

# Composite

Linda Marshall

Department of Computer Science  
University of Pretoria

22 August 2022

## Name and Classification:

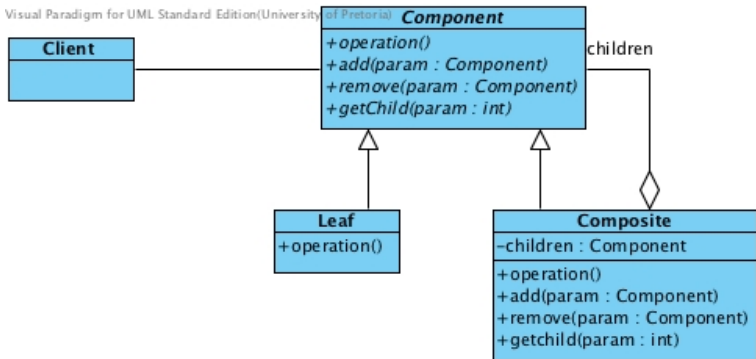
Composite (Object Structural)

## Intent:

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

GoF(163)

Visual Paradigm for UML Standard Edition (University of Pretoria)



## 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

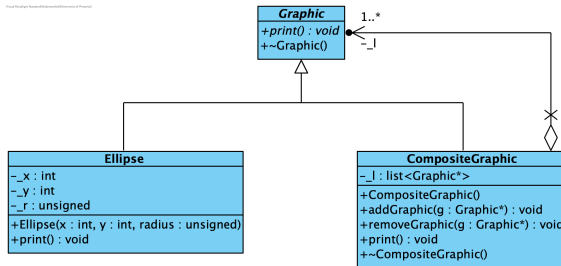
- Used in hierarchies where some objects are composites of others
- Makes use of a “*structure*” for the children defined by Composite

## 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.

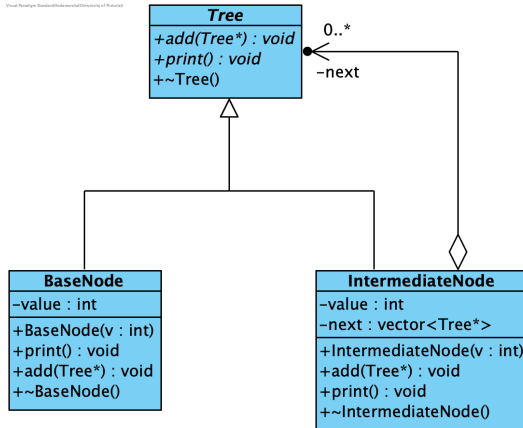




```
// Anonymous objects
class A { // all the class stuff };

class B {
    public:
        B(A* in) { a = in; };
        virtual ~B() {delete a; };
    private:
        A* a;
};

int main() {
    // Some client code
    B b(new A());
    return 0;
}
```



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

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

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

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

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