LAB 6

Learning Outcomes

- Implement missing methods in a queue made with a Linked List structure
- Use the queue collection to track cyclical movement patterns
- Consider and compare the implications of using different implementations of queues
- Observe the results of a simulated snail race!

Pre-Lab

- Create a new Java project called Lab6
- Download the files: QueueADT.java, LinkedQueue.java, LinearNode.java, EmptyCollectionException.java, TestQueue.java, Snail.java, and SnailRace.java
- Save these downloaded files into the Lab6 src folder

Exercise 1 – Completing the Linked Queue class

1. Open LinkedQueue.java and TestQueue.java and examine the code in both classes.
2. Note that LinkedQueue is not complete; all its methods (other than the constructor) are left empty. Without adding any additional instance variables or methods, complete the six empty methods given the following instructions:
    - `enqueue()` – take in an element of the generic type. Create a new LinearNode with the given input element. Make all the proper connections including special ones if this is going to be the first element of the queue. Update count as well.
    - `dequeue()` – if the queue is empty, throw an EmptyCollectionException. If not, retrieve the element at the front of the queue and remove it and then return it. Remember to update the front pointer when this element is being removed and if this is the only item then update rear as well. Decrement count to indicate this removed item.
    - `first()` – if the queue is empty, throw an EmptyCollectionException. If not, retrieve the element at the front of the queue and return it without removing it.
    - `isEmpty()` – return true if there are no elements, and false otherwise.
    - `size()` – return the number of elements in the queue.
    - `toString()` – if the queue is empty, return "The queue is empty." Otherwise return a single-line string starting with "Queue: " and then containing all the elements of the queue. There must be a comma and space between elements but the last element must end with a period instead. i.e.

Queue: 1, 2, 3, 4, 5.
Hint: how can you determine whether a node is the last element?

3. Run the TestQueue file. This will test that the LinkedQueue methods were properly implemented. If any of the test failed, fix the corresponding methods in LinkedQueue until all the tests pass. You may add print lines in LinkedQueue to help with the debugging. Do not add print lines in TestQueue or alter this file in any way.

Exercise 2 – Completing the Snail class

Before you begin this exercise, read this overview to explain what you're working on. You'll be simulating a snail race using queues that contain the snails' movement patterns. These queues will need to cycle so that the pattern continues repeatedly until the race is over.

1. Open Snail.java and SnailRace.java and examine the code. Note that the methods in Snail.java are empty. This class is used for representing each of the snails in the race.
2. Complete the constructor:
   - initialize the snail's position to 0
   - initialize the snail's queue "movePattern" and add each of the step sizes from the int array to the snail's queue in the same order they come in the array
3. Complete the `move()` method:
   - dequeue the first move pattern number and store it in a variable
   - re-enqueue this movement to the back of the queue so it will cycle around
   - increment the snail's position by the move pattern value
   - don't let the snail go **past** the finish line; after the movement, check its position compared to the race length (this is a variable in the SnailRace class so examine how the variable was made to determine how you can access it from here) and just leave it at the finish line if was going to be past the line

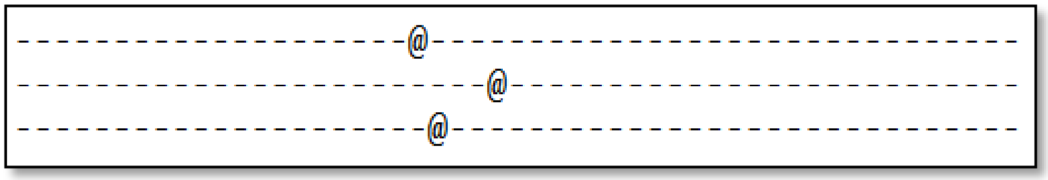
Questions: Consider the possibility of using an Array Queue or a Circular Array Queue instead of the Linked Queue being used in this simulation. Would the simulation's results be impacted by switching to a different Queue implementation? Which of these classes/methods would you have to modify if you were going to use one of the other queue implementations?

4. Complete the `getPosition()` method to simply return (get) the position
5. Complete the `display()` method:
   - create two int variables called "dashesBefore" and "dashesAfter" and assign them the values that represent the distance before and after the snail, i.e. when

the snail is at the start, dashesBefore is 0 and dashesAfter is 50. If the snail moves up 1 space, dashesBefore is 1 and dashesAfter is 49, etc.
- use the dashesBefore value to help in displaying that number of dashes on a single line (use System.out.print() rather than .println() to avoid new lines)
- display the snail's shell using the icon variable (@)
- use the dashesAfter value to display the dashes to the right of the snail icon (all on one line still)
- finally print out a newline character so the next snail's path will be on a different line than this one

6. When you have completed the Snail.java methods, go into SnailRace.java and run it. Watch the live race in the console (you may need to resize the console to clearly watch the race unfold)
7. Experiment with different movement patterns for the snails and you can even add more snails to the race. This is all done in the SnailRace constructor.

Screenshot of a simulated snail race about halfway through the race:



```
--------------------@------------------------------
------------------------@--------------------------
---------------------@-----------------------------
```

Submission

When you have completed the lab, navigate to the weekly module page on OWL and click the Lab link (where you found this document). Make sure you are in the page for the correct lab. Upload the files listed below and remember to hit Save and Submit. Check that your submission went through and look for an automatic OWL email to verify that it was submitted successfully.

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the lab.
- Do not ZIP or use any other form of compressed file for your files. Attach them individually.
- Submit the lab on time. Late submissions will receive a penalty.
- Forgetting to hit "Submit" is not a valid excuse for submitting late.
- Submitting the files in an incorrect submission page will receive a penalty.

- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular lab deadline will receive a penalty.

Files to submit

- LinkedQueue.java
- Snail.java
- SnailRace.java