

Design Patterns

Iterator & Composite Pattern

Eriq Muhammad Adams J.

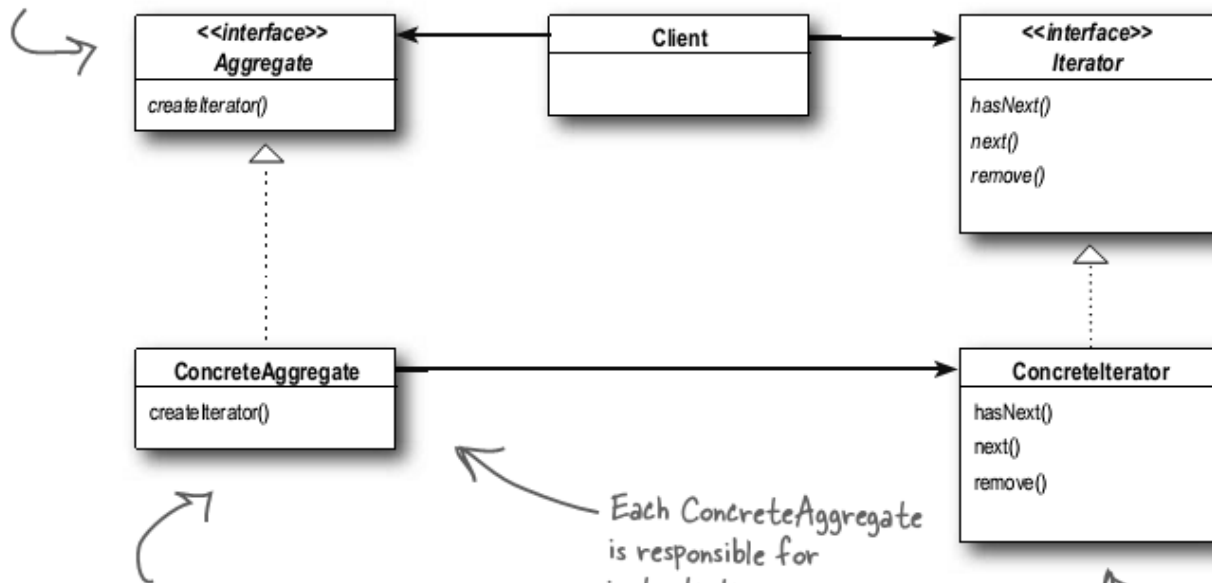
Mail : eriq.adams@ub.ac.id | Blog : <http://eriq.lecture.ub.ac.id>

Iterator Pattern

- * provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Iterator Pattern (cont.)

Having a common interface for your aggregates is handy for your client; it decouples your client from the implementation of your collection of objects.



The Iterator interface provides the interface that all iterators must implement, and a set of methods for traversing over elements of a collection. Here we're using the `java.util.Iterator`. If you don't want to use Java's Iterator interface, you can always create your own.

The **ConcreteAggregate** has a collection of objects and implements the method that returns an **Iterator** for its collection.

Each **ConcreteAggregate** is responsible for instantiating a **ConcreteIterator** that can iterate over its collection of objects.

The **ConcreteIterator** is responsible for managing the current position of the iteration.

Composite Pattern

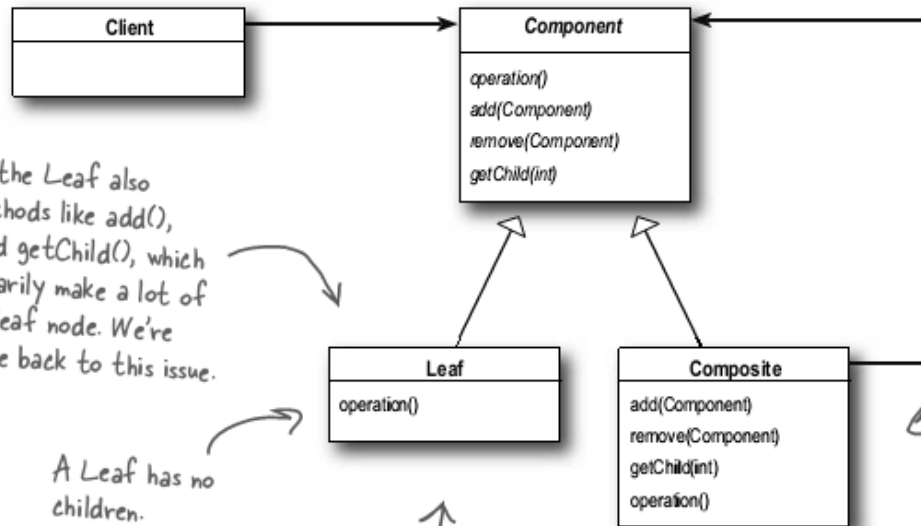
- * allows you to compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly

Composite Pattern (cont.)

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.

References

- * O'Reilly – Head First Design Pattern by Eric Freeman & Elisabeth Freeman (2004).
- * CRC Press – Software Architecture Design Pattern in Java by Partha Kuchana (2004).