المعهد الوطني للبريد والمواصلات
ⵓⵔⴰⵔⵉ ⵏⵓⵟⵟⴰⵏ ⵏ ⵜⵓⵍⵉⵖⵜ ⴰ ⵜⵉⵎⵓⵍⵉⵜ
Institut National des Postes et Télécommunications

Étude de cas : le design pattern *Composite*

Filière : SUD-Cloud & IoT

Travail réalisé par :

- Chrif El Asri Hanane
- El Gamous Khalid

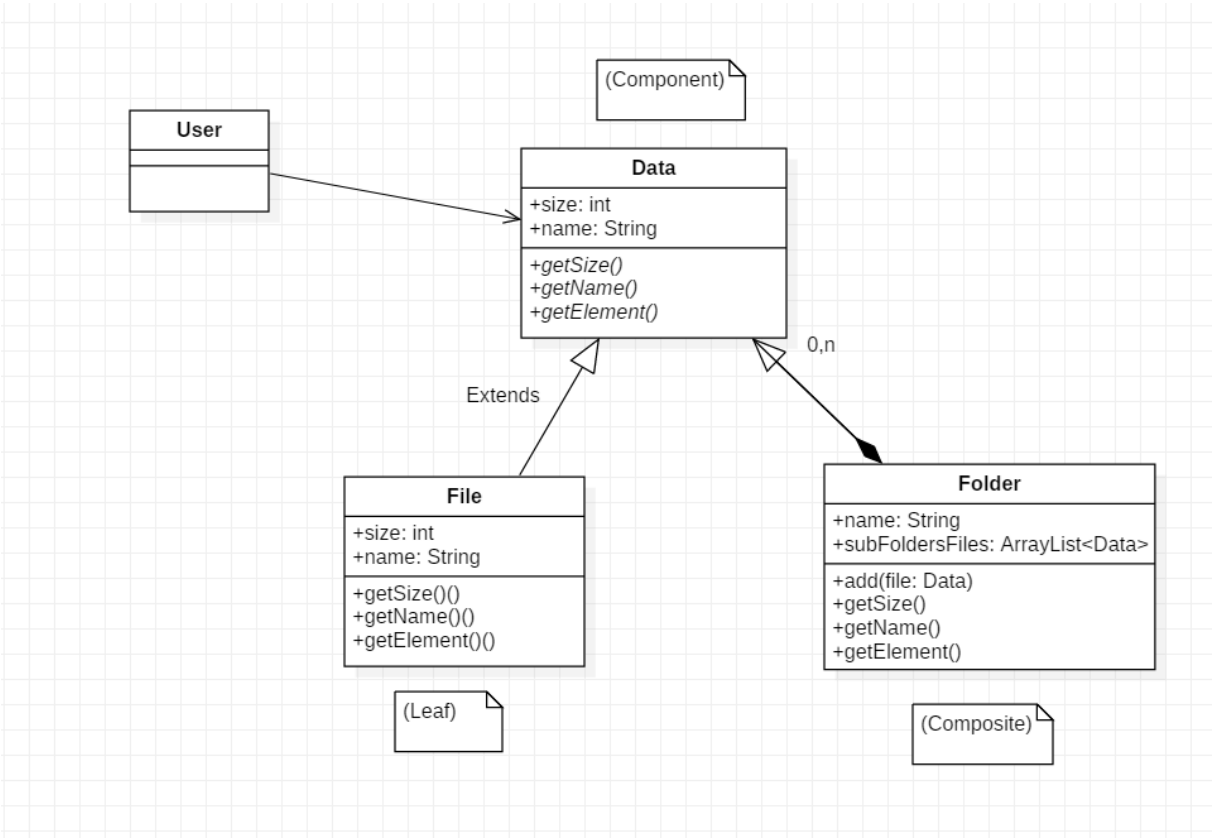
Encadrant:

Prof. Abdeslam EN-NOUAARY

Nous voulons développer une solution orientée objet pour gérer l'espace disque d'un utilisateur Windows. Notre espace disque est composés de fichiers et répertoires, ces derniers sont contenus dans des répertoires et possèdent un nom et une taille en nombre d'octets.
Un répertoire a donc une structure récursive qu'on peut en profiter en implémentant avec la même interface logicielle sur les fichiers et les répertoires afin qu'ils soient manipulés de la même manière.

Alors on propose d'avoir une classe Data qui est un **composant** général qui décrit les opérations communes (getSize, getName et getElement) aux objets simples (fichiers) et complexes (répertoire).
Ensuite on a deux classes File et Folder qui héritent de cette classe Data. La classe File représente une **Feuille qui** est un élément de base d'une branche qui n'a pas de sous-élément, dans notre cas un fichier. Pendant que la classe Folder représente un **conteneur** (*composite*) qui est un élément composé de sous-éléments : des feuilles ou d'autres conteneurs, dans notre cas un répertoire (contenant des fichiers et d'autres répertoires). Un conteneur ne connait pas les classes de ses enfants. Il passe par l'interface composant pour interagir avec ses sous-éléments.

Voici le diagramme de classe pour cette solution.



Pour pouvoir tester notre solution, on a besoin d'une classe User qui va contenir la méthode main. Pour notre test on va créer 2 répertoires (A et B) et 3 fichiers (a, b et c). On va les placer ensuite de la manière suivante :

```
Folder 1 -> file1, file2 and folder2
```

```
Folder 2 -> f3
```

On fait appel ensuite aux méthodes getSize() et getElement() pour le répertoire 1. Et voilà le résultat qu'on obtient :

```
Folder 1 size: 30
folder name : A has a size of : 30
file name : a has a size of : 15
file name : b has a size of : 10
folder name : B has a size of : 5
file name : c has a size of : 5
```

Si on applique les deux méthodes sur le répertoire 2 on obtient :

```
Folder 2 size: 5
folder name : B has a size of : 5
file name : c has a size of : 5
```

Il existe plusieurs exemples de problèmes similaires à cette étude de cas. Citons l'exemple d'une boîte qui peut contenir plusieurs produits ainsi qu'un certain nombre de boîtes plus petites. Ces petites boîtes peuvent également contenir quelques produits ou même d'autres boîtes encore plus petites, et ainsi de suite. On peut avoir recoure au design pattern composite si on a besoin d'affecter à chaque produit un prix et à chaque boite le prix de ses composants.

NB : Vous pouvez trouver le code source dans ce lien :

www.github.com/hanane-ca/Design-Patterns.git