Composite Design Pattern

Jonathan Williams

CSPP 51023 - OO Architecture, Design & Methodology

Composite Definition

- Gang of Four Definition:
  - Allow you to compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.[1]
- My interpretation:
  - If you have objects within the domain that have a hierarchical organization or if you have objects that need to be treated uniformly, use the structural composite pattern.

Possible examples where Composite might makes sense

- Menus with menu items, which could also be a menu itself.
- Employee manager to subordinate hierarchies.
- Colleges with divisions that have departments.
- Directories with files, but within a directory there could be other directories.

The Composite pattern not only allows you to represent the objects as a tree but also allows you to easily manipulate individual and composite objects in the same way.[2]

Composite Class Diagram[3]

Client

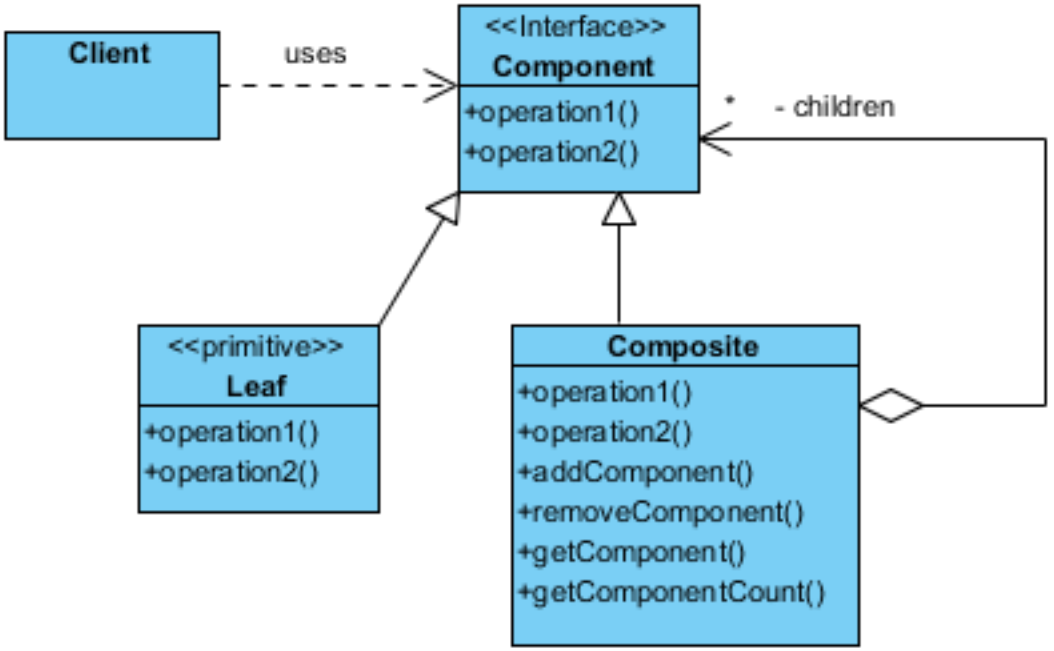
- Utilizes component interface to "uniformly" manipulate objects within the composite group.

Component

- Abstract class that defines the interface to all objects in the composed group (leafs & compositions) for manipulation.
- Implements default behavior that is used by all classes.

Leaf

- Represents a primitive object or a composite without components.
- Implements behavior.

