Limiter la complexité

- Limiter la taille du code
- YAGNI : You ain't gonna need it
- Commencer par des cas simples
- Similitudes → généraliser
- Dépendances ⚠

Gérer les changements anticipés

- Besoin et contraintes changent
- Code (réduit par la qualité)

↳ localiser les changements

⚠ Ne pas tous les gérer (impossible trop coûteux !)

Casser les mauvaises dépendances :

CACHER CE QUI VA CHANGER

Principes SOLID

SRP : Une classe ne doit avoir qu'une seul raison de changer

OCP : On doit pouvoir étendre une classe sans la changer -> polymorphisme.

LSP : Les sous type doivent pouvoir être substitués à leurs super types

I.S.P : les interfaces parties publique doivent être restreintes au besoin

D.I.P : il faut dépendre d'abstractions, pas d'implémentations.

Design patterns

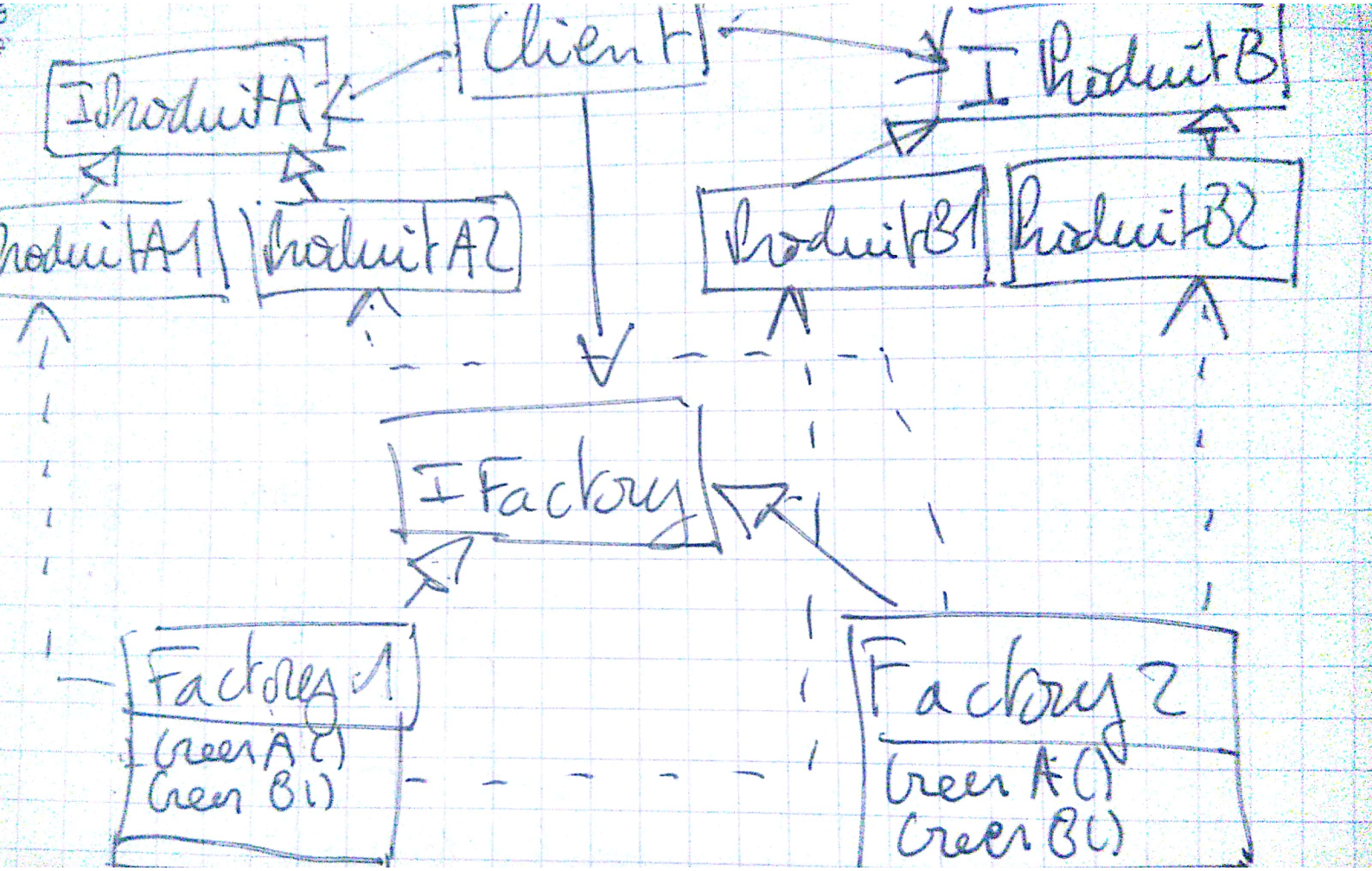
Factory: Déléguer la création des classes instrables à des classes Factorys qui implémentent une interface (injection de dépendance possible).

Abstract Factory

-> Une Interface pour créer des Familles de classes Assorties

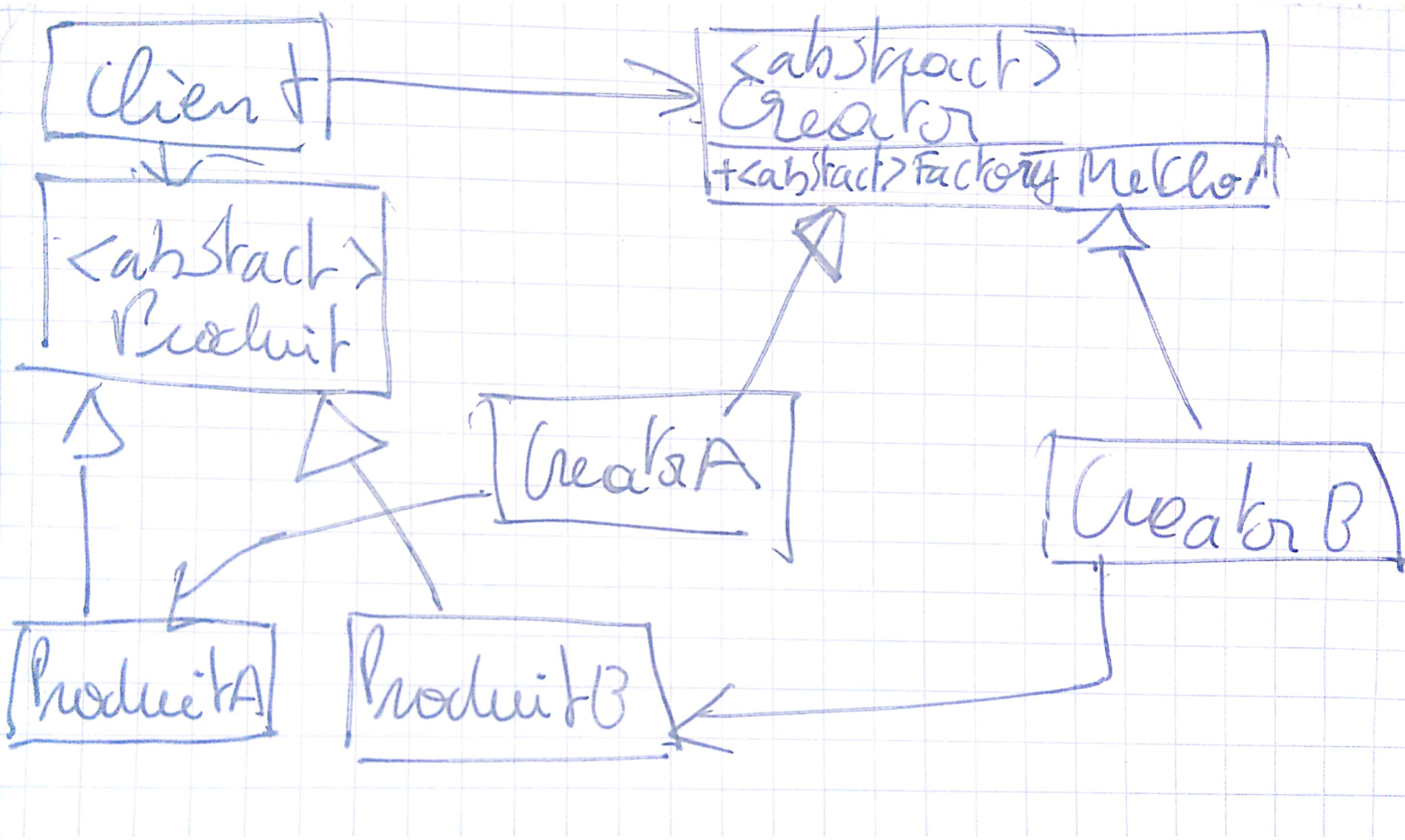
-> Contraintes d'assortiment à la création (dans l'interface)

-> Des classes Factory dédiées à la création des familles de classe.



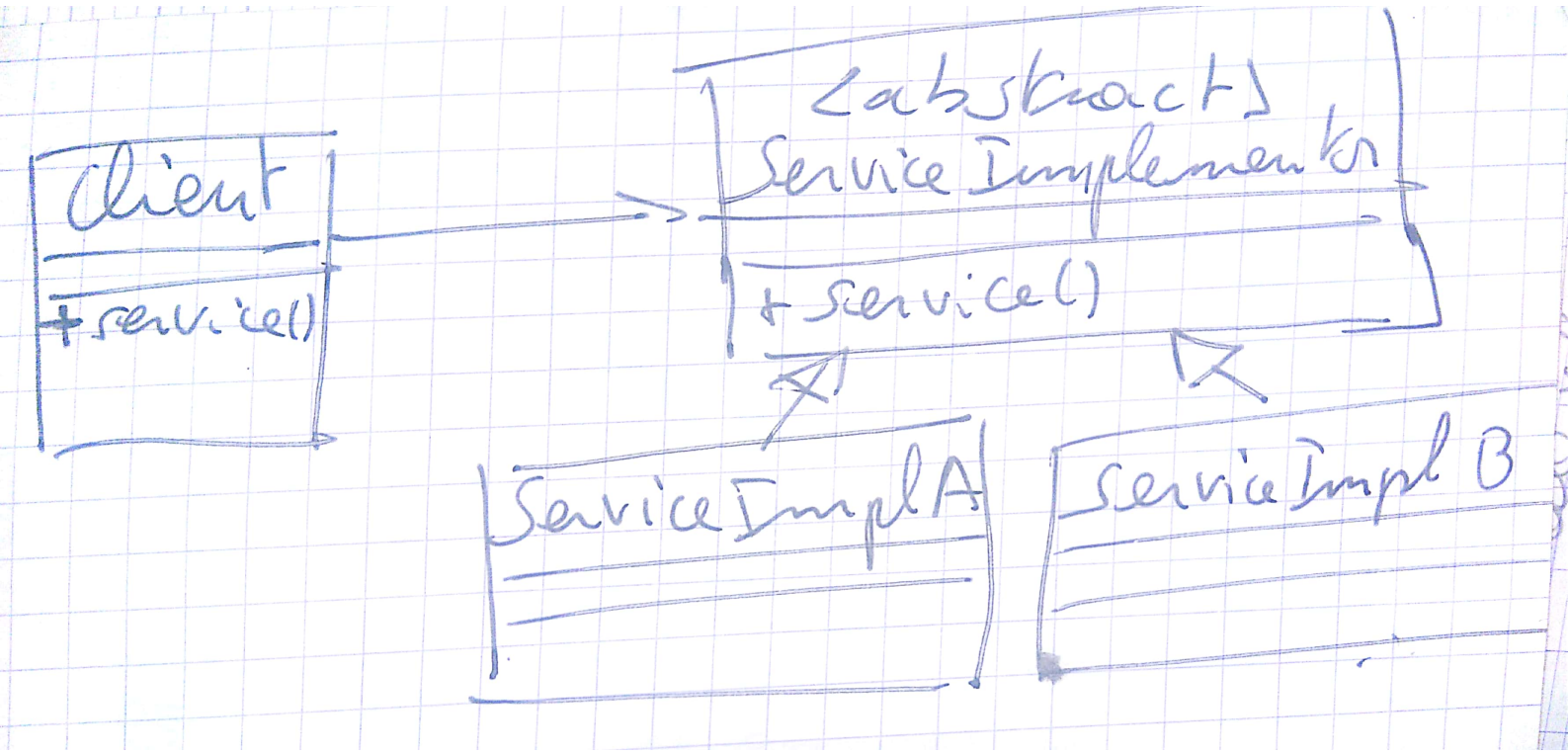
Factory Method

- Instanciation par une méthode polymorphe
- La méthode est implémentée dans les classes d'amortissement



Bridge

- Déléguer un service à une abstraction
→ Découpler l'objet et l'implémentation.



Décorateur

- Enrichir une classe sans la modifier.
-> Emballage de l'interface de base
-> Délégation de s comportements existants non modifiés