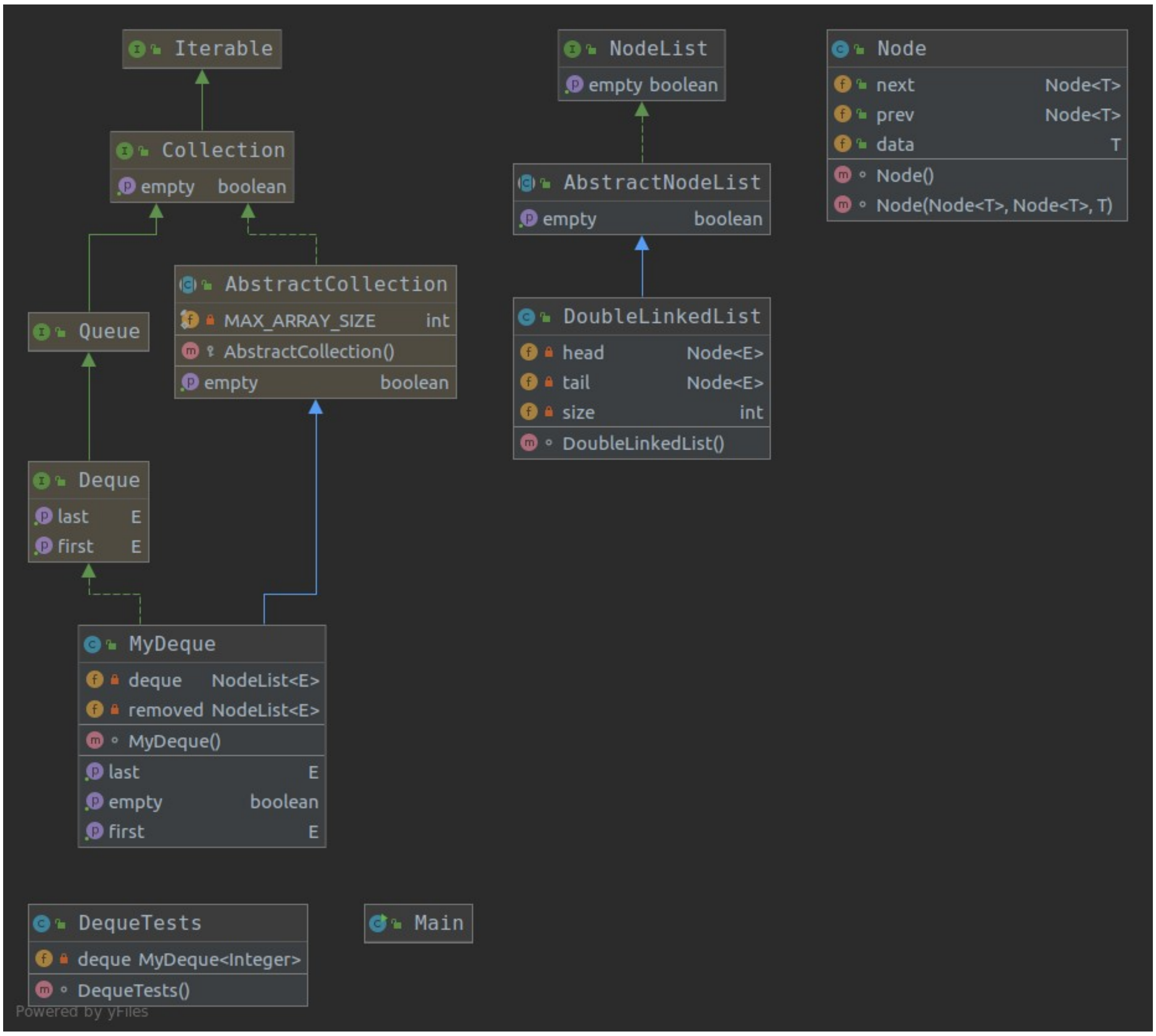
GTU Department of Computer Engineering
CSE 222/505 - Spring 2020
Homework 4 Report

Buğra Eren Yılmaz
1801042669

1. Class Diagrams for Q2



2. Problem Solution Approach for Q2

For the solution of this problem, I needed a way to make a DoublyLinkedList that can utilize existing/already created nodes for it’s structure.

I started with creating that, what I called a NodeList. It is basically a List interface but with extra ability of utilizing externally created Nodes for it’s structure.

Then I made the AbstractNodeList class which just implements bare minimum things like isEmpty, or some things that can be implemented without the user implementation.

Add method is a great example of this. Add method is just a method that adds element to list but it creates a new node and add it to list.

On the other hand addNode method gets an externally created Node as a parameter and adds that to list. Thus, add method can be implemented using addNode method. These types of methods implemented in AbstractNodeList class.

Also I handled the iterator for NodeList here, creating a custom NodeListIterator class.

Then I made the DoubleLinkedList class extending from AbstractNodeList class, which just implements how does the nodes get linked and removed.

At that point I had the right tool for this job which is a linkedlist class that can utilize externally created Nodes for it’s structure.

