Moose

A postmodern object system for Perl 5

Hi!

Peter Shangov

- mechanicalrevolution.com
- search.cpan.org/~pshangov
- github.com/pshangov

Perl 5 OO

- Powerful
- Flexible
- Damian Conway's "Object Oriented Perl"

- Unintuitive
- Repetitive
- Unsightful
- Class::* namespace

Moose

- Inspirations: Perl 6, Smalltalk, CLOS
- Apocalypse 12 released on 14 April 2004
- First Moose release - 15 March 2006
- 1.0 release -25 March 2010

What Moose does for you

- Attributes
- Roles
- Method modifiers
- Meta programming

Attributes

Attributes are simple

```
package MyFirstProgram;
use Moose;

has greeting => (
    is       => 'ro',
    isa      => 'Str',
    required => 1,
    default  => 'Hello',
);

sub greet {
    my $self = shift;
    print $self->greeting . ", world!";
}
```

Attributes are simple

```
my $program = MyFirstProgram->new;

$program->greet;
# Hello, world!

...

my $terminator = MyFirstProgram->new(
    greeting => 'Hasta la vista',
);

$another_program->greet;
# Hasta la vista, world!
```

Types

```
has greeting => (
   ...
   isa => 'Str',
   ...
);

my $will_not_work = MyFirstProgram->new( greeting => [1,2,3] );

# Attribute (greeting) does not pass the type constraint because:
# Validation failed for 'Str' with value ARRAY(0x3811c)
```

Core Moose types

- `Any`
- `Bool`
- `Undef`
- `Str`
- `Num`
- `Int`
- `ScalarRef`
- `ArrayRef`
- `HashRef`
- `CodeRef`
- `RegexpRef`
- `GlobRef`
- `Object`

... plus any class name

Or DIY ...

```
subtype 'PositiveInt'
      => as 'Int'
      => where { $_ > 0 }
      => message { "$_ is not a positive number!" };

enum 'RGB' => qw( red green blue );
```

Or get them from CPAN ...

- `MooseX::Types::DateTime`
- `MooseX::Types::Email`
- `MooseX::Types::IO`
- `MooseX::Types::IPv4`
- `MooseX::Types::URI`
- `etc ...`

And you can combine them ...

```
isa => 'ArrayRef[My::Class]'

isa => 'HashRef[RegexpRef]'

isa => 'My::Class|My::Other::Class'
```

Type coercion

```
subtype Now => as Str => where { $_ eq 'now' };

subtype DateTime => as class_type 'DateTime';

coerce DateTime
  => from Num
    => via { DateTime->from_epoch( epoch => $_ ) }
  => from HashRef
    => via { DateTime->new( %$_ ) }
  => from Now
     => via { DateTime->now };
```

Type coercion

```
package Reminder {
  has time => ( is => 'ro', isa => 'DateTime');
}

Reminder->new( time => 'now' );

Reminder->new( time => 1298221153 );

Reminder->new( time => {
  year  => 2011
  month => 2
  day   => 26
  ...
});
```

The power stuff: Defaults

```
has time => (
  is      => 'ro',
  isa     => 'DateTime',
  default => sub { DateTime->now },
);
```

More power: builders

```
has time => (
  is      => 'ro',
  isa     => 'DateTime',
  builder => '_build_time',
);

sub _build_time {
  return DateTime->now;
}
```

Lazy builders

```
has time => (
  is      => 'ro',
  isa     => 'DateTime',
  lazy    => 1,
  builder => '_build_time',
);

has time => (
  is         => 'ro',
  isa        => 'DateTime',
  lazy_build => 1,
);
```

Builders: before

```
has [qw(dsn user pass)] => ( is => 'ro', isa => 'Str');
has limit => ( is => 'ro', isa => 'Int');

sub do_stuff_in_db
{
  my $self = shift;

  my $dbh = DBI->connect($self->dsn, $self->user, $self->pass);
  my $sth_calculate = $dbh->selectall_arrayref('SELECT ...');

  if ($sth_calculate->[0][0] > $self->limit )
  {
    my $sth_do_stuff = $dbh->prepare( ... );
    $sth_do_stuff->execute($arg);
  } else {
    say "No worries yet, quitting!";
  }
}
```

Builders: after

```
has dsn           => ... # Str
has user          => ... # Str
has pass          => ... # Str
has limit         => ... # Int
has dbh           => ... # DBI::dbh, lazy_build
has sth_calculate => ... # DBI::sth, lazy_build
has sth_do_stuff  => ... # DBI::sth, lazy_build
has exceeds_limit => ... # Bool,     lazy_build
has message       => ... # Str,      default => "No worrries ..."
```

