9.3 Class Data

What about "class data", data items common to each object in a class? What would you want that for? Well, in your Person class, you might like to keep track of the total people alive. How do you implement that?

You *could* make it a global variable called $Person::Census. But about only reason you'd do that would be if you *wanted* people to be able to get at your class data directly. They could just say $Person::Census and play around with it. Maybe this is ok in your design scheme. You might even conceivably want to make it an exported variable. To be exportable, a variable must be a (package) global. If this were a traditional module rather than an object-oriented one, you might do that.

While this approach is expected in most traditional modules, it's generally considered rather poor form in most object modules. In an object module, you should set up a protective veil to separate interface from implementation. So provide a class method to access class data just as you provide object methods to access object data.

So, you *could* still keep $Census as a package global and rely upon others to honor the contract of the module and therefore not play around with its implementation. You could even be supertricky and make $Census a tied object as described in *perltie*, thereby intercepting all accesses.

But more often than not, you just want to make your class data a file-scoped lexical. To do so, simply put this at the top of the file:

```
    my $Census = 0;
```

Even though the scope of a my() normally expires when the block in which it was declared is done (in this case the whole file being required or used), Perl's deep binding of lexical variables guarantees that the variable will not be deallocated, remaining accessible to functions declared within that scope. This doesn't work with global variables given temporary values via local(), though.

Irrespective of whether you leave $Census a package global or make it instead a file-scoped lexical, you should make these changes to your Person::new() constructor:

```
    sub new {
        my $class = shift;
        my $self  = {};
        $Census++;
        $self->{NAME}   = undef;
        $self->{AGE}    = undef;
        $self->{PEERS}  = [];
        bless ($self, $class);
        return $self;
    }

    sub population {
        return $Census;
    }
```

Now that we've done this, we certainly do need a destructor so that when Person is destroyed, the $Census goes down. Here's how this could be done:

```
    sub DESTROY { --$Census }
```

Notice how there's no memory to deallocate in the destructor? That's something that Perl takes care of for you all by itself.

Alternatively, you could use the Class::Data::Inheritable module from CPAN.