

Composite Pattern

There are times when a program needs to manipulate a tree data structure and it is necessary to treat both Branches as well as Leaf Nodes uniformly. Consider for example a program that manipulates a file system. A file system is a tree structure that contains Branches which are Folders as well as Leaf nodes which are Files. Note that a folder object usually contains one or more file or folder objects and thus is a complex object where a file is a simple object. Note also that since files and folders have many operations and attributes in common, such as moving and copying a file or a folder, listing file or folder attributes such as file name and size, it would be easier and more convenient to treat both file and folder objects uniformly by defining a File System Resource Interface.

Intent

- The intent of this pattern is to compose objects into tree structures to represent part-whole hierarchies.
- Composite lets clients treat individual objects and compositions of objects uniformly.

Implementation

The figure below shows a UML class diagram for the Composite Pattern:

Composite Pattern

Motivation

There are times when a program needs to manipulate a tree data structure and it is necessary to treat both Branches as well as Leaf Nodes uniformly. Consider for example a program that manipulates a file system. A file system is a tree structure that contains Branches which are Folders as well as Leaf nodes which are Files. Note that a folder object usually contains one or more file or folder objects and thus is a complex object where a file is a simple object. Note also that since files and folders have many operations and attributes in common, such as moving and copying a file or a folder, listing file or folder attributes such as file name and size, it would be easier and more convenient to treat both file and folder objects uniformly by defining a File System Resource Interface.

Intent

- The intent of this pattern is to compose objects into tree structures to represent part-whole hierarchies.
- Composite lets clients treat individual objects and compositions of objects uniformly.

Implementation

The figure below shows a UML class diagram for the Composite Pattern:

- Component - Component is the abstraction for leafs and composites. It defines the interface that must be implemented by the objects in the composition. For example a file system resource defines move, copy, rename, and getSize methods for files and folders.

- Leaf - Leafs are objects that have no children. They implement services described by the Component interface. For example a file object implements move, copy, rename, as well as getSize methods which are related to the Component interface.

- Composite - A Composite stores child components in addition to implementing methods defined by the component interface. Composites implement methods defined in the Component interface by delegating to child components. In addition composites provide additional methods for adding, removing, as well as getting components.

- Client - The client manipulates objects in the hierarchy using the component interface.

A client has a reference to a tree data structure and needs to perform operations on all nodes independent of the fact that a node might be a branch or a leaf. The client simply obtains reference to the required node using the component interface, and deals with the node using this interface; it doesn't matter if the node is a composite or a leaf.

Applicability & Examples

The composite pattern applies when there is a part-whole hierarchy of objects and a client needs to deal with objects uniformly regardless of the fact that an object might be a leaf or a branch.

Example - Graphics Drawing Editor.

In graphics editors a shape can be basic or complex. An example of a simple shape is a line, where a complex shape is a rectangle which is made of four line objects. Since shapes have many operations in common such as rendering the shape to screen, and since shapes follow a part-whole hierarchy, composite pattern can be used to enable the program to deal with all shapes uniformly.

In the example we can see the following actors:

- Shape (Component) - Shape is the abstraction for Lines, Rectangles (leafs) and and ComplexShapes (composites).
- Line, Rectangle (Leafs) - objects that have no children. They implement services described by the Shape interface.
- ComplexShape (Composite) - A Composite stores child Shapes in addition to implementing methods defined by the Shape interface.
- GraphicsEditor (Client) - The GraphicsEditor manipulates Shapes in the hierarchy.

Alternative Implementation: Note that in the previous example there were times when we have avoided dealing with composite objects through the Shape interface and we have specifically dealt with them as composites (when using the method `addToShape()`). To avoid such situations and to further increase uniformity one can add methods to add, remove, as well as get child components to the Shape interface. The UML diagram below shows it:

Source: [Click here to see java source code](#)

Specific problems and implementation

Graphics Editors use composite pattern to implement complex and simple graphics as previously explained.

File System implementations use the composite design pattern as described previously.

Consequences

- The composite pattern defines class hierarchies consisting of primitive objects and composite objects. Primitive objects can be composed into more complex objects, which in turn can be composed.
- Clients treat primitive and composite objects uniformly through a component interface which makes client code simple.
- Adding new components can be easy and client code does not need to be changed since client deals with the new components through the component interface.

Related Patterns

Decorator Pattern - Decorator is often used with Composite. When decorators and composites are used together, they will usually have a common parent class. So decorators will have to support the Component interface with operations like Add, Remove, and GetChild.

Known Uses

File System Implementation as discussed previously.

Graphics Editors as discussed previously.