# INF 4067 - UML ET DESIGN PATTERNS

## Devoir 2 - Implémentation des patterns de Structuration

**Author**
CHEDJOUN KENGUEP Dave 20U2757
Département Informatique - Master 1 - Genie Logiciel
Octobre - Novembre 2023

# Diagrammes UML et résultats

Dans cette 2ème partie de notre devoir nous implémenterons les designs patterns de structuration. Pour se faire, nous avons pour chaque implémentation, des versions de codes ainsi que des versions de diagrammes correspondants à chaque code. Nous vous les présenterons ci dessous en vous donnant, une brève description, des screenshots des résultats et les images avec les modèles UML. Toutes les implémentations se trouvent sur le dépôt gitHub `https://github.com/DavePhil/INF4067_Implementation_Patterns_Structuration.git`.

## 1 Patron Adapter

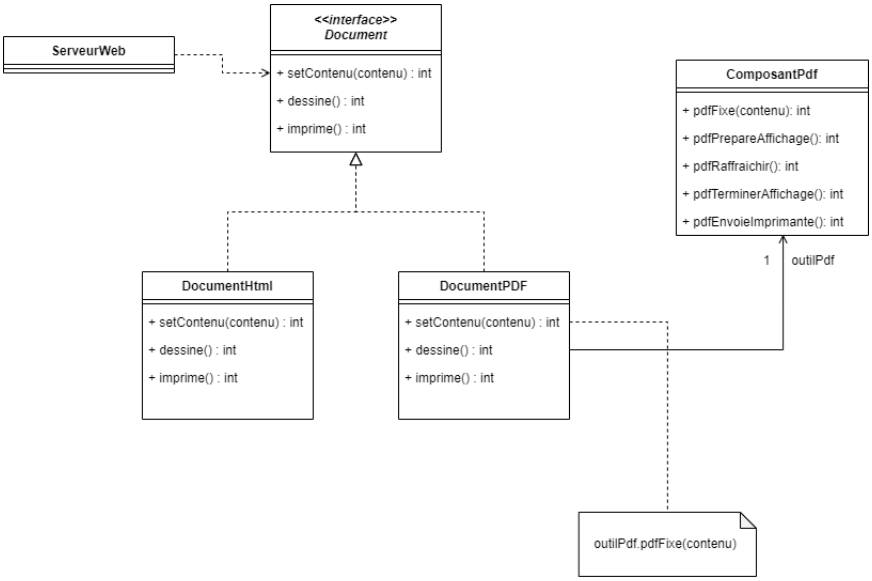
Le patron adapter est un patron, qui permet la collaboration d'instance des classes qui ont des interfaces non compatibles. Son but est de convertir l'interface d'une classe existante en une autre attendue par le client également existants afin qu'il puisse travailler ensemble. Trois exemples du cours ont été fait pour implémenter ce patron. Nous aurons une version du premier exemple (Adaptation Document), le second exemple(Adaptation de classe Gestion qui utilise les piles, pour utiliser les listes chainées) avec deux versions et enfin un troisième exemple (l'adaptation d'une interface pour les carré qu'elle gère les rectangles). Les participants à ce patron sont:

- l'interface: Qui introduit la signature des méthodes de l'objet
- le client : Programme ou classe qui interagit avec les objets qui répondent à l'interface
- l'adapteur : Implémente les méthodes de l'adapter en invoquant les méthodes de l'adaptée
- l'adaptée: Objet dont l'interface doit être adaptée pour correspondre à l'interface

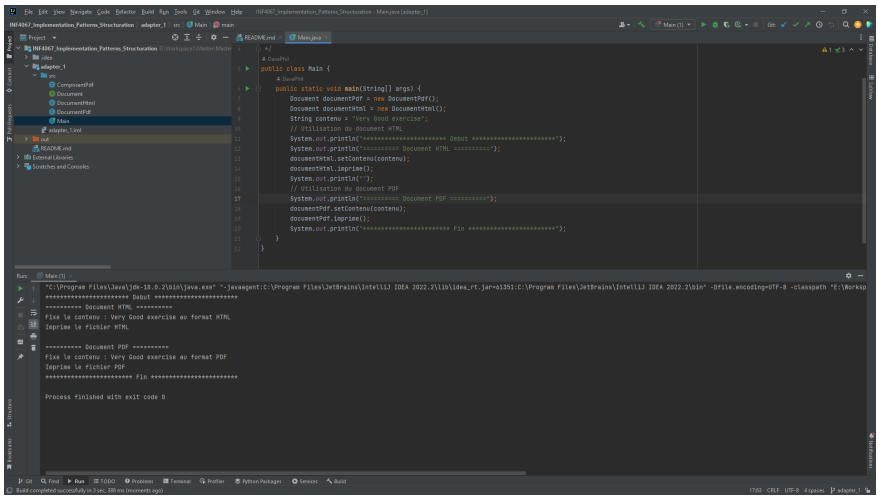
### 1.1 adapter_1

Il s'agit ici de l'implémentation de l'adaptation d'une interface qui traite les documents HTML, mais ayant désormais des composants PDF, on aimerait que par cette interface l'on traite les Documents PDF. Son implémentation se trouve sur le dépôt git donné plus haut et sur la brance **adapter_1**

### 1.1.1 Modèle



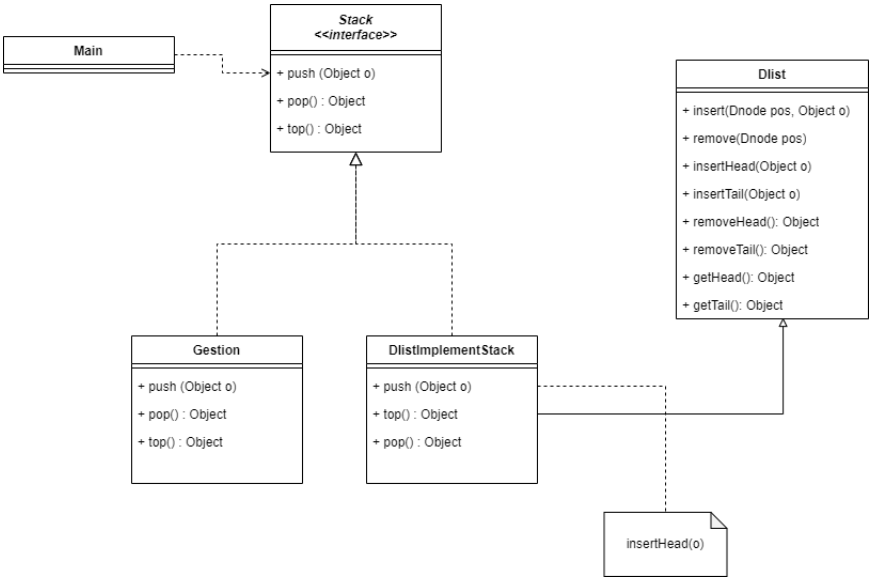

### 1.1.2 Résultats



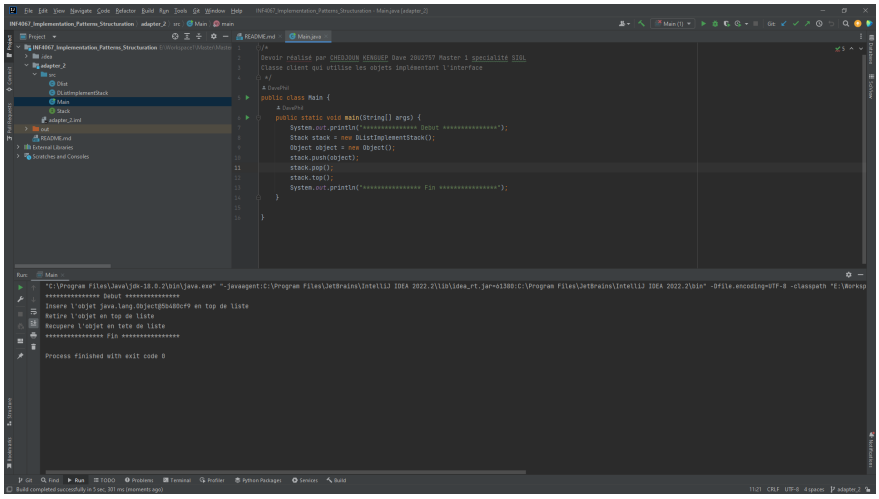
## 1.2 adapter_2

Il s'agit ici ayant une classe DList qui définit une liste doublement chainée et d'une autre classe (Gestion) qui sait manipuler les piles, d'utiliser la classe Dlist dans Gestion sans modifier Dlist. Ici, nous choisissons comme façon de faire, utiliser l'héritage et l'utilisation d'une interface sur la classe Adapteur. L'implémentation de problème se trouve sur la branche **adapter_2**

### 1.2.1 Modèle



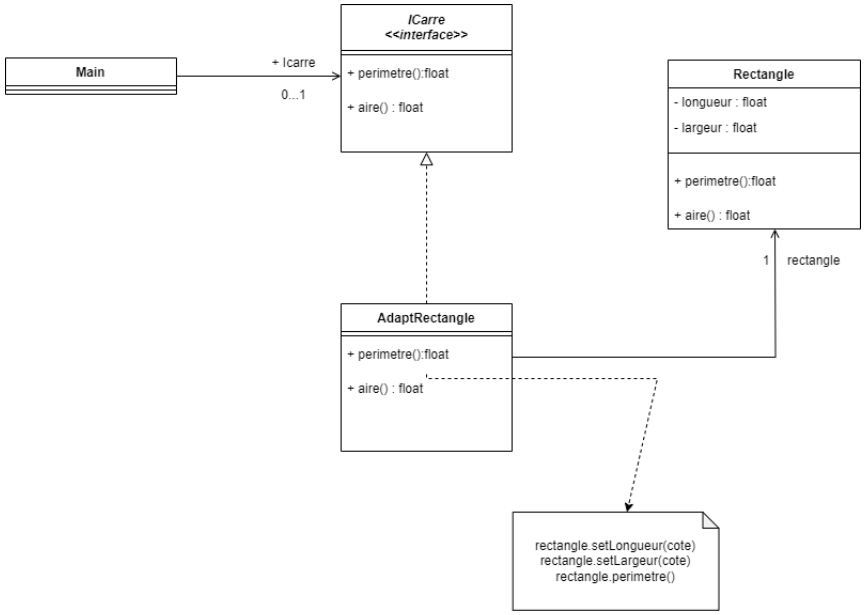

### 1.2.2 Résultats



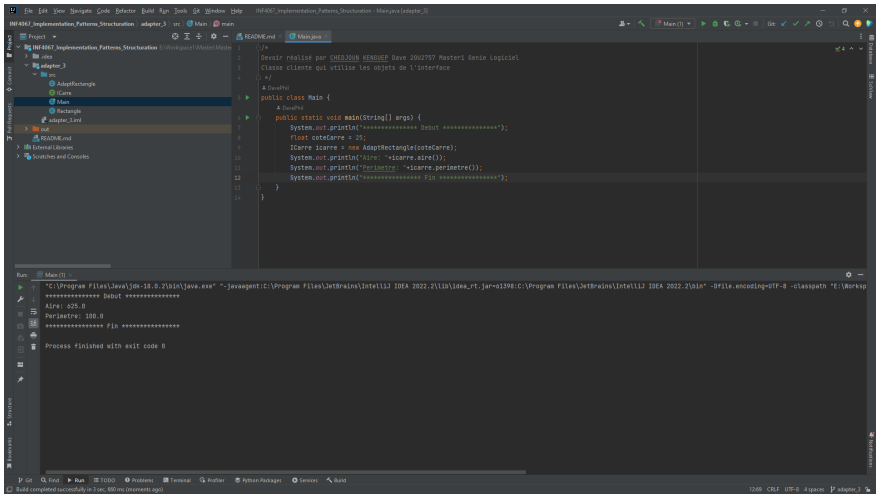
## 1.3 adapter_3

Il s'agit ici d'adapter un programme utilisant une classe qui sait gérer les carrés, de tel sorte qu'elle puisse gérer les rectangles. Son implémentation se trouve sur la branche **adpter_3** du dépot git.

