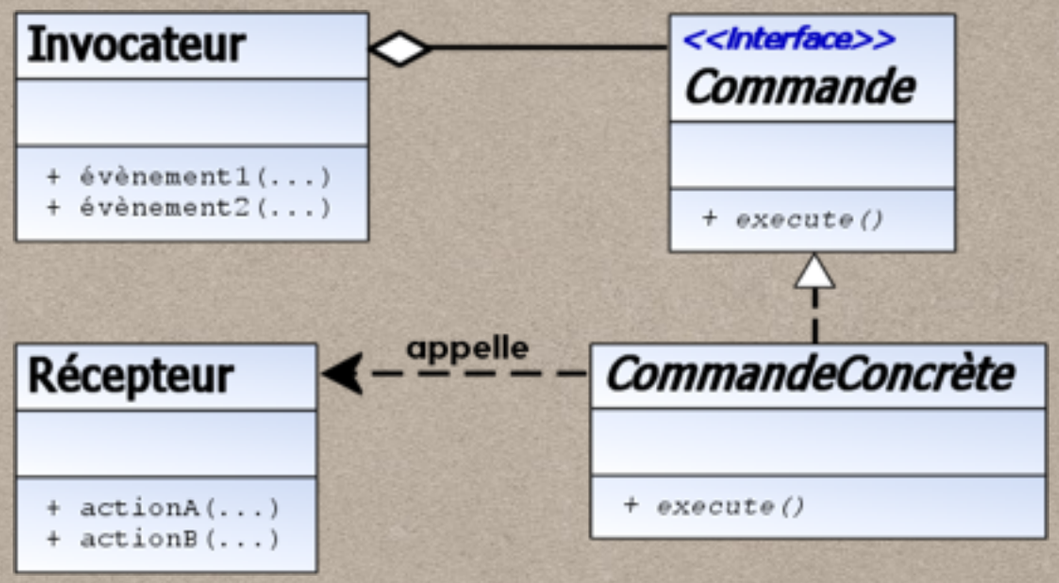
PATTERN PROXY



Un proxy est une classe se substituant à une autre classe. Par convention et simplicité, le proxy implémente la même interface que la classe à laquelle il se substitue. L'utilisation de ce proxy ajoute une indirection à l'utilisation de la classe à substituer.

Exercice : Créer un proxy pour une connexion à une BDD.

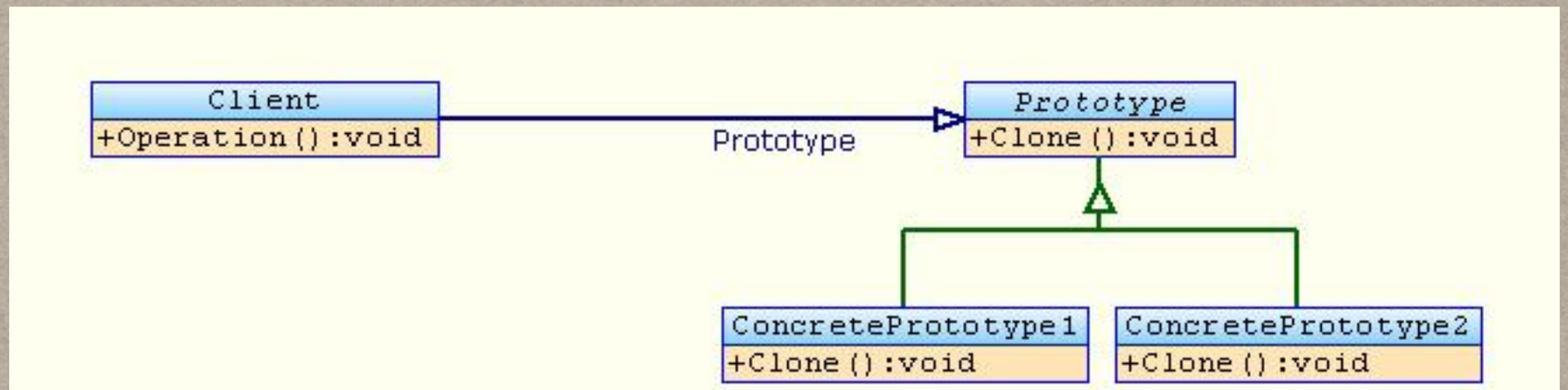
PATTERN COMMAND



Il permet de séparer complètement le code initiateur de l'action, du code de l'action elle-même.

Exercice : Créer un système de queue d'envoi de mail.

