BENEFITS

The following lists the benefits of using the Composite pattern:

- Defines class hierarchies consisting of primitive objects and composite objects
- Makes it easier to add new kinds of components
- Provides flexibility of structure and a manageable interface

WHEN TO USE

You should use the Composite pattern when:

- You want to represent the whole hierarchy or a part of the hierarchy of objects.
- You want clients to be able to ignore the difference between compositions of objects and individual objects.
- The structure can have any level of complexity, and is dynamic.

Decorator Pattern

The Decorator pattern enables you to add or remove object functionality without changing the external appearance or function of the object. It changes the functionality of an object in a way that is transparent to its clients by using an instance of a subclass of the original class that delegates operations to the original object. The Decorator pattern attaches additional responsibilities to an object dynamically to provide a flexible alternative to changing object functionality without using static inheritance. Figure 4–9 illustrates the Decorator pattern.

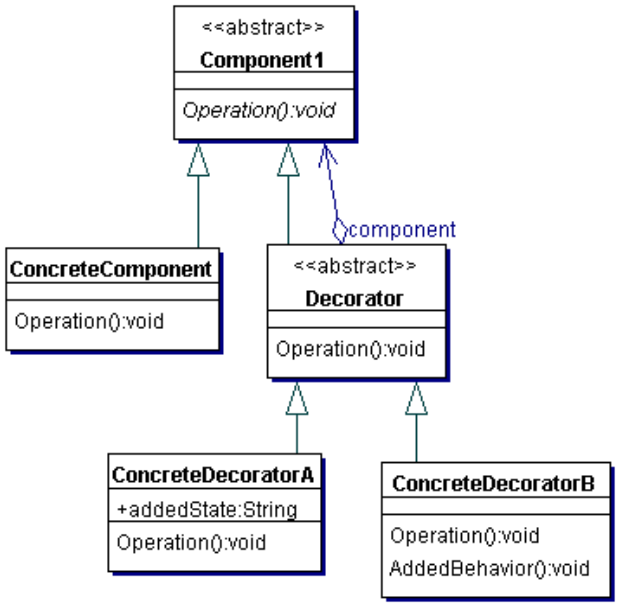

Figure 4–9 *The Decorator Pattern*

BENEFITS

The following lists the benefits of using the Decorator pattern:

- More flexibility than static inheritance
- Avoids feature-laden classes high up in the hierarchy
- Simplifies coding because you write a series of classes, each targeted at a specific part of the functionality, rather than coding all behavior into the object
- Enhances the object's extensibility because you make changes by coding new classes

WHEN TO USE

You should use the Decorator pattern when:

- You want to add responsibilities to individual objects dynamically and transparently, that is, without affecting other objects.
- You want to add responsibilities to the object that you might want to change in the future.
- When extension by static subclassing is impractical.

Façade Pattern

The Façade pattern provides a unified interface to a group of interfaces in a subsystem. The Façade pattern defines a higher-level interface that makes the subsystem easier to use because you have only one interface. This unified interface enables an object to access the subsystem using the interface to communicate with the subsystem. Figure 4–10 illustrates the Façade pattern.