Meta Object Protocol (MOP)

Upasana

me@upasana.me

About me

- Software developer at booking.com

We are hiring

https://workingatbooking.com/

Backstory

- GNOME Outreach Program for Women internship in 2013
- Structured exceptions in Moose
- Want to share whatever I learnt during my internship

Topics

1) Little bit about object oriented programming (OOP)
2) Little bit about OOP in perl (the old style)
3) Meta object protocol (MOP)
4) History of MOP
5) Applications of MOP

Topics

6) Implementing MOP in Perl (the easy way)
7) Why #6 might not be a good idea
8) Metaclass incompatibility
9) Mixins
10) MOP in Moose
11) Drawbacks of MOP
12) Where to go next?

How a class looks like?

- Class name
- Superclasses

How a class looks like?

- Attributes
  - is read only or read-write
  - type (int, float etc.)
  - default value if any
  - getter method (accessor)
  - setter method (mutator)

How a class looks like?

- Methods
  - method name
  - body

Classes in Perl

- Perl doesn't provide any special syntax for classes
- Perl packages are classes

Attributes in Perl classes

- No special syntax or support for declaring and manipulating attributes
- Attributes are stored in the object itself
- As a hash of key-value pairs

Object?

- A hash reference
- blessed into a class

OOP in Perl

```
package Rectangle;
sub new {
    my $self       = shift;
    my $attributes = { @_ };
    bless $attributes, $self;
}

Rectangle->new(
    height => 10,
    width  => 20,
);
```

OOP in Perl

```
package Rectangle;
sub new {
    my $self       = shift;
    my $attributes = { @_ };
    bless $attributes, $self;
}

Rectangle->new(
    height => 10,
    width  => 20,
);
```

OOP in Perl

```
package Rectangle;
sub new {
    my $self       = shift;
    my $attributes = { @_ };
    bless $attributes, $self;
}

Rectangle->new(
    height => 10,
    width  => 20,
);
```

OOP in Perl

```
package Rectangle;
sub new {
    my $self       = shift;
    my $attributes = { @_ };
    bless $attributes, $self;
}

Rectangle->new(
    height => 10,
    width  => 20,
);
```

OOP in Perl

```
package Rectangle;
sub new {
    my $self       = shift;
    my $attributes = { @_ };
    bless $attributes, $self;
}

Rectangle->new(
    height => 10,
    width  => 20,
);
```

OOP in Perl

```
package Rectangle;
sub new {
    my $self       = shift;
    my $attributes = { @_ };
    bless $attributes, $self;
}

Rectangle->new(
    height => 10,
    width  => 20,
);
```

OOP in Perl

```
package Rectangle;
sub new {
    my $self       = shift;
    my $attributes = { @_ };
    bless $attributes, $self;
}

Rectangle->new(
    height => 10,
    width  => 20,
);
```

What is MOP?

- provides the vocabulary to access and manipulate the structure and behavior of objects.

Functions of MOP

- Creating new classes
- Deleting existing classes
- Changing the class structure
- Changing methods of the class
- At runtime

History of MOP

- First introduced in the Smalltalk
- Common LISP Object System (CLOS) was influenced by Smalltalk
- CLOS allowed multiple inheritance unlike Smalltalk

MOP in modern languages

- Javascript has Joose
- OpenC++
- Java has Reflection API
- **Perl has Moose**

Why do we need a MOP?

Testing

Testing

- I work at booking.com
- Our website is moving very fast
- Many rollouts in a day

Testing

- We don't test everything
- At one point, rollouts used to be hard
- Some things need to be tested manually

Testing

```
package Web::Handler {
    has 'search' => (
        url => '/search', #...
    );
    has 'hotel' => (
        url => '/hotel', #...
    );
    # ...
}
```

Testing

```
package Web::Handler {
    has 'search' => (
        url => '/search', #...
    );
    has 'hotel' => (
        url => '/hotel', #...
    );
    # ...
}
```

Testing

```
package Web::Handler {
    has 'search' => (
        url => '/search', #...
    );
    has 'hotel' => (
        url => '/hotel', #...
    );
    # ...
}
```

Introspection

- Give me all the attributes of Web::Handler.
- Run tests for all the attributes.

Testing

```
# This is pseudocode, don't expect this
# to compile
my $attr =
    Web::Handler->meta->get_attributes_list;
foreach my $a ( @$attr ) {
    next unless $a->attribute_exists('url');
    my $url = $a->get_attribute('url');
    die “test fails...\n”
        if( !LWP::Simple::get($url) );
}
```

Testing

```
# This is pseudocode, don't expect this
# to compile
my $attr =
    Web::Handler->meta->get_attributes_list;
foreach my $a ( @$attr ) {
    next unless $a->attribute_exists('url');
    my $url = $a->get_attribute('url');
    die “test fails...\n”
        if( !LWP::Simple::get($url) );
}
```

Testing

```
# don't expect this to compile
my $attr =
    Web::Handler->meta->get_attributes_list;
foreach my $a ( @$attr ) {
    next unless $a->attribute_exists('url');
    my $url = $a->get_attribute('url');
    die “test fails...\n”
        if( !LWP::Simple::get($url) );
}
```

Testing

```
# don't expect this to compile
my $attr =
    Web::Handler->meta->get_attributes_list;
foreach my $a ( @$attr ) {
    next unless $a->attribute_exists('url');
    my $url = $a->get_attribute('url');
    die “test fails...\n”
        if( !LWP::Simple::get($url) );
}
```

Testing

```
# don't expect this to compile
my $attr =
    Web::Handler->meta->get_attributes_list;
foreach my $a ( @$attr ) {
    next unless $a->attribute_exists('url');
    my $url = $a->get_attribute('url');
    die “test fails...\n”
        if( !LWP::Simple::get($url) );
}
```

Testing

```
# don't expect this to compile
my $attr =
    Web::Handler->meta->get_attributes_list;
foreach my $a ( @$attr ) {
    next unless $a->attribute_exists('url');
    my $url = $a->get_attribute('url');
    die “test fails...\n”
        if( !LWP::Simple::get($url) );
}
```

Object Relational Mapping (ORM)

ORM

```
my $create_table_statement =<<END;
CREATE TABLE Hotel (
    id INT PRIMARY KEY,
    name VARCHAR(255),
    address VARCHAR(255)
);
END
```

ORM

```
my $sql_parser =
    SQL::Parser->new( $create_table_statement );

my $class_name = $sql_parser->table_name;

my $c =
    Moose::Meta::Class->create( $class_name );
$c->set_superclass( 'SomeDB::Class::Thing' );
```

ORM

```
my $sql_parser =
    SQL::Parser->new( $create_table_statement );

my $class_name = $sql_parser->table_name;

my $c =
    Moose::Meta::Class->create( $class_name );
$c->set_superclass( 'SomeDB::Class::Thing' );
```

ORM

```
my $sql_parser =
    SQL::Parser->new( $create_table_statement );

my $class_name = $sql_parser->table_name;

my $c =
    Moose::Meta::Class->create( $class_name );
$c->set_superclass( 'SomeDB::Class::Thing' );
```

ORM

```
my $sql_parser =
    SQL::Parser->new( $create_table_statement );

my $class_name = $sql_parser->table_name;

my $c =
    Moose::Meta::Class->create( $class_name );
$c->set_superclass( 'SomeDB::Class::Thing' );
```

ORM

```
my $sql_parser =
    SQL::Parser->new( $create_table_statement );

my $class_name = $sql_parser->table_name;

my $c =
    Moose::Meta::Class->create( $class_name );
$c->set_superclass( 'SomeDB::Class::Thing' );
```

ORM

```
foreach my $f ( $sql_parser->fields ) {
    my $tc = find_type_constraint( $f->type );
    $c->add_attribute(
        Moose::Meta::Attribute->new(
            $f->name,
            isa    => $tc,
            reader => 'get_' . $f->name,
            writer => 'set_' . $f->name,
        );
    );
}
```

ORM

```
foreach my $f ( $sql_parser->fields ) {
    my $tc = find_type_constraint( $f->type );
    $c->add_attribute(
        Moose::Meta::Attribute->new(
            $f->name,
            isa    => $tc,
            reader => 'get_' . $f->name,
            writer => 'set_' . $f->name,
        );
    );
}
```

ORM

```
foreach my $f ( $sql_parser->fields ) {
    my $tc = find_type_constraint( $f->type );
    $c->add_attribute(
        Moose::Meta::Attribute->new(
            $f->name,
            isa    => $tc,
            reader => 'get_' . $f->name,
            writer => 'set_' . $f->name,
        );
    );
}
```

ORM

```
foreach my $f ( $sql_parser->fields ) {
    my $tc = find_type_constraint( $f->type );
    $c->add_attribute(
        Moose::Meta::Attribute->new(
            $f->name,
            isa    => $tc,
            reader => 'get_' . $f->name,
            writer => 'set_' . $f->name,
        );
    );
}
```

ORM

```
foreach my $f ( $sql_parser->fields ) {
    my $tc = find_type_constraint( $f->type );
    $c->add_attribute(
        Moose::Meta::Attribute->new(
            $f->name,
            isa    => $tc,
            reader => 'get_' . $f->name,
            writer => 'set_' . $f->name,
        );
    );
}
```

ORM

```
foreach my $f ( $sql_parser->fields ) {
    my $tc = find_type_constraint( $f->type );
    $c->add_attribute(
        Moose::Meta::Attribute->new(
            $f->name,
            isa    => $tc,
            reader => 'get_' . $f->name,
            writer => 'set_' . $f->name,
        );
    );
}
```

ORM

```
foreach my $f ( $sql_parser->fields ) {
    my $tc = find_type_constraint( $f->type );
    $c->add_attribute(
        Moose::Meta::Attribute->new(
            $f->name,
            isa    => $tc,
            reader => 'get_' . $f->name,
            writer => 'set_' . $f->name,
        );
    );
}
```

ORM

```
# return me the hotel with id 123
my $h = Hotel->retrieve( 123 );
my $hotel_name = $h->name;
$h->set_name( 'asdfasdf' );
```

ORM

```
# return me the hotel with id 123
my $h = Hotel->retrieve( 123 ); # return me the hotel id 123
my $hotel_name = $h->name;
$h->set_name( 'asdfasdf' );
```

ORM

```
# return me the hotel with id 123
my $h = Hotel->retrieve( 123 ); # return me the hotel id 123
my $hotel_name = $h->name;
$h->set_name( 'asdfasdf' );
```

Implementing MOP in Perl

Creating a class at runtime

- Perl class is a package
- Every package has a symbol table

Symbol table

- Hash of subroutines/variables defined in a package
- package name with two colons appended

