```java
/*  Wesley Febrian
 *  CS111C - Jessica Masters
 *
 *  Lab D Part A (40 points)
 *  Implement the ADT queue by using a circular linked chain,
 *  as shown in Figure 24- 12. Recall that this chain has only
 *  an external reference to its last node.
 *  Recall that with a circular linked chain, you only have
 *  one reference to lastNode. To retrieve the first node, you
 *  can reference lastNode.getNextNode().
 */


public class CircularLinkedQueue<T> implements QueueInterface<T>, java.io.Serializable{

    /*
     * Recall that with a circular linked chain, you only have one reference to
     * lastNode. To retrieve the first node, you can reference
     * lastNode.getNextNode().
     */

    private Node lastNode; //references last node in queue

    public CircularLinkedQueue(){
        lastNode = null;
    } // end default constructor

    public void enqueue(T newEntry) {
        // Adds element to the end of this queue.
        Node newNode = new Node(newEntry);

        if (lastNode == null) {
            lastNode = newNode;
            lastNode.setNextNode(lastNode);
        } else {
            // links the newNode to the lastNode's pointer and then 're'points
            // lastNode to the newNode.
            if (lastNode.getNextNode() == null) {
                lastNode.setNextNode(newNode);
                newNode.setNextNode(lastNode);
            } else {
                newNode.setNextNode(lastNode.getNextNode());
                lastNode.setNextNode(newNode);
            }
        }
        // 'repositions' lastNode at the end of the queue.
        lastNode = newNode;
    }

    public T dequeue(){
        T entry;

        if (isEmpty()) { // empty queue
            return null;
        } else

        // 1 entry queue
        if (lastNode.getNextNode().equals(lastNode)) {
            entry = lastNode.getData();
            lastNode.setData(null); // deleting the node's entry
            lastNode.next = null; // deleting the node
            return entry;
        } else {

            // queue more than 1 entry
            Node nodeToRemove = lastNode.getNextNode(); // 1st node
            Node newFirstNode = lastNode.getNextNode().getNextNode();

            entry = nodeToRemove.getData();
            nodeToRemove.next = null; // delete the front entry

            lastNode.next = newFirstNode;

            return entry;
        }
    }

    public T getFront() {
        T front = null;

        if (!isEmpty())
          front = lastNode.getNextNode().getData();
```

```
        return front;
    }

    public boolean isEmpty() {
        // Returns true if this queue is empty; otherwise, returns false.
        boolean result = false;
            if (lastNode == null || lastNode.next == null){
                result = true;;
            }

            return result;
    }

    public void clear() {
        lastNode = null;
        /*
         * OR:
         * while(!isEmpty()){
         * dequeue();
         * {
         */
    }

    private class Node implements java.io.Serializable{
        private T data; //queue entry
        private Node next; //link to next node

        private Node(){
            data = null;
            next = null;
        } // end default constructor

        private T getData(){
            return data;
        }

        private Node(T dataPortion)
        {
            data = dataPortion;
            next = null;
        }

        private Node(Node nextNode)
        {
            data = null;
            next = nextNode;
        }

        private Node(T dataPortion, Node nextNode)
        {
            data = dataPortion;
            next = nextNode;
        }

        private void setData(T entry){
            data = entry;
        }

        private Node getNextNode(){
            return next;
        }

        private void setNextNode(Node entry){
            next = entry;
        }

    } // end Node

} // end CircularLinkedQueue
```