Institute of Technology, University of Washington Tacoma
TCSS 342 Data Structures, Winter 2019
Assignment 2
A singly linked implementation of an output-restricted double-ended queue
Value: 5% of the course grade
Due: Thursday, 31 January 2019, 23:59:59

Purpose:
This project has a number of purposes. It is intended to provide practice with the following:
- extension of an interface to provide new functionality
- implementation of an extension to a linked based collection
- development of unit tests
- use queues to solve a problem

This project also serves to exercise your ability to use Eclipse and plugin tools to support the development of high quality code that adheres to conventions.

Program Description:

Part 1 (50%):
An output-restricted double-ended queue supports insertions from both ends, but accesses and deletions only from the front. You will develop a singly linked implementation of an output-restricted double-ended queue.

You will extend the provided **QueueADT** interface with a new interface called **OutputRestrictedDequeADT** which will declare one new operation:
```
void enqueueAtFront(E theElement);
```

You will develop a new class called **LinkedOutputRestrictedDeque.** This class will extend the provided **LinkedQueue** class and will implement the **OutputRestrictedDequeADT** interface.

If you make ANY changes to the **LinkedQueue** class then you MUST list these changes (and your reasons for these changes) in your executive summary.

Provide JUnit tests for all public **OutputRestrictedDeque** operations (including those inherited from **LinkedQueue**). Your tests should provide complete code coverage (including the `toString()` method). Think carefully about border cases and exceptional cases as you develop your code and your tests. Your unit tests should not produce any console output. Place your unit tests in the 'tests' package.

Part 2 (50%):
Write a static method called ‘radixSort’ to sort a queue of integers and a program called RadixSortDemo.java to use that method.

Your program MUST have the following features:

- Your program must include a static method which implements radix sort (described in class and shown in the Queue ADT lecture slides). The ONLY data structure you may use internally in this implementation is your **LinkedOutputRestrictedDeque.** (You may also use an array to hold the 10 digit queues). Your implementation of this algorithm must be a static method called ‘radixSort’ which accepts a queue (of unsorted integers) as a parameter and returns a queue of sorted integers.
- Your program must have the ability to read a text file containing unsorted integers entered one per line. I have provided one such file (input100.txt) in the project starter code. It might be useful for you to create other (smaller? larger?) input files to use as you develop and test your code.
- Your program must use your implementation of radix sort to sort the integers read from the file into sorted order from smallest to largest.
- Your program must write the sorted integers to an output text file called “output.txt”, one integer per line. Do not write the output back to the input file, use a separate output file (so that you do not change the input file.)
- Place your radix sort program in the 'applications' package.

You have freedom to make many design choices according to your own preferences:

- Your program may be a console application or a GUI application
- You application must provide the user a way to enter an input filename.
- If you would like to write your program to accept a command line parameter for the input filename, that is fine (and gets cool points!) Lots of computer science happens at the command line.

You do NOT need to write unit tests for your program or for your implementation of radix sort; however, the code should adhere to course coding conventions and should be fully documented with Javadoc comments.

Part 3 (5% extra credit):
For extra credit, provide a simple GUI to demonstrate the operations of your **LinkedOutputRestrictedDeque**. Place your demo code in the 'applications' package.

At minimum your GUI should allow a user to call enqueue(), enqueueAtFront(), and dequeue(). Your GUI should display the state of the queue after each operation (perhaps by using the toString() method).

I suggest that you start by using the StackDemo and StackDemoGUI classes provided in the StackDemo project provided on the assignment 2 page on Canvas. Rename the classes appropriately and make appropriate modifications to the code to provide a useful demo of the queue operations. You do NOT have to use the StackDemo and StackDemoGUI classes. If you wish, you can develop your own GUI for this demo.

You do not need to write unit tests for your demo. Your demo will receive full credit if it meets the minimum requirements stated above, meets course coding conventions, and if the user interaction is intuitive.

Getting Started:

Create your Eclipse project by downloading the hw2-project.zip file from the Assignment 2 page on Canvas, importing it into your workspace, and using “Refactor” to change "username” in the project name to your UWNetID. Incorrectly-named projects *will lose points.* Be sure to rename your project BEFORE sharing it to your SVN repository.

NOTE:

The Eclipse plugin tools generate a few warnings about the provided starter code. You are not required to silence these warnings. You can safely ignore the warnings in the starter code.

Documentation within the code:

Include complete javadoc comments in your **OutputRestrictedDequeADT** interface, in your **LinkedOutputRestrictedDeque** class, and in your radix sort program. That is, include a javadoc comment for each interface, class, field, constant, and method including those declared private. Try to emulate the style and quality of the javadoc comments in the Java API.

Submission and Grading:

When you have checked in the revision of your code you wish to submit, make a note of its Subversion revision number. To get the revision number, *perform an update* on the top level of your project; the revision number will then be displayed next to the project name. Include your project's revision number in your executive summary.

Part of your program's score will come from its "external correctness." For part 'A' this is measured by running MY unit tests on your code. For part 'B' this is measured by using your program to attempt to sort several files of unsorted integers.

Another part of your program's score will come from its "internal correctness." Internal correctness includes meaningful and systematically assigned identifier names, proper encapsulation, avoidance of redundancy, good choices of data representation, the use of javadoc on all class members, the use of non-javadoc implementation comments on particularly complex code sections, the inclusion of file header comments, and adherence to coding conventions as identified by the output of the Eclipse plugin tools.

The assignment will be graded using the following breakdown : 10% executive summary, 55% external correctness, 20% internal correctness, and 15% for your unit tests (which I might run on an incorrect implementation to see if your tests identify the issues).