

R3.04

Le pattern Décorateur



**Université
de Limoges**

Maïa LORIDAN, Dylan RICHARD, Samuel RIGAUD, Maxime PERRIER

C'est quoi un Design Pattern ?

- Un **Patron** de conception



- Gain de temps



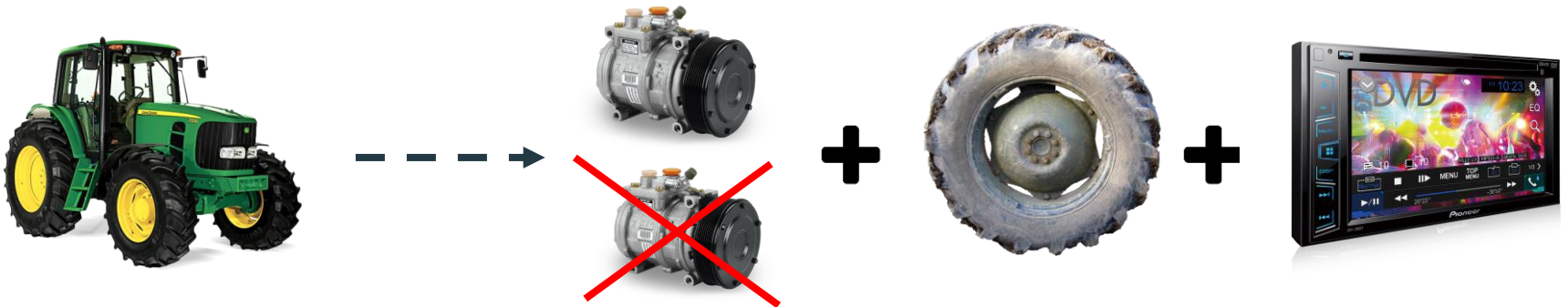
- Simplifier nos conceptions



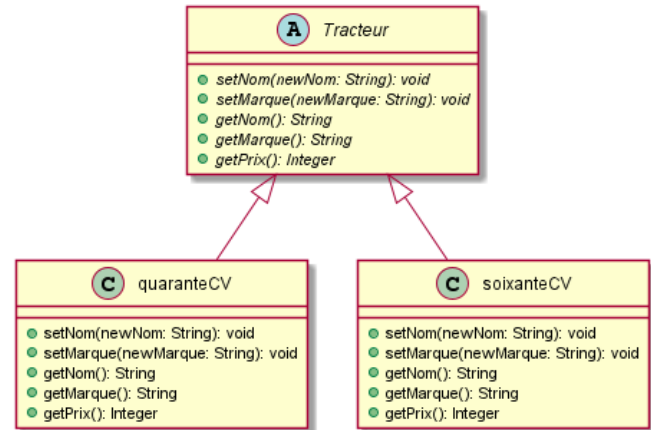
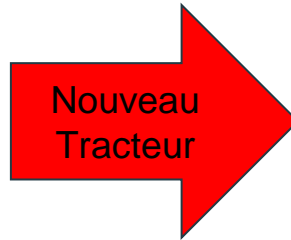
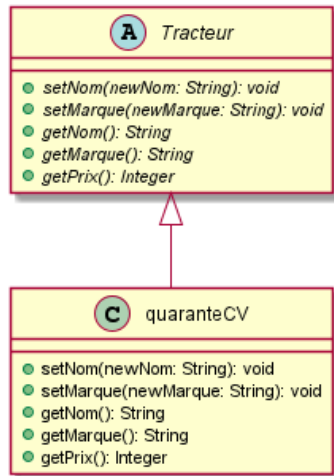
Énoncé du besoin 1 - TRACTEUR



- Vendre des tracteurs
- Tracteur avec des options
- Mettre les options que l'on veut dessus



