Stock kate, joe;

This creates two objects of the Stock class, one named kate and one named joe.

Next, consider how to use a member function with one of these objects. The answer, as with structures and structure members, is to use the membership operator:

kate.show();  // the kate object calls the member function
joe.show();   // the joe object calls the member function

The first call invokes show() as a member of the kate object. This means the method interprets shares as kate.shares and share_val as kate.share_val. Similarly, the call joe.show() makes the show() method interpret shares and share_val as joe.shares and joe.share_val, respectively.

Remember

When you call a member function, it uses the data members of the particular object used to invoke the member function.

Similarly, the function call kate.sell() invokes the set_tot() function as if it were kate.set_tot(), causing that function to get its data from the kate object.

Each new object you create contains storage for its own internal variables, the class members. But all objects of the same class share the same set of class methods, with just one copy of each method. Suppose, for example, that kate and joe are Stock objects. Then, kate.shares occupies one chunk of memory and joe.shares occupies a second chunk of memory. But kate.show() and joe.show() both invoke the same method, that is, both execute the same block of code. They just apply the code to different data. Calling a member function is what some OOP languages term sending a message. Thus, sending the same message to two different objects invokes the same method but applies it to two different objects. (See Figure 10.2.)

Figure 10.2. Objects, data, and member functions.