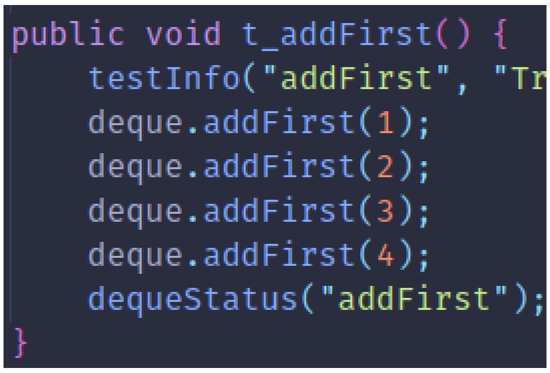
Then I started the custom Deque class, MyDeque. It utilizes two of these custom DoubleLinkedList classes for its structure. Basically it holds the removed Nodes inside one list and actual deque in another. That way we can reuse removed nodes when adding to deque, instead of creating new ones.

```
deque.addNode(removed.removeNode())
```

is a perfectly valid solution for that. It removes last node of the removed list and uses that node to addNode to actual deque list. This way we are re-using the removed nodes and preventing garbage collector to slow down our system.

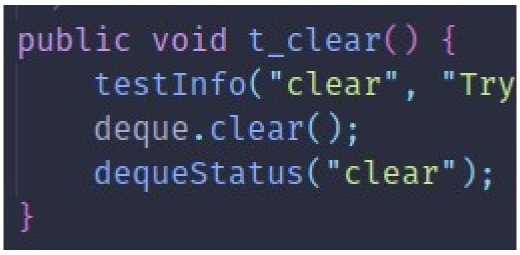
3. Test Cases for Q2

1. addFirst tests



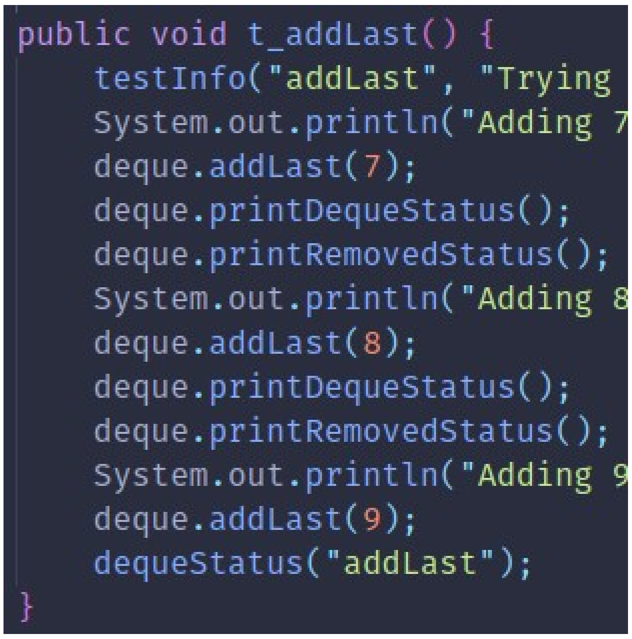
```
public void t_addFirst() {
    testInfo("addFirst", "Tr
    deque.addFirst(1);
    deque.addFirst(2);
    deque.addFirst(3);
    deque.addFirst(4);
    dequeStatus("addFirst");
}
```

2. demo with clear tests, that show removed elements store



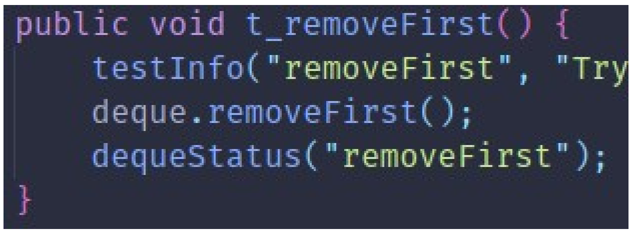
```
public void t_clear() {
    testInfo("clear", "Try
    deque.clear();
    dequeStatus("clear");
}
```

3. addLast test, that shows re-use of removed elements



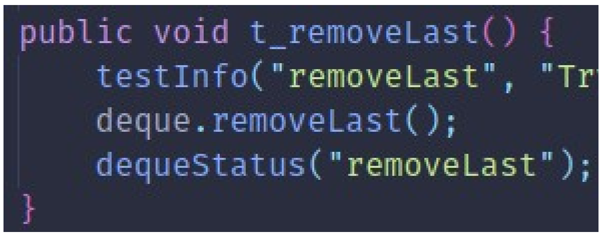
```
public void t_addLast() {
    testInfo("addLast", "Trying
    System.out.println("Adding 7
    deque.addLast(7);
    deque.printDequeStatus();
    deque.printRemovedStatus();
    System.out.println("Adding 8
    deque.addLast(8);
    deque.printDequeStatus();
    deque.printRemovedStatus();
    System.out.println("Adding 9
    deque.addLast(9);
    dequeStatus("addLast");
}
```

