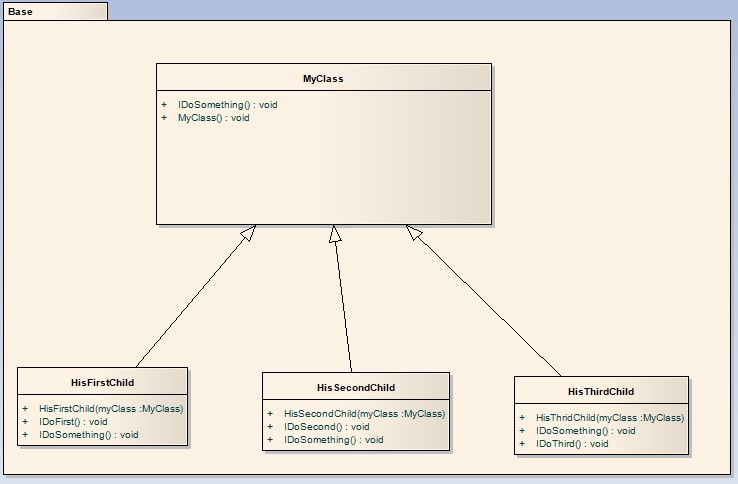
Le design pattern Décorateur

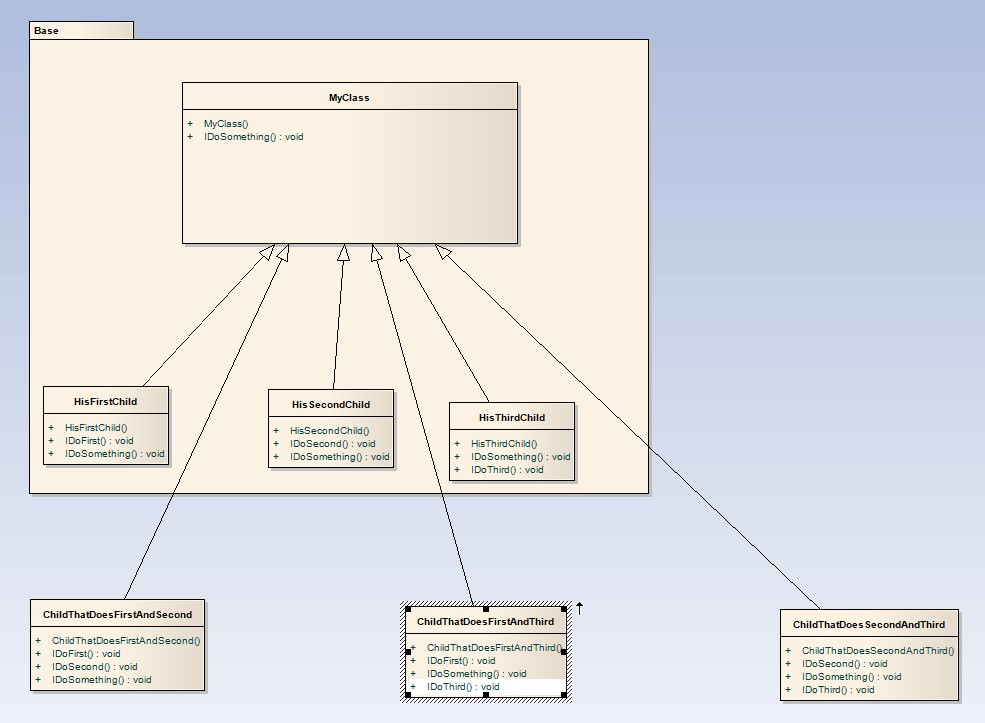
Source:  
<https://refactoring.guru/fr/design-patterns/decorator><https://design-patterns.fr/decorateur-en-java>

1. Définition pattern

Le design pattern décorateur est un design pattern structurel.  
Un design pattern est une solution donnée à un problème récurrent en conception de logiciel.  
  
