```java
// Lab 9: ArrayQueue
// I did not like this lab.
// Andrea Smith
// CSCI 1913

import java.util.Iterator;

//  ARRAY QUEUE. A fixed length queue implemented as a circular
array.

class ArrayQueue<Base>
{
  private int front;      //  Index of front object in BASES.
  private int rear;       //  Index of rear object in BASES.
  private Base [] bases;  //  The BASEs in the queue.

//  Constructor. Make a new empty queue that can hold SIZE - 1
elements.

  public ArrayQueue(int size)
  {
    if (size >= 1)
    {
      front = 0;
      rear = 0;
      bases = (Base []) new Object[size];
    }
    else
    {
      throw new IllegalArgumentException("Size must be at least 1.");
    }
  }

  public Iterator<Base> iterator()
  {
    return new ArrayQueueIterator(front, rear, bases);
  }


//  DEQUEUE. Remove a BASE from the front of the queue and return it.

  public Base dequeue()
  {
    if (isEmpty())
    {
      throw new IllegalStateException("Queue is empty.");
    }
```

```
46      else
47      {
48        front = (front + 1) % bases.length;
49        Base temp = bases[front];
50        bases[front] = null;
51        return temp;
52      }
53    }
54
55  //  ENQUEUE. Add BASE to the rear of the queue.
56
57    public void enqueue(Base base)
58    {
59      int nextRear = (rear + 1) % bases.length;
60      if (front == nextRear)
61      {
62        throw new IllegalStateException("Queue is full.");
63      }
64      else
65      {
66        rear = nextRear;
67        bases[rear] = base;
68      }
69    }
70
71  //  IS EMPTY. Test if the queue is empty.
72
73    public boolean isEmpty()
74    {
75      return front == rear;
76    }
77
78  //  IS FULL. Test if the queue can hold no more elements.
79
80    public boolean isFull()
81    {
82      return front == (rear + 1) % bases.length;
83    }
84
85  // ITERATOR
86
87    private class ArrayQueueIterator implements Iterator<Base>
88    {
89        private int front;
90        private int rear;
91        private Base [] bases;
```

```
92
93        private ArrayQueueIterator(int front, int rear, Base [] bases)
94        {
95          this.front = front;
96          this.rear = rear;
97          this.bases = bases;
98        }
99
100       public boolean hasNext()
101       {
102         return front != rear;
103       }
104
105       public Base next()
106       {
107         if (!hasNext())
108         {
109           throw new IllegalStateException("No more elements.");
110         }
111         else
112         {
113           front = (front + 1) % bases.length;
114           Base temp = bases[front];
115
116           return temp;
117         }
118       }
119
120       public void remove()
121       {}
122     }
123
124 }
125
126
127 //  QUEUETERATOR. Test ARRAY QUEUE's iterator. It's worth 20 points.
128
129 class Queueterator
130 {
131
132 //  MAIN. Start execution here.
133
134   public static void main(String [] args)
135   {
136
137 //  Make an ARRAY QUEUE and enqueue some STRINGs. It can hold at
    most three.
```

```
138
139     ArrayQueue<String> queue = new ArrayQueue<String>(4);
140
141     queue.enqueue("A");
142     queue.enqueue("B");
143     queue.enqueue("C");
144
145 //  Make a FIRST iterator for QUEUE and use it to visit QUEUE's
    • elements.
146
147     Iterator<String> first = queue.iterator();
148     while (first.hasNext())
149     {
150       System.out.println(first.next());  //  A B C one per line
    •       5 pts.
151     }
152
153 //  Make sure FIRST hasn't changed QUEUE.
154
155     System.out.println(queue.isEmpty());   //  false
    •     1 pt.
156     System.out.println(queue.dequeue());   //  A
    •     1 pt.
157     System.out.println(queue.dequeue());   //  B
    •     1 pt.
158     System.out.println(queue.dequeue());   //  C
    •     1 pt.
159     System.out.println(queue.isEmpty());   //  true
    •     1 pt.
160
161 //  Let's enqueue more three more things to QUEUE.
162
163     queue.enqueue("X");
164     queue.enqueue("Y");
165     queue.enqueue("Z");
166
167 //  Now make a SECOND iterator for QUEUE. The FIRST one does not
    • work any more,
168 //  because QUEUE has changed. Use SECOND to visit QUEUE's new
    • elements.
169
170     Iterator<String> second = queue.iterator();
171     while (second.hasNext())
172     {
173       System.out.println(second.next());   //  X Y Z one per line
    •       5 pts.
174     }
```

```
175
176  //  The new iterator hasn't changed QUEUE either.
177
178      System.out.println(queue.isEmpty());   //  false
  •      1 pt.
179      System.out.println(queue.dequeue());   //  X
  •      1 pt.
180      System.out.println(queue.dequeue());   //  Y
  •      1 pt.
181      System.out.println(queue.dequeue());   //  Z
  •      1 pt.
182      System.out.println(queue.isEmpty());   //  true
  •      1 pt.
183    }
184  }
185
```