Perl Objects 101

Simple OOP with Perl

Terms

Class - a template describing data and behavior specific to the object

Object - a container for “some data” which is associated with “some behavior”

Method - a function associated with a class

Attribute - private data associated with each unique object

Perl Terms

Class - a package containing methods

Object - a reference with an associated class

Method - a subroutine which takes an object or class as its first argument

Attribute - any sort of Perl data should work

Bless = magic

```
bless $reference, Classname
```

`$reference` is any reference (scalar, hash, array, function, file handle)

Classname is any package name.

The result is a blessed reference; an object.

Methods

Methods are subs defined in the class’s package.

Call them using the dereferencing arrow:

```
$obj->method( $param1, $param2 );
```

The first parameter will be an object reference.

Accessing the data

Since the object is still a reference, the underlying data are easily accessible.

```
$obj->{attr1} = 42;
```

No Protection

What you just saw is considered BAD.

Any code outside of the class can easily tinker with the insides of the object.

Class Methods

Same syntax to call Class methods:

```
$my_class->method();
```

Except the package name is used instead of the object reference.

Constructors

Constructors are class methods that return an object.

The name is arbitrary, although new() is most commonly used.

```
package MyClass;
sub new {
    bless { attr1 => 42 }, ‘MyClass’;
}
```

Better Constructor

It’s first parameter will normally be the class name.

```
package MyClass;
sub new {
   my ( $class ) = @_;
   bless { attr1 => 42 }, $class;
}
```

Inheritance

@ISA array

It is a package global.

```
package MyInheritedClass;
use vars qw( @ISA );
@ISA = qw( MyClass );
sub my_method2 {};
```

@ISA Continued

When an object or class method is called, Perl gets the package and tries to find a sub in that package with the same name as the method.

If found, it calls that.

If not, it looks up the @ISA array for other packages, and tries to find the method in those.

UNIVERSAL

All classes implicitly inherit from class "UNIVERSAL" as their last base class.

Inheriting Shortcut

This is tedious:

```
use vars qw(@ISA);
@ISA = qw(MyClass);
```

There is a shortcut:

```
use base ‘MyClass’;
```

Calling Inherited Methods

We can do:

```
package MyInheritedClass;
sub method1 {
    my ( $self ) = @_;
    MyClass::method1($self);
}
```

That’s not OOish and won’t work if MyClass doesn’t have a sub method1.

SUPER Pseudo-Class

The right thing would be to do:

```
package MyInheritedClass;
sub method1 {
    my ( $self ) = @_;
    $self->SUPER::method1($self);
}
```

Inheriting Constructors

Normally, a base class constructor will bless the reference into the correct class, so we just need to do some initialization:

```
package MyInheritedClass;
sub new {
      my ( $class, %params ) = @_;
      my $self = $class->SUPER::new(%params);
      # do something with the params here...
      return $self;
}
```

Mostly, such constructors are not needed.

Destructors

Perl has automatic garbage collection, so mostly, destructors aren’t needed.

If they are, create a sub called DESTROY:

```
sub DESTROY {
   my ($self) = @_;
   # free some resources
}
```

Multiple Inheritance

Just put more stuff into @ISA

```
@ISA = qw(Class1, Class2);
```

or

```
use base qw(Class1 Class2);
```

Data Storage for Objects

Most objects use hashrefs.

Convenient to use.

No protection.

- You can’t prevent code from tinkering.

Well, there is a thing called “Inside Out Objects”, but we won’t cover that today.

Other ways of OOP

Many, many modules have been created to help with this.

Currently, Moo/Mouse/Moose is The One True Way

This is Perl, so you know that isn’t quite true.

Let’s look at

…some code.