4. removeFirst tests



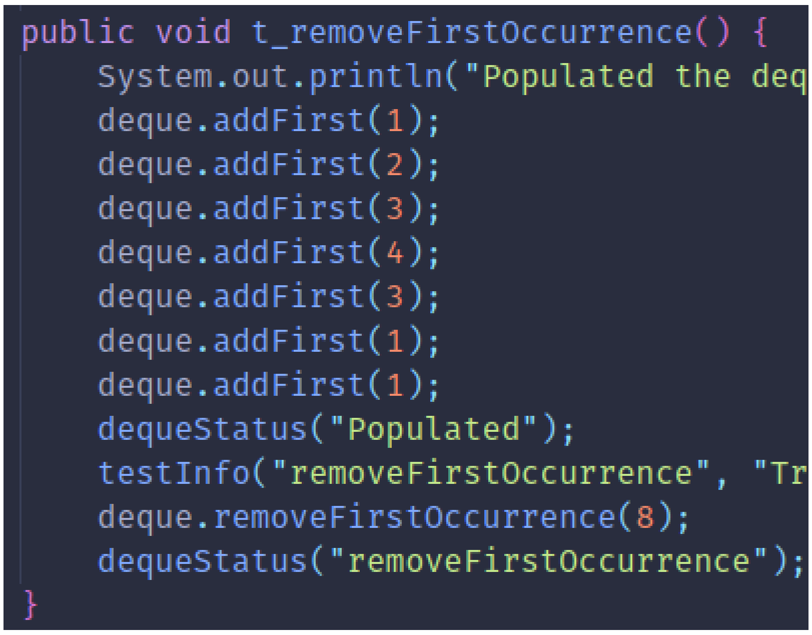
```
public void t_removeFirst() {
    testInfo("removeFirst", "Try
    deque.removeFirst();
    dequeStatus("removeFirst");
}
```

5. removeLast tests



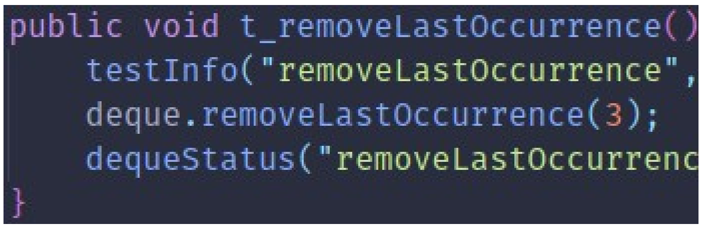
```
public void t_removeLast() {
    testInfo("removeLast", "Tr
    deque.removeLast();
    dequeStatus("removeLast");
}
```

6. removeFirstOccurrence tests



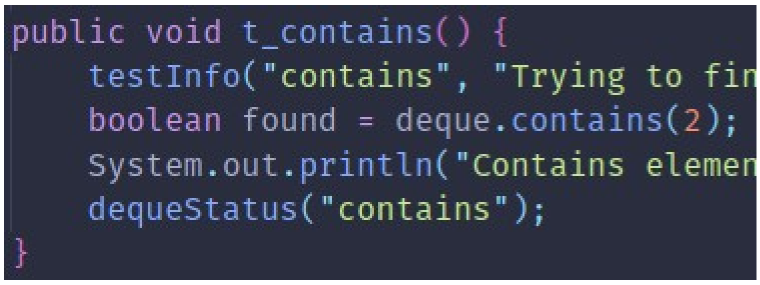
```
public void t_removeFirstOccurrence() {
    System.out.println("Populated the deq
    deque.addFirst(1);
    deque.addFirst(2);
    deque.addFirst(3);
    deque.addFirst(4);
    deque.addFirst(3);
    deque.addFirst(1);
    deque.addFirst(1);
    dequeStatus("Populated");
    testInfo("removeFirstOccurrence", "Tr
    deque.removeFirstOccurrence(8);
    dequeStatus("removeFirstOccurrence");
}
```

7. removeLastOccurrence tests



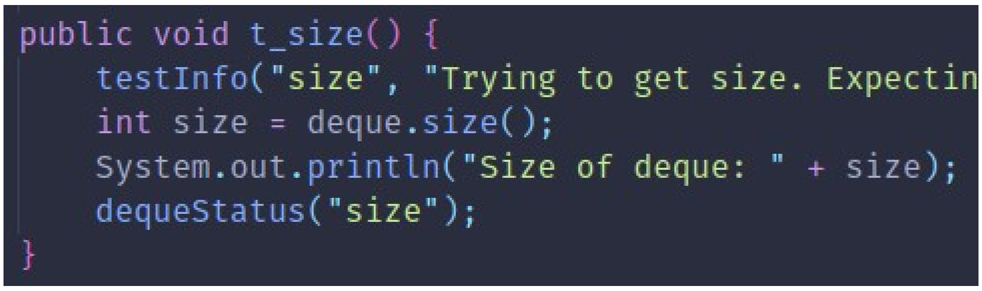
```
public void t_removeLastOccurrence()
    testInfo("removeLastOccurrence",
    deque.removeLastOccurrence(3);
    dequeStatus("removeLastOccurrenc
}
```

