be written are `addFront`, `removeBack`, and `getBack`. To do this, we need to make the data members of the original queue class protected, after which we can write the additional methods. This is left as Exercise 15.5.

Summary

This chapter described the implementation of the stack and queue classes. Both the stack and queue can be implemented by using a contiguous array or a linked list. In each case, all operations use constant time; thus all operations are fast.

In other programming languages, the array implementation typically uses more memory but less time than the linked-list version, thereby yielding a classic time versus space tradeoff. In Java, this is a nonissue.

```
 1 // Double ended queue (Deque) class
 2 //
 3 // CONSTRUCTION: with no initializer;
 4 //
 5 // ******************PUBLIC OPERATIONS*********************
 6 // void addFront( Object x ) --> Insert x at front
 7 // void addBack( Object x )  --> Insert x at back
 8 // void removeFront( )       --> Remove front item
 9 // void removeBack( )        --> Remove back item
10 // Object getFront( )        --> Return front item
11 // Object getBack( )         --> Return back item
12 // boolean isEmpty( )        --> If empty return true else false
13 // void makeEmpty( )         --> Remove all items
14 // enqueue and dequeue are available, but should not be used
15 // ******************ERRORS********************************
16 // Exception thrown for get or remove on empty deque
17
18 class Deque extends QueueAr
19 {
20
21     public void addFront( Object x )
22       { /* Exercise 15.5 */ }
23     public void addBack( Object x )
24       { enqueue( x ); }
25     public Object removeFront( ) throws Underflow
26       { return dequeue( ); }
27     public void removeBack( ) throws Underflow
28       { /* Exercise 15.5 */ }
29     public Object getBack( ) throws Underflow
30       { /* Exercise 15.5 */ }
31     // isEmpty, makeEmpty, getFront are all inherited
32 }
```

Figure 15.26 Double-ended queue class `Deque` derived from `QueueAr`