Le terme structurel ajoute la notion de solution liée à la structure des objets et des classes.  
  
 2. Problème à résoudre

Imaginons dans le cadre d’une conception logiciel, un même objet qui doit pouvoir changer ces comportements au fur et à mesure que le programme évolue.

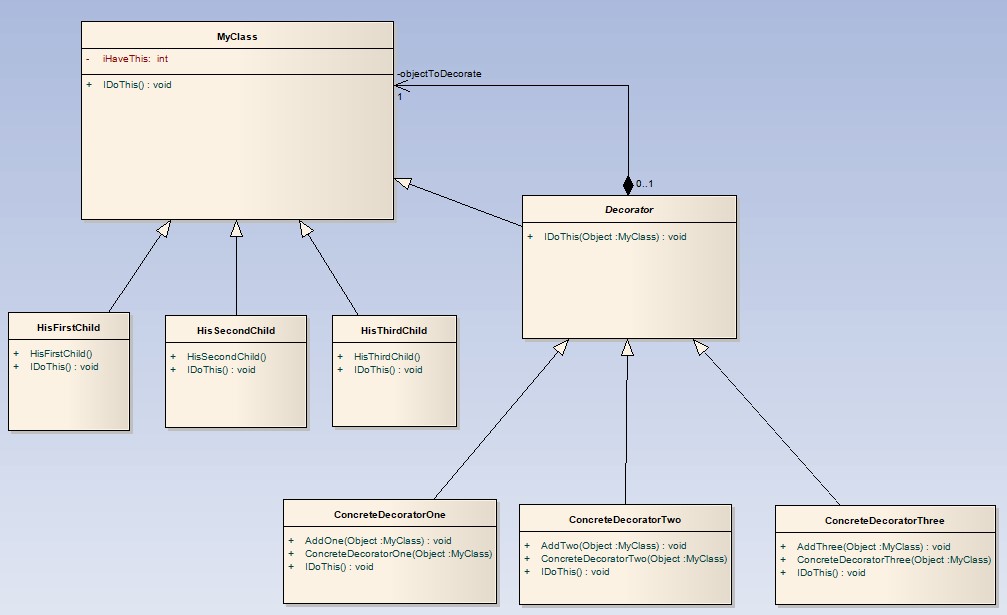
La logique voudrait que l’on utilise un système d’héritage pour faire évoluer la classe qui instancie l’objet pour le faire évoluer.  
  
Seulement nous nous heurtons à un problème, J’ai désormais besoin que mon objet possède les comportements de plusieurs classes en même temps.  
  
Déjà cela casse le principe de responsabilité unique, et aussi cela demande à créer de nouvelle sous-classe comme le montre le schéma suivant.



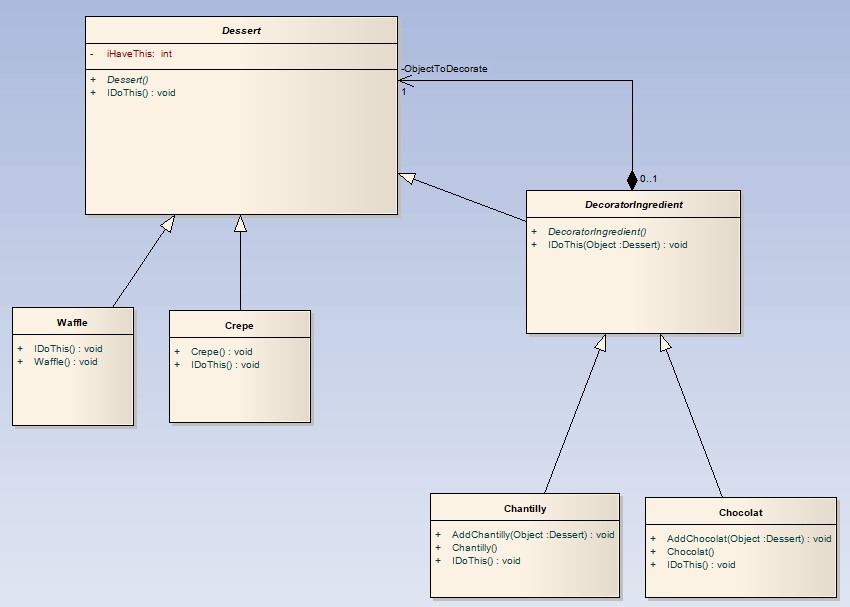
Cela peut sembler être une bonne approche. Mais il suffit d’imaginer 10 classes qui hérite de la première, pour se rendre compte du nombre de sous classe que cela ferait. Le code deviendrait horrible à maintenir et surtout très illisible dû à la répétition du code dans plusieurs classe.  
  
 3. Solution – Pattern Décorateur

Le pattern décorateur propose une solution mettant en scène une classe nommé “décorateur” qui hérite de la classe abstraite d’où provienne les objets qu’il doit “décorer” (nous reviendrons sur cette notion plus tard), et prend également en paramètre un de ces objets.  
  
Il possède ensuite lui-même un tas de sous-classe qui contienne toutes sorte de méthodes, variable ect.. Nous appellerons ces sous-classe décoration pour vulgariser l’explication.

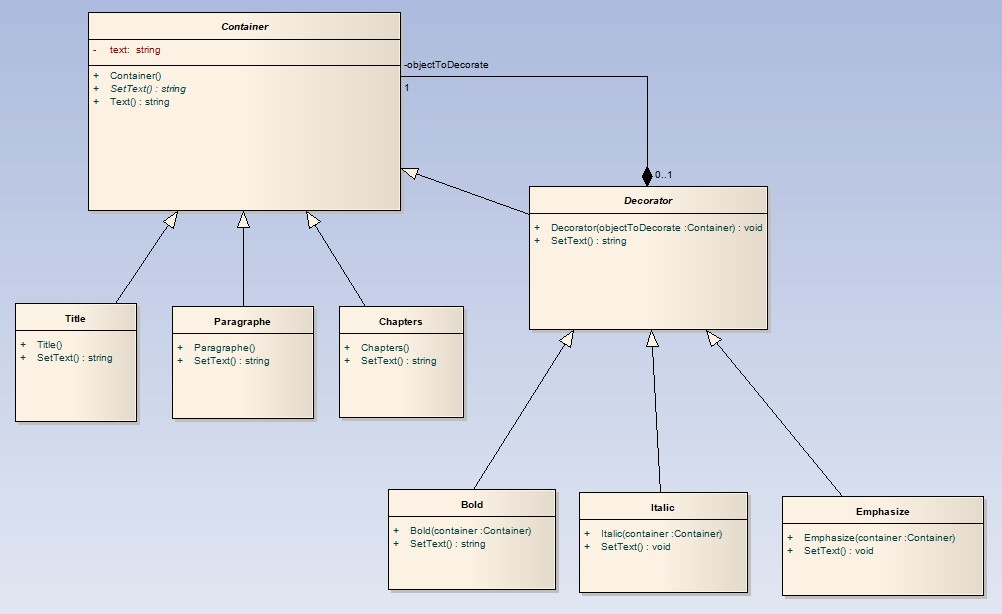
Voici un exemple de diagramme UML générique de ce pattern :

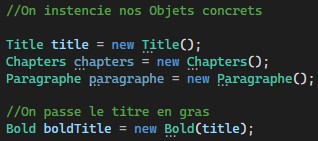


L’objet rentrant dans la classe décorateur se verra ajouter une “couche” de “décoration” tout en gardant son type de base.

