Assignment 2
Doubly Linked List Revisited

The goal of this assignment is to improve on your assignment one and implement Iterator, null object, decorator and strategy patterns.

Due Feb 28

1. Write XUnit tests for adding elements to the doubly linked list from assignment 1. Make sure that the tests adequately test adding elements to the list. Record those tests. When you are done with the assignment determine how good the tests were. That is after making the changes required in this assignment how confident are you that your code works after running the tests. Did you have to write new tests as you refactored your code?

2. Refactor your tree code to use standard names for methods, remove helper methods on the linked list that deal with nodes, and any other clean up you feel is needed in your code. You might find the refactorings rename and move useful here. In Eclipse these refactorings can be found in the Refactoring menu.

3. The linked list class 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.)

4. Make the parent class of your linked list class the parent determined in problem 3. Rename your existing methods to conform to the collection classes standards. You may need to stub some methods to satisfy the parent class’s constraints. Note we will only be interested in implementing a few of these methods. You do not have to implement all the methods in the parent class. We will need at least the add method, toArray and the toString method. As in assignment 1 don’t use arrays or other collection classes to implement your linked list.

5. Use the strategy pattern to allow you determine how a linked list object will be ordered when a linked list object is created.

6. Implement an iterator for your linked list. Don’t covert your linked list to an array or other collection to implement your iterator.

7. Use the null object pattern to represent head and tail nodes in the linked list. Can you remove any null checks?

8. Implement what we will for now will call OnProbationFilter. Using Java syntax the class will have the methods given below. People using other languages may need to implement different methods to conform to their language’s conventions.
OnProbationFilter(Iterator input) - constructor. Make sure that the argument is an Iterator.

boolean hasNext() - returns true if the iteration has more student elements that are on probation.

next() - returns the next student element in the iteration that is on probation.

9. Create a decorator class OnProbationDecorator that decorates your Linked list class. The decorator modifies the toString, toArray and iterator methods to return just student elements that are on probation. Does this make sense as a decorator?

10. For those using Java are you able to use the methods forEach and stream() methods on your Linked List class?

Grading

<table>
<thead>
<tr>
<th>Item</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working Code</td>
<td>10</td>
</tr>
<tr>
<td>Unit Tests</td>
<td>10</td>
</tr>
<tr>
<td>Proper implementation of Patterns</td>
<td>10 per Pattern (50 points total)</td>
</tr>
<tr>
<td>Quality of Code</td>
<td>10</td>
</tr>
<tr>
<td>Proper Parent Class &amp; Method names</td>
<td>10</td>
</tr>
<tr>
<td>Written Answers to questions 1, 7, 9, 10</td>
<td>10</td>
</tr>
</tbody>
</table>

Turning in your Assignment

Turn in an electronic copy of your code. Include a README file that contains answers to questions 1, 7, 9, 10. The assignment is given as a series of steps. Turn in the code as it is when you have finished all steps. Do not turn in multiple copies of the same class. Zip up all your source files and upload the zip file to the course portal (http://bismarck.sdsu.edu/CoursePortal).