8. contains tests



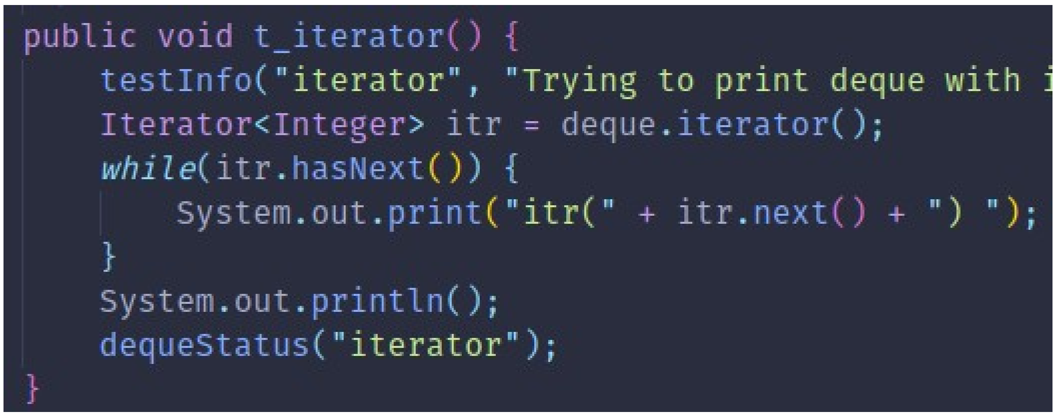
```
public void t_contains() {
    testInfo("contains", "Trying to fin
    boolean found = deque.contains(2);
    System.out.println("Contains elemen
    dequeStatus("contains");
}
```

9. size tests



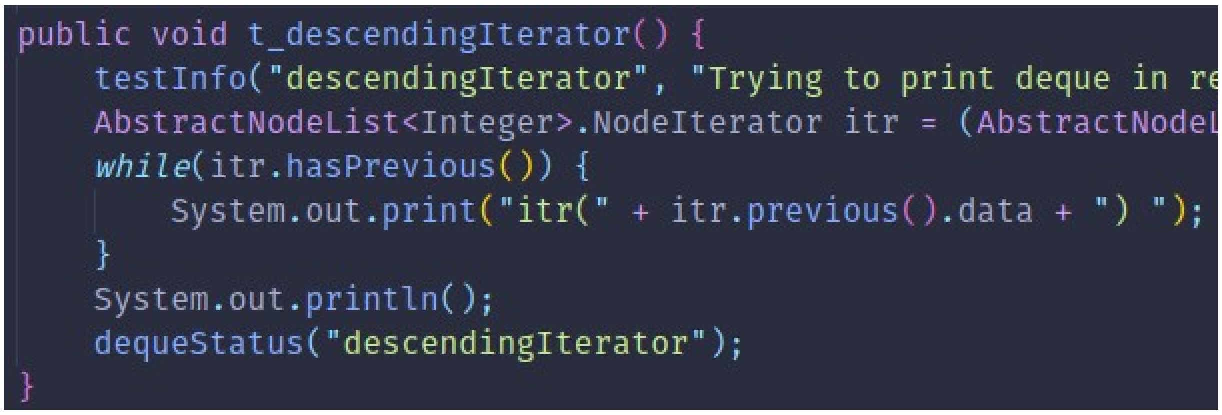
```
public void t_size() {
    testInfo("size", "Trying to get size. Expectin
    int size = deque.size();
    System.out.println("Size of deque: " + size);
    dequeStatus("size");
}
```

10. iterator tests



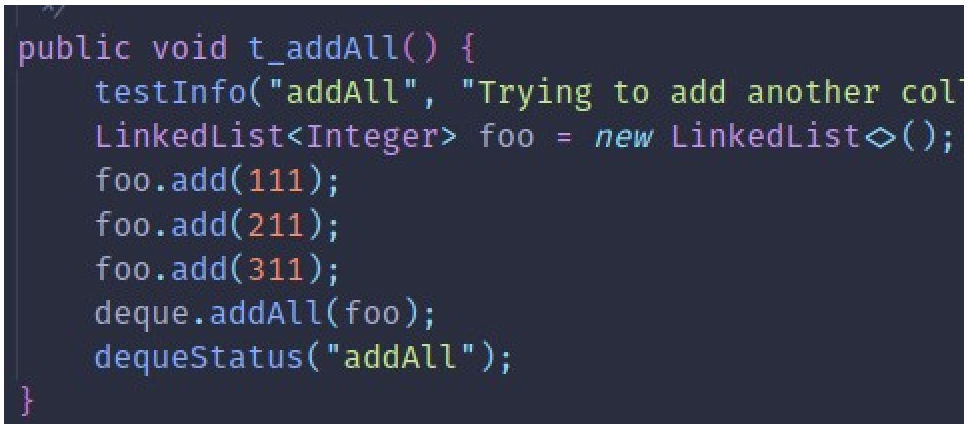
```
public void t_iterator() {
    testInfo("iterator", "Trying to print deque with i
    Iterator<Integer> itr = deque.iterator();
    while(itr.hasNext()) {
        System.out.print("itr(" + itr.next() + ") ");
    }
    System.out.println();
    dequeStatus("iterator");
}
```

