Homework 4: Dark Room
Due: Wed. 2/6/2016 at 11:59pm

Overview:

In this assignment you will implement data structures that provide an implementation for three abstract data types: A queue using a circular array, a singly linked list and a stack. The stack will be implemented using an adaptive design pattern mechanism. In addition, you'll use these data structures in a method that implements a search in a "dark room".

Provided Files

DoubleEndedSinglyLLInterface.java
Stack_QueueInterface.java
DarkRoomInterface.java
Location.java
DarkRoom.java (starter code)
smallRoom.txt
zip containing various room examples

MyStack class (+tester)	(20 points)
MyQueue class (+tester)	(25 points)
DoubleEndedSinglyLL(+tester)	(25 points)
Test cases (5 in total)	(50 points)
Style (comments etc)	(20 points)
Compiles	(10 points)
Other (hw4.txt, correct choice of mapping etc)	(10 points)
Total:	**(150 points)**

Room Representation:

You will be given a text file with a picture of a room with obstacles. Your starting position is marked with "S", the door's position is marked with "D". A room will have varying dimensions and number of obstacles, but will always be rectangular and have a start state.

Example: 6x5 Room
The numbers on the top indicate how many rows (6) and how many columns (5) are in the room respectively.

' * ' represents a wall
'@' represents an obstacle

```
6 5
*****
*S  *
* @ *
* D@*
*   *
*****
```

Location Representation:

A location in a room is determined by a row and column index based on the chars in the txt file. For the example room above the location of the "S" is (1,1). Indexing starts with a 0.
An object of type `Location` represents your location. (Check `Location.java`)

Overview of Algorithm to find "D"

```
read the file and initialize the array that represents the room
find the location of S within that room
//you will store locations in a queue for one implementation
//and in a stack for the other
store this location
repeat until either "D" is found or all locations in storage
have been explored:
     get a location L from storage
     for all valid locations J next L:
          check if J is the door 'D'
               if so terminate
     add all valid locations J next to L to storage
```

Note:

- A **valid location** next to L is defined as a location that is LEFT, UP, RIGHT or DOWN from L and is **not a wall or an obstacle**.

- **YOU MUST** check for valid locations in this order

  LEFT -> UP -> RIGHT -> DOWN

  A failure to do so will result in output that is not consistent with our tester and you will be penalized.

Grading Notes:

- Include the following at the top of **EVERY REQUIRED FILE:**

```
NAME: <your name>
ID: <your student ID>
LOGIN: <your class login>
```

- Commenting:
  - ALL methods must be commented with appropriate headers and inline comments where necessary.
- Every method should be tested. The number of tests is up to you.
- You must accept one input argument and that will be a filename pointing to the room
- You must use the given print methods, **AND THEY MUST NOT BE MODIFIED**
- For the given example, your output must exactly match the given output solution
- Your code should be in a package named hw4
- **ANY** exceptions that may occur should be implemented the same way Java's Stack or Java's Queue would handle the exception. If you're unsure, ask on Piazza

Development Guide:

1. Create `DoubleEndedLL` class that implements `DoubleEndedLLInterface`.
   a. Generic class
   b. Implementing a singly-linked list that includes head and tail pointers
   c. Includes an inner class Node.
2. Create `DoubleEndedLLTester` to test your linked list class methods. All methods must be tested.
3. **`Note:`** `Stack_QueueInterface` shared methods between a stack and a queue, that's why the names of the methods are not stack/queue specific.

Also, when you implement methods for a stack and a queue, make sure to choose an appropriate exception if needed.

4. Create `MyStack` class that implements `Stack_QueueInterface`
    a. Generic class
    b. You **must** use adapter design pattern implementation in order to receive a full credit for this part
    c. Use a `DoubleEndedLL` class that you created and match the *appropriate* methods to your stack class methods.
    d. In your hw4.txt file indicate what methods were chosen to perform stack operations and why.
5. Create `MyStackTester.java` to test your class methods.
6. Create `MyQueue` class that implemcents `Stack_QueueInterface`
    a. Generic class
    b. Must use circular array for implementation to receive full credit (no arrayList)
    c. If there is no more room to add, you should enlarge (double) your array.
7. Create `MyQueueTester.java` to test all of your class methods.
8. In `Location.java` comment every line of code (so you understand it better)
9. In `DarkRoom.java`:
    a. comment every line of code in `readFromFile()`
    b. fill in helper methods
        i. test as you develop
    c. complete `escapeDarkRoom()` using the algorithm that was detailed in *Overview of Algorithm to find "D".*
        i. **Note**: In your `escapeDarkRoom` method, your code should work correctly without knowing whether a Stack or Queue will be used. You might want to declare a variable of type `Stack_QueueInterface`. Then, at run time (in main), the correct class, `MyQueue` or `MyStack` can be instantiated.
        ii. **Hint**: Here is how your code could look like

```
Stack_QueueInterface <Location> storage ;
if ("Stack".equals(choice) ) storage = new MyStack<Location>();
else storage = new MyQueue<Location>();
```

10. In `Escape.java`:
    a. calls DarkRoom.java methods to read the file and find the door in two ways, once using a "Stack" and once using a "Queue".
    b. Hint: first try to make it work without a command line argument. When everything works, change Escape.java to accept one command-line argument and submit this file.

Expected Output:

`Escape.java` that will have the main function. It will accept **the name of the text file containing the room data as a command line argument** and nothing else.
Do not use local paths in your main method.

The output is shown below: One uses a stack to keep locations, and the second uses a queue. **Remember:** you must follow the strict order when explore your location and add them to the storage: **LEFT, UP, RIGHT, and DOWN**. Your output must be consistent with our own.

```
Marinas-MacBook-Air-2:src marinalanglois$ java hw4.Escape smallRoom.txt
Goal found (with stack): It took 2 explored positions
There is (are) 1 position(s) left to explore in stack
*****
*S  *
*.@ *
*.D@*
*   *
*****
Goal found (with queue): It took 4 explored positions
There is (are) 1 position(s) left to explore in queue
*****
*S..*
*.@ *
*.D@*
*   *
*****
```

Turning in Your Assignment:

Required files:

Location.java (with your comments)

MyStack.java
MyStackTester.java
MyQueue.java
MyQueueTester.java
DoubleEndedLL.java
DoubleEndedLLTester.java
DarkRoom.java
Escape.java
hw4.txt

Remember:

No folders!
All relevant files must be in package `hw4`
Make sure you hit **submit** button to submit your code
Do not forget to check submission report for correctness