  `$Rectangle::`

Symbol table

- Hash of subroutines/variables defined in a package
- package name with two colons appended

```
$Rectangle::
```

```
package Metaclass;

sub create_class {
    my ($self, %options) = @_;
    my $class   = $options{ package };
    my $methods = $options{ methods };

    while( my ($method, $body) = each( %$methods ) ) {
        no strict 'refs';
        *{ "${class}::$method" } = $body;
    } # end while loop
}

1;
```

```
package Metaclass;

sub create_class {
    my ($self, %options) = @_;
    my $class   = $options{ package };
    my $methods = $options{ methods };

    while( my ($method, $body) = each( %$methods ) ) {
        no strict 'refs';
        *{ "${class}::$method" } = $body;
    } # end while loop
}

1;
```

```
package Metaclass;

sub create_class {
    my ($self, %options) = @_;
    my $class   = $options{ package };
    my $methods = $options{ methods };

    while( my ($method, $body) = each( %$methods ) ) {
        no strict 'refs';
        *{ "${class}::$method" } = $body;
    } # end while loop
}

1;
```

```
package Metaclass;

sub create_class {
    my ($self, %options) = @_;
    my $class   = $options{ package };
    my $methods = $options{ methods };

    while( my ($method, $body) = each( %$methods ) ) {
        no strict 'refs';
        *{ "${class}::$method" } = $body;
    } # end while loop
}

1;
```

```
package Metaclass;

sub create_class {
    my ($self, %options) = @_;
    my $class   = $options{ package };
    my $methods = $options{ methods };

    while( my ($method, $body) = each( %$methods ) ) {
        no strict 'refs';
        *{ "${class}::$method" } = $body;
    } # end while loop
}

1;
```

```
package Metaclass;

sub create_class {
    my ($self, %options) = @_;
    my $class   = $options{ package };
    my $methods = $options{ methods };

    while( my ($method, $body) = each( %$methods ) ) {
        no strict 'refs';
        *{ "${class}::$method" } = $body;
    } # end while loop
}

1;
```

```
package Metaclass;

sub create_class {
    my ($self, %options) = @_;
    my $class   = $options{ package };
    my $methods = $options{ methods };

    while( my ($method, $body) = each( %$methods ) ) {
        no strict 'refs';
        *{ "${class}::$method" } = $body;
    } # end while loop
}

1;
```

```
package Metaclass;

sub create_class {
    my ($self, %options) = @_;
    my $class   = $options{ package };
    my $methods = $options{ methods };

    while( my ($method, $body) = each( %$methods ) ) {
        no strict 'refs';
        *{ "${class}::$method" } = $body;
    } # end while loop
}

1;
```

```
Metaclass->create_class(
    package => 'Rectangle',
    methods => {
        new => sub {
            my ($self)     = shift;
            my $attributes = { @_ };
            return bless $attributes, $self;
        },
    }
);

Rectangle->new(
    height => 10,
    Width  => 20
);
```

```
Metaclass->create_class(
    package => 'Rectangle',
    methods => {
        new => sub {
            my ($self)     = shift;
            my $attributes = { @_ };
            return bless $attributes, $self;
        },
    }
);

Rectangle->new(
    height => 10,
    Width  => 20
);
```

```
Metaclass->create_class(
    package => 'Rectangle',
    methods => {
        new => sub {
            my ($self)     = shift;
            my $attributes = { @_ };
            return bless $attributes, $self;
        },
    }
);

Rectangle->new(
    height => 10,
    Width  => 20
);
```

```
Metaclass->create_class(
    package => 'Rectangle',
    methods => {
        new => sub {
            my ($self)     = shift;
            my $attributes = { @_ };
            return bless $attributes, $self;
        },
    }
);

Rectangle->new(
    height => 10,
    Width  => 20
);
```

```
Metaclass->create_class(
    package => 'Rectangle',
    methods => {
        new => sub {
            my ($self)     = shift;
            my $attributes = { @_ };
            return bless $attributes, $self;
        },
    }
);

Rectangle->new(
    height => 10,
    Width  => 20
);
```

```
sub create_class {
    my ($self, %options) = @_;
    my $class   = $options{ package };
    $options{ methods }->{ meta } = \&get_meta;

    my $methods = $options{ methods };

    while( my ($method, $body) = each( %$methods ) ) {
        no strict 'refs';
        *{ "${class}::$method" } = $body;
    } # end while loop
}
```

```
my %meta_to_class;

sub get_meta {
    my $class = shift;
    Metaclass->get_metaclass( $class );
};

sub get_metaclass {
    my $class = shift;
    return bless $meta_to_class{ $_[ 0 ] }, $class;
}
```

```
my %meta_to_class;

sub get_meta {
    my $class = shift;
    Metaclass->get_metaclass( $class );
};

sub get_metaclass {
    my $class = shift;
    return bless $meta_to_class{ $_[ 0 ] }, $class;
}
```

```
my %meta_to_class;

sub get_meta {
    my $class = shift;
    Metaclass->get_metaclass( $class );
};

sub get_metaclass {
    my $class = shift;
    return bless $meta_to_class{ $_[ 0 ] }, $class;
}
```

```
my %meta_to_class;

sub get_meta {
    my $class = shift;
    Metaclass->get_metaclass( $class );
};

sub get_metaclass {
    my $class = shift;
    return bless $meta_to_class{ $_[ 0 ] }, $class;
}
```

```
my %meta_to_class;

sub get_meta {
    my $class = shift;
    Metaclass->get_metaclass( $class );
};

sub get_metaclass {
    my $class = shift;
    return bless $meta_to_class{ $_[ 0 ] }, $class;
}
```

```
my %meta_to_class;

sub get_meta {
    my $class = shift;
    Metaclass->get_metaclass( $class );
};

sub get_metaclass {
    my $class = shift;
    return bless $meta_to_class{ $_[ 0 ] }, $class;
}
```

```
sub create_class {
    my ($self, %options) = @_;
    my $class   = $options{ package };
    $options{ methods }->{ meta } = \&get_meta;

    my $methods = $options{ methods };
    no strict 'refs';
    while( my ($method, $body) = each( %$methods ) ) {
        *{ "${class}::$method" } = $body;
    } # end while loop
    use strict;

    set_metaclass( $class, \%options );
}
```

```
my %meta_to_class;

sub get_meta {
    my $class = shift;
    Metaclass->get_metaclass( $class );
};

sub get_metaclass {
    my $class = shift;
    return bless $meta_to_class{ $_[ 0 ] }, $class;
}

sub set_metaclass {
    $meta_to_class{ $_[ 0 ] } = $_[ 1 ];
}
```

```
my %meta_to_class;

sub get_meta {
    my $class = shift;
    Metaclass->get_metaclass( $class );
};

sub get_metaclass {
    my $class = shift;
    return bless $meta_to_class{ $_[ 0 ] }, $class;
}

sub set_metaclass {
    $meta_to_class{ $_[ 0 ] } = $_[ 1 ];
}
```

```
my %meta_to_class;

sub get_meta {
    my $class = shift;
    Metaclass->get_metaclass( $class );
};

sub get_metaclass {
    my $class = shift;
    return bless $meta_to_class{ $_[ 0 ] }, $class;
}

sub set_metaclass {
    $meta_to_class{ $_[ 0 ] } = $_[ 1 ];
}
```

