After you've used either method (no arguments or default values for all arguments) to create the default constructor, you can declare object variables without initializing them explicitly:

```c++
Stock first;                // calls default constructor implicitly
Stock first = Stock();     // calls it explicitly
Stock *prelief = new Stock; // calls it implicitly
```

However, don't be misled by the implicit form of the nondefault constructor:

```c++
Stock first("Concrete Conglomerate");      // calls constructor
Stock second();                            // declares a function
Stock third;                               // calls default constructor
```

The first declaration calls the nondefault constructor, that is, the one that takes arguments. The second declaration states that `second()` is a function that returns a `Stock` object. When you implicitly call the default constructor, don't use parentheses.

**Destructors**

When you use a constructor to create an object, the program undertakes the responsibility of tracking that object until it expires. At that time, the program automatically calls a special member function bearing the formidable title of destructor. The destructor should clean up any debris, so it actually serves a constructive purpose. For example, if your constructor uses `new` to allocate memory, the destructor should use `delete` to free that memory. The `Stock` constructor doesn't do anything fancy like using `new`, so it doesn't really need a destructor. But it's a good idea to provide one anyway in case a future class revision needs one.

Like a constructor, the destructor has a special name: the class name preceded by a tilde (~). Thus, the destructor for the `Stock` class is called `~Stock()`. Also, like a constructor, a destructor can have no return value and has no declared type. Unlike the constructor, the destructor must have no arguments. Thus, the prototype for a `Stock` destructor must be this: