Introduction to Moose

Bryan Smith
Weds. Jan 26, 2011
Objectives

- What is Moose?
- Why should I use Moose?
- How can I use Moose?
Disclaimers
package Circle;
use Math::Trig;

sub new {
    my $class = shift;
    my $self = { _radius => shift };
    bless $self, $class;
    return $self;
}

sub radius {
    my $self = shift;
    my $radius = shift;
    $self->{ _radius } = $radius if defined( $radius );
    return $self->{ _radius };
}

sub circumference {
    my $self = shift;
    return 2 * pi * $self->radius;
}

sub area {
    my $self = shift;
    return pi * $self->radius * $self->radius;
}
Testing our code

package main;

# Make circle
my $circle = Circle->new( 3 );

# Prints: 3 18.849559215388 28.2743338823081
print $circle->radius . ' ' . $circle->circumference . ' ' . $circle->area . "\n";

# Change radius
$circle->radius( 4 );

# Prints: 4 25.1327412287183 50.2654824574367
print $circle->radius . ' ' . $circle->circumference . ' ' . $circle->area . "\n";
What is good about this code?
What would we like to improve?
What’s right with old-style OOP?

- Semantic
- Compact and encapsulated
- Idiomatic
- Flexibility
What would we like to improve?

- Remove implementation to focus on semantics
  - Semantics buried in implementation
  - Hash container visible in constructor and radius subroutine
  - Constructor parameters are ordered, not named
- Use type checking
- Reduce code (a circle is a very simple concept)
OOP with Moose

package Circle;
use Math::Trig;
use Moose;

has 'radius' => (
    is => 'rw',
    isa => 'Num',
);

sub circumference {
    my $self = shift;
    return 2 * pi * $self->radius;
}

sub area {
    my $self = shift;
    return pi * $self->radius * $self->radius;
}
Testing our code

```perl
package main;

# Make circle
my $circle = Circle->new( radius => 3 );

# Prints: 3 18.849559215388 28.274338823081
print $circle->radius . ' ' . $circle->circumference . ' ' . $circle->area . "\n";

# Change radius
$circle->radius( 4 );

# Prints: 4 25.1327412287183 50.2654824574367
print $circle->radius . ' ' . $circle->circumference . ' ' . $circle->area . "\n";
```
What improvements did we make?

• Remove implementation to focus on semantics:
  – code describes circle, not how OOP is accomplished
  – Self documenting!

• Use type checking: $circle->radius( 'two' );

  Attribute (radius) does not pass the type constraint because:
  Validation failed for 'Num' with value two at ./circle2.plx line 32

• Reduce code: 28 lines -> 18 lines
Introducing Moose

Moose is a complete object system for Perl 5... [W]ith Moose, you define your class declaratively, without needing to know about blessed hashrefs, accessor methods, and so on.

Why object-oriented?

- Encourages orthogonality
- Easy to locate logic
- Focus on semantics, not implementation
- Object-oriented patterns
- Easy to unit test
Why Moose?

- Object-oriented syntax
- Compact
- Type constraints
- Hooks
- Lots of goodies
- Easy to use
Moose Fundamentals

1. Class
2. Attribute
3. Method
4. Role
5. Method modifiers
6. Type
7. Delegation
package Circle;
use Moose;  # Now we have a class

...  

package main;
# Use the class
package Circle;
use Moose;

has 'radius' => (  
is => 'rw',
  isa => 'Num',
);
Method

- Same as old-style OOP
  - But without getter/setter

```perl
package Circle;
use Moose;
...

sub circumference {
    my $self = shift;
    return 2 * pi * $self->radius;
}
```
Role

- Adds functionality to class (like mixin)
- Used to include or require attributes or subroutines
- Can be used as type (i.e., interface)

```perl
package Displayable;
use Moose::Role;

requires 'html';

# Anything that accepts Displayable
# will also accept an Item.
package Item;
use Moose;

has 'html' => (  
is => 'rw',
   isa => 'Str',
);

with 'Displayable';
```
package Breakable;

use Moose::Role;

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

sub break {  
my $self = shift;

print "I broke\n";

$self->is_broken(1);  
}

package Car;

use Moose;

with 'Breakable';

has 'engine' => (  
is  => 'ro',  
isA  => 'Engine',  
);

# ­­­­­­­­­­­­­­­­­­­­­­­­
package main;

my $car = Car->new( engine => Engine->new ) ;

$car->break;

print $car->is_broken ? 'Busted' : 'Still working';

Source: http://search.cpan.org/perldoc?Moose::Manual::Roles#A_SIMPLE_ROLE
Method Modifiers