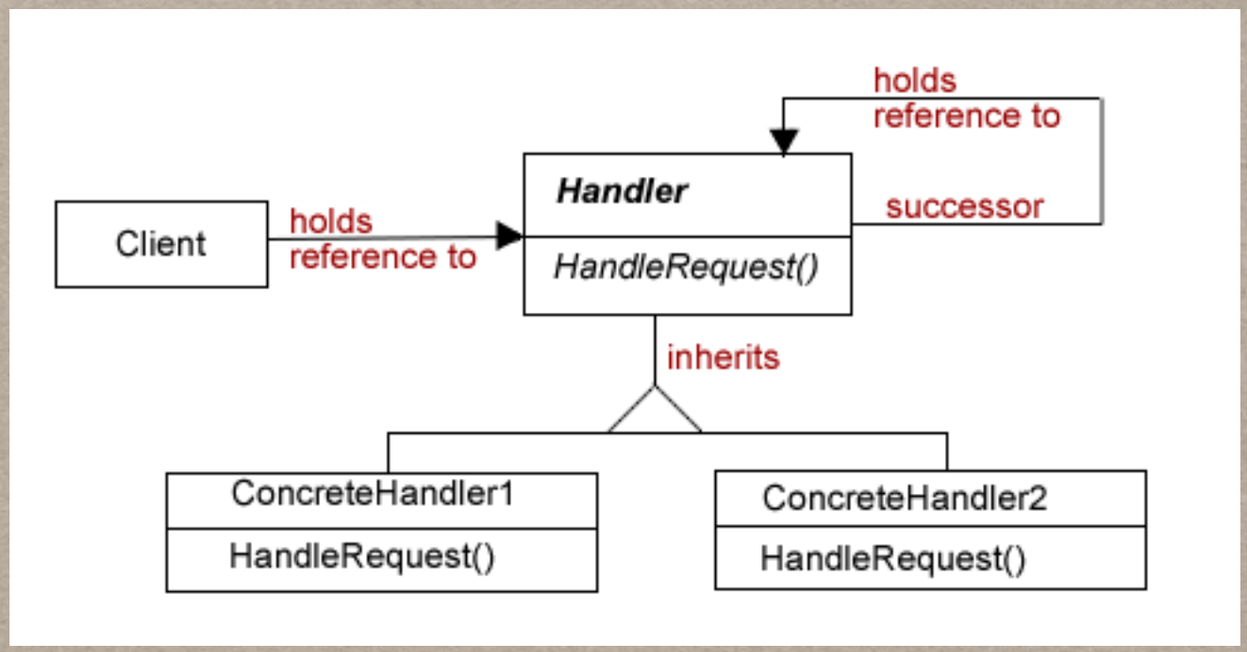
PATTERN PROTOTYPE



Le patron de conception prototype est utilisé lorsque la création d'une instance est complexe ou consommatrice en temps. Plutôt que créer plusieurs instances de la classe, on copie la première instance et on modifie la copie de façon appropriée.

Exercice : Créer plusieurs prototypes de hamburgers à partir d'un hamburger.

