**Wentworth Institute of Technology**

**Department of Computer Science and Networking.**

**Comp 2000 Data Structures Fall 2019**

**Lab #6 List with Doubly Linked Chain** **and its application**

1. Create a class DoublyLinkedList <T> which implements ListInterface<T> and

Iterable <T>.

1. The implementation uses a doubly linked chain: the nodes have references to both next node and the previous node.

head

tail

You will see that the search and adding to the end are easier than with one-directional chain; adding in the middle and removal needs resetting both links.

1. For implementing the method remove (T entry),

add (int givenPosition, T newEntry) take advantage of the doubly linked structure. It means: traverse the chain with one node, and get previous and next nodes directly. Handle the cases of adding to tail, removing tail or head.

1. When creating the iterator, implement the methods remove.
2. In addition to the interface methods, create a method reverse which reverses the list, swapping next and prev links and swapping head and tail. Do it in place (without creating an array).
3. Create a class Line with private fields start and end of the type Point2d.

Define constructors

Line (double x1, double y1, double x2, double y) and

Line (Point2d start, Point2d end)

and the methods needed for implementation of the Polygon. You will definitely need equals() and toString() in both classes.

1. For testing: create a class Polygon with one private field DoublyLinkedList <Line> lines. Define a constructor Polygon( String filename).

You will initialize this object from the attached file *polygon.dat.*

Define a method display.

Also, define a method reverse for the polygon. Pay attention: it not only reverses the chain, but swaps the lines ends. For this purpose, ListIterator is useful. When creating it, you may opt to not implement the methods set and remove, but implement the method add.

In the main method, fill a polygon from the file polygon.txt, display it, then reverse the polygon and display again.

Make sure the attached file PolygonTesting.java runs and produces correct result.

1. (bonus 5 pts): define methods allowing to fill gaps in the polygon. In the attached file, the gaps are between 4th and 5th lines, and between tail and head. You may use ListIterator, in which the method add is implemented.
2. If you want more bonus points, ask me specifically. The problem is: having two polygons with one common side, generate the union: a polygon covering common area.