```
// Lab 11: Deque
// Lecture notes were very helpful for this lab.
// Andrea Smith
// CSCI 1913

class Deque<Base>
{
  private class Node
  {
    private Base object;
    private Node right;
    private Node left;
    private Node (Base object, Node right, Node left)
    {
      this.object = object;
      this.right = right;
      this.left = left;
    }
  }
  // Head node points to head of circular linked list
  private Node head;

  public Deque()
  {
    head = new Node(null, null, null);
    head.right = head;
    head.left = head;
  }

  public void enqueueFront(Base object)
  {
    Node whereR = head;
    Node newNode = new Node(object, whereR.right, whereR);
    whereR.right.left = newNode; // From lecture
    whereR.right = newNode;
  }

  public void enqueueRear(Base object)
  {
    Node whereL = head;
    Node newNode = new Node(object, whereL, whereL.left);
    whereL.left.right = newNode;
    whereL.left = newNode;
  }

  public Base dequeueFront()
  {
```

```
    if (isEmpty())
    {
      throw new IllegalStateException("The Deque is empty.");
    }
    else
    {
      Node where = head.right; // Get the node at the front of the
      Deque
      where.right.left = head;
      head.right = where.right; // Delete the node at the front
      return where.object; // Return the object

    }
  }

  public Base dequeueRear()
  {
    if (isEmpty())
    {
      throw new IllegalStateException("The Deque is empty.");
    }
    else
    {
      Node where = head.left; // Get the node at the rear of the
      Deque
      where.left.right = head;
      head.left = where.left; // Delete the node at the rear
      return where.object; // Return the object
    }
  }

  public boolean isEmpty()
  {
    return head.right == head && head.left == head;
  }

}

//  OBSERVATION DEQUE. Test the class DEQUE. 40 points total.

class ObservationDeque
{

//  MAIN. Test the DEQUE on various example arguments.

  public static void main(String [] args)
  {
```

```
92    {
93      Deque<String> deque = new Deque<String>();
94
95      System.out.println(deque.isEmpty());       //
        true                2 points.
96
97      try
98      {
99        System.out.println(deque.dequeueFront());
100     }
101     catch (IllegalStateException ignore)
102     {
103       System.out.println("No dequeueFront.");  //  No
          dequeueFront.   2 points.
104     }
105
106     try
107     {
108       System.out.println(deque.dequeueRear());
109     }
110     catch (IllegalStateException ignore)
111     {
112       System.out.println("No dequeueRear.");   //  No
          dequeueRear.    2 points.
113     }
114
115 //  Enqueueing to the rear and dequeueing from the rear makes the
    DEQUE act
116 //  like a stack.
117
118     deque.enqueueRear("A");
119     deque.enqueueRear("B");
120     deque.enqueueRear("C");
121
122     System.out.println(deque.isEmpty());       //
        false               2 points.
123
124     System.out.println(deque.dequeueRear());   //
        C                   2 points.
125     System.out.println(deque.dequeueRear());   //
        B                   2 points.
126     System.out.println(deque.dequeueRear());   //
        A                   2 points.
127
128     System.out.println(deque.isEmpty());       //
        true                2 points.
129
```

```
130 //  Enqueueing to the rear and dequeueing from the front makes the
  • DEQUE act
131 //  like a queue.
132
133     deque.enqueueRear("A");
134     deque.enqueueRear("B");
135     deque.enqueueRear("C");
136
137     System.out.println(deque.dequeueFront());  //
  •     A                  2 points.
138     System.out.println(deque.dequeueFront());  //
  •     B                  2 points.
139     System.out.println(deque.dequeueFront());  //
  •     C                  2 points.
140
141     System.out.println(deque.isEmpty());       //
  •     true               2 points.
142
143 //  Enqueueing to the front and dequeueing from the front makes the
  • DEQUE act
144 //  like a stack.
145
146     deque.enqueueFront("A");
147     deque.enqueueFront("B");
148     deque.enqueueFront("C");
149
150     System.out.println(deque.dequeueFront());  //
  •     C                  2 points.
151     System.out.println(deque.dequeueFront());  //
  •     B                  2 points.
152     System.out.println(deque.dequeueFront());  //
  •     A                  2 points.
153
154     System.out.println(deque.isEmpty());       //
  •     true               2 points.
155
156 //  Enqueueing to the front and dequeueing from the rear makes the
  • DEQUE act
157 //  like a queue.
158
159     deque.enqueueFront("A");
160     deque.enqueueFront("B");
161     deque.enqueueFront("C");
162
163     System.out.println(deque.dequeueRear());   //
  •     A                  2 points.
164     System.out.println(deque.dequeueRear());   //
```

```
  •       B                   2 points.
165       System.out.println(deque.dequeueRear());   //
  •       C                   2 points.
166
167       System.out.println(deque.isEmpty());       //
  •       true                2 points.
168     }
169   }
170
```