OO Perl with Moose

Nelo Onyiah

OO Perl with Moose

What is Moose?

OO Perl with Moose

What is Moose?

Moose is a postmodern object system for Perl 5 that takes the tedium out of writing object-oriented Perl. It borrows all the best features from Perl 6, CLOS (LISP), Smalltalk, Java, BETA, OCaml, Ruby and more, while still keeping true to its Perl 5 roots.

OO Perl with Moose

Why Moose?

OO Perl with Moose

Why Moose?

- makes Perl 5 OO both simpler and more powerful

OO Perl with Moose

Why Moose?

- makes Perl 5 OO both simpler and more powerful
- define your class declaratively

OO Perl with Moose

Why Moose?

- makes Perl 5 OO both simpler and more powerful
- define your class declaratively
- offers "sugar" for object construction, attributes, e.t.c

OO Perl with Moose

Why Moose?

- makes Perl 5 OO both simpler and more powerful
- define your class declaratively
- offers "sugar" for object construction, attributes, e.t.c
- don't need to care how they are implemented

OO Perl with Moose

Why Moose?

- makes Perl 5 OO both simpler and more powerful
- define your class declaratively
- offers "sugar" for object construction, attributes, e.t.c
- don't need to care how they are implemented
- concentrate on the logical structure of your classes

OO Perl with Moose

Why Moose?

- makes Perl 5 OO both simpler and more powerful
- define your class declaratively
- offers "sugar" for object construction, attributes, e.t.c
- don't need to care how they are implemented
- concentrate on the logical structure of your classes
- don't need to be a wizard to use it

OO Perl with Moose

Why Moose?

- makes Perl 5 OO both simpler and more powerful
- define your class declaratively
- offers "sugar" for object construction, attributes, e.t.c
- don't need to care how they are implemented
- concentrate on the logical structure of your classes
- don't need to be a wizard to use it
- but if you are, lets you dig about in the guts and extend it

OO Perl with Moose

```
package Person;
1;
```

To make a class you start with a package

OO Perl with Moose

```
package Person;
use Moose;

1;
```

To make a class you start with a package and just use Moose

OO Perl with Moose

```
package Person;
use Moose;

1;
```

This is a complete class definition

OO Perl with Moose

```
package Person;
use Moose;

1;
```

This is a complete class definition

not terribly useful though

OO Perl with Moose

```
package Person;
use Moose;

1;
```

Under the hood Moose is doing a lot

OO Perl with Moose

```
package Person;
use Moose;

1;
```

Under the hood Moose is doing a lot

(won't go into that though)

OO Perl with Moose

```
package Person;
use Moose;

1;
```

Classes have zero or more attributes

OO Perl with Moose

```
package Person;
use Moose;

has 'birth_date' => (
);

1;
```

Attributes are declared using the has function

OO Perl with Moose

```
package Person;
use Moose;

has 'birth_date' => (
);

1;
```

Attributes are declared using the has function

Attributes have properties

OO Perl with Moose

```
package Person;
use Moose;

has 'birth_date' => (
);

1;
```

Attributes are declared using the has function

Attributes have properties

probably the most powerful feature of Moose

OO Perl with Moose

```
package Person;
use Moose;

has 'birth_date' => (
);

1;
```

Can be provided with accessors

OO Perl with Moose

```
package Person;
use Moose;

has 'birth_date' => (
    is => 'rw',
);

1;
```

Can be provided with accessors by stating that attribute is read-writeable

OO Perl with Moose

```
package Person;
use Moose;

has 'birth_date' => (
    is => 'ro',
);

1;
```

or read-only

OO Perl with Moose

or you can provide your own reader and/or writer

```perl
package Person;
use Moose;

has 'birth_date' => (
    is => 'ro',
    writer => '_set_birth_date',
);

1;
```

OO Perl with Moose

```perl
package Person;
use Moose;

has 'birth_date' => (
    is => 'ro',
    isa => 'Str',
);

1;
```

You can specify a type for your attribute

OO Perl with Moose

```
package Person;
use Moose;

has 'birth_date' => (
    is => 'ro',
    isa => 'Str',
);

1;
```

Only values that pass the type check will be accepted for the attribute

OO Perl with Moose

```
package Person;
use Moose;

has 'birth_date' => (
    is => 'ro',
    isa => 'Str',
);

1;
```

Built in types include:

- Str
- Num
- ArrayRef
- CodeRef
- Any
- more ...

OO Perl with Moose

```
package Person;
use Moose;
use DateTime;

has 'birth_date' => (
    is => 'ro',
    isa => 'DateTime',
);

1;
```

Class names are treated as types

OO Perl with Moose

```
package Person;
use Moose;
use Moose::Util::TypeConstraints;
use DateTime;

subtype 'ModernDateTime'

has 'birth_date' => (
    is => 'ro',
    isa => 'DateTime',
);

1;
```

You can create your own types

OO Perl with Moose

```
package Person;
use Moose;
use Moose::Util::TypeConstraints;
use DateTime;

subtype 'ModernDateTime'
    => as 'DateTime'

has 'birth_date' => (
    is => 'ro',
    isa => 'DateTime',
);

1;
```

You can create your own types

from existing types

OO Perl with Moose

```
package Person;
use Moose;
use Moose::Util::TypeConstraints;
use DateTime;

subtype 'ModernDateTime'
    => as 'DateTime'
    => where { $_->year >= 2000 };

has 'birth_date' => (
    is => 'ro',
    isa => 'DateTime',
);

1;
```

You can create your own types

from existing types

and apply your own constraints on them

OO Perl with Moose

```
package Person;
use Moose;
use Moose::Util::TypeConstraints;
use DateTime;

subtype 'ModernDateTime'
    => as 'DateTime'
    => where { $_->year >= 2000 };

has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
);

1;
```

You can create your own types

from existing types

and apply your own constraints on them

and use them

OO Perl with Moose

```
package Person;
use Moose;
use Moose::Util::TypeConstraints;
use DateTime;

subtype 'ModernDateTime'
    => as 'DateTime'
    => where { $_->year >= 2000 };

has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
);

1;
```

You can also coerce one type into another

OO Perl with Moose

```
package Person;
use Moose;
use Moose::Util::TypeConstraints;
use DateTime;

subtype 'ModernDateTime'
    => as 'DateTime'
    => where { $_->year >= 2000 };

has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
);

1;
```

See Moose::Manual::Types for more details

OO Perl with Moose

```
package Person;
use Moose;
use Moose::Util::TypeConstraints;
use DateTime;

subtype 'ModernDateTime'
    => as 'DateTime'
    => where { $_->year >= 2000 };

has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
);

1;
```

A person with no birth date seems odd

OO Perl with Moose

```
package Person;
use Moose;
use Moose::Util::TypeConstraints;
use DateTime;

subtype 'ModernDateTime'
    => as 'DateTime'
    => where { $_->year >= 2000 };

has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
    required => 1,
);

1;
```

A person with no birth date seems odd

so it can be made compulsary with the required flag

OO Perl with Moose

You can also set default values for the attribute

```
package Person;
use Moose;
use Moose::Util::TypeConstraints;
use DateTime;

subtype 'ModernDateTime'
    => as 'DateTime'
    => where { $_->year >= 2000 };

has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
    required => 1,
    default => '2000-01-01',
);

1;
```

OO Perl with Moose

Complex defaults need to be set in a sub ref

```
package Person;
use Moose;
use Moose::Util::TypeConstraints;
use DateTime;

subtype 'ModernDateTime'
    => as 'DateTime'
    => where { $_->year >= 2000 };

has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
    required => 1,
    default => sub { DateTime->now },
);

1;
```

OO Perl with Moose

or you could write a separate builder method

```
package Person;
use Moose;

# subtype ...
has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
    required => 1,
    builder => '_build_birth_date',
);

sub _build_birth_date {
    DateTime->now;
}

1;
```

OO Perl with Moose

```
package Person;
use Moose;

# subtype ...
has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
    required => 1,
    builder => '_build_birth_date',
    lazy => 1,
);

sub _build_birth_date {
    DateTime->now;
}

1;
```

and make it lazy

OO Perl with Moose

```
package Person;
use Moose;

# subtype ...
has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
    required => 1,
    lazy_build => 1,
);

sub _build_birth_date {
    DateTime->now;
}

1;
```

or in one step

OO Perl with Moose

Attributes *handle* delegation

```
package Person;
use Moose;

# subtype ...
has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
    required => 1,
    lazy_build => 1,
    handles => { birth_year => 'year' },
);

sub _build_birth_date {
    DateTime->now;
}

1;
```

OO Perl with Moose

```
package Person;
use Moose;

# subtype ...
has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
    required => 1,
    lazy_build => 1,
    handles => { birth_year => 'year' },
);

sub _build_birth_date {
    DateTime->now;
}

1;
```

Attributes *handle* delegation

Calling `$person->birth_year` delegates to `$person->birth_date->year`

OO Perl with Moose

Delegation is one option to inheritance

```
package Person;
use Moose;

# subtype ...
has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
    required => 1,
    lazy_build => 1,
    handles => { birth_year => 'year' },
);

sub _build_birth_date {
    DateTime->now;
}

1;
```

OO Perl with Moose

```
package Person;
use Moose;

# subtype ...
has 'birth_date' => (
    is => 'ro',
    isa => 'ModernDateTime',
    required => 1,
    lazy_build => 1,
    handles => { birth_year => 'year' },
);

sub _build_birth_date {
    DateTime->now;
}

1;
```

Delegation is one option to inheritance

Especially when inheriting from non-Moose based classes

OO Perl with Moose

```
package Employee;
use Moose;
extends qw( Person );

1;
```

Inheritance is achieved with the extends function

OO Perl with Moose

```perl
package Employee;
use Moose;
extends qw( Person );

1;
```

Inheritance is achieved with the extends function

Moose supports multiple inheritance

OO Perl with Moose

```
package Employee;
use Moose;
extends qw( Person );

1;
```

Inheritance is achieved with the extends function

Moose supports multiple inheritance just pass more class names to extends

OO Perl with Moose

Override parent methods with the override function

```
package Employee;
use Moose;
extends qw( Person );

override '_build_birth_date' => sub {
    # ...
}
1;
```

OO Perl with Moose

Call the parent method with the super function

```
package Employee;
use Moose;
extends qw( Person );

override '_build_birth_date' => sub {
    # ...

    super();
}
1;
```

OO Perl with Moose

```
package Employee;
use Moose;
extends qw( Person );

has '+birth_date' => (
    # ...
);
1;
```

You can also override attributes

OO Perl with Moose

```
package Employee;
use Moose;
extends qw( Person );

has '+birth_date' => (
    # ...
);
1;
```

You can also override attributes (carefully)

OO Perl with Moose

```
package Science;

1;
```

Moose also has a concept of roles

OO Perl with Moose

```
package Science;
use Moose::Role;

1;
```

Moose also has a concept of roles

Declared by using Moose::Role

OO Perl with Moose

```
package Science;
use Moose::Role;

1;
```

Similar to Smalltalk traits, Ruby Mixins and Java interfaces

OO Perl with Moose

```
package Science;
use Moose::Role;

1;
```

Similar to Smalltalk traits, Ruby Mixins and Java interfaces

Most similar to Perl 6 Roles

OO Perl with Moose

```
package Science;
use Moose::Role;

1;
```

A collection of reusable traits (attributes)

OO Perl with Moose

```
package Science;
use Moose::Role;

has 'speciality' => (
    # ...
);
1;
```

A collection of reusable traits (attributes)

OO Perl with Moose

```
package Science;
use Moose::Role;

has 'speciality' => (
    # ...
);

sub research {
    # ...
}
1;
```