Voici un schéma pour vous aider à comprendre l’explication qui suivra la définition ci-dessus :

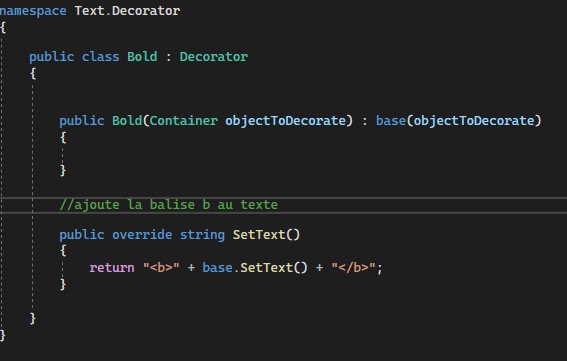
Ma crêpe ici qui est un dessert, est envoyer à la classe “DecorateurIngredient” qui est elle aussi un dessert, pour lui ajouter soit une couche Chantilly, soit une couche nutella, ici choisissons Chocolat.  
Ma crêpe devient donc une “crêpe au Chocolat” mais, reste une crêpe.  
Nous venons d’ajouter le comportement “AddChocolat()” à la crêpe sans avoir eu recourt à une sous-classe “crêpeAuChocolat”.  
  
 4. Exemple concret

Voici un exemple concret d’utilisation du pattern décorateur dans le cadre du développement d’un programme.  
  
Ici nous avons un ”Conteneur” qui contient du Texte et une méthode pour ”Modifier” ce text.  
Cette méthode se propage à toutes les classes qui hérite du Conteneur.  
  
Ici, **3 Objets concret** : ”Titre, Chapitre, Paragraphe”. (Html is that you?)  
Pour l’instant ces 3 Objets n’ont rien de diffèrent et peuvent simplement posséder du texte.  
  
Maintenant, je souhaiterais que mon Titre possède la caractéristique et le comportement **”Bold”** qui signifie **”Gras”** .  
  
Grâce au pattern décorateur, plutôt que de créer une classe ”TitreEnGras”. Je vais simplement décorer mon objet Titre avec la décoration ”Bold”.

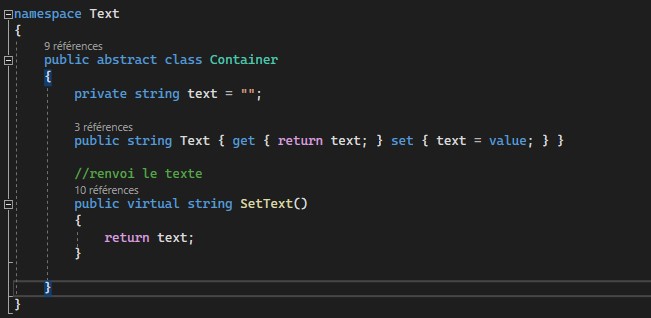
Ici, j’instancie un Objet de type Title qui hérite de Text.  
Je l’envoie à ma classe **”Bold”** qui hérite de ”Décorator” et mon Objet possède maintenant les propriétés d’un Objet de type Bold tout en restant un Objet ”Text” (voir diagramme UML au-dessus si besoin).

Résultat :



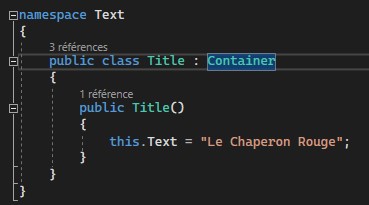
Code de la classe **Bold:**

La méthode SetText hériter de Decorator, elle-même hérité de Text.  
Est OVERRIDE (réécrite) pour changer son comportement afin de renvoyer la STRING suivante:  
<b> le texte du parent le plus vieux <b>

Afin de mieux comprendre voici le contenue de la classe parente Container:  
  


Le code contient une variable string vide. (Qui est remplit lors de la création de l’objet Title), Un getter et un setter.  
Et une fonction qui renvoi le contenue de la variable string “Texte” sous forme de string.

