INF5153 – Génie logiciel : conception

Patrons GoF

Jacques Berger

Objectifs

Présenter les patrons du GoF

Prérequis

GRASP

GoF

Gang of Four

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Auteurs du livre Design Patterns : Elements of Reusable Object-Oriented Software

Singleton

Objectif : S'assurer qu'un objet en particulier ne possède qu'une seule instance dans l'application et fournir un point d'accès global à cet objet

Singleton

Une variable globale rend l'objet accessible mais ne garantit pas qu'il n'y a qu'une seule instance

L'objet doit donc s'assurer lui-même qu'il n'a qu'une instance

Singleton

Le Singleton encapsule son instance, il peut donc contrôler l'accès à l'instance

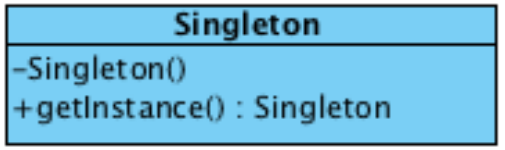
Permet de modifier facilement le nombre d'instances permis

Singleton

Le constructeur du Singleton sera privé pour empêcher l'instantiation

C'est le Singleton qui va se créer lui-même au travers d'une méthode statique

Singleton



Facade

Objectif : Fournir une interface pour un ensemble d'interfaces d'un sous-système. Définition d'une interface de haut niveau qui rend le sous-système plus facile à utiliser

Facade

Une seule interface simplifiée pour faciliter la manipulation d'un ensemble de fonctionnalités

Facade

La Facade délègue les tâches aux objets du sous-système

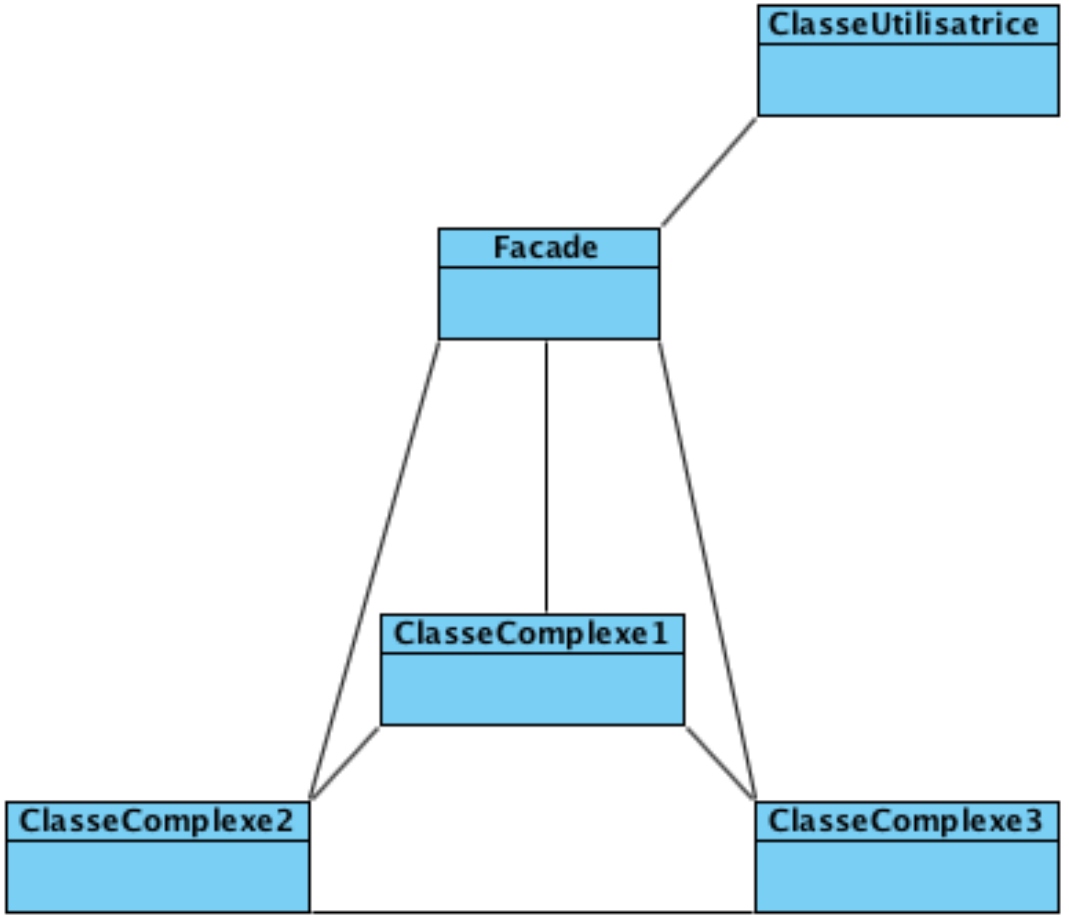
Ces objets n'ont aucune connaissance de la Facade

Facade

Réduit passablement le nombre d'objets que l'appelant doit connaître

Les Facades permettent d'éliminer les dépendances circulaires

Facade



Strategy

Objectif : Définir un ensemble d'algorithmes encapsulés dans des classes interchangeables à l'exécution

Permet de changer d'algorithme en fonction de l'utilisateur

Strategy

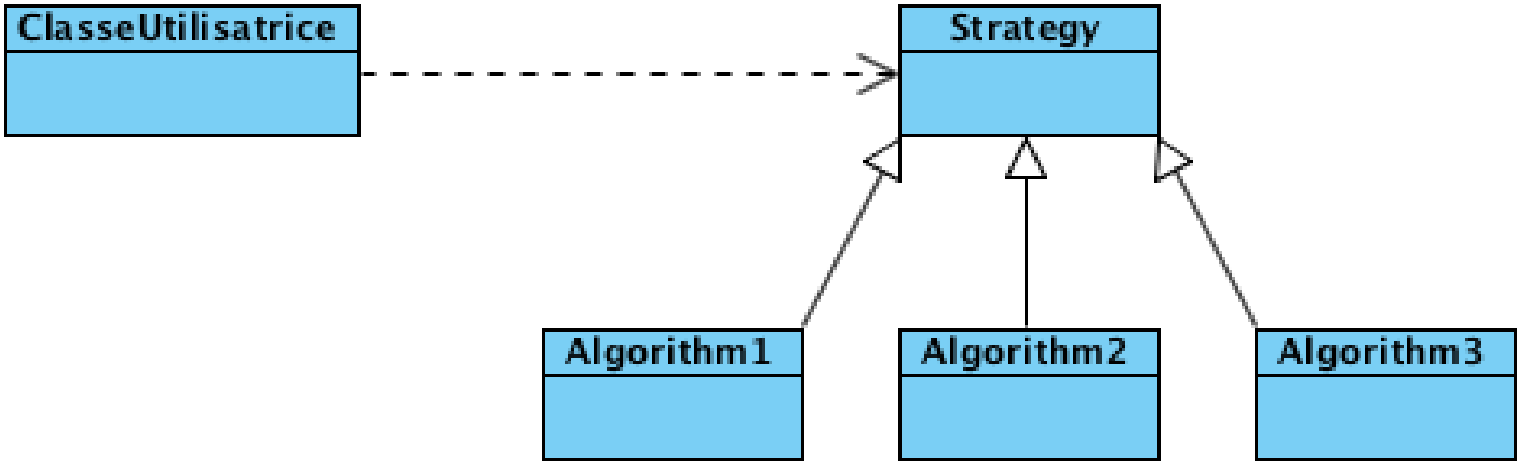
Différents algorithmes vont être utiles à différents moments ou pour différents utilisateurs