11. descendingIterator tests



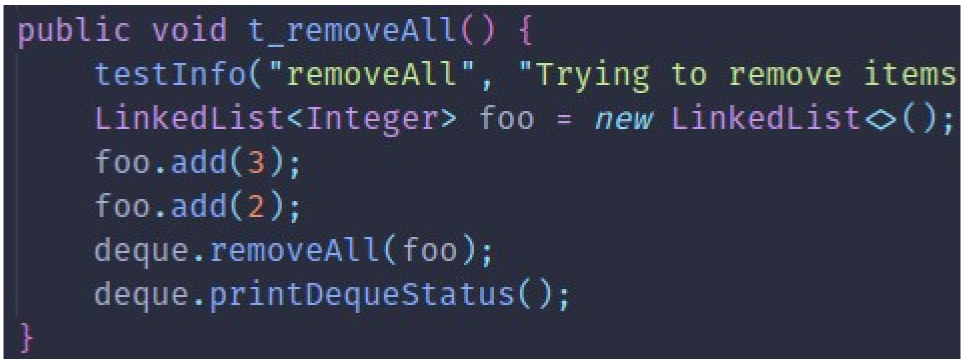
```
public void t_descendingIterator() {
    testInfo("descendingIterator", "Trying to print deque in re
    AbstractNodeList<Integer>.NodeIterator itr = (AbstractNodeL
    while(itr.hasPrevious()) {
        System.out.print("itr(" + itr.previous().data + ") ");
    }
    System.out.println();
    dequeStatus("descendingIterator");
}
```

12. Collection addAll tests showing that it is a valid AbstractCollection



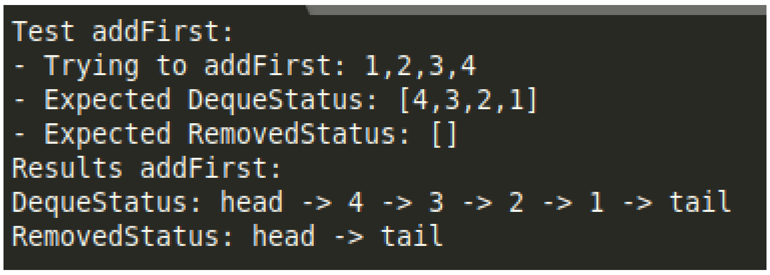
```
public void t_addAll() {
    testInfo("addAll", "Trying to add another col
    LinkedList<Integer> foo = new LinkedList<>();
    foo.add(111);
    foo.add(211);
    foo.add(311);
    deque.addAll(foo);
    dequeStatus("addAll");
}
```

13. Collection removeAll tests showing that it is a valid AbstractCollection

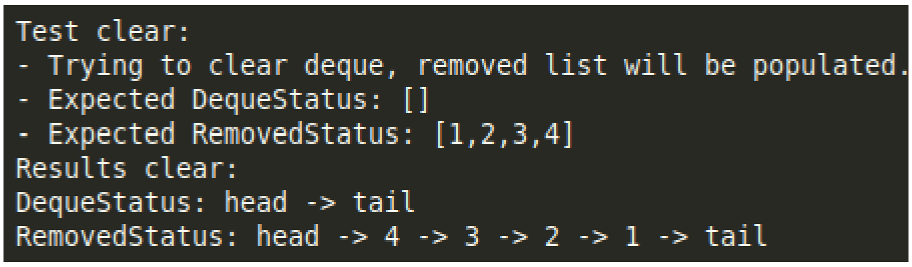


```
public void t_removeAll() {
    testInfo("removeAll", "Trying to remove items
    LinkedList<Integer> foo = new LinkedList<>();
    foo.add(3);
    foo.add(2);
    deque.removeAll(foo);
    deque.printDequeStatus();
}
```