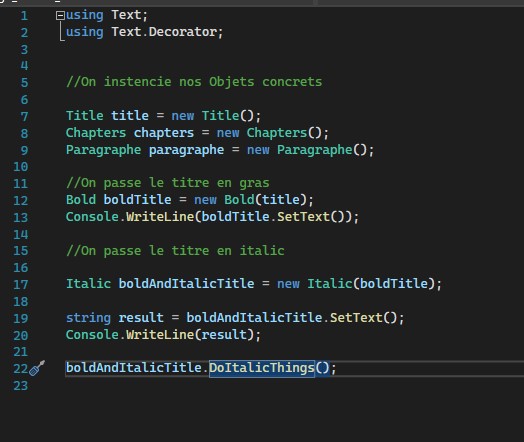
Contenue de la classe Title:



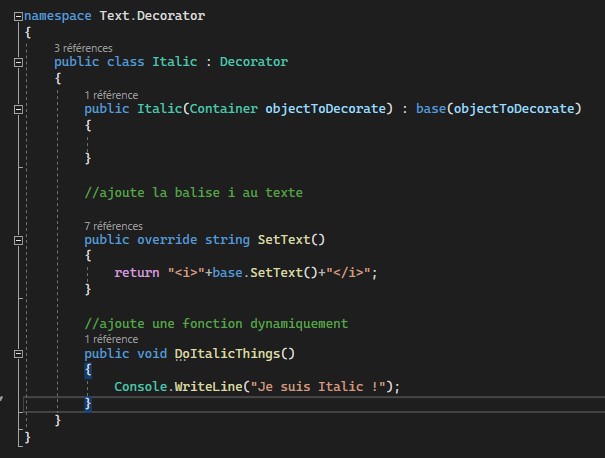
EXPLICATION COMPLIQUEE :

J’instancie un objet “Titre” qui hérite de “Container”.  
Je créer un Objet “boldTitle” qui est un Objet de type Bold, qui hérite de “Décorator” qui hérite lui-même de Container et qui en plus, contient un Objet de type Container.

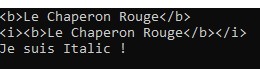
Pour instancier mon Objet Bold, j’ai donc besoin d’un Objet de type Container. Qui sera mon titre.  
Mon titre est donc utilisé comme base, et se voie ajouter une couche Bold pour devenir un objet de type (vulgairement parlant) “TitreBold”.

Pour aller plus loin, je vais maintenant transformer mon **boldTitle** en boldAnditalicTitle. Et en profiter pour lui ajouter un comportement que seul Italic connait qui est “DoItalicThings()”.

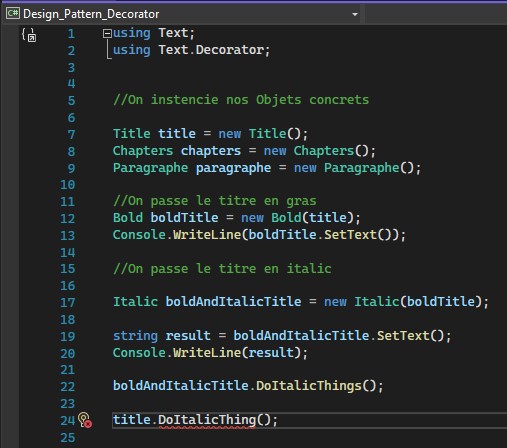
Ici je prends donc mon Objet boldTitle, qui est un Objet Bold, hérité de Container avec le comportement d’un “titre” (c’est donc en terme technique, un Objet titre hérité de Container avec une couche Bold). Je l’envoi à mon décorateur pour lui ajouter une couche Italic.



Résultat:



Maintenant mon Objet title a reçu des couches de savoir qui lui permette de faire des choses qu’il ne savait pas avant.  
Illustration:



Voilà une utilisation concrète du pattern décorateur o/ . 😀 . 😊 .

