```java
/*  Wesley Febrian
 *  CS111C - Jessica Masters
 *
 *  Part B (60 points): Implement a new kind of queue that only accepts a single copy of an object in the queue
 *  (e.g., it does not allow duplicates).
 *  Use an array-based implementation.
 *  For this ADT, if an object is added that is already in the queue, the queue will remain unchanged.
 *  This ADT will have a method moveToBack that takes an object in the queue and moves it to the back.
 *  If the object is not already in the queue, it adds it to the back of the queue.
 *  Your class should implement NoDuplicatesQueueInterface, which I have provided.
 *  I also provided the driver program to test your implementation.
 *  You might also find it helpful to review some of the files provided in previous assignments.
 */

public class NoDuplicatesArrayQueue <T> implements NoDuplicatesQueueInterface<T>, java.io.Serializable{

    private T[] queue; // circular array of queue entries and one unused location
    private int frontIndex;
    private int backIndex;
    private static final int DEFAULT_INITIAL_CAPACITY = 50;

    public NoDuplicatesArrayQueue() {
        this(DEFAULT_INITIAL_CAPACITY);
    } // end default constructor

    public NoDuplicatesArrayQueue(int initialCapacity) {
        queue = (T[]) new Object[initialCapacity + 1];
        frontIndex = 0;
        backIndex = initialCapacity;
    } // end constructor

    public void enqueue(T newEntry) {
        if (!isDuplicate(newEntry)) { //to make sure no duplicates
            if (isArrayFull()) // isArrayFull and
                doubleArray(); // doubleArray are private

            backIndex = (backIndex + 1) % queue.length;
            queue[backIndex] = newEntry;
        }
    }

    public T dequeue() {
        T front = null;

        if (!isEmpty()) {
            front = queue[frontIndex];
            queue[frontIndex] = null;
            frontIndex = (frontIndex + 1) % queue.length;
        } // end if

        return front;
    }

    public T getFront() {
        T front = null;

        if (!isEmpty())
            front = queue[frontIndex];

        return front;
    }

    public boolean isEmpty() {
        return frontIndex == ((backIndex + 1) % queue.length);
    } // end isEmpty

    public void clear() {
        for(int i = frontIndex; i <= backIndex; i++){
            queue[i] = null;
        }
    }

    public void moveToBack(T entry) {
        if (!isDuplicate(entry)) { // meaning the entry does not exist yet
                backIndex = backIndex + 1;
                queue[backIndex] = entry;
        }

        if(isDuplicate(entry)){
            int indexToBeMoved = 0;
            for(int i = 0; i < queue.length; i++){
                if(entry.equals(queue[i])){
                    indexToBeMoved = i;
```

```
                }
            }
            backIndex = backIndex+1;
            queue[backIndex] = queue[indexToBeMoved];
            queue[indexToBeMoved] = null;

            //shift everything -1 to make array not sparse after moving to the back
            for(int i = indexToBeMoved; i<=backIndex; i++){
                queue[i] = queue[i+1];
                if(queue[backIndex] == null){
                    backIndex = backIndex-1;
                }
            }
        }

    }

    public void display(){
        for (int i = frontIndex; i<=backIndex; i++){
            if(queue[i]==null){
                //do nothing (do not print) if queue[i] is null
            }else
                //System.out.println("Index "+ i + " is " + queue[i]);
                System.out.println(queue[i]);
        }
    }

    //check if an entry exists
    private boolean isDuplicate(T entry){
        boolean isDuplicate = false;

        for(int i = 0; !isDuplicate && i < queue.length; i++){
            if(entry.equals(queue[i])){
                isDuplicate = true;
            }
        }

        return isDuplicate;
    }


    private boolean isArrayFull() {
        return frontIndex == ((backIndex + 2) % queue.length);
        } // end isArrayFull

    private void doubleArray() {
        T[] oldQueue = queue;
        int oldSize = oldQueue.length;

        queue = (T[]) new Object[2 * oldSize];

        for (int index = 0; index < oldSize - 1; index++)
            {
            queue[index] = oldQueue[frontIndex];
            frontIndex = (frontIndex + 1) % oldSize;
            } // end for

        frontIndex = 0;
        backIndex = oldSize - 2;
        } // end doubleArray
}
```