Sometimes it would be nice to have symbolic constants of class scope. For example, the Stock class declaration used a literal 30 to specify the array size for company. Also, because the constant is the same for all objects, it would be nice to create a single constant shared by all objects. You might think the following would be a solution:

```cpp
class Stock
{
private:
    const int Len = 30;  // declare a constant? FAILS
    char company[Len];
...
```

But this won't work because declaring a class describes what an object looks like but doesn't create an object. Hence, until you create an object, there's no place to store a value. There are, however, a couple of ways to achieve essentially the same desired effect.

First, you can declare an enumeration within a class. An enumeration given in a class declaration has class scope, so you can use enumerations to provide class scope symbolic names for integer constants. That is, you can start off the Stock declaration this way:

```cpp
class Stock
{
private:
    enum {Len = 30};  // class-specific constant
    char company[Len];
...
```

Note that declaring an enumeration in this fashion does not create a class data member. That is, each individual object does not carry an enumeration in it. Rather, Len is just a symbolic name that the compiler replaces with 30 when it encounters it in code in class scope.

Because this uses the enumeration merely to create a symbolic constant with no intent of creating variables of the enumeration type, you needn't provide an enumeration tag. Incidentally, for many implementations, the ios_base class does something similar in its public section; that's the source of identifiers such as ios_base::fixed. Here fixed is typically an enumerator defined in the ios_base class.