A collection of reusable traits (attributes) and behaviour (methods)

OO Perl with Moose

```
package Science;
use Moose::Role;

has 'speciality' => (
    # ...
);

sub research {
    # ...
}
1;
```

Roles are not classes

OO Perl with Moose

```
package Science;
use Moose::Role;

has 'speciality' => (
    # ...
);

sub research {
    # ...
}
1;
```

Roles are not classes

- cannot instantiate a role

OO Perl with Moose

```
package Science;
use Moose::Role;

has 'speciality' => (
    # ...
);

sub research {
    # ...
}
1;
```

Roles are not classes

- cannot instantiate a role
- cannot inherit from a role

OO Perl with Moose

```
package Science;
use Moose::Role;

has 'speciality' => (
    # ...
);

sub research {
    # ...
}
1;
```

Roles are another option to inheritance

OO Perl with Moose

```
package Science;
use Moose::Role;

has 'speciality' => (
    # ...
);

sub research {
    # ...
}
1;
```

Roles are composed into consuming classes/roles

OO Perl with Moose

```
package Science;
use Moose::Role;

has 'speciality' => (
    # ...
);

sub research {
    # ...
}
1;
```

Roles are composed into consuming classes/roles

Attributes and methods from role are flattened into consuming class/role

OO Perl with Moose

```
package Science;
use Moose::Role;

has 'speciality' => (
    # ...
);

sub research {
    # ...
}
1;
```

Roles can insist that consuming classes implement certain methods

OO Perl with Moose

```
package Science;
use Moose::Role;

requires qw( research );
has 'speciality' => (
    # ...
);
1;
```

Roles can insist that consuming classes implement certain methods

Use the requires function

OO Perl with Moose

```
package Science;
use Moose::Role;

requires qw( research );
has 'speciality' => (
    # ...
);
1;
```

Roles can insist that consuming classes implement certain methods

Use the requires function

Consuming classes must now implement the research function

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
1;
```

Roles are consumed into classes by using the with keyword

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
1;
```

Roles are consumed into classes by using the with keyword

More than one role can be consumed into a class

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
1;
```

Roles are consumed into classes by using the with keyword

More than one role can be consumed into a class just pass more roles to with

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
1;
```

Class methods and attributes are prioritized

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
1;
```

Class methods and attributes are prioritized

Conflicts are resolved at compile time

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
1;
```

See Moose::Manual::Roles for details

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
1;
```

Moose is not perfect

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
1;
```

Moose is not perfect

Biggest caveat is start up time

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
1;
```

Moose is not perfect

Biggest caveat is start up time

Actively being worked on

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
1;
```

Moose is not perfect

Biggest caveat is start up time

Actively being worked on

But you can help

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
__PACKAGE__->meta->make_immutable();
1;
```

Make your classes immutable

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
__PACKAGE__->meta->make_immutable();
1;
```

Make your classes immutable

This lets Moose create an inline constructor for your class

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
__PACKAGE__->meta->make_immutable();
1;
```

Make your classes immutable

This lets Moose create an inline constructor for your class

Greatly speeding up start up time

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
__PACKAGE__
    ->meta->make_immutable();
1;
```

Also you are adviced to clean up after your class

i.e remove all Moose sugar from packages using your classes

OO Perl with Moose

```
package Scientist;
use Moose;
extends qw( Person );
with qw( Science );

sub research { ... }
__PACKAGE__
    ->meta->make_immutable();
no Moose;
1;
```

Also you are adviced to clean up after your class

i.e remove all Moose sugar from packages using your classes

OO Perl with Moose

```
package Scientist;
use Moose;
use namespace::clean
    -except => [qw( meta )];
extends qw( Person );
with qw( Science );

sub research { ... }
__PACKAGE__
    ->meta->make_immutable();
1;
```

Also you are adviced to clean up after your class

i.e remove all Moose sugar from packages using your classes

Alternatively

OO Perl with Moose

- Moose is also extensible

OO Perl with Moose

- Moose is also extensible
- Done by manipulating metaclass objects

OO Perl with Moose

- Moose is also extensible
- Done by manipulating metaclass objects
- This is where the wizards roam free