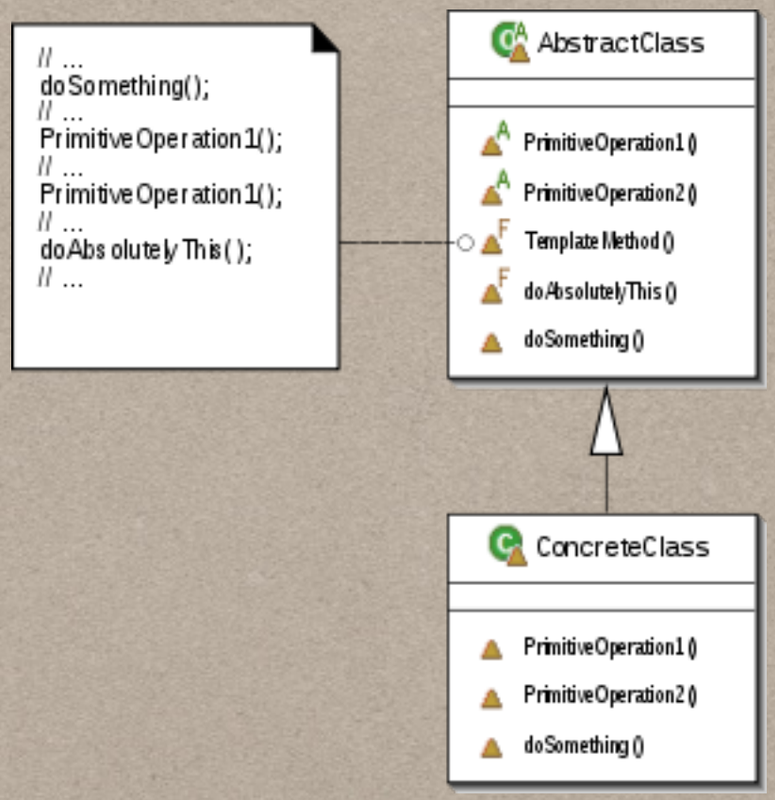
PATTERN CHAIN OF RESPONSIBILITY



Il permet à un nombre quelconque de classes d'essayer de répondre à une requête sans connaître les possibilités des autres classes sur cette requête.

Exercice : Créer une chaine de création d'un hamburger.

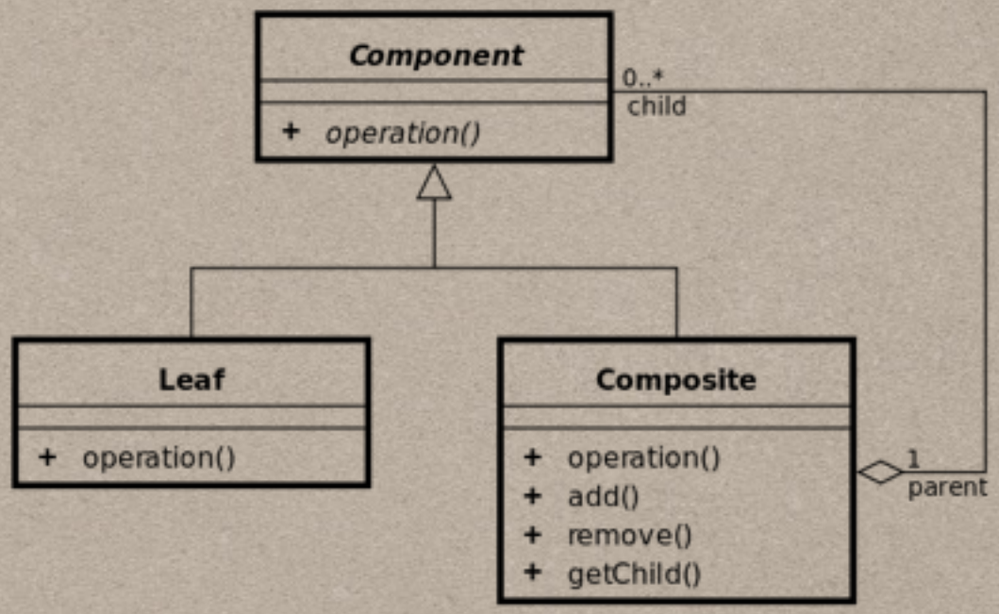
PATTERN TEMPLATE



Le pattern template définit le squelette d'un algorithme à l'aide d'opérations abstraites dont le comportement concret se trouvera dans les sous-classes, qui implémenteront ces opérations.

Exercice : Mettre en place ce pattern sur un jeu de société.

PATTERN COMPOSITE



En programmation objet, 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".

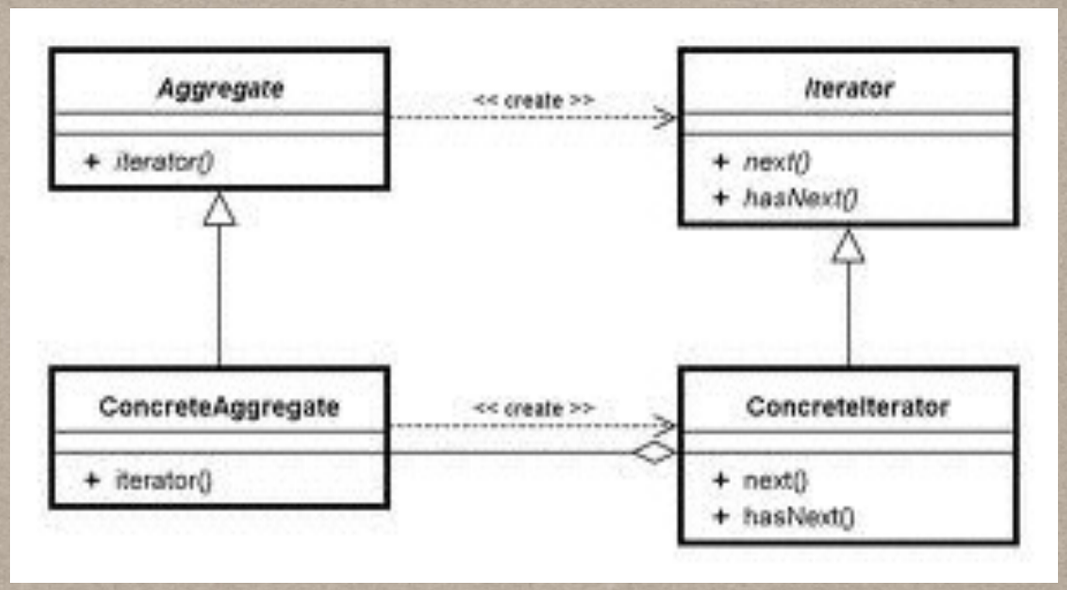
Exercice : Appliquer ce pattern sur un arbre et des feuilles

PATTERN FLYWEIGHT

Lorsque de nombreux (petits) objets doivent être manipulés, mais qu'il serait trop coûteux en mémoire s'il fallait instancier tous ces objets, il est judicieux d'implémenter le poids-mouche.

Exercice : Mettre en place le traitement d’un gros document texte.

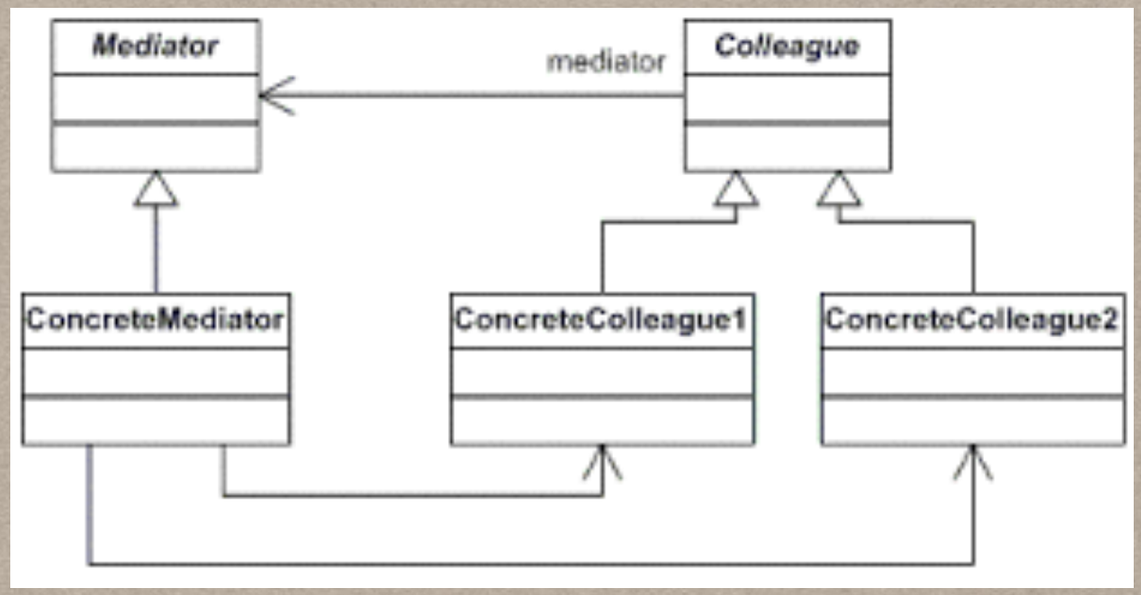
PATTERN ITERATOR



Un itérateur est un objet qui permet de parcourir tous les éléments contenus dans un autre objet, le plus souvent un conteneur (liste, arbre, etc). Un synonyme d'itérateur est curseur, notamment dans le contexte des bases de données.

Exercice : Mettre en place un iterator sur une liste d’élève.

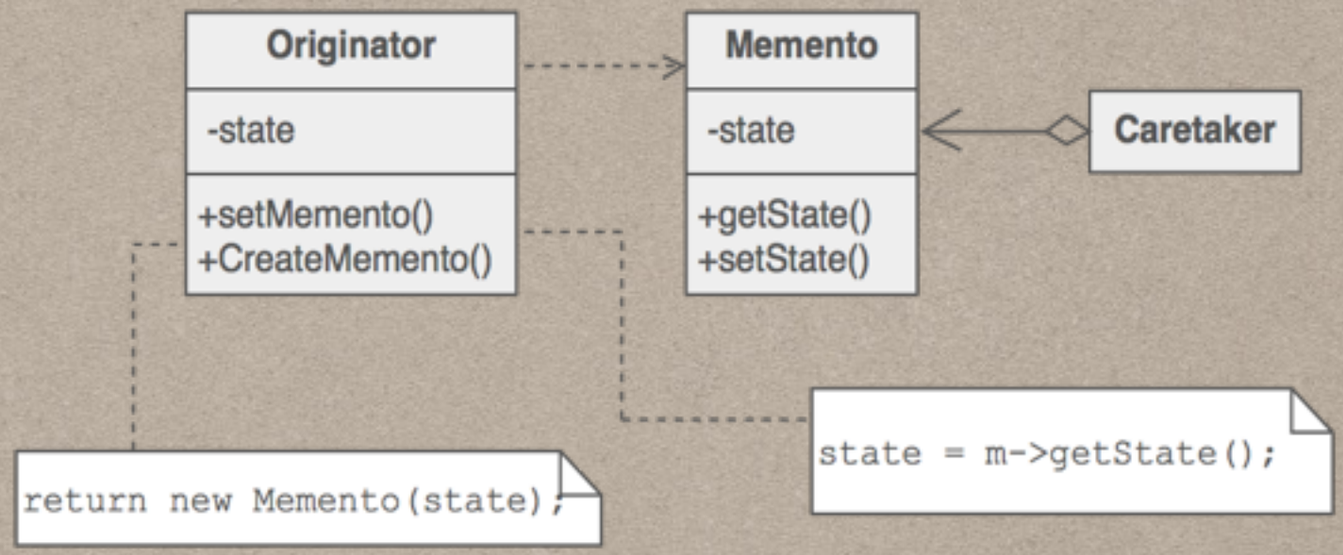
PATTERN MEDIATOR



Médiateur fournit une interface unifiée pour un ensemble d'interfaces d'un sous-système. Il est utilisé pour réduire le couplage entre plusieurs classes.

Exercice : Mettre en place une modification de texte (couleur, casse) via ce pattern.

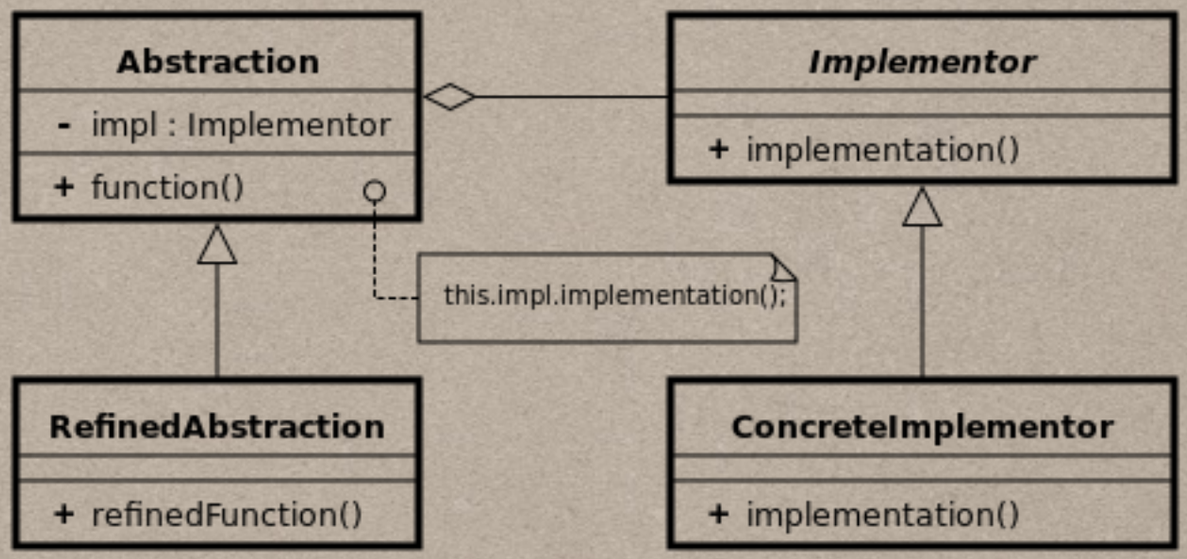
PATTERN MEMENTO



Le patron mémento est un patron de conception logiciel qui permet de restaurer un état précédent d'un objet (retour arrière) sans violer le principe d'encapsulation.

Exercice : Mettre en place une sauvegarde de la minute quand on coupe le lecteur DVD.

PATTERN BRIDGE



Le pattern bridge permet de découpler l'interface d'une classe et son implémentation.

Exercice : Créer une télécommande Sony qui fonctionne avec toutes les sortes de TV.