# 16.8 Class `PriorityQueue` and Interface `Queue`

Recall that a queue is a collection that represents a waiting line—typically, *insertions* are made at the back of a queue and *deletions* are made from the front. In Section 21.6, we'll discuss and implement a queue data structure. In Chapter 23, Concurrency, we'll use concurrent queues. In this section, we investigate Java's `Queue` interface and `PriorityQueue` class from package `java.util`. Interface `Queue` extends interface `Collection` and provides additional operations for *inserting*, *removing* and *inspecting* elements in a queue. `PriorityQueue`, which implements the `Queue` interface, orders elements by their natural ordering as specified by `Comparable` elements' `compareTo` method or by a `Comparator` object that's supplied to the constructor.

Class `PriorityQueue` provides functionality that enables *insertions in sorted order* into the underlying data structure and *deletions* from the *front* of the underlying data structure. When adding elements to a `PriorityQueue`, the elements are inserted in priority order such that the *highest-priority element* (i.e., the largest value) will be the first element removed from the `PriorityQueue`.

The common `PriorityQueue` operations are `offer` to *insert* an element at the appropriate location based on priority order, `poll` to *remove* the highest-priority element of the priority queue (i.e., the head of the queue), `peek` to get a reference to the highest-priority element of the priority queue (without removing that element), `clear` to *remove all elements* in the priority queue and `size` to get the number of elements in the priority queue.

Figure 16.14 demonstrates class `PriorityQueue`. Line 8 creates a `PriorityQueue` that stores `Double`s with an *initial capacity* of `11` elements and orders the elements according to the object's natural ordering (the defaults for a `PriorityQueue`). `PriorityQueue` is a generic class. Line 8 instantiates a `PriorityQueue` with a type argument `Double`. Class `PriorityQueue` provides several additional constructors. One of these takes an `int` and a `Comparator` object to create a `PriorityQueue` with the *initial capacity* specified by the `int` and the *ordering* by the `Comparator`. Lines 11–13 use method `offer` to add elements to the priority queue. Method `offer` throws a `NullPointerException` if the program attempts to add a `null` object to the queue. The loop in lines 18–21 uses method `size` to determine whether the priority queue is *empty* (line 18). While there are more elements, line 19 uses `PriorityQueue` method `peek` to retrieve the *highest-priority element* in the queue for output (*without* actually removing it from the queue). Line 20 removes the highest-priority element in the queue with method `poll`, which returns the removed element.

```
1   // Fig. 16.14: PriorityQueueTest.java
2   // PriorityQueue test program.
3   import java.util.PriorityQueue;
4
5   public class PriorityQueueTest {
6      public static void main(String[] args) {
7         // queue of capacity 11
8         PriorityQueue<Double> queue = new Priority
9
10        // insert elements to queue
11           queue.offer(3.2);
12           queue.offer(9.8);
13           queue.offer(5.4);
14
15        System.out.print("Polling from queue: ");
16
17        // display elements in queue
18           while (queue.size() > 0) {
19              System.out.printf("%.1f ", queue.peek()
20              queue.poll(); // remove top element
21           }
22        }
23  }
```

```
Polling from queue: 3.2 5.4 9.8
```

# Fig. 16.14

`PriorityQueue` test program.