DLList Iterator Completion
Due: Day of lab at 11:59PM

1 DLList Iterator Completion Specification

1.1 Available Resources

- Lecture slides
- Other sections of the provided code
- me
- The textbook
- **DO NOT** refer to or use online implementations

1.2 Lab Instructions

- This is an *individual* lab.
- Make sure to read through all of the specifications so your submission is complete.
- Follow all the submission steps in the Setup document by the lab deadline.

1.3 Lab Link

The skeleton code for the lab is available at `https://classroom.github.com/a/irLIkie2`.

1.4 Implementation

This lab introduces a `DLList` class to implement a list. The list can only be modified using a `ListIterator` – all of the class methods of the List are gone.

Note the class header: `DLList<E> implements Iterable<E>`. This promises that DLList has an `iterator()` function. In addition, an inner class `KWListIter<F> implements ListIterator<E>` is defined. `ListIterator` is an extension of the `Iterator` interface, with extra operations available. The tests class has many examples that use this iterator.

I have provided the class code in the **edu.wit.cs.comp2000** package. You will implement the remaining method – `set`. Descriptions of all of the method's expected behavior are included in the lecture slides (on the ListIterator page). In addition, Javadoc comments for them are visible if you hover over the method names in Eclipse, as they implement the `ListIterator` interface.

For your implementation, consider what effect the method should have on the data structure. Don't forget that `set` changes a recently returned value, so it should reset `lastItemReturned`. It should also throw an exception when appropriate.

1.5 Testing

The other goals of this lab are to practice with more unit testing and debugging. We will use this framework this semester to verify that our data structure implementations work the way we expect them to.

In addition to the `DLList` code, JUnit tests are provided in the **edu.wit.cs.comp2000.tests** package. You can run these tests to see if the `DLList` implementation is performing correctly. The tests that I have provided check that most of the iterator operations are behaving as expected. They also provide examples of how the JUnit methods **assertTrue** and **assertEquals** work and examples of how expected exceptions can be detected.

For the two unimplemented test methods, implement tests that check if that method works with the iterator implementation. You can follow the steps that other test methods take for each test – create a list and

iterator, modify the list, and then assert the `ListIterator` behaves the way you expect. Make sure that you actually call the `ListIterator` method that you are testing. (You should also delete the `Test not implemented` placeholder lines in the tests you implement.)

Note that one of the iterator methods has a bug, so correctly written tests should fail until you complete the **Debugging** section.

1.6 Debugging

One of the iterator methods has an error in it! It works in some situations but not others. Using the JUnit tests and your knowledge of the operation, fix that method in `DLList` so that it works correctly in every case.

Think about which cases the method currently covers, and which cases it's missing. Change the code so that it covers all of the cases correctly. Once you do, the unit tests should correctly pass for it.

1.7 JUnit Assertion Syntax

For each of the JUnit assert* methods, the first argument is the string that prints out if the test fails. Use any of these four assert methods in your tests to confirm the functionality of the `SLList` methods:

- assertTrue
- assertEquals
- assertNotEquals
- assertNull

1.8 Considerations For Each Method

Be considerate in testing your ADT. Your goal is to test all of the possible cases for each `DLList` method. Each test method may have several assert method calls depending on how many possibilities you test for. In future labs, you will be writing more complete test cases like these to test your own code.

1.8.1 Common Edge Cases

How might our list be structured when we run an operation? Consider all the different possibilities for each operation. For each case that applies, we will want to write some code that sets up a list, followed by at least one assert statement.

- Normal results (no bad inputs, no special cases in the operation)
- Bad index
- Adding/removing data from the front or end
- Expecting an exception

2 Double Check:

- Have you implemented the `set` ListIterator method?
- Have you written two JUnit tests?
- Have you debugged the DLList code so all the tests pass?
- Have you committed/pushed your code from the two files?

3 Grading

Each of the 4 **TODO** sections is worth $\frac{1}{4}$ of the lab grade.

Grades and any comments for the lab will be posted to your project on github. Grades will also be posted to Brightspace, eventually.