```
 1 // MemCell interface: simulate one generic RAM cell
 2 //
 3 // ******************PUBLIC OPERATIONS*********************
 4 // Object read( )         --> Returns the stored value
 5 // void write( Object x ) --> Stores x
 6
 7 public interface MemCell
 8 {
 9     Object read( );
10     void write( Object x );
11 }
```

Figure 6.1 Interface for the abstract memory cell class

Data structures allow us to achieve component reuse.

This approach — the separation of the interface and implementation — is part of the object-oriented paradigm. The user of the data structure does not need to see the implementation, only the available operations. This is the encapsulation and information-hiding part of object-oriented programming. However, another important part of object-oriented programming is *abstraction*. We must think more carefully about the design of the data structures because we must write programs that use these data structures without having their implementations. This in turn makes the interface cleaner, more flexible (that is, more reusable), and generally easier to implement.

All the data structures are easy to implement if we are not concerned about performance. This allows us to plug "cheap" components into our program for the purposes of debugging. The exercises at the end of this chapter ask you to write inefficient implementations that are suitable for processing small amounts of data. Later, we replace the "cheap" data structure implementations with implementations that have better time-performance and/or space-performance properties and that are suitable for processing large amounts of data. Because the interfaces are fixed, these changes require virtually no change to the programs that use the data structures.

This chapter describes data structures by using interfaces. In Parts IV and V we implement the interface and at that time derive a new class. For instance, the stack is specified by the interface `Stack`. When we implement it in Chapter 15, a resulting class will be named `StackAr` (for an array-based implementation). In all cases, the new class will implement the specifications of the interface and may include some additional functionality.

As an example, in Figure 6.1 is an interface for the memory cell described in Section 4.6. The interface describes the available functions; the concrete derived class must provide a definition. The implementation is shown in Figure 6.2 and is identical to that in Figure 4.16, except for the `implements` clause. The `main` routine in Figure 4.17 can be used without change.