

Project 2: Doubly-Linked Lists

Due: Thursday, October 8, 2015, at 23:59:59 (without generics)

Due: Saturday, October 10, 2015, at 23:59:59 with generics.

Points: 230 + 25 for extra credit.

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 is an extra credit problem at the end.

Details

Part 1a - 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 your `DoublyLinkedList12` classes are supposed to do.

Tester for DoublyLinkedList12

Download the supplied file LinkedListTester.java. This is a starter file, 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.

Part 1b

In Part 1b, you need to change your tester file so that it works with the *generic* version of DoublyLinkedList12 from Part 2a. A lot (if not all) of your test cases can be reused. Your test routines should not "peer" into the internal workings (e.g., instance variables) of the DoublyLinkedList12 class you are testing.

Important: You are allowed to add more tests to your tester file.

Note: your test methods should not return anything, nor should they print out anything. Instead, each of your test methods should include at least one call to an "assert" static method in the Assert class to verify that a specific condition is true.

Part 2a – DoublyLinkedList12 implementation (without generics)

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.

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? We will do our best to find tiny bugs in your code that can result in data structure inconsistencies.

Part 2b – `DoublyLinkedList12` implementation (with generics)

"Generify" your `DoublyLinkedList12` from Part 1a so that it accepts elements of type `E`, not *object*.

Change your `DoublyLinkedList12` to:

```
public class DoublyLinkedList12<E> extends AbstractList<E> {...  
and MyListIterator to  
protected class MyListIterator implements ListIterator<E> {..
```

The rest should be straightforward. Make Eclipse happy and get rid of all warnings.

Important: Your both versions of `DoublyLinkedList12` should be the same, except for the generics part. You can't keep on fixing errors after the Thursday deadline (except extra credit problems).

Part 3a – 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).

Part 3b – Iterators for DoublyLinkedList12

Just like in part 2b you need to convert your code into a generic one.

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: <code>double x = 5;</code>
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 <code>sort()</code> method
7.	T	F	You are not able to change variable values from a subclass
8.	T	F	method declarations <code>void A(double x, integer k){};</code> and <code>void A(integer k, double x) {};</code> 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	<code>("Give me Liberty".split(" ").length)</code> evaluates to 3
12.	T	F	<code>String.equals</code> and <code>==</code> will always return the same result
13.	T	F	If the following statements are the only two statements in a method, <code>String X = "thing one";</code> and <code>String Y="thing one";</code> then <code>X.equals(Y)</code> evaluates to <code>true</code> , but <code>X == Y</code> evaluates to <code>false</code> within that method.
14.	T	F	A class declared as <code>final</code> cannot be inherited via the <code>extends</code> keyword.
15.	T	F	consider the statement: <code>String S = "Out of Gas";</code> then the statement: <code>S[7] = 'g';</code> will change "Gas" to "gas".
16.	T	F	boolean primitive variables can only be assigned values: <code>true</code> , <code>false</code> , or <code>null</code> .
17.	T	F	You can index into an array with a variable of type <code>double</code> as long as the there are no digits past the decimal point.
18.	T	F	A subclass inherits all of the <i>public</i> , <i>private</i> and <i>protected</i> 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	<code>int i = Math.sqrt(4.0);</code> is a valid statement.
23.	T	F	the <code>protected</code> keyword can only be applied to instance variables.
24.	T	F	Consider the statement: <code>throw new IllegalArgumentException();</code> This always causes the program to immediately exit. 
25.	T	F	Suppose you have the following declaration: <code>int xyz = 4;</code> Then, in the body of a <code>switch</code> statement block <code>case xyz: System.out.println("4"); break;</code> is legal.

Part 5 – Extra challenge problems

You have a few problems to choose from

1. In your DoublyLinkedList12.java class add two more methods. Do not forget about test cases for those.
 - a. Write a function to *concatenate* two linked lists. Given lists List1 = (3, 5, 7) and List2 = (2, 4), after return from concatenate (List1, List2) the List1 should be changed to be List1 = (3, 5, 7, 2, 4). Your function should **not** change List2 and should not directly link nodes from List1 to List2 (i.e. the nodes inserted into List1 should be copies of the nodes from List2.)

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

- b. Write a function to insert a number into a sorted linked list. Assume the list is sorted from smallest to largest value. After insertion, the list should still be sorted. Given the list List1 = (3, 5, 7, 9) and the value 6, on return List1 be the list (3, 5, 6, 7, 9).

Method signature: *public void addSorted(E element);*

2. 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 DoublyLinkedList12 of 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.

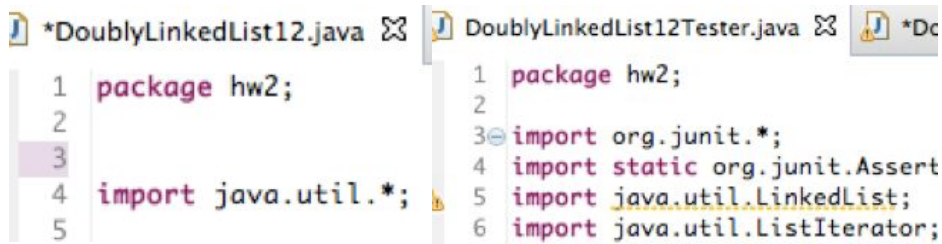
The exact formatting of the I/O is up to you, as long as this is the behavior of your program.

Submission

Thursday (October 8): The part without generics (1a, 2a, 3a)

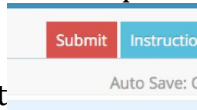
- DoublyLinkedList12Tester.java
- DoublyLinkedList12.java
- README.txt

- 1) You must create **hw2** package and add both .java files there
- 2) When you submit, both files should have these lines on the top:



```
1 package hw2;
2
3
4 import java.util.*;
5
1 package hw2;
2
3 import org.junit.*;
4 import static org.junit.Assert
5 import java.util.LinkedList;
6 import java.util.ListIterator;
```

3. Login to Vocareum.
4. Choose HW2.Part2
5. Click My Work and Upload 3 files (no folders)



6. Click Submit

7. Then you will see a script running, it might take a few seconds.
8. When it is done, you should be able to see following:

Executing 'Submission' script...

Wed Oct 7 19:21:18 PDT 2015

testGet ... [Pass]
testEmpty ... [Pass]
testIterator ... [Pass]
testListSize ... [Pass]

stdout:

Passed [File exists (DoublyLinkedList12.java)]
Passed [File exists (DoublyLinkedList12Tester.java)]
Passed [File exists (README.txt)]
Passed: [javac -d . -cp /home/prod/vlibs/java/junit-4.12.jar DoublyLinkedList12Tester.java DoublyLinkedList12.java]
Passed: [javac -d . -cp /home/prod/vlibs/java/junit-4.12.jar DoublyLinkedList12.java
/home/prod/libs/vc_2_0_ucsd_0902001/.vocasn/vc_2_0_ucsd_0902001_6/DoublyLinkedList12Tester.java]

JUnit version 4.12

.Check get(0)
.Check if Empty
.Check iterator
.Check List sizes

Time: 0.007

OK (4 tests)

Passed: [java -cp ./home/prod/vlibs/java/junit-4.12.jar:/home/prod/vlibs/java/hamcrest-core-1.3.jar
org.junit.runner.JUnitCore hw2.DoublyLinkedList12Tester]



We are checking if your files are correctly named and code compiles. And we also run a few basic tests against your `DoublyLinkedList12.java` implementation.

However, if your code does not pass all the tests, it should not prevent you from submitting your homework

Saturday (October 10): The part with generics (1b, 2b, 3b, 4, 5)

- `DoublyLinkedList12Tester.java`
- `DoublyLinkedList12.java` (with optional methods)
- `README.txt`
- `Problem4.txt`
- `DNASplice.java` (optional)

If you implemented extra credit problems, please specify it in your `README.txt` file.

Same steps as before. Once I'm done testing my script I will open the submission.

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? 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.