R3.04 - Pattern Décorateur

Maïa LORIDAN, Dylan RICHARD, Samuel RIGAUD, Maxime PERRIER

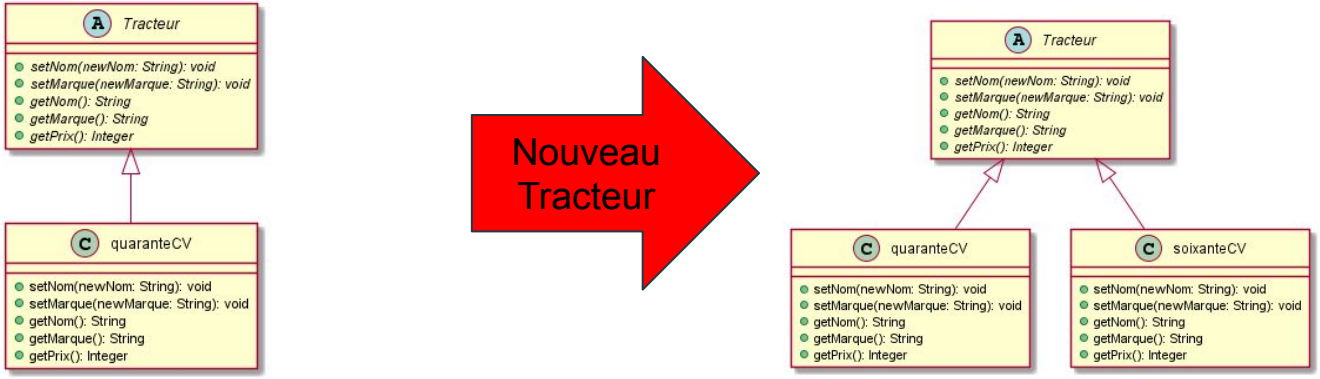
C’est quoi un Design Pattern ?

- Un **Patron de conception** 
- Gain de temps 
- Simplifier nos conceptions 

Énoncé du besoin 1 - TRACTEUR

- Vendre des tracteur
- Tracteur avec des option
- Mettre les options que l’on veut dessus

Proposition de modélisation



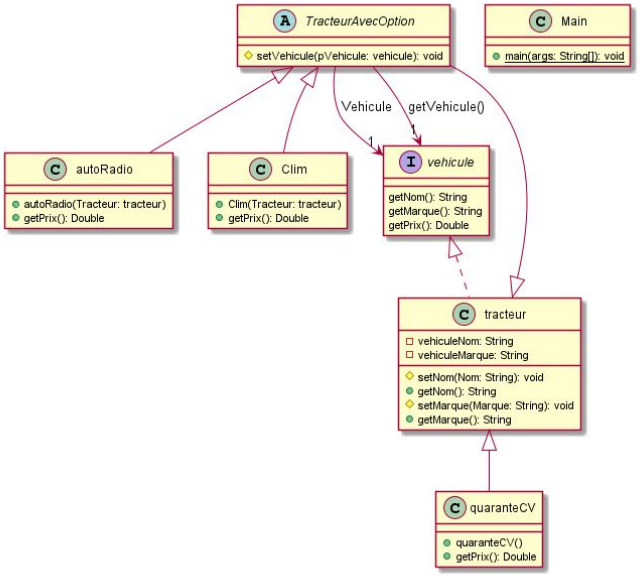
Problèmes de cette modélisation

- Nouveau tracteur = nouvelle classe à ajouter pour le nom et la marque (**explosion combinatoire**)
- Pour les prix :
  - si le prix d'une option change, il faut changer les prix de tous les tracteur qui ont cette option
- Manque de flexibilité

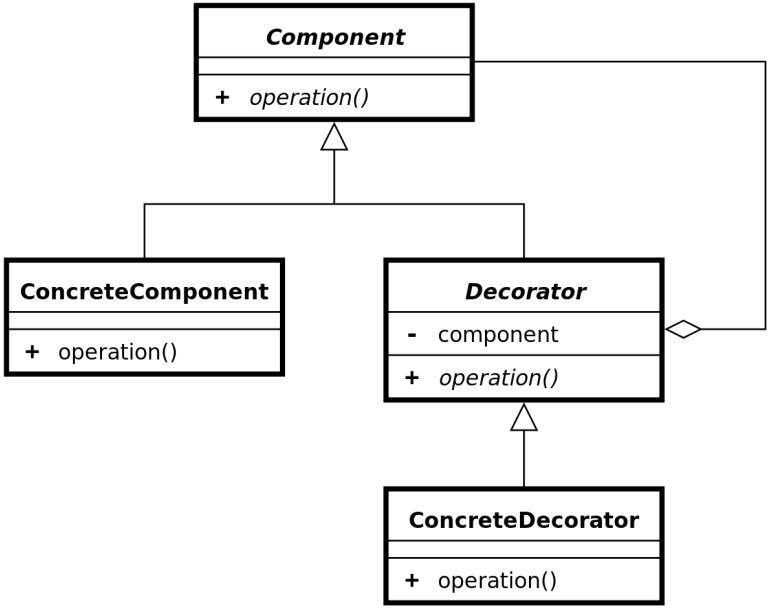
Solution pour cette modélisation

Nous allons rendre notre conception statique en conception dynamique

Nouvelle modélisation



Généralisation du diagramme de classes

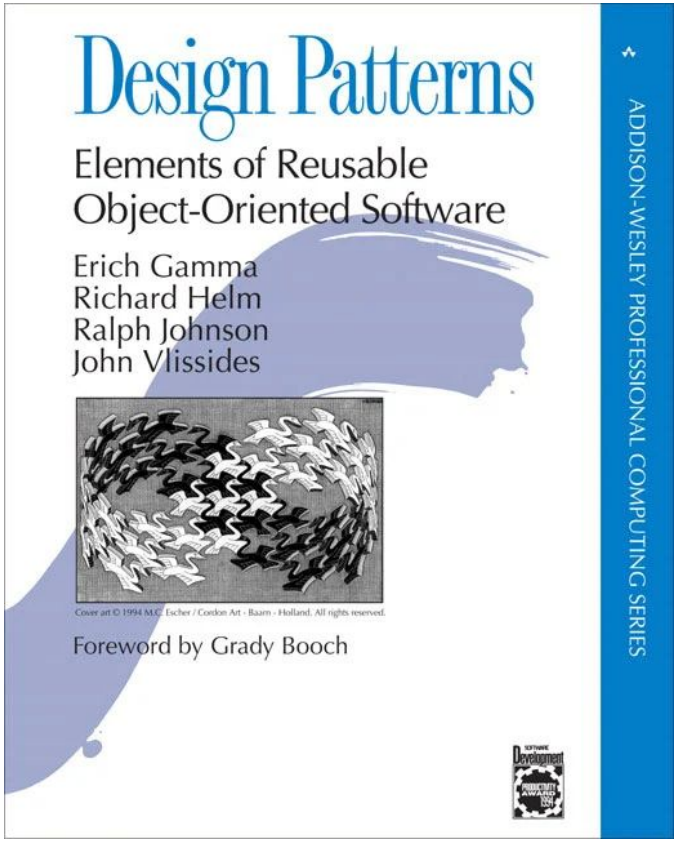


Le Pattern Décorateur

- GoF (Gang of Four) : 23 design patterns

- 1994

- 3 catégories : *Création*, *Structure*, *Comportement*



Utilité du pattern Décorateur

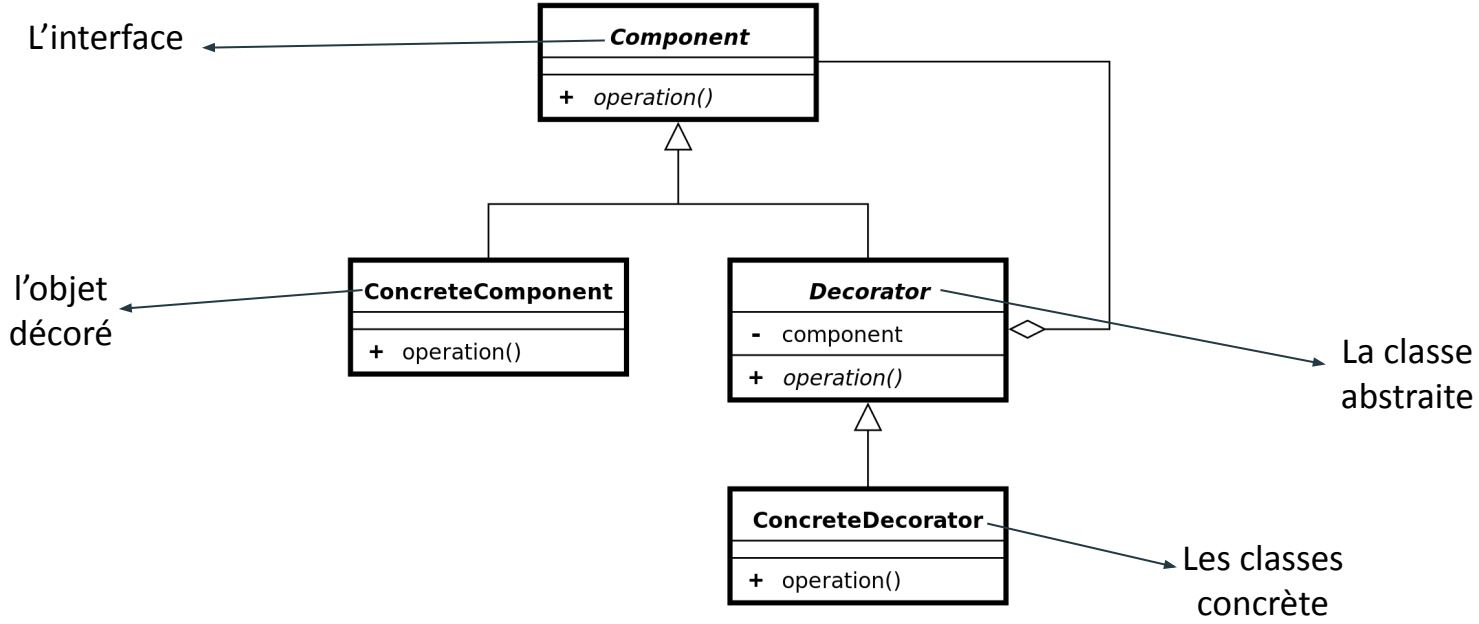
- Permet de modifier un objet dynamiquement
- Permet d'ajouter des fonctionnalités à l’exécutions du programme
- Plus flexible que l’héritage
- Simplifie le code car on peut ajouter des fonctionnalités en utilisant des classes simples
- On peut écrire du nouveau code au lieu de réécrire l’ancien code

Solution du pattern Décorateur

donné par la littérature:

“Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality”.

Détail du rôle des classes



Les principes SOLID

SRP Créer des sous-classes plutôt que de tout mettre dans 1 seule classe

OCP On a pas besoin de réécrire le vieux code juste d'y ajouter des extensions

LSP Notre héritage ne nous donne pas des méthodes inutiles dans notre classe

ISP Créer une autre interface plutôt que tout mettre sur une seule

DIP Création d'une interface et d'une classe abstraite pour découpler nos dépendances (cf interface kebab et abstract class décoration)

Les limites du pattern Décorateur

- Complexification de la construction (instanciation) de l'objet
- Pattern Builder
- Plus complexe à construire
- Code plus complexe dans sa globalité

Rapprochement avec d’autres patterns

Notre Pattern est proche des pattern Visiteur, Composite et Adapter

Rapprochement avec d’autres patterns

Notre Pattern est proche des pattern Visiteur, Composite et Adapter

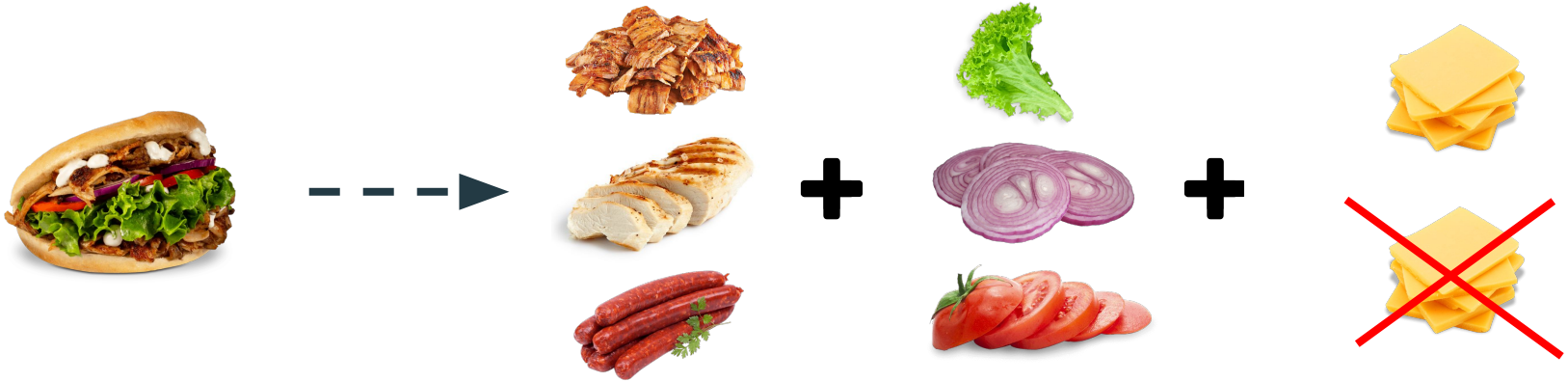
Classes Javadoc qui mettent en oeuvre le pattern Décorateur

Java.io.InputStream , OutputStream, Reader, Writer

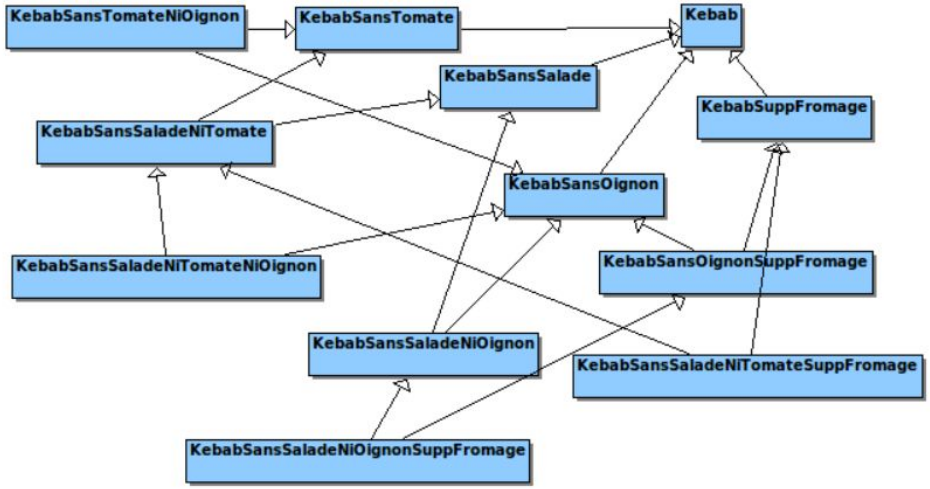
Code qui utilise les flux

Énoncé du besoin 2 - KEBAB

- Préparer des kebabs
- À la carte : kebab personnalisable
- Ajouter des ingrédients et des fonctionnalités à l'avenir



Proposition de modélisation



Problèmes de cette modélisation

- Nouveau type de kebab= nouvelle classe à ajouter (**explosion combinatoire**)
- Pour les prix :
  - si le prix d'un ingrédient change, il faut changer les prix de tous les kebabs contenant cet ingrédient
- Manque de flexibilité

Solution pour cette modélisation

Nous allons rendre notre conception statique en conception dynamique

Nouvelle modélisation