queues[3]: Natalie Attired (at the front), followed by Gene Pool
queues[2]: Emanual Transmission (at the front), followed by Kay Sera
queues[1]: empty
queues[0]: Ginger Snap

When an item needs to be removed, we move down through the ordinary queues, starting with the highest priority, until we find a nonempty queue. We then remove the front item from this nonempty queue. This process could be made more efficient if we also keep a member variable to indicate which is the highest numbered ordinary queue that is non-empty.

With this approach, we can implement a priority queue as an array of ordinary queues and do not need to duplicate a lot of the previous work. In fact, it is possible to implement the priority queue with no instance variables other than the queues array. However, some efficiency is gained by adding two other instance variables.

The first is a number, totalSize, which keeps track of the total number of items in all of the queues. With this instance variable, a size method for the priority queue can operate in constant time.

The second extra instance variable is an integer, which keeps track of the highest priority that is currently in the priority queue. This makes remove more efficient since, without the extra instance variable, we must always search from highest downward until we find a nonempty queue. With the extra private instance variable, remove can go straight to the correct queue and remove an item. (Of course, this extra instance variable also must be maintained correctly so that its value is always the priority of the highest item.) We leave the remainder of the details of writing this array-based implementation for Programming Project 4 on page 396.

Priority Queue ADT—A Direct Implementation

If the number of possible priorities is large, then an array of queues might be impractical. In this case, you might think of several alternatives. One possibility is to implement the priority queue as an ordinary linked list in which the data in each node contains two things: the item from the queue and the priority of that item. This implementation works regardless of how large the priority range is. We will leave the details of the implementation as another exercise (Programming Project 5 on page 396).

Java's Priority Queue

Java has a different form of a priority queue in java.util.PriorityQueue. This generic class, PriorityQueue<E>, has a generic type parameter for the elements in the queue, and these elements are usually comparable to each other (rather than having a separate priority value for each insertion into the queue).

Self-Test Exercises for Section 7.4

23. Implement the add method of the priority queue. Use the implementation that has an array called queues of ordinary queues.
24. Suppose you know that the number of priorities that will be used in a priority queue application is small, but you do not know what the actual priorities will be. For example, suppose you know that the priorities can be any integer values in the range 0 to 1,000,000, but you also know that there will be at most 25 different numbers used. How might you implement the priority queue as an array of ordinary queues in such a way that the array size is much smaller than 1,000,000?

CHAPTER SUMMARY

- A queue is a first-in/first-out data structure.
- A queue can be used to buffer data that is being sent from a fast computer component to a slower component. Queues also have many other applications: in simulation programs, operating systems, and elsewhere.
- A queue can be implemented as a partially filled circular array.
- A queue can be implemented as a linked list.
- When implementing a stack, you need keep track of only one end of the list of items, but when implementing a queue, you need to keep track of both ends of the list.
- A priority queue can be implemented as an array of ordinary queues or in various ways as a linked list.

Solutions to Self-Test Exercises

1. LIFO (Last-In/First-Out) and FIFO (First-In/First-Out) refer to the order in which entries must be removed.
2. For adding: insert or enqueue ("enter queue"); for removing: getFront and dequeue ("delete from queue")
3. The isEmpty function tests for an empty queue.
4. The operations are add 'L', add 'I', add 'N', add 'E', followed by four remove operations that return 'L', then 'I', then 'N', then 'E'.
5. Reading input from a keyboard and sending output to a printer.
6. Read the characters two at a time. Each pair of characters has the first character placed in queue number 1 and the second character placed in queue number 2. After all the reading is done, print all characters from queue number 1 on one line, and print all characters from queue number 2 on a second line.

7. A straightforward approach is to use a second queue, called `line`. As the input is being read, each character is placed in the `line` queue (as well as being placed in the original stack and queue). During the comparison phase, we also keep track of how many characters correctly match. If a mismatch occurs, we can then print an appropriate amount of the `line` queue as part of the error message.

8. Do not apply the `toUpperCase` method.

9. Yes.

10. We assumed that no more than one customer arrives during any particular second.

11. (a) Sometimes add a new customer to the arrivals queue; (b) sometimes start a new car through the washer; (c) tell the washer that another second has passed.

12. These are cars that arrived during the simulation, but they are still waiting in line at the end of the simulation.

13. The total number of cars can never exceed `Integer.MAX_VALUE` because there is at most one car per second and the total number of seconds in the simulation is an integer.

14. A helper method is a private method that is useful when a class requires an operation that does not need to be part of the public interface.

