Cameron Moberg
CS260-02
September 9th, 2016

CS 260: Queue implemented by a LinkedList

The assignment was to create a Queue using a provided a LinkedList class. Since a queue can very easily be made using a LinkedList the first thing I did was think about how the methods relate to each other, for example, insertAtEnd is basically the same thing as adding to the queue. However some things had to be changed, for example for DeQueue, it is like deleting the first position in the list, however there isn't the right code in the LinkedList class, and since it would be too specific to change the LinkedList I decided to add some logic to the Queue class that would output that you can't dequeue an empty queue. Although I tried not to, I did have to add 2 getter methods to the CustomLinkedList class, "getStart" and "getEnd" since they were private variables, the only way I could get them so I could display in the CustomQueue class was by adding those methods, since it would be bad practice to change the private variables to public. I also added a getSize() method to Queue rather than just keeping the 5 methods (peek, enqueue, dequeue, isFull, isEmpty). The isFull method is practically useless because a LinkedList can virtually never be full, so I've made it so it always returns false.

Outputs: Read from left to right.

```
/usr/lib/jvm/java-8-jdk/bin/java ...
Custom Queue

Queue Options

1. EnQueue
2. DeQueue
3. Check Empty
4. Display
5. Peek
6. Get Size
7. Exit
1
Enter integer element to insert
10
Queue => 10
Front Pointer => 10
Rear Pointer  => 10
```

```
Queue Options

1. EnQueue
2. DeQueue
3. Check Empty
4. Display
5. Peek
6. Get Size
7. Exit

1
Enter integer element to insert
20
Queue => 10,20
Front Pointer => 10
Rear Pointer  => 20
```

```
Queue Options

1. EnQueue
2. DeQueue
3. Check Empty
4. Display
5. Peek
6. Get Size
7. Exit
1
Enter integer element to insert
30
Queue => 10,20,30
Front Pointer => 10
Rear Pointer  => 30
```

```
Queue Options

1. EnQueue
2. DeQueue
3. Check Empty
4. Display
5. Peek
6. Get Size
7. Exit
2
Queue => 20,30
Front Pointer => 20
Rear Pointer  => 30
```

```
Queue Options

1. EnQueue
2. DeQueue
3. Check Empty
4. Display
5. Peek
6. Get Size
7. Exit
2
Queue => 30
Front Pointer => 30
Rear Pointer  => 30
```

```
Queue Options

1. EnQueue
2. DeQueue
3. Check Empty
4. Display
5. Peek
6. Get Size
7. Exit
2
Queue is empty.
```

```
Queue Options

1. EnQueue
2. DeQueue
3. Check Empty
4. Display
5. Peek
6. Get Size
7. Exit
2

Cannot DeQueue an empty Queue.
Queue is empty.
```

```
 1:
 2: public class Node {
 3:     private int data;
 4:     private Node next;
 5:
 6:     public Node() {
 7:         next = null;
 8:         data = 0;
 9:     }
10:
11:     public Node(int d, Node n) {
12:         data = d;
13:         next = n;
14:     }
15:
16:     public void setNext(Node n) {
17:         next = n;
18:     }
19:
20:     public void setData(int d) {
21:         data = d;
22:     }
23:
24:     public Node getNext() {
25:         return next;
26:     }
27:
28:     public int getData() {
29:         return data;
30:     }
31: }
```

```
 1: import java.util.Scanner;
 2:
 3: public class QueueDriver {
 4:     public static void main(String[] args) {
 5:         Scanner scan = new Scanner(System.in);
 6:         // Creating object of class queue
 7:         CustomQueue queue = new CustomQueue();
 8:         System.out.println("Custom Queue");
 9:
10: //              Perform list operations
11:         while (true) {
12:             System.out.println("\nQueue Options\n");
13:             System.out.println("1. EnQueue");
14:             System.out.println("2. DeQueue");
15:             System.out.println("3. Check Empty");
16:             System.out.println("4. Display");
17:             System.out.println("5. Peek");
18:             System.out.println("6. Get Size");
19:             System.out.println("7. Exit");
20:             int choice = scan.nextInt();
21:             switch (choice) {
22:                 case 1: //enqueue
23:                     System.out.println("Enter integer element to insert");
24:                     queue.enQueue(scan.nextInt());
25:                     break;
26:                 case 2: //dequeue
27:                     queue.deQueue();
28:                     break;
29:                 case 3: //check empty
30:                     if (queue.isEmpty())
31:                         System.out.println("Queue is empty.");
32:                     else
33:                         System.out.println("Queue is not empty");
34:                     break;
35:                 case 4: //display
36:                     queue.display();
37:                     break;
38:                 case 5: //peek
39:                     queue.peek();
40:                     break;
41:                 case 6: //get size
42:                     System.out.println("Size = " + queue.getSize() + " \n");
43:                     break;
44:                 case 7: //terminate
45:                     scan.close();
46:                     System.exit(0);
47:                 default:
48:                     System.out.println("Wrong Entry \n ");
49:                     break;
50:             }
51:             /* Display List */
52:             queue.display();
53:         }
54:     }
55: }
```

```java
/**
 * Created by bold on 9/1/16.
 */
public class CustomLinkedList {
    private Node start;
    private Node end;
    private int size;

    public CustomLinkedList() {
        start = null;
        end = null;
        size = 0;
    }

    public boolean isEmpty() {
        if (start == null)
