OOP - Object Oriented Perl

OOP bless new parent base @ISA

Perl provides some basic tools upon which user can build various object oriented systems. On this page you'll find information on the most commonly used "hash-based" object system with Perl with some helper modules.

Moo and Moose are two other hash-based object systems that are mostly compatble with this, but provide lots of extra features that if we wanted, we would need to create ourselves in the classic Perl OOP described on this page.

1. Core Perl OOP: Constructor
2. Core Perl OOP: Getter - Setter
3. What should setters return?
4. constructor-arguments (pro)
5. Destructor
6. Attributes, attribute types (members)
7. Create your own type
8. Getters/Setters
9. Inheritance
10. Polymorphism
11. Encapsulation
12. Singleton
13. Destructor (DESTROY)
14. OOP: Bless, or what you will see in the wild
15. OOP: Class::Accessor A small scale object oriented system in Perl
16. Class methods and Instance methods
17. Automatic Class creation
18. Operator overloading

Class declaration

A class is just a `namespace` created using the `package` keyword. It is usually implemented in a module having the same name. For example the class `My::Date` would be implemented in a file called `Date.pm` located in a directory called `My` having the following content:

```
1. package My::Date;
   use strict;
   use warnings;

5.
```

```
1;
```

The `1;` at the end is needed to indicate successful loading of the file.

This code isn't really a class without a constructor.

Constructor

While `new` is not a reserved word in Perl, most people implement the constructor as the `new` method.

```
1. sub new {
       my ($class, %args) = @_;
       return bless \%args, $class;
   }
```

Instance / Object

An instance or object is a `blessed reference`. In the most common case, as described in this article, it is a `blessed reference to a hash`

Destructor

Perl automatically cleans-up objects when they go out of scope, or when the program ends and usually there is no need to implement a destructor. With that said, there is a special function called `DESTROY`. If it is implemented, it will be called just before the object is destroyed and memory reclaimed by Perl.

```
1. sub DESTROY {
      my ($self) = @_;
      ...
   }
```

Inheritance

You can declare inheritance using the parent directive which replaced the older base directive. In the end they are both just manipulating the `@ISA` array that defines the inheritance.

The main script loads a module, calls its constructor and then calls two methods on it:

examples/oop/inheritance1/main.pl

```
1. #!/usr/bin/perl
   use strict;
   use warnings;
```

```
use MyModule;

my $myObj = MyModule->new;
$myObj->say_hi;
$myObj->say_hello;

```

The module itself declares its inheritance using the `parent` directive.

examples/oop/inheritance1/MyModule.pm

```
package MyModule;
use strict;
use warnings;

use parent 'MyParent';

sub say_hello {
    print "Hello from MyModule\n";
}

1;
```

The module from where we inherit, declares the constructor and another method.

examples/oop/inheritance1/MyParent.pm

```
package MyParent;
use strict;
use warnings;

sub new {
    my ($class) = @_;
    return bless {}, $class;
}

sub say_hi {
    my ($self) = @_;
    print "Hi from MyParent\n";
    return;
}

1;
```

When we call the `new` method on "MyModule" Perl will see that MyModule does not have a 'new' function and it will look at at the next module in the **inheritance chain**. In this case it will look at the `MyParent` module and call `new` there.

The same will happen when we call `say_hi`.

On the other hand when we call `say_hello` perl will already find it in the `MyModule` and call it.

Instead of the `parent` directive, old school code uses the `base` directive:

```
use base 'MyParent';
```

If you are interested in the fully manual process (you should probably never do this), then you can add the parent module to the `@ISA` array directly, but then you also need to load the module yourself.

```
use MyParent;
our @ISA = ('MyParent');
```

One side note. Never ever call your module "Parent.pm" or "Base.pm". That will break your code when you try to run it on an operating system with case insensitive filesystem such as MS Windows or Apple Mac OSX. I know. I fell in that trap while preparing this example.