for show() fails to guarantee that it won't modify the invoking object, which, as it is const, should not be altered. We've solved this kind of problem before by declaring a function's argument to be a const reference or a pointer to const. But here we have a syntax problem: The show() method doesn't have any arguments. Instead, the object it uses is provided implicitly by the method invocation. What's needed is a new syntax, one that says a function promises not to modify the invoking object. The C++ solution is to place the const keyword after the function parentheses. That is, the show() declaration should look like this:

```cpp
void show() const;    // promises not to change invoking object
```

Similarly, the beginning of the function definition should look like this:

```cpp
void stock::show() const   // promises not to change invoking object
```

Class functions declared and defined this way are called const member functions. Just as you should use const references and pointers as formal function arguments whenever appropriate, you should make class methods const whenever they don't modify the invoking object. We'll follow this rule from here on out.

**Constructors and Destructors in Review**

Now that we've gone through a few examples of constructors and destructors, you might want to pause and assimilate what has passed. To help you, here is a summary of these methods.

A constructor is a special class member function that's called whenever an object of that class is created. A class constructor has the same name as its class, but, through the miracle of function overloading, you can have more than one constructor with the same name, provided that each has its own signature, or argument list. Also, a constructor has no declared type. Usually, the constructor is used to initialize members of a class object. Your initialization should match the constructor's argument list. For example, suppose the Bozo class has the following prototype for a class constructor:

```cpp
Bozo(char * fname, char * lname);   // constructor prototype
```

Then, you would use it to initialize new objects as follows: