# CONCEPT EXERCISES

**13.1** Declare a PriorityQueue object – and the associated Comparator-implementing class – in which the highest-priority element is the String object of greatest length in the priority queue; for elements of equal length, use lexicographical order. For example, if the elements are “yes”, “maybe”, and “no”, the highest-priority element would be “maybe”.

```
import java.util.*;

public class ReverseLengthHeap
{
        public static void main (String[ ] args)
        {
                PurePriorityQueue<String> heap =
                                new Heap<String> (new ReverseLength());

                heap.add ("yes");
                heap.add ("maybe");
                heap.add ("no");
                while (!heap.isEmpty())
                        System.out.println (heap.removeMin());
        } // method main

} // class ReverseLengthHeap


class ReverseLength implements Comparator<String>
{
        /**
         *  Compares two specified String objects lexicographically if they
         *  have the same length, and otherwise returns the difference in
         *  their lengths.
         *
         *  @param s1 - one of the specified String objects.
         *  @param s2 - the other specified String object.
         *
         *  @return s1.compareTo (s2) if s1 and s2 have the same length;
         *              otherwise, return s2.length() - s1.length().
         *
         */
        public int compare (String s1, String s2)
        {
                int len1 = s1.length(),
```

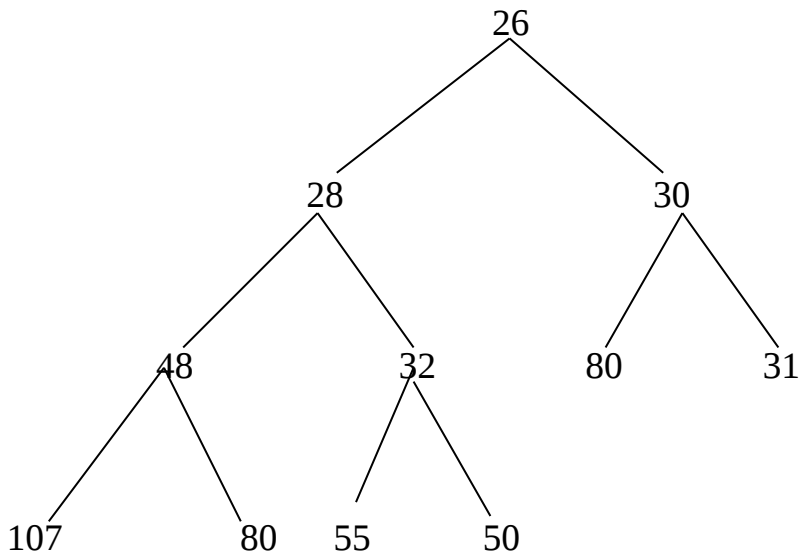
```
            len2 = s2.length();
        if (len1 == len2)
            return s1.compareTo (s2);
        return len2 - len1;
    } // method compare

} // class ReverseLength
```

**13.2** In practical terms, what is the difference between the Comparable interface and the Comparator interface? Give an example in which the Comparator interface must be used.

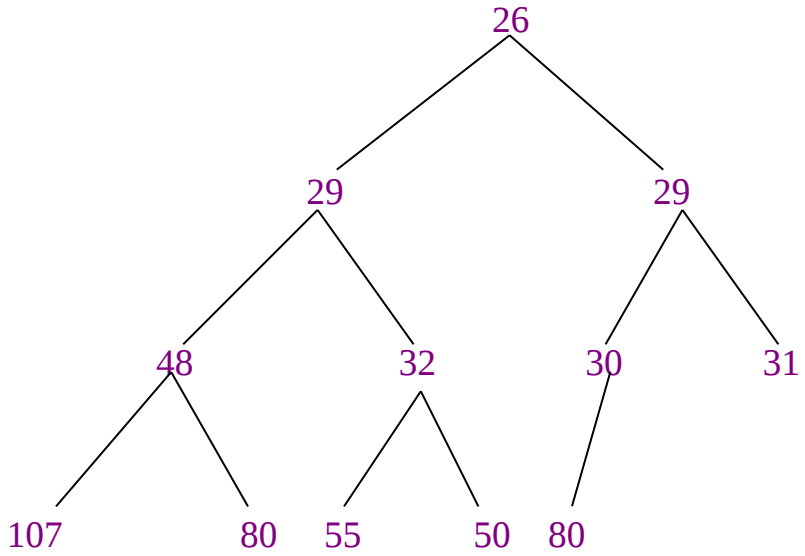
The Comparable interface's compareTo method compares the calling object to the argument in the method call. The Comparator interface's compare method compares the two arguments in the method call. The String class implements the Comparable interface.

An implementation of the Comparator interface is needed when a user wants a different ordering than the one supplied for the elements' class, and that class cannot be modified by the user. For example, in the String class, the compareTo method performs a lexicographic comparison. If, as in Concept Exercise 13.1, the highest-priority string is to be the one whose length is largest, an implementation of the Comparator interface is required.

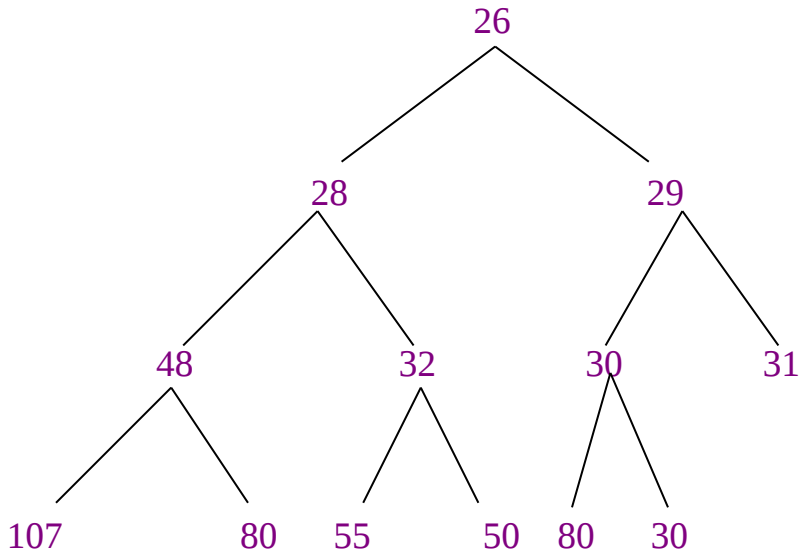
**13.3** Show the resulting heap after each of the following alterations is made, consecutively, to the following heap:



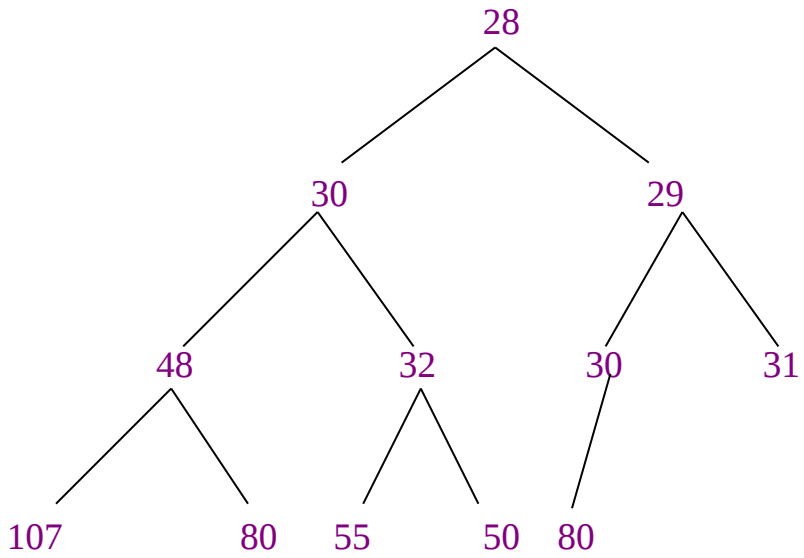

**a.** add (29);



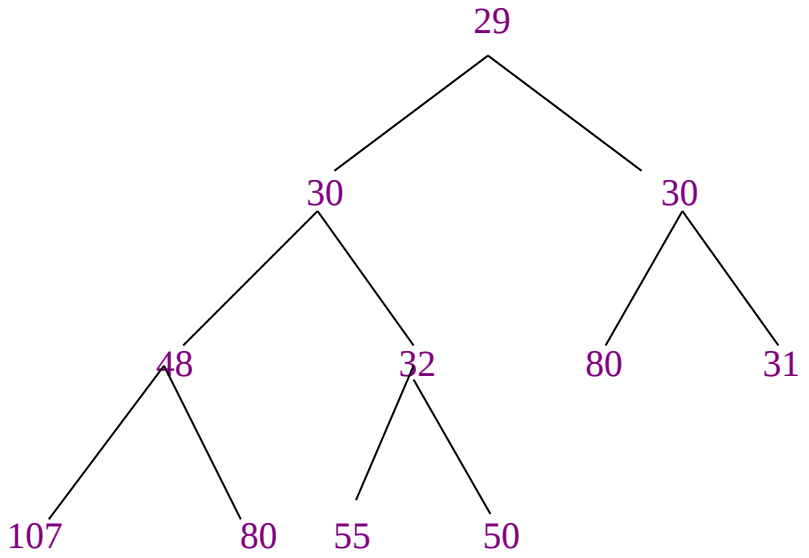

**b.** add (30);




**c.** remove();

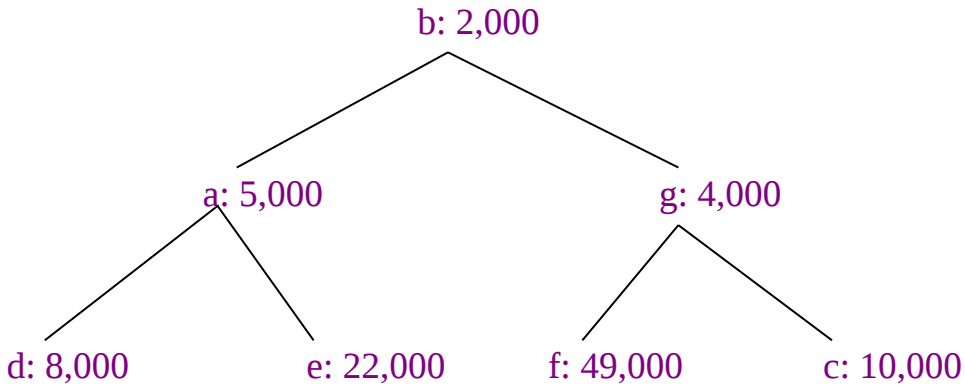



**d.** remove();




**13.4** For the following character frequencies, create the heap of character-frequency pairs (highest priority = lowest frequency):

a: 5,000
b: 2,000
c: 10,000
d: 8,000
e: 22,000
f: 49,000
g: 4,000



## PROGRAMMING EXERCISES

**13.1** In Section 13.3.1, the PriorityQueueExample class creates a heap, pq1, of Student objects; the Student class implements the Comparable interface. Re-write the project so that the Comparator interface is implemented instead for pq1. Re-run the project to confirm your revisions.

Replace

```
PriorityQueue<Student> pq1 = new PriorityQueue<Student>(),
```

with

```
PriorityQueue<Student> pq1 = new PriorityQueue<Student>( new ByLowestGPA()),
```

and add the following class

```
public class ByLowestGPA implements Comparator<Student>
{
        public int compare (Student stu1, Student stu2)
        {
                return stu1.compareTo (stu2);
        } // method compare
} // class ByLowestGPA
```