Strategy

Utile lorsque nous avons plusieurs variantes d'un même algorithme

Utilise le polymorphisme

Strategy



Abstract Factory

Objectif : Fournir une interface pour créer une famille d'objets sans spécifier leur classe concrète

Abstract Factory

Les appelants n'ont aucune connaissance des classes concrètes utilisées

Une Factory par famille d'objets

Abstract Factory

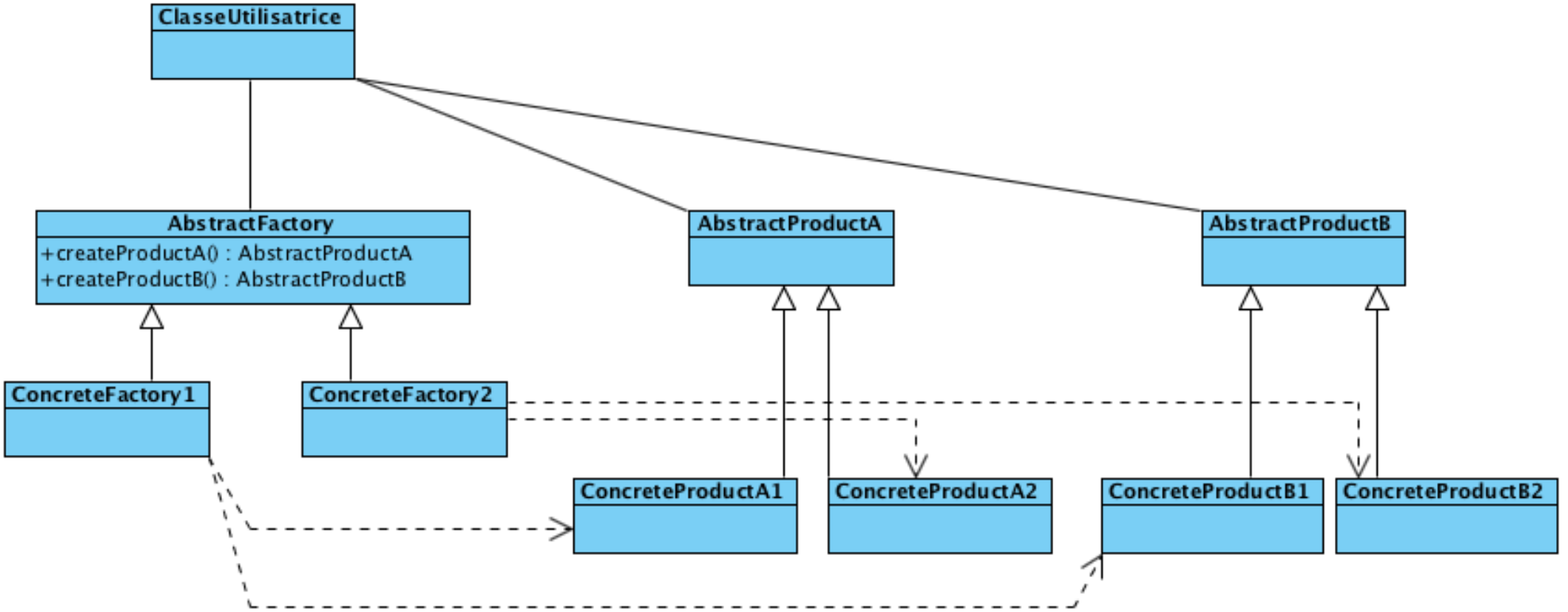
Utile lorsque :

Le système doit être indépendant de la façon dont ses produits sont créés

Le système peut être configuré pour utiliser une famille de produits

Fournir une librairie de produits en ne révélant que des interfaces et non pas les implémentations

Abstract Factory



Factory Method

Objectif : Définir une interface pour créer un objet, mais c'est une sous-classe qui détermine quelle classe instancier

Factory Method

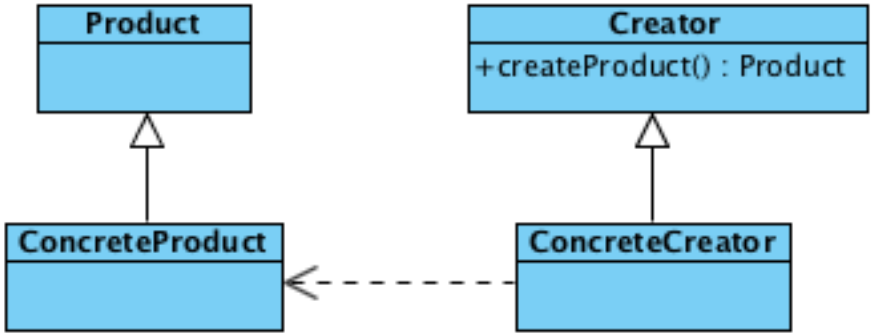
Encapsulation de la logique visant à choisir le bon objet à instancier

Le Factory Method est la fonction qui crée l'objet à retourner

Factory Method

Utile lorsqu'on ne peut anticiper le type d'un objet à créer

Factory Method



Command

Objectif : Encapsuler une opération dans un objet, ce qui permet d'utiliser plusieurs types d'opération, de les gérer dans des files ou d'annuler leur effet

Command

Lorsqu'il est nécessaire de manipuler des requêtes de différents types sans connaître les types en question

Command

La commande peut être appliquée immédiatement sur un objet ou alors placée dans une liste et être appliquée plus tard

Toute l'information nécessaire pour appliquer la commande est stocké dans l'objet

Command

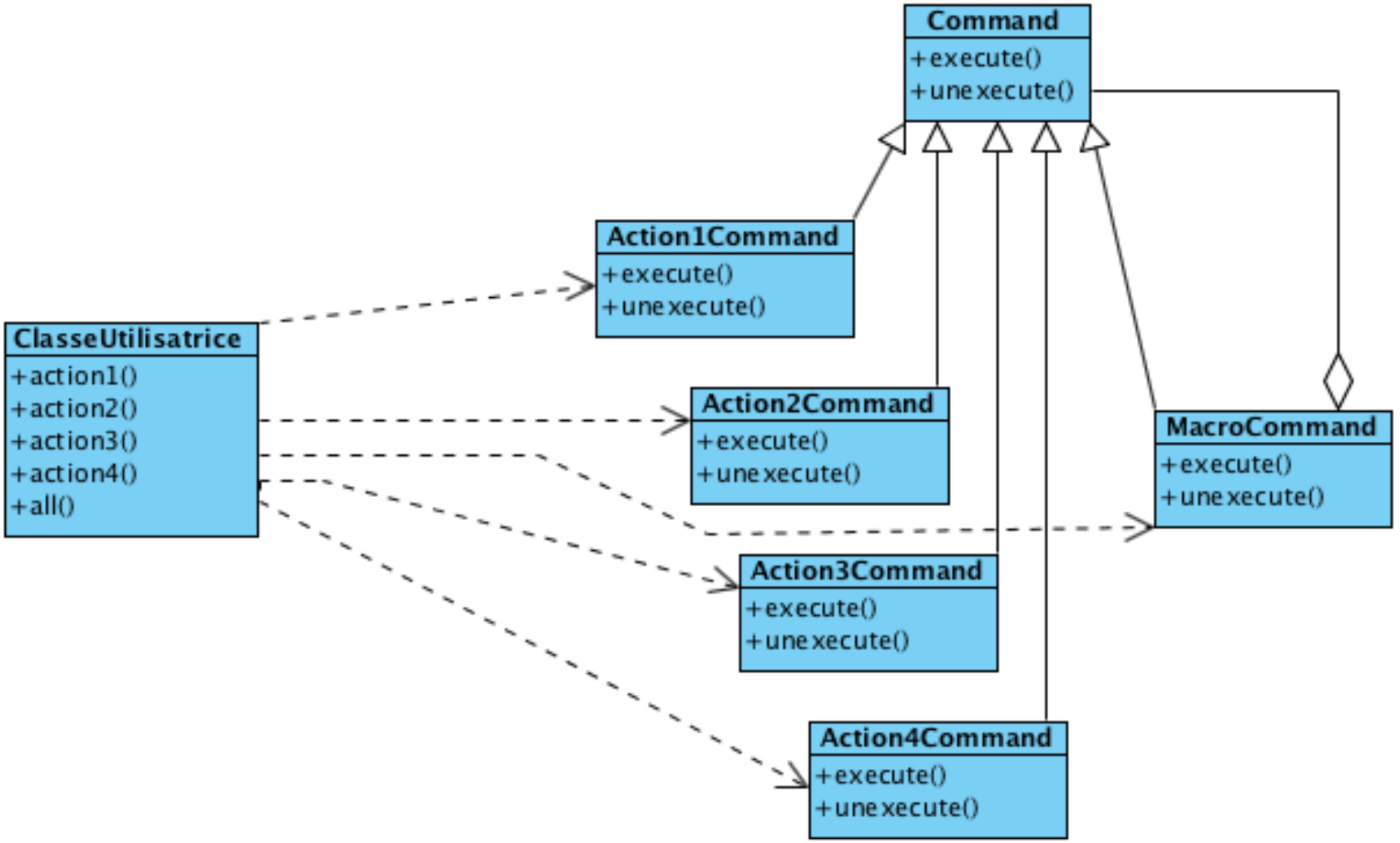
Optionnellement, une commande pourrait être en mesure d'annuler son effet sur l'objet

Dans ce cas, la commande contient l'information nécessaire pour annuler son effet

Command

Un objet MacroCommand est une commande qui contient d'autres commandes et qui les exécutent d'un seul coup

Command



Memento

Objectif : Conserver l'état interne d'un objet dans le but de le restituer plus tard, sans défaire l'encapsulation de sa structure interne

Memento

Permet de conserver un snapshot de l'état d'un objet

Le memento peut être conservé dans le but d'éventuellement remettre l'objet à un état précédent

Memento

L'objectif est de faire ce snapshot sans exposer la structure interne de l'objet

C'est l'objet qui est responsable de fournir son Memento

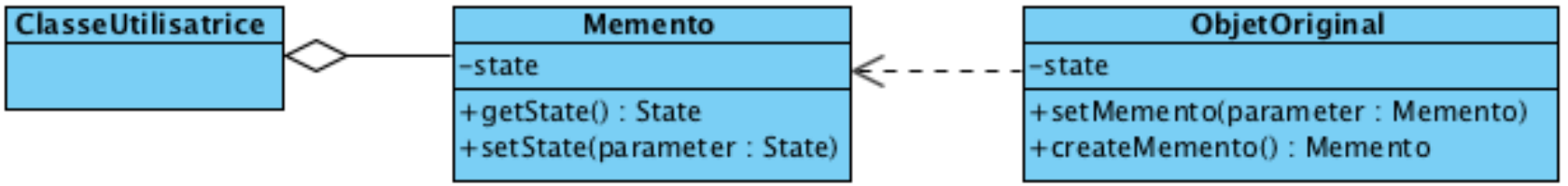
Memento

Le memento expose 2 interfaces :

Une pour l'objet qui le crée

Une pour les objets qui le manipulent

Memento



Observer

Objectif : Dans une relation un-à-plusieurs, si un objet est modifié, les objets qui en dépendent sont avisés du changement

Observer

Utile lorsqu'il est nécessaire de maintenir une cohérence entre plusieurs objets

Ex. : Redessiner un élément graphique lorsque l'objet qu'il représente change

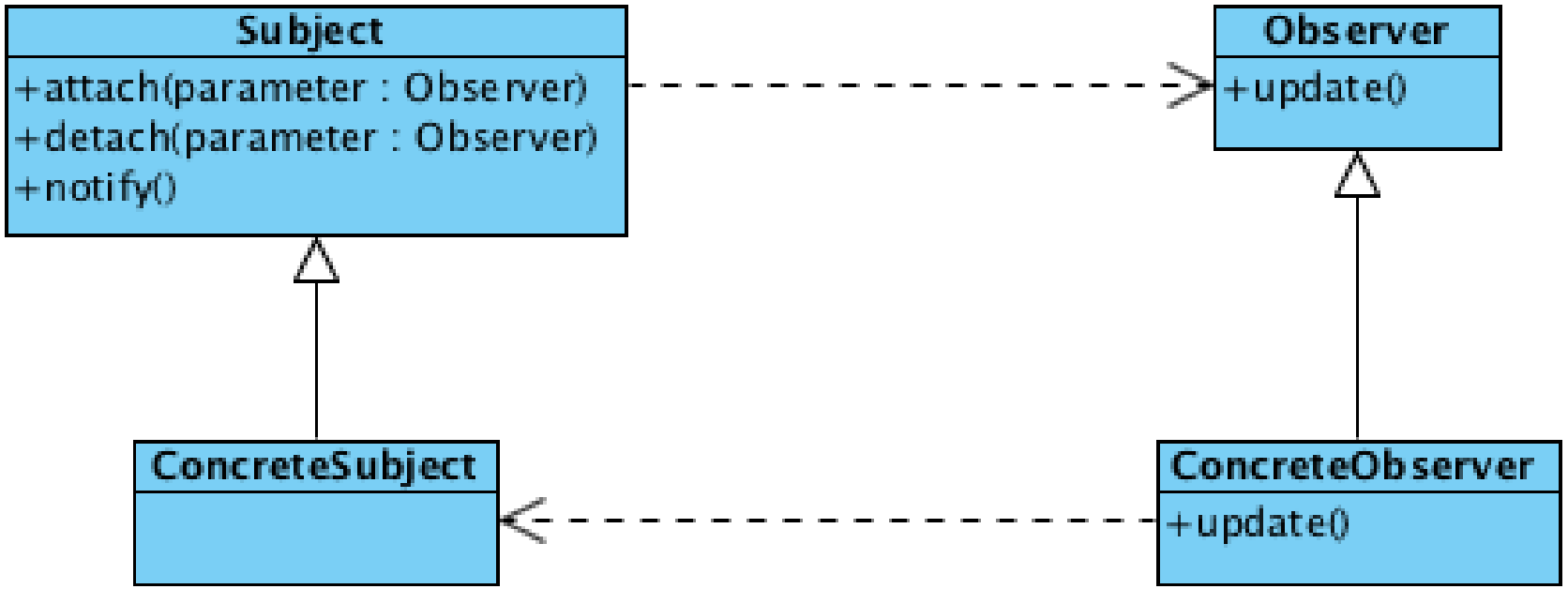
Observer

Permet de découpler les objets de sorte que l'objet observé n'a pas besoin de connaître le type spécifique des objets qui l'observe

Observer

Si un objet observe plusieurs autres objets, il peut être utile de lui passer l'objet observé en paramètre lorsqu'il change

Observer



Chain of Responsibility

Objectif : Éviter le couplage entre un objet qui émet une requête et l'objet qui y répondra

Permet à plusieurs objets de répondre à la requête en traversant une chaîne d'objets jusqu'à ce que l'un d'entre eux puisse répondre à la requête

Chain of Responsibility

L'objet qui répond à la requête n'est pas nécessairement connu par l'objet qui émet la requête

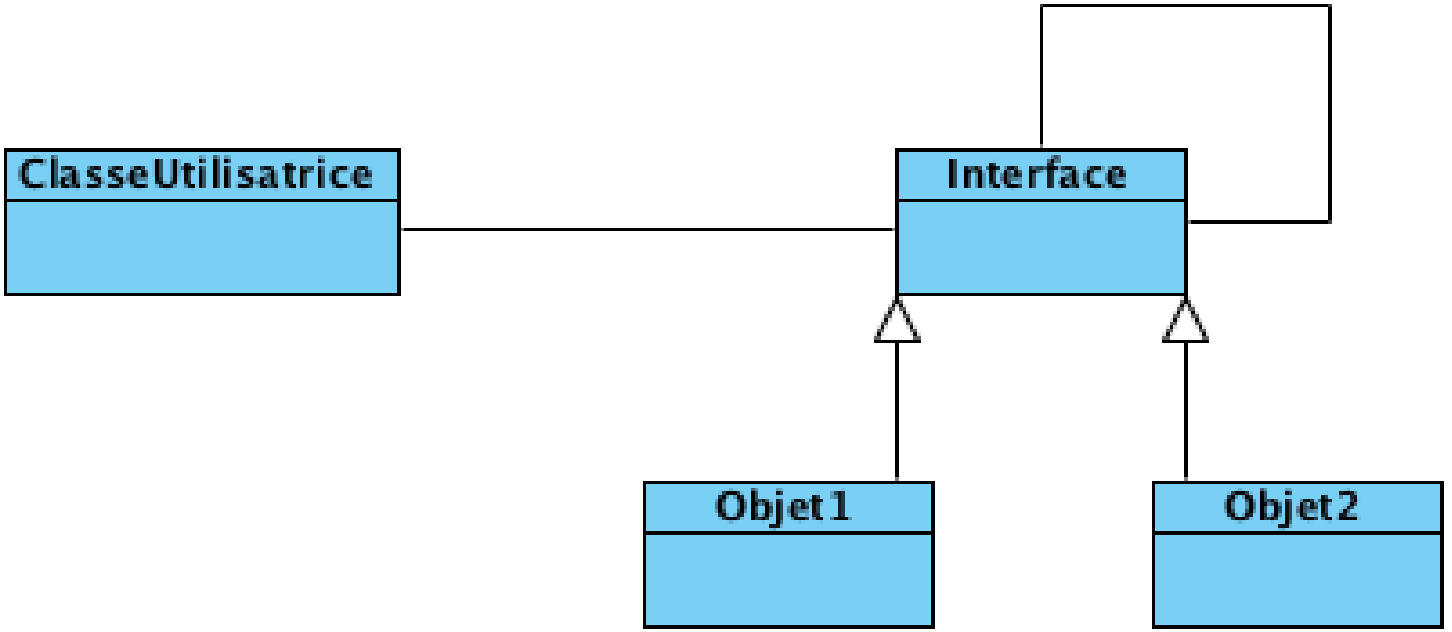
Chain of Responsibility

Le premier objet à recevoir la requête va la passer à un autre objet s'il ne peut pas y répondre lui-même

Chain of Responsibility

Tous les objets de la chaîne implémentent la même interface et peuvent détenir une référence vers le prochain maillon de la chaîne

Chain of Responsibility



Builder

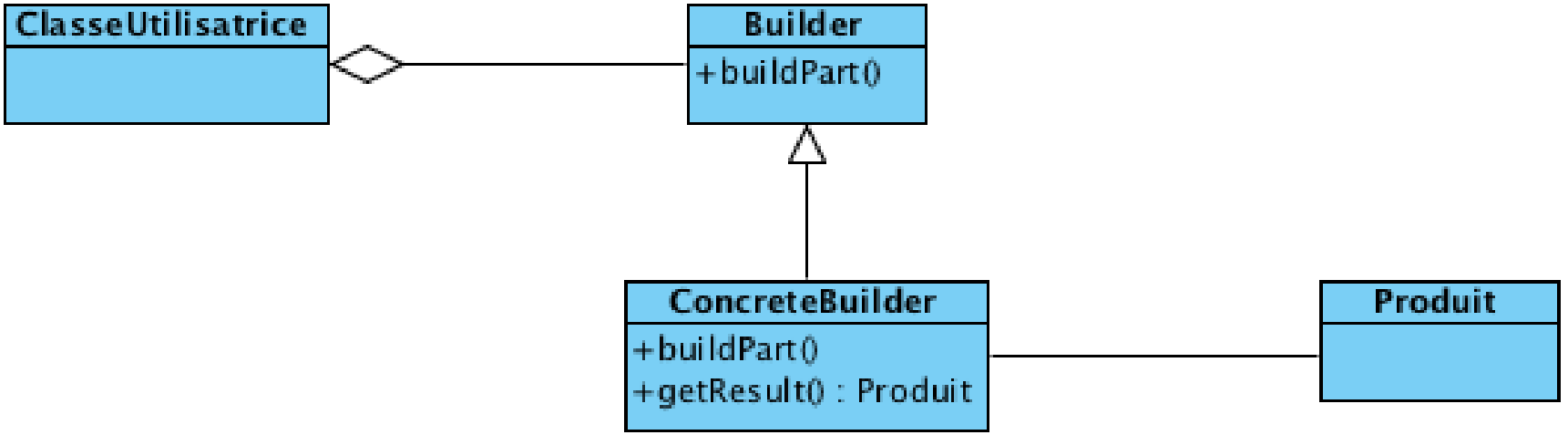
Objectif : Séparer la construction d'un objet complexe de sa représentation

Le processus de construction peut être réutilisé pour différentes instances

Builder

Un autre objet, uniquement conçu à cet effet, est responsable de créer une instance de l'objet

Builder



Decorator

Objectif : Ajouter des fonctionnalités à un objet dynamiquement

Decorator

Utile lorsqu'on veut ajouter une fonctionnalité à certaines instances d'un objet mais pas à l'objet en tant que tel

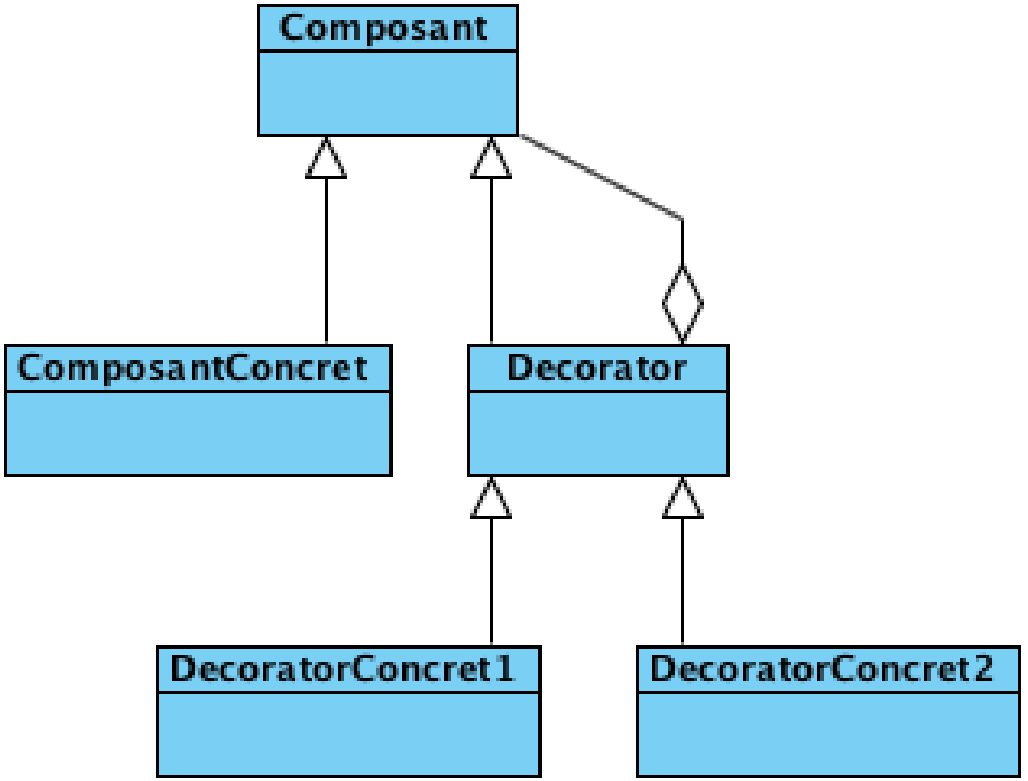
Decorator

L'objet Decorator se fait passer pour l'objet décoré, mais il ne l'est pas vraiment

Il conserve une référence sur l'objet décoré pour l'invoquer au besoin

La présence du Decorator n'est pas connu de l'utilisateur de l'interface

Decorator



State

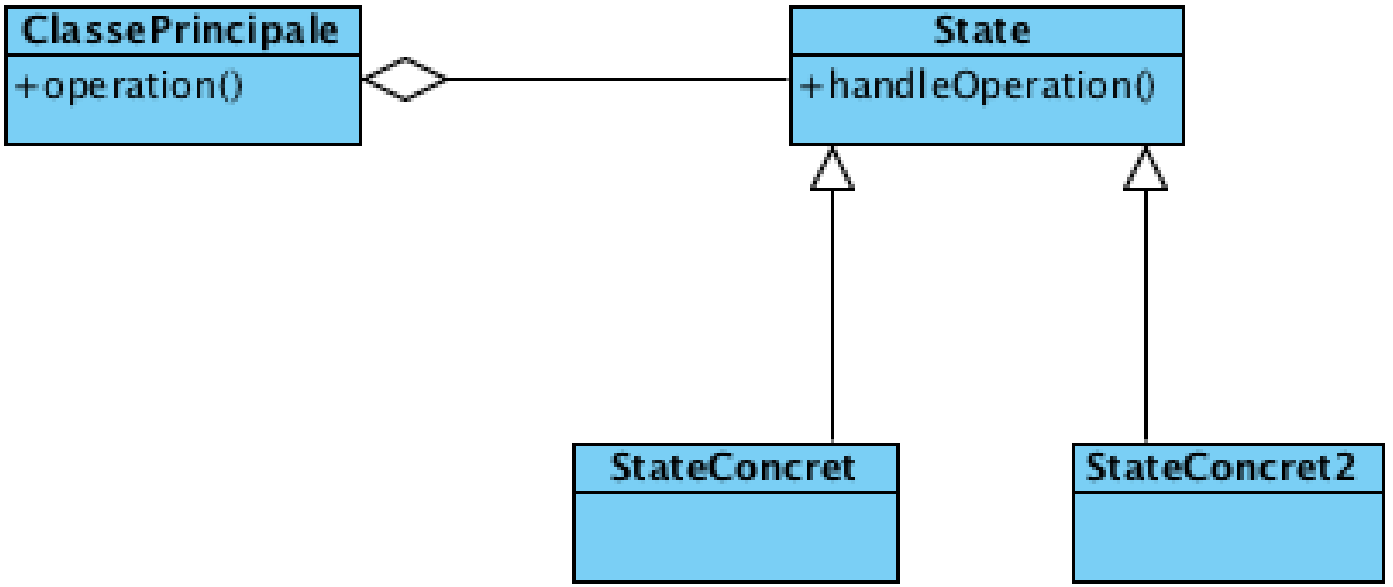
Objectif : L'objet peut changer de comportement lorsque sa structure interne change

State

L'objet possède un lien avec un objet interne qui peut changer d'état

Chaque état possible est une classe concrète implémentant une même interface

State



Prototype

Objectif : Créer un prototype qui servira de référence pour créer d'autres objets à partir de celui-ci

