Assignment 6 (Project)

Design Patterns

Introduction

In class, we covered a variety of design patterns. In this assignment, you get a chance to implement three of them: Decorator, Observer, and Composite. You may use any Object-Oriented programming language to implement the patterns: C++, Java. Note: only implement the minimal amount of code you need to demonstrate the design pattern (e.g. these should be relatively small programs).

Instructions

1. Write a program that demonstrates the advantages of the Observer pattern. Your program should create an object that has state that needs to be observed. It should attach at least two observers to this object and it should then change the state of the observed object twice. Each time, the observers should indicate that they received the state change notification and retrieved and/or did something with the new state.
2. Write a program that demonstrates the advantages of the Decorator pattern. Create an object with two operations that produce output when they are called. Create a decorator for that object that adds two additional operations that produce output when they are called. (For simplicity, you should have the operations write to standard out, e.g. do not try to implement a GUI program that makes use of decorators graphically.) Instantiate an undecorated object and have your program call both operations. Then instantiate a decorator, use it to "decorate" the original object, then call all four operations provided by the decorator. The output of the original two operations should be decorated in some way. That is, it should be obvious that the original output of the undecorated object is being manipulated in some way by the decorator.
3. Write a program that demonstrates the advantages of the Composite design pattern. The program should create a tree of objects consisting of both composite and leaf nodes (demonstrating that the composite has correctly implemented a subset of the child management functions, such as `addChild()`, `getChild()`, and `getParent()`). Make sure that your tree consists of at least three levels, e.g. a root node with multiple children, where at least one of those nodes is a composite node with children. The program should then call an operation on the root of the tree. All composite objects should delegate the operation to their children while all leaf nodes should perform the action. Note: you can optionally have the

composite nodes do something in response to the operation but they should then delegate the operation to their children.

Evaluation

We will be looking to see that your implementations match the **structure** of the design pattern as well as match the scenarios described above. Please create an archive of your source code in .zip format and submit to BB. Your archive should contain a single directory that contains three subdirectories, one for each pattern. Each pattern directory should contain the source code for implementing the pattern and a text file containing the output produced by the program.