**Hooks**

• Useful for logging, backups, tracers, processing

```perl
package StrongParagraph;
use Moose;

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

sub to_html {
    my $self = shift;
    print $self->value;
}

before 'to_html' => sub { print '<p>' ;

after 'to_html' => sub { print '</p>' . "\n" ;

around 'to_html' => sub {
    my $orig = shift;
    my $self = shift;

    print '<strong>' ;
    $self->$orig;
    print '</strong>' ;

    package main;
    my $p = StrongParagraph->new( value => 'Hello, World!' );

    # Prints "<p><strong>Hello, World!</strong></p>"
    $p->to_html;
```
Type

- Str, Num, Int, ClassName, RoleName, Ref, ScalarRef, ArrayRef, HashRef, CodeRef, RegexpRef, GlobRef, FileHandle, Object

- Define subtypes

```perl
subtype 'PositiveInt'
  => as 'Int'
  => where { $$_ > 0 }
  => message { "The number you provided, $$_, was not a positive number" }
```

package Organization;
use Moose;

has 'name' => (
    is => 'rw',
    isa => 'Str',
);

has 'members' => (
    is => 'rw',
    isa => 'ArrayRef',
    default => sub { [] }, // Default value is empty array ref
);

# - - - - - - - -
package main;

my $a2pm = Organization->new( name => 'Ann Arbor Perl Mongers' );
push @{ $a2pm->members }, Member->new( name => 'Bryan Smith' );
Delegation

- Proxy methods
Example: Boxes

Boxes

Ann Arbor Perl Mongers (1)
- Consort

Go club (4)
- go-club@umich.edu
- gc-president@umich.edu
- gc-vicepres@umich.edu
- gc-cos@umich.edu
- Consortium

Korean (4)
- Consortium
- Flash cards / Decks
- Online Korean resource
- Practice reading

MPub (9)
- Planner
- Email (Outlook & Gmail)
- Exchange calendar

Wedding (3)
- Documents in git (e.g., budget, guest list, venues)
- Google document: Wedding playlist
- Google document: Wedding to-do
Boxes UML

Box Diagram:
- **Displayable**
  - +html()
- **Item**
  - +html
  - 0..*
- **Category**
  - +title
  - +id
  - +items
  - 1
- **Storage**
  - +load()
  - +store()
Displayable (role)

package Displayable;
use Moose::Role;

requires 'html';

no Moose;
1;
package Item;
use Moose;
use MooseX::Storage;

with Storage('format' => 'YAML', 'io' => 'File');

require 'lib/displayable.pl';

has 'html' => (  
is => 'rw',
  isa => 'Str',
);

with 'Displayable';

no Moose;
__PACKAGE__->meta->make_immutable;
package Category;

use Moose;
use MooseX::Storage;

with Storage('format' => 'YAML', 'io' => 'File');

require 'lib/displayable.pl';

has 'title' => (
  is => 'rw',
  isa => 'Str',
);

has 'id' => (
  is => 'rw',
  isa => 'Str',
);

has 'items' => (
  is => 'rw',
  isa => 'ArrayRef',
  default => sub { [] },
);

no Moose;
__PACKAGE__->meta->make_immutable;
sub saveCategories {
    my @categories = getCategories();

    for $category (@categories) {
        my $filename = 'db/' . $category->id . '.yaml';

        $category->store( $filename );
    }
}

sub getCategories {
    my @categories = ();

    for my $filename ( glob( 'db/*.yaml' ) ) {
        my $category = Category->load($filename);

        push( @categories, $category );
    }

    return @categories;
}
sub printSectionLinks {
    my @categories = getCategories();
    for my $category (@categories) {
        my $anchor = wrap( $category->title, 'a', [[ 'href', '#' . $category->id ]]);
        println( wrap( $anchor, 'li' ) );
    }
}

sub printSections {
    my @categories = getCategories();
    for my $nextCat (@categories) {
        printBox( $nextCat );
    }
}
sub printBox {
    my $category = shift;

    my $items = $category->items;
    my $itemCount = '('. scalar( @{$items} ) . ')';
    my $title = $category->title . ' ' . wrap( $itemCount, 'span' );

    my $header = wrap( $title, 'a', [[ 'name', $category->id ]]);
    $header = wrap( wrap( $header, 'h2' ), 'header' );

    my $section;
    my $items = $category->items;

    for my $item ( @$items ) {
        $section .= wrap( $item->html, 'div', [[ 'class', 'item' ]]);
    }

    println wrap( $header . $section, 'section' );
}
Overkill?

- Easy to maintain
- Composite pattern available
Conclusion

- **What is Moose?:** Object-oriented system for Perl

- **Why should I use Moose?:** Object-oriented for easier maintenance, Moose for easier object-oriented programming in Perl

- **How can I use Moose?:**
  
Notes

• Any original content released under CC0 (no rights reserved)
  [ http://creativecommons.org/choose/zero/ ]

• “Yellow Red Blending” OpenOffice.org theme
  [ http://templates.services.openoffice.org/en/node/184 ]