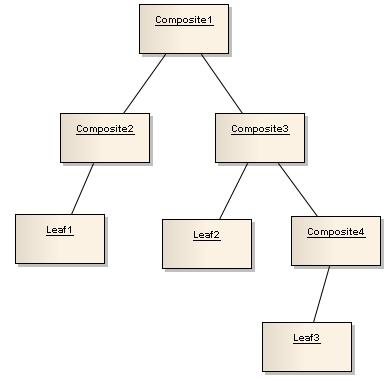
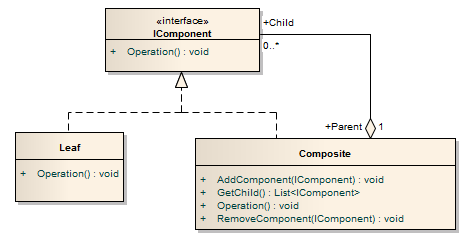
The [**composite design pattern**](http://www.devlake.com/) allows you to set up a tree structure and ask each element in the tree structure to perform a task. A typical tree structure would be a company organization chart, where the CEO is at the top and other employees at the bottom. After the tree structure is established, you can then ask each element, or employee, to perform a common operation.

The [**composite pattern**](http://www.devlake.com/) classifies each element in the tree as a composite or a leaf. A composite means that there can be other elements below it, whereas a leaf cannot have any elements below it. Therefore the leaf must be at the very bottom of the tree. The concept is shown in the diagram below:

[](http://www.devlake.com/)

Let's take a look at the [**UML**](http://www.devlake.com/UML-Quick-Reference)of the [**composite pattern**](http://www.devlake.com/) first, then we will do an example to see how it works. Below is the [**UML**](http://www.devlake.com/UML-Quick-Reference) of the [**Composite Design Pattern**](http://www.devlake.com/), where you see the distinction between a composite element and the leaf:

[](http://www.devlake.com/)

* The IComponent interface defines the methods that both the Composite class and the Leaf class must implement. The Operation method is the common method that all elements in the tree structure can perform. The IComponent simply represents an element in the tree.
* The Leaf class are elements that cannot have any elements below it, and it only has Operation method to perform the task for the element.
* The Composite class are elements that can have 0 or more elements below it. The methods that it supports are as follows:
  + The AddComponent method adds an element below it
  + The GetChild method gets all the elements below it
  + The Operation method performs the task for the element itself
  + The RemoveComponent method deletes an element below it