Decorator and Composite

2-17-2004

Opening Discussion

- Virtuals in constructors and destructors don't behave like virtuals.
- Check assignment to self and copy everything in operator=.
- Page 63 in the Java book lists the things you need to do/should do to make a class immutable. Immutable objects are generally safer and sharing often reduces memory usage.
- Inheritance breaks encapsulation.
- Do you have any questions on the reading for today?

Recap of the Bridge

- I did a horrible job of describing Bridge last time so let's redo that before hitting new patterns.
- Remember that the idea is to separate the interface from the implementation and have two separate type hierarchies for each.
- The example the GoF uses is a windowing system where you have different types of windows and different platforms for the windows. A singe hierarchy causes real problems here.

Decorator

- This pattern lets you attach extra responsibilities to an object dynamically.
- The way it works is that you wrap an existing object inside of another object that will pass on calls to the existing object, but also has extra processing or extra abilities.
- This pattern is also called a wrapper because that is basically what the decorator does.
- You have an abstract base type and all decorations inherit from it.

Example

- The Java I/O libraries are built on this pattern. Things like BufferedInputStream and DataInputStream decorate other InputStreams.
- The GoF book uses an example from a graphics program where things like scrollbars and borders are added as decorators on components that go into the graphics.

Benefits and Drawbacks

- This is more flexible than just inheriting from a class to add extra features. Allows easier mix and match.
- It's easy to create and add new decorator types later so you don't have to think of all of them to start with.
- It can cause problems with object identity because an object wrapped in a decorator isn't that object itself.
- This creates systems with lots of little objects that can be hard to understand.

Composite

- This pattern allows for the composing of large objects from smaller objects in a tree-like structure. The main benefit being that all the objects are treated the same way.
- We get this by having all of the objects inherit from the same interface/superclass. This way all that we see is a tree of nodes and it doesn't matter that each node might actually be of a different type, running different code.

Example

- Graphics hold great examples of the Composite pattern. I have used one in my PAD2 class that we can look at. The idea being that we have a tree of some Node type, but it is really made from many subtypes.
- Scene graphics in 3D graphics are similar. The GoF book uses an example from typesetting or word processing where a tree represents the document and nodes can be text, drawings, etc.

Benefits and Drawbacks

- Allows you to build complex structures with a simple interface for the client.
- Makes it very easy to add new types of components, simply make a new class that extends the interface.
- The main drawback is that the interface will typically be fairly restrictive and you have to use dynamic type information if you want specific information on the a particular subtype.

Progress Reports

- Who wants to present progress reports today?