Object Oriented Perl using Moose

OOP Moose object oriented class object instance constructor getter setter accessor

Prev Next

In the next few articles we are going to learn how to write Object Oriented code in Perl. We will start with some simple examples and extend them step by step. We start using Moose but we'll also learn how to create classes in other ways.

A constructor with Moose

Let's start by writing a simple script that uses the Person `class`. We don't do anything special yet, just load the module and call the `constructor` to create an `instance`.

```perl
use strict;
use warnings;
use v5.10;

use Person;
my $teacher = Person->new;
```

Save this in somedir/bin/app.pl

This should not be new to you as I am sure you have already used other modules in a similar way. Our focus is how the Person class was implemented:

```perl
package Person;
use Moose;

1;
```

That's it.

This code is saved in somedir/lib/Person.pm.

All you need to do to create a `class` is to create a `package` with the name of the class, add `use Moose;` to it, end the file with a true value, and save it in a file with the same name (case sensitive!) as the package, and with a .pm extension.

Loading Moose automatically sets up `use strict` and `use warnings`. This is nice, but be careful you don't get so used to the convenience that you forget them in non-Moose code.

Loading Moose also automatically adds a default constructor called `new`.

As a side note, it is not a requirement in Perl that the constructor will be called new, but in most cases that's what the author chooses anyway.

Attributes and accessors

Having an empty class is not much fun. Let's go further in our use:

```
use strict;
use warnings;
use v5.10;

use Person;
my $teacher = Person->new;

$teacher->name('Joe');
say $teacher->name;
```

In this code, after creating the `object`, we call the "name" `method` with a string as a parameter; this sets the "name" `attribute` of the class to be 'Joe'. Because this method sets the respective attribute it is also called a `setter`.

Then we call the same method again, this time without any parameter. That will fetch the value previously stored. Because this gets the value this is also called a `getter`.

In our case the `getter` and the `setter` have the same name but it isn't a requirement either.

In general `getters` and `setters` are called `accessors`.

The code implementing the new class is this:

```
package Person;
use Moose;

has 'name' => (is => 'rw');

1;
```

The new part, `has 'name' => (is => 'rw');` says that

"The Person class `has` an attribute called `'name'` which `is` `r`eadable and `w`riteable"

This automatically creates a method called "name" which becomes both a setter (for writing) and a getter (for reading).

Try the code

In order to try this create a directory called "somedir", with a subdirectory called "lib" inside it, and save the Person.pm file inside the "lib" subdirectory. Also create a subdirectory called "bin" and save the script there called person.pl.

You should have

```
somedir/lib/Person.pm
somedir/bin/person.pl
```

Open your terminal (or cmd window on Windows), change the directory to be in "somedir" and type `perl -Ilib bin/person.pl`

(On MS Windows you might need to use back-slashes: \ )

Constructor parameters

In the next script we pass a key-value pair to the constructor, corresponding to the name of the attribute and its value.

```perl
use strict;
use warnings;
use v5.10;

use Person;

my $teacher = Person->new( name => 'Joe' );
say $teacher->name;
```

This works too with the same module as we already have:

Using the constructor to set the initial value of an attribute in this way works without making any changes to the Person module itself.

Moose automatically accepts every `member` (another name for attributes) to be passed during construction.