Project 2: Doubly-Linked Lists, 225 points

(based on Prof. Alvarado's writeup).

Due: Wednesday , January 20, at 23:59:59

Overview

1. You will create a lot of JUnit tests to check your methods.
2. In Project 2 you will implement a doubly-linked list, including an Iterator over that list. In particular, you will implement the List12 interface and several additional methods. Note that you *must* implement a doubly-linked list -- you cannot, for instance, instead implement an array-based list.
3. You will implement ListIterator class in order to traverse a list.
4. You will complete a list of Java related questions.
5. If you want more challenge, there are a few extra credit problem at the end.
    a. Submit it(them) under a separate assignment submission on Vocareum.

What to submit:

Details

Part 1 - Understanding and Testing First

In this homework assignment you will be developing a doubly linked list. In part 2 you'll be implementing DoublyLinkedList12 but in this part you will develop a tester to test the methods that you will implement.
NOTE: **For this part only (the tester), you are allowed to discuss and share ideas about test cases with your classmates.** This is to make the process of developing tests a little more fun.

First: Understand what DoublyLinkedList12 will do

In order to write a good tester, you need a deep understanding of how the classes and methods you are testing are supposed to work. So before you start writing your tester, read part 2 to understand what DoublyLinkedList12 classes are supposed to do.

Tester for DoublyLinkedList12

Download the supplied file LinkedListTester.java. This is a starter file and it defines a small number of tests against the Java Collection's Framework LinkedList class.

- Compile and run LinkedListTester.java as-is.
- Create a set of meaningful tests on the public methods described in part 2 which you are responsible for implementing.
- Notice that LinkedListTester.java tests Java's built-in LinkedList class. This is to allow you to have something to test against before you have your DoublyLinkedList12 class implemented. Make sure your tests pass against Java's LinkedList class.
- In your README file, briefly describe what each of these tests attempts to validate. This will test against LinkedList's.
- **After you have completed at least some of part 2, and are ready to test your** DoublyLinkedList12 **class, copy** your LinkedListTester.java and rename it to be DoublyLinkedList12Tester.java. (Don't forget to redefine the class from LinkedListTester to DoublyLinkedList12Tester), and modify the tests to create DoublyLinkedList12Tester objects.
- Notice you do not have to change the type of the iterator because your MyListIterator class ***is-a*** ListIterator. However, during testing you may have used the "QQQ" methods to create your list iterator for testing (see below). Be sure in the final tester that you submit that these methods have been renamed to the iterator() and listIterator() methods that are in the List interface.
- In the end, you will be graded on how well your tester find errors across a buggy implementation of the linked lists.

Part 2 – Doubly Linked Lists

Note: you should not allow "null" objects to be inserted into the list. You should throw a NullPointerException if the user attempts to do so. Note, that LinkedList from Java Collection's Framework allows you to store null objects.

Download a new DoublyLinkedList12.java, it has method headers and you need to fill in the gaps. Your DoublyLinkedList12 class is just a subset of the Java Collection's Framework LinkedList, and therefore should match its behavior on the described subset.

To make your life easier make sure that your class extends AbstractList, If you are not sure where @*Override* tag came from, then you need to read the online documentation on AbstractList. What happens if you do not override the add() method?

You may use "dummy" head and tail nodes if you wish (I recommend this because I think it's easier), but you are not required to do so.

In doubly linked lists, the removal of items can be especially tricky as you need to be sure to properly update the neighbor's next and previous pointers, as well as handle the special cases for removal from the head (front) or tail of the list.

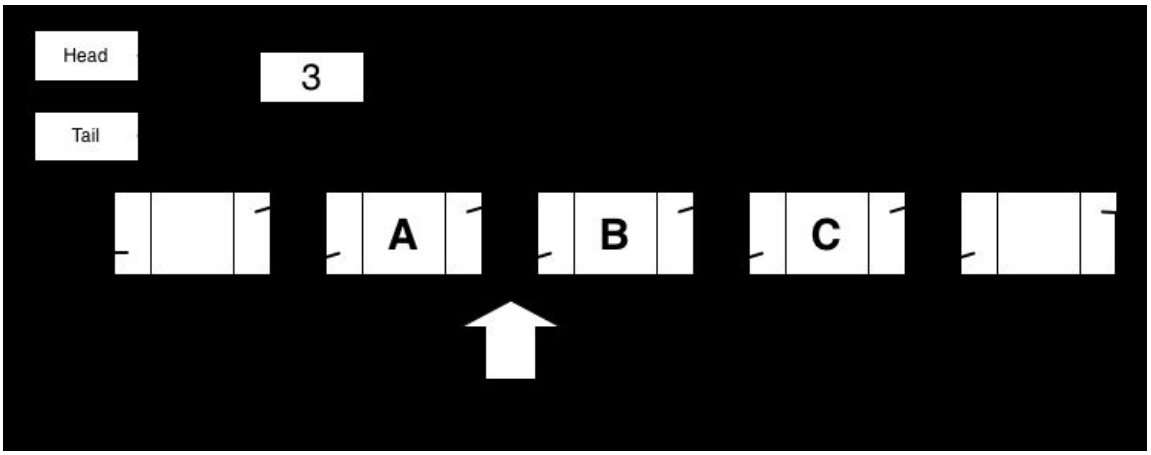
Your program will be graded using an automatic script. It will also be manually inspected (to make sure you actually implemented a linked list and not an array-based list). Try to anticipate the kinds of tests we will run on your code -- how might we "stress" your code to its limit?

Constructors
You should only need to have a single public 0-argument constructor that creates an empty list and initializes all the necessary variables.

Part 3 – Iterators for DoublyLinkedList12

Once your DoublyLinkedList12 class is working, you will need to create a ListIterator which is returned by the **listIterator()** method. The best approach is to create an inner class (contained inside DoublyLinkedList12 class).

The ListIterator is able to traverse the list by moving a space at a time in either direction. It might be helpful to consider that the iterator has size+1 positions in the list: just after the sentinel head node (i.e. just before the first node containing data), between the 0 and 1 index, ..., just before the sentinel tail node (i.e., just after the last node containing data).



Which Exceptions to throw for ListIterator?

Your methods should properly throw IllegalStateException, NoSuchElement, and NullPointer exceptions. This means you DO NOT HAVE TO IMPLEMENT throwing ALL exceptions as the javadoc would indicate. In addition to the documentation above, the starter file already indicates through the method header which exceptions you should be throwing in which methods.

Programming Hints

It is useful to have a number of state fields to simplify the writing of the various methods above. Since the cursor of the ListIterator exists logically between 2 nodes, It is useful to just keep references to the next Node and the previous Node. It may also be helpful to keep an int value of the index of the next node.

If you construct your DoublyLinkedList12 to use sentinel nodes as discussed in the book and lecture, and you properly throw exceptions for going out of range, you shouldn't have to worry about checking for null values at the ends of the list since the sentinel nodes are there.

Since set() and remove() both change based on what direction was last being traversed, it makes sense to keep a boolean flag to indicate a forward or reverse direction.

Public Methods to add to MyLinkedList.java

Once you are **sure** your iterator is working, you should override the following methods in DoublyLinkedList12. Each of these should just create a new MyLinkedListIterator and return it.
ListIterator<T> listIterator()
Iterator<T> iterator()

have these factory methods return your ListIterator class for the current list.
Note: You inherit a working ListIterator from AbstractList, but the one you create will be more efficient. We suggest that while you are building and initially testing your ListIterator, you create a differently named factory method to use. You could use names like QQQiterator() and QQQlistIterator() (as we do in the starter code) until you are sure it is working correctly. If you jump right into overriding iterator()/listIterator() then things like toString() may stop working for you.
Be sure to have it called just iterator() and listIterator() in your submitted MyLinkedList and JUnit code.

Apportioning your Time

This is a much more comprehensive assignment than HW1. If you approach it incorrectly, it will take too long. Suggested amount of time and order of doing things. Don't try to do the assignment all at once.

1. Write tests against LinkedList that will help you develop your Linked list implementation without testing the ListIterator class (beyond what is already in the supplied starting tests). This will help you understand the LinkedList interface. You (obviously) only need to write tests for methods that are common to both LinkedList and DoublyLinkedList12. Make sure your test suite runs properly against LinkedList instances. (1-2 hours)
2. Copy/rename your testing class to DoublyLinkedList12Tester, compile it, run it against the supplied outline of DoublyLinkedList12. You should have many test failures.
3. Develop the linked list methods (Node inner class, methods). Develop until all the tests you developed passed.
4. Develop some test methods for testing the iterator (see the code and notes above). The ListIterator is more involved because of tracking states. Try some tests that move the iterator forwards and backwards. (2 - 4 hours)
5. Develop the ListIterator implementation using your tests to help you (2 - 3 hours).
6. Uncomment the lines near the end of the starter DoublyLinkedList12.java file, to make your ListIterator active.

Part 4 – Java Related Questions

True/False. Create a text file called Problem4.**txt** and write down then answers with the number of the question followed by either the word *True* or *False*. **One answer/line. eg.**

1. True
2. False

and so on. This allows us to grade this part electronically.

> This part is open book and open notes. You may use Google to help you determine the answers to these questions, and you may run any Java code to help you determine the answers. However, you may not ask your classmates for the answers nor may you give the answers to any of your classmates. The point is to *understand* the answers, as we assume that you have this knowledge from CSE 11 or CSE 8B and we will build on it.

1.	T	F	An instance variable declared as private can be seen only by the class in which it was declared and all its sub classes
2.	T	F	If a class C is declared as abstract, then private C myC = new C(); is valid.
3.	T	F	A variable declared as final cannot ever be modified, once it has been declared and initialized.
4.	T	F	The following is a legal statement: double x = 5;
5.	T	F	Code that does not explicitly handle checked exceptions, results in a compilation error.
6.	T	F	You are not able to sort a two dimensional array using the sort() method
7.	T	F	You are not able to change variable values from a subclass
8.	T	F	method declarations void A(double x, integer k){}; and void A(integer k, double x) {}; have identical signatures
9.	T	F	The binary search algorithm will work properly on all integer arrays.
10.	T	F	Interface is a class in Java
11.	T	F	("Give me Liberty".split(" ").length) evaluates to 3
12.	T	F	String.equals and == will always return the same result
13.	T	F	If the following statements are the only two statements in a method, String X = "thing one"; and String Y="thing one"; then X.equals(Y) evaluates to true, but X == Y evaluates to false within that method.
14.	T	F	A class declared as final cannot be inherited via the extends keyword.
15.	T	F	consider the statement: String S = "Out of Gas"; then the statement: S[7] = 'g'; will change "Gas" to "gas".
16.	T	F	boolean primitive variables can only be assigned values: true, false, or null.
17.	T	F	You can index into an array with a variable of type double as long as the there are no digits past the decimal point.
18.	T	F	A subclass inherits all of the *public, private* and *protected* members of its parent if the subclass is in the same package as its parent
19.	T	F	Any for loop can be rewritten using a while loop.
20.	T	F	It is legal to define more than one class in a java source file.
21.	T	F	A class can implement multiple interfaces
22.	T	F	int i = Math.sqrt(4.0); is a valid statement.
23.	T	F	the protected keyword can only be applied to instance variables.
24.	T	F	Consider the statement: throw new IllegalArgumentException(); This always causes the program to immediately exit.
25.	T	F	Suppose you have the following declaration: int xyz = 4; Then, in the body of a switch statement block

Part 5 – Extra credit problems

You have a few problems to choose from

1 (5 points). In your DoublyLinkedList12.java class add a method that takes a linked list and conctatenates the reversed version of itself. Original list is unchanged. For example: Input list is (3, 5, 7) the output list is (3, 5, 7, 7, 5, 3). Provide a file with a JUnit test(s) as well.

Method signature: *public void reverseAndConcat (*DoublyLinkedList12<E> list *);*

2. (5 points) You will use your DoublyLinkedList12 to write a DNA splicing simulation program. Specifically, you are going to implementing restriction enzyme cleaving. For the background on the biology of this process, you should refer to this very nice explanation (which is the homework assignment for another CS class.) Don't look too much at the specifics of the code, as I'll describe what you will implement below. Use this document to understand the process:

http://www.cs.arizona.edu/people/mercer/Projects/DNASplicing10.pdf

You should create a new class named DNASplice, in a file named DNASplice.java. Your class should be part of the hw2 package. Your class should support the following static method:

```
public static int cutAndSplice(DoublyLinkedList12<Character> enzyme,
                               int index,
                               DoublyLinkedList12<Character> strand,
                               DoublyLinkedList12<Character> toBeSpliced)
```

This method takes as input a restriction enzyme as a DoublyLinkedList12 of Characters, with one entry for each character in the enzyme string as well as the index that represents the number of nucleotides to be left on the left in the restriction enzyme after the splice. It also takes the starting DNA strand and the DNA to be spliced in, also as DoublyLinkedList12of Characters. This method *modifies* the original DNA strand so that the toBeSpliced strand is spliced in at *every* occurrence of the restriction enzyme in the original DNA strand. If it does not find the restriction enzyme in the DNA strand, then the original strand remains unchanged. It returns the number of locations where the restriction enzyme was found in the DNA strand (i.e. the number of splices performed).

Then, in the main method of your DNASplice class you should implement a program that behaves as follows:

- Prompts the user for an index and restriction enzyme.
- Prompts the user for a DNA strand.

· Prompts the user for a strand of DNA to be spliced in.
· Calls cutAndSplice to splice the DNA
· Prints out the new DNA strand as well as the number of times that it found the restriction enzyme.

The exact formatting of the I/O is up to you, as long as this is the behavior of your program.

Submission

1. DoublyLinkedList12Tester.java
2. DoublyLinkedList12.java
3. README.txt
4. Problem4.txt

Via vocareum, make sure you pass the submission script. It checks the names of your files, compiles it and runs simple JUnit tests.

Grading

How your assignment will be graded:

Coding Style

- Does your class and tester properly generate javadoc documentation
- consistent indentation (Please use SPACES, not tabs)
- meaningful variable names
- helper methods used when needed to reduce code duplication

Correctness

- Does your code compile
- Does it pass all of your own unit tests
- Does it pass all of our unit tests
- Does your code have any errors (e.g. generates exceptions when it isn't supposed to)

Unit test coverage

- Have you created **at least** 10 more meaningful unit tests? Remember EACH method must be tested. 10 is a MINIMUM requirement.
- Does it detect errors in a reasonably buggy implementation of DoublyLinkedList12?
- Does your unit testing approach for listIterator() appear to be sufficient?
- You might want to include testing what happens before the head and after the tail, as well as more specific tests for each of the methods.