Composite ID

Student:

Izabela Kuźniar

Teacher:

Andrea Corradini

Course:

Software Design Patterns

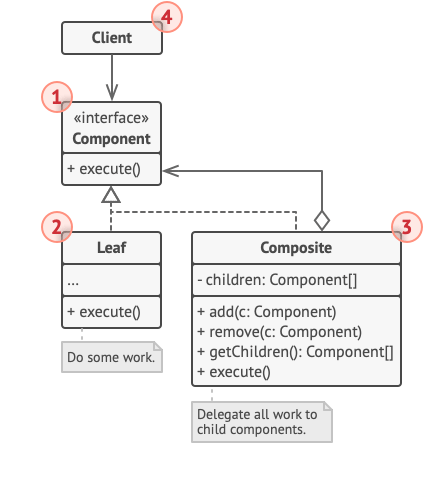
# Name and category

Composite, also known as Object Tree is a structural pattern.

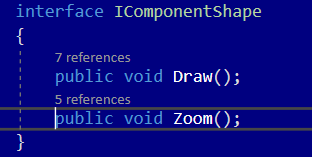
# Intent:

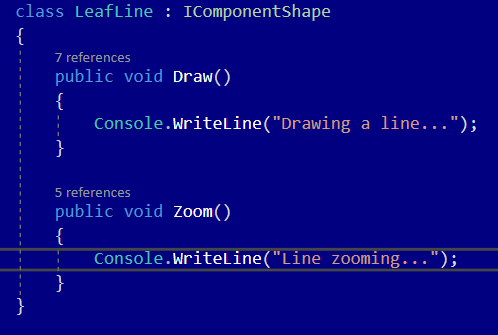
Helps compose objects into tree structures and then allows working together with these structures as if they were individual objects.

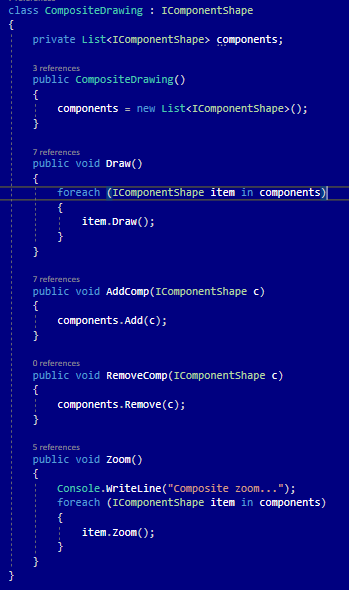
# Structure as a UML class diagram

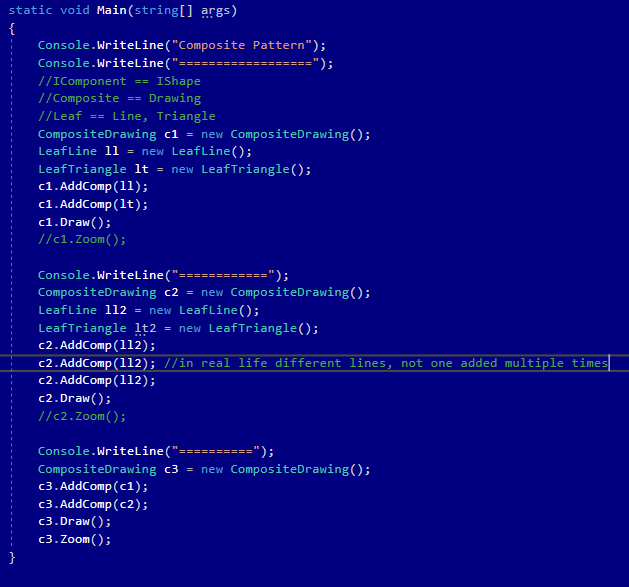


# Implementation:









# Consequences:

Benefits:

* Ability to work with complex structures, trees more conveniently: use polymorphism and recursion.
* Open/Closed Principle. Ability to introduce new element types into the app without breaking the existing code, which now works with the object tree.

Drawbacks:

* Probable difficulties in providing the common interface for classes with completely different functionalities. In some cases there is a need to overgeneralize the component interface.

# Known uses

* Java GUI layouts and widgets

# Related patterns

1. Ability to use Builder when creating the complex Composite trees, and so ability to program its construction steps to work recursively.
2. Chain of Responsibility is often used in conjunction with Composite. When a leaf gets a request, it may pass it further to the parent components down to the root.
3. Ability to traverse Composite trees using iterators.
4. To execute an operation over an entire Composite tree we can use Visitor.
5. Leaf nodes od the Composite tree implemented as Flyweights can save up some RAM.
6. Composite and Decorator have similar structure diagrams since both rely on recursive composition to organize an open-ended number of objects.
7. Applying Prototype Pattern gives the ability to clone complex structures (Composite trees) instead of re-constructing them again.