Motivation

- When dealing with Tree-structured data, programmers often have to discriminate between a leaf-node and a branch. This makes code more complex, and therefore, more error prone.
- An interface that allows treating complex and primitive objects uniformly
- The key concept is that you can manipulate a single instance of the object just as you would manipulate a group of them.

Composite Design Pattern

- A structural design pattern that lets you compose objects into tree structures.
- Define a unified Component interface for both part (Leaf) objects and whole (Composite) objects.
- Individual Leaf objects implement the Component interface directly, and Composite objects forward requests to their child components.
- This enables clients to work through the Component interface to treat Leaf and Composite objects uniformly
- This makes client classes easier to implement, change, test, and reuse.

Structure

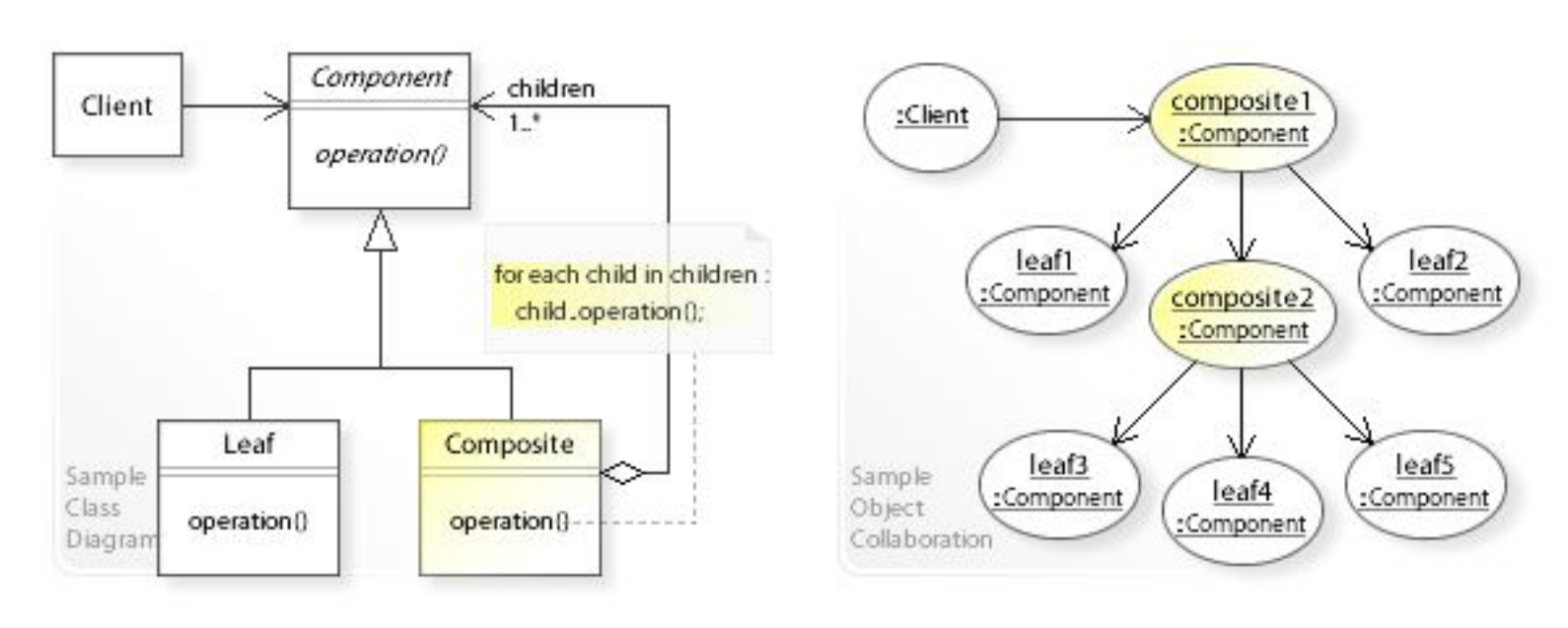

Image Source: w3sDesign.com

Code examples

When to use this pattern?

1. Use the Composite pattern when you have to implement a tree-like object structure with often having identical code to handle each of them.

2. Use the pattern when you want the client code to treat both simple and complex elements uniformly, thus making it easy to work with them.

3. Open/Closed Principle - You can introduce new element types into the app without breaking the existing code, which now works with the object tree. Thus the structure helps a lot in this.

When not to use?

1. Difficult to provide a common interface for classes whose functionality differs too much.

2. All classes in the hierarchy must follow the abstract interface that can lead to overly general classes. Thus prevent at such places where there is no heirarchy as such.

References

1. Composite Design Pattern at https://student.cs.uwaterloo.ca/~cs446/1171/Arch_Design_Activity/Composite.pdf
2. Composite Structural Pattern at https://refactoring.guru/design-patterns/composite
3. Head First Design patterns book
4. Design Patterns: Elements of Reusable object oriented software book