Homework 9

1 Part I: Using Queues

1. Show the contents of two initially empty **queues** q1 and q2 after **each** of the following statements execute. Assume a, b, c, and d are objects. Make sure you specify which side is the front and which side is the back.

   - q1.enqueue(a)
   - q1.enqueue(b)
   - q2.enqueue(c)
   - q2.enqueue(d)
   - q1.enqueue(q2.dequeue())
   - q1.enqueue(q2.getFront())
   - q2.enqueue(q1.getFront())
   - q2.dequeue()

   The front of the queues will be on the left, and the back of the queues will be on the right.

   - q1.enqueue(a)
     q1: a
     q2:
   - q1.enqueue(b)
     q1: a, b
     q2:
   - q2.enqueue(c)
     q1: a, b
     q2: c
   - q2.enqueue(d)
     q1: a, b
     q2: c, d
   - q1.enqueue(q2.dequeue())
     q1: a, b, c
     q2: d
   - q1.enqueue(q2.getFront())
     q1: a, b, c, d
     q2: d
   - q2.enqueue(q1.getFront())
     q1: a, b, c, d
     q2: d, a
   - q2.dequeue()
     q1: a, b, c, d
     q2: a

2. Show the contents of two initially empty **deques** dq1 and dq2 after **each** of the following statements execute. Assume a, b, c, and d are objects. Make sure you specify which side is the front and which side is the back.

- dq1.addToFront(a)
- dq1.addToBack(b)
- dq1.addToFront(c)
- dq2.addToFront(c)
- dq2.addToBack(d)
- dq1.addToFront(dq2.removeBack())
- dq1.addToBack(dq2.getFront())
- dq2.addToFront(dq1.removeBack())
- dq2.removeFront()

The front of the deques will be on the left, and the back of the deques will be on the right.

- dq1.addToFront(a)
  dq1: a
  dq2:
- dq1.addToBack(b)
  dq1: a, b
  dq2:
- dq1.addToFront(c)
  dq1: c, a, b
  dq2:
- dq2.addToFront(c)
  dq1: c, a, b
  dq2: c
- dq2.addToBack(d)
  dq1: c, a, b
  dq2: c, d
- dq1.addToFront(dq2.removeBack())
  dq1: d, c, a, b
  dq2: c
- dq1.addToBack(dq2.getFront())
  dq1: d, c, a, b, c
  dq2: c
- dq2.addToFront(dq1.removeBack())
  dq1: d, c, a, b
  dq2: c, c
- dq2.removeFront()
  dq1: d, c, a, b
  dq2: c

3. Show the contents of an initially empty **priority queue** pq1 after **each** of the following statements execute. Assume that the first character of the object below specifies its priority. For example, object 1a is an object with priority 1. Assume lower numbers have higher priorities. So 2b has a higher priority than 3d; 2b has the same priority as 2e. Make sure yo specify which side is the front and which side is the back.

- pq1.add(2c)
- pq1.add(1d)
- pq1.add(2a)
- pq1.add(1b)
- pq1.remove()
- pq1.add(pq1.getFront())
- pq1.add(pq1.remove())
- pq1.remove()

The front of the priority queue is to the left, and the back of the priority queue is to the right.

- pq1.add(2c)
  pq1: 2c
- pq1.add(1d)
  pq1: 1d, 2c
- pq1.add(2a)
  pq1: 1d, 2c, 2a
- pq1.add(1b)
  pq1: 1d, 1b, 2c, 2a
- pq1.remove()
  pq1: 1b, 2c, 2a
- pq1.add(pq1.getFront())
  pq1: 1b, 1b, 2c, 2a
- pq1.add(pq1.remove())
  pq1: 1b, 1b, 2c, 2a
- pq1.remove()
  pq1: 1b, 2c, 2a

4. Complete the simulation begun in Figure 23-6. Let Customer 6 enter the line at time 10 with a transaction time of 2.

The front of the queue will be on the left, and the back of the queue will be on the right. Each entry in the queue will have the following format: (customer number, waiting time before service, time left for transaction)

- Time: 0
  Customer **1** enters line with a 5-minute transaction.
  Customer **1** begins service after waiting 0 minutes.
  Queue: (1, 0, 5)
- Time: 1
  Customer **1** continues to be served.
  Queue: (1, 0, 4)
- Time: 2
  Customer **1** continues to be served.
  Customer **2** enters line with a 3-minute transaction.
  Queue: (1, 0, 3), (2, 3, 3)

- Time: 3
  Customer **1** continues to be served.
  Queue: (1, 0, 2), (2, 2, 3)
- Time: 4
  Customer **1** continues to be served.
  Customer **3** enters line with a 1-minute transaction.
  Queue: (1, 0, 1), (2, 1, 3), (3, 4, 1)
- Time: 5
  Customer **1** finishes and departs
  Customer **2** begins service after waiting 3 minutes.
  Customer **4** enters line with a 2-minute transaction.
  Queue: (2, 0, 3), (3, 3, 1), (4, 4, 2)
- Time: 6
  Customer **2** continues to be served.
  Queue: (2, 0, 2), (3, 2, 1), (4, 3, 2)
- Time: 7
  Customer **2** continues to be served.
  Customer **5** enters line with a 4-minute transaction.
  Queue: (2, 0, 1), (3, 1, 1), (4, 2, 2), (5, 4, 4)
- Time: 8
  Customer **2** finishes and departs.
  Customer **3** begins service after waiting 4 minutes.
  Queue: (3, 0, 1), (4, 1, 2), (5, 3, 4)
- Time: 9
  Customer **3** finishes and departs.
  Customer **4** begins service after waiting 4 minutes.
  Queue: (4, 0, 2), (5, 2, 4)
- Time: 10
  Customer **4** continues to be served.
  Customer **6** enters line with a 2-minute transaction.
  Queue: (4, 0, 1), (5, 1, 4), (6, 5, 2)
- Time: 11
  Customer **4** finishes and departs.
  Customer **5** begins service after waiting 4 minutes.
  Queue: (5, 0, 4), (6, 4, 2)
- Time: 12
  Customer **5** continues to be served.
  Queue: (5, 0, 3), (6, 3, 2)
- Time: 13
  Customer **5** continues to be served.
  Queue: (5, 0, 2), (6, 2, 2)
- Time: 14
  Customer **5** continues to be served.
  Queue: (5, 0, 1), (6, 1, 2)

- Time: 15
  Customer **5** finishes and departs.
  Customer **6** begins service after waiting 5 minutes.
  Queue: (6, 0, 2)
- Time: 16
  Customer **6** continues to be served.
  Queue: (6, 0, 1)
- Time: 17
  Customer **6** finishes and departs.
  Queue:

5. Write a method from the client perspective to test whether a String is a palindrome using a deque. Your method header should be: `public boolean isPalindrome(String s)`. Your method should not be recursive. Use only the methods of `DequeInterface`, defined in Section 15. (You can create a new object of type LinkedDeque, such as: `DequeInterface stack = new LinkedDeque();`).

   Refer to Homework9Driver.java

2 Part II: Queue Implementations

3. Suppose that we want to add a method to a class of queues that will splice two queues together. This method adds to the end of a queue all items that are in a second queue. The header of the method could be as follows:

   ```
   public void splice(QueueInterface<T> anotherQueue)
   ```

   Write this method in such a way that it will work in any class that implements `QueueInterface<T>`.

   - Be careful to either not destroy the second queue, or to recreate it if it has been destroyed.
   - You can only invoke methods from the `QueueInterface` class on the `anotherQueue` parameter.
     The `QueueInterface.java` file is provided.

   Refer to LinkedQueue.java

4. Consider the method `splice` that Exercise 3 describes. Implement this method specifically for the class `ArrayQueue`. Take advantage of your ability to manipulate the array representation of the queue.

   - Be careful to either not destroy the second queue, or to recreate it if it has been destroyed.
   - This method would go inside the `ArrayQueue` class. The parameter of the method is of type `ArrayQueue`. The `ArrayQueue.java` file is provided. You should take advantage of the fact that you can access the array directly. Hint: it's more efficient if you do not destroy the second array and thus do not have to recreate it. Your implementation should be different from #3

   Refer to ArrayQueue.java

5. Consider the method `splice` that Exercise 3 describes. Implement this method specifically for the class `LinkedQueue`. Take advantage of your ability to manipulate the chain that represents the queue.

   - Be careful to either not destroy the second queue, or to recreate it if it has been destroyed.
   - This method would go inside the `LinkedQueue` class. The parameter of the method is of type `LinkedQueue`. The `LinkedQueue.java` file is provided. You should take advantage of the fact that you can access the linked nodes directly. (See hint for #4.) Your implementation should be different from #3.

   Refer to LinkedQueue.java

6. Implement the ADT queue by using an ADT list to contain its entries.

   - Similar to #7, except that you are using an instance variable of type `ListInterface` as the data structure.

   Refer to QueueFromList.java

7. Implement the ADT queue by using an ADT deque to contain its entries.

   - You are writing a class with the header: `public class Queue<T> implements QueueInterface<T>`. You will have an instance variable of type `DequeInterface`. This is the data structure you should use to implement the `QueueInterface` methods. In addition to implementing all of the `QueueInterface` methods, you will need a constructor. I have provided the `LinkedDeque.java` file as one implementation of a deque that you might want to use in your constructor.

   Refer to QueueFromDeque.java

8. Implement the ADT stack by using an ADT deque to contain its entries.

   - This is similar to #7, except that you are using a `DequeInterface` object to implement a stack. This class is: `public class Stack<T> implements StackInterface<T>`

   Refer to StackFromDeque.java