CS 635 Advanced Object-Oriented Programming
Spring Semester, 2007
Assignment 2

San Diego State University -- This page last updated 2/12/10

Assignment 2
Due Feb. 25

This assignment will build on assignment one.

1. Review your units tests for adding elements to the binary tree from assignment 1. Make sure that the tests adequately test adding elements to the tree and obtaining the elements in the tree that end in 'n' or 's'. Record those tests. When you are done with the assignment determine how good the tests were. That after making the changes required in this assignment you were confident that worked after running the tests. Did you have add to or modify your tests?

2. Refactor your tree code to use standard names for methods, remove helper methods on the tree that deal with tree nodes, and any other clean up you feel is needed in your code. You will still need the helper method for finding strings ending in 'n' or 's'. You might find the refactorings rename and move useful here. In Eclipse these refactorings can be found in the Refactoring menu.

3. The binary search tree class in assignment one is a collection. Determine the correct location in your language's collection class hierarchy. Find **all** methods that you need to implement in-order to add your class in the language's collection class hierarchy. (C++ people get a pass on this problem as STL is painful to subclass.) Note we will only be interested in implementing one of these methods - see problem 4.

4. Make the parent class of your binary search tree the parent determined in problem 1. Rename your existing methods to conform to the collection classes standards. One may need to stub some methods to satisfy the parent class's constraints.

5. Use the Null Object pattern to add a null node to your tree to eliminate the need to check for null references or pointers in your tree.

6. Implement an iterator for your binary search tree. It is tempting to convert the tree into a collection and the then use the iterator from the collection. This is a quick trick that can be useful. However the point of this assignment is to help us understand how to implement patterns, which this trick will not help us do. So don't use the trick. Ruby people can use an internal iterator if they wish, but may find it instructional to implement an external iterator.

7. Implement what we will for now will call EndsInIterator. Using Java syntax the class will have the methods given below. One temptation here is to read all the elements into a collection and then return an iterator on that collection. This assumes that all the elements we are iterating over are all available at one time and will fit into memory. This is not always the case, for example one may be iterating over elements in a very large file. So do not use this trick either. C#, Ruby and C++ people may need to implement different methods to conform to their language's conventions.

EndsInIterator(Iterator input, char[] endings) - constructor

boolean hasNext() - returns true if the iteration has more elements that end with one of the characters in endings.

next() - returns the next element in the iteration that ends with one of the characters in the array endings.

Grading

Item	Percent of Grade
Working Code	15%
Unit Tests	10%
Proper implementation of Patterns	60%
Quality of Code	15%

Proper implementation of Patterns. The goal of the **assignment** is to better understand the iterator and null object patterns. So 60% of your grade is on producing a proper implementation of the patterns.

What to Turn in

Turn in hard copy of your code and unit tests. Do not turn in Question for Thought listed below.

Late Policy

The penalty for turning in the assignment late is 3% of the grade per day. Once solutions to the assignment have been posted or discussed in class no more late assignments will be accepted.

Questions for Thought

1. Internal iterators have proven very useful in Smalltalk, yet do not exist in C++ or Java. How would you implement an internal iterator in C++ or Java? How easy would it be to use your internal iterator? For C++ take a look at the Boost Lambda library.

2. What do you see as the advantages and disadvantages of using the null object pattern in the binary search tree?

3. One problem with external iterators is that collection you are iterating over can change while the iterator still exists. This can put the iterator in an undefined state. (How?) One solution is to make the iterator robust, that is ensure that insertions and deletions do not change interfere with the iterator. (see page 261 of the text). How would you make your iterator robust?

4. Java introduced fail-safe external iterators in JDK 1.2. These iterators allow the collection being iterated over to be changed by the iterator. However, if the collection is changed not using the iterator then the next time you call a method on the iterator an exception is thrown. How would you implement a fail-safe iterator?

5. Which type of external iterator is better a robust or fail-safe? Why?

6. How would you implement a factory method to return a polymorphic external iterator? What is the advantage of doing this?

7. The text states that polymorphic iterators in C++ must be on the heap. Explain why this is true.

8. In Smalltalk one can use the method become: to turn object A into object B. That is object A will become an instance of the B's class. All previous references anywhere in your program to B will be changed to refer to A. B will be turned into a instance of A' class and all previous references to A will now refer to B. Is this possible in Java or C++?

9. One of the listed disadvantages on the NullObject pattern is that NullObject objects can not transform themselves into a RealObject. Does the become: method negate this disadvantage in Smalltalk?