Pair Programming: **NOT ALLOWED**
Due: **9:59PM on April 19th**

P08 Bottle Factory

Overview

In this assignment you will implement and use queues to simulate the working of a bottle factory. You are provided a driver class simulating the working of the bottle factory machine which takes in empty bottles and produces filled & capped utilizing four queues (production, supply, filling and capping). You will implement two variants of the queue data structure (linked and circular) and a queue iterator.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P08 pre-assignment quiz is accessible through Canvas before having access to this specification by **11:59PM on Sunday 04/16/2023**. Access to the pre-assignment quiz will be unavailable passing its deadline.
15 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should run additional tests of your own.
20 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.
10 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.

Learning Objectives

The goals of this assignment include:

- Get more practice on implementing a generic interface.
- Get practice and experience both implementing a data structure queue and using that queue to solve some other problems.

- Get practice using both linked and circular indexing array queues.
- Learn how to implement the Iterable interface and an iterator to iterate over a queue.
- Develop unit tests to verify the functionality of a simple data structure.

Assignment Requirements and Notes

(Please read carefully!)

- Pair programming is **NOT ALLOWED** for p08. You MUST complete and submit p08 individually.
- **The ONLY external libraries** you may use in your submitted files are:

  `java.util.Arrays (only the copyOf method), java.util.Iterator and any relevant exceptions.`

- Only the `BottleFactoryTester` class contains a **main** method.
- You MUST NOT add any additional fields either instance or static to your program, and any public methods either static or instance to your program, other than those defined in this write-up.
- You CAN define local variables that you may need to implement the methods defined in this program.
- You CAN define **private** methods to help implement the different public methods defined in this program, if needed.
- Any source code provided in this specification may be included verbatim in your program without attribution.
- Your assignment must conform to the CS300 Course Style Guide. Note that in this assignment you will be writing overriding methods. It might be a good idea to check the CS300 Course Style Guide about the @Override notation.
- If you need assistance, please check the list of our Resources.
- You MUST adhere to the Academic Conduct Expectations and Advice

1 Getting Started

- To get started, let's first create a new **Java17** project within Eclipse. You can name this project whatever you like, but "**p08 Bottle Factory**" is a good choice. As usual, make sure that you are using **Java 17**, don't add a module, and that you use the default package.
- Download the provided code and add the files to the src folder of your project. These files are also available on the P08 Assignment page on canvas.
  1. LinkedNode.java: Generic class defining the node of a singly linked-list
  2. Bottle.java: Defines the Bottle class and its fields.
  3. QueueADT.java: The Queue Interface.
  4. BottleFactoryTester.java: Template for the tester class.
  5. BottleFillerMachine.java: Driver class simulating the working of the bottle factory. Add this to the src folder **after you complete the assignment**.
- Create four more classes within the src folder. Their source files should be named as follows:
  1. LinkedBottleQueue.java, does not contain a main method.
  2. BottleQueueIterator.java, does not contain a main method.
  3. CircularBottleQueue.java, does not contain a main method.

2 Read through and test Bottle Class

- The Bottle class defines the properties of the bottles in the bottle factory. The bottle properties include:
  - `color`, a private instance field of type String, defining the color of the bottle.
  - **BOTTLE_ORDER**, a final private instance field of type integer indicating the bottle's order of production/creation in the factory.
- Additionally the bottle also has fields indicating its fill and seal status
- Each bottle is identified using its serial number calculated from its properties in the following format:

```
"SN<BOTTLE_ORDER><color>"
For example SN1Blue
```

- Now implement the *bottleTester* method in the BottleFactoryTester class. Your `bottleTester` method must ensure the correctness of the constructor of the Bottle class, and its `toString()` and `equals()` methods. You should make sure that you understand how the Bottle class works before moving on to other classes.

3 Create and Test the LinkedBottleQueue Class

- Let's implement the first variant of the queue data structure, the LinkedBottleQueue class. It implements the QueueADT interface using a linked list to store objects of type Bottle. Each node in the linked list is of type LinkedNode .
- Begin by defining the following data fields:
  - `front` and `back`, private instance fields of type $LinkedNode < Bottle >$ indicating the begin and end bottles in the linked list queue
  - `size`, a private instance field of type int, indicating the number of bottles in the queue
  - `capacity`, a private instance field of type int, defining the max number of bottles the queue can hold
- Now implement the constructor of LinkedBottleQueue , which takes the capacity of the queue as a parameter and throws an exception if it is invalid.
- Your constructor should initialize all the fields appropriately and throw an exception if the arguments are invalid. Before moving on to the next methods, make sure to test your constructor in BottleFactoryTester .

3.1 Implement the QueueADT interface

- Implement the following methods defined by the QueueADT interface: *dequeue, enqueue, peek, isEmpty, isFull, copy* and *size*. The LinkedBottleQueue class additionally requires these methods to throw exceptions when encountering invalid states like an empty or full queue.
- Note that you have to implement these methods using a linked list. Refer to the LinkedNode for the structure of a LinkedNode and use its methods to update the list when enqueuing or dequeuing.
- Before moving on, make sure to test these methods on different scenarios:
  1. Test all the methods on empty, full and partially filled queues.
  2. Perform a sequence of **enqueue** and **dequeue** operations and verify the order of elements dequeued or enqueued, and the queue's size

3. Test your copy method to make sure it returns a **deep copy** and not a shallow copy.

- Now, before moving on to the `toString` method, you might have noticed that while you have implemented the basic functionality of a queue, there is no way for the user to traverse the queue. We would like to use a **for-each loop** to traverse the queue from its back to its front. So let's now work on the BottleQueueIterator class. Then, we can make the LinkedBottleQueue class to implement the `Iterable` interface.

4 Create and Test the BottleQueueIterator Class

- The BottleQueueIterator class provides the ability to iterate over a queue of bottles that has implemented the QueueADT interface. It *implements* the Iterator interface and iterates over a deep copy of the bottle queue. You can start implementing the class by defining the following data fields:
  - `bottleQueue`, a private instance field of type $QueueADT < Bottle >$, defines the queue of bottles to be iterated over
- The constructor initializes the `bottleQueue` field to a **deep copy** of the queue provided in the parameter of type $QueueADT < Bottle >$.
- Now implement the `hasNext()` and `next()` methods to iterate over the `bottleQueue`. The BottleQueueIterator should work with any queue of bottles that has implemented the *QueueADT* interface. It should be able to iterate over a deep copy of the `bottleQueue` using the *isEmpty()* and *dequeue()* methods. This said, the constructor of the BottleQueueIterator should assign the `bottleQueue` instance field to a deep copy of the original input queue, so that the iterator does not make any changes to the state and contents of original queue.
- Now that we have a way to iterate over our queues, let's go back to the LinkedBottleQueue class and implement the *iterator()* method to return an instance of BottleQueueIterator .
- Let's now implement the `toString` method of the LinkedBottleQueue class. The method should return a string representation of the queue from the front to its back with the string representation of each Bottle in a separate line. We can now use the iterator to iterate over the queue and access each bottle.
- Before we move on to the second variant of queue, make sure to test your BottleQueueIterator in the BottleFactoryTester class. You would want to verify the correctness of your iterator by iterating over a queue of arbitrary size and verifying the contents returned by the iterator.

5 Create and Test the CircularBottleQueue Class

- Let's now work on the second variant of the queue data structure, CircularBottleQueue class. It implements the queue using a circular-indexing array to stores elements of type Bottle.
- Begin by defining the following data fields:
  - `bottles`, a private instance field of type Bottle[], is the array of bottles.
  - `front` & `back`, private instance fields of type int, indicting the earliest added bottle and recently added bottle respectively.
  - `size`, a private instance field of type int, indicating the number of bottles in the queue.
- We again initialize all the data fields in our constructor and make sure to test the constructor before moving on to implement the queue methods and the queue iterator.
- Now similar to LinkedBottleQueue class, you will implement the methods defined in QueueADT interface; the difference being CircularBottleQueue uses a circular-indexing array instead of a linked-list. Make sure to not have any processed bottles in the array i.e. all unused spaces in the array are null.
- The BottleQueueIterator works for both the queue data structures, you can now implement the *iterator* and *toString* methods similar to the previous version.
- Again before moving on to other methods, make sure you test these methods in the scenarios defined earlier for LinkedBottleQueue class for the queue methods and the `toString` method.

6 Bottle Factory Driver

- As the final step for this assignment, download and add the BottleFillerMachine.java file to your src folder. This is the driver code to simulate the working of a bottle filling machine. You are not going to edit or implement any components of the driver, but it would be a useful way to test your queue implementations.
- The BottleFillerMachine has the following data fields:
  - `fillingQueue` & `cappingQueue`, private instance fields of type CircularBottleQueue indicating the queues of bottles to be filled and capped respectively
  - `supplyLine` & `productionLine`, private instance fields of type LinkedBottleQueue indicating the queues of finished and empty bottles respectively
  - `remainingBottlesCount`, private instance field of type int, is a helper variable to track progress of the machine

- The BottleFillerMachine uses these four queues to fill and cap empty bottles in the order of their arrival. All the four queues will use your implementation of LinkedBottleQueue and CircularBottleQueue. The capacity of these queues is initialized in the BottlefillerMachine constructor.
- The *runMachine* method simulates working of the machine and takes as input the number of bottles that need to be filled and capped. It works by moving bottles across the queues whenever they are full or there are no new bottles to add to production. When a bottle is finished (filled, capped and ready to be dispatched), the simulator prints a message to the console.
- An example output for producing 15, 10, 5 and 0 bottles for a given configuration is shown below.

```
Bottle Filler Machine Configuration
Production Line(3) | Filling Queue(2) | Capping Queue(2) | SupplyLine(3)
Bottle Filler Machine to produce 15 bottles.
Bottle SN1Blue:Filled:Capped!
Number of bottles remaining to fill & cap:14
Bottle SN2Blue:Filled:Capped!
Number of bottles remaining to fill & cap:13
Bottle SN3Blue:Filled:Capped!
Number of bottles remaining to fill & cap:12
Bottle SN4Blue:Filled:Capped!
Number of bottles remaining to fill & cap:11
Bottle SN5Blue:Filled:Capped!
Number of bottles remaining to fill & cap:10
Bottle SN6Blue:Filled:Capped!
Number of bottles remaining to fill & cap:9
Bottle SN7Blue:Filled:Capped!
Number of bottles remaining to fill & cap:8
Bottle SN8Blue:Filled:Capped!
Number of bottles remaining to fill & cap:7
Bottle SN9Blue:Filled:Capped!
Number of bottles remaining to fill & cap:6
Bottle SN10Blue:Filled:Capped!
Number of bottles remaining to fill & cap:5
Bottle SN11Blue:Filled:Capped!
Number of bottles remaining to fill & cap:4
Bottle SN12Blue:Filled:Capped!
Number of bottles remaining to fill & cap:3
Bottle SN13Blue:Filled:Capped!
Number of bottles remaining to fill & cap:2
Bottle SN14Blue:Filled:Capped!
```

```
Number of bottles remaining to fill & cap:1
Bottle SN15Blue:Filled:Capped!
Number of bottles remaining to fill & cap:0
All bottles filled!
Bottle Filler Machine to produce 10 bottles.
Bottle SN1Blue:Filled:Capped!
Number of bottles remaining to fill & cap:9
Bottle SN2Blue:Filled:Capped!
Number of bottles remaining to fill & cap:8
Bottle SN3Blue:Filled:Capped!
Number of bottles remaining to fill & cap:7
Bottle SN4Blue:Filled:Capped!
Number of bottles remaining to fill & cap:6
Bottle SN5Blue:Filled:Capped!
Number of bottles remaining to fill & cap:5
Bottle SN6Blue:Filled:Capped!
Number of bottles remaining to fill & cap:4
Bottle SN7Blue:Filled:Capped!
Number of bottles remaining to fill & cap:3
Bottle SN8Blue:Filled:Capped!
Number of bottles remaining to fill & cap:2
Bottle SN9Blue:Filled:Capped!
Number of bottles remaining to fill & cap:1
Bottle SN10Blue:Filled:Capped!
Number of bottles remaining to fill & cap:0
All bottles filled!
Bottle Filler Machine to produce 5 bottles.
Bottle SN1Blue:Filled:Capped!
Number of bottles remaining to fill & cap:4
Bottle SN2Blue:Filled:Capped!
Number of bottles remaining to fill & cap:3
Bottle SN3Blue:Filled:Capped!
Number of bottles remaining to fill & cap:2
Bottle SN4Blue:Filled:Capped!
Number of bottles remaining to fill & cap:1
Bottle SN5Blue:Filled:Capped!
Number of bottles remaining to fill & cap:0
All bottles filled!
No bottles to fill! Non-positive number of bottles 0 requested
```

- If you are going to use the simulation for multiple runs, note that you might want to reset your bottle counter if you want to reset the bottle order sequence for every run of the

simulation.

7 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through Gradescope. The only FOUR files that you must submit are:

- `LinkedBottleQueue.java`
- `CircularBottleQueue.java`
- `BottleQueueIterator.java`
- `BottleFactoryTester.java`.

Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline of **9:59PM on April 20**th. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline.