# Composite

The composite pattern allows combining objects into a larger structure, representing a hierarchy of components. For instance, the composite pattern can be employed to depict an organizational hierarchy, a menu hierarchy, or a file hierarchy. It enables clients to treat individual objects and compositions of objects in the same manner. This can simplify client code and make it more generic.

The participants in this pattern are:

**The component**: Declares a common interface for all objects. Defines default behavior for all classes.

**Leaf**: This represents the leaf, implementing elementary behavior.

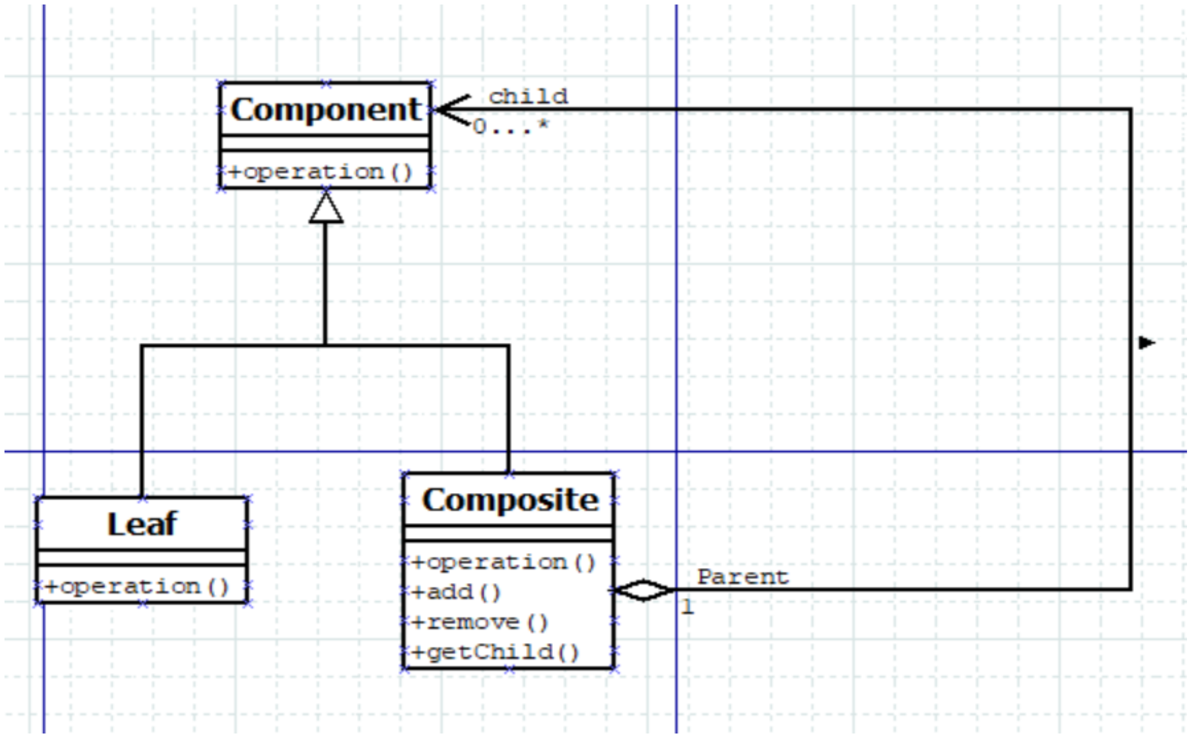
**Composite**: Defines the behavior of components with children, stores the children, and implements the necessary operations for their management.

**The client**: If the receiver is a leaf, the request is directly handled. Otherwise, the composite forwards it to all its children.

## 1. Initial

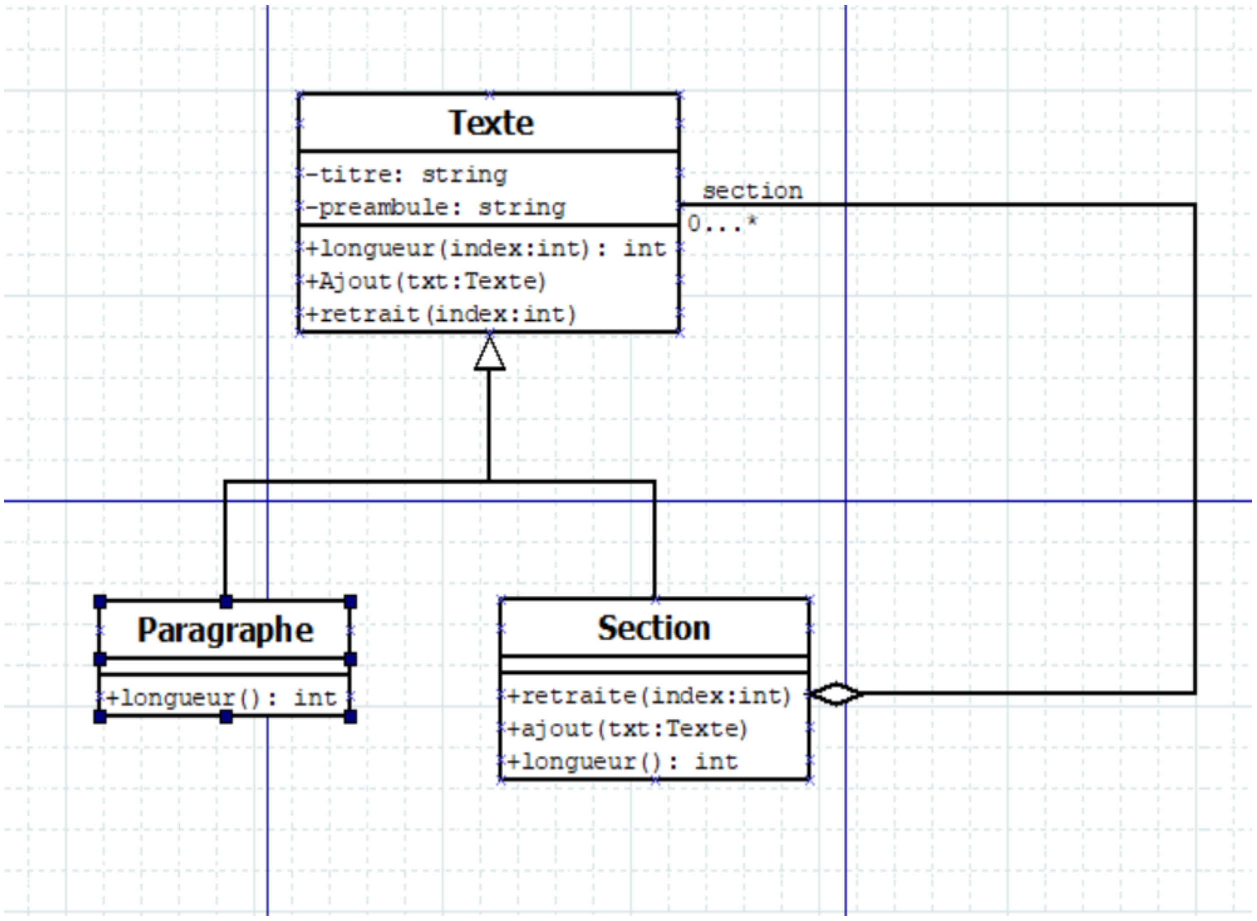
This first implementation represents the general model for the composite Class. It is as follows:

**Model**



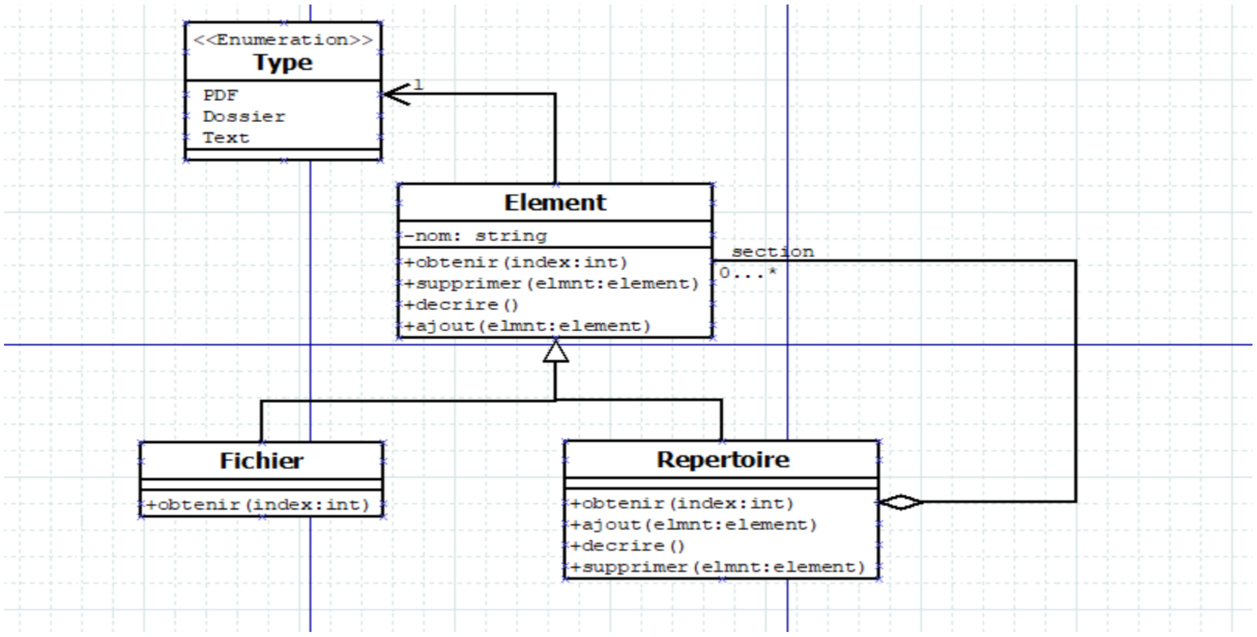
2. The second version of the composite corresponds to the second example in the course. It considers a section that is a text and can have multiple subsections with paragraphs. It is found in the composite pattern of the github repository.

**Model**



3. The question here was to solve a file management problem. The system manages directories, which can have subdirectories or files of the PDF or TXT type. Its implementation is found in the composite pattern folder of the github repository.

**Model**



<<Enumeration>>
Type
PDF
Dossier
Text
1
Element
-nom: string
+obtenir(index:int)
+supprimer(elmnt:element)
+decrire()
+ajout(elmnt:element)
section
0...*
Fichier
+obtenir(index:int)
Repertoire
+obtenir(index:int)
+ajout(elmnt:element)
+decrire()
+supprimer(elmnt:element)