15. The body of the method should check that `size( ) > 0` and then return `data[rear]`.

16. The main problem is that you cannot tell when the queue is empty because `front` and `rear` have no meaning for an empty queue.

17. Here's the private method with its precondition/postcondition contract:

```
private int NextIndex(int i)
// Precondition: 0 <= i and i < data.length
// Postcondition: If i+1 is data.length,
// then the return value is zero; otherwise
// the return value is i+1.
{
   if (++i == data.length)
      return 0;
   else
      return i;
}
```

18. The `trimToSize` method should handle these cases: (1) If the length of the array is already equal to the number of items, then do no work; (2) If the number of items is zero, then change `data` to an array of capacity zero and return with no other work; (3) If `(front<=rear)`, then allocate a new array of the correct size and activate `System.arraycopy` once to copy the items from the old array to the new array; (4) If `(front>rear)`, then allocate a new array of the correct size and activate `System.arraycopy` twice to copy the two segments into the new array.

    The fourth case is similar to the final case of `ensureCapacity`. If you copy the items into the front of the new array, then make sure you also change `front` and `rear` to reflect the new positions of the items.

19. Here's the extra work after initially setting `answer` with `super.clone`:

```
Node[ ] cloneInfo;
cloneInfo =
   Node.listCopyWithTail(front);
answer.front = cloneInfo[0];
answer.rear = cloneInfo[1];
return answer;
```

20. The body of the method should check that `rear` is not `null` and then return with:

```
return rear.getData( );
```

21. The body of the method should check that `size( ) > 0` and then return with:

```
return rear.getData( );
```

22. Removals will be hard to implement.

23. The implementation should check that the priority is valid. Then activate the method `queues[priority].add[item]`. If you are keeping track of other items (such as the current highest priority and the total size), then those items should be updated, too.

24. Since you know that there are at most 25 different priorities, you could get by with an array of only 25 ordinary queues plus a second array of integers called `value`. The number in `value[i]` would be the priority of all the items in `queues[i]`. It might also make sense to keep the arrays sorted according to priorities.

PROGRAMMING PROJECTS

1 In Figure 7.3 on page 356, we presented a program that checks a string to see if the letters in the string read the same forward and backward. These strings are called palindromes. Another kind of palindrome is one in which we look at words rather than letters. A **word-by-word palindrome** is a string of words such that the words read the same forward and backward. For example, the quote "You can cage a swallow, can't you, but, you can't swallow a cage, can you?" is a word-by-word palindrome. Write a program to test an input string and tell whether or not it is a word-by-word palindrome. Consider upper- and lowercase letters to be the same letter. Define a word as any string consisting of only letters or an apostrophe and bounded at each end with one of the following: a space, a punctuation mark, the beginning of the line, or the end of the line. Your program should have a friendly interface and allow the user to check more lines until the user wishes to quit the program.

2 In Figure 7.3 on page 356, we presented a program that checks a string to see if the letters in the string read the same forward and backward. The previous exercise performed a similar check using words in place of letters. In this exercise, you are to write a program that runs a similar check using lines rather than words or letters. Write a program that reads in several lines of text and decides if the passage reads the same whether you read the lines top to bottom or bottom to top; this is yet another kind of palindrome. For example, the following poem by J.A. Lindon reads the same whether you read the lines from top to bottom or from bottom to top:

As I was passing near the jail
I met a man, but hurried by.
His face was ghastly, grimly pale.
He had a gun. I wondered why.
He had. A gun? I wondered...why,
His face was ghastly! Grimly pale,
I met a man but hurried by,
As I was passing near the jail.

Consider upper- and lowercase versions of a letter to be the same letter. Consider word boundaries to be significant, so for example, the words in the first line must read the same as the words in the last line in order to pass the test (as opposed to just the letters reading the same). However, consider all word delimiters as being equivalent; i.e., a punctuation mark, any number of spaces, the beginning or end of a line, or any combination of these are all considered to be equivalent. The end of the passage should be marked by a line containing only the word "end," spelled with any combination of upper- and lowercase letters and possibly with blanks before and/or after it. Your program should have a friendly interface and allow the user to check more passages until the user wishes to quit the program. Note that to test your program, you need not use such well-constructed poems. Your program will check any passage, regardless of its literary merit.

3 Enhance the car wash simulation method in Figure 7.8 on page 370 so that it has the following additional property. There is an additional parameter, which is a maximum length for the queue. When the queue gets as long as this maximum, any customer who arrives will leave