14. Building a complex object hierarchy

Imagine that you need to construct a structure where objects need to act as containers to similar objects. It could be that you are building a tree-like data structure. Or perhaps you are building a representation of a file storage system where folders can contain files and other folders.

Suitable design patterns

Composite

Composite is a design pattern that allows you to build a tree-like structure in an efficient way.

There are two types of classes involved – a leaf class and a composite class. Both of these classes implement the same interface; however, a composite class would have more methods on it.

A leaf class is the simplest class in the structure. It cannot have any additional members. A composite class, on the other hand, can have a collection of members, which can be absolutely any type that implements the interface that both leaf and composite classes implement.

Essentially, a composite class can contain other instances of composite class and leaf instances. Therefore, a composite class would have methods that would allow it to manipulate the internal collection of its children, such as `Add`, `Remove`, etc.

The best analogy would be a file storage structure. You can think of files as leaf objects and folders as composite objects. A folder can contain other folders and files, while a file is a stand-alone object that cannot contain any other objects.

Why would you use Composite

- Really easy way of building any tree-like structures.
- Different types of members of the tree structure are easily distinguishable.