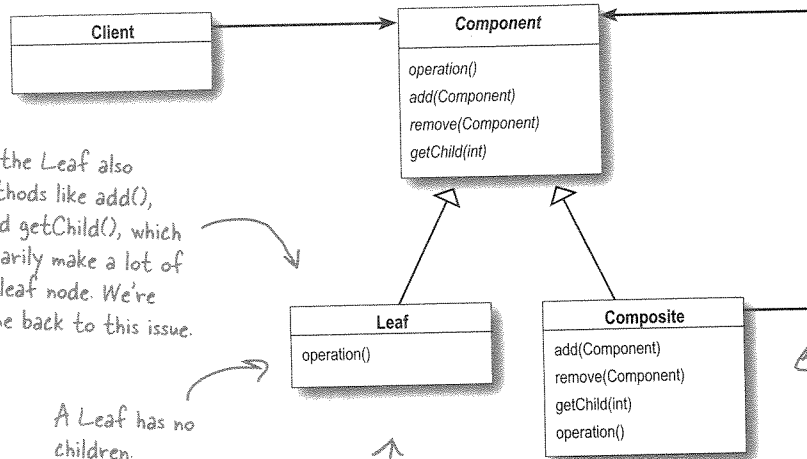


composite pattern class diagram

The Client uses the Component interface to manipulate the objects in the composition.

The Component defines an interface for all objects in the composition: both the composite and the leaf nodes.

The Component may implement a default behavior for add(), remove(), getChild() and its operations.



Note that the Leaf also inherits methods like add(), remove() and getChild(), which don't necessarily make a lot of sense for a leaf node. We're going to come back to this issue.

A Leaf has no children.

A Leaf defines the behavior for the elements in the composition. It does this by implementing the operations the Composite supports.

The Composite's role is to define behavior of the components having children and to store child components.

The Composite also implements the Leaf-related operations. Note that some of these may not make sense on a Composite, so in that case an exception might be generated.

Q: Component, Composite, Trees? I'm confused.

A: A composite contains components. Components come in two flavors: composites and leaf elements. Sound recursive? It is. A composite holds a set of children, those children may be other composites or leaf elements.

there are no Dumb Questions

When you organize data in this way you end up with a tree structure (actually an upside down tree structure) with a composite at the root and branches of composites growing up to leaf nodes.

Q: How does this relate to iterators?

A: Remember, we're taking a new approach. We're going to re-implement the menus with a new solution: the Composite Pattern. So don't look for some magical transformation from an iterator to a composite. That said, the two work very nicely together. You'll soon see that we can use iterators in a couple of ways in the composite implementation.