            return true;
        else
            return false;
    }

    public int getSize() {
        return size;
    }

    public void insertAtStart(int val) {
        Node node = new Node(val, null);
        size++;
        if (start == null) {
            start = node;
            end = start;
        } else {
            node.setNext(start);
            start = node;
        }
    }

    public void insertAtEnd(int val) {
        Node node = new Node(val, null);
        size++;
        if (start == null) {
            start = node;
            end = start;
        } else {
            end.setNext(node);
            end = node;
        }
    }

    public void insertAtPos(int val, int pos) {
        if (pos == 1 || pos >= size) {
            System.err.println("Invalid Position\n");
            return;
        }

        Node node = new Node(val, null);
        Node aNode = start;
        pos = pos - 1;
        for (int i = 1; i < size; i++) {
            if (i == pos) {
                Node tmp = aNode.getNext();
                aNode.setNext(node);
                node.setNext(tmp);
                break;
            }
            aNode = aNode.getNext();
        }
        size++;
    }

    public void deleteAtPos(int pos) {
        if (pos == 1) {
            start = start.getNext();
            size--;
            return;
        }
        if (pos == size) {
            Node s = start;
            Node t = start;
            while (s != end) {
                t = s;
                s = s.getNext();
            }
            end = t;
            end.setNext(null);
            size--;
            return;
        }
        Node aNode = start;
        pos = pos - 1;
        for (int i = 1; i < size - 1; i++) {
            if (i == pos) {
                Node tmp = aNode.getNext();
                tmp = tmp.getNext();
                aNode.setNext(tmp);
                break;
            }
            aNode = aNode.getNext();
        }
        size--;
    }

    public void display() {
        System.out.print("Linked List = ");
        if (size == 0) {
            System.out.println("empty");
            return;
        }
        if (start.getNext() == null) {
            System.out.println(start.getData());
            return;
        }
        Node aNode = start;
        System.out.print(start.getData() + "->");
        aNode = start.getNext();
        while (aNode.getNext() != null) {
            System.out.print(aNode.getData() + "->");
            aNode = aNode.getNext();
        }
        System.out.print(aNode.getData() + "\n");
    }

    public void updateAtPosition(int pos, int data) {
        if (pos == 1) {
            start.getNext().setData(data);
        }
        if (pos == size) {
            Node s = start;
            while (s.getNext() != null) {
                s = s.getNext();
            }
            s.setData(data);
```

```
133:             return;
134:         }
135:         Node aNode = start;
136:         pos = pos - 1;
137:         for (int i = 1; i < size - 1; i++) {
138:             if (i == pos) {
139:                 aNode.setData(data);
140:             }
141:             aNode = aNode.getNext();
142:         }
143:         return;
144:     }
145:
146:     public int searchByValue(int val) {
147:         if (size == 0) {
148:             System.err.println("List is empty");
149:             //returns impossible vindex.
150:             return -1;
151:         }
152:         Node aNode = start;
153:         for (int i = 1; i < size - 1; i++) {
154:             if (val == aNode.getData()) {
155:                 return i;
156:             }
157:             aNode = aNode.getNext();
158:         }
159:         //if not found, returns impossible index.
160:         return -1;
161:     }
162:
163:     public Node getStart() {
164:         return start;
165:     }
166:
167:     public Node getEnd() {
168:         return end;
169:     }
170:
171: }
```

```
 1: /**
 2:  * Created by bold on 9/6/16.
 3:  */
 4: public class CustomQueue {
 5:     CustomLinkedList list = new CustomLinkedList();
 6:
 7:     public void enQueue(int val) {
 8:         list.insertAtEnd(val);
 9:     }
10:
11:     public void deQueue() {
12:         if (list.getSize() < 1) {
13:             System.out.println("\nCannot DeQueue an empty Queue.");
14:             return;
15:         }
16:         list.deleteAtPos(1);
17:     }
18:
19:     public void peek() {
20:         if (!isEmpty()) {
21:             System.out.println(list.getStart().getData());
22:             return;
23:         }
24:         System.out.println("\nCannot peek on empty Queue.");
25:     }
26:
27:     public Boolean isFull() {
28:         //LinkedList is only capped by physical memory
29:         return false;
30:     }
31:
32:     public Boolean isEmpty() {
33:         return list.getSize() == 0;
34:     }
35:
36:     public void display() {
37:         if (isEmpty()) {
38:             System.out.println("Queue is empty.");
39:             return;
40:         }
41:         Node temp = list.getStart();
42:
43:         System.out.print("Queue => ");
44:         while (temp.getNext() != null) {
45:             System.out.print(temp.getData() + ",");
46:             temp = temp.getNext();
47:         }
48:         System.out.println(temp.getData());
49:
50:         System.out.println("Front Pointer => " + list.getStart().getData());
51:         System.out.println("Rear Pointer  => " + list.getEnd().getData() + "\n");
52:     }
53:
54:     public int getSize() {
55:         return list.getSize();
56:     }
57: }
```