Introspection

```
Metaclass->create_class(
    package => 'Rectangle',
    methods => {
        new => sub {
            my ($self)     = shift;
            my $attributes = { @_ };
            return bless $attributes, $self;
        },
    },
);
print Dumper( Rectangle->meta );
```

```
bless({
    'package' => 'Rectangle',
    'methods' => {
        'meta' => sub { "DUMMY" },
        'new'  => sub { "DUMMY" }
    }
}, 'Metaclass' );
```

```
bless({
    'package' => 'Rectangle',
    'methods' => {
        'meta' => sub { "DUMMY" },
        'new'  => sub { "DUMMY" }
    }
}, 'Metaclass' );
```

Inheritance

- Every package's symbol table has an array named ISA
- `@PackageName::ISA`

Inheritance

```
if( $options{ superclasses } && @{$options{ superclasses }} )
{
    @{"${class}::ISA"} = @{$options{ superclasses }}
}
```

Inheritance

```
if( $options{ superclasses } && @{$options{ superclasses }} )
{
    @{"${class}::ISA"} = @{$options{ superclasses }};
}
```

```
Metaclass->create_class(
    package      => 'ColoredRectangle',
    superclasses => [ 'Rectangle' ],
);
```

```
Metaclass->create_class(
    package      => 'ColoredRectangle',
    superclasses => [ 'Rectangle' ],
);
```

And it works, I can do

```
ColoredRectangle->new();
```

But please don't try aforementioned things

It's incomplete & may be fragile

But why?

“Manipulating stashes (Perl's symbol tables) is occasionally necessary, but incredibly messy, and easy to get wrong. This module hides all of that behind a simple API.”

`man Package::Stash`

But why?

- `use Package::Stash;`
- `use Symbol::Table;`

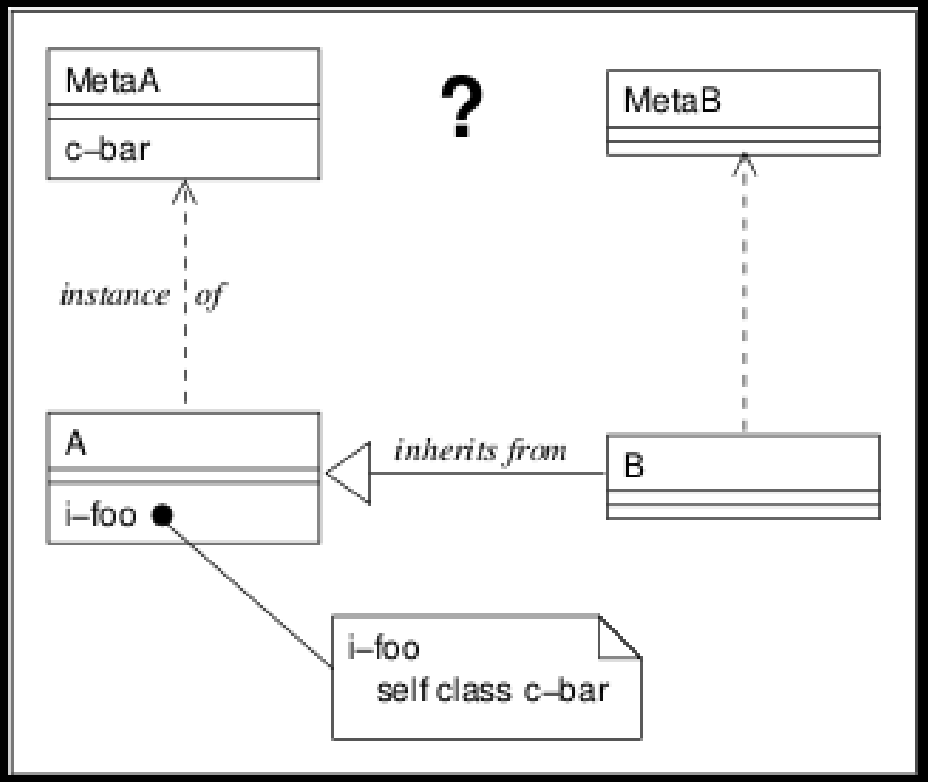
But why?