### 1.3.1 Modèle



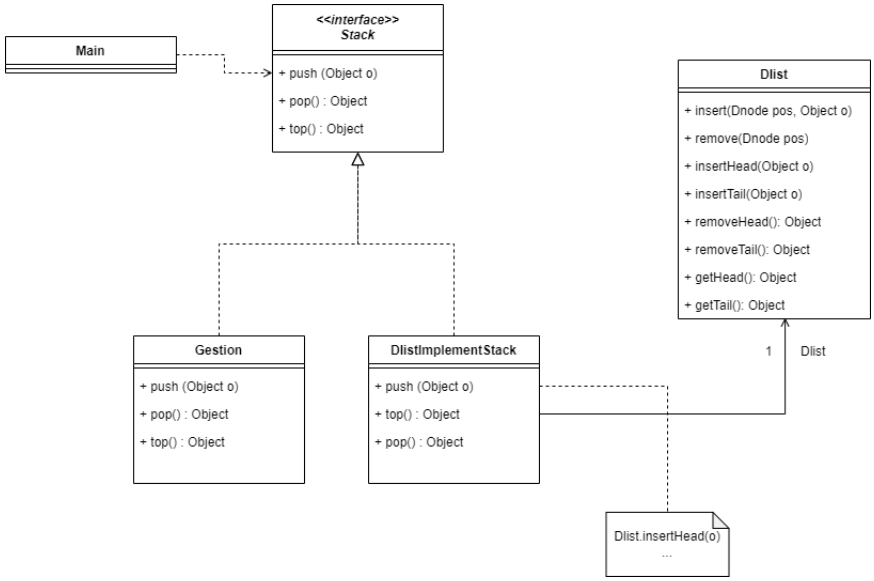

### 1.3.2 Résultats



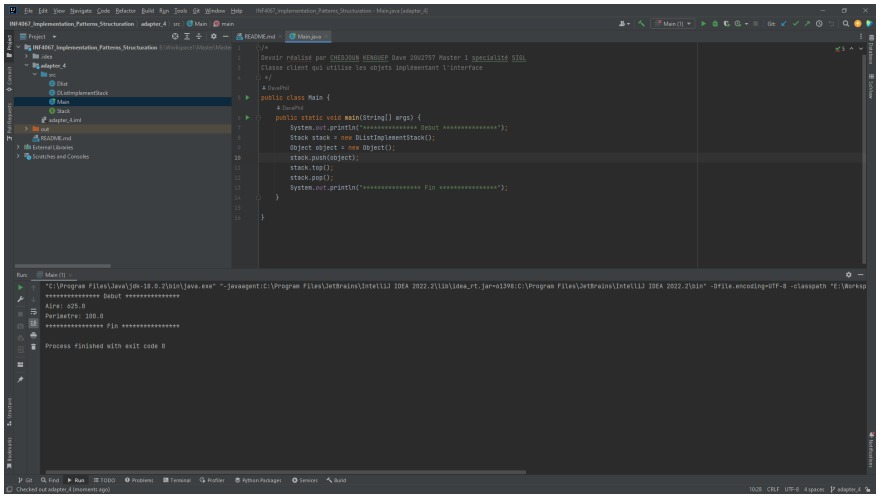
## 1.4 adapter_4

Ici, il s'agissait d'implémenter une deuxième version pour l'adapter_2. Nous utilisons ici plutôt qu'un heritage, une relation entre la classe adapter et la classe adaptée. L'implémentation se trouve sur la branche **adapter_4**.

