**Prosit 2 : Ce n’est pas un ouvrage de BTP**

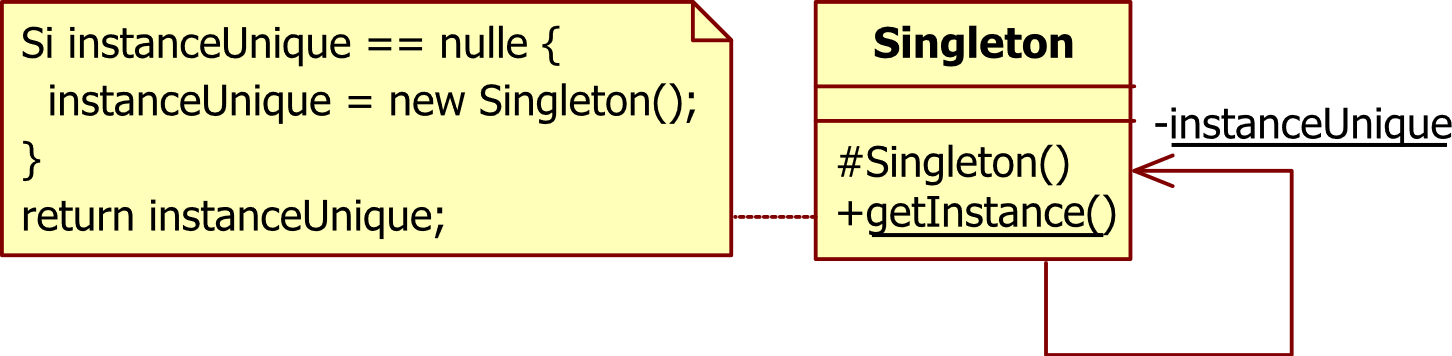
# Contexte :

2F Roaming essaie d’adopter des bonnes pratiques de développement afin de minimiser les couts.

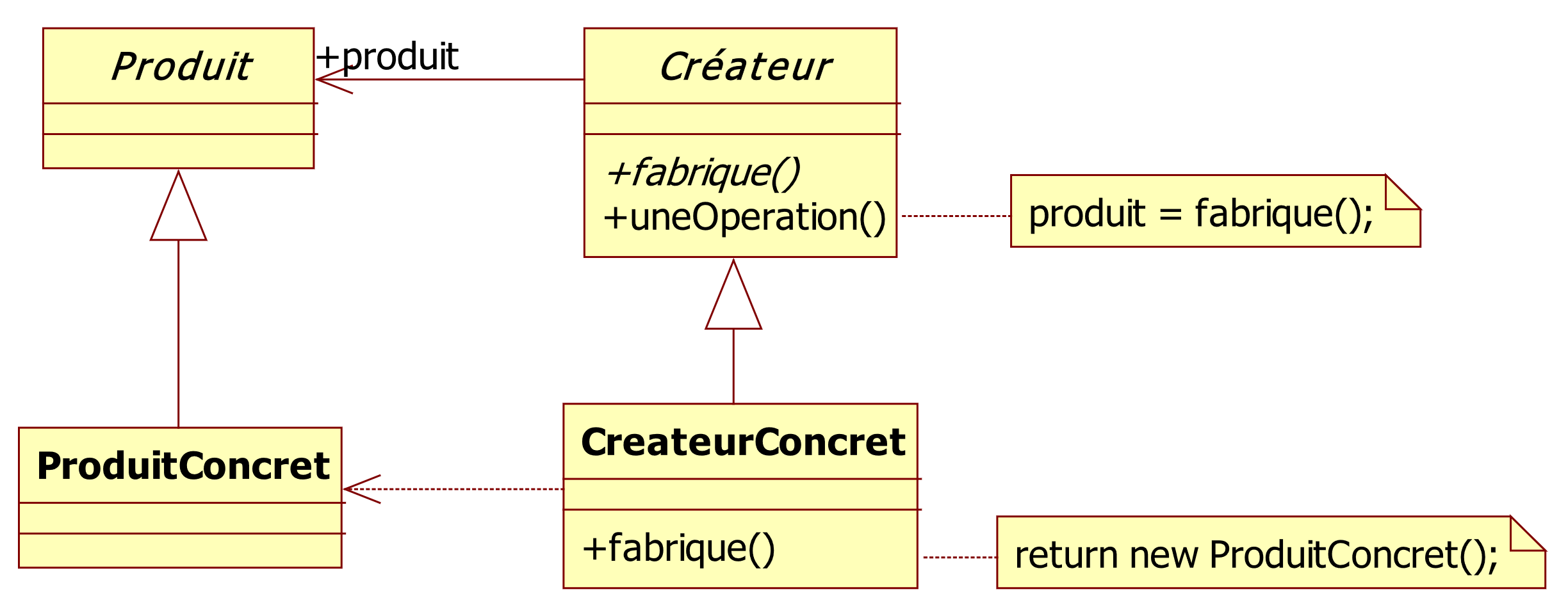
# Mot clés :

Design Pattern (DP) : En français, "Design Pattern" se traduit par "Patron de Conception". Il s'agit d'une solution réutilisable à un problème de conception fréquemment rencontré dans le développement logiciel. Les design patterns offrent des solutions structurées pour résoudre des problèmes courants et améliorer la flexibilité, la maintenabilité et la compréhension du code.

Singleton : Singleton est un patron de conception qui garantit qu'une classe n'a qu'une seule instance et fournit un point d'accès global à cette instance. Cela est souvent utilisé lorsque vous souhaitez qu'une seule instance d'une classe soit disponible dans toute l'application.

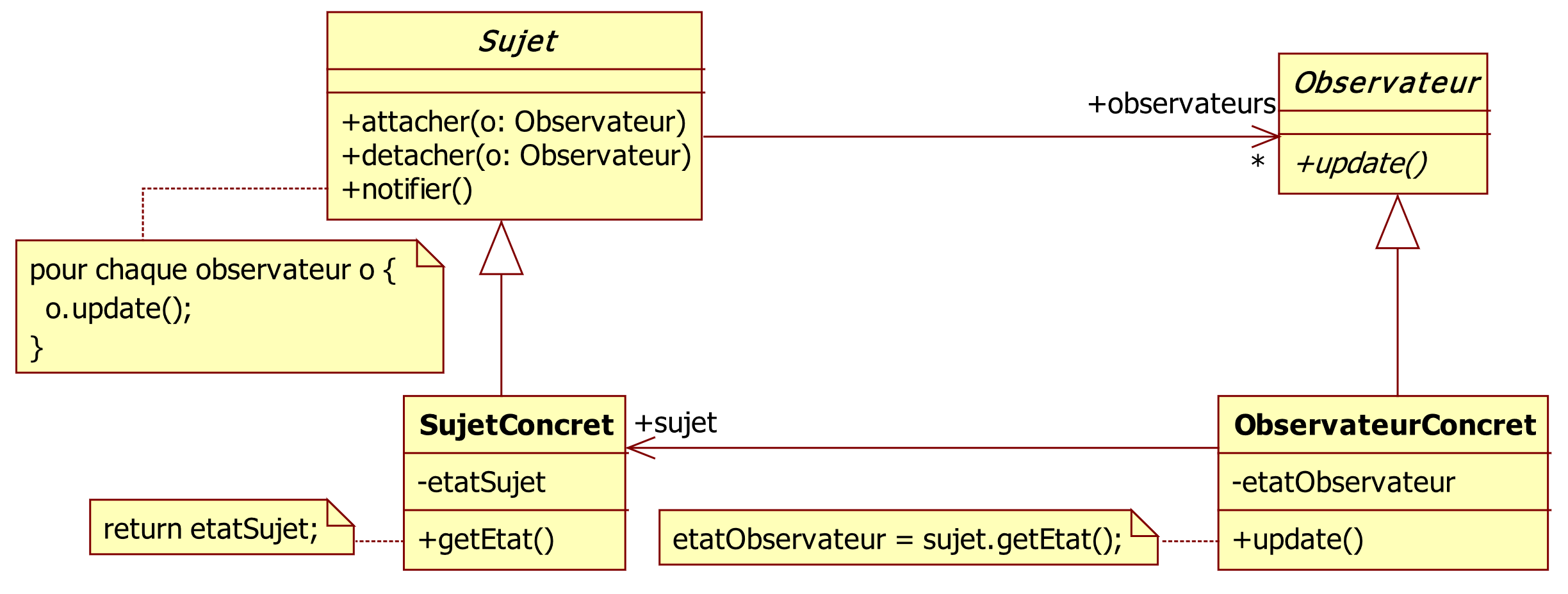


Usine : Le patron de conception "Usine" (Factory) implique la création d'un objet sans spécifier la classe exacte de l'objet qui sera créé. Il utilise une interface commune pour créer différents types d'objets.

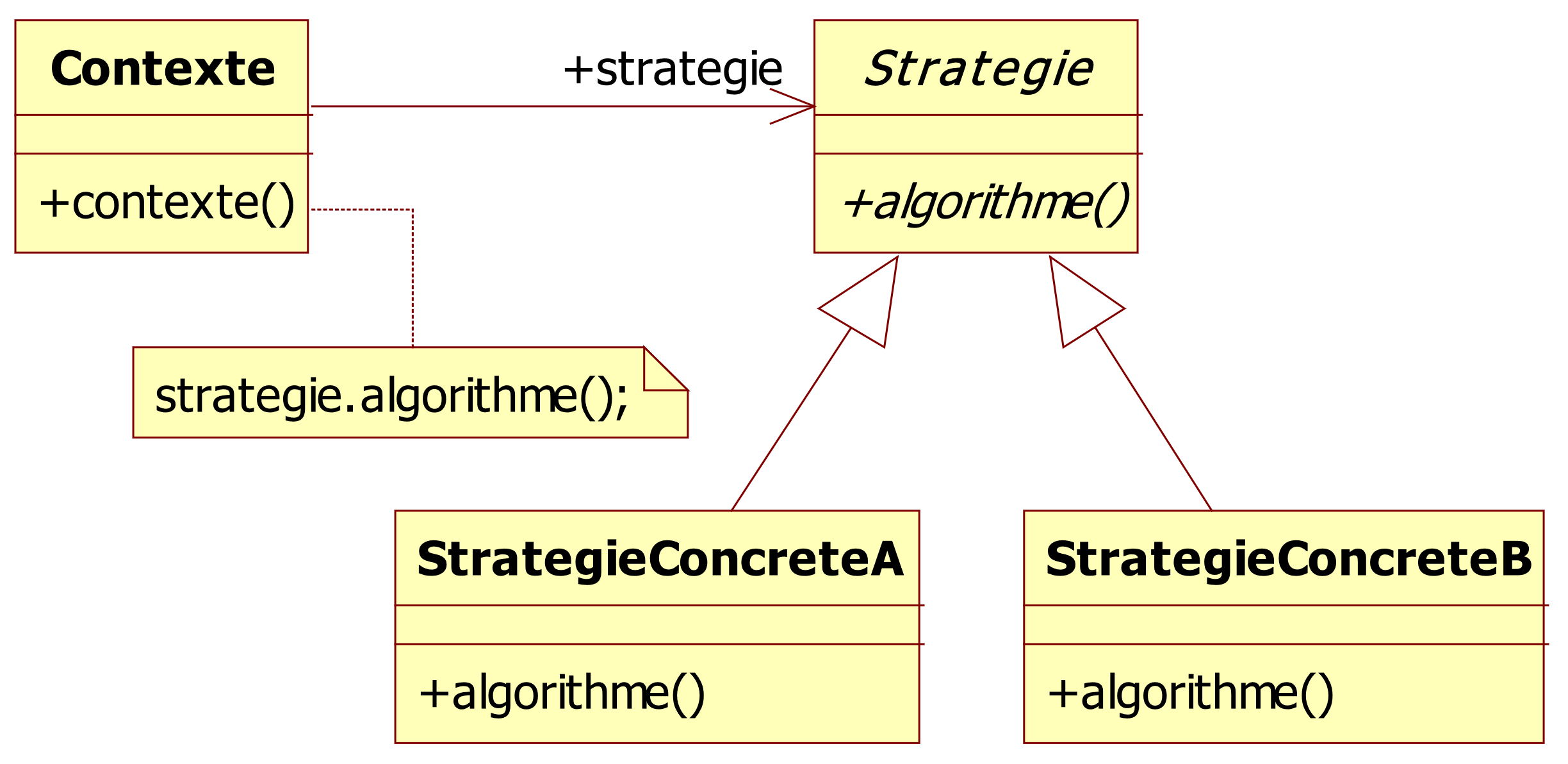


Pont : Le patron de conception "Pont" (Bridge) sépare une abstraction de son implémentation afin que les deux puissent évoluer indépendamment. Cela permet de réduire la complexité en évitant une liaison permanente entre une abstraction et son implémentation.

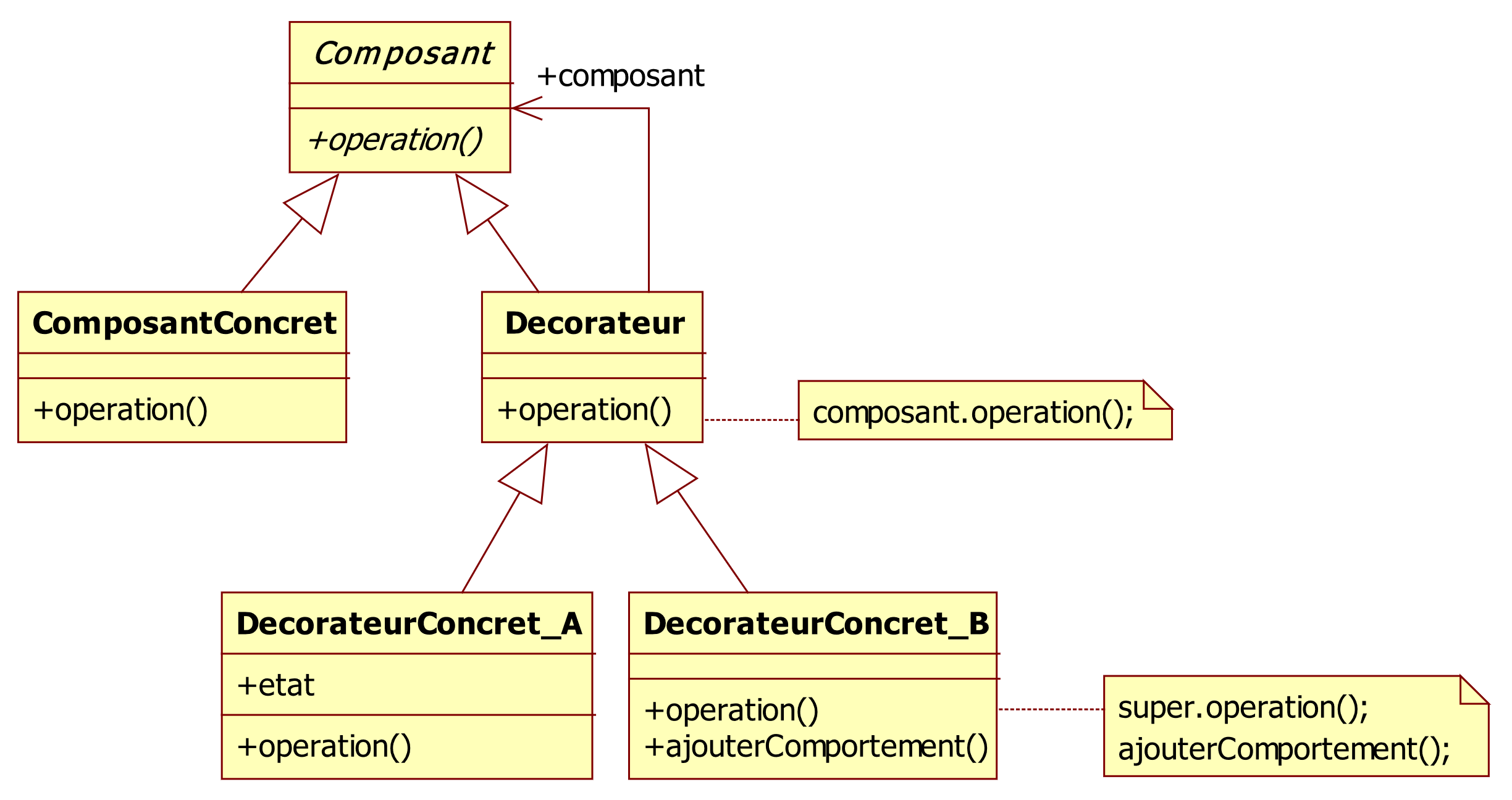
Observateur : Le patron de conception "Observateur" (Observer) définit une dépendance entre objets de manière à ce que lorsque l'état d'un objet change, tous ses dépendants soient notifiés et mis à jour automatiquement.



Stratégie : Le patron de conception "Stratégie" (Strategy) définit une famille d'algorithmes, les encapsule et les rend interchangeables. Il permet au client de choisir l'algorithme approprié à utiliser à partir d'une famille sans modifier son code.



Décorateur : Le patron de conception "Décorateur" (Decorator) permet d'attacher de nouvelles fonctionnalités à un objet en fournissant une alternative souple à l'héritage de sous-classes.



Reusable Object-Oriented : En français, "Object-Oriented" se traduit par "Orienté Objet". La programmation orientée objet (POO) est un paradigme de programmation qui utilise des objets, qui regroupent des données et les méthodes qui opèrent sur ces données, pour concevoir et organiser le code.

# Problématique :

**Quels sont les concepts à s’approprier pour diminuer les couts de développements ?**

**Quels sont les types de Design Pattern et lequel utiliser ?**

# Contraintes :

DP orienté C#

# Généralisation :

Maitriser les DP

# Livrables :

Coder les 6 DP en C#

# Pistes de solution :

Se servir des DPs pour trouver des solutions aux problèmes récurrent

Singleton, Usine, pont, Observateur, Stratégie et décorateur

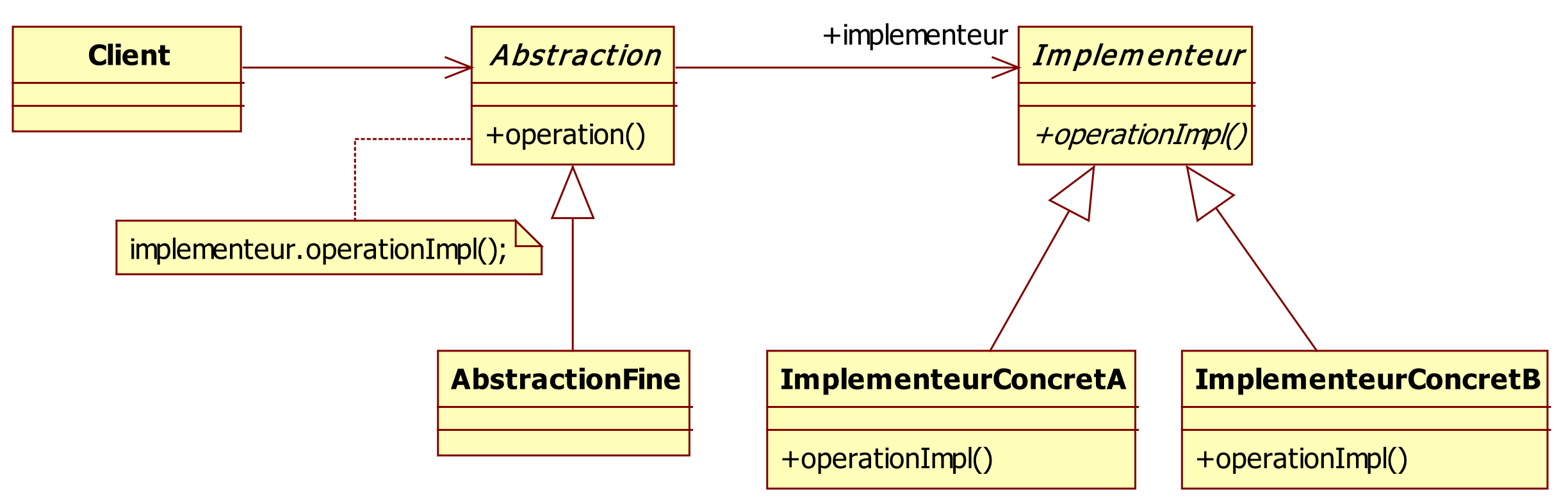
# Plan d’action :

Faire la corbeille

Recherche des DP

# Réalisation du plan d’action :

Je n’ai pas trouvé la corbeille très parlante du coup, j’ai cherché sur internet pour bien comprendre la notion de dp bridge :



Et j’ai ensuite réalisé un script c# pour Bridge et un autre pour factory, les deux sont dispos sur mon git : https://dev.azure.com/hugolaplace/Easysave/\_git/prosit2 :

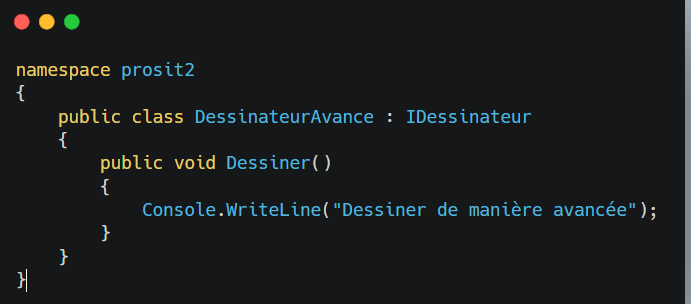
Voici celui du bridge :



Une image contenant texte, capture d’écran, Police

Description générée automatiquement

*DessinateurSimple.cs*

**

*DessinateurAvance.cs*

*Une image contenant texte, capture d’écran, Police

Description générée automatiquement*

*Forme.cs*

# Sources / Cours :

<https://www.goprod.bouhours.net/?lang=fr>

https://refactoring.guru/fr/design-patterns/bridge/csharp/example