

The Composite Design Pattern

November 18, 2009

The general form for the composite pattern is illustrated in Figure 1. The motivation for the composite pattern is to provide an interface that allows treatment of both complex and primitive objects uniformly. The interface, shown in Figure 1, is class `Component`, and the classes that will be treated uniformly are `Leaf` and `Composite`. However, in a typical implementation there will likely be lots of leaves.

The famous and authoritative resource, *Wikipedia*, provides an interesting code example of the composite pattern, illustrated in Figure 2. An illustration of the data structure that results from this pattern is illustrated in Figure ??.

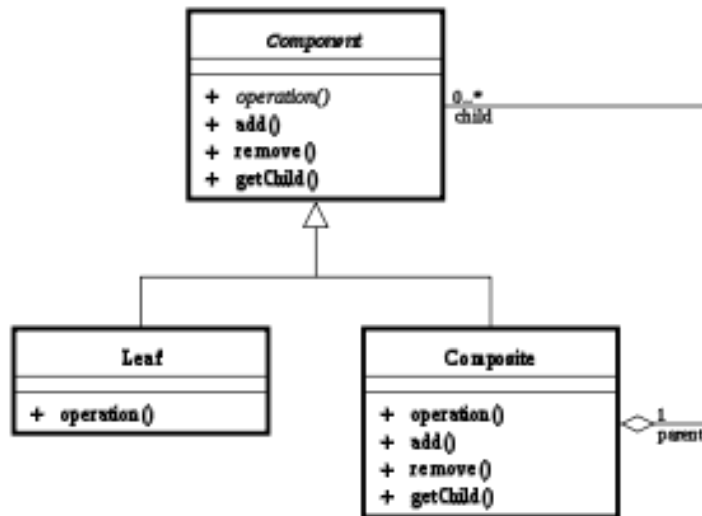


Figure 1: General Form for Composite Pattern.

```

class Component {
public:
    virtual void traverse() = 0;
};

class Leaf: public Component {
    int value;
public:
    Leaf(int val) { value = val; }
    void traverse() { cout << value << ' '; }
};

class Composite: public Component {
    vector < Component * > children;
public:
    void add(Component *ele) {
        children.push_back(ele);
    }
    void traverse() {
        for (int i = 0; i < children.size(); i++)
            children[i]->traverse();
    }
};

int main() {
    Composite containers[4];

    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 3; j++)
            containers[i].add(new Leaf(i *3+j));

    for (int i = 1; i < 4; i++)
        containers[0].add(&(containers[i]));

    for (int i = 0; i < 4; i++) {
        containers[i].traverse();
        cout << endl;
    }
    return 0;
}

```

Figure 2: Code example of Composite Pattern, taken shamelessly from Wikipedia.

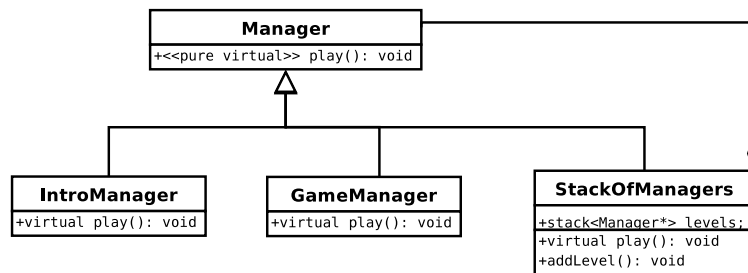


Figure 3: Use of Composite Pattern in Games.
