

Homework M11: Queues

Due Apr 25 at 11:59pm **Points** 100 **Questions** 27
Available until May 30 at 11:59pm **Time Limit** None

Instructions

Review the [Homework FAQ](#) page for details about submitting homework. In this homework, you will:

- trace the use of queues, dequeues, and priority queues
- write code to use queues as a client
- write code to go inside the queue classes (from the implementation perspective)
- write code to create a deque implementation



Homework Files

Below is the driver/tester program. I strongly recommend using this to test your code before submitting. You can ignore the *test methods* at the end of the file.

Important Note: Use the provided files below for the homework. In order to get the tester program to run, I modified the two queue classes and made the instance data variables public. I also added the extra credit method to the queue interface.

[HomeworkM11Files.zip](#) (https://ccsf.instructure.com/courses/47904/files/7254654/download?download_frd=1)

This quiz was locked May 30 at 11:59pm.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	5,394 minutes	98 out of 100

Score for this quiz: **98** out of 100

Submitted Apr 24 at 1:43am

This attempt took 5,394 minutes.

Coding Questions

The next three questions ask you to write **splice** methods. A splice method combines two queues by adding all elements of a second queue to the end of a first queue.

For full credit:

- The secondQueue should **not** be altered when the method completes.
 - If you destroy the queue, be sure to rebuild it.
 - For a linked queue, do not connect the secondQueue to the first- add the actual *elements* to the first queue.
- Write an efficient solution.
 - For full credit, your code should be $O(n)$.
- When working from the implementation perspective:
 - Take advantage of being able to directly access the array or linked nodes of the secondQueue.
 - This means your solution should **not** be the same as the client perspective.
 - In other words, do **not** destroy and rebuild secondQueue. Access it directly through the array or linked nodes.
 - **You will receive 0 points for solutions that are the same as the client perspective solution.**
 - Note that it **is** okay to invoke any the $O(1)$ methods in the current ArrayQueue or LinkedList class.
 - You **can** invoke the enqueue method because it is $O(1)$.
- Use **only** queues in your answer. Do **not** use another data structure (e.g., arrays, ArrayList, LinkedList, etc.).

Question 1

15 / 15 pts

Write a splice method from the **client perspective**.

```
public static void splice(QueueInterface firstQueue, QueueInterface secondQueue)
```

- Because this is the client perspective, you do **not** know how the queues are implemented. Use **only** on the methods in QueueInterface.
- The secondQueue should not be altered when the method completes. If you destroy the queue, be sure to rebuild it.

Your Answer:

```
public static void splice(QueueInterface firstQueue, QueueInterface secondQueue) {
    if (!secondQueue.isEmpty()) {
        secondQueue.enqueue(null);

        while (secondQueue.getFront() != null) {
            Object queueItem = secondQueue.dequeue();
            firstQueue.enqueue(queueItem);
            secondQueue.enqueue(queueItem);
        }
        secondQueue.dequeue();
    }
}
```

Question 2

15 / 15 pts

Write a splice method from the **implementation perspective** for the **ArrayQueue** class.

```
public void splice(ArrayQueue<T> secondQueue)
```

- The secondQueue should **not** be altered when the method completes.
- Make sure to account for the "wrap around" nature of the ArrayQueue.
- Do **not** submit the same solution as the client version you wrote in Question 1.
 - Do **not** empty and rebuild the secondQueue.
 - Take advantage of directly accessing the array of secondQueue.

Your Answer:

```

public void splice(ArrayQueue<T> anotherQueue) {
    if (!anotherQueue.isEmpty()) {
        // modulus accounts for out of bounds index error
        // or accounts for looping
        for (int i = anotherQueue.frontIndex; i != anotherQueue.backIndex;
i = (i + 1)
            % anotherQueue.queue.length) {
            this.enqueue(anotherQueue.queue[i]);
        }
        // have to explicitly code the last enqueue because "last loop" doe
sn't
        // execute
        this.enqueue(anotherQueue.queue[anotherQueue.backIndex]);
    }
}

```

Question 3

14 / 15 pts

Write a splice method from the **implementation perspective** for the **LinkedList** class.

```
public void splice(LinkedList<T> secondQueue)
```

- The secondQueue should **not** be altered when the method completes.
- Make sure to account for special cases like empty and singleton queues.
- Do **not** submit the same solution as the client version you wrote in Question 1.
 - Do **not** empty and rebuild the secondQueue.
 - Take advantage of directly accessing the node variables of secondQueue.

Your Answer:

```

public void splice(LinkedList<T> anotherQueue) {
    if (!this.isEmpty()) {
        Node currentNode = anotherQueue.firstNode;

        while (currentNode != null) {
            this.enqueue(currentNode.data);

            currentNode = currentNode.getNextNode();
        }
    } else {
        firstNode = anotherQueue.firstNode;
    }
}

```

```
        lastNode = anotherQueue.lastNode;
    }
}
```

-1 when the current queue is empty, this implementation links the actual elements of the second chain onto the first one- rather than creating new nodes to include in the original queue; this links the queues together so that future changes to secondQueue will affect the spliced firstQueue

Implement DequeInterface using a list as the underlying data structure. The class header is:

```
public class ListDeque<T> implements DequeInterface
<T>
```

The instance data variable is:

```
private List<T> list; // note: this is List from th
e Java standard library java.util package
```

- Invoke methods on the List object to implements the methods of the DequeInterface.
 - Use the List methods whenever possible rather than manually replicating the code.
- You should throw an exception from methods that fail (e.g., removing from the back of an empty deque).
- In addition to implementing all of the DequeInterface methods, you will need a constructor.
 - Carefully choose how to initialize your list object so that you create an efficient deque.
- I did **not** include test cases in the provided driver file.
 - I recommend you write your own test cases to make sure that your deque works as it should.
 - You can also modify and run the ArrayDequeTester or LinkedDequeTester file from the lecture notes using this new deque implementation.

Question 4**11 / 14 pts**

Upload your complete ListDeque.java class here.

↓ [ListDeque.java \(https://ccsf.instructure.com/files/7736350/download\)](https://ccsf.instructure.com/files/7736350/download)

-3 use the methods of the List interface rather than recreating an array-based implementation of the deque (e.g., `get(0)`, `get(list.size()-1)`, `add(0, element)`, `remove(0)`, etc.); initializing with `LinkedList` instead of `ArrayList` will still allow this to be an efficient implementation and is a much better way to use the behind-the-scenes data object; post a comment on this submission if you want to discuss or possibly revise

Short Answer and Multiple Choice Questions

Question 5**1 / 1 pts**

The enqueue and dequeue operations of a queue are both linear $O(n)$.

☐ True

Correct!☒ False**Question 6****1 / 1 pts**

In an ArrayQueue object, the front of the queue is always located in index 0.

☐ True**Correct!**☒ False

The front of the queue will be in frontIndex, which can be located anywhere in the array.

Question 7**1 / 1 pts**

In LinkedList, if firstNode==lastNode returns true, that means the queue is empty.

☐ True**Correct!**☒ False

The queue could be empty but it could also be a singleton queue.

Question 8**2 / 2 pts**

A queue is initially empty. What are the contents of the queue after the following code is executed? Answer choices are listed **FRONT ... BACK**.

```
queue.enqueue(1);  
queue.enqueue(4);  
queue.enqueue(2);  
queue.dequeue();
```

- ☐ front 2 4 1 back
- ☐ none of these is correct
- ☐ front 1 4 2 back
- ☐ front 2 4 back
- ☒ front 4 2 back

Correct!**Question 9****2 / 2 pts**

A queue is initially empty. What are the contents of the queue after the following code is executed? Answer choices are listed **FRONT ... BACK**.

```
queue.enqueue(5);  
queue.enqueue(2);  
queue.enqueue(7);  
System.out.println(queue.getFront());
```

- ☐ none of these is correct
- ☐ front 2 5 back

Correct!☐ front 7 7 2 5 back☒ front 5 2 7 back☐ front 5 5 2 7 back☐ front 7 2 5 back☐ front 2 7 back**Question 10****2 / 2 pts**

A queue is initially empty. What are the contents of the queue after the following code is executed? Answer choices are listed **FRONT ... BACK**.

```
queue.enqueue(4);  
queue.enqueue(queue.getFront());
```

☐ front 4 back☐ none of these is correct☐ front 4 4 4 back**Correct!**☒ front 4 4 back☐ the queue will be empty**Question 11****2 / 2 pts**

A queue is initially empty. What are the contents of the queue after the

following code is executed? Answer choices are listed **FRONT ... BACK**.

```
queue.enqueue(3);  
queue.enqueue(6);  
queue.enqueue(1);  
queue.enqueue(queue.dequeue());
```

☐ front 6 1 3 6 back

☐ front 3 1 6 back

☐ front 3 6 1 3 back

☐ front 3 3 6 1 back

Correct!

☒ front 6 1 3 back

☐ none of these is correct

☐ front 6 6 1 3 back

☐ front 3 6 1 back

Question 12

2 / 2 pts

A deque is initially empty. What are the contents of the deque after the following code is executed? Answer choices are listed **FRONT ... BACK**.

```
deque.addToFront(1);  
deque.addToBack(4);  
deque.addToFront(2);
```

Correct!

