ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

03
Chapitre

Les patrons de conception

2I1AC3 : Génie logiciel et Patrons de conception

Régis Clouard, ENSICAEN - GREYC

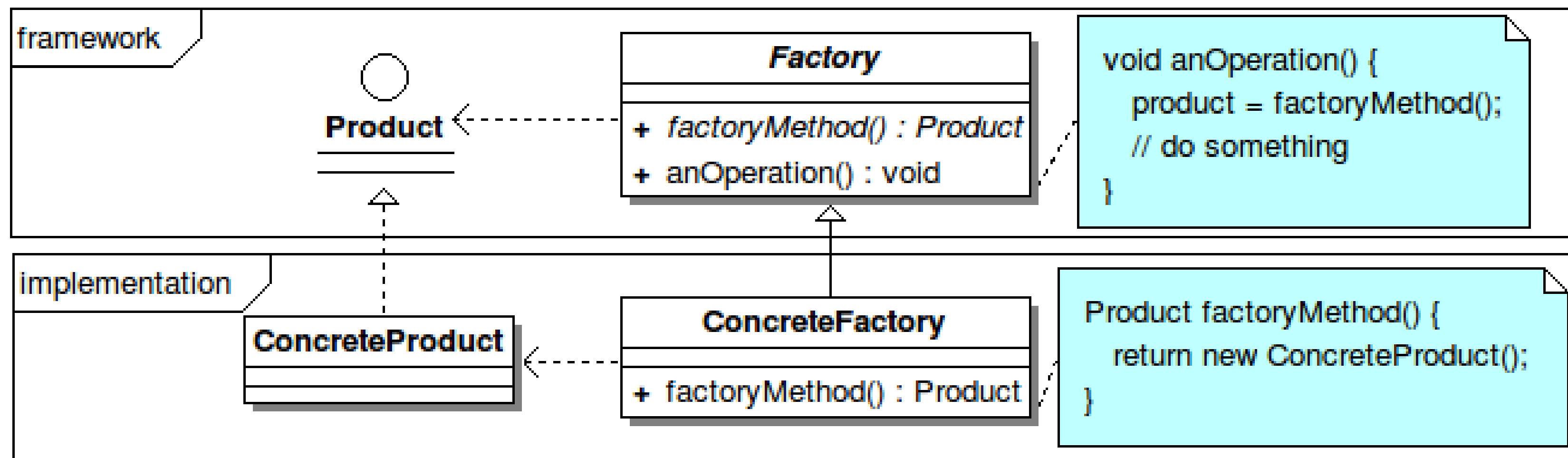
« Les patrons de conception vous aident
à apprendre des succès des autres
plutôt que de vos propres échecs. »
Mark Johnson

Plan

- Les patrons de conception de la bande des quatre

Méthode Fabrique

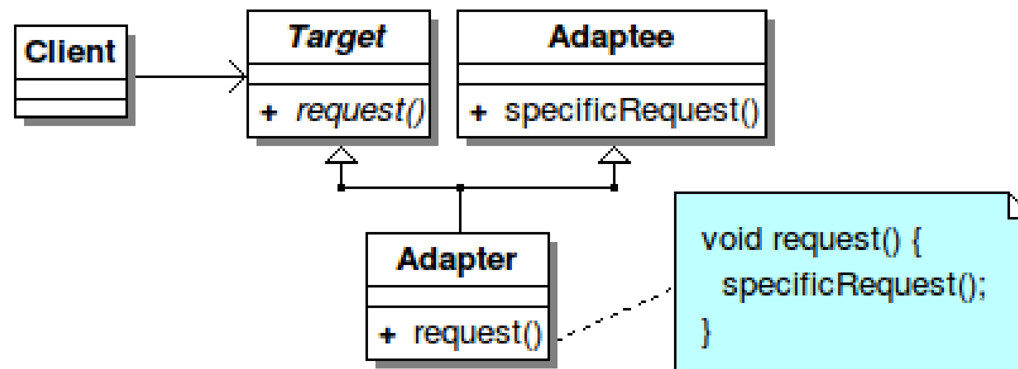
- Quand
  - On souhaite localiser la création d'objets de même type à un seul endroit.
- Solution
  - Définir une interface pour créer un objet, mais laisser aux sous-classe la décision de la classe à instancier.



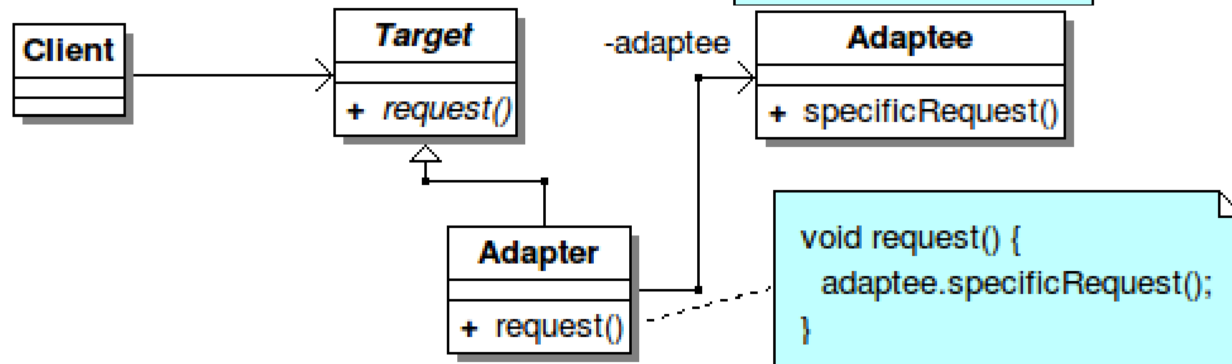
Adaptateur

- Quand
  - A class (a system) has the right data and the right methods but a wrong interface that cannot by modified.
- Solution
  - Create a class Adapter that encapsulates the class (the system) and provides the right interface.

Inheritance

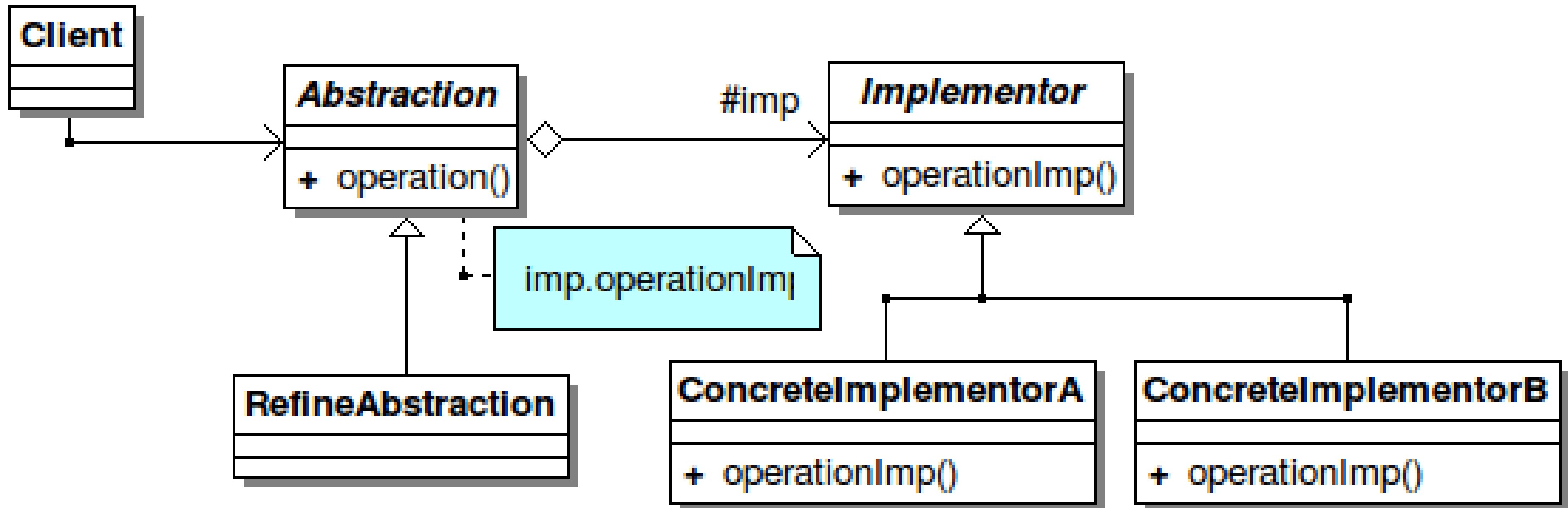


Composition



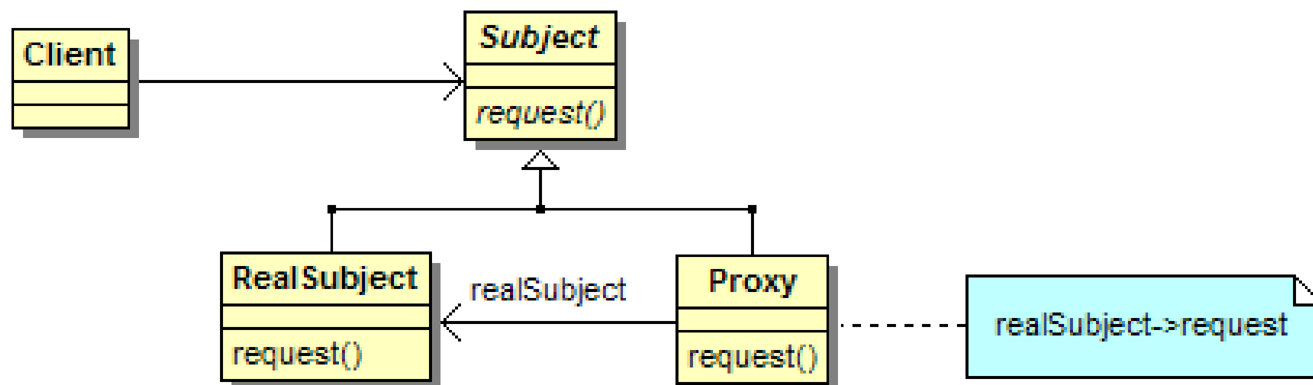
Pont

- **Quand**
  - Il existe une variété d'implémentations possibles pour une variété de composants.
- **Solution**
  - encapsuler chacune des variation dans une hiérarchie et relier ces deux hiérarchies.



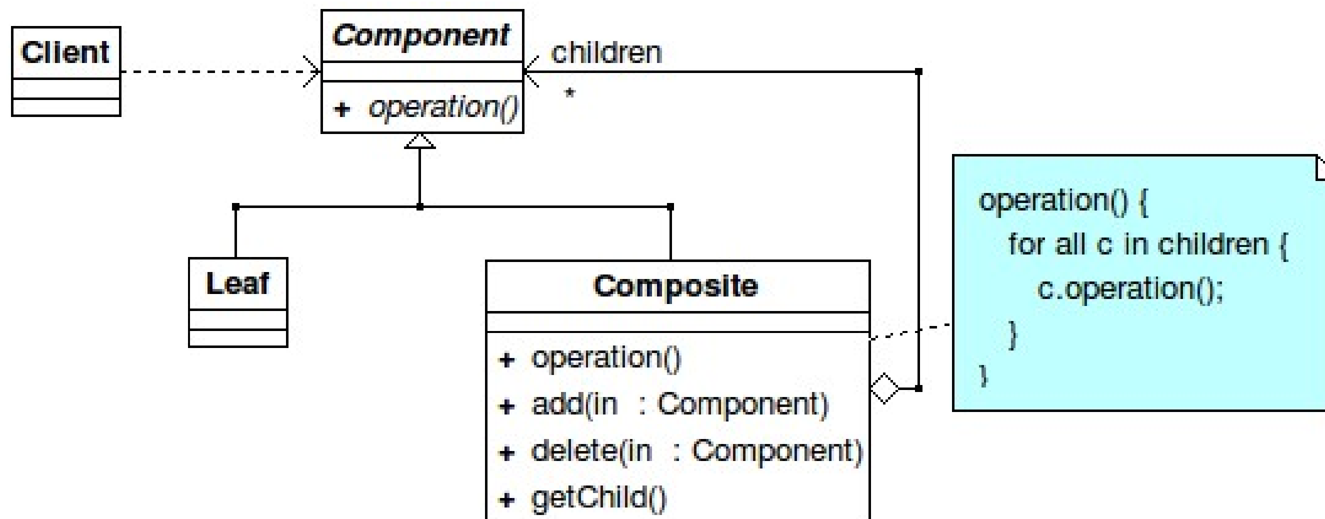
Procuration

- **Quand**
  - Fournir un représentant d'un autre objet pour en contrôler l'accès.
  - Fournir un représentant local d'un objet à distance.
  - Un client léger pour un composant lourd.
  - Une procuration de protection d'un composant à sensible.
  - une référence *intelligente* à un composant (smart pointer en C++).
- **Solution**
  - Créer un objet subrogé qui instancie l'objet concret la première fois que le client souhaite réellement manipuler l'objet ;



Composite (page 6)

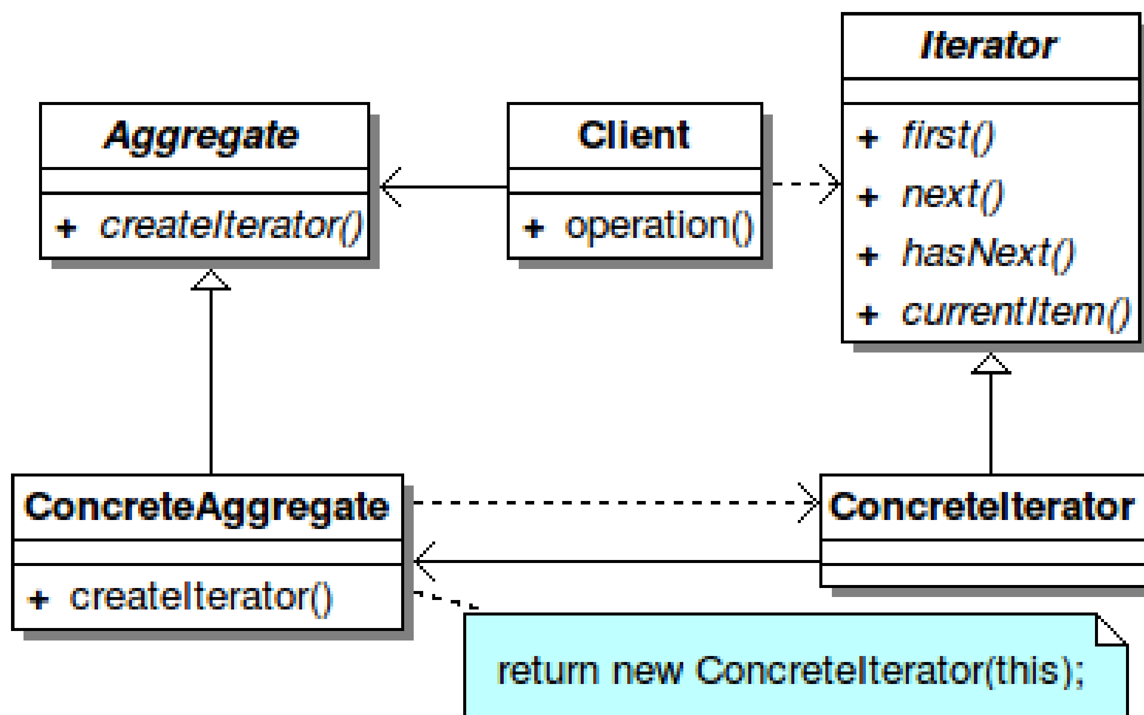
- **Quand**
  - On souhaite créer des objets complexes par composition d'objet simples.
  - Les objets simples et les objets complexes doivent présenter la même interface.
- **Solution**
  - Une composition hiérarchique des objets



- **Rema**
  - Pr

Itérateur (page 13)

- **Quand**
  - On souhaite parcourir n'importe quel agrégat avec une interface unique sans risque pour l'agrégat.
- **Solution**

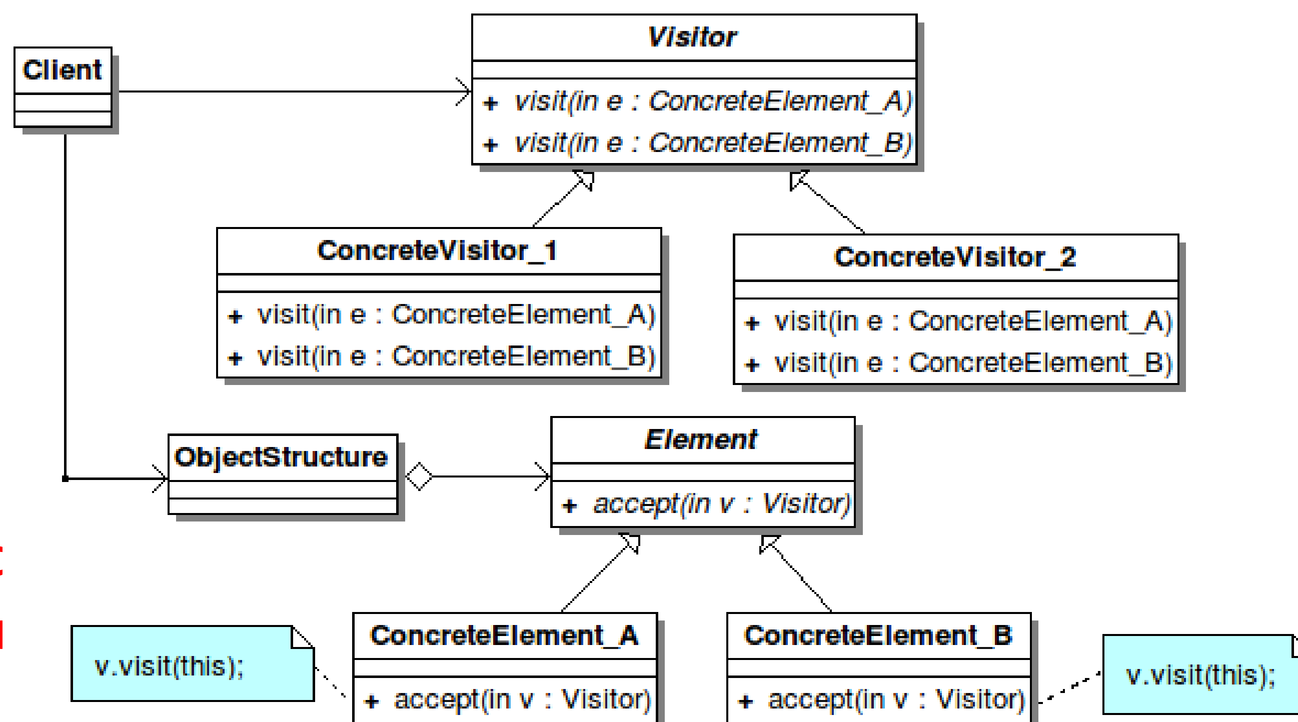


Itérateur (page 13)

- Implémentation
  - La classe Itérateur doit avoir accès privilégié à la représentation de l'agrégat.
  - Pour ne pas rompre l'encapsulation :
    - **En Java** : utiliser des classes internes.
    - **En C++,** on utilise les classes amies.
- Remarque
  - **Itérateur actif** : c'est le client qui fait avancer sur les éléments. Le patron correspond aux itérateurs actifs.
  - **Itérateur passif** : le client demande l'application sur tous les éléments cf. stream en java.

Visiteur (page 25)

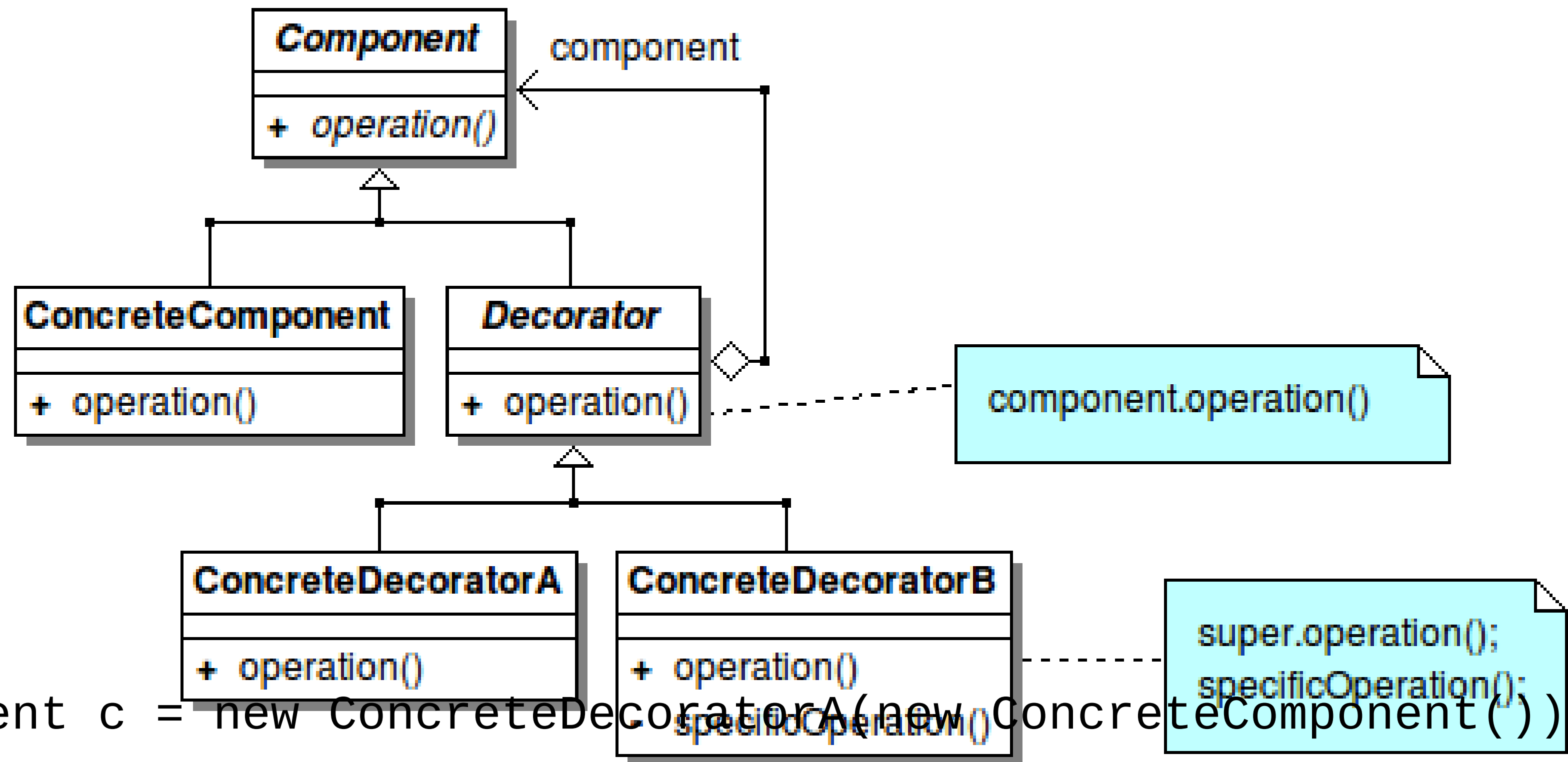
- **Quand**
  - On possède une structure de données qui **n'évolue plus**, et on souhaite ajouter des fonctionnalités à cette structure de données (*à la manière de plugins.*)
  - **Ce qui varie :** la liste des méthodes associées aux classes.
- **Solution**



- **Important**
  - **La structure de c**
  - **Augmente forte**

Décorateur (page 7)

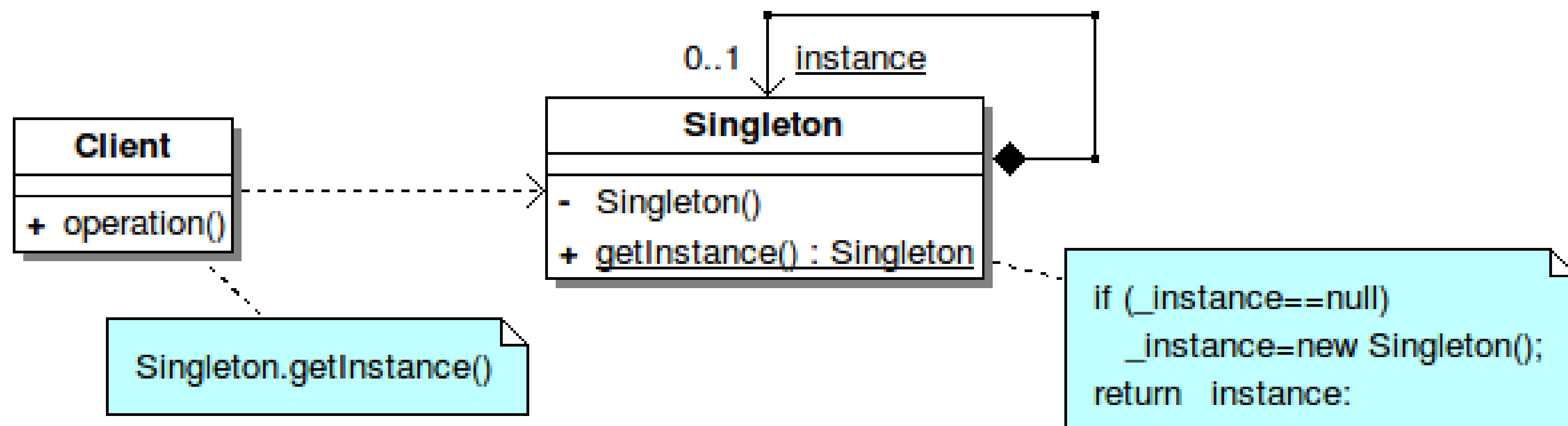
- **Quand**
  - On souhaite ajouter dynamiquement des fonctionnalités supplémentaires au comportement d'un objet.
  - **Ce qui varie** : le comportement d'un service
- **Solution**



- **Création**
  - `Component c = new ConcreteDecoratorA(new ConcreteComponent())`

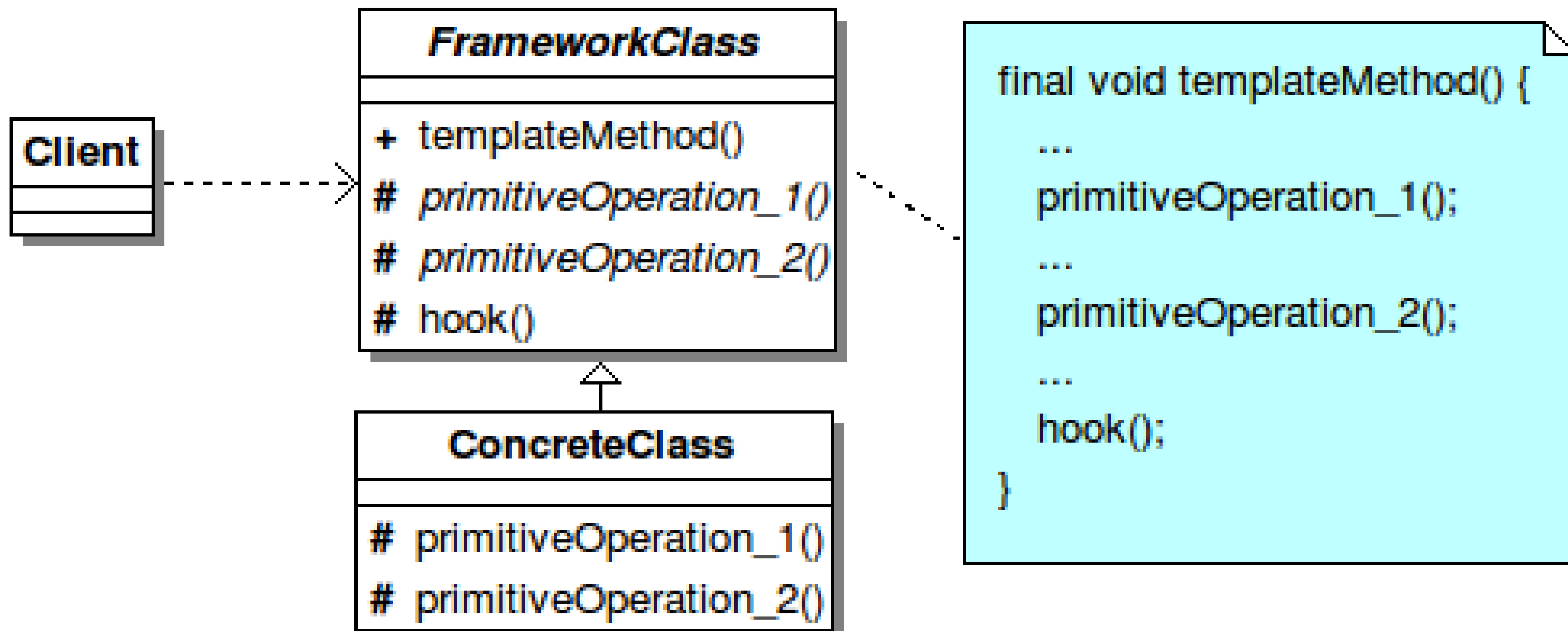
Singleton (page 23)

- **Quand**
  - On souhaite garantir qu'il n'y a qu'une instance d'une classe
  - Problème critique si on ne garantie pas cette unicité
    - contrôleur d'impression
    - gestionnaire de fenêtre
    - contrôle central
- **Solution**
  - Utiliser une méthode statique de création de l'instance



Template method

- **Quand ?**
  - On souhaite une méthode avec des parties dépendante d'un type d'objet
- **What varies**
  - Les parties d'un algorithme
- **Solution**

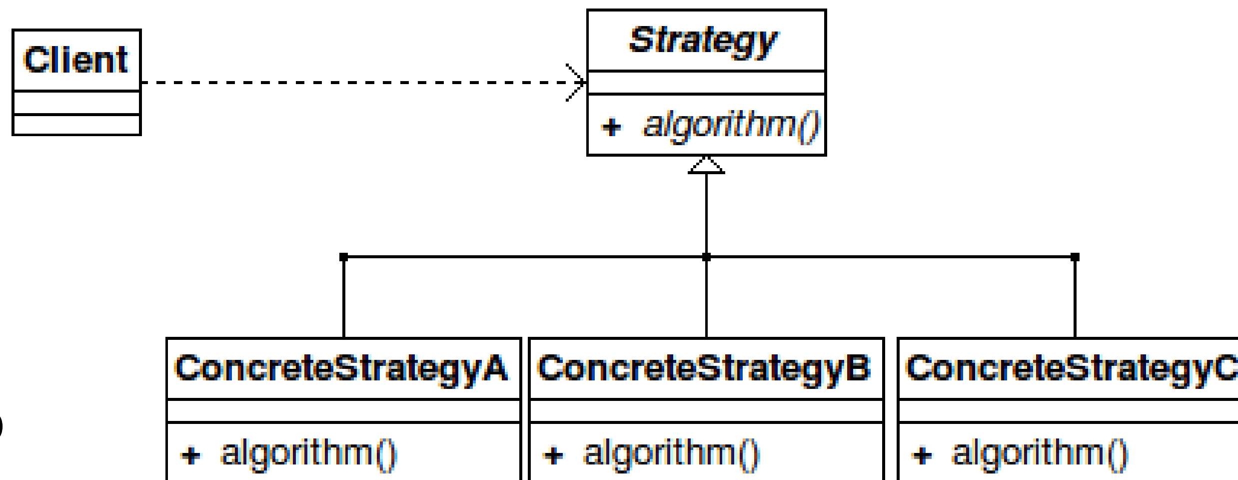


Template method

- **Implémentation**
  - Les opérations appelées par le patron de méthode sont en mode "`protected`". Les primitives qui doivent être surchargées sont abstraites (virtuelles pures en C++).
  - La méthode qui définit l'algorithme doit être protégée contre la surcharge (`final` en Java).
  - Il est possible d'ajouter une méthode `hook()` concrète mais vide dans la classe de base, qui peut être redéfinie dans les classes dérivées pour ajouter une fonctionnalité optionnelle.

Strategy

- **Quand ?**
  - On souhaite disposer de plusieurs variantes d'un même service que l'on peut interchanger en fonction du contexte
- **What varies**
  - L'algorithme complet implémentant un service
- **Solution**



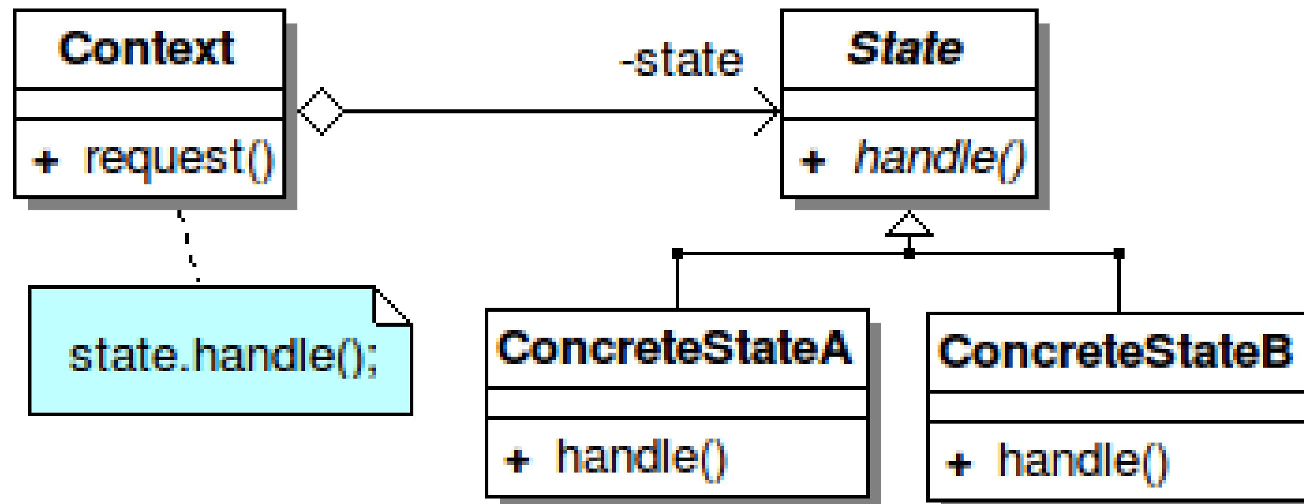
- **Variante : Po**
  - Une liste de

Strategy

- **Implémentation**
  - Les interfaces de Stratégie et de Client doivent assurer à une stratégie concrète l’accès aux données d’un client dont elle a besoin :
    - Les informations peuvent être passées sous forme de paramètres :
      - public void algorithm(Parameter arg);
    - Le client peut fournir sa référence en paramètre, et la stratégie lui réclame les données :
      - public void algorithm(Client client);
      - Ici il est nécessaire d'avoir un lien privilégié vers le client
        - C++: classe amie
        - Java : classe interne

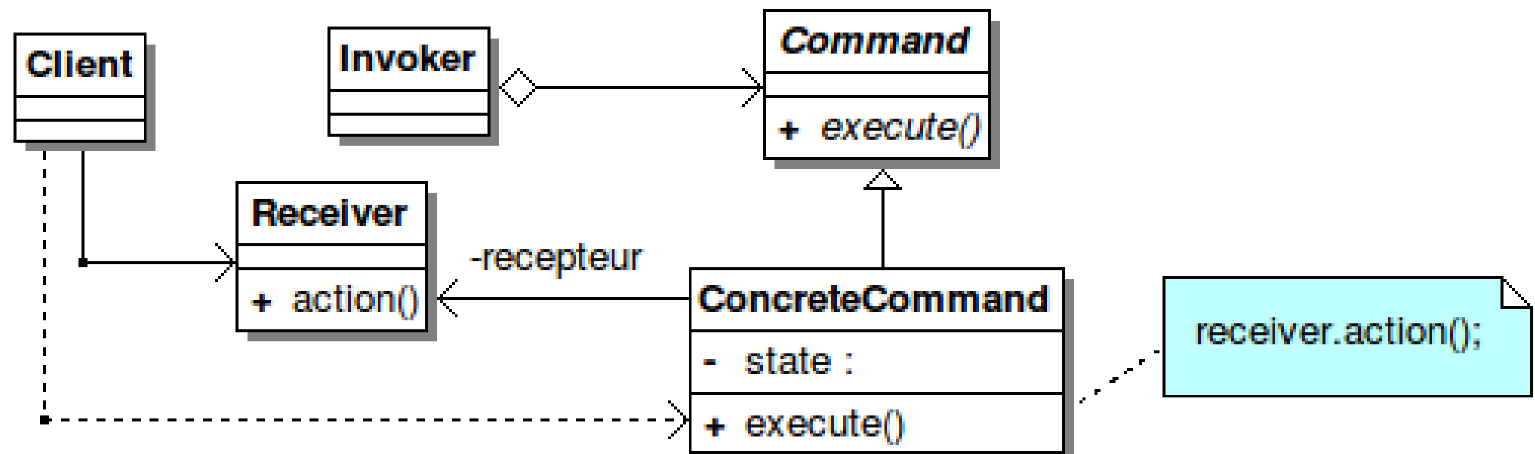
State

- **Quand ?**
  - Un objet peut prendre plusieurs états, ce qui induits des comportements différents pour ses services à partir de l'appel du même nom.
- **What varies**
  - La nature des services rendus en fonction de son état.
- **Solution**



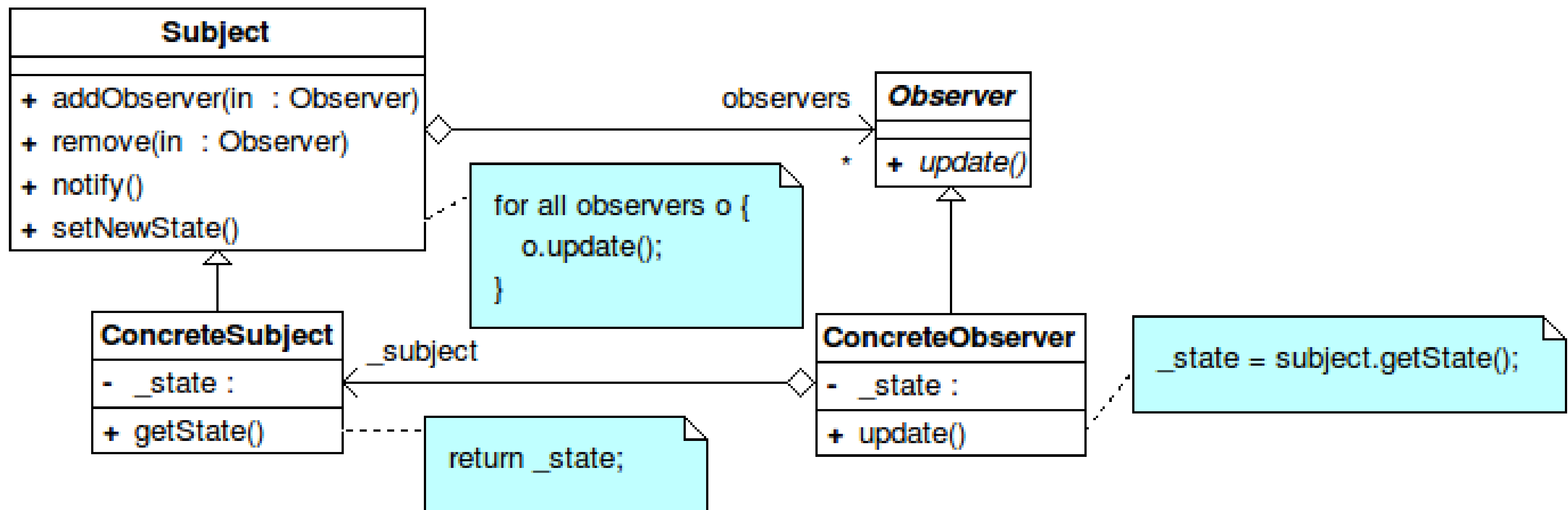
Command

- Quand ?
  - Encapsuler une requête comme un objet pour permettre :
    - de gérer une liste d'attente
    - de gérer un historique
    - d'assurer le traitement d'opérations réversibles
    - de pouvoir réaliser un film des actions
- Solution
  - Encapsuler la requête dans un objet, la stoker dans un objet invocateur.



Observer

- **Quand ?**
  - On veut avertir une liste **variable** d'objets qu'un événement a eu lieu
- **What varies**
  - La liste des objets qui dépendent des modifications d'un objet de référence
- **Solution**

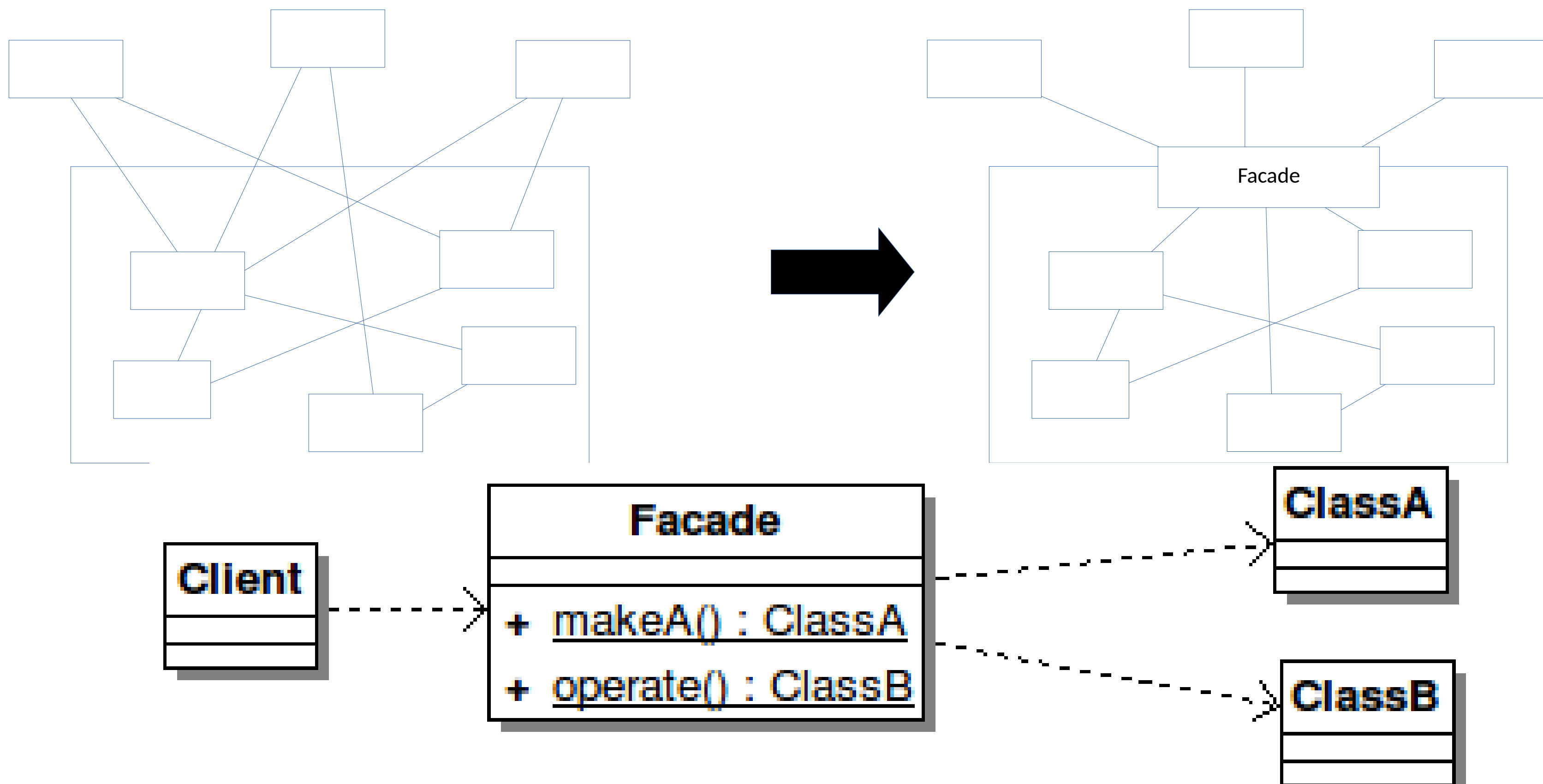


Observer

- Deux modèles du processus de notification :
  - **push**
    - `public void update( Data data );`
  - **pull**
    - ```
      public void update() {
          ...
          _subject.getData();
          ...
      }
      ```
      ```

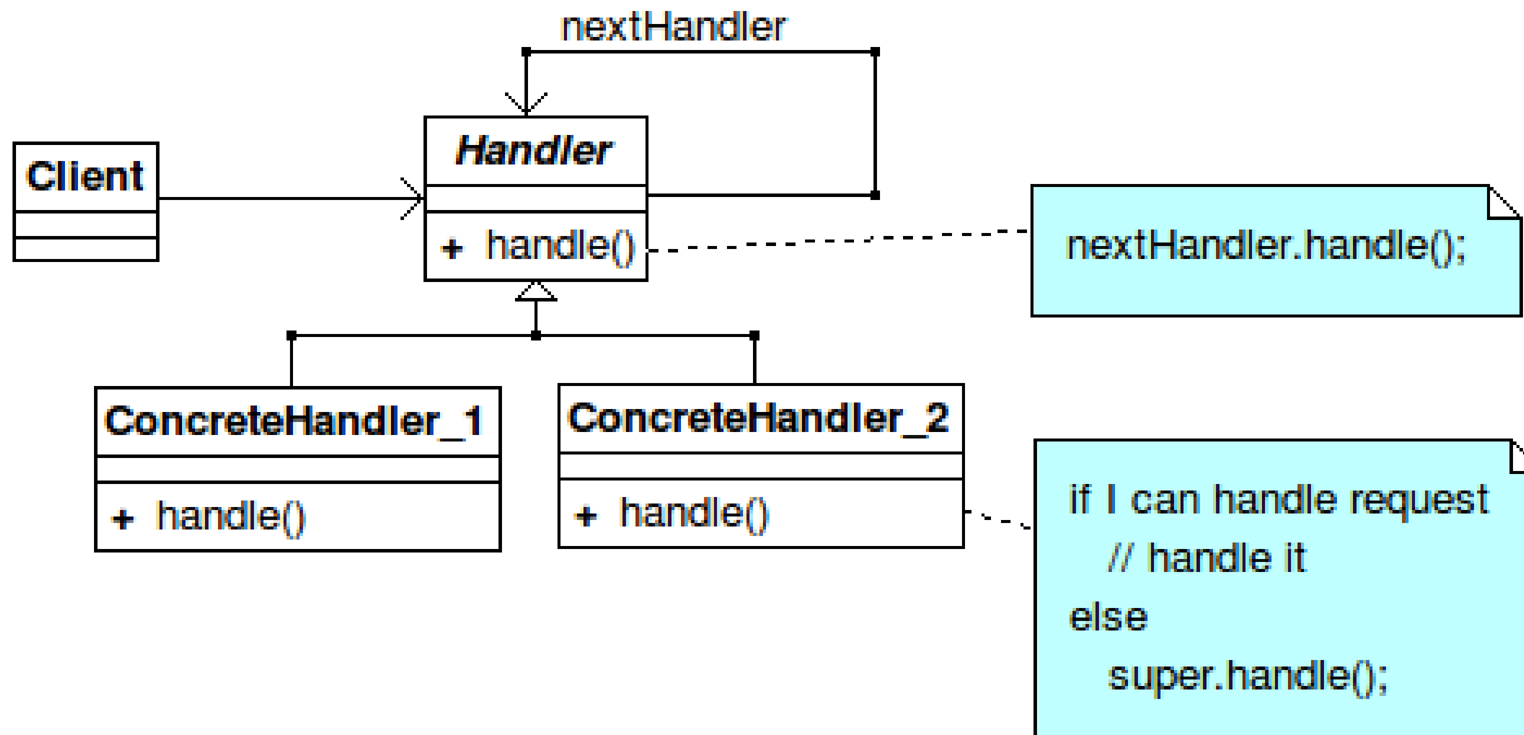
Facade

- **Quand** ?
  - On souhaite simplifier l'interface d'accès à un sous-système
- **Solution**
  - Une classe abstraite qui regroupe les éléments de l'interface.
  - Cf. Principe de ségrégation des interfaces.



Chain of responsibility

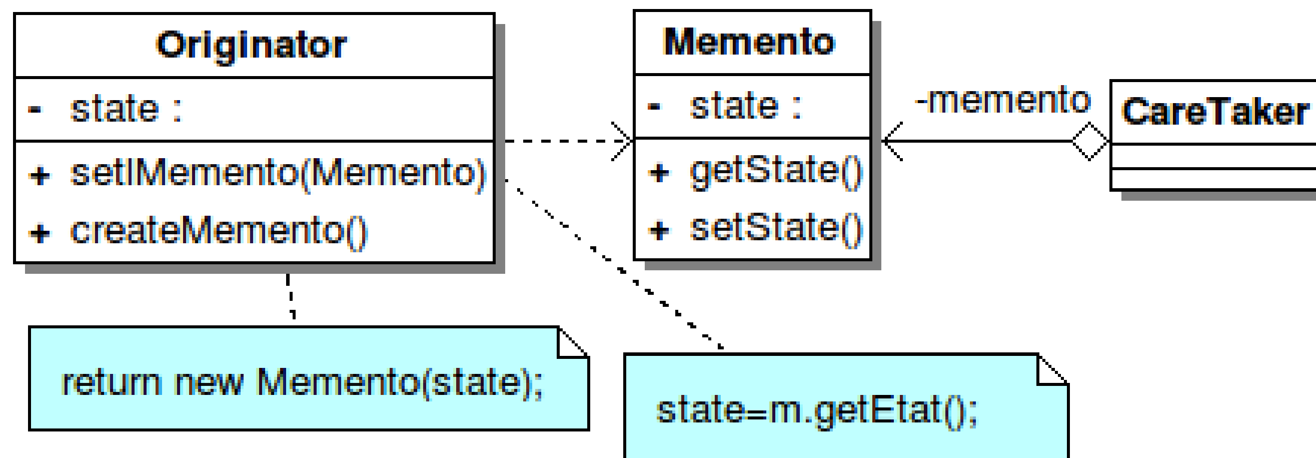
- **When**?
  - Déléguer la réalisation d'une fonctionnalité à une chaîne de délégation inconnue (cf. le tondeur de pelouse).
- **What varies**
  - L'objet qui peut répondre à un service
- **Solution**



		Role		
		Creation	**Structure**	**Behavioral**
Domain	**Class**	Factory method	Adapter	~~Interpreter~~ Factory Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Memento

- Quand ?
  - On souhaite mémoriser un état interne pour :
    - pouvoir revenir à un état antérieur
    - mettre ne place un mécanisme de réversion après erreur
    - réaliser une sauvegarde intermédiaire (jeux)
    - transmettre un état sur le réseau
- Solution
  - Délocaliser le stockage d'un état interne vers un autre objet (memento)



Messenger

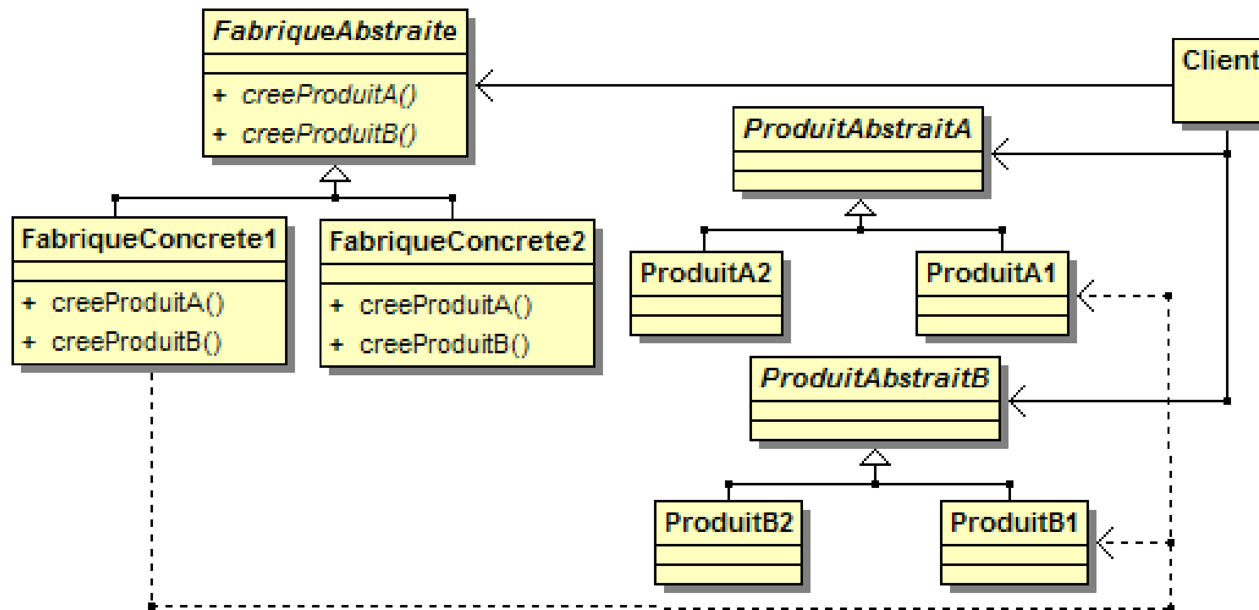
- Un objet qui agit comme une simple structure
  - sans méthode
  - données sont « **public** »
- When?
  - Simplifier la liste d'arguments de méthodes
- Example
  - Une instance de la classe Point mieux que les deux entiers x et y
- Increase cohesion

Collecte de paramètres

- Classe qui collecte des données auprès de plusieurs méthodes
  - Une abeille qui collecte pollen
- Implémentation
  - Généralement, une table de hashage (a.k.a tableau associatif, dictionnaire)
- Example
  - La classe Properties du Java qui permet de collecter les valeurs des paramètres d'une application, ici la taille de la fenêtre.
    - Property p = new Property()
    - p.addInt("WindowWidth", 120)

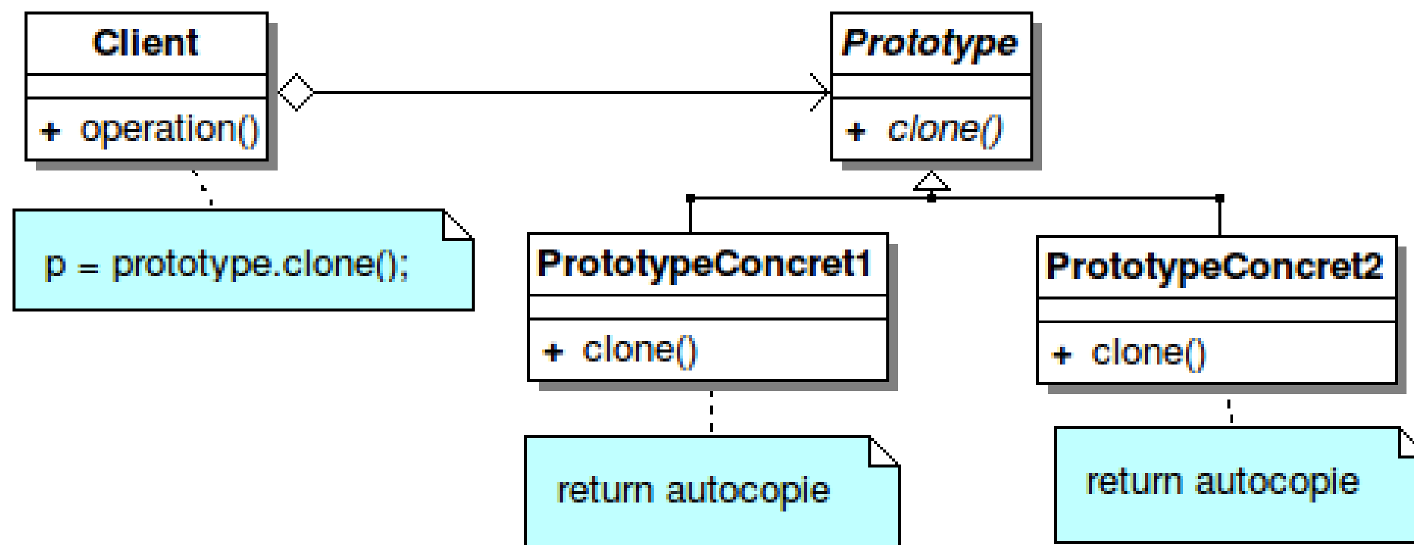
Fabrique abstraite

- Quand
  - Vous souhaitez des familles ou des ensembles d'objets un client particulier (ou cas particulier)
- Solution
  - Définir une interface qui permet de créer chaque membre de la famille d'objet. Typiquement, chaque famille est créée à partir d'une unique fabrique concrète.



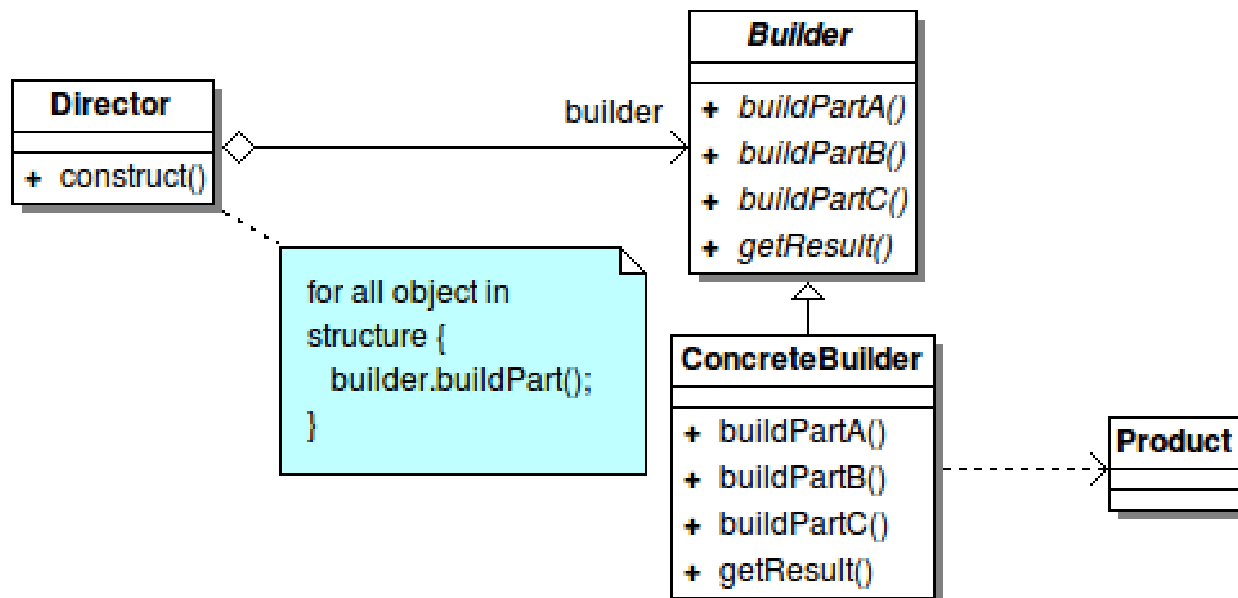
Prototype

- Quand
  - La copie d'un objet est difficile
    - La structure de l'objet est très complexe.
    - Les parties d'un objet sont protégées.
- **Solution**
  - Donner la possibilité aux objets de créer leur propre copie.



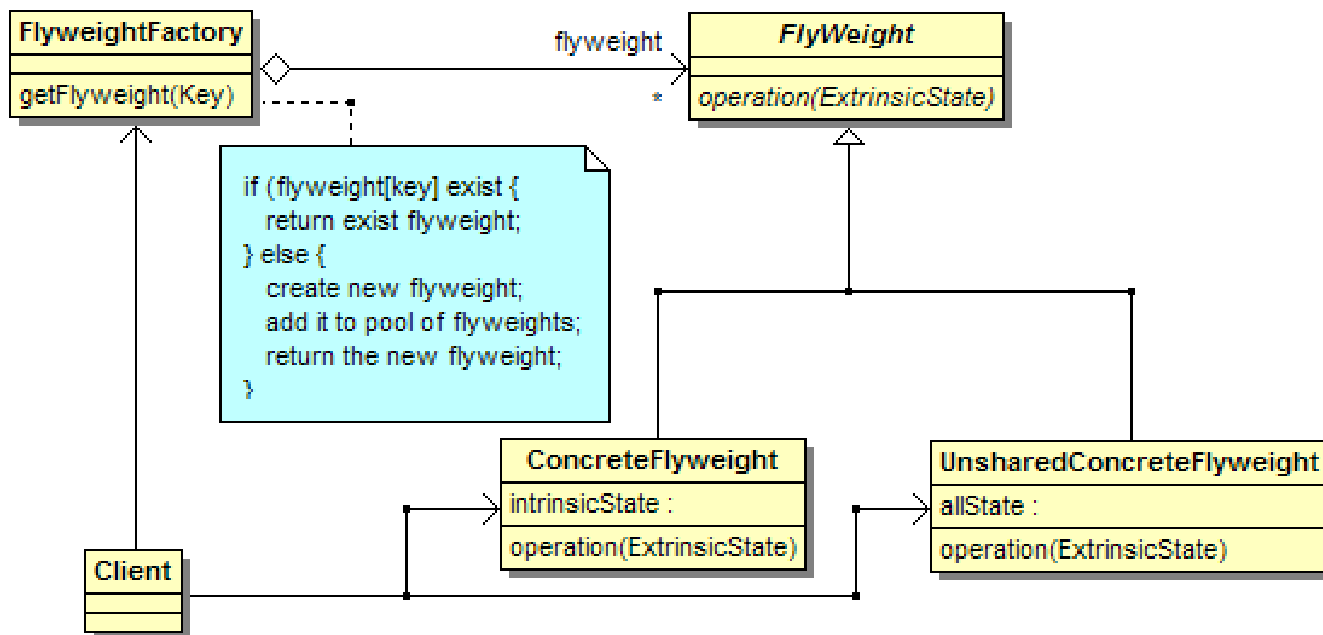
Monteur

- **Quand ?**
  - On souhaite créer un objet constitué de plusieurs parties
    - La liste des parties est fixée pour tous les types d'objet
    - Le type des parties est fixée par type d'objet
- **Solution**



Flyweight

- **Quand** ?
  - Il existe un très grand nombre de petits objets
- **Solution**
  - Si les objet présentent essentiellement un état extrinsèque alors on peut créer une fabrique d'objet qui garantie l'unicité de chaque objet.



Mediator

- **When?**
  - On souhaite créer un contrôle centralisé qui organise la communication entre des objets qui s'ignorent mutuellement.
- **Solution**