Lab 10: Stacks and Queues

Implement a Stack and a Queue Data Structure

Specifications

- Read all of these instructions carefully. Name things exactly as described.
- Do all your work in a public repository (matching the example provided by your instructor) called `data-structures-and-algorithms`, with a well-formatted, detailed top-level README.md
- Create a branch in your repository called `stack_and_queue`
- On your branch, create…
  - *C#*: Create a new console app named `StacksAndQueues`.
  - *JavaScript*: a folder named `stacksAndQueues` which contains a file called `stacks-and-queues.js`
  - *Python*: a folder named `stacks_and_queues` which contains a file called `stacks_and_queues.py`
  - *Java*: a package named `stacksandqueues` which contains files called `Stack.java`, `Queue.java`, and `Node.java`.

- Include any language-specific configuration files required for this challenge to become an individual component, module, library, etc.
  - *NOTE: You can find an example of this configuration for your course in your class lecture repository.*

Features

- Create a `Node` class that has properties for the value stored in the Node, and a pointer to the next node.
- Create a `Stack` class that has a top property. It creates an empty Stack when instantiated.
  - This object should be aware of a default empty value assigned to `top` when the stack is created.
  - Define a method called `push` which takes any value as an argument and adds a new node with that value to the `top` of the stack with an O(1) Time performance.
  - Define a method called `pop` that does not take any argument, removes the node from the top of the stack, and returns the node's value.
  - Define a method called `peek` that does not take an argument and returns the value of the node located on top of the stack, without removing it from the stack.

- Create a `Queue` class that has a front property. It creates an empty Queue when instantiated.
    - This object should be aware of a default empty value assigned to `front` when the queue is created.
    - Define a method called `enqueue` which takes any value as an argument and adds a new node with that value to the back of the queue with an O(1) Time performance.
    - Define a method called `dequeue` that does not take any argument, removes the node from the front of the queue, and returns the node's value.
    - Define a method called `peek` that does not take an argument and returns the value of the node located in the front of the queue, without removing it from the queue.

- At no time should an exception or stack trace be shown to the end user. Catch and handle any such exceptions and return a printed value or operation which cleanly represents the state and either stops execution cleanly, or provides the user with clear direction and output.
- Be sure to follow your languages best practices for naming conventions.

You have access to the Node class and all the properties on the Linked List class.

Structure and Testing

Utilize the Single-responsibility principle: any methods you write should be clean, reusable, abstract component parts to the whole challenge. You will be given feedback and marked down if you attempt to define a large, complex algorithm in one function definition.

Write tests to prove the following functionality:

1. Can successfully push onto a stack
2. Can successfully push multiple values onto a stack
3. Can successfully pop off the stack
4. Can successfully empty a stack after multiple pops
5. Can successfully peek the next item on the stack
6. Can successfully instantiate an empty stack
7. Can successfully enqueue into a queue
8. Can successfully enqueue multiple values into a queue
9. Can successfully dequeue out of a queue the expected value
10. Can successfully peek into a queue, seeing the expected value
11. Can successfully empty a queue after multiple dequeues
12. Can successfully instantiate an empty queue

Ensure your tests are passing before you submit your solution.

Documentation: Your README.md

```
# Stacks and Queues
<!-- Short summary or background information -->

## Challenge
<!-- Description of the challenge -->

## Approach & Efficiency
<!-- What approach did you take? Why? What is the Big O space/time for this

## API
<!-- Description of each method publicly available to your Stack and Queue-
```

Submission Instructions

1. Create a pull request from your branch to your `master` branch
2. In your open pull request, leave as a comment a checklist of the specifications and tasks above, with the actual steps that you completed checked off
3. Submitting your completed work to Canvas:
    1. Copy the link to your open pull request and paste it into the corresponding Canvas assignment
    2. Leave a description of how long this assignment took you in the comments box
    3. Add any additional comments you like about your process or any difficulties you may have had with the assignment

4. Merge your branch into `master`, and delete your branch (don't worry, the PR link will still work)