Proposition de modélisation



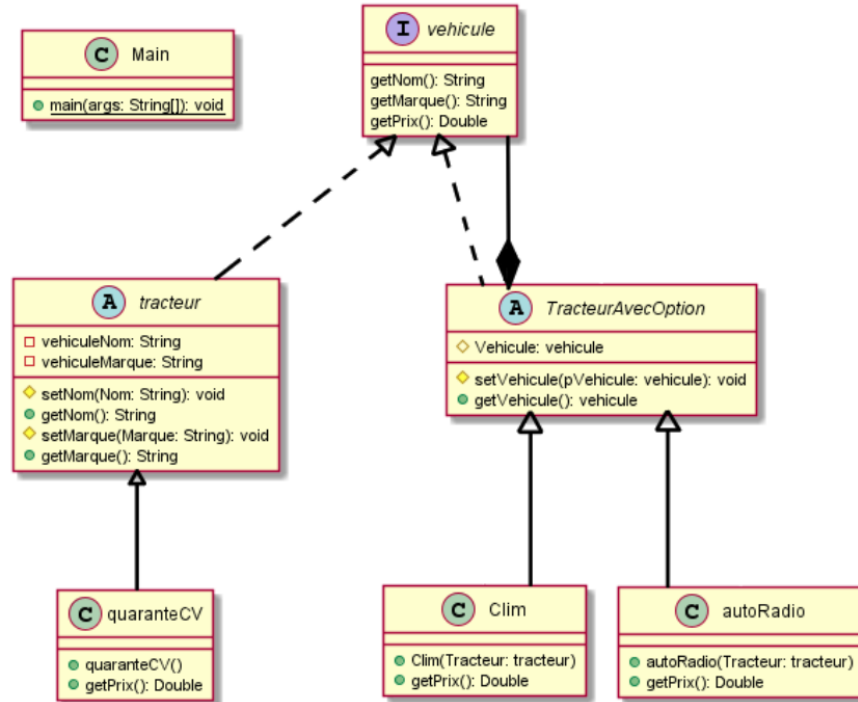
Problèmes de cette modélisation

- 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 tracteurs qui ont cette option
- Manque de flexibilité

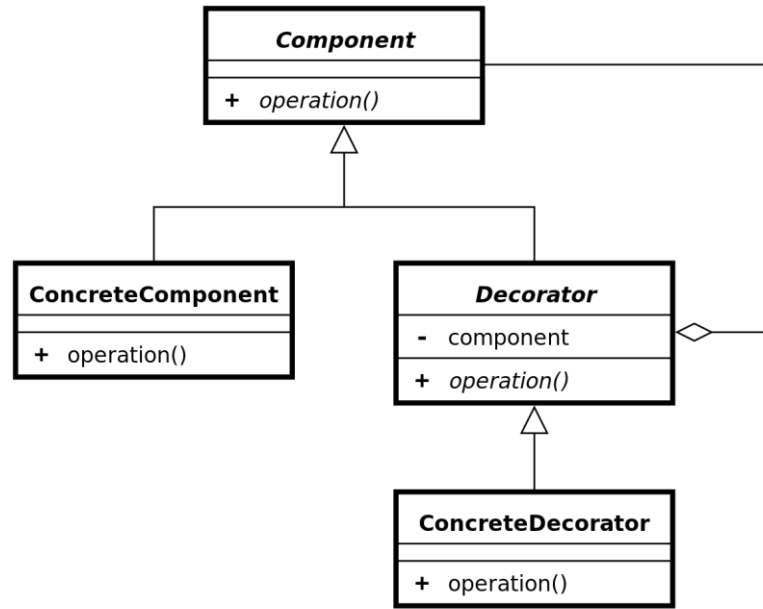
Solution pour cette modélisation

Nous allons rendre notre conception statique en conception dynamique

Nouvelle modélisation

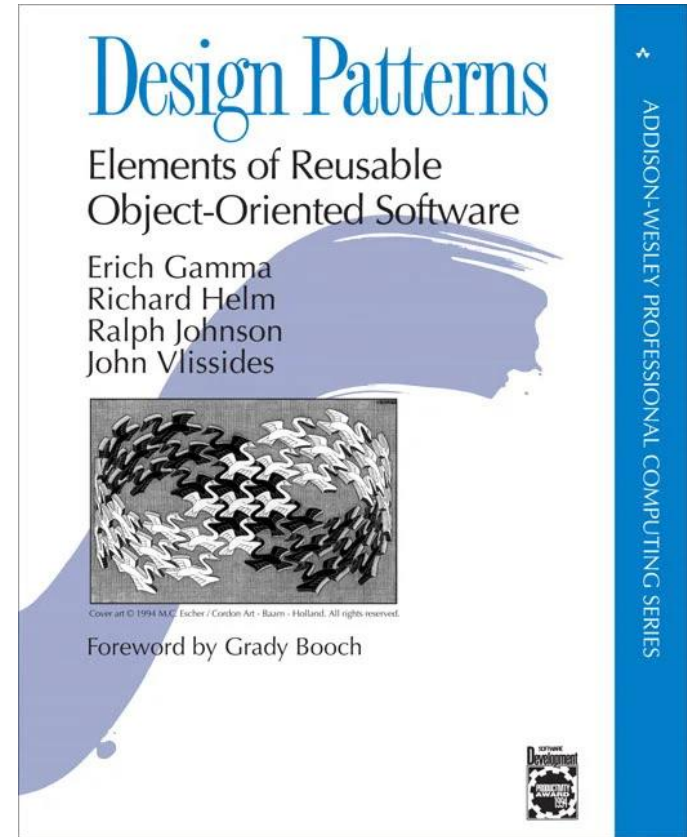


Généralisation du diagramme de classes



Le Pattern Décorateur

- GoF (Gang of Four) : 23 design patterns
- 1994
- 3 catégories : *Création*, Structure, *Comportemental*



Utilité du pattern Décorateur

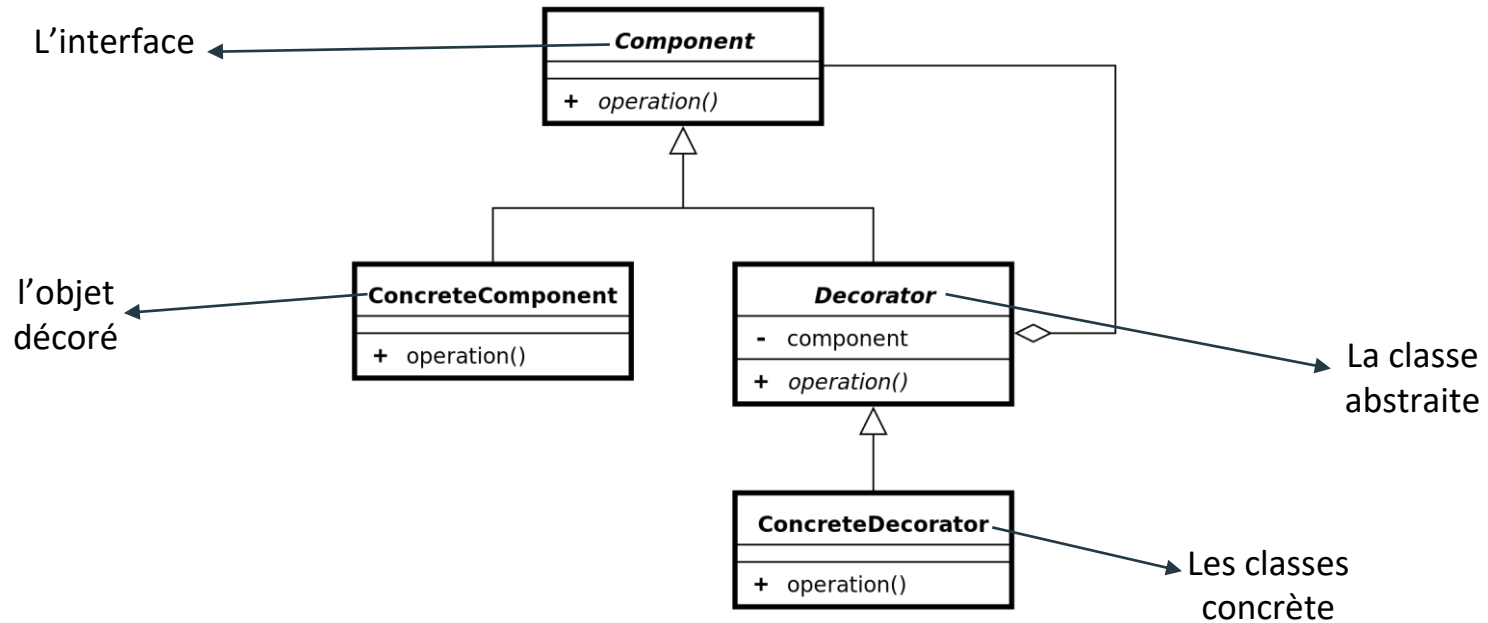
- Permet de modifier un objet dynamiquement
- Permet d'ajouter des fonctionnalités à l'exécution du programme
- Plus flexible que l'héritage
- Simplifie le code car on peut ajouter des fonctionnalités en utilisant des classes simples
- On peut écrire du nouveau code au lieu de réécrire l'ancien code

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

```
public interface vehicule {  
    public String getNom();  
    public String getMarque();  
    public Double getPrix();  
}
```

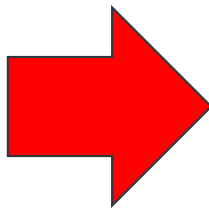
Le véhicule donne lui-même son nom.

Le véhicule donne lui-même sa marque.

Le véhicule donne lui-même son prix.

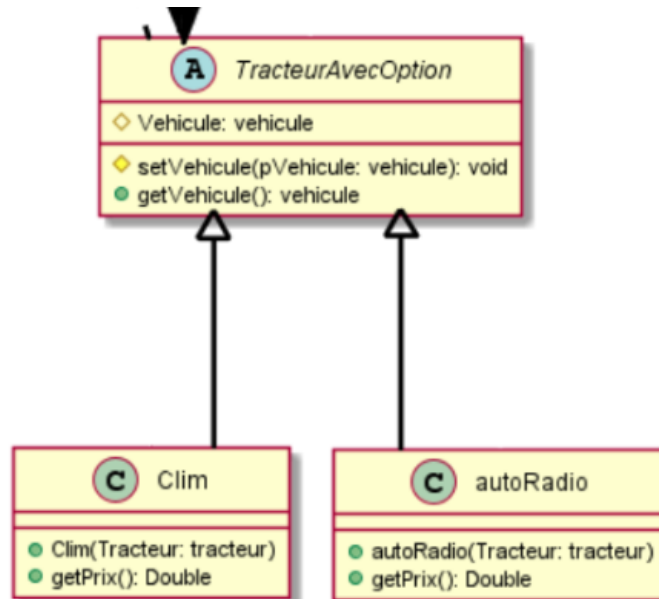
OCP

```
public abstract class Tracteur {  
    public abstract void setNom(String newNom);  
    public abstract void setMarque(String newMarque);  
    public abstract String getNom();  
    public abstract String getMarque();  
    public abstract Integer getPrix();  
}
```

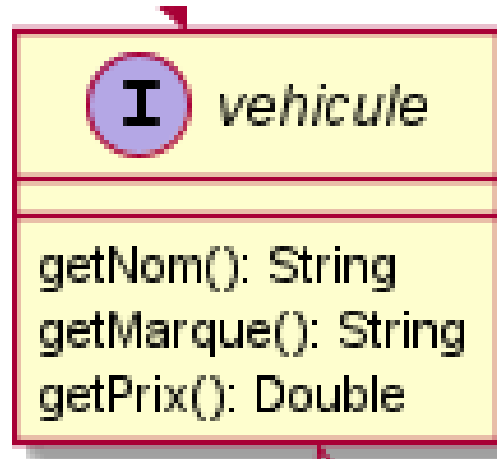


```
public interface vehicule {  
    public String getNom();  
    public String getMarque();  
    public Double getPrix();  
}
```

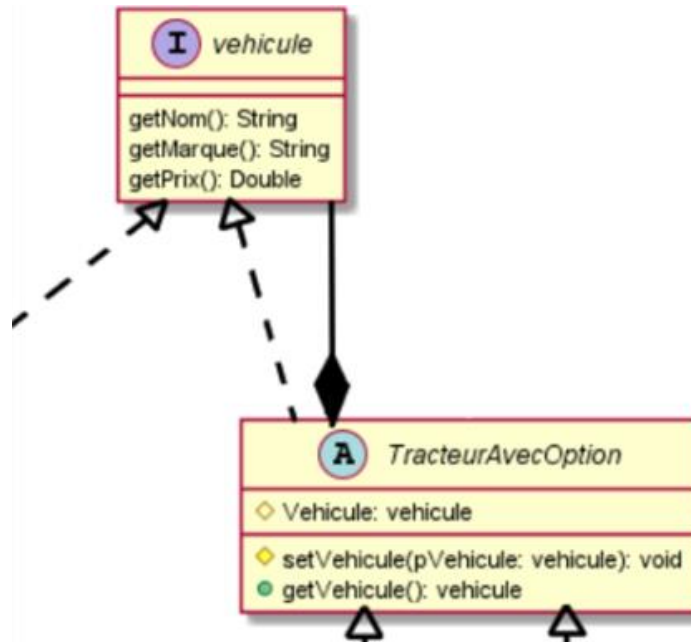
LSP



ISP



DIP



Les limites du pattern Décorateur

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

Rapprochement avec d'autres patterns

Adapter

Composite

Visiteur

Classes **Javadoc** qui mettent en oeuvre le
pattern Décorateur

Java.io.InputStream - OutputStream - Reader - Writer

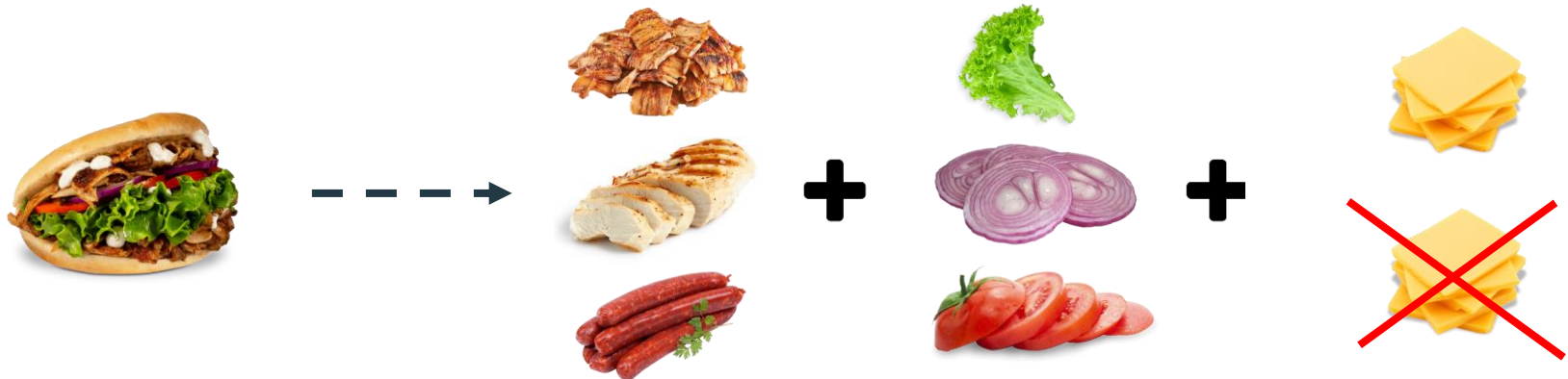
(Gestion des flux)

<https://www.oracle.com/technical-resources/articles/jones-owsm.html>

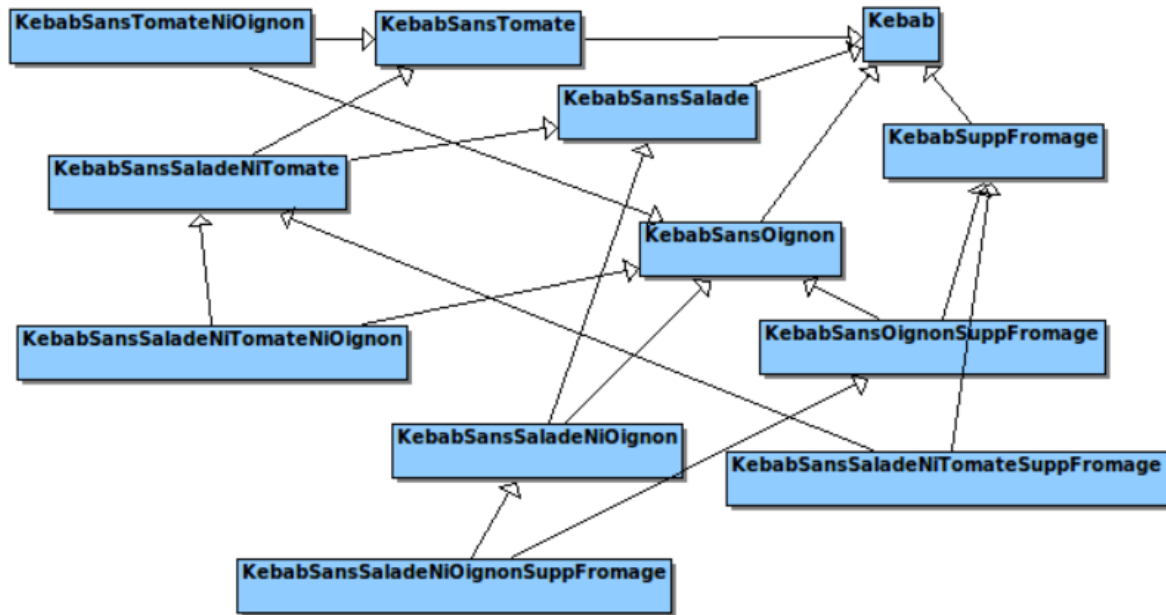
Énoncé du besoin 2 - **KEBAB**



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



Proposition de modélisation



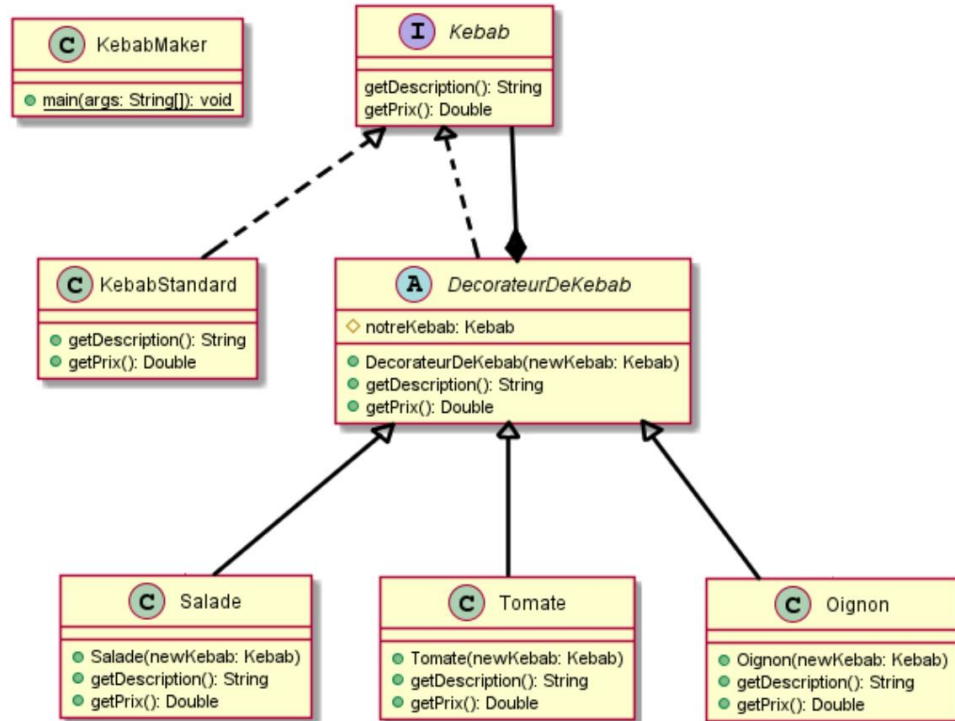
Problèmes de cette modélisation

- 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



IDE window showing Java code for the Decorator pattern. The code defines a `Dignon` class that extends `DecorateurDeKebab`.

```
1 package Creation;
2
3 public class Dignon extends DecorateurDeKebab{
4
5     public Dignon(Kebab newKebab) {
6         super(newKebab);
7     }
8
9     public String getDescription() {
10         return notreKebab.getDescription() + ", oignon";
11     }
12
13     public Double getPrix() {
14         System.out.println("Prix de 1 oignon : " + 0.20);
15         return notreKebab.getPrix() + 0.20;
16     }
17
18 }
```

Below the code editor, a class diagram for **Tomate** is shown:

- Tomate(newKebab: Kebab)
- getDescription(): String
- getPrix(): Double

Webographie

<https://refactoring.guru/fr/design-patterns/decorator/java/example>

<https://blog.cellenza.com/developpement-specifique/le-design-pattern-decorator-decorateur/>

<https://www.adimeo.com/blog-technique/design-patterns-a-quoi-ca-sert-et-comment-les-utiliser>

<https://ryax.tech/fr/design-pattern-cest-quoi-et-pourquoi-lutiliser/>

<https://www.youtube.com/watch?v=j40kRwSm4VE> (pizzeria)

<https://www.youtube.com/watch?v=GCraGHx6gso> (autre vidéo)

https://www.youtube.com/watch?v=WghYGmUd9nQ&ab_channel=BracketShow (quebec)

[https://fr.wikipedia.org/wiki/D%C3%A9corateur_\(patron_de_conception\)](https://fr.wikipedia.org/wiki/D%C3%A9corateur_(patron_de_conception)) wikipedia

<https://www.elao.com/blog/dev/design-pattern-decorator>

The image features three concentric black circles centered on a black background. The circles have a subtle radial gradient, appearing slightly lighter in the center. Overlaid on these circles is the text "The End" in a white, elegant script font. The text is positioned horizontally across the middle of the circles. A soft, dark shadow of the text is visible beneath it, creating a sense of depth.

The End

QCM

https://docs.google.com/forms/d/e/1FAIpQLSdwXvqXDT7sC6px1cee7SXMC1wEq9ovdX7gvnHJnJJhJKRD1A/viewform?usp=sf_link

<https://bit.ly/3C9MW4k>

