

Patrons de création (PARTIE VII - Patrons de conception)

Bruno Bachelet
Christophe Duhamel
Luc Touraille

- Abstraction du processus de création
 - ❑ Indépendance du type réel
 - ❑ Indépendance de l'initialisation
 - ❑ Indépendance de la composition
- Niveau classe
 - ❑ Utilisation de l'héritage
- Niveau objet
 - ❑ Délégation de l'instanciation
- Utiles pour la création d'objets par composition
 - ❑ Une tendance actuelle des systèmes logiciels
 - ❑ *Component-Based Development*
 - ❑ Introduit de la flexibilité dans l'assemblage

- (Méthode) Fabrique / *Factory Method*
 - ❑ Déléguer la création d'un objet
- Fabrique abstraite / *Abstract Factory*
 - ❑ Créer une famille d'objets cohérents
- Monteur / *Builder*
 - ❑ Séparer la construction d'un objet complexe de sa représentation
- Prototype / *Prototype*
 - ❑ Créer un objet par clonage d'une instance modèle
- Singleton / *Singleton*
 - ❑ Garantir une seule instance pour une classe

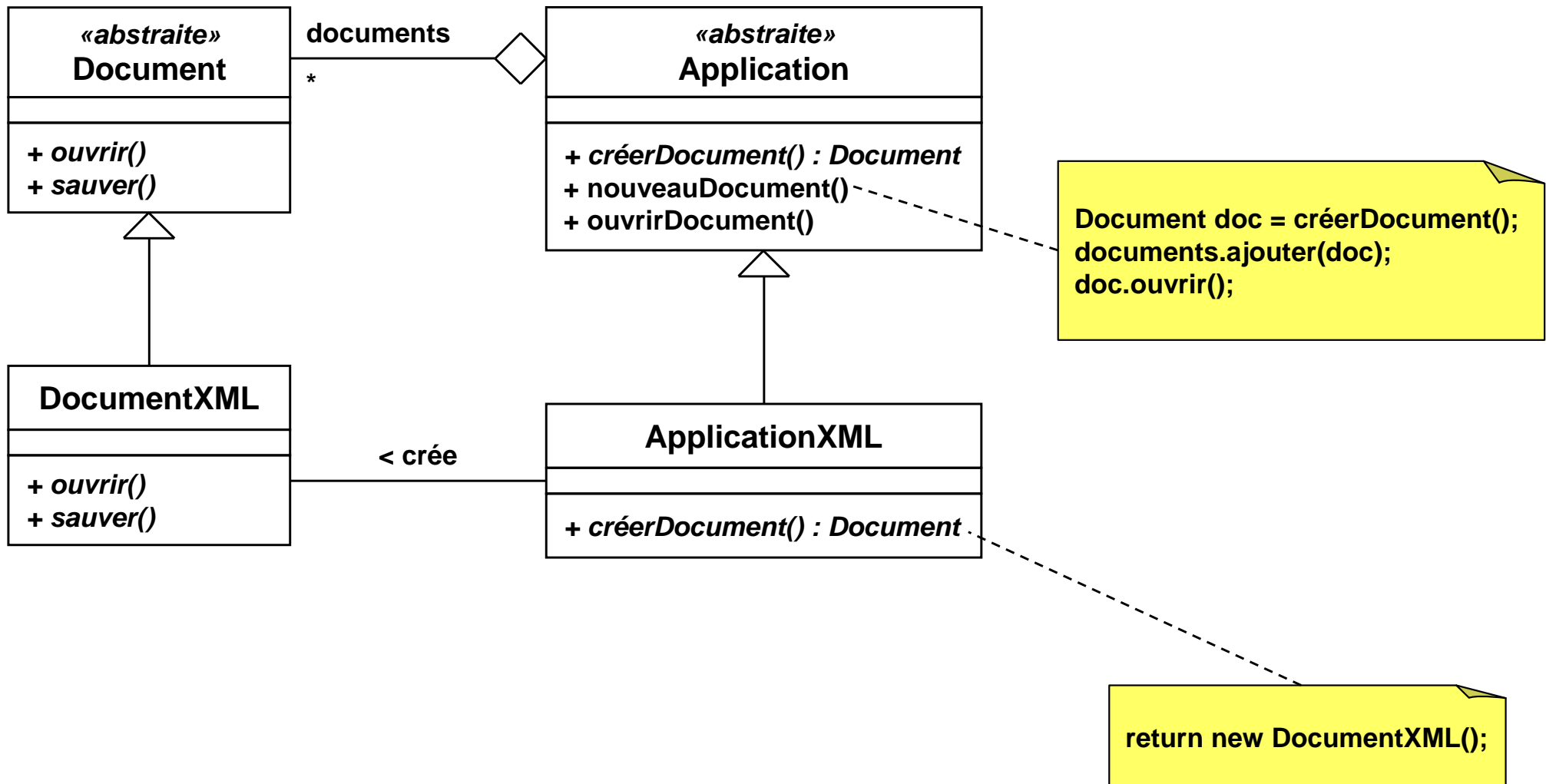
(Méthode) Fabrique / *Factory Method* (1/4)

- Objectif
 - Déléguer la création d'un objet

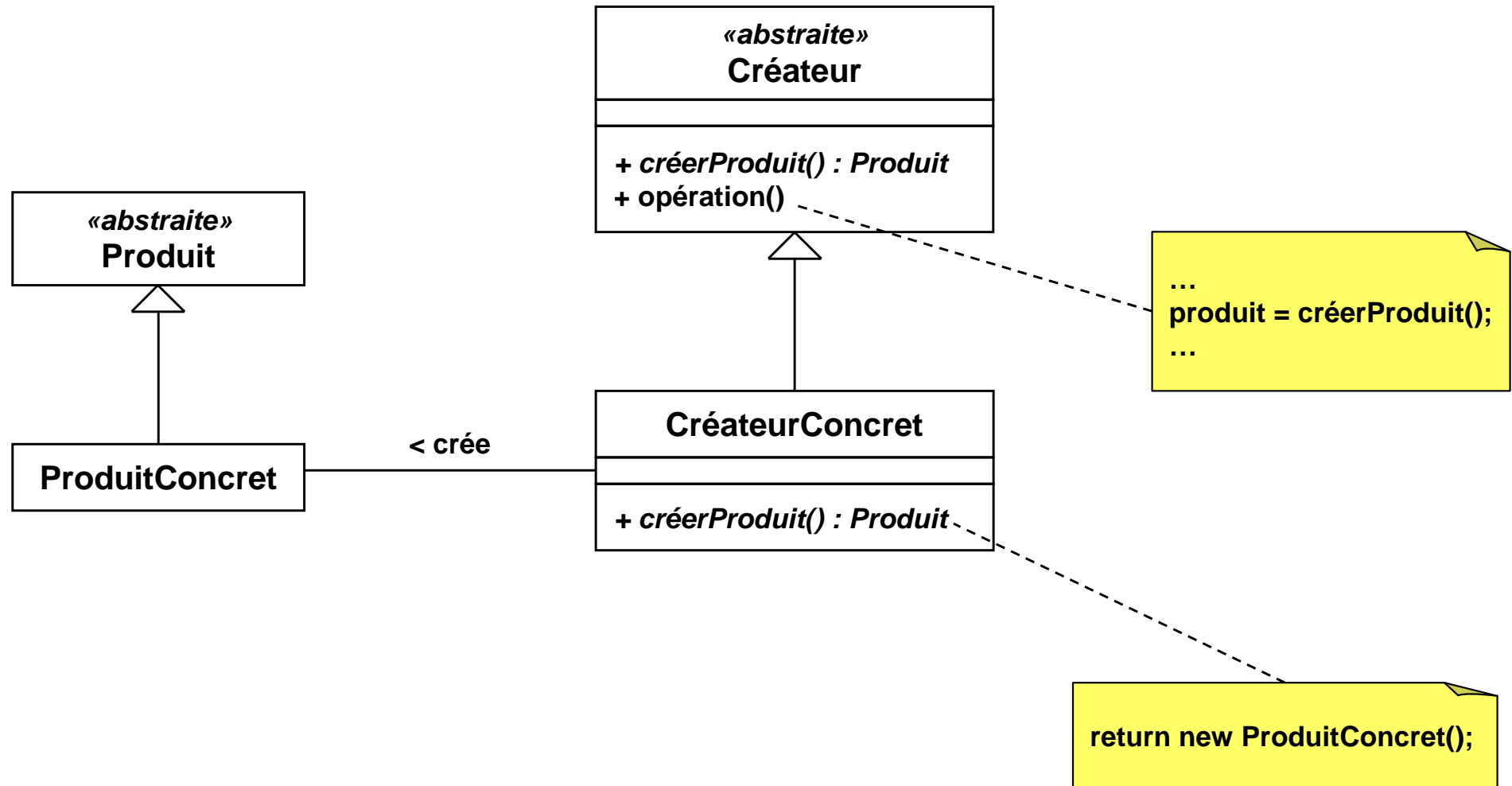
- Principe
 - Définir une interface pour créer un objet \Rightarrow le «créateur»
 - Le processus de création peut être changé par héritage
 - Le client demande au créateur de lui fournir une instance
 - Le client ne connaît que l'interface de l'objet
 - Seul le créateur connaît la classe réelle de l'objet

- Motivation
 - Application qui manipule des documents de types différents

(Méthode) Fabrique / *Factory Method* (2/4)



(Méthode) Fabrique / *Factory Method* (3/4)



(Méthode) Fabrique / *Factory Method* (4/4)

- Méthode «**créerProduit**» = méthode «fabrique»
- Appelé aussi «constructeur virtuel»
- Intérêt
 - Isolation de la classe concrète
 - Créateur responsable de la création
 - La méthode fabrique peut choisir le type d'objet à créer
 - Exemple: **créerProduit(type:Chaîne)**
- Relations avec d'autres patrons
 - Fabrique abstraite
 - Utilise la fabrique dans son implémentation
 - Méthode patron
 - La méthode fabrique est une méthode patron

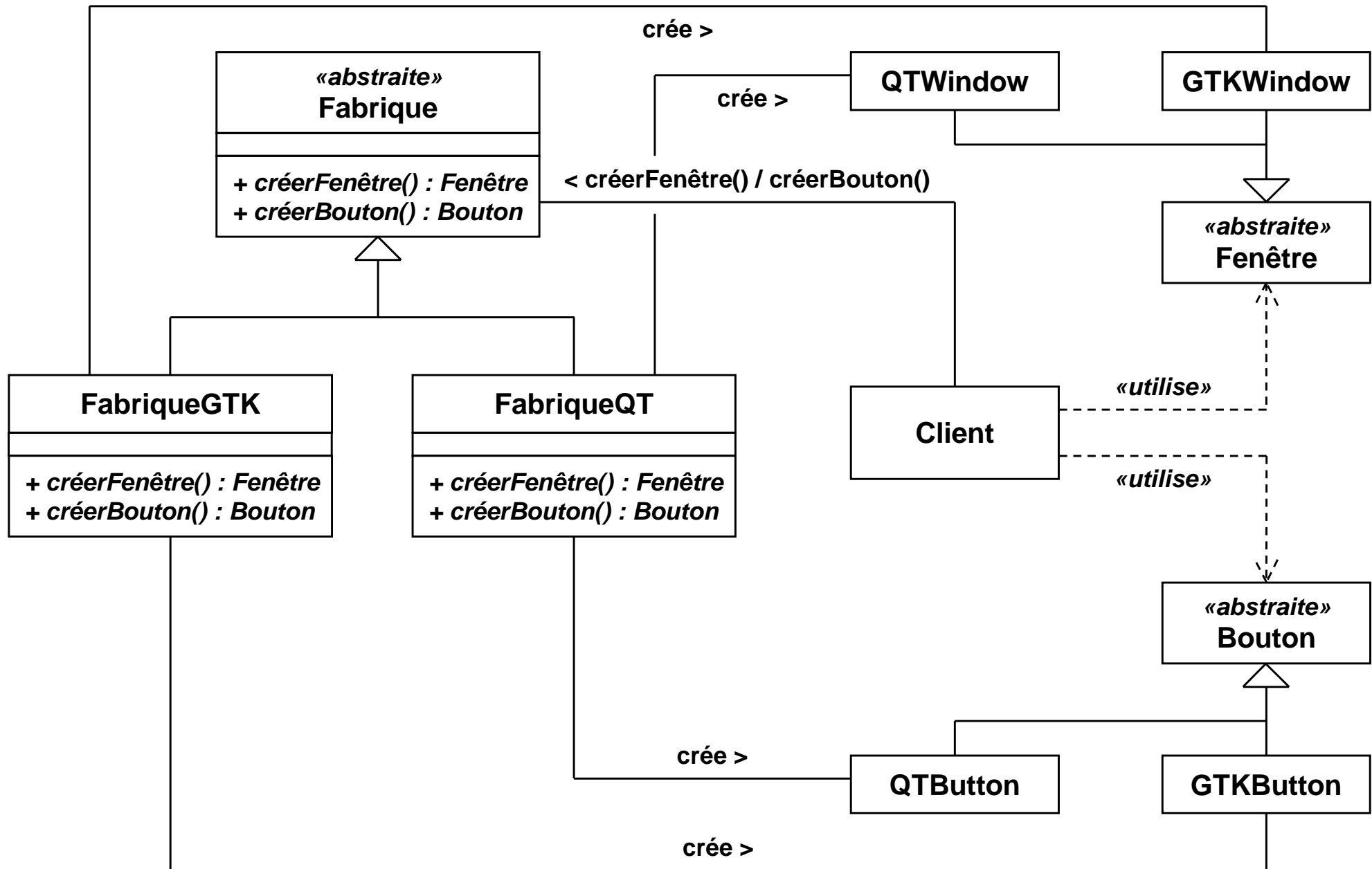
Fabrique abstraite / *Abstract Factory* (1/4)

- Objectif
 - Créer une famille d'objets cohérents
 - Des objets de classes différentes sont à créer
 - Mais les classes doivent être cohérentes entre elles

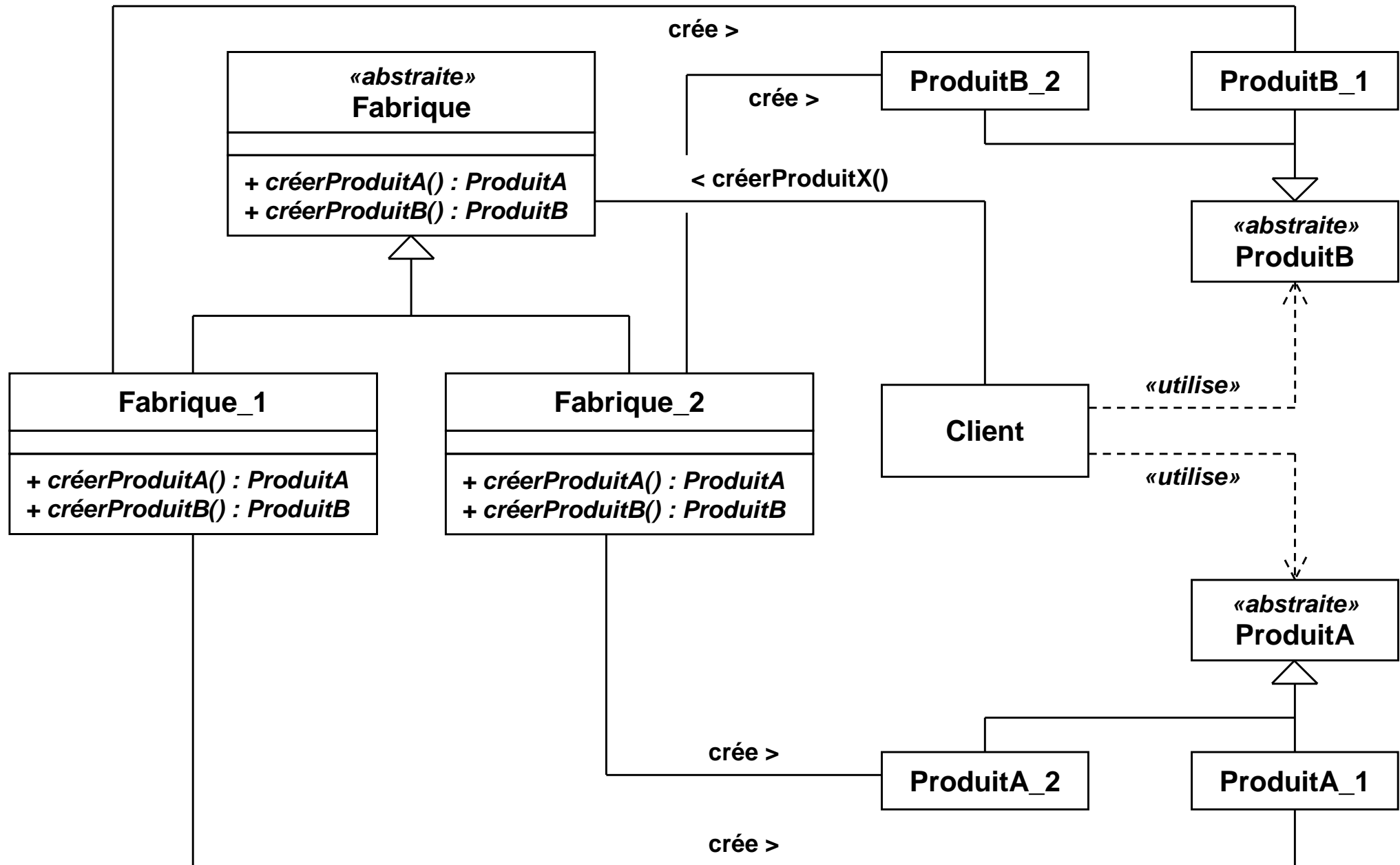
- Principe
 - Fournir une interface pour créer une famille d'objets \Rightarrow la «fabrique»
 - Le client demande à la fabrique de lui fournir des instances
 - Le client ne connaît que les interfaces des objets
 - Seule la fabrique connaît les classes réelles des objets

- Motivation
 - Système indépendant de l'interface graphique
 - Créer des composants graphiques cohérents selon la plateforme

Fabrique abstraite / *Abstract Factory* (2/4)



Fabrique abstraite / *Abstract Factory* (3/4)



Fabrique abstraite / *Abstract Factory* (4/4)

- Appelé aussi «kit»
- Intérêts
 - Isolation des classes concrètes
 - Fabrique responsable de la création
 - Echange de famille de produits très facile
 - Remplacer la fabrique concrète par une autre
- Relations avec d'autres patrons
 - Singleton
 - Souvent, une seule instance de chaque fabrique
 - (Méthode) Fabrique
 - Une méthode fabrique par type de produit

■ Objectif

- ❑ Séparer la construction d'un objet complexe de sa représentation
- ❑ Même processus de construction mais représentations différentes

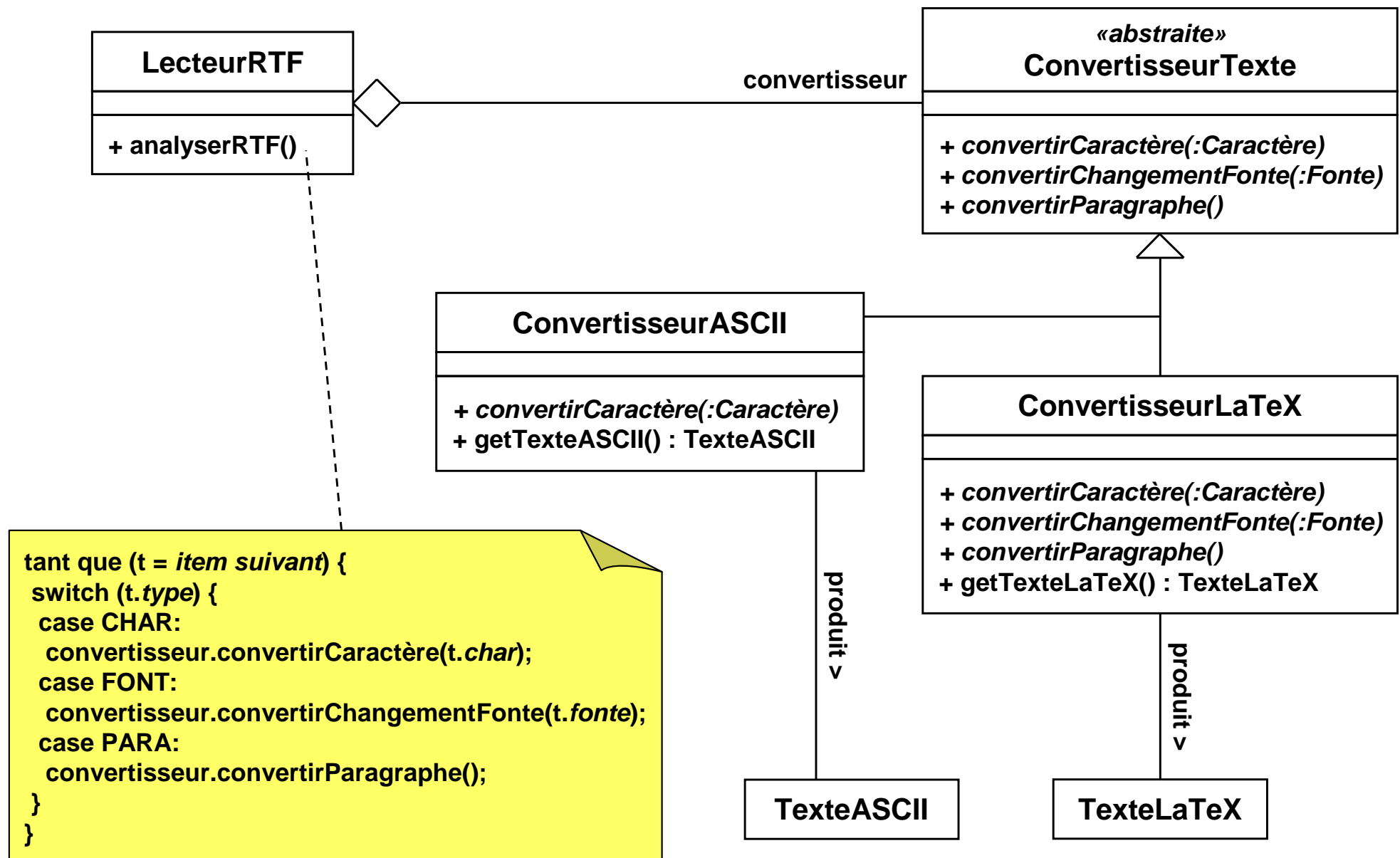
■ Principe

- ❑ Un «directeur» construit une structure complexe
- ❑ Il délègue la création des parties à un «monteur»
- ❑ Le directeur ne connaît que l'interface des parties
- ❑ Seul le monteur connaît la classe réelle des parties

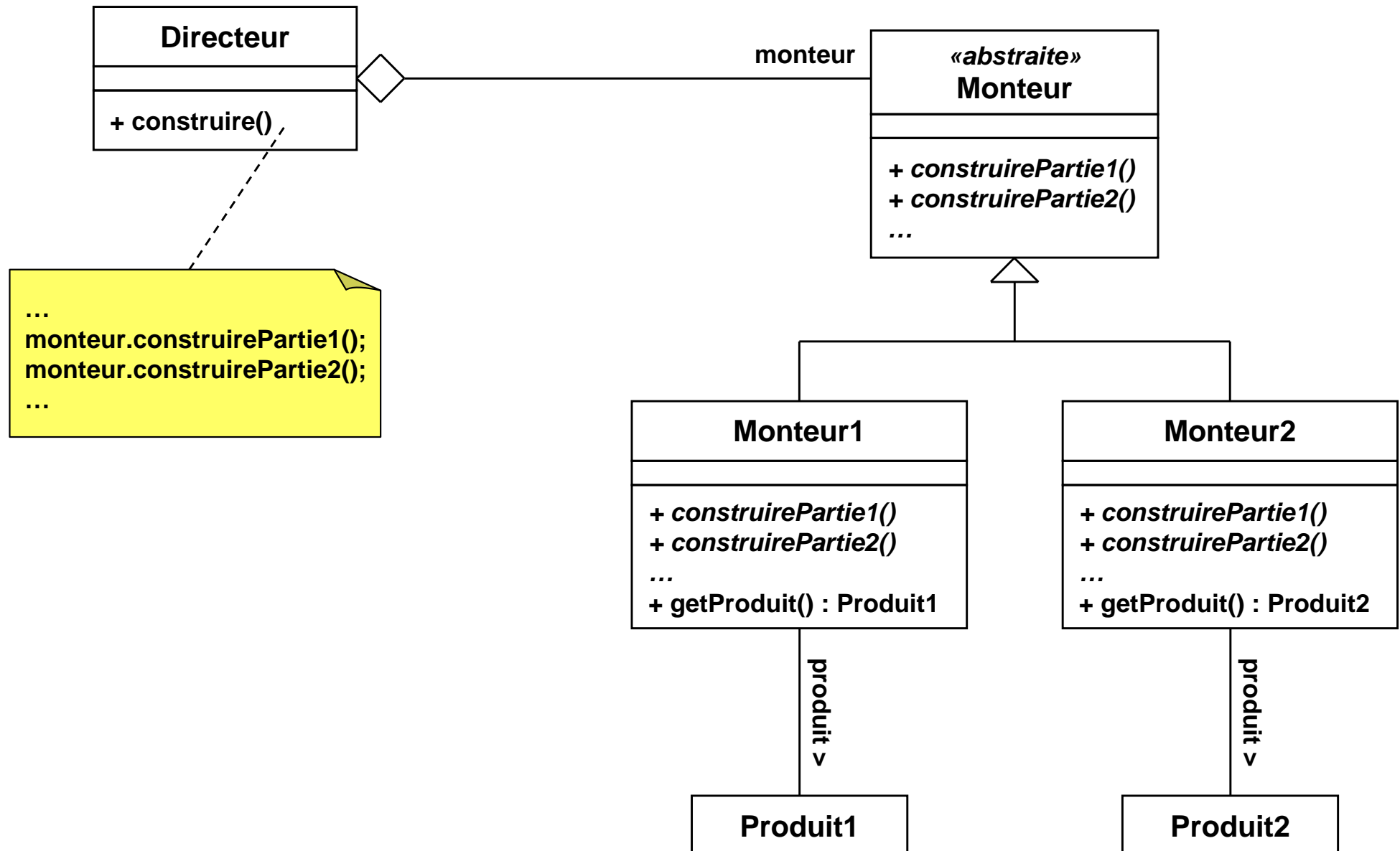
■ Motivation

- ❑ Conversion d'un format de fichier vers un format cible
- ❑ Le format cible peut changer

Monteur / Builder (2/4)



Monteur / Builder (3/4)



- Intérêts
 - Isolation de la construction et de la représentation
 - Le processus de création des parties est masqué
 - Le processus d'assemblage des parties est masqué
 - Echange de représentation d'une structure complexe facile
 - Remplacer le monteur concret par un autre
 - Malgré l'abstraction, contrôle précis du processus de création
 - Produit construit pas à pas
 - Sous la supervision du directeur
 - Accès au produit une fois le processus terminé
- Relations avec d'autres patrons
 - Fabrique abstraite
 - Fabrique abstraite = construction d'une famille de produits
 - Monteur = construction pas à pas d'un produit complexe

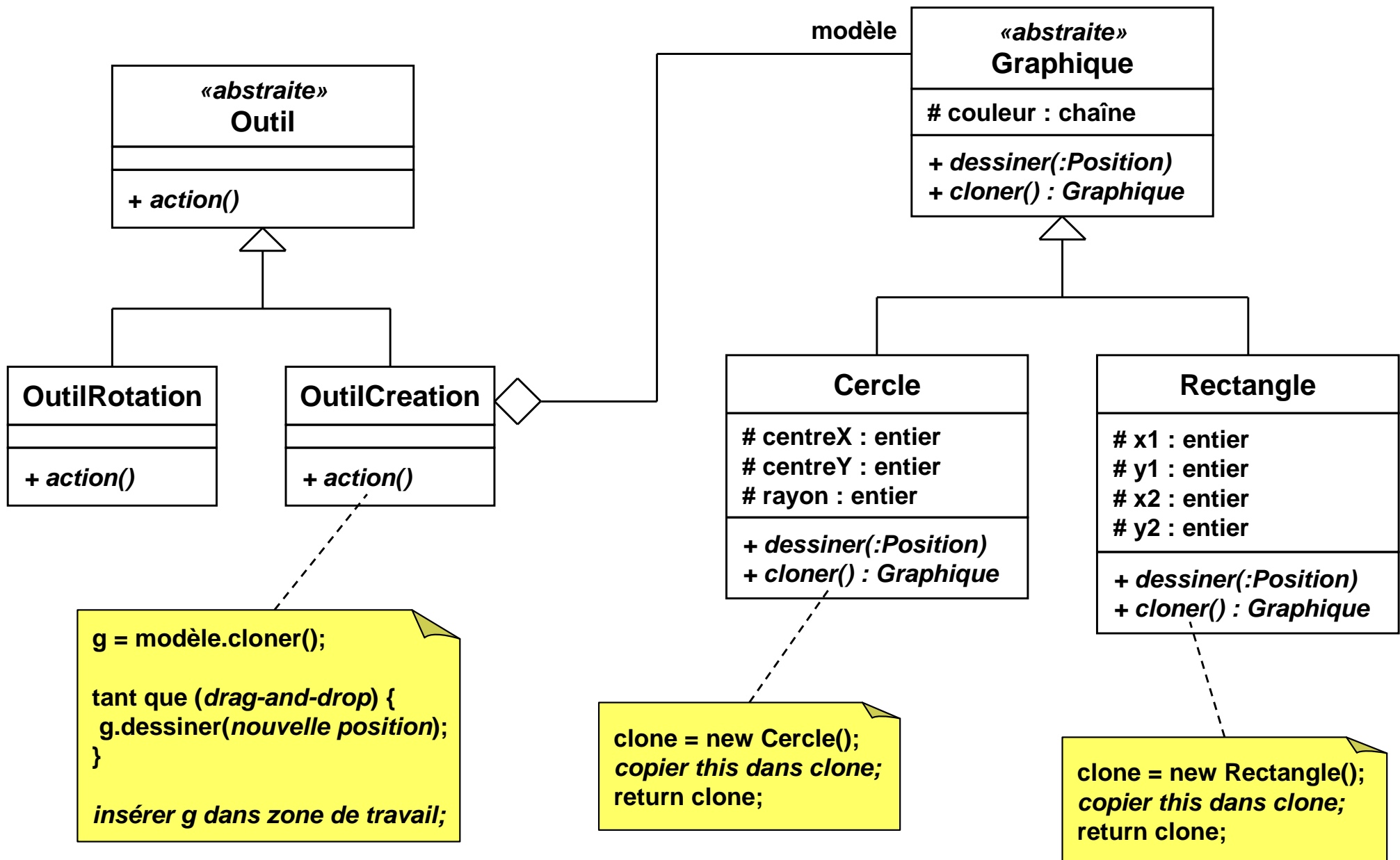
Prototype / *Prototype* (1/4)

- Objectif
 - ❑ Créer un objet par clonage d'une instance modèle
 - ❑ Le type de l'objet est déterminé par celui de l'instance modèle

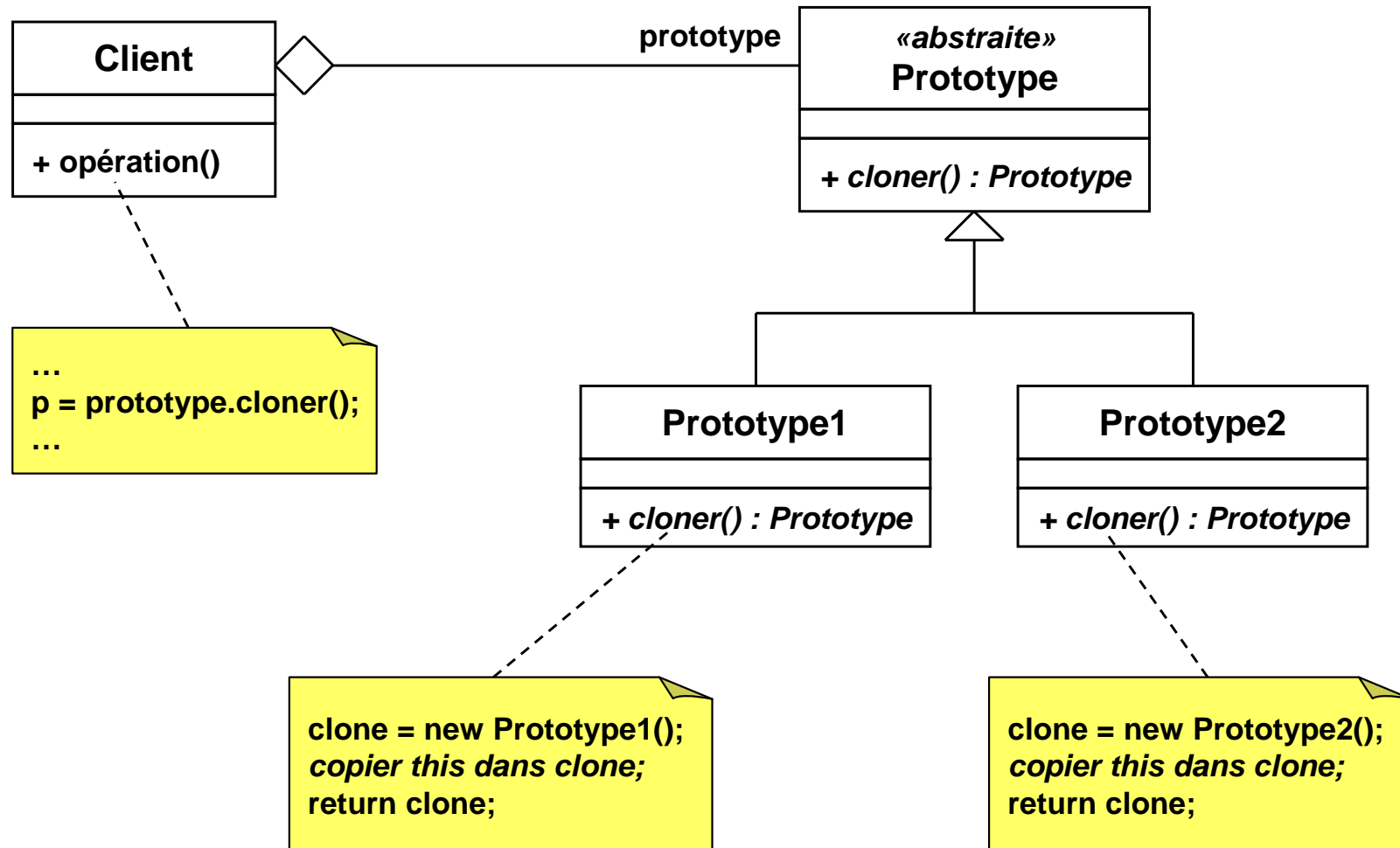
- Principe
 - ❑ Un objet «prototype» est fourni
 - ❑ Il possède une méthode de clonage
 - ❑ Le client utilise cette méthode pour obtenir une copie de l'objet

- Motivation
 - ❑ Boîte à outils: déposer des objets par *drag-and-drop*
 - ❑ Une copie du modèle est déposée sur la zone de travail

Prototype / Prototype (2/4)



Prototype / Prototype (3/4)



■ Intérêts

- ❑ Abstraction de la construction
 - 2 instances différentes \Rightarrow 2 initialisations différentes
- ❑ Utile lorsque la phase d'initialisation est coûteuse
 - Plus rapide de recopier une instance

■ Relations avec d'autres patrons

- ❑ Fabrique abstraite
 - Peut utiliser des prototypes pour créer les objets

■ En Java

- ❑ Tous les objets appartiennent à la classe «**Object**»
- ❑ Cette classe fournit une méthode «**clone**»

■ Objectif

- ❑ Garantir une seule instance pour une classe
- ❑ Fournir un point d'accès global à cette instance

■ Principe

- ❑ Masquer les constructeurs de cette classe
 - Impossibilité de créer un objet en dehors de la classe
- ❑ Fournir une méthode de classe qui retourne l'objet unique

■ Motivation

- ❑ Représentation de ressources physiques uniques
- ❑ Exemple: flux d'entrée et sortie standards

Singleton / Singleton (2/3)

■ Exemple C++

```
class Singleton {  
    private: static Singleton unique;  
  
    // Attributs du singleton  
  
    private:  
        Singleton(...) { ... }  
        Singleton(const Singleton &);  
        Singleton & operator=(const Singleton &);  
  
    public: static Singleton & getInstance()  
        { return unique; }  
  
    // Méthodes du singleton  
};  
  
Singleton Singleton::unique(...);
```

Singleton
- <u>unique</u> : Singleton - état : Etat
+ <u>getInstance()</u> : Singleton + opérations() + <u>getEtat()</u> : Etat

■ Création et copie d'un objet interdites

- ❑ Opérateurs privés

■ Seule possibilité: utiliser l'instance unique

- ❑ Autorisé: `Singleton::getInstance()`

■ Intérêts

- ❑ Contrôler la création des objets d'une classe
 - Permet notamment d'imposer le nombre d'instances
- ❑ Contrôler l'accès aux instances
 - Exemple: accès protégé pour le *multithreading*
- ❑ Fournit un espace de nommage
 - Alternative aux variables globales
 - Alternative aux fonctions
- ❑ Extension possible par héritage

■ Relations avec d'autres patrons

- ❑ Fabrique abstraite / Monteur / Prototype
 - Ils peuvent utiliser le singleton dans leur implémentation