Lab 1 – due 9/10 at 11pm

Review for generics, linked list, and stack – do not hardcode the data into your driver

1. Use the linked list class in Example13-0. Change the LinkedList class to use an E (element) instead of Item as the generic data type. Add a delete() method. Add an atEndOfList() method to stop using getCurData() to check for end of list. Add a GoToHead() method. Add a printListDataType() to print the type of data in the list.

   Write a driver (main method – make it a separate class – don't put it in the Linked List class) to read an input file (nums.txt) and put the numbers into a linked list for doubles. Create another linked list object and read an input file(strings.txt) and put the strings into a linked list for strings. After reading in all the data, use the printListDataType() method to print the data type of the data in the list.

   Do the following until the linked list is empty: make the driver (main) print out the list on one line, remove the head, and print again. Do this until the list is empty.

   My solution is:

```
$ java Driver
The data type for the data in this linked list is Double

List is 15.9 3.3 2.0 6.0 1.7 1.5 8.1 7.2 7.9 3.7 3.2 15.4 15.5
List is 3.3 2.0 6.0 1.7 1.5 8.1 7.2 7.9 3.7 3.2 15.4 15.5
List is 2.0 6.0 1.7 1.5 8.1 7.2 7.9 3.7 3.2 15.4 15.5
List is 6.0 1.7 1.5 8.1 7.2 7.9 3.7 3.2 15.4 15.5
List is 1.7 1.5 8.1 7.2 7.9 3.7 3.2 15.4 15.5
List is 1.5 8.1 7.2 7.9 3.7 3.2 15.4 15.5
List is 8.1 7.2 7.9 3.7 3.2 15.4 15.5
List is 7.2 7.9 3.7 3.2 15.4 15.5
List is 7.9 3.7 3.2 15.4 15.5
List is 3.7 3.2 15.4 15.5
List is 3.2 15.4 15.5
List is 15.4 15.5
List is 15.5

The data type for the data in this linked list is String

List is why did she give us this assignment
List is did she give us this assignment
List is she give us this assignment
List is give us this assignment
List is us this assignment
List is this assignment
List is assignment
```

2. Create a generic stack class with a linked list as its underlying data structure. You will instantiate the stack class for strings in your driver. You must write this on your own - do not get code from the internet or the book (but you can use the book code or internet code to help you write your code). You will need to use the proper method names – push, pop. You will need other methods as well.

Write a driver (main) that will push the string onto the stack if it is not a "$'. If it is a "$", it will pop the head of the list and print it. Use the input file (hamlet.txt). All the "heavy lifting" should be done by the Stack class. Your main method should read the data, put it onto the stack or take it off. And then print.

Once you have processed the input and printed the results of the $'s, print whatever is left on the stack.

The answer should be:

```
be not or question the is
Left on stack: that be to to
```

Either add to the driver above or create a second driver that will instantiate a stack object for postfix math equation for integers. Use the input file (equation.txt).

The answer should be:

```
The result of the postfix equation is 22
```

Submission: submit your code in gradescope and screen shots showing each run with inputs and outputs.

Rubric: Each problem is worth 50 points:

- 5 points – style – used comments and proper indention
- 10 points – created the proper driver(s) to "drive" your data structures to get the correct answer
- 10 points – you created the proper data structures as specified in the write up (generic stack with list as underlying data attributes, generic queue with array as underlying data attributes)
- 5 points – use the proper method names for the classes (insert for linked list; push, pop for stack)
- 10 points – prints the correct output (answers)
- 10 points – reads in the input as specified above (as files)
- You will get a 0 if you:
    a) Do not submit the source code
    b) Did not write the stack class on your own (for problem 2)
    c) Did not submit the screenshot of the run(s)