Introduction to Object Oriented Programming with Moose

What is an Object?

- Holds information about it's self
- Does stuff (functions)
- defined by a class (not an object)

OO in Perl?

- DateTime
- WWW::Mechanize
- DBI
- IO::Handle

Often created with new

- my $mech = WWW::Mechanize->new
- my $dt = DateTime->new(year => 2000, month => ...);

But not always

- my $dt = DateTime->now(time_zone => ‘America/Chicago’);

Holds information

- my $mech = WWW::Mechanize->new
- $mech->get($url)
- my $content = $mech->content()

Not an object.

- use Cwd;
- my $dir = getcwd();

Some times globals point the need for an object

```
#!/usr/bin/env perl
use strict;
use warnings;
use WWW::Mechanize;

my $base = 'http://www.google.com/search?q=';
my $mech = $mech->new;

print query("perl")
print query("moose")

sub query {
  my $query = shift;
  $mech->get($url . $query);
  return $mech->content;
}
```

Becomes....

```
package SearchEngine;
use Moose;
use WWW::Mechanize;

has 'url_base' => (isa => Str, is => 'rw',
                   default => 'http://www.google.com/search?q=');
has 'mech' => (isa => Object, is => 'ro',
                   default => sub {WWW::Mechanize->new});

sub query {
  my $self = shift;
  my $query = shift;

  $self->mech->get($self->url_base . $query);
  return $self->mech->content;
}
1;
```

Now I can query away

```
use SearchEngine;

my $google = SearchEngine->new;
my $alta = SearchEngine->new(url_base =>
'http://www.altavista.com/web/results?q=');

print $google->query('perl');
print $alta->query('perl');

print $google->query('Moose');
print $alta->query('Moose');
```

Change engines after new

```
use SearchEngine;

my $search = SearchEngine->new; # defaults to google
$search->url_base('http://www.altavista.com/web/results?q=');

print $search->query('perl');
```

Moose out of the box

- Type constraints (http://tinyurl.com/moose-types)
- use strict;
- use warnings
- don't have to define new
- Delegation

Delegation

```
package SearchEngine;
use Moose;
use WWW::Mechanize;

has 'url_base' => (isa => Str, is => 'rw',
                   default => 'http://www.google.com/search?q=');
has 'mech' => (isa => Object, is => 'ro',
                   default => sub {WWW::Mechanize->new},
                   handles => { 'content' => 'content' }
                   handles => { 'get' => 'get' } );

sub query {
  my $self = shift;
  my $query = shift;

  #$self->mech->get($self->url_base . $query);
  $self->get($self->url_base . $query);

  #return $self->mech->content;
  return $self->content;
}
1;
```

Why Moose?

- See slide 25 - Intro to Moose

Why Moose

- Moose means less code
  - (less code == less bugs)
- Less tests
  - (why test what moose already does)
- over 6000 unit tests
- 0.24 seconds load time
- ~ 3mb for perl+moose

No Moose

- `no Moose` at the end of a package is a best practice
- Just do it

Immutability

Stevan's Incantation of Fleet-Footedness

```
package Person;

use Moose;


__PACKAGE__->meta->make_immutable;
```

What make_immutable does

Magic

- Uses `eval` to "inline" a constructor
- Memoizes a lot of meta-information
- Makes loading your class slower
- Makes object creation *much* faster

no immutable Moose;

```
package SearchEngine;
use Moose;
use WWW::Mechanize;

has 'url_base' => (isa => Str, is => 'rw',
                   default => 'http://www.google.com/search?q=');
has 'mech' => (isa => Object, is => 'ro',
                   default => sub {WWW::Mechanize->new},
                   handles => { 'content' => 'content' }
                   handles => { 'get' => 'get' } );

sub query {
  my $self = shift;
  my $query = shift;

  #$self->mech->get($self->url_base . $query);
  $self->get($self->url_base . $query);

  #return $self->mech->content;
  return $self->content;
}
__PACKAGE__->meta->make_immutable
no Moose;
1;
```

Moose is cool for tests!

```
# no_net.t

package pseudomech;
sub new{return bless {}}
sub get{
  my $self = shift;
  $self->{query} = shift;
}
sub content {
  return shift->{query} . " response";
}

package main;
use Test::More tests => 2;
use SearchEngine;

my $google = SearchEngine->new(mech => pseudomech->new);

is($google->query('perl'), 'perl response');
is($google->query('Moose'), 'Moose response');

exit;
```

Roles

- Usually what you want when you try to do inheritance
- lets 1 class take on the features of multiple packages.

More Reading

- Moose Cookbook (heavy on the inheritance)
  - http://search.cpan.org/dist/Moose/lib/Moose/Cookbook.pod
- Moose Website
  - http://moose.perl.org