commonly use classes to implement class descriptions while restricting structures to representing pure data objects or, occasionally, classes with no private component.

Implementing Class Member Functions

We still have the second part of the class specification to do: providing code for those member functions represented by a prototype in the class declaration. Let’s look at that next. Member function definitions are much like regular function definitions. They have a function heading and a function body. They can have return types and arguments. But they also have two special characteristics:

- When you define a member function, you use the scope operator (::) to identify the class to which the function belongs.

- Class methods can access the private components of the class.

Let’s look at these points now.

First, the function heading for a member function uses the scope operator (::) to indicate to which class the function belongs. For example, the heading for the `update()` member function looks like this:

```cpp
void Stock::update(double price)
```

This notation means we are defining the `update()` function that is a member of the `Stock` class. Not only does this identify `update()` as a member function, it means we can use the same name for a member function for a different class. For example, an `update()` function for a `Buffoon` class would have this function heading:

```cpp
void Buffoon::update()
```

Thus, the scope resolution operator resolves the identity of the class to which a method definition applies. We say that the identifier `update()` has class scope. Other member functions of the `Stock` class can, if necessary, use the `update()` method without using the scope resolution operator. That's because they belong to the same class, making