

Patrons de structure (PARTIE VII - Patrons de conception)

Bruno Bachelet
Christophe Duhamel
Luc Touraille

- Concevoir de nouveaux composants par assemblage
 - Pour former des structures plus vastes
 - Avec un comportement plus complexe
- Objectif: exploiter les capacités d'un composant et les adapter à de nouveaux besoins
- Niveau classe
 - Utilisation de l'héritage
 - ⇒ Composition d'interfaces ou d'implémentations
- Niveau objet
 - Utilisation de la composition

- Adaptateur / *Adapter*
 - Adapter l'interface d'une classe à ses besoins

- Pont / *Bridge*
 - Découpler l'interface d'un composant de son l'implémentation

- Composite / *Composite*
 - Composer des objets sous forme arborescente

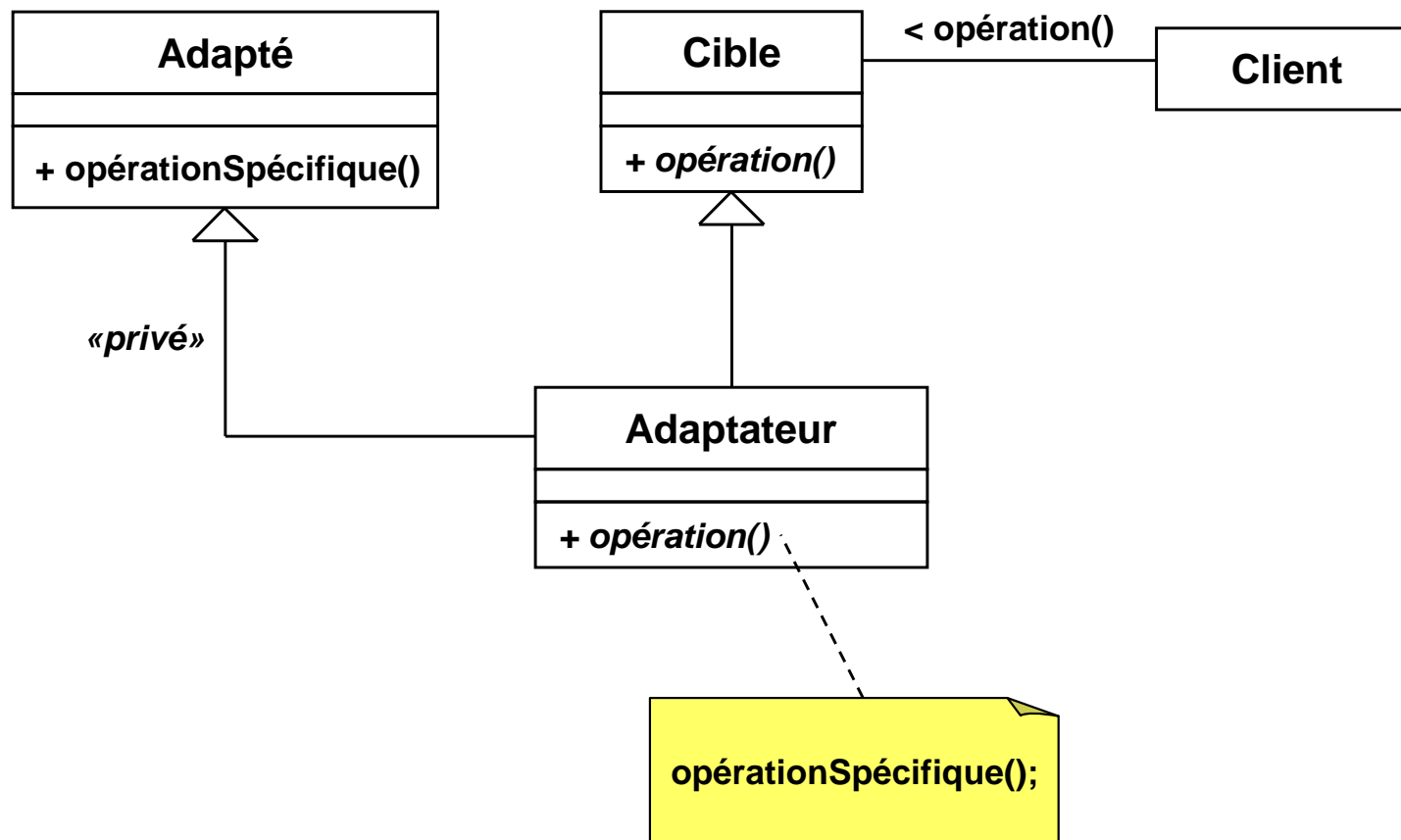
- Décorateur / *Decorator*
 - Ajouter dynamiquement des fonctionnalités à un objet
- Façade / *Facade*
 - Découpler un sous-système de ses clients
- Poids-mouche / *Flyweight*
 - Partager des instances pour éviter un nombre trop important
- Proxy / *Proxy*
 - Fournir un substitut pour accéder à un objet

Adaptateur / Adapter (1/5)

- Objectif
 - ❑ Adapter l'interface d'une classe à ses besoins
 - ❑ Permettre le dialogue entre classes incompatibles
- Principe
 - ❑ Deux approches
 - ❑ Classe «adaptateur»
 - Héritage de la nouvelle interface
 - Héritage de l'implémentation de l'ancienne interface
 - ❑ Objet «adaptateur»
 - Héritage de la nouvelle interface
 - Agrégation d'un objet de l'ancienne interface, et délégation
- Motivation
 - ❑ Utiliser une fonctionnalité d'une bibliothèque tierce
 - ❑ Mais l'interface n'est pas adaptée

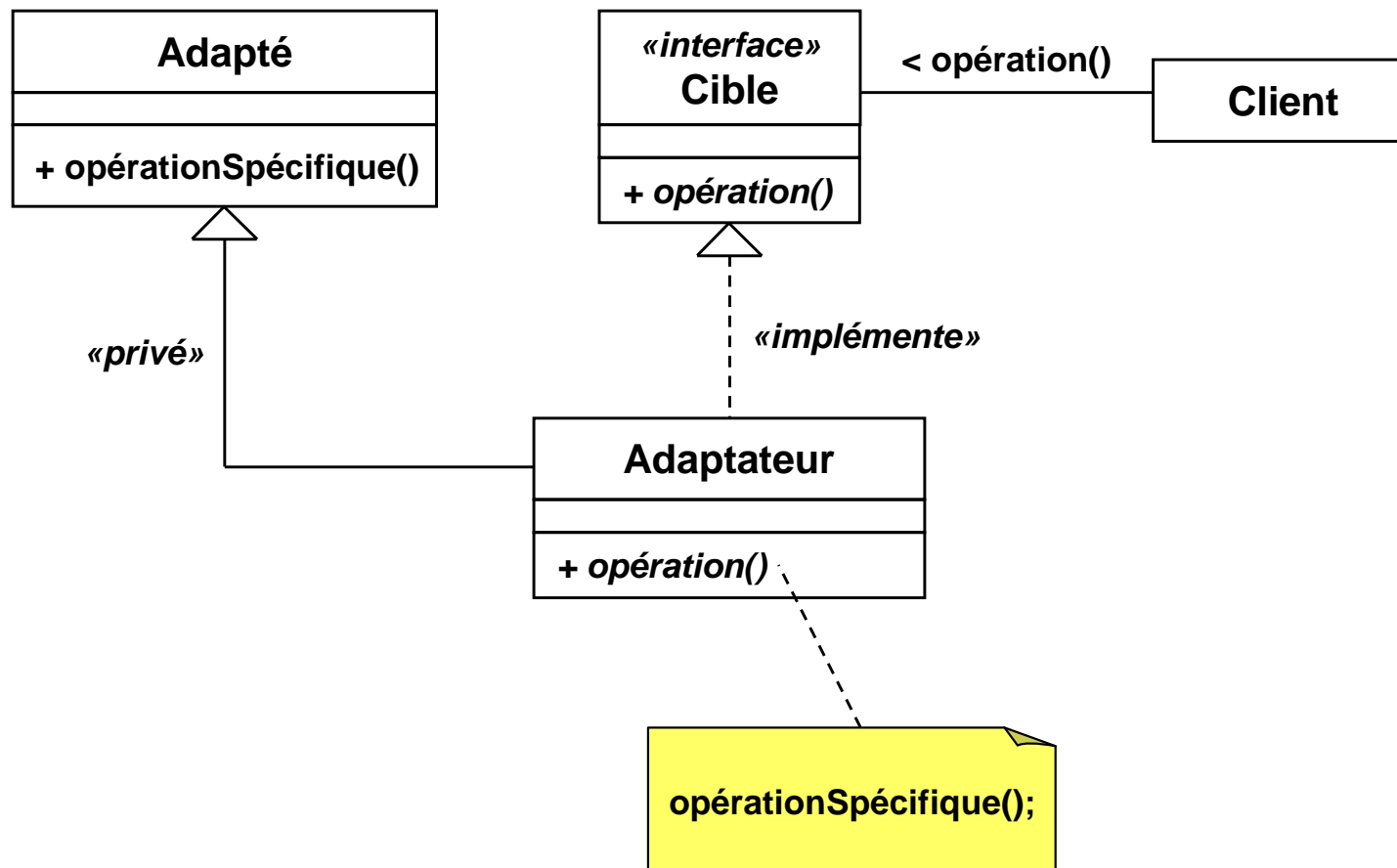
Adaptateur / Adapter (2/5)

- Classe adaptateur, version 1
 - ❑ Héritage de l'implémentation de l'adapté = héritage privé
 - ❑ Inconvénient: héritage multiple



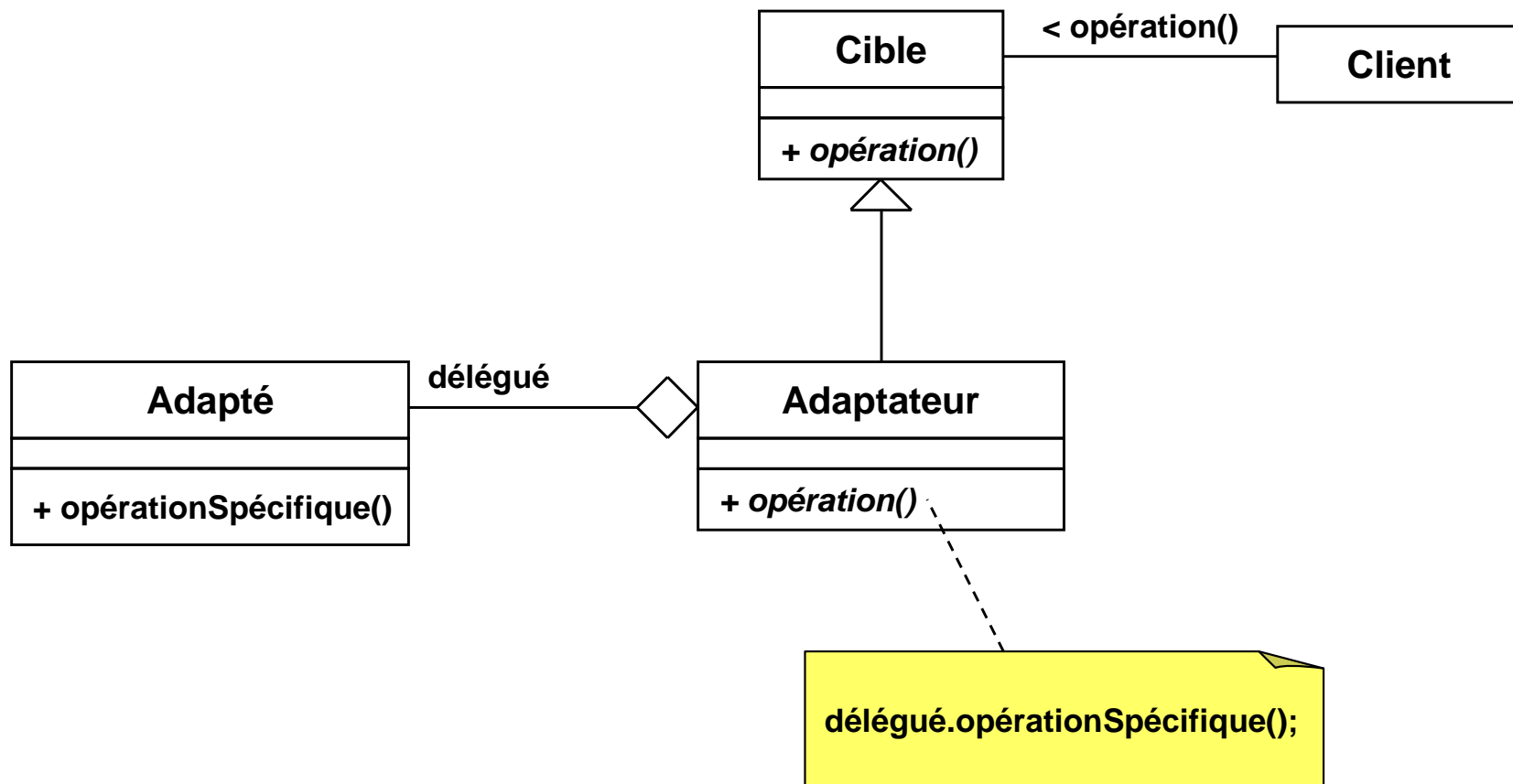
Adaptateur / Adapter (3/5)

- Classe adaptateur, version 2
 - ❑ La cible est une interface, et non une classe
 - ❑ Plus de problème d'héritage multiple



Adaptateur / Adapter (4/5)

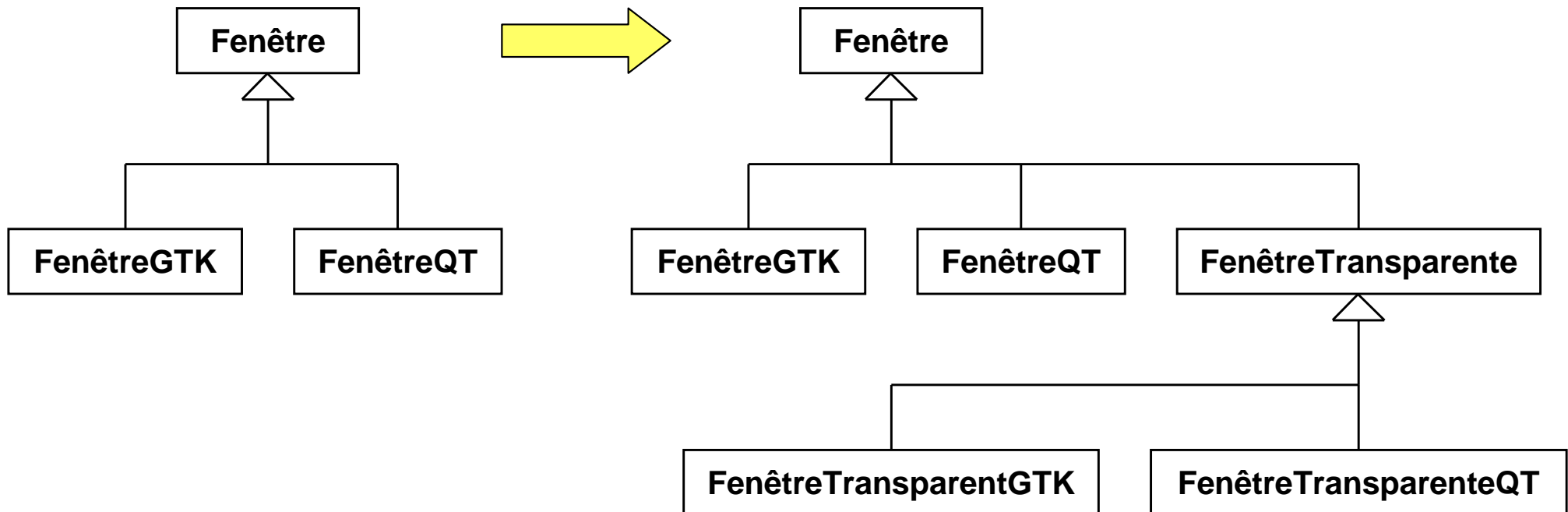
- Objet adaptateur, délégation
 - ❑ Un objet de la classe adaptée est agrégé dans l'adaptateur
 - ❑ Plus d'héritage privé, ni multiple



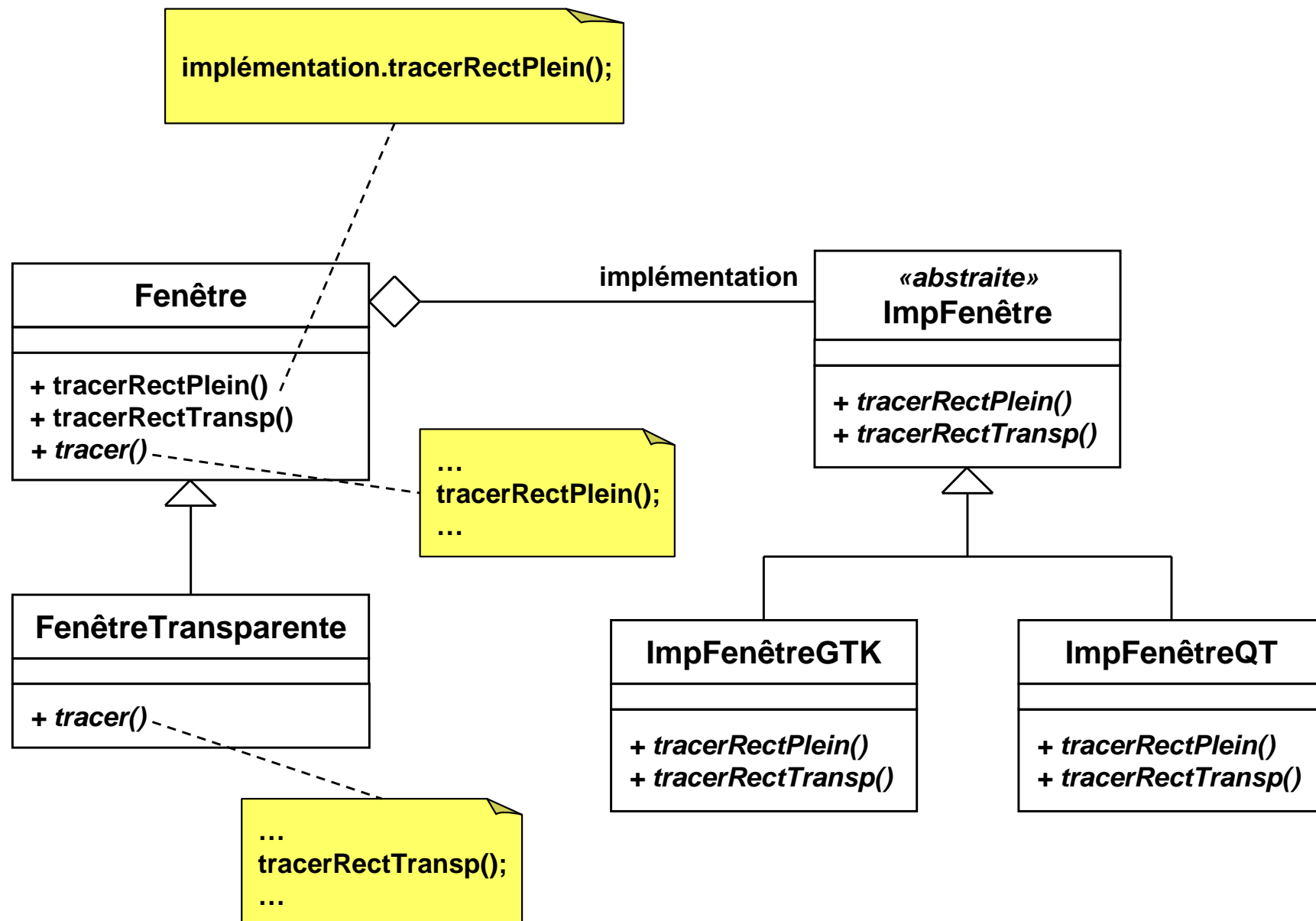
- Appelé aussi «*wrapper*»
- Intérêts
 - ❑ Classe adaptateur
 - Permet une surcharge simple des fonctionnalités de l'adapté
 - L'adaptation ne crée qu'un seul objet
 - ❑ Objet adaptateur
 - Plus d'héritage multiple
 - L'adaptation se fait sur une classe et ses sous-classes
- Patrons similaires
 - ❑ Pont: séparation interface / implémentation
 - ❑ Décorateur: ajout dynamique de fonctionnalités à un objet
 - ❑ Proxy: accès à un objet à travers un intermédiaire

- Objectif
 - Découpler l'interface d'un composant de son implémentation
- Principe
 - Séparation de la classe en deux classes
 - L'une représente l'interface du composant
 - L'autre l'implémentation du composant
 - L'interface agrège une implémentation à laquelle elle délègue les appels aux méthodes
- Motivation
 - Plusieurs logiques d'héritage peuvent s'entremêler

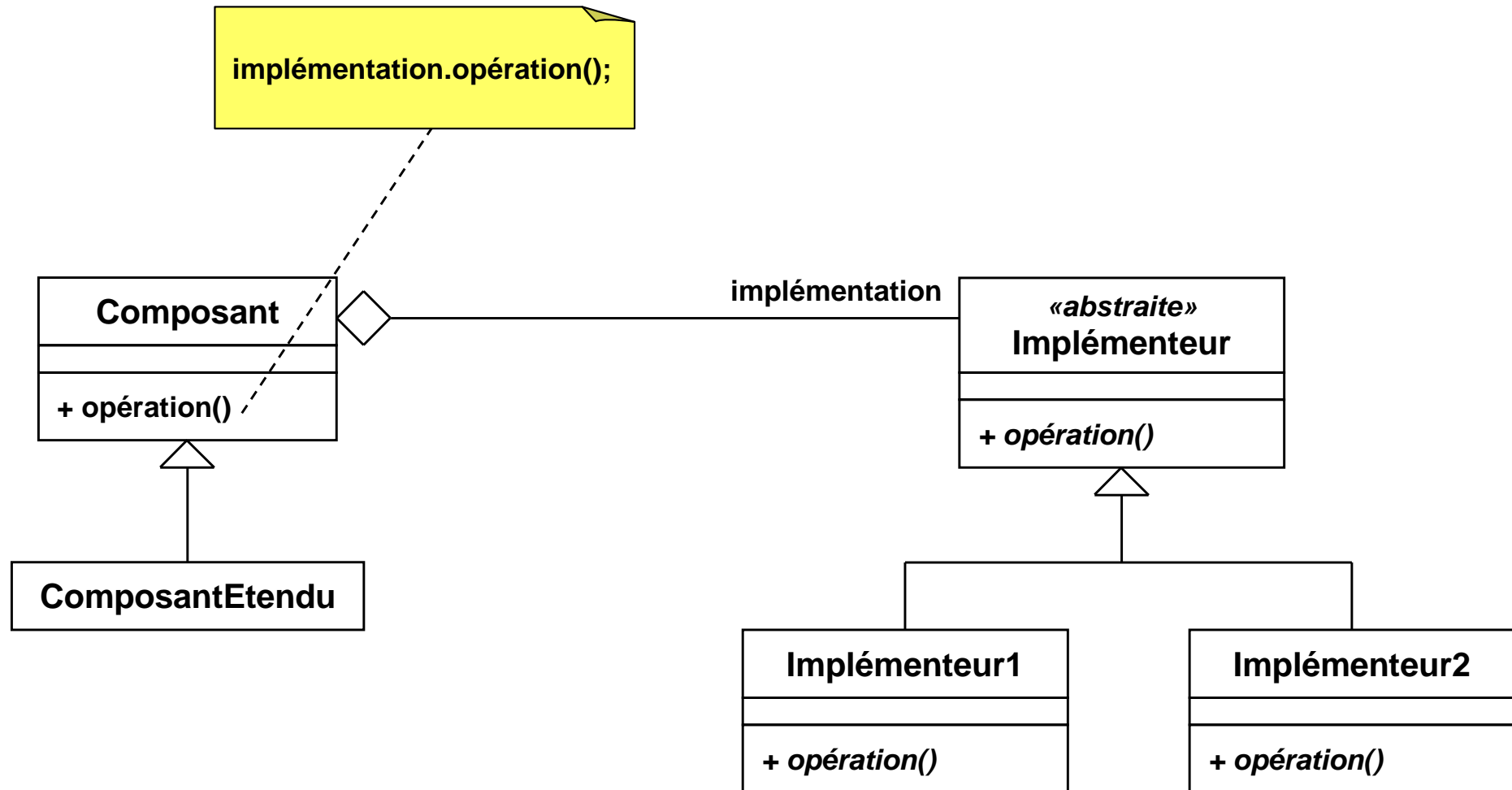
- Exemple d'entremêlement
 - ❑ Classe «Fenêtre» spécialisée pour 2 bibliothèques graphiques
 - ❑ Sous-classe «FenêtreTransparente» \Rightarrow 2 spécialisations



Pont / Bridge (3/5)



Pont / Bridge (4/5)



- Appelé aussi «*handle*» ou «*body*»
- Intérêts
 - Découplage interface / implémentation
 - Pas de lien permanent entre les deux
 - Augmentation de l'extensibilité
 - Deux hiérarchies séparées: composant et implémenteur
 - Masquer totalement l'implémentation
 - Plus d'attributs déclarés dans le composant
- Relations avec d'autres patrons
 - Fabrique abstraite
 - Peut être utilisée pour construire un pont
 - Adaptateur
 - Utilisation *a posteriori* (contrairement au pont)

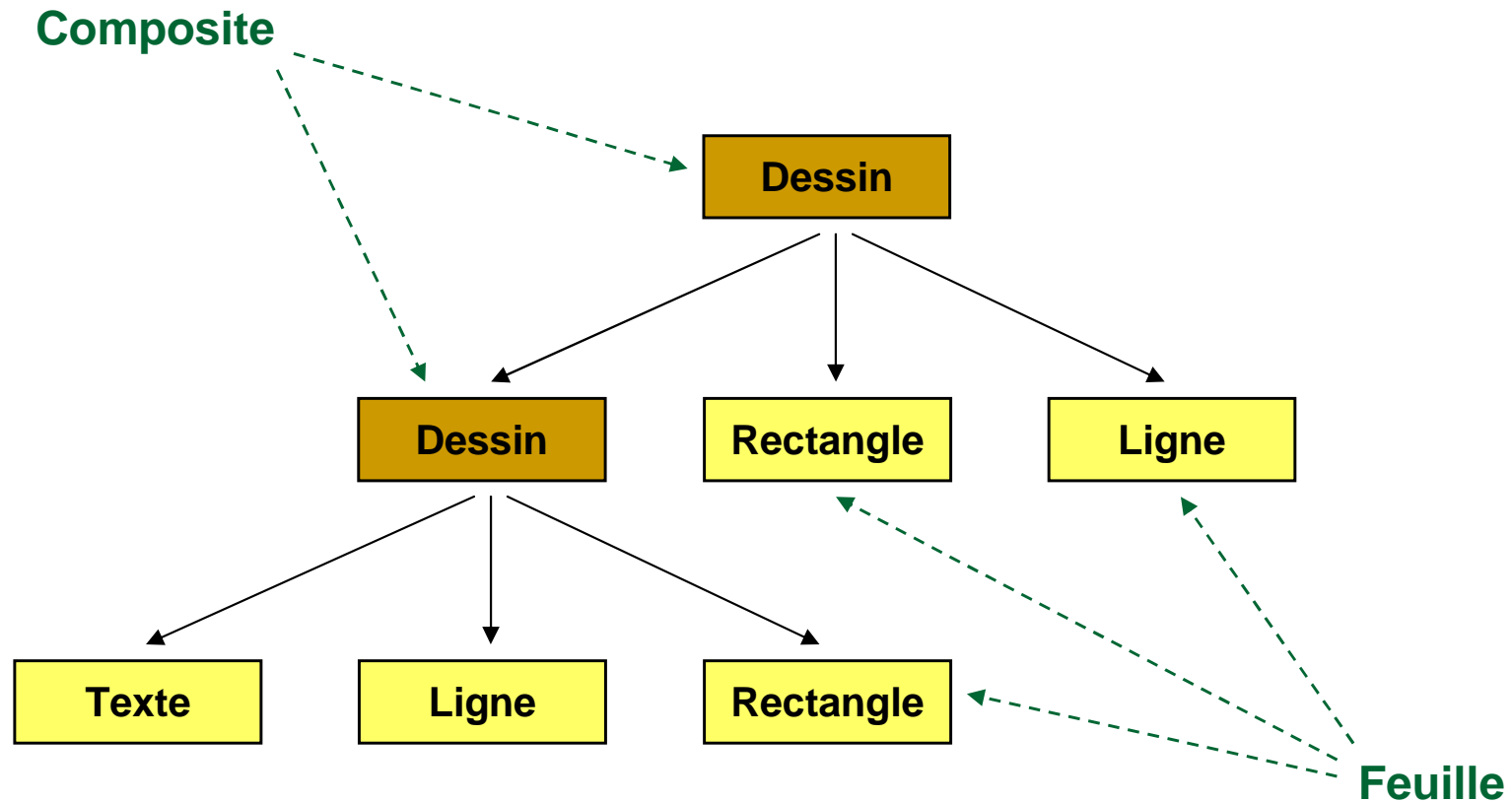
- Objectif
 - Composer des objets sous forme arborescente
 - Objet individuel ou composition traités de la même manière

- Principe
 - Un objet est composé d'autres objets
 - Ces objets peuvent également être des agrégats d'objets

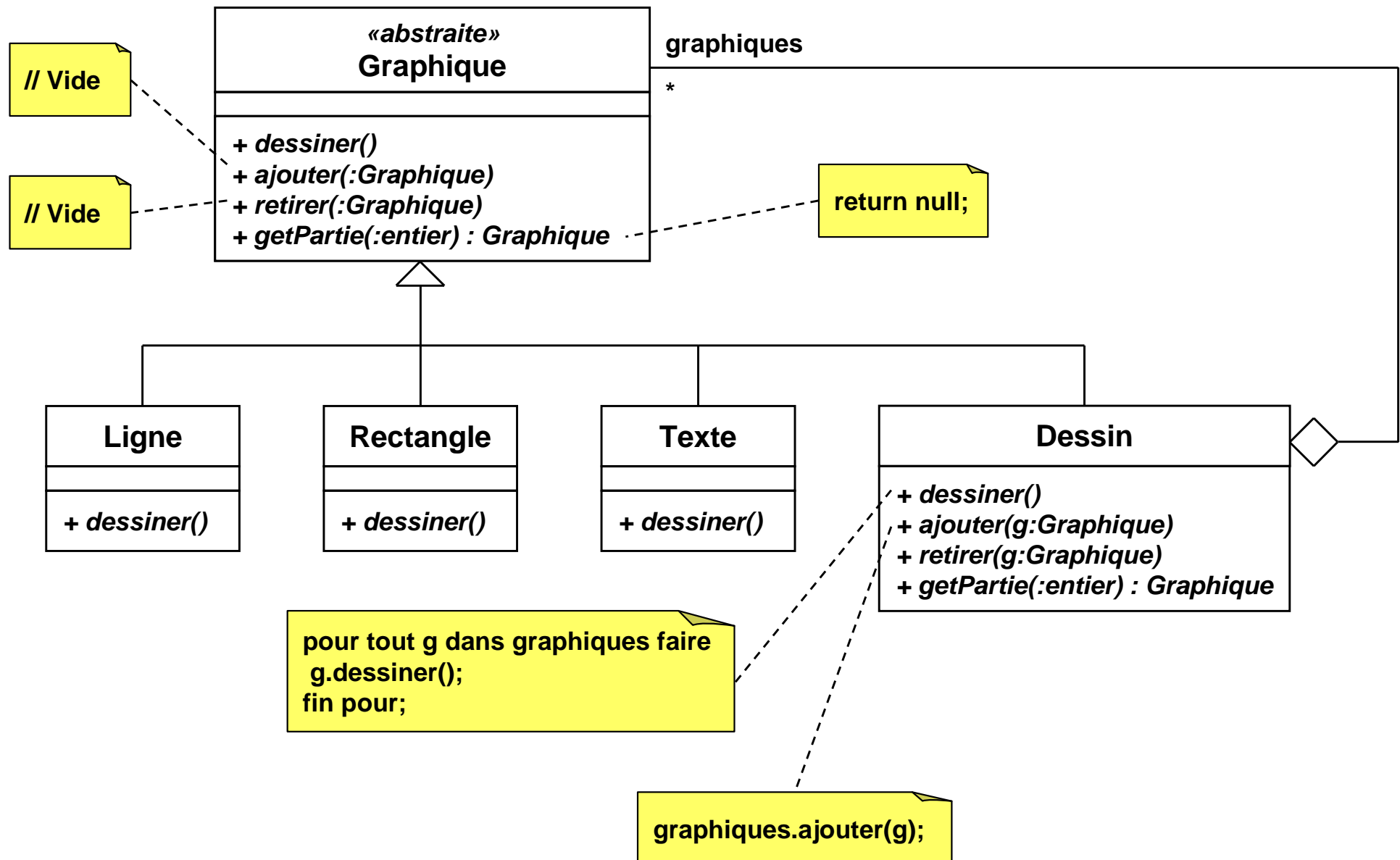
⇒ Récursivité dans la composition

- Motivation
 - Schéma/dessin composé d'objets graphiques
 - Hiérarchie d'héritage des objets graphiques
 - Un objet graphique peut être un groupement d'objets graphiques

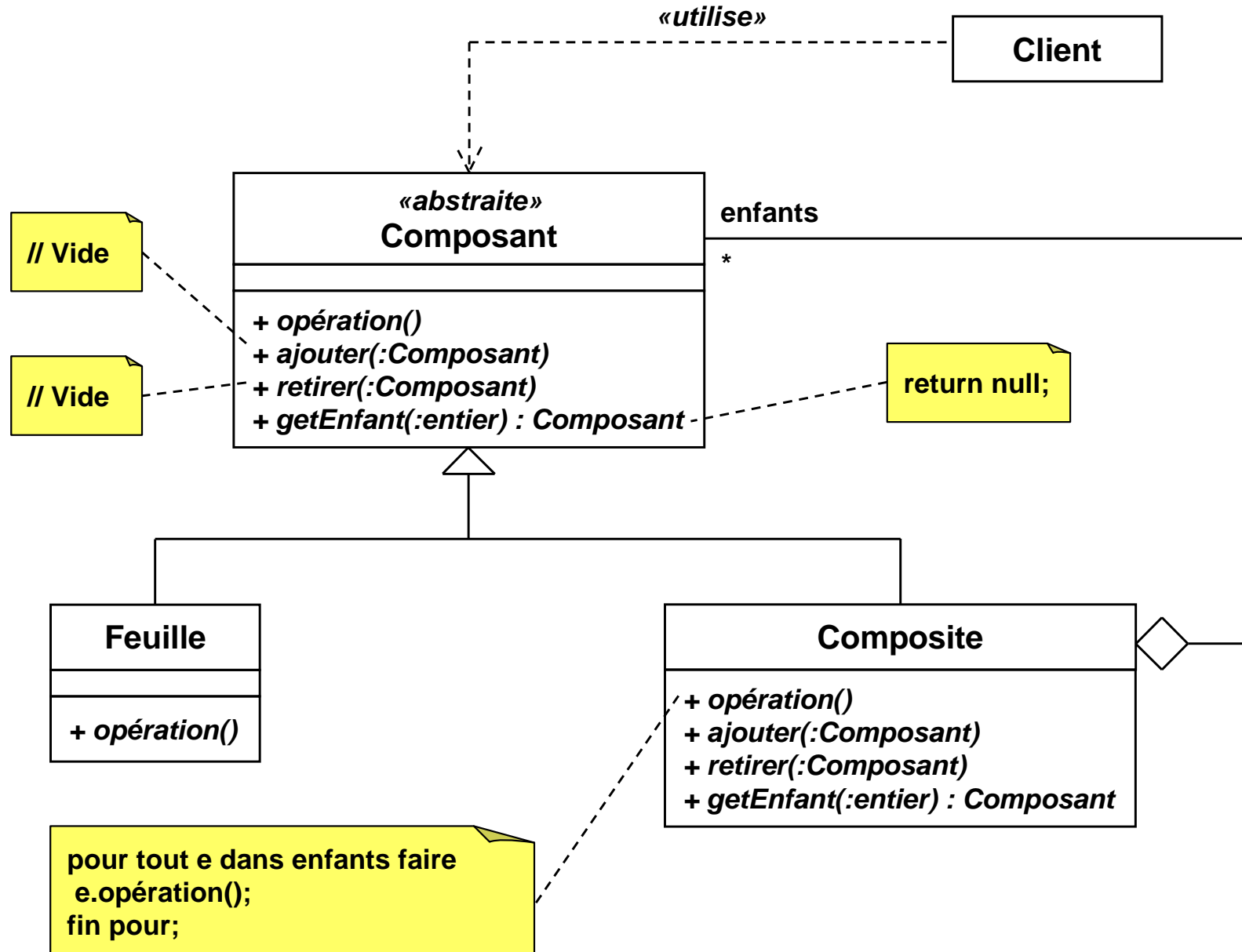
Composite / Composite (2/6)



Composite / Composite (3/6)



Composite / Composite (4/6)



■ Intérêts

- ❑ Le client fait abstraction de la classe réelle des composants
- ❑ S'il peut manipuler un objet simple, il peut manipuler un agrégat
- ❑ L'ajout d'un nouveau type de composant est très simple
 - Sans modification, le client saura le manipuler
 - Sans modification, il pourra être ajouté dans un composite

■ Implémentation

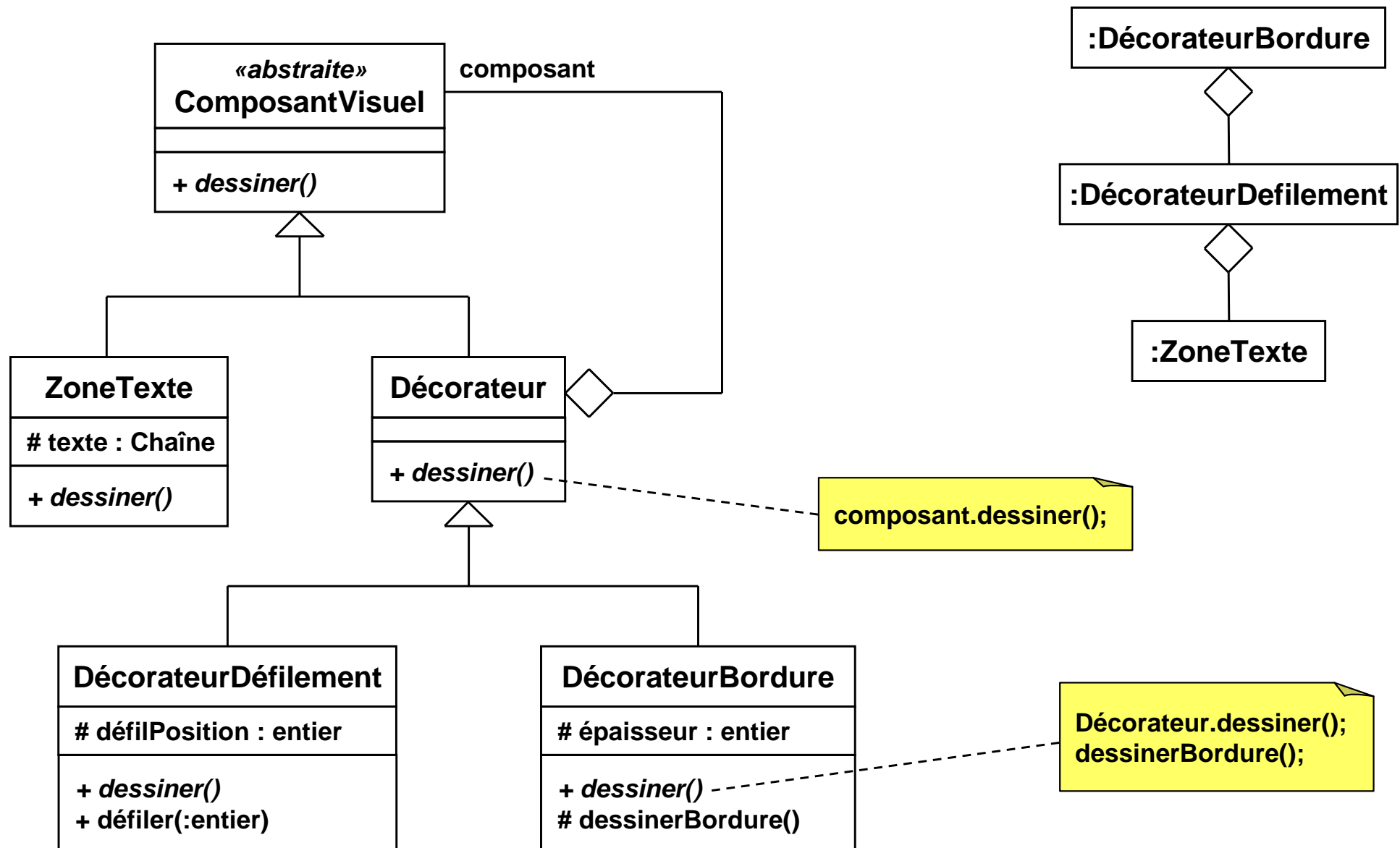
- ❑ Référence au parent ?
 - Pour faciliter certaines manipulations, l'enfant peut connaître son parent
 - Mais, plus délicat si l'enfant fait partie de plusieurs composites

- Implémentation
 - ❑ L'interface du composant peut avoir tendance à « gonfler »
 - Tendance à tout faire passer par la classe « **Composant** »
 - Pour gérer les méthodes spécifiques
 - ❑ Solution 1: Méthodes abstraites dans la classe « **Composant** »
 - ❑ Solution 2: Reconnaissance dynamique de type et conversion
 - ❑ Suppression d'un composite
 - Enfants supprimés, détachés ou rattachés au parent ?
- Relations avec d'autres patrons
 - ❑ Décorateur
 - Implémentation sous forme de composite
 - ❑ Itérateur
 - Utilisé pour parcourir les composants
 - ❑ Visiteur
 - Utilisé pour appliquer une opération à tous les composants

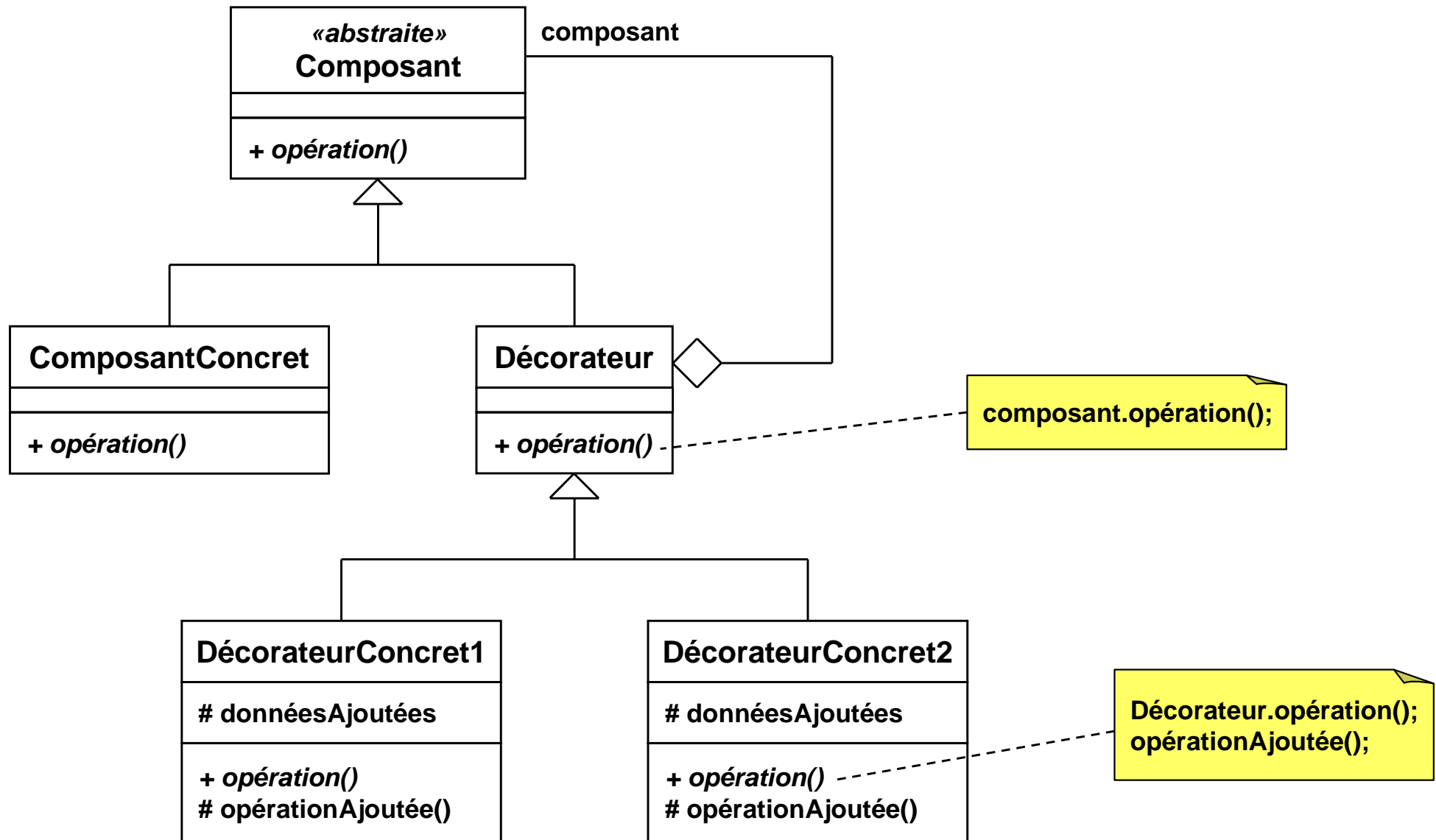
Décorateur / *Decorator* (1/5)

- Objectif
 - ❑ Ajouter dynamiquement des fonctionnalités à un objet
 - ❑ Alternative à l'héritage pour étendre les fonctionnalités
- Principe
 - ❑ Le «décorateur» agrège le composant qu'il adapte
 - ❑ Fournit la même interface de base que le composant
 - ❑ Il est donc manipulé comme le composant
- Motivation
 - ❑ Ajout de fonctionnalités à un composant graphique
 - ❑ Eviter l'héritage (car hiérarchie trop complexe)
 - ❑ Exemple: zone de texte avec bordure et barre de défilement

Décorateur / Decorator (2/5)



Décorateur / Decorator (3/5)

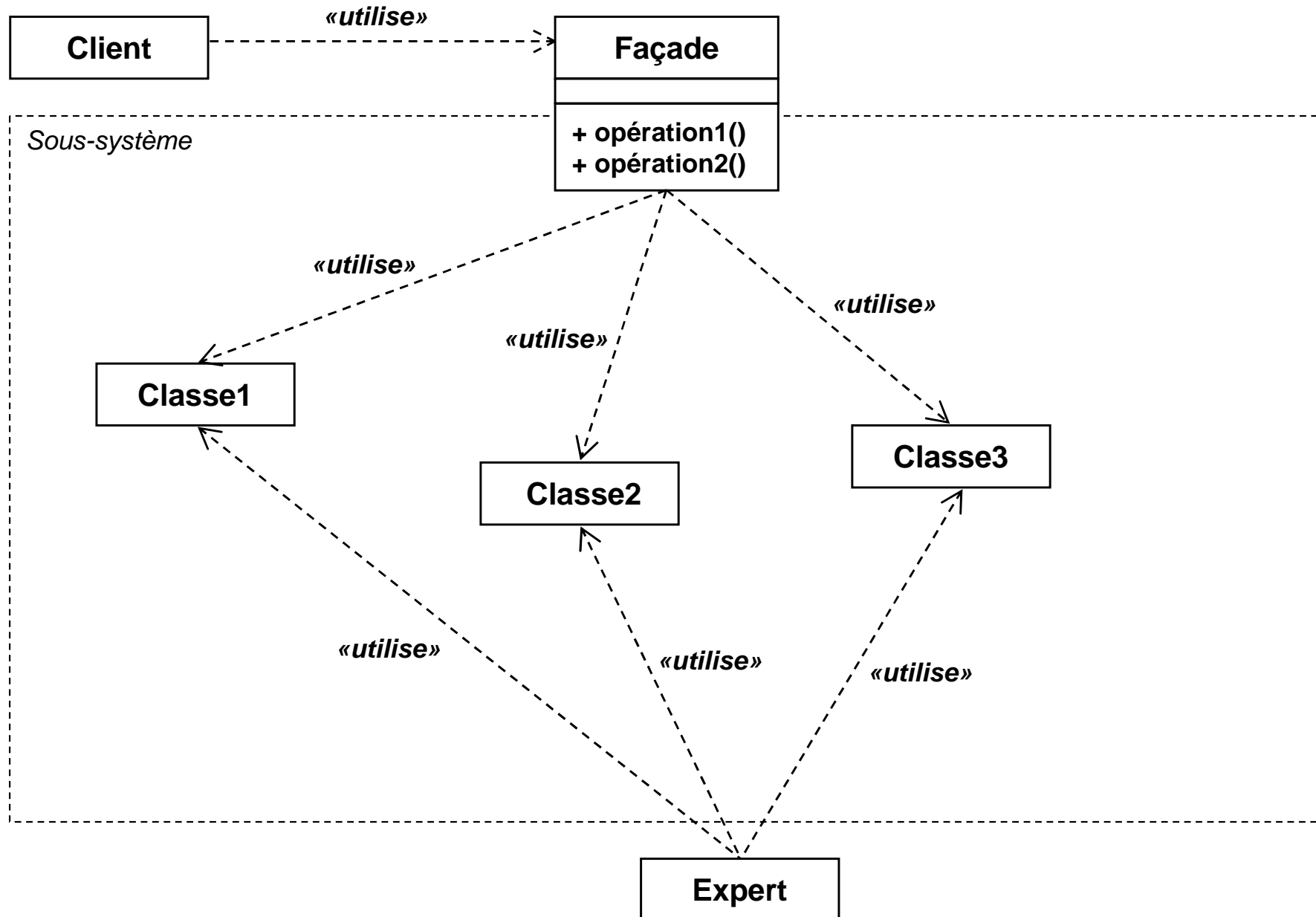


- Appelé aussi «*wrapper*»
- Intérêts
 - Evite l'extension par héritage
 - Ajout dynamique de fonctionnalités
 - Ajout individualisé (un seul objet est touché)
 - L'héritage pourrait conduire à une hiérarchie lourde
 - Exemple de la zone de texte
 - 3 héritages sont nécessaires (bordure, défilement, les deux)
 - Extension de la zone de texte \Rightarrow Extension des 3 classes
 - Mais le décorateur ajoute un objet à chaque décoration

- Relations avec d'autres patrons
 - Adaptateur
 - Similaires, mais l'adaptateur modifie l'interface
 - Composite
 - Utilisation «dégénérée» (1 seul enfant) du patron composite
 - Stratégie
 - Similaires: ils changent les fonctionnalités
 - Décorateur: ajoute des fonctionnalités par agrégation
 - Stratégie: change l'implémentation de fonctionnalités par héritage

- Objectif
 - Découpler un sous-système de ses clients
 - Fournir une interface unifiée pour l'ensemble des composants
 - Fournir une interface de plus haut niveau pour faciliter son utilisation
- Principe
 - Une interface «simplifiée» est proposée: la «façade»
 - Elle connaît les détails du sous-système
 - Le client envoie ses requêtes à la façade
 - La façade délègue les requêtes aux composants du sous-système
- Motivation
 - Bibliothèque complexe, avec beaucoup d'interfaces
 - Complexité nécessaire pour des clients experts
 - Mais inutile pour une majorité de clients
 - Objectif
 - Garder la puissance de la bibliothèque
 - Tout en fournissant une interface simplifiée

Façade / Facade (2/3)



- Intérêts
 - Découple le sous-système de ses clients
 - Un seul point d'entrée
 - Laisse la liberté au client d'utiliser le jeu d'interfaces bas niveau
 - Nécessaire pour des utilisateurs experts
 - Permet d'utiliser toute la puissance du sous-système
- Relations avec d'autres patrons
 - Fabrique abstraite
 - Permet d'assurer une construction cohérente d'objets du sous-système
 - Médiateur
 - Similaires, intermédiaires qui masquent des composants
 - Mais le but du médiateur est de centralisé / abstraire des communications
 - Singleton
 - Souvent, un seul objet façade par programme

■ Objectif

- ❑ Partager des instances pour éviter un nombre trop important

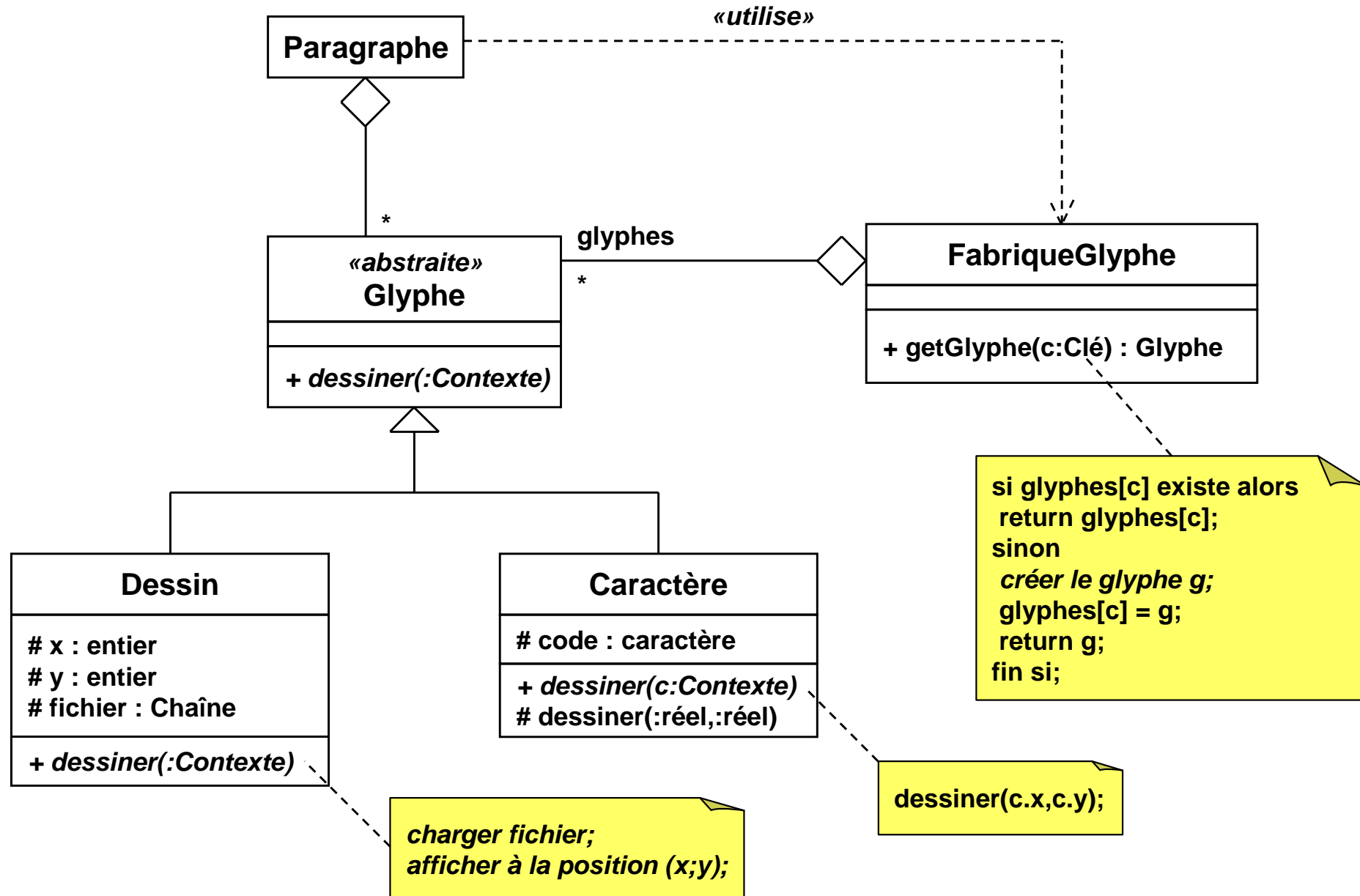
■ Principe

- ❑ Séparation de l'état d'un objet en deux parties
 - Etat intrinsèque: indépendant du contexte
 - Etat extrinsèque: dépendant du contexte
- ❑ Etat intrinsèque stocké dans l'objet
- ❑ Etat extrinsèque fourni en paramètre par le client

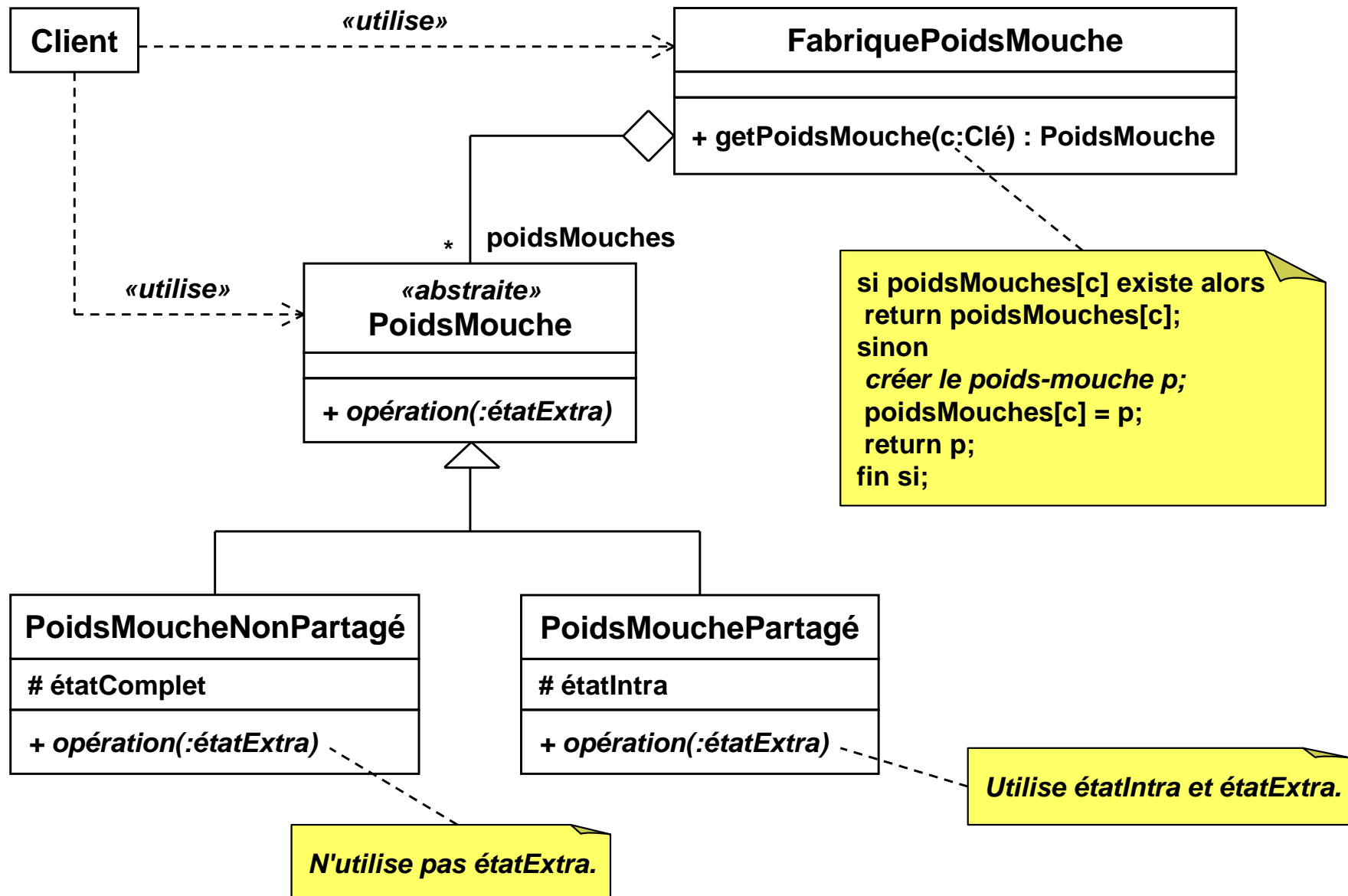
■ Motivation

- ❑ Représentation des caractères dans un traitement de texte
- ❑ Modéliser chaque caractère comme un objet
- ❑ Mais éviter d'avoir effectivement un objet par caractère

Poids-mouche / *Flyweight* (2/4)



Poids-mouche / *Flyweight* (3/4)



■ Intérêts

- ❑ Evite la duplication inutile de données
 - Etat intrinsèque jamais dupliqué
 - Etat extrinsèque calculé ou mémorisé
- ❑ Mais génère un surcoût à l'exécution
 - Lié à la transmission de l'état extrinsèque

■ Relations avec d'autres patrons

- ❑ Composite
 - Peuvent être combinés
 - Pour obtenir une arborescence avec feuilles partagées

■ Intérêts

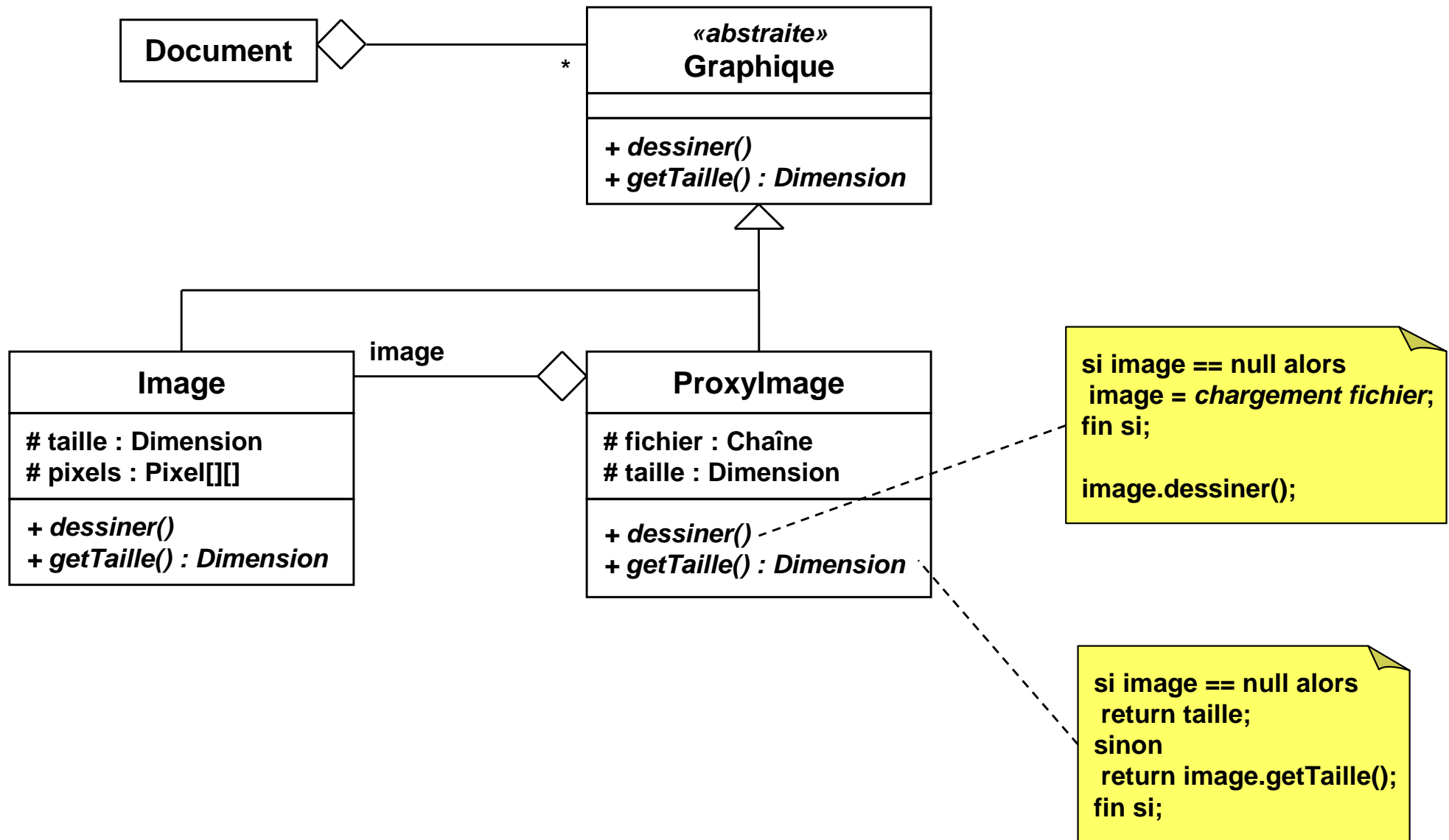
- ❑ Evite la duplication inutile de données
 - Etat intrinsèque jamais dupliqué
 - Etat extrinsèque calculé ou mémorisé
- ❑ Mais génère un surcoût à l'exécution
 - Lié à la transmission de l'état extrinsèque

■ Relations avec d'autres patrons

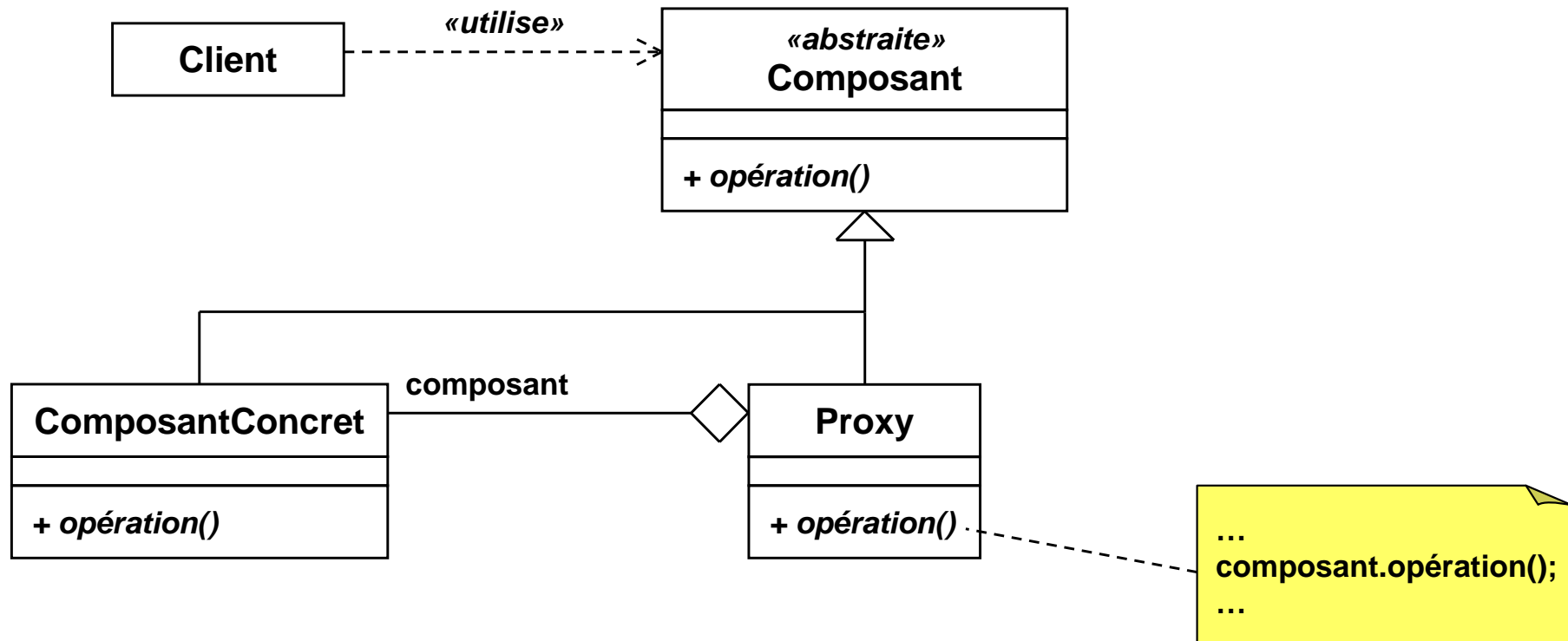
- ❑ Composite
 - Peuvent être combinés
 - Pour obtenir une arborescence avec feuilles partagées

- Objectif
 - ❑ Fournir un substitut, un intermédiaire, pour accéder à un objet
 - ❑ Permettre ainsi de contrôler l'accès
- Principe
 - ❑ Le substitut, le «proxy», possède la même interface que l'objet
 - ❑ Lorsqu'il reçoit un message, il le transmet à l'objet
 - ❑ Il peut effectuer un contrôle sur le message
 - Refuser de le retransmettre
 - Différer la retransmission
 - Altérer le message
- Motivation
 - ❑ Différer la création d'un objet car elle est coûteuse
 - ❑ Exemple: chargement d'un document avec des images
 - Différer la lecture des images au moment où celles-ci sont visibles

Proxy / Proxy (2/4)



Proxy / Proxy (3/4)



- Appelé aussi «*surrogate*» (substitut)
- Relations avec d'autres patrons
 - Adaptateur
 - Similaires, mais l'adaptateur change l'interface de l'objet
 - Décorateur
 - Similaires, mais des buts différents
 - Décorateur: ajouter des fonctionnalités
 - Proxy: contrôler les accès
- Intérêts
 - Abstraction de l'accès à un objet
 - Niveau d'indirection supplémentaire
 - Permet une représentation locale d'un objet distant
 - Autre zone mémoire, sur disque ou réseau
 - Permet des optimisations d'exécution des méthodes
 - Technique de cache
 - Création différée («*lazy*»)