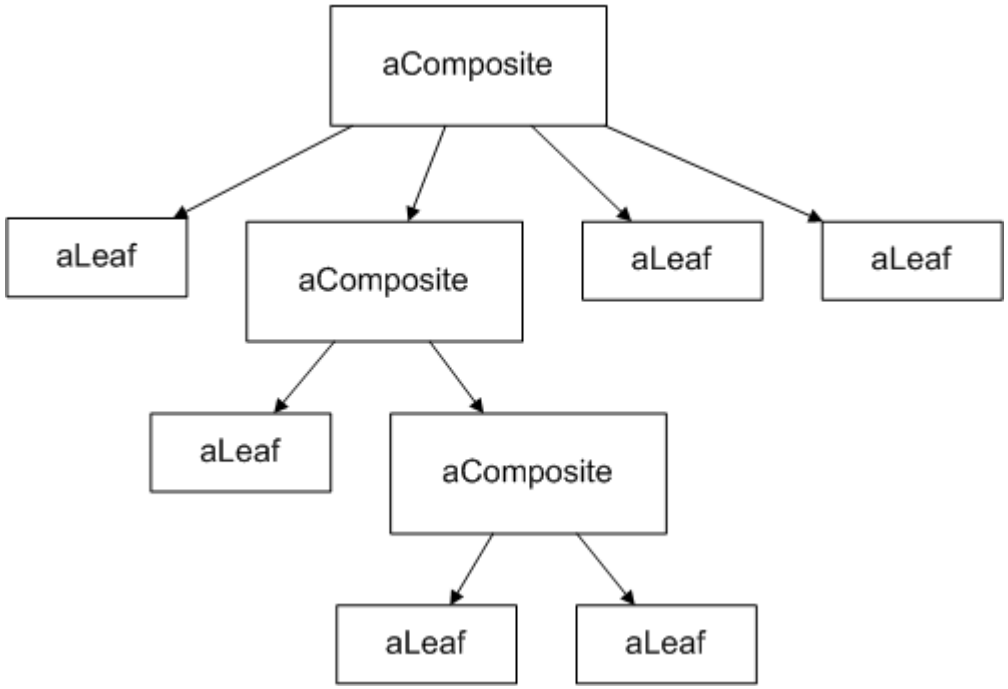


Source: Erich Gamma, et al, "Composite" *Design Patterns*. (Boston: Addison-Wesley, 1995) 164.

Composite

- Extends base class that represents primitive objects.
- Maintains a collection of children components.
- Implements child operations in the component interface.

Graphical representation of a Composite object structure



Source: Erich Gamma, et al, “Composite" *Design Patterns*. (Boston: Addison-Wesley, 1995) 165.

Example code 1 of 2

Component interface

Graphic.java | CompositeGraphic.java | SimpleGraphic.java | Client.java

```
// Component interface
public interface Graphic {
    public void add (Graphic g);
    public void remove (Graphic g);
    public Graphic get (int index);
    public void paint();
}// end graphic
```

Composite

Graphic.java | CompositeGraphic.java | SimpleGraphic.java | Client.java

```
import java.io.*;
import java.util.ArrayList;

// Composite
public class CompositeGraphic implements Graphic {

    private ArrayList<Graphic> children = new ArrayList<Graphic>();

    public void paint() {
        //paint operation
        for(Graphic g: children) {
            g.paint();
        } // end for
    }// end paint

    public void add(Graphic g) {
        children.add(g);
    } // end add

    public void remove(Graphic g){
        if (children.contains(g)){
            children.remove(g);
        }// end if
    } // end remove

    public Graphic get(int index){
        if(index < children.size()){
            return children.get(index);
        } // end if
        return children.get(1);
    }//end get
} // composite
```

Example code 2 of 2

Leaf

Graphic.java | CompositeGraphic.java | **SimpleGraphic.java** | Client.java

```
import java.io.*;
import java.util.ArrayList;

//Leaf
public class SimpleGraphic implements Graphic{
    public void paint (){
        //run paint
    }// end pain

    public void add(Graphic g){
        // leaf can't do this
    } // end add

    public void remove (Graphic g){
        // leaf can't do this
    } // end remove

    public Graphic get(int index){
        throw new IllegalArgumentException("is invalid");
    } // end get

} // end SimpleGraphic
```

Client example

Graphic.java | CompositeGraphic.java | SimpleGraphic.java | **Client.java**

```
// pseudo code code for a Client
public class Client{
    // given a graphic can call paint regardless if
    //it's a leaf or Composite

        paint(Graphic g){
            g.paint();
        }// end paint

} // end Client
```

Soureces

1. Erich Gamma, et al, “Composite.“ *Design Patterns*. (Boston: Addison-Wesley, 1995) 163-174.
2. Ralph Johnson, “An Introduction to Patterns“ *The Patterns Handbook: Techniques, Strategies, and Applications*. Ed. Linda Rising. (New York: Cambridge University Press, 1998) 354.
3. Erich Gamma 354-357.