- Metaclass.pm is very basic
- But actually Metaclasses are not so simple
- Look at Moose

Moose

- Metaclasses for attributes
- Metaclasses for methods

Inheritance & metaclass compatibility



- A has a method i-foo
  - Calls c-bar of MetaA
- B inherits from A
  - B has i-foo
- MetaB may not have c-bar

Inheritance & metaclass compatibility

```
package MetaA;
....
sub c_bar {
    print "in c_bar\n";
}
1;
```

Inheritance & metaclass compatibility

```
MetaA->create_class(
    package => 'A',
    methods => {
        new => sub {
            my ($self)     = shift;
            my $attributes = { @_ };
            return bless $attributes, $self;
        },
        i_foo => sub {
            my ($self) = shift;
            my $meta   = $self->meta;
            $meta->c_bar;
        },
    },
);
A->i_foo();
```

Inheritance & metaclass compatibility

```
MetaB->create_class(
    package => 'B',
    methods => {
        new => sub {
            my ($self)     = shift;
            my $attributes = { @_ };
            return bless $attributes, $self;
        },
    },
    superclasses => [ 'A' ],
);

B->i_foo;
```

Inheritance & metaclass compatibility

```
Can't locate object method "c_bar" via package
"MetaB" at test.pl line 24.
```

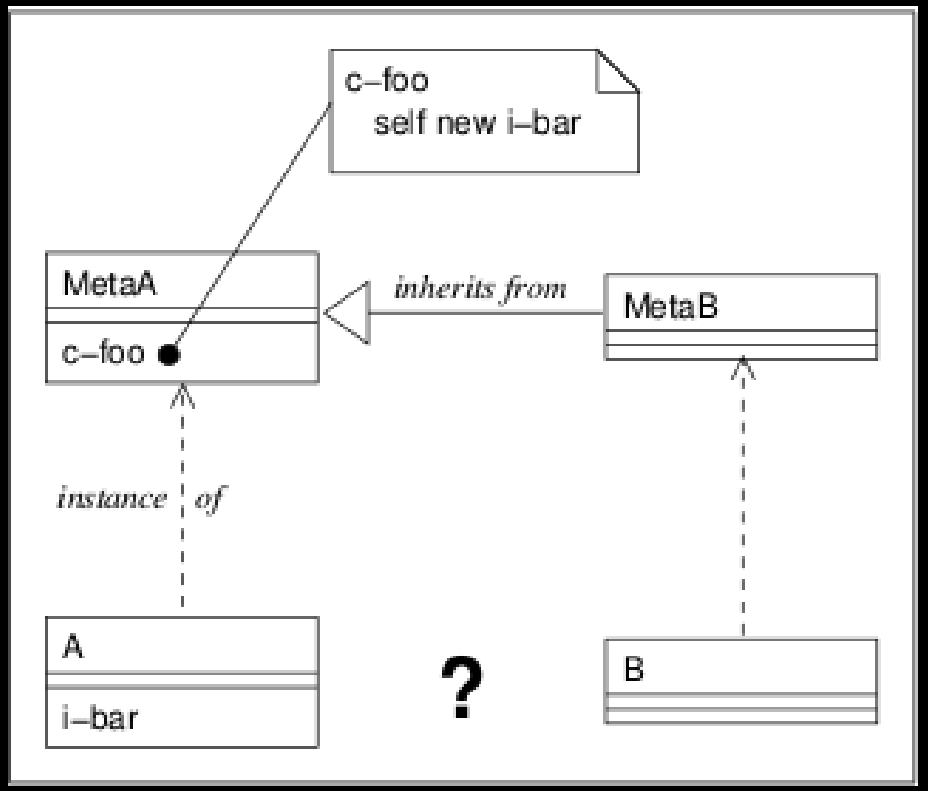
Inheritance & metaclass compatibility

```
i_foo => sub {
    my ($self) = shift;
    my $meta   = $self->meta;
    $meta->c_bar;
},
```

Inheritance & metaclass compatibility

```
package MetaB;
….
# NO c_bar
1;
```

Inheritance & metaclass compatibility



- MetaA has a method c-foo
- c-foo needs to call i-bar in A
- MