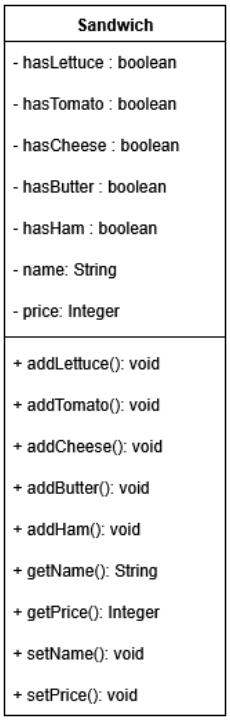
Fiche résumé – Pattern Décorateur

Le pattern **Décorateur** (Decorator) est un modèle structurel qui permet d'ajouter des responsabilités à un objet de manière dynamique, sans modifier sa structure ni multiplier les sous-classes. L'idée est de "décorer" un objet de base avec de nouvelles fonctionnalités, selon les besoins. Ce pattern est particulièrement utile lorsque les variations d'un objet peuvent croître rapidement, comme dans le cas d'un système modulable.

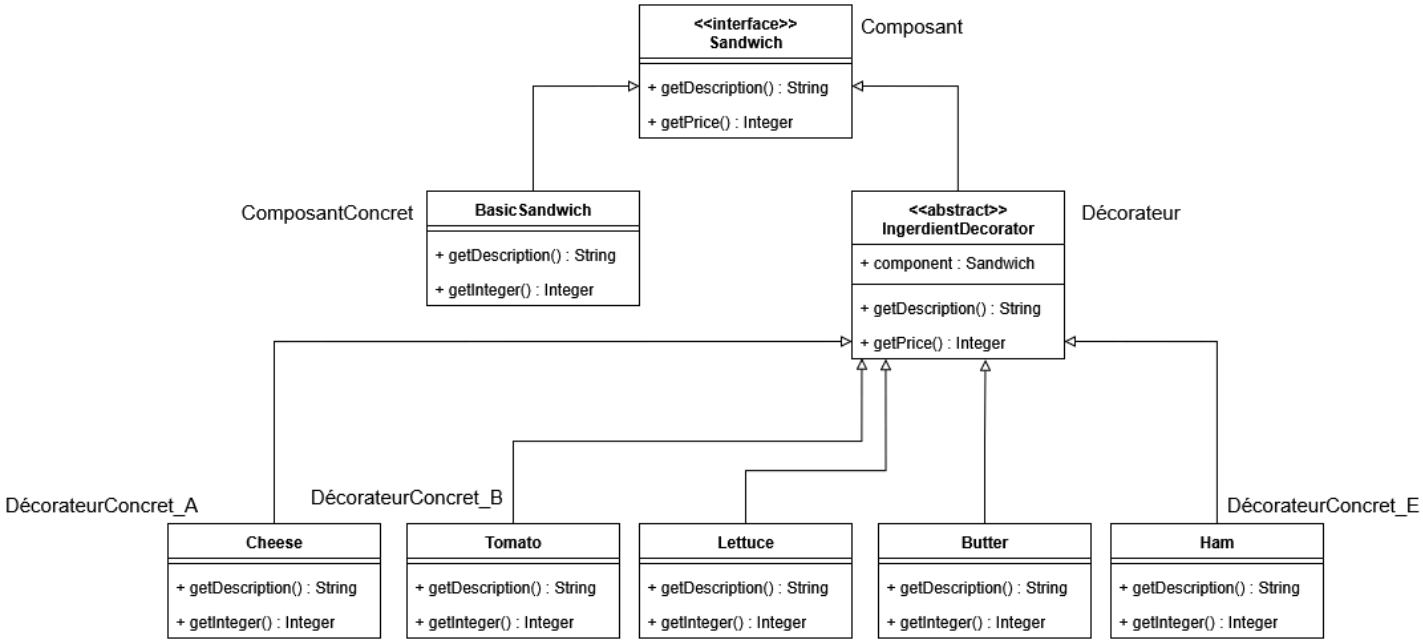
Exemple pour introduire le besoin

Imaginons un service de création de sandwichs personnalisables. Chaque sandwich peut contenir différents ingrédients : salade, tomate, fromage, etc. Le problème avec une approche classique est que chaque combinaison d'ingrédients nécessiterait une sous-classe spécifique, ce qui devient vite ingérable :



Nous pouvons donc recourir au pattern Décorateur, ce qui rendra la structure immédiatement plus lisible et facile à gérer.

Solution avec le pattern Décorateur



Chaque décorateur peut s'empiler dynamiquement, permettant de composer un sandwich avec différents ingrédients sans créer de nouvelles sous-classes pour chaque combinaison.

Rôle des classes participantes

Classe	Rôle
Composant (Component)	Interface ou classe abstraite définissant les opérations communes.
ComposantConcret (ConcreteComponent)	Objet de base à décorer.
Décorateur (Decorator)	Contient une référence au Component et délègue les appels.
DécorateurConcret_A (ConcreteDecorator)	Ajoute des fonctionnalités supplémentaires, combinables avec d'autres décorateurs.

SOLID

SRP : chaque décorateur a une responsabilité unique.

OCP : nouvelles fonctionnalités sans modifier les classes existantes.

DIP : le client dépend de l'interface Component et non des classes concrètes.

Résultat : code modulable, extensible et clair.

Limites

- Trop de décorateurs → code difficile à lire.
- Gestion complexe des objets “entourés”.
- Peu pertinent pour des objets très simples ou peu variés.

Conclusion

Le pattern Décorateur permet d’ajouter des fonctionnalités à un objet de manière flexible et dynamique, tout en préservant la maintenabilité et la clarté du code, et en respectant les principes SOLID.