☒ front 2 1 4 back

☐ front 1 4 2 back

- ☐ front 4 2 1 back
- ☐ front 2 4 1 back
- ☐ none of these is correct
- ☐ front 1 2 4 back
- ☐ front 4 1 2 back

Question 13**2 / 2 pts**

A deque is initially empty. What are the contents of the deque after the following code is executed? Answer choices are listed **FRONT ... BACK**.

```
deque.addToBack(3);  
deque.addToFront(2);  
deque.addToFront(6);  
deque.removeBack();
```

- ☐ front 2 3 back
- ☐ none of these is correct
- ☐ front 2 6 back
- ☐ front 3 2 back
- ☒ front 6 2 back
- ☐ front 3 6 back
- ☐ front 6 3 back

Correct!

Question 14**2 / 2 pts**

A deque is initially empty. What are the contents of the deque after the following code is executed? Answer choices are listed **FRONT ... BACK**.

```
deque.addToBack(7);  
deque.addToBack(2);  
deque.addToBack(3);  
deque.removeFront();
```

- ☐ front 3 2 back
- ☐ none of these is correct
- ☐ front 2 7 back
- ☒ front 2 3 back
- ☐ front 7 3 back
- ☐ front 3 7 back
- ☐ front 7 2 back

Correct!**Question 15****2 / 2 pts**

A deque is initially empty. What are the contents of the deque after the following code is executed? Answer choices are listed **FRONT ... BACK**.

```
deque.addToFront(2);  
deque.addToFront(8);  
deque.addToFront(deque.getFront());
```

Correct!

- ☐ front 8 8 back
- ☐ front 8 2 back
- ☐ front 8 2 8 back
- ☐ none of these is correct
- ☐ front 2 8 8 back
- ☒ front 8 8 2 back
- ☐ front 2 8 back

Question 16**2 / 2 pts**

A deque is initially empty. What are the contents of the deque after the following code is executed? Answer choices are listed **FRONT ... BACK**.

```
deque.addToFront(1);  
deque.addToBack(4);  
deque.addToFront(deque.getBack());
```

- ☐ front 4 1 back
- ☐ front 1 1 back
- ☐ front 1 4 1 back
- ☐ front 4 4 back
- ☐ front 4 1 1 back
- ☐ front 1 1 back

Correct!☒ none of these is correct☐ front 1 1 4 back

The next set of questions asks about an initially empty **priority queue**.

Assume that "obj-X" represents an object and "priority-#" represents the priority. For example:

- [priority-1, obj-A] is an element that holds some object called A and has priority 1
- [priority-2, obj-A] is an element that holds some object called A and has priority 2
- [priority-1, obj-B] is an element that holds some object called B and has priority 1

Lower numbers have higher priorities. For example:

- [priority-1, obj-A] has the same priority as [priority-1, obj-B]
- [priority-2, obj-B] has a higher priority than [priority-3, obj-A]

The **front** of the priority queue is on the **left** and the back on the right.

Question 17

2 / 2 pts

A priorityQueue is initially empty. What are the contents of the priorityQueue after the following code is executed? Answer choices are listed **FRONT ... BACK**.

The answers are abbreviated (e.g., [priority-4, obj-F] is written 4-F in the answer choice.

```
priorityQueue.add([priority-1, obj-D]);  
priorityQueue.add([priority-2, obj-E]);
```

```
priorityQueue.add([priority-1, obj-A];  
priorityQueue.add([priority-2, obj-A];
```

☐ front [1-A] [1-D] [2-A] [2-E] back

☐ front [2-A] [2-E] [1-A] [1-D] back

☐ front [1-D] [2-E] [1-A] [2-A] back

Correct!

☒ front [1-D] [1-A] [2-E] [2-A] back

☐ none of these is correct

☐ front [2-E] [2-A] [1-D] [1-A] back

☐ front [1-A] [2-A] [1-D] [2-E] back

Question 18

2 / 2 pts

A priorityQueue is initially empty. What are the contents of the priorityQueue after the following code is executed? Answer choices are listed **FRONT ... BACK**.

The answers are abbreviated (e.g., [priority-4, obj-F] is written 4-F in the answer choice.

```
priorityQueue.add([priority-2, obj-B];  
priorityQueue.add([priority-3, obj-C];  
priorityQueue.add([priority-1, obj-A];  
priorityQueue.add([priority-1, obj-D];
```

☐ front [3-C] [2-B] [1-A] [1-D] back

Correct!

☒ front [1-A] [1-D] [2-B] [3-C] back

- ☐ none of these is correct
- ☐ front [2-B] [3-C] [1-A] [1-D] back
- ☐ front [1-D] [1-A] [2-B] [3-C] back

Question 19**2 / 2 pts**

A priorityQueue is initially empty. What are the contents of the priorityQueue after the following code is executed? Answer choices are listed **FRONT ... BACK**.

