Assignments

Each GoF pattern discussion is accompanied by a set of programming exercises intended to allow you to explore the requirements and possibilities of implementing a particular pattern's solution. The exercises are all part of the same program. The program is an implementation of a simple XML parser. Each exercise will expand the program with new capabilities or refactor the design. The exercises are grouped into clusters of two to three that work together and form a single assignment. The exercises do not necessarily require code changes for a solution. Some are complete as the code currently exists. Some patterns are not suitable for the application described. In those cases you may state this and explain why or what alternate pattern applies.

I have provided a framework, described below, for implementing the program by providing a partial implementation of a collection of interfaces that duplicates a subset of the org.w3c.dom Java package functionality, a class that recognizes a subset of the tokens that make up XML to simplify parsing a file, a class that converts a DOM tree back into a text file, a class that validates the structure of an XML document, and a translation of those classes into C++ and Swift for those who would rather use those languages. There are entry points for test programs in the Java tokenizer, serialization and validation classes and a separate main() function for the C++ and Swift classes. You may translate the program into another language.

The names of new classes and methods should reflect their roles in the pattern that they implement. Comments should be added to the declaration of existing classes to identify their roles. For example if the Null Object pattern were applied to the DOM structure, the new class might be called NullNode, and the null methods have comments that identify the fact that they do nothing.

The grade for each assignment will be determined by how faithfully the implementation follows the pattern according to the table in section "2. Assignments" in the Syllabus. If the code requires no modification to implement the pattern, or that an existing class or method fills a role in the pattern without modification, or that the pattern is not applicable be sure to document that decision and the reasoning. (The reason that this project is too small or doesn't require the flexibility provided by the pattern is not acceptable since the assignments are small by necessity. This statement does not contradict the last entry in the "Appropriateness" row of the table.)

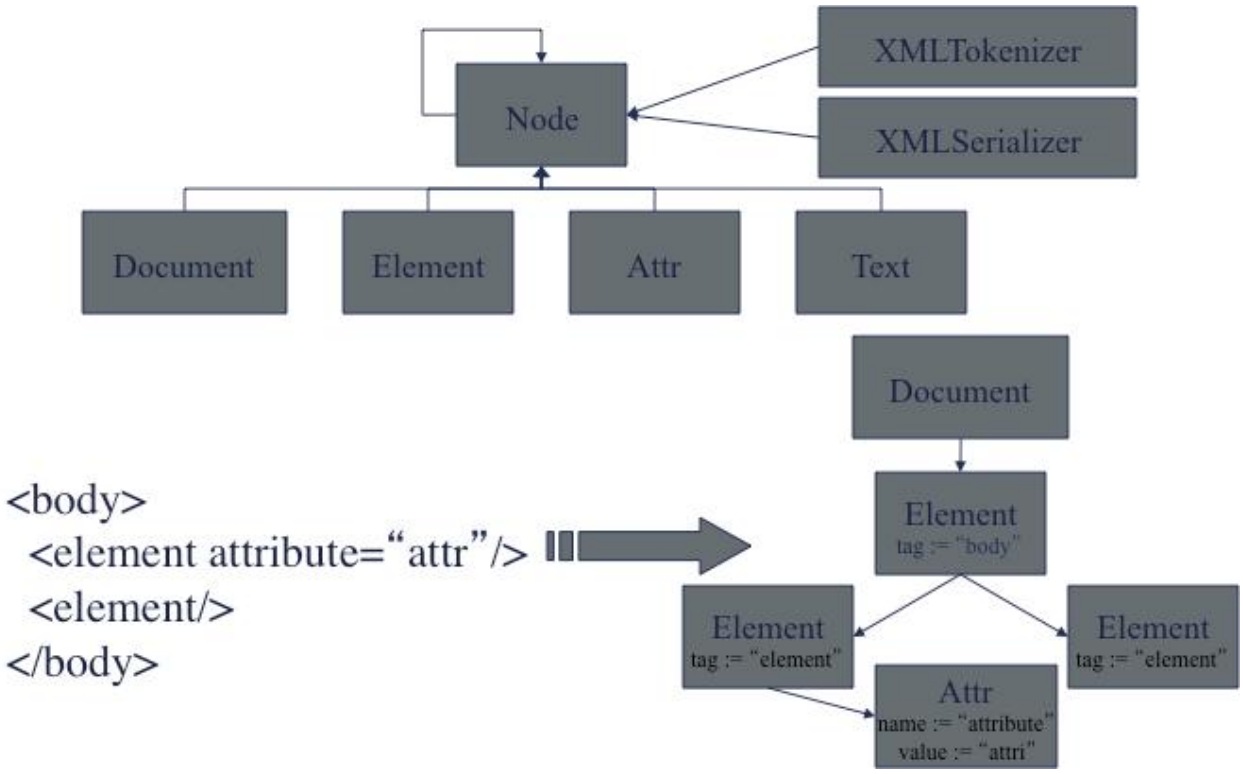
The following items should be turned in for each assignment:

- Source code including the make or build.xml files and excluding object files. The pattern participants in the code should be clearly highlighted with a comment containing the role name as given in the description of the pattern's structure. For example, the implementation of Factory Method should have the "Creator", "Product", "ConcreteCreator", "ConcreteProduct" and "FactoryMethod" roles commented with exactly those names.
- Class diagram clearly identifying the classes participating in each pattern or any changes made to existing classes to fit a pattern.
- Brief description of the roles filled by classes in the pattern.

Homework should be turned in electronically in a zip file or tar file via the submission page on Blackboard. The existing make or build.xml files should be used. Turn in only the framework language that you use, i.e., don't turn in the C++ code if you are using the Java version. If you are using the C++ version, please stick to C++ and POSIX libraries, avoiding any Windows-specific code; command-line interaction is adequate for all of the assignments. Assignments are due at the time given on the Blackboard submission page for each.

Assignment Framework

The framework illustrated in the figure consists of a reimplementation of the interfaces in Java's org.w3c.dom, a minimal implementation of those interfaces (enough of the methods to complete the assignments) and some additional classes that exercise the DOM structure. The interfaces are reimplemented in the "replacement" package because some of the assignments require modifying them, which is not practical for the interfaces in the Java API package. The C++ and Swift versions follows the same structure as nearly as possible; the primary difference being that there is a separate test program instead of separate main() methods in the XMLTokenizer, XMLSerializer and XMLValidator classes. The algorithms and relationships in the Java, Swift, and C++ implementations are identical.



The top of the figure shows the DOM interfaces which can be found in the edu.jhu.apl.patterns_class.dom.replacement package and the dom namespace in C++. There is a corresponding implementation class for each interface in the Node inheritance hierarchy in the edu.jhu.apl.patterns_class.dom package and the default namespace in C++. Finally the three classes that exercise the DOM structure are found in edu.jhu.apl.patterns_class and the default namespace.

XMLTokenizer reads an XML file and extracts the simple tokens to simplify converting the file into the representative DOM structure. XMLSerializer performs the opposite, taking an input DOM structure and converting it back to a file. XMLValidator, not shown in the figure, implements a configurable algorithm for checking whether or not an input XML file follows a particular structure. For example, if the validator were configured for the XML file in the figure it would signal an error if it encountered an Element with any name other than “element” inside of the Element named “body”. Each of these classes contains a main function illustrating the class’s use. In the C++ version, the test program contains the test code for all three classes.

The bottom part of the figure shows a simple XML file, and the DOM structure that would result from parsing that file. Alternatively it shows a simple DOM structure and the XML file that would result from serializing it.

The framework also includes a couple of small test XML files, a build.xml file for building the Java version, an XCode project file for Swift, a CMakeLists.txt file for building the C++ version with GCC, and the open source regex library to support the C++ XMLTokenizer class. Please use the build.xml or Makefile supplied with the framework for building your assignment.