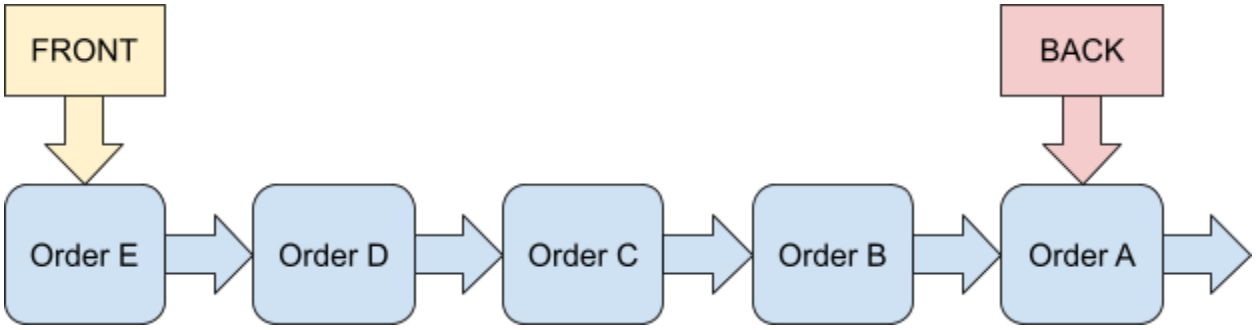
P08 Order Up!

Overview

An incredibly common application of a queue is in a restaurant kitchen. Orders are generally added to the queue in the order in which they are placed, and then the one that's been in the queue longest will be the next one filled.

(In practice skilled chefs will have a more sophisticated queue system, but this is the general idea.)

For this assignment you will implement a basic linked queue system to store Orders for a restaurant. Pay attention -- we'll be revisiting this idea later…



Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas only until 11:59PM on **04/12**.
15 points	**Immediate Automated Tests**: accessible by submission to Gradescope. You will receive feedback from these tests *before* the submission deadline and may make changes to your code in order to pass these tests. Passing all immediate automated tests does **not** guarantee full credit for the assignment.
25 points	**Additional Automated Tests**: these will also run on submission to Gradescope, but you will not receive feedback from these tests until after the submission deadline.
5 points	**Manual Grading Feedback**: TAs or graders will manually review your code, focusing on algorithms, use of programming constructs, and style/readability.

Learning Objectives

The goals of this assignment are:

- Explore a basic linked queue implementation
- Learn to use Java's Iterator and Iterable interfaces to enable enhanced-for loops
- Additional practice with object-oriented programming and unit test development

Additional Assignment Requirements and Notes

As you work on your code and before you submit your assignment, verify that:

- ▢ You have NOT added any instance or class constants, or instance or class variables, or public methods (including constructors) beyond what is defined or provided in this writeup. You are allowed to add *private* helper methods.
- ▢ You *CAN* have defined any local variables you may need to implement the methods in this specification.
- ▢ All methods, public or private, have their own Javadoc-style method header comments in accordance with the CS 300 Course Style Guide. You are allowed to use any text provided in your Javadoc comments.
- ▢ You have NOT submitted the provided Order, LinkedOrder, and QueueADT files to Gradescope.
- ▢ You have NOT imported anything into your files EXCEPT java.util.**Iterator** and java.util.**NoSuchElementException**.
- ▢ You have adhered to the Academic Conduct Expectations and Advice. If you're feeling the strain of this incredibly difficult Spring 2021 semester and panicking, *please* don't hesitate to contact your instructor. We understand, we're feeling it too. We want you to succeed in this course, but not at the expense of your mental or physical health, or your academic integrity. We can always help you work something out.

CS 300 Assignment Requirements

You are responsible for following the requirements listed on both of these pages on all CS 300 assignments, whether you've read them recently or not. Take a moment to review them if it's been a while:

- Academic Conduct Expectations and Advice, which addresses such questions as:
  - How much can you talk to your classmates?
  - How much can you look up on the internet?
  - What do I do about hardware problems?
  - and more!
- Course Style Guide, which addresses such questions as:
  - What should my source code look like?
  - How much should I comment?
  - and more!

Getting Started

1. Create a new project in Eclipse, called something like **P08 OrderUp**.
   a. Ensure this project uses Java 11. Select "JavaSE-11" under "Use an execution environment JRE" in the New Java Project dialog box.
   b. Do **not** create a project-specific package; use the default package.

2. Download the following three (3) Java source files and add them to your project's src folder:
   a. Order.java
   b. LinkedOrder.java
   c. QueueADT.java

3. Create three (3) additional Java source files within that project's src folder:
   a. **OrderIterator**.java (implements java.util.Iterator<Order>, does NOT include a main method)
   b. **OrderQueue**.java (implements QueueADT<Order> and java.util.Iterable<Order>; does NOT include a main method)
   c. **OrderQueueTester**.java (includes a main method)

Read through the provided source code files. Don't alter them! You are not allowed to submit these files, so your code will use our versions of them (identical to what's provided here).

Order is a straightforward object, similar to P07's Box class.

LinkedOrder is a singly-linked list node, similar to P07's LinkedBox class.

QueueADT is an interface providing abstract methods for your queue to implement.

Implementation Requirements Overview

OrderQueueTester is your unit testing class. As before, your tester methods should be static boolean methods with no parameters.

You are required to have the following methods *exactly*:

- `public static boolean runAllTests()`
  - Returns true if and only if all test methods succeed; false otherwise
- `public static void main(String[] args)`
  - Calls your runAllTests() method

As you develop the rest of your program, you should create AT LEAST FOUR test methods of your own design, which you should call from runAllTests(). We will call your test methods on several OrderQueue implementations (some working, some broken) in our automated tests. Protect against exceptions!

OrderIterator implements the **Iterator<Order>** interface. As such, it's very short.

It contains only one private data field:

- `current`, the LinkedOrder that it's currently using

And three public methods:

- `public OrderIterator(LinkedOrder start)`
  - Constructor, initializes `current` to the provided starting LinkedOrder.
  - Does not care whether the argument value is null.
- `public boolean hasNext()`
  - Returns true if and only if the iteration has more orders
- `public Order next() throws NoSuchElementException`
  - Throws a NoSuchElementException with a descriptive error message if the iteration does not have more orders to return.
  - Otherwise returns the next Order and updates the `current` field appropriately.

STOP! Test your iterator in your tester file now. Create a chain of LinkedOrders using the provided LinkedOrder constructor and mutator method, then create an OrderIterator with the first LinkedOrder in the chain and verify that these methods work as you expect.

(Next page…)

OrderQueue implements the **QueueADT<Order>** AND **Iterable<Order>** interfaces.

It contains three private data fields:

- `front`, a reference to the LinkedOrder at the front of the queue
- `back`, a reference to the LinkedOrder at the back of the queue
- `size`, an integer variable tracking the number of Orders currently in the queue

The only methods you will need to implement for this class come from the two interfaces. From **Iterable**:

- `public Iterator<Order> iterator()`
  - Creates and returns a new OrderIterator beginning with the current value of front

And as you add these methods from **QueueADT**, predict what they should do and **TEST THEM**:

- `public void enqueue(Order newElement)`
  - Adds a new LinkedOrder containing newElement to the **back** of the queue, updating the size variable and front/back references appropriately
- `public Order dequeue() throws NoSuchElementException`
  - Removes the next LinkedOrder from the **front** of the queue and returns its Order, updating the size variable and front/back references appropriately
  - Throws a NoSuchElementException if the queue is empty
- `public Order peek() throws NoSuchElementException`
  - Returns the Order from the LinkedOrder at the **front** of the queue *without* removing the LinkedOrder from the queue
  - Throws a NoSuchElementException if the queue is empty
- `public boolean isEmpty()`
  - Returns true if and only if the queue is empty

OrderQueue must also contain the `toString()` method given on the next page. Do not modify this code.

Additionally, be sure to watch Piazza for some sample output coming in the next few days…

```java
/**
 * Creates and returns a String representation of this OrderQueue
 * using an enhanced-for loop. For example, a queue with three Orders
 * might look like this:
 *  1001: fries (2) -> 1002: shake (1) -> 1003: burger (3) -> END
 *
 * @return A String representation of the queue
 */
@Override
public String toString() {
    if (this.size == 0) return "";
    String qString = "";
    for (Order o : this) {
      qString += o.toString();
      qString += " -> ";
    }
    qString += "END";
    return qString;
}
```

Assignment Submission

Hooray, you've finished this CS 300 programming assignment!

> ** **Before you submit** ** go over the checklist on page 2 again, and verify that you've followed all the requirements listed there.

Once you're satisfied with your work, both in terms of adherence to this specification and the academic conduct and style guide requirements, submit your source code through Gradescope.

For full credit, please submit ONLY the following files (source code, *not* .class files):

- OrderIterator.java
- OrderQueue.java
- OrderQueueTester.java

If you submit any other files, you may lose points! Be careful.

Your score for this assignment will be based on the submission marked "**active**" prior to the deadline. You may select which submission to mark active at any time, but by default this will be your most recent submission.

Copyright Notice