The answers are abbreviated (e.g., [priority-4, obj-F] is written 4-F in the answer choice.

```
priorityQueue.add([priority-2, obj-B];  
priorityQueue.add([priority-2, obj-C];  
priorityQueue.add([priority-1, obj-A];  
priorityQueue.remove();
```

- ☐ front [2-C] [1-A] back
- ☐ front [2-C] [2-B] back
- ☒ front [2-B] [2-C] back
- ☐ front [2-B] [1-A] back
- ☐ front [1-A] [2-B] back
- ☐ front [1-A] [2-C] back
- ☐ none of these is correct

Correct!

Question 20**0 / 2 pts**

A priorityQueue is initially empty. What are the contents of the priorityQueue after the following code is executed? Answer choices are listed **FRONT ... BACK**.

The answers are abbreviated (e.g., [priority-4, obj-F] is written 4-F in the answer choice.

```
priorityQueue.add([priority-1, obj-A];  
priorityQueue.add([priority-2, obj-B];  
priorityQueue.add(priorityQueue.remove());
```

you Answered☐ front [2-B] [1-A] back☒ none of these is correct☐ front [2-B] [1-A] [1-A] back☐ front [2-B] [2-B] [1-A] back☐ front [1-A] [2-B] [2-B] back☐ front [1-A] [2-B] [1-A] back**Correct Answer**☐ front [1-A] [2-B] back**Question 21****2 / 2 pts**

A priorityQueue is initially empty. What are the contents of the priorityQueue after the following code is executed? Answer choices are listed **FRONT ... BACK**.

The answers are abbreviated (e.g., [priority-4, obj-F] is written 4-F in the

answer choice.

```
priorityQueue.add([priority-1, obj-C];
priorityQueue.add([priority-2, obj-F];
priorityQueue.add(priorityQueue.getFront());
```

Correct!

☒ front [1-C] [1-C] [2-F] back

☐ front [1-C] [2-F] [2-F] back

☐ front [1-C] [1-C] back

☐ front [1-C] [2-F] back

☐ none of these is correct

☐ front [1-C] [2-F] [1-C] back

☐ front [2-F] [1-C] [1-C] back

For the next questions, use the WaitLine simulation described in Segment 7.8 (or 10.8 in the older editions). In this simulation:

- only one customer can be served at a time
- a customer can be served as soon as the previous customer leaves (e.g., if one customer leaves at time 4, the next customer can begin being served at time 4)

The partially-complete table below gives some enter, begin, wait, and departure times and the transaction length for each customer.

Customer	Enter Time	Begin Time	Wait Time Length	Transaction Length	Departure Time
CustomerA	5	5	0	2	7
CustomerB	6	7		3	

CustomerC	7			4	
-----------	---	--	--	---	--

Question 22**2 / 2 pts**

What is CustomerB's wait time?

Correct!**Correct Answers**

1 (with margin: 0)

Question 23**0 / 2 pts**

What is CustomerB's departure time?

Not Answered**Correct Answers**

10 (with margin: 0)

Question 24**0 / 2 pts**

What is CustomerC's wait time?

Not Answered**Correct Answers**

3 (with margin: 0)

Use this table of a new simulation for the next questions.

Customer	Enter Time	Begin Time	Wait Time Length	Transaction Length	Departure Time
CustomerQ	2	2	0	3	5
CustomerR	5	5		2	
CustomerS	9			4	

Question 25**2 / 2 pts**

What is CustomerR's departure time?

Correct!**Correct Answers**

7 (with margin: 0)

Question 26**2 / 2 pts**

What is CustomerS's wait time?

Correct!**Correct Answers**

0 (with margin: 0)

Optional Extra Credit (10 Points)

A queue lets you get the front entry without removing it. For some applications, you might also want to look at the entry behind the front entry without removing it.

Write **two** `getSecond` methods, one for **`ArrayQueue`** and one for **`LinkedList`**.

- If the queue has two entries or more, `getSecond` returns the second entry (the entry after the front) without altering the queue.
- If the queue has fewer than two entries, `getSecond` should throw an exception (to be consistent with the 5th edition versions of the queues).

Here is the method header for both classes:

```
public T getSecond()
```

Question 27

8 / 0 pts

Paste your **two** complete `getSecond` methods here.

Your Answer:

```
public T getSecond() {
    if (queue.length < 2) {
        throw new EmptyQueueException();
    } else {
        return queue[((frontIndex + 1) % queue.length)];
    }
}

public T getSecond() {
    Node current = firstNode;

    if (!this.isEmpty() && current.next != null) {
        return current.next.data;
    } else {
        throw new EmptyQueueException();
    }
}
```

```
}
```

-2 queue.length is not the number of elements in the queue-
cannot use this to determine the size

Quiz Score: **98** out of 100