First, we have to think a bit about how to represent stocks. We could take one share of stock as the basic unit and define a class to represent a share. However, that implies that you would need 100 objects to represent 100 shares, and that's not practical. Instead, let's represent a person's current holdings in a particular stock as a basic unit. The number of shares owned would be part of the data representation. A realistic approach would have to maintain records of such things as initial purchase price and date of purchase, for tax purposes. Also, it would have to manage events such as stock splits. That seems a bit ambitious for our first effort at defining a class, so let's take an idealized, simplified view of matters. In particular, let's limit the operations we can perform to the following:

- Acquire stock in a company.
- Buy more shares of the same stock.
- Sell stock.
- Update the per-share value of a stock.
- Display information about the holdings.

We can use this list to define the public interface for the stock class, leaving additional features as exercises for the interested. To support this interface, we need to store some information. Again, we use a simplified approach. For example, we won't worry about the standard U.S. practice of evaluating stocks in multiples of eighths of a dollar. (Apparently the New York Stock Exchange must have seen this simplification in a previous edition of the book, for it has decided to change over to the system used here.) We will store the following information:

- Name of company
- Number of stocks owned
- Value of each share
- Total value of all shares

Next, let's define the class. Generally, a class specification has two parts:

- A **class declaration**, which describes the data component, in terms of data members, and the public interface, in terms of member functions