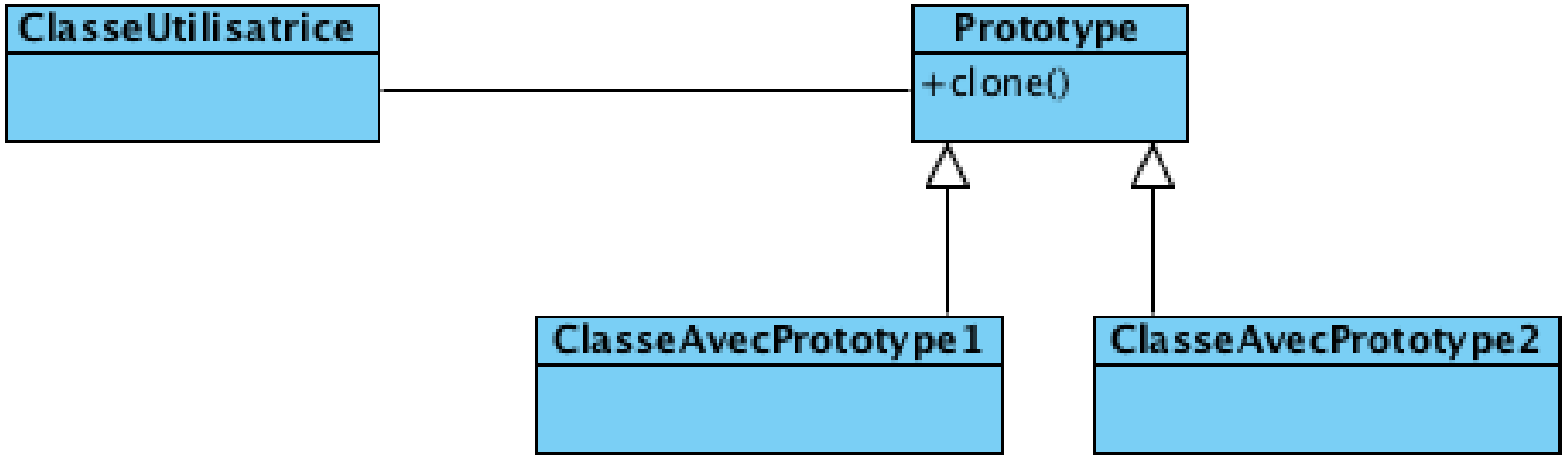
Prototype

À utiliser lorsqu'on veut instancier plusieurs objets avec uniquement quelques différences mineures

Prototype

On crée un prototype et on demande un clone du prototype pour chaque instance désirée

Prototype



Bridge

Objectif : Séparer l'interface et l'implémentation pour que les deux puissent évoluer indépendamment

Bridge

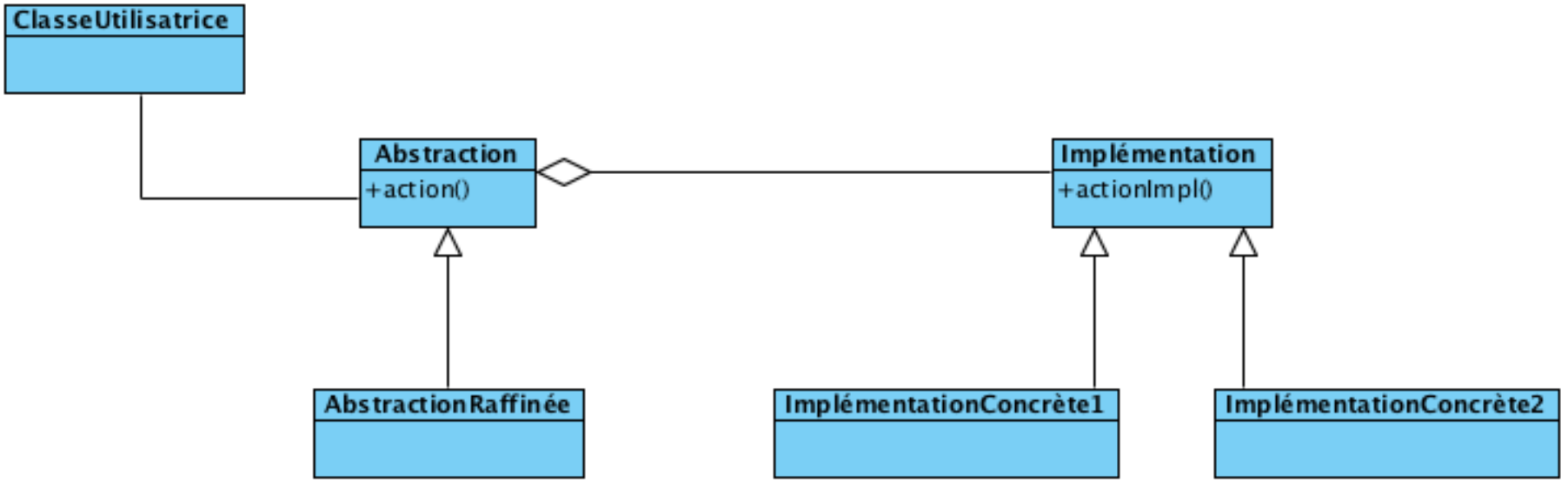
Pour éviter d'avoir une relation forte entre l'abstraction et l'implémentation

Également utile si l'implémentation doit pouvoir être changée durant l'exécution

Bridge

L'implémentation et l'interface sont tous les deux extensibles au travers de l'héritage

Bridge



Proxy

Objectif : Fournir un substitut qui gère l'accès ou l'utilisation d'une autre classe

Proxy

Applicable lorsqu'un point d'accès plus complexe qu'un pointeur ou une référence est nécessaire

Proxy

Remote proxy : Représentation locale pour un objet distant

Virtual proxy : Création d'objets lourds à la demande

Protection proxy : Contrôle l'accès à l'objet original

Proxy

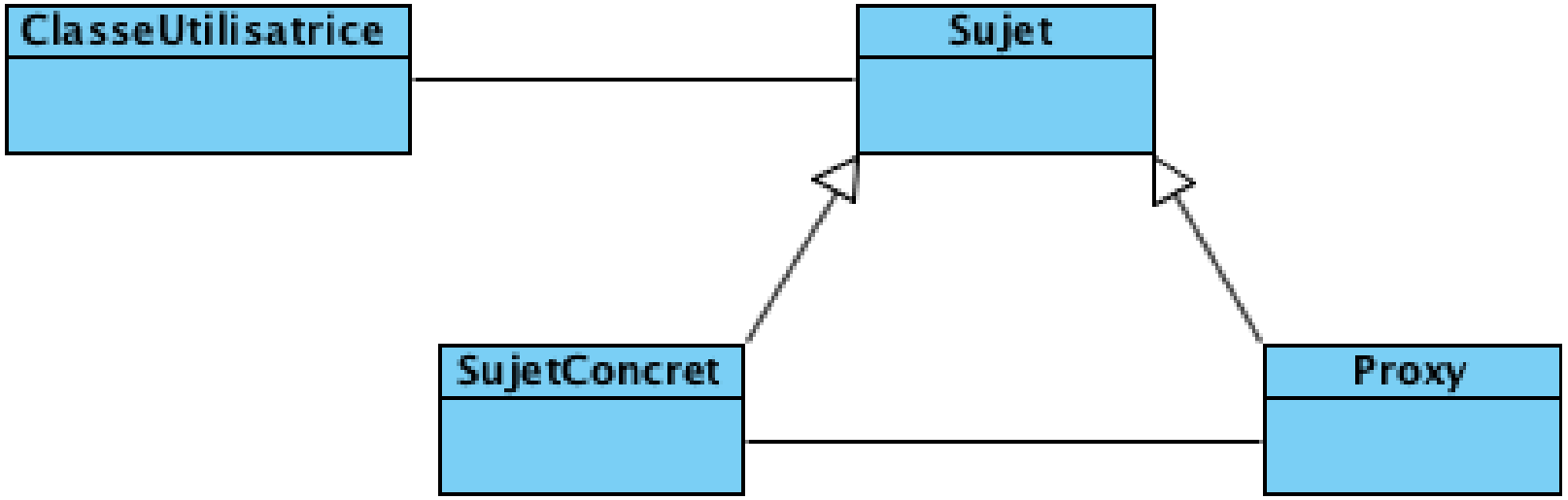
Le Proxy conserve une référence sur l'objet et l'utilisateur passe par le proxy pour y accéder

Le Proxy implémente le même interface que l'objet dans le but de le remplacer

Proxy

Niveau d'indirection supplémentaire pour accéder à un objet

Proxy



Adapter

Objectif : Convertir l'interface d'une classe pour l'adapter à l'interface désirée

Sert à faire communiquer deux interfaces incompatibles

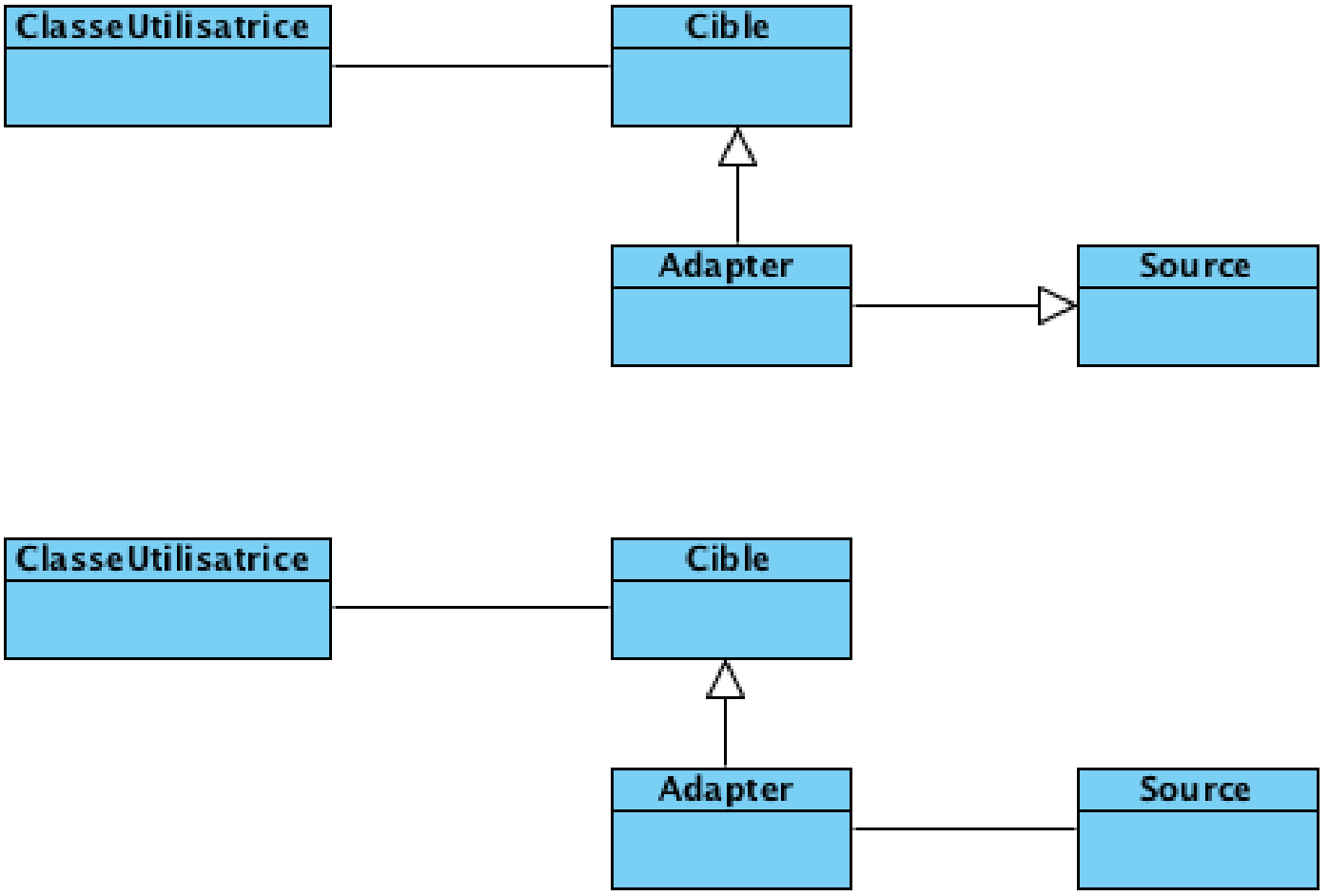
Adapter

Lorsqu'on veut utiliser une classe existante mais que son interface ne correspond pas à ce qu'on a besoin

Adapter

L'Adapter peut utiliser l'héritage ou la composition

Adapter

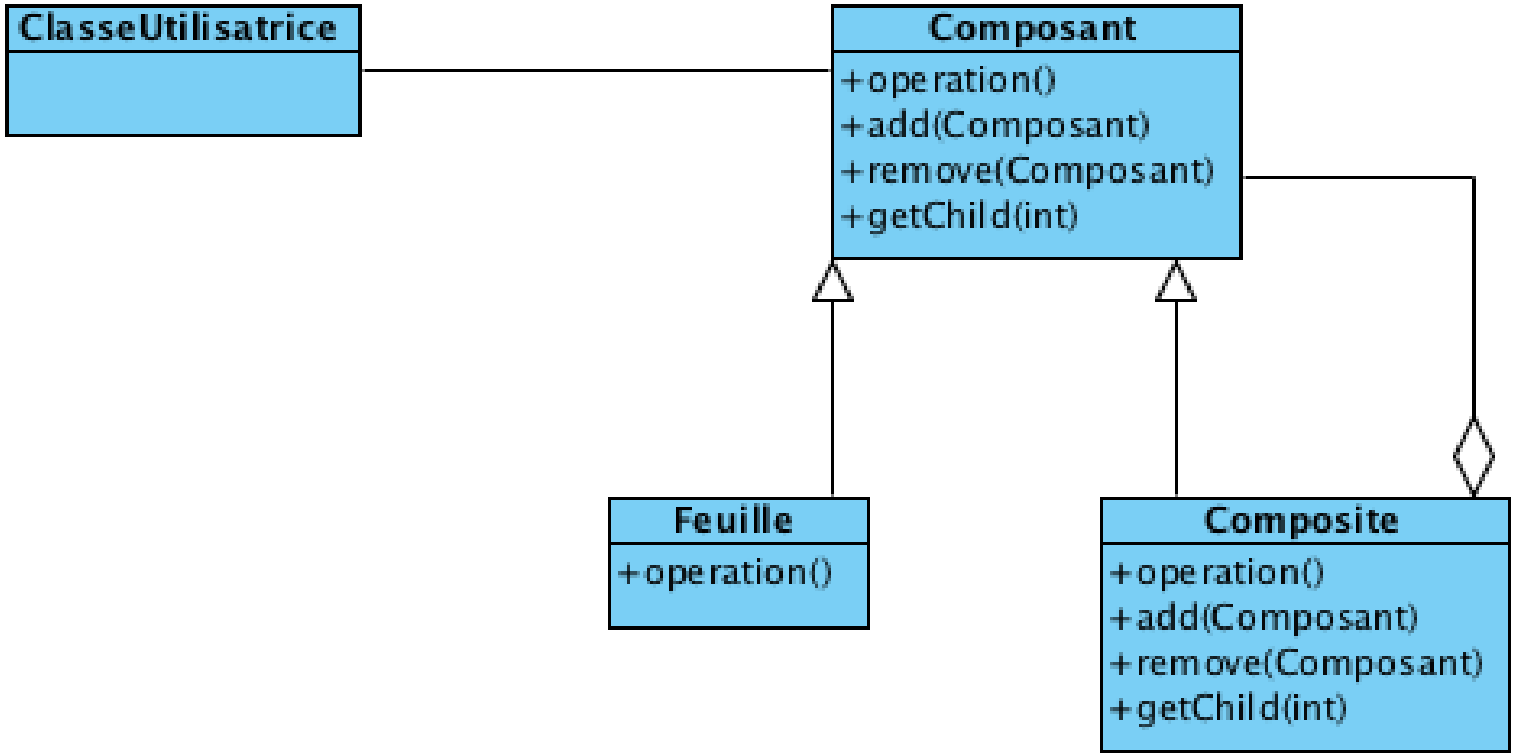


Composite

Objectif : Construit un objet selon une structure arborescente

Permet de traiter uniformément un objet ou un ensemble d'objets

Composite



Iterator

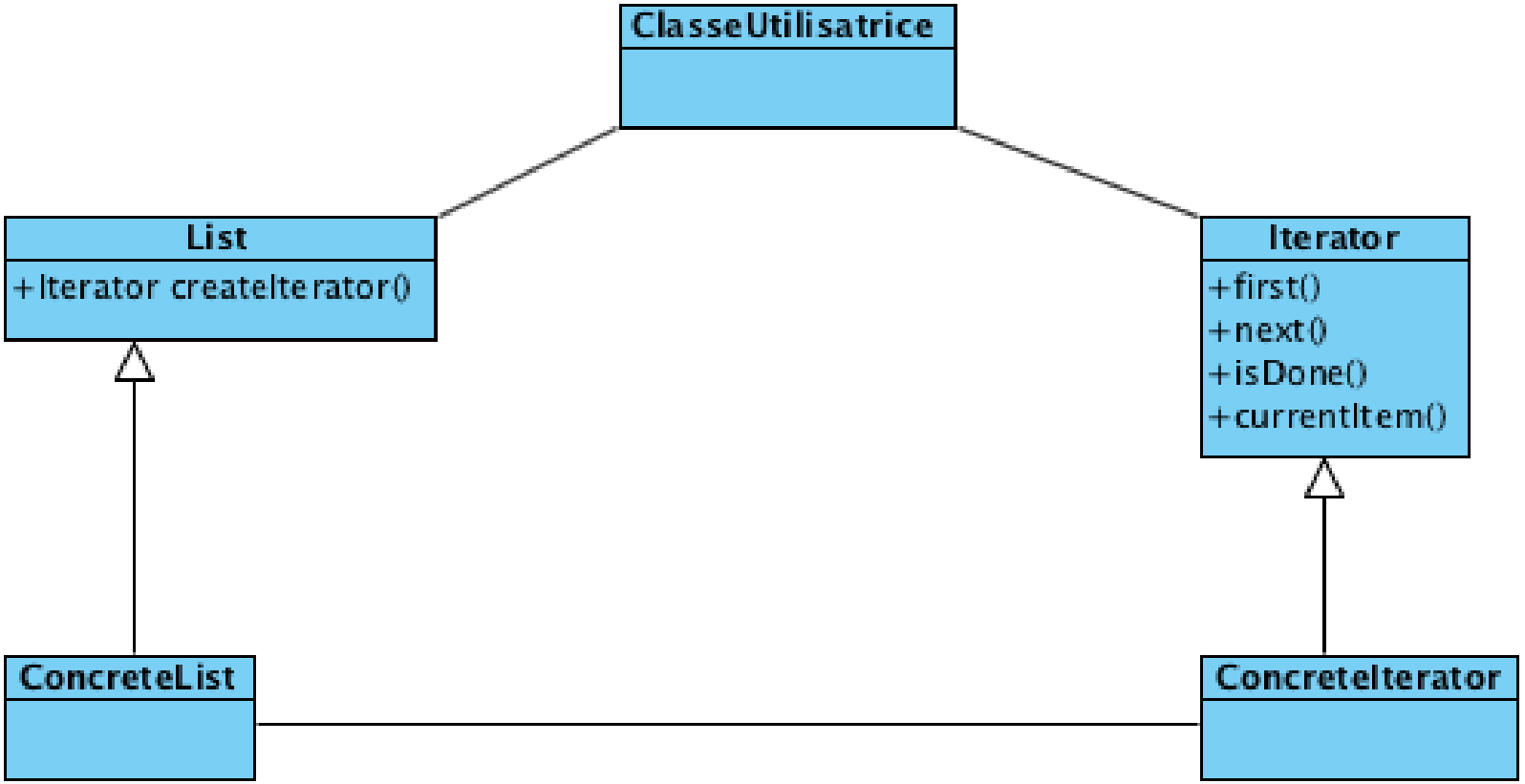
Objectif : Permet d'accéder aux objets d'une collection sans exposer la représentation interne

Iterator

Permet d'itérer sur n'importe quel type d'objet

Patron utilisé par la STL de C++

Iterator



Flyweight

Objectif : Partage d'objets pour supporter un grand nombre de petits objets efficacement

Interpreter

Objectif : Donne une représentation d'une grammaire d'un langage et un interpréteur pour interpréter des phrases du langage

Mediator

Objectif : Encapsulation de l'interaction d'un ensemble d'objets

Empêche les objets de faire référence les uns aux autres

Template Method

Objectif : Définir le squelette d'un algorithme et laisser le soin aux classes concrètes de redéfinir certaines étapes sans changer la structure de l'algorithme

Visitor

Objectif : Une opération à effectuer sur des éléments d'une structure de données

Permet de définir une nouvelle opération sur les classes qui opèrent la structure

Plus loin...

Design Patterns : Elements of Reusable Object-Oriented Software
Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
Addison-Wesley, 1995