```
// Lab 12: PriorityQueue
// Had to turn this in late, sorry :(
// Andrea Smith
// CSCI 1913

class PriorityQueue<Base>
{
  private class Node
  {
    private Base object;
    private int rank;
    private Node left;
    private Node right;

    private Node(Base object, int rank)
    {
      this.object = object;
      this.rank = rank;
      this.left = left;
      this.right = right;
    }
  }
  private Node root; // Root of BST
  public PriorityQueue()
  {
    root = new Node(null,-1);
  }


  public Base dequeue()
  {
    if (isEmpty())
    {
      throw new IllegalStateException("The priority queue is
      empty.");
    }
    else
    {
      Node temp = root;
      Node tempR = root.right;

      while(true)
      {
        if (tempR.left == null)
        {

          if (temp.left != tempR)
```

```
47            {
48              temp.right = tempR.right;
49              return tempR.object;
50            }
51            else
52            {
53              temp.left = tempR.right;
54              return tempR.object;
55            }
56          }
57
58          else
59          {
60            temp = tempR;
61            tempR = tempR.left;
62          }
63        }
64      }
65  }
66
67    public void enqueue(Base object, int rank)
68    {
69      if (rank < 0)
70      {
71        throw new IllegalArgumentException("Rank is negative.");
72      }
73
74      else
75      {
76        Node temp = root;
77        while(true)
78        {
79          if (rank >= temp.rank)
80          {
81            if (temp.right != null)
82            {
83              temp = temp.right;
84            }
85
86            else
87            {
88              temp.right = new Node(object, rank);
89              break;
90            }
91          }
92
```

```
93          else
94          {
95            if(temp.left != null)
96            {
97              temp = temp.left;
98            }
99
100           else
101           {
102             temp.left = new Node(object, rank);
103             break;
104           }
105
106         }
107       }
108     }
109   }
110
111   public boolean isEmpty()
112   {
113     return ((root.right == null) && (root.left == null));
114   }
115
116 }
117
118
119 //  SNOBBERY. How the aristocracy behaves in a queue. 20 points.
120
121 class Snobbery
122 {
123
124 //  MAIN. Queue them up.
125
126   public static void main(String[] args)
127   {
128     PriorityQueue<String> queue = new PriorityQueue<String>();
129
130     System.out.println(queue.isEmpty());  //  true        2 points
131
132     try
133     {
134       System.out.println(queue.dequeue());
135     }
136     catch (IllegalStateException ignore)
137     {
138       System.out.println("Blimey!");      //  Blimey!     2 points
139     }
```

```
140
141     queue.enqueue("Lancelot",  5);
142     queue.enqueue("Fawlty",    7);
143     queue.enqueue("Elizabeth", 0);
144     queue.enqueue("Charles",   1);
145     queue.enqueue("Turing",    7);
146
147     try
148     {
149       queue.enqueue("Zeus", -100);
150     }
151     catch (IllegalArgumentException ignore)
152     {
153       System.out.println("No gods!");     //  No gods!    2 points
154     }
155
156     System.out.println(queue.isEmpty());  //  false       2 points
157
158     System.out.println(queue.dequeue());  //  Elizabeth   2 points
159     System.out.println(queue.dequeue());  //  Charles     2 points
160     System.out.println(queue.dequeue());  //  Lancelot    2 points
161     System.out.println(queue.dequeue());  //  Turing      2 points
162     System.out.println(queue.dequeue());  //  Fawlty      2 points
163
164 //  It's OK if Fawlty comes out before Turing, but both must come
  • out last.
165
166     System.out.println(queue.isEmpty());  //  true        2 points.
167   }
168
169 }
170
```