Builders: after

```
sub _build_dbh {
  DBI->connect($_[0]->dsn, $_[0]->user, $_[0]->pass);
}

sub _build_sth_calculate {
  $_[0]->prepare(...);
}

sub _build_sth_do_stuff {
  $_[0]->prepare(...);
}

sub _build_exceeds_limit {
  $_[0]->sth_calculate->fetchall_arrayref->[0][0] > $_[0]->limit;
}
```

Builders: after

```
sub do_stuff_in_db {
  my ($self, $arg) = @_;

  if ($self->exceeds_limit )
  {
    $self->sth_do_stuff->execute($arg);
  } else {
    say $self->message;
  }
}
```

Delegation

```
package Website {

  has 'uri' => (
    is      => 'ro',
    isa     => 'URI',
    handles => ['host', 'path'],
  );

}

my $site = Website->new(
  uri => 'http://hackers.com/~pshangov',
  ...
);

print $site->host;
# hackers.com
```

Native type delegation

```
package SoftwareProject {

  has bugs => (
    is      => 'rw',
    isa     => 'ArrayRef[SoftwareBug]',
    traits  => ['Array'],
    handles => { submit_bug => 'push' },
  );

}

my $bug = SoftwareBug->new(...);

$project->submit_bug($bug);
# $project->bugs->push($bug)
```

Native type delegation

- `Array:  shift, push, elements, map, grep ...`
- `Hash:   get, set, keys, values ...`
- `Bool:   set, unset, toggle ...`
- `String: append, prepend, chomp, match, substr, length ...`
- `Code:   execute ...`
- `etc ...`

Roles

a.k.a. traits, a.k.a. mixins

Roles

- A recent development in OO programming
- Based on Smalltalk traits, Ruby mixins and Java interfaces
- Developed and made popular by Perl 6
- Growing adoptions, including PHP and Python

Repeatable behaviors

```
package Engine;

has [qw(type consumption ignition)] => ...;

has is_broken => (is => 'rw', isa => 'Bool');

sub break {
  print "OMG, I broke!";
  $_[0]->is_broken(1);
}

package Glass;

has [qw(material height form)] => ...;

has is_broken => (is => 'rw', isa => 'Bool');

sub break {
  print "OMG, I broke!";
  $_[0]->is_broken(1);
}
```

Inheritance

```
package Breakable;

has is_broken => (is => 'rw', isa => 'Bool');

sub break {
  print "OMG, I broke!";
  $_[0]->is_broken(1);
}


package Engine;
extends 'Breakable';
has [qw(type consumption ignition)] => ...;



package Glass;
extends 'Breakable';
has [qw(material height form)] => ...;
```

Role composition

```
package Breakable;

use Moose::Role;

has is_broken => (is => 'rw', isa => 'Bool');

sub break {
  print "OMG, I broke!";
  $_[0]->is_broken(1);
}


package Engine;
with 'Breakable';
has [qw(type consumption ignition)] => ...;


package Glass;
with 'Breakable';
has [qw(material height form)] => ...;
```

Required methods

```
package Breakable;

use Moose::Role;

requires 'break';

has is_broken => (is => 'rw', isa => 'Bool');
```

Role application

```
package LotsOfMoney {
  use Moose::Role;
  has lots_of_money => ( isa => 'Bool', default => 1 );
}

### COMPILE-TIME ROLE APPLICATION ###
package MyGirlfriend { with 'LotsOfMoney'; ... }

my $girlfriend = MyGirlfriend->new;

### RUNTIME ROLE APPLICATION ###
package MyGirlfriend { ... }

my $girlfriend = MyGirlfriend->new();
LotsOfMoney->meta->apply($girlfriend);

# or
my $girlfriend = MyGirlfriend->with_traits('LotsOfMoney')->new();
```

Roles in real life

```
package DatabaseStuff;

use Moose::Role;
use DBI;

requires => qw(dsn user pass);

has dbh => ( is => 'ro', isa => 'DBI::dbh', lazy_build => 1);

sub _build_dbh {
    DBI->connect($_[0]->dsn, $_[0]->user, $_[0]->pass);
}

sub DEMOLISH { $_[0]->dbh->disconnect; }
```

Method Modifiers

Method modifiers

```
package  FileManager;

has logger => ( is => 'ro', isa => 'Log::Log4perl' );

sub delete_file {
  my ($self, $filename) = @_;
  unlink $filename;
  return $filename;
}

after 'delete_file' => {
    my ($self, $filename) = @_;
    $self->logger->info("Successfully deleted $filename");
}
```

Modifier types

```
before method => sub { ... }

after method => sub { ... }

around method => sub {
    my ($self, $orig) = @_;
    ...
    return $self->$orig(...);
}
```

Method modifiers and roles

```
package Transactional;

use Moose::Role;

before write => sub {
    begin();
};

after write => sub {
    commit();
}
```

Method modifiers and roles

```
package MyDatabase {
    use Moose;

    sub read { ... }
    sub write { ... }
}

package MyDatabase::Simple {
  use Moose;
  extends 'MyDatabase';
  ...
}

package MyDatabase::MultiUser {
  use Moose;
  extends 'MyDatabase';
  with 'Transactional';
  ...
}
```

Method modifiers and roles

```
package Breakable;

use Moose::Role;

has is_broken => (is => 'rw', isa => 'Bool');

requires 'break';

after 'break' => sub {
  print "OMG, I broke!";
  $_[0]->is_broken(1);
}
```

Metaprogramming

Basic Perl 5 metaprogramming

```
$object->can($method_name);

$object->isa($class);

$object->DOES($role_name);
```

Meta object protocol

Class::MOP

- `Class::MOP::Class`
- `Class::MOP::Instance`
- `Class::MOP::Method`
- `Class::MOP::Attribute`
- `... etc.`

Moose

- `Moose::Meta::*`
- `Moose::Meta::Role`
- `Moose::Meta::TypeConstraint`
- `... etc.`

The "meta" class

```
# from a class
package MyClass;
use Moose;
extends 'MyBaseClass' ;
say for MyClass->meta->superclasses;

# from an instance
my $instance = MyClass->new;
say for $instance->meta->superclasses;

# from any class
my $meta = Class::MOP::Class->initialize('AnotherClass');
say for $meta->superclasses;
```

Introspection

Moose::Meta::Class

- `superclasses`
- `get_method_list`
- `get_attribute_list`
- `calculate_all_roles`

Moose::Meta::Role

- `get_roles`
- `get_method_list`
- `get_attribute_list`
- `get_required_method_list`

Moose::Meta::Method

- `name`
- `package_name`
- `body`

Moose::Meta::Attribute

- `is_required`
- `is_lazy`
- `type_constraint`
- `should_coerce`
- `handles`

Manipulation

Moose::Meta::Class

- `Moose::Meta::Class->create($class_name)`
- `$meta->new_object(%params)`
- `$meta->superclasses(@superclasses)`
- `$meta->add_role($role_name)`
- `$meta->add_attribute($attr_name, %params)`
- `$meta->remove_attribute($attr_name)`
- `$meta->add_method($method_name, $method)`
- `$meta->remove_method($method_name)`
- `$meta->add_$type_method_modifier($method_name, $code)`
- `etc ...`

Messing with metaclasses

```
# CUSTOM TRAITS:
# Moose::Meta::Attribute::Custom::Trait::Chained

has some_option => (
  is     => 'ro',
  traits => ['Chained'],
);

# CUSTOM METACLASS:
# Moose::Meta::Attribute::Custom::Trait::Chained

has some_option => (
  is        => 'ro',
  metaclass => 'Chained',
);

# CUSTOM SYNTAX:
# use MooseX::Plugin::ChainedAttributes;
has_chained => ( ... );
```

Conclusion

- The Big Picture
- Plugins
- Stability
- Performance
- Documentation

The Big Picture

- Use attributes to design APIs before implementing logic
- Use lazy attributes, coercion, and method modifiers to reduce method size and separate concerns
- Use roles and delegation to avoid multiple inheritance or "God" objects
- Use default "thin" constructors to create objects that are easy to test and debug

Plugins

- MooseX::ClassAttribute
- MooseX::Singleton
- MooseX::SimpleConfig
- MooseX::Params::Validate
- MooseX::Object::Pluggable
- MooseX::Getopt
- MooseX::Daemonize
- MooseX::LogDispatch

MooseX::Declare

```
use MooseX::Declare;

class BankAccount {

    has 'balance' => ( isa => 'Num', is => 'rw', default => 0 );

    method deposit (Num $amount) {
        $self->balance( $self->balance + $amount );
    }

    method withdraw (Num $amount) {
        my $current_balance = $self->balance();
        $self->balance( $current_balance - $amount );
    }
}
```

How stable is Moose?

- Mature and tested
- Large new projects (Dist::Zilla, Fey::ORM, HTML::FormHandler, Perlanet, KiokuDB)
- Existing infrastructure libraries (Catalyst, HTML::FormFu, Mason, DateTime)
- Used by BBC, Opera Software, Venda, etc ...

Performance

- __PACKAGE__->meta->make_immutable()
- Mouse
- Moo
- Non-Moose implementations:
  - Class::Method::Modifiers
  - Role::Basic

What next?

- Books
  - Modern Perl
  - Intermediate Perl, 3rd ed
- Moose manual
- Moose website (moose.perl.org)
- Talks, articles, blog posts
- Perl 6 Apocalypses and Synopses

Notable talks

- Dave Cross: Evolving Software with Moose
- Curtis "Ovid" Poe: Inheritance vs Roles
- Tomas Doran: Herding a Cat with Antlers
- Shawn Moore: Extending Moose

Alan Kay:

"I invented object oriented programming, I wasn’t thinking of C++"

Thank you!

use Moose;