



Object-Oriented Perl

Alberto Manuel Simões
 ambs@di.uminho.pt

Although Perl is a scripting language you can use it as an imperative language, a functional language or an object oriented language. I present a brief introduction to Object Oriented Perl programming.

First Steps

The usual way to program Perl with objects is to create a package (also called a “module”) for each necessary object. These packages will act like the classes in Java and other languages.

To start, I create a Perl file `ootest.pl` with this initial code (The entire file is at the end of this article):

```
package Counter;
sub new {
    my $class = shift;
    my $self = { value => 0 };
    return bless $self, $class;
}
```

The first line declares a new package named `Counter`. This starts my class which implements a counter object. Then I declare a function (or method) named `new`. This function is called a “constructor” since it creates a new object. Although it is usual to name `new` for constructors since other languages require that name, Perl doesn’t care which name I use so I can use any valid function name.

The constructor declares a scalar variable, `$self`, which is a hash reference. The hash contains a key named `value` with the initial value of 0. The next line returns the object. Objects are simply references (usually to hashes or arrays) blessed to the corresponding package.

In a script I can create an object even though I can’t do anything with it yet.

```
my $object = Counter->new;
```

First, I want to be able to get the current value of my counter. I add another method I call `value` that returns current counter value.

```
sub value {
    my $self = shift;
    return $self->{value};
}
```

All object methods in Perl receive as their first argument the reference to the object. In this case, once I get the object it is simple to return the current counter value by accessing its value from the hash reference.

Now my script is a little more useful, but not by much since I can access the value that I already know.

```
my $counter = Counter->new();
my $value = $counter->value; # returns 0
```

I need one more function to increment the counter. My counter isn’t any good unless I can change the value.

```
sub inc {
    my $self = shift;
    ++$self->{value};
}
```

As in the previous method, I take the object reference from the argument list and then increment the object value. In this case I wrote the Perl `++` auto-increment operator before the variable itself because this way the function increments the counter, and since that’s the last evaluated expression it also becomes the return value.

To test these methods, I write the following code at the end of the document:

```
package main;

my $counter = Counter->new;

$counter->inc;
$counter->inc;

print $counter->value; ## prints 2
```

Notice that package `main` is the default package for any perl code. The next line makes `$counter`