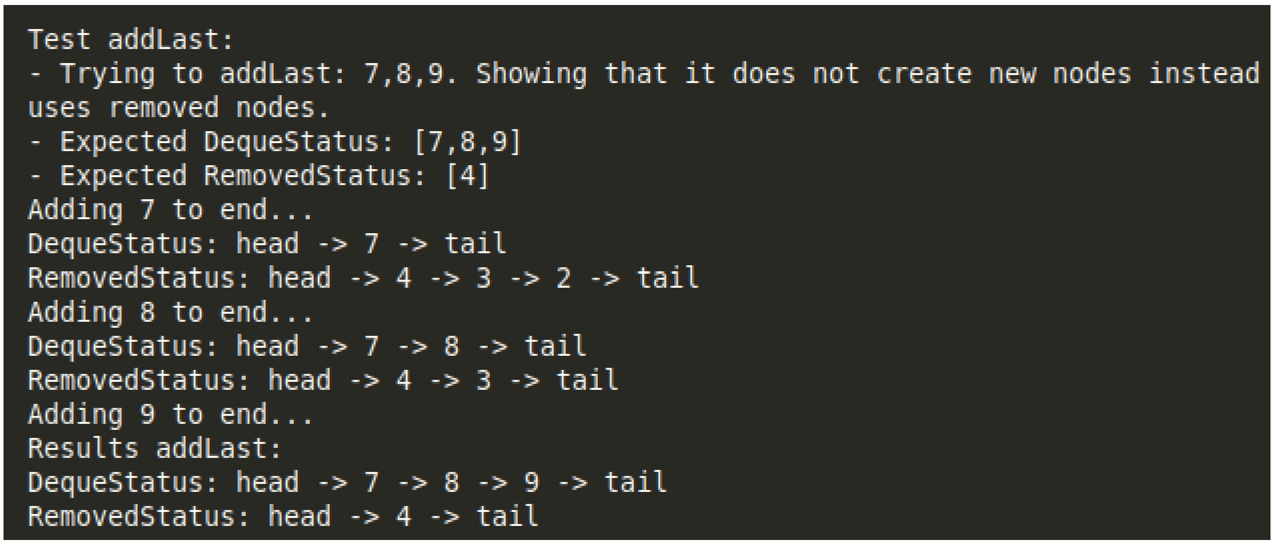
4. Running results



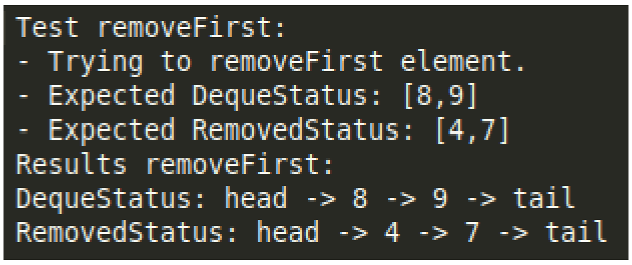
```
Test addFirst:
- Trying to addFirst: 1,2,3,4
- Expected DequeStatus: [4,3,2,1]
- Expected RemovedStatus: []
Results addFirst:
DequeStatus: head -> 4 -> 3 -> 2 -> 1 -> tail
RemovedStatus: head -> tail
```



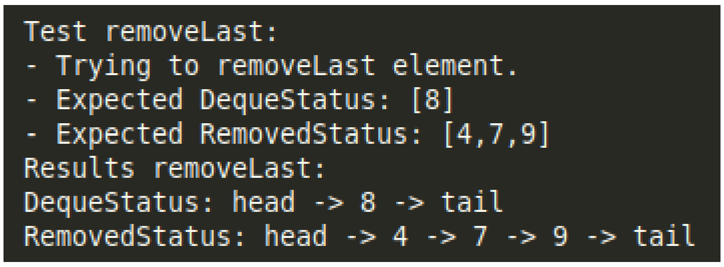
```
Test clear:
- Trying to clear deque, removed list will be populated.
- Expected DequeStatus: []
- Expected RemovedStatus: [1,2,3,4]
Results clear:
DequeStatus: head -> tail
RemovedStatus: head -> 4 -> 3 -> 2 -> 1 -> tail
```



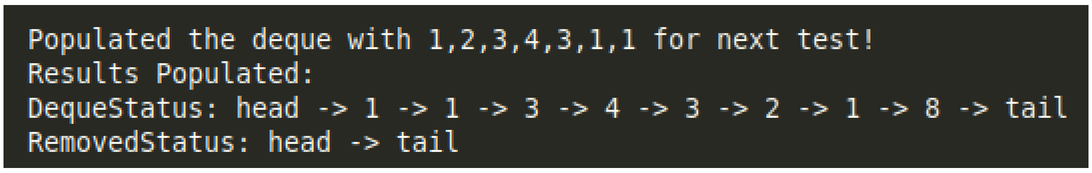
```
Test addLast:
- Trying to addLast: 7,8,9. Showing that it does not create new nodes instead
uses removed nodes.
- Expected DequeStatus: [7,8,9]
- Expected RemovedStatus: [4]
Adding 7 to end...
DequeStatus: head -> 7 -> tail
RemovedStatus: head -> 4 -> 3 -> 2 -> tail
Adding 8 to end...
DequeStatus: head -> 7 -> 8 -> tail
RemovedStatus: head -> 4 -> 3 -> tail
Adding 9 to end...
Results addLast:
DequeStatus: head -> 7 -> 8 -> 9 -> tail
RemovedStatus: head -> 4 -> tail
```



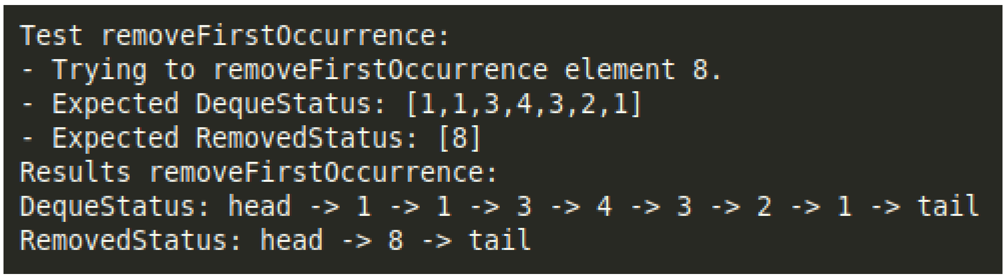
```
Test removeFirst:
- Trying to removeFirst element.
- Expected DequeStatus: [8,9]
- Expected RemovedStatus: [4,7]
Results removeFirst:
DequeStatus: head -> 8 -> 9 -> tail
RemovedStatus: head -> 4 -> 7 -> tail
```



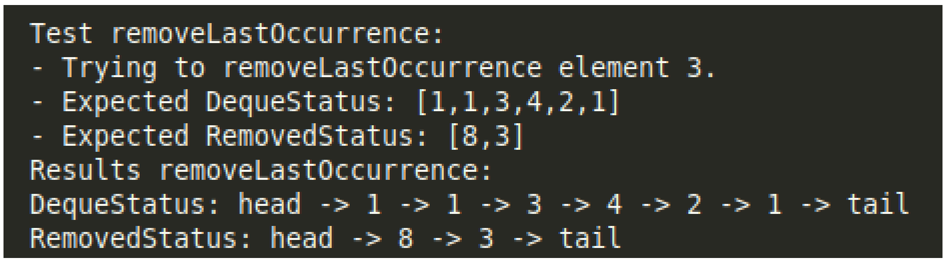
```
Test removeLast:
- Trying to removeLast element.
- Expected DequeStatus: [8]
- Expected RemovedStatus: [4,7,9]
Results removeLast:
DequeStatus: head -> 8 -> tail
RemovedStatus: head -> 4 -> 7 -> 9 -> tail
```



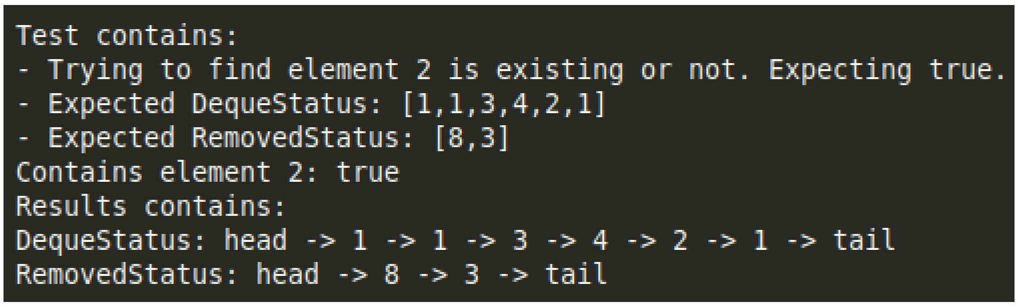
```
Populated the deque with 1,2,3,4,3,1,1 for next test!
Results Populated:
DequeStatus: head -> 1 -> 1 -> 3 -> 4 -> 3 -> 2 -> 1 -> 8 -> tail
RemovedStatus: head -> tail
```



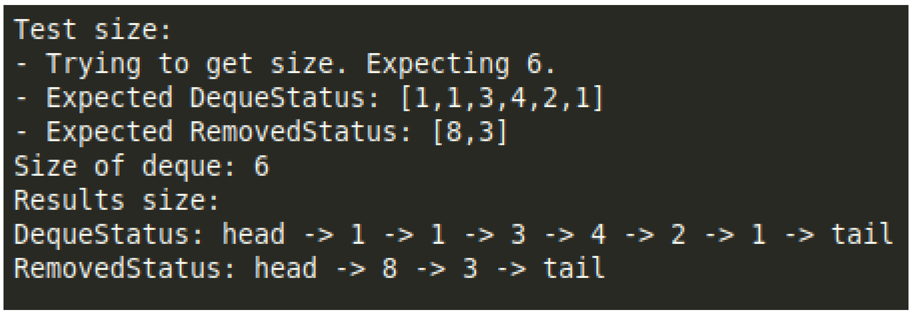
```
Test removeFirstOccurrence:
- Trying to removeFirstOccurrence element 8.
- Expected DequeStatus: [1,1,3,4,3,2,1]
- Expected RemovedStatus: [8]
Results removeFirstOccurrence:
DequeStatus: head -> 1 -> 1 -> 3 -> 4 -> 3 -> 2 -> 1 -> tail
RemovedStatus: head -> 8 -> tail
```



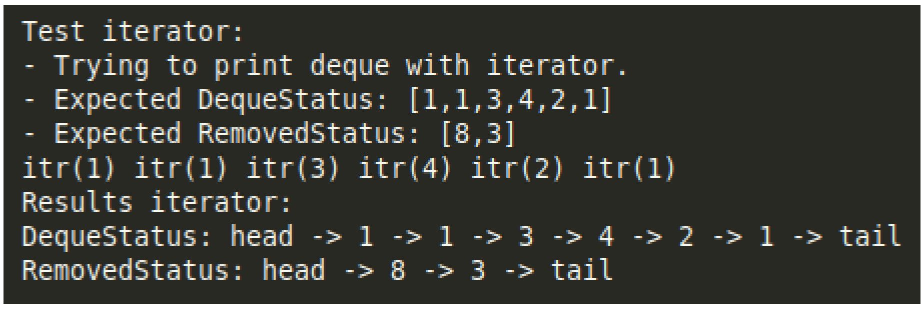
```
Test removeLastOccurrence:
- Trying to removeLastOccurrence element 3.
- Expected DequeStatus: [1,1,3,4,2,1]
- Expected RemovedStatus: [8,3]
Results removeLastOccurrence:
DequeStatus: head -> 1 -> 1 -> 3 -> 4 -> 2 -> 1 -> tail
RemovedStatus: head -> 8 -> 3 -> tail
```



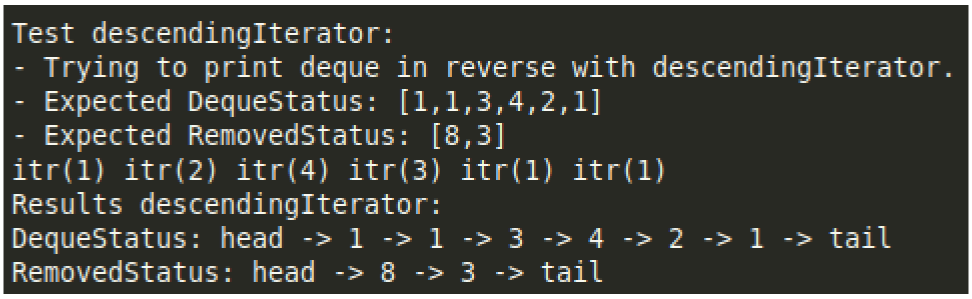
```
Test contains:
- Trying to find element 2 is existing or not. Expecting true.
- Expected DequeStatus: [1,1,3,4,2,1]
- Expected RemovedStatus: [8,3]
Contains element 2: true
Results contains:
DequeStatus: head -> 1 -> 1 -> 3 -> 4 -> 2 -> 1 -> tail
RemovedStatus: head -> 8 -> 3 -> tail
```



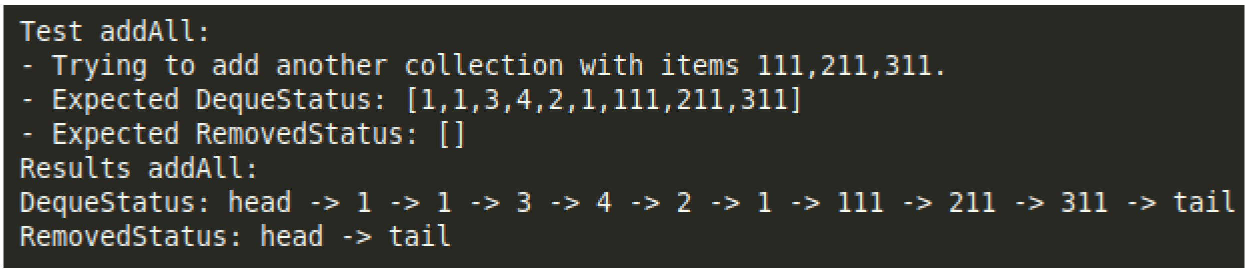
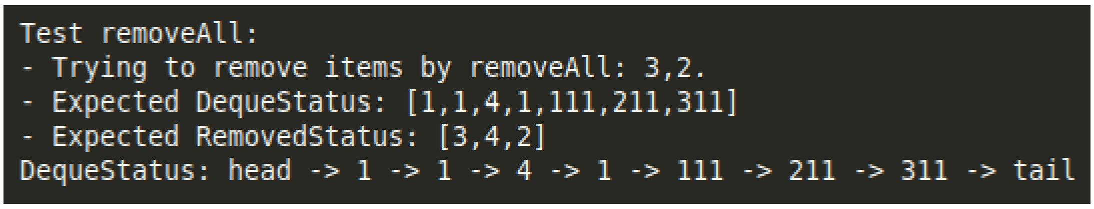
```
Test size:
- Trying to get size. Expecting 6.
- Expected DequeStatus: [1,1,3,4,2,1]
- Expected RemovedStatus: [8,3]
Size of deque: 6
Results size:
DequeStatus: head -> 1 -> 1 -> 3 -> 4 -> 2 -> 1 -> tail
RemovedStatus: head -> 8 -> 3 -> tail
```



```
Test iterator:
- Trying to print deque with iterator.
- Expected DequeStatus: [1,1,3,4,2,1]
- Expected RemovedStatus: [8,3]
itr(1) itr(1) itr(3) itr(4) itr(2) itr(1)
Results iterator:
DequeStatus: head -> 1 -> 1 -> 3 -> 4 -> 2 -> 1 -> tail
RemovedStatus: head -> 8 -> 3 -> tail
```



```
Test descendingIterator:
- Trying to print deque in reverse with descendingIterator.
- Expected DequeStatus: [1,1,3,4,2,1]
- Expected RemovedStatus: [8,3]
itr(1) itr(2) itr(4) itr(3) itr(1) itr(1)
Results descendingIterator:
DequeStatus: head -> 1 -> 1 -> 3 -> 4 -> 2 -> 1 -> tail
RemovedStatus: head -> 8 -> 3 -> tail
```


```
Test addAll:
- Trying to add another collection with items 111,211,311.
- Expected DequeStatus: [1,1,3,4,2,1,111,211,311]
- Expected RemovedStatus: []
Results addAll:
DequeStatus: head -> 1 -> 1 -> 3 -> 4 -> 2 -> 1 -> 111 -> 211 -> 311 -> tail
RemovedStatus: head -> tail
```

```
Test removeAll:
- Trying to remove items by removeAll: 3,2.
- Expected DequeStatus: [1,1,4,1,111,211,311]
- Expected RemovedStatus: [3,4,2]
DequeStatus: head -> 1 -> 1 -> 4 -> 1 -> 111 -> 211 -> 311 -> tail
```