21.6 Queues

Another commonly used data structure is the queue. A queue is similar to a checkout line in a supermarket—the cashier services the person at the *beginning* of the line *first*. Other customers enter the line only at the end and wait for service.

Queue Operations

Queue nodes are removed only from the head (or front) of the queue and are inserted only at the tail (or end). For this reason, a queue is a **first-in, first-out** (**FIFO**) data structure. The insert and remove operations are known as **enqueue** and **dequeue**.

Queue Applications

Queues have many uses in computer systems. Each core in a computer CPU can service only one application at a time. Each application requiring processor time can be placed in a queue. The application at the front of the queue is the next to receive service. Each application gradually advances to the front as the applications before it receive service.

Queues are also used to support **print spooling**. For example,

a single printer might be shared by all users of a network. Many users can send print jobs to the printer, even when the printer is already busy. These print jobs can be placed in a queue until the printer becomes available. A program called a **spooler** manages the queue to ensure that, as each print job completes, the next one is sent to the printer.

Information packets also *wait* in queues in computer networks. Each time a packet arrives at a network node, it must be routed to the next node along the path to the packet's final destination. The routing node routes one packet at a time, so additional packets are enqueued until the router can route them.

A file server in a computer network handles file-access requests from many clients throughout the network. Servers have a limited capacity to service requests from clients. When that capacity is exceeded, client requests *wait* in queues.

`Queue<E>` Class That Contains a `List<E>`

Figure 21.11 creates a `Queue<E>` class that contains a `List<E>` (Fig. 21.3) object and provides methods `enqueue`, `dequeue`, `isEmpty` and `print`. Class `List<E>` contains some methods (e.g., `insertAtFront` and `removeFromBack`) that shouldn't be accessible through `Queue<E>`'s `public` interface. Using composition enables

us to hide class `List<E>`'s other `public` methods from `Queue<E>`'s client code. Each `Queue<E>` method delegates to a `List<E>` method—`enqueue` calls `List<E>` method `insertAtBack` (Fig. 21.11, line 14), `dequeue` calls `List<E>` method `removeFromFront` (line 18), `isEmpty` calls `List<E>` method `isEmpty` (line 22) and `print` calls `List<E>` method `print` (line 25). For reuse, class `Queue<E>` is declared in package `com.deitel.datastructures`. Again, we do not import `List<E>` because it's in the same package.

```
1   // Fig. 21.11: Queue.java
2   // Queue uses class List.
3   package com.deitel.datastructures;
4
5   import java.util.NoSuchElementException;
6
7   public class Queue<E> {
8      private List<E> queueList;
9
10     // constructor
11     public Queue() {queueList = new List<E>("queu
12
13     // add object to queue
14     public void enqueue(E object) {queueList.inse
15
16     // remove object from queue
17     public E dequeue() throws NoSuchElementExcept
18        return  queueList.removeFromFront();
19     }
20
21     // determine if queue is empty
22     public boolean isEmpty() {return queueList.is
23
24     // output queue contents
25     public void print() {queueList.print();}
```

```
26    }
```

Fig. 21.11

`Queue` uses class `List`.

Testing Class `Queue<E>`

Class `QueueTest`'s (Fig. 21.12) `main` method creates and initializes `Queue<E>` variable `queue` (line 8). Lines 11, 13, 15 and 17 enqueue four integers, taking advantage of *auto-boxing* to insert `Integer` objects into the queue. Lines 21–33 dequeue the objects in first-in, first-out order. When the queue is empty, method `dequeue` throws a `NoSuchElementException` and the program displays the exception's stack trace.

```
1   // Fig. 21.12: QueueTest.java
2   // Class QueueTest.
3   import com.deitel.datastructures.Queue;
4   import java.util.NoSuchElementException;
5
6   public class QueueTest {
7     public static void main(String[] args) {
8       Queue<Integer> queue = new Queue<>();
9
10      // use enqueue method
11        queue.enqueue(-1);
12        queue.print();
```

```
13          queue.enqueue(0);
14            queue.print();
15          queue.enqueue(1);
16            queue.print();
17          queue.enqueue(5);
18            queue.print();
19
20        // remove objects from queue
21        boolean continueLoop = true;
22
23           while (continueLoop) {
24              try {
25                int removedItem = queue.dequeue(); /
26                System.out.printf("%n%d dequeued%n",
27                   queue.print();
28              }
29             catch (NoSuchElementException noSuchEle
30                continueLoop = false;
31                noSuchElementException.printStackTra
32              }
33           }
34        }
35     }
```

```
The queue is: -1
The queue is: -1 0
The queue is: -1 0 1
The queue is: -1 0 1 5

-1 dequeued
The queue is: 0 1 5

0 dequeued
The queue is: 1 5

1 dequeued
The queue is: 5
```

```
                    5 dequeued
                    Empty queue
java.util.NoSuchElementException: queue is empty
        at com.deitel.datastructures.List.removeFromF
        at com.deitel.datastructures.Queue.dequeue(Qu
           at QueueTest.main(QueueTest.java:25)
```

Fig. 21.12

Class `QueueTest`.