Project 4. Deadline: October 24th. 11:59pm. (150 points total)

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. In addition, you'll use these data structures in a method that implements a search in a "dark room". Also, along the way you will get more experience with implementing Java interfaces, writing JUnit test cases, and using the Adapter design pattern.

In EACH AND EVERY FILE that you turn in, we need the following in comments at the top of each file.

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

The idea of the project:

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". This is a small example of the room 5x5: 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@*
*   *
*****
```

In order to keep track of your location and store possible future locations you will need a data structure. In this assignment you will use a stack and a queue, and compare their solutions.

Your Location in a room: Representation and Storage.

A location in a room is determined by a row and column index. For the 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`)

The algorithm to find your way out:

1) Read the file and initialize your array.
2) Find location of 'S' (Start location)
3) Add the starting location onto a storage (it will be either a stack, or a queue).
4) Repeat the following steps until either the door 'D' is found or no more locations to investigate left in storage:

- Get a current location from the storage.
- For each **valid** location that is to the LEFT, ABOVE, RIGHT, and to the BELOW of the current location:
    - if valid location is the door 'D', terminate.
    - else, add the new location onto the storage.

The following files are provided for you and can be found on the HW4 page:

- DoubleEndedSinglyLLInterface.java
- Stack_QueueInterface.java
- DarkRoomInterface.java
- Location.java
- DarkRoom.java (starter code)
- smallDoor.txt

Expected Output:

You will create a class `Escape.java` that will have the main function. It will accept the name of the text file as a command line argument and nothing else.

The output is shown below: One uses a stack to keep locations, and the second uses a queue. You **MUST** follow the strict order when explore your location and add them to the storage: LEFT, ABOVE, RIGHT, and to the BELOW. This way you will get the same result as everyone else (including me!).

```
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@*
*   *
*****
```

Implementation Steps.

1. This step must be completed for all classes: Every method must be commented.
2. Create `DoubleEndedLL` class that implements `DoubleEndedLLInterface`.
    a. generic class implementing a singly-linked list that includes head and tail pointers. It should include an inner class Node.
3. Create `DoubleEndedLLTester` to test your linked list class methods.
4. Create `MyStack` class that implements `Stack_QueueInterface`.
    a. generic class, uses adapter design pattern for implementation, using the linked list that you created.
    b. Pick the mapping between a stack and a list carefully.
    c. In your **hw4.txt** file indicate what methods were chosen to perform stack operations and why.
    d. You must use adapter design pattern implementation in order to receive a full credit for this part.
5. Create `MyStackTester.java` to test your class methods.
6. Create `MyQueue` class that implements `Stack_QueueInterface`.
    a. generic class, uses a circular **array** implementation. Do not use arrayLists.
    b. If there is no more room to add, you should enlarge (double) your array.
7. Create `MyQueueTester.java` to test your class methods.
8. Open `Location.java` and comment EVERY LINE of code in order to understand what it does and get credit.
9. Open `DarkRoom.java` and comment EVERY LINE of code in *readFromFile()* method in order to understand how the parsing works and get credit.
    a. Start filling in the helper methods and testing them.
    b. Finally complete `escapeDarkRoom()` method, using the algorithm above:
    c. **Note**: In your `escapeDarkRoom` method, your code should work correctly without knowing whether a Stack or Queue will be used. You

might what to declare a variable of type Stack_QueueInterface. Then, at run time, the correct class, MyQueue or MyStack can be instantiated.

d. 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. Your last class to implement is `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. First try to make it work without a command line argument.
    c. When everything works, change `Escape.java` to accept one command-line argument.

Keep in mind (to get full credit):

1. You must accept one input argument and that will be a filename pointing to the room.
2. You must write your solution to the console (System.out.print/println, etc).
3. For the given example, your output must exactly match the given output solution
    a. I will provide you with a few more examples.
4. Your code should be in a package named hw4.
5. Your code should not be submitted in a directory
6. **The following files must be submitted**:
    a. Location.java (with your comments)
    b. MyStack.java
    c. MyStackTester.java
    d. MyQueue.java
    e. MyQueueTester.java
    f. DoubleEndedLL.java
    g. DoubleEndedLLTester.java
    h. DarkRoom.java
    i. Escape.java
    j. hw4..txt

Turn in via Vocareum

1) No folders
2) Do not forget package `hw4`; on the top of each file
3) Do not forget to hit SUBMIT button
4) Do not forget to check submission report to make sure everything looks good.