Introduction to Object-Oriented Design Patterns
Suggested Exercises

Exercise 1

1. Create a version of a standard data structure (stack, queue, set) or select one from a Java, C++, or similar library.
2. Define a different but functionally equivalent interface to the same data structure (via a Java interface or C++ class with no state and pure virtual functions).
3. Develop a small driver program with respect to the new, different interface.
4. Have students individually or in small teams develop an object adapter that:
    a. Conforms to the new interface.
    b. Has a private instance variable referencing an object of the original data structure.
    c. Forwards calls on the new interface methods to the corresponding methods in the original data structure object. There may be some simple pre or post processing involved.

Exercise 2

1. Start with the same setup as in Exercise 1.
2. Have students develop a class adapter that:
    a. Inherits from the original data structure class.
    b. Inherits from the desired interface.
    c. On calls to the new interface methods, simply calls the corresponding methods in the original data structure (possibly with some simple pre and post processing).
3. Note that this can be done in either Java or C++ if the desired interface has no state or functionality as indicated in Exercise 1.

Notes

I have had success with a simple stack:

- The desired interface has four methods, push: pop, top, and empty.
    - Push and pop are pure mutators (e.g., they return no value)
    - Top and empty are pure observers (e.g., they return information without changing the state).
- The implemented interface has three methods: push, pop, and size.
    - Push is a pure mutator as before.
    - Pop removes and returns the top value in the stack.
    - Size is an observer that returns the number of elements in the stack.

This approach forces students to do a bit of work in the adapters while driving home the point that the adapter is, for the most part, a simple interface translator.

Exercise 3

Have students reflect on their programming experiences to date and provide a short summary of patterns they've observed in their code. It is unlikely that these will be design patterns; it is more likely that they will identify coding patterns for loops, etc., that frequently recur. They should

write up at least one such pattern using the GOF format (though the overall document should be limited to 1-2 pages).