C: > Users > YangYiShu > Desktop > lab > assignment5 > assignment5.java > Test > LinkedQueue<E>

```java
import java.util.*;
import java.io.*;
public class Test {
    //stack interface
    public interface Stack<E> {
        int size();
        boolean isEmpty();
        E top();
        void push(E element);
        E pop();
        }
    //queue interface
    public interface Queue<E> {
        int size();
        boolean isEmpty();
        E first();
        void enqueue(E e);
        E dequeue();
        }
//LinkedQueue class
public class LinkedQueue<E> implements Queue<E>
{
  private SinglyLinkedList<E> list = new SinglyLinkedList<>();
  public LinkedQueue() { }
  public int size() { return list.size(); }
  public boolean isEmpty() { return list.isEmpty(); }
  public void enqueue(E element) { list.addLast(element); }
  public E first() { return list.first(); }
  public E dequeue() { return list.removeFirst(); }
  public String toString()
  {
    return list.toString();
  }
}
//LinkedStack class
public class LinkedStack<E> implements Stack<E>
{
  private SinglyLinkedList<E> list = new SinglyLinkedList<>();
  public LinkedStack() { }
  public int size() { return list.size(); }
  public boolean isEmpty() { return list.isEmpty(); }
  public void push(E element) { list.addFirst(element); }
  public E top() { return list.first(); }
  public E pop() { return list.removeFirst(); }
  public String toString() {
    return list.toString();
  }
}
    // this method implements the required algorithm to search for an element in
    // a stack using a Queue as intermediate storage. returns true if found,
    // else false
```

```java
    public static <E> boolean contains(Stack<E> stk, E element) {
        // creating an empty queue for temporary storage. you can also pass this
        // queue as argument if you prefer that.
        Queue<E> q = new LinkedQueue<E>();
        // initializing a flag, indicating whether the element is found or not,
        // to false
        boolean found = false;
        // looping until either stack is empty or found is true
        while (!stk.isEmpty() && !found) {
            // removing top value from stack
            E value = stk.pop();
            // adding to the end of queue
            q.enqueue(value);
            // looping for q.size()-1 number of times
            for (int i = 0; i < q.size() - 1; i++) {
                // removing front element from queue, adding to the end, so that
                // elements from stack will be added into to the queue in
                // reverse order
                q.enqueue(q.dequeue());
            }
            // if value equals target element, setting found to true
            if (value.equals(element)) {
                found = true;
            }
        }
        // now just moving all elements remaining in q back to the stk
        while (!q.isEmpty()) {
            stk.push(q.dequeue());
        }
        // returning status of found variable
        return found;
    }

    Run | Debug
    public static void main(String[] args) {
        // creating a stack and adding numbers from 1 to 10
        LinkedStack<Integer> stk = new LinkedStack<Integer>();
        for (int i = 1; i <= 10; i++) {
            stk.push(i);
        }
        // testing above method
        System.out.println(contains(stk, element: 5)); // true
        System.out.println(contains(stk, element: 15)); // false
        System.out.println(contains(stk, element: 3)); // true

        // printing the stack to verify that it is not altered. should print
        // numbers from 10 to 1 in reverse order
        System.out.print(s: "Stack: ");
        while (!stk.isEmpty()) {
            System.out.print(stk.pop() + " ");
        }
        System.out.println();
    }
}
```