The ***Composite Pattern*** allows you to compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

* The Composite Pattern allows us to build structures of objects in the form of tress that contain both compositions of objects and individual objects as nodes.
* Using a composite structure, we can apply the same operations over both composites and individual objects. In other words, in most cases we can ignore the differences between composition of objects and individual objects.
* A Component is any object in a Composite structure. Components may be other composites or leaf nodes.
* There are many design tradeoffs in implementing Composite. You need to balance transparency and safety with your needs.

Component

operation()

add(Component)

remove(Component)

getChild(int)

Client

Composite

add(Component)

remove(Component)

getChild(int)

operation()

Leaf

operation()

All components must implement the “Component” interface; however, because leaves and nodes have different roles we can’t always define a default implementation for each method that makes sense. Sometimes the best you can do is throw a runtime exception.