

Composite

Linda Marshall and Vreda Pieterse

Department of Computer Science
University of Pretoria

(2014)

Overview

- 1 Identification
- 2 Structure
- 3 Participants
- 4 Observations
- 5 Related Patterns
- 6 Example 1 - Graphic
- 7 Example 2 - Tree

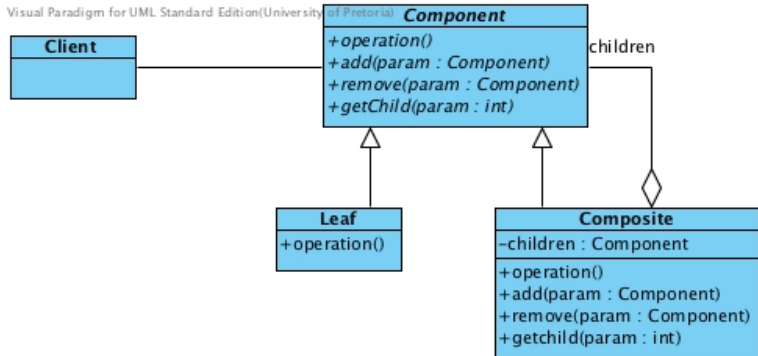
Name and Classification:

Composite (Object Structural)

Intent:

“Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.”

GoF(163)



Component

- provides the interface with which the client interacts

Leaf

- do not have children, define the primitive objects of the composition

Composite

- contain children that are either composites or leaves

Client

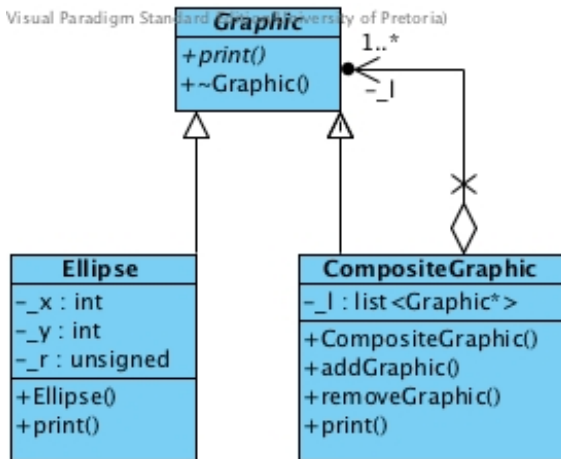
- manipulates the objects that comprise the composite

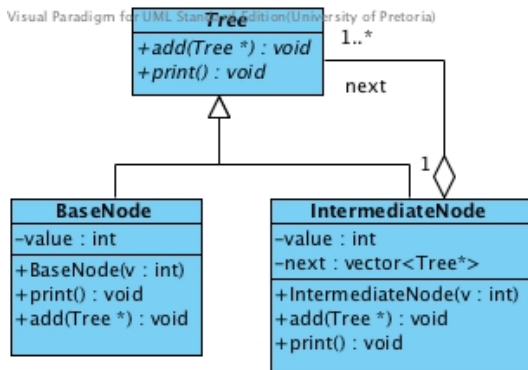
- Used in hierarchies where some objects are composites of others
- Makes use of a store for the children defined by Composite

Related Patterns

- **Chain of Responsibility** (223) :
component-parent link.
- **Decorator** (175): Used in conjunction
with components. Usually share the
same parent class.

- **Flyweight** (195): Allows sharing of objects, particularly the leaf nodes.
- **Iterator** (257) and **Visitor** (331): Used to traverse the composite structure.





```
class Tree {  
    public:  
        virtual void add(Tree*) = 0;  
        virtual void print() = 0;  
        virtual ~Tree() {}; // Added  
};
```

```
class BaseNode : public Tree {  
    public:  
        BaseNode(int v) : value(v) {};  
        virtual void print() {...};  
        virtual void add(Tree*) {};  
        virtual ~BaseNode() {}; // Added  
    private:  
        int value; };
```

```
class IntermediateNode: public Tree {  
    public:  
        ...  
        // Add  
        virtual ~IntermediateNode();  
    private:  
        ...  
};
```

```
IntermediateNode::~~IntermediateNode(){  
    vector<Tree*>:: iterator it;  
  
    for (it = next.begin(); it != next.e  
        delete *it;  
}
```

```
delete b;  
    // Not linked into Tree t  
    // and therefore needs to  
    // be deleted separately  
delete t;
```