OO Perl with Moose

- Moose is also extensible
- Done by manipulating metaclass objects
- This is where the wizards roam free
- A lot of extensions exist in the MooseX:: namespace

OO Perl with Moose

- Moose is also extensible
- Done by manipulating metaclass objects
- This is where the wizards roam free
- A lot of extensions exist in the MooseX:: namespace
- New ideas usually start life here

OO Perl with Moose

- Moose is also extensible
- Done by manipulating metaclass objects
- This is where the wizards roam free
- A lot of extensions exist in the MooseX:: namespace
- New ideas usually start life here
- Good ones get incorporated into Moose

OO Perl with Moose

- Moose is also extensible
- Done by manipulating metaclass objects
- This is where the wizards roam free
- A lot of extensions exist in the MooseX:: namespace
- New ideas usually start life here
- Good ones get incorporated into Moose
- An example is MooseX::AttributeHelpers

OO Perl with Moose

- Moose is also extensible
- Done by manipulating metaclass objects
- This is where the wizards roam free
- A lot of extensions exist in the MooseX:: namespace
- New ideas usually start life here
- Good ones get incorporated into Moose
- An example is MooseX::AttributeHelpers
- These were incorporated in the Moose::Meta::Attribute::Native namespace

OO Perl with Moose

```
package Person;
use Moose;
use namespace::clean
    -except => [qw( meta )];
# attributes and methods

__PACKAGE__->meta->make_immutable();
1;
```

Lastly you will note that Moose introduces its own boiler plate code

OO Perl with Moose

```
package Person;
use Moose;
use namespace::clean
    -except => [qw( meta )];
# attributes and methods

__PACKAGE__->meta->make_immutable();
1;
```

Lastly you will note that Moose introduces its own boiler plate code

There is an extension that reduces this

OO Perl with Moose

```
package Person;
use Moose;
use namespace::clean
    -except => [qw( meta )];
# attributes and methods

__PACKAGE__->meta->make_immutable();
1;
```

Lastly you will note that Moose introduces its own boiler plate code

There is an extension that reduces this

MooseX::Declare

OO Perl with Moose

```
use MooseX::Declare;

class Person {
    # attributes and methods
}
```

Declaring classes becomes *even* more declarative

OO Perl with Moose

```
use MooseX::Declare;

class Person {
    # attributes and methods
}
```

Combines the power of Moose with Devel::Declare to produce keywords for Perl 5 written in Perl 5

OO Perl with Moose

```
use MooseX::Declare;

class Person {
    # attributes and methods
    method research() { ... }
}
```

Combines the power of Moose with Devel::Declare to produce keywords for Perl 5 written in Perl 5

Keywords include class, role, method

OO Perl with Moose

```
use MooseX::Declare;

class Person {
    # attributes and methods
    method research() { ... }
}
```

Combines the power of Moose with Devel::Declare to produce keywords for Perl 5 written in Perl 5

Keywords include class, role, method

Note that the methods have signatures complete with type checking

OO Perl with Moose

```
use MooseX::Declare;

class Person {
    # attributes and methods
    method research() { ... }
}
```

Combines the power of Moose with Devel::Declare to produce keywords for Perl 5 written in Perl 5

Keywords include class, role, method

Note that the methods have signatures complete with type checking

So using MooseX::Declare the Scientist class example could look like the following:

OO Perl with Moose

```
use MooseX::Declare;

class Scientist extends Person with Science {
    use Duration; # fictional class one could write
    has 'funding' => (
        is  => 'rw',
        isa => 'Num',
        lazy_build => 1,
    );
    method research( Duration $contract_duration ) {
        unless ( $self->has_funding ) {
            confess 'need funding to work';
        }
        while ( not $contract_duration->expired ) {
            # do your research ...
        }
    }
}
```

Thank you

Questions?