### 1.4.1 Modèle




### 1.4.2 Résultats



# 2 Patron Composite

Le patron composite, permet de combiner des objets en structure plus grande. il permet de représenter une hiérarchie de composants. Par exemple, nous pouvons utiliser le patron composite pour représenter une hiérarchie d'organisations, une hiérarchie de menus ou une hiérarchie de fichiers. Permettre aux clients de traiter des objets individuels et des compositions d'objets de la même manière. Cela peut simplifier le code client et le rendre plus générique. Ces participants sont:

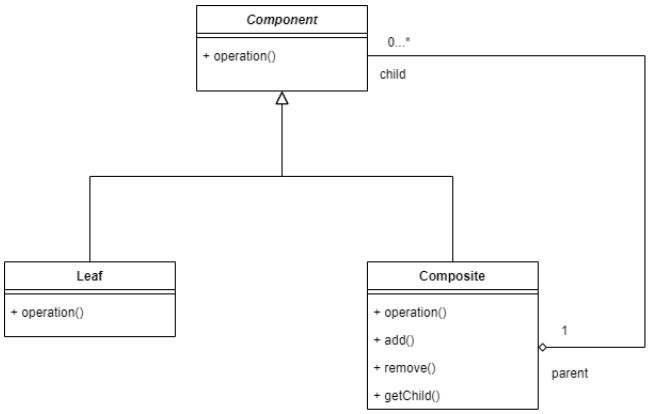
- Le component : déclare une interface commune à tous les objets. Déclare le comportement par défaut pour toutes les classes.
- Leaf: C'est la feuille. Elle implémenente le comportement élementaire

- Composite : définit le comportement des composants ayant des fils, stocke les fils et implémente les opérations necessaires à leur gestion.
- Le client: si le receveur est une feuille la requête est directement traité. Sinon le composite retransmet à tous ses fils.

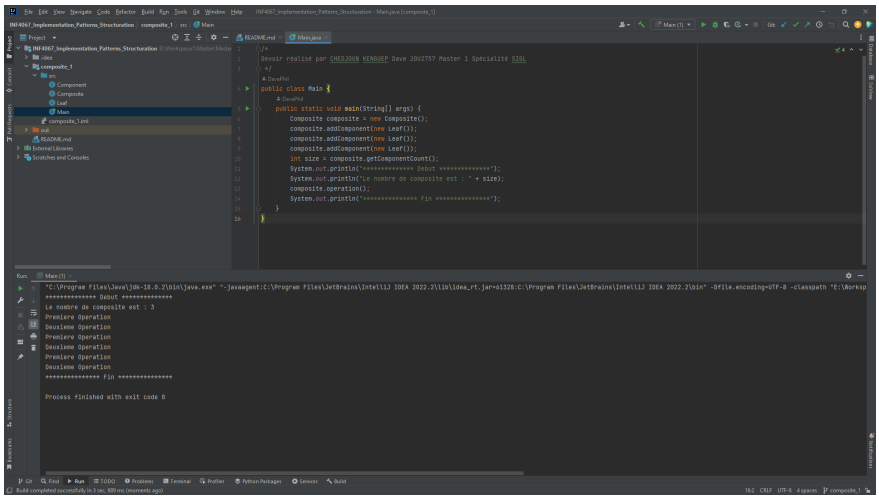
## 2.1 composite_1

Cette première implémentation représente, limplémentation du modèle générique du composite. Elle se trouvera sur la branche **composite_1** du dépôt git du devoir.

### 2.1.1 Modèle



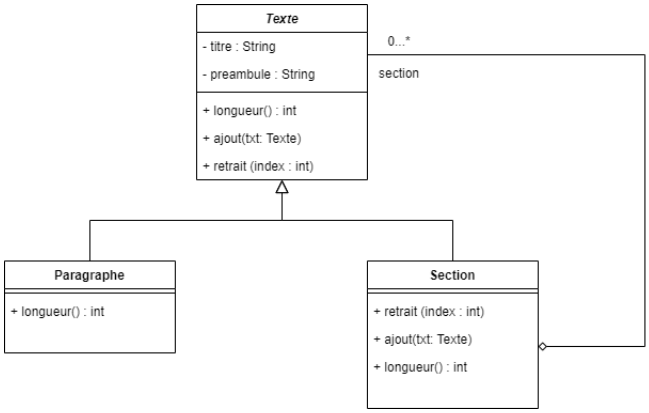

### 2.1.2 Résultats



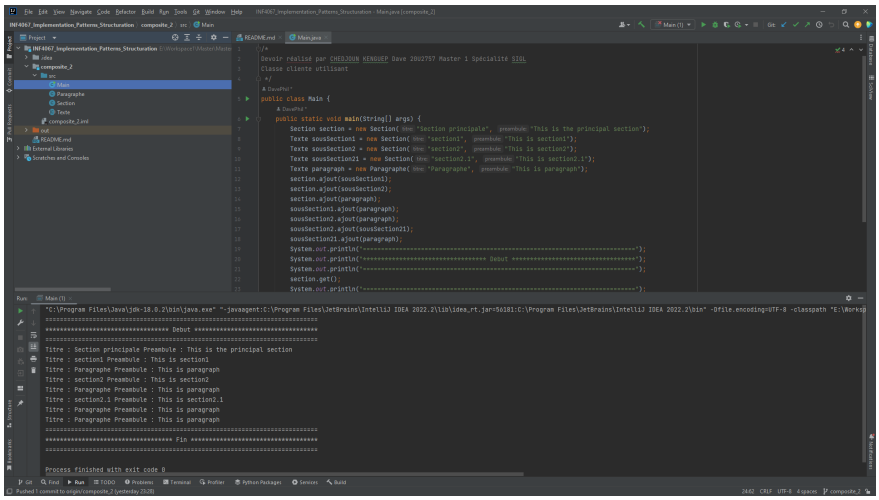
## 2.2 composite_2

La deuxième version du composite, correspond au deuxième exemple du cours, ou considère une section qui est un texte, et pouvant avoir plusieurs sous sections avec des paragraphes. l'implementation se trouve sur la branche **composite_2**.

### 2.2.1 Modèle



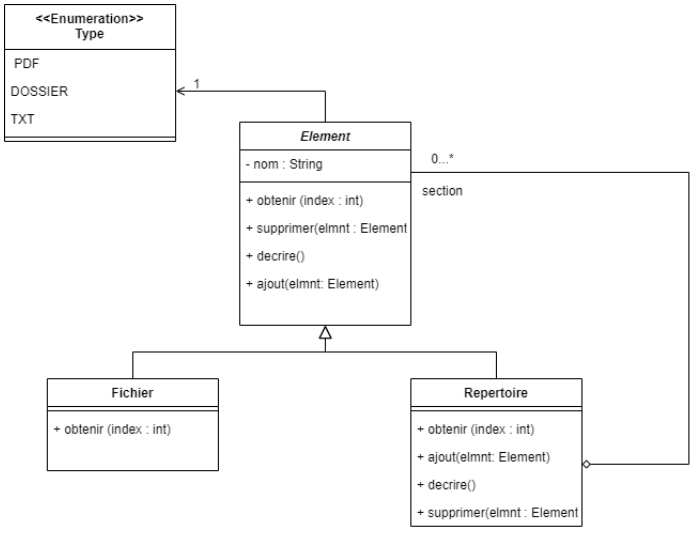

### 2.2.2 Résultats



## 2.3 composite_3

Il a été question ici, de résoudre un problème de gestions de fichiers. le système gère des repertoires, qui peuvent avoir des sous repertoires, ou alors des fichiers. de type PDF ou TXT. son implémentation se trouve sur la branche **composite_3**

### 2.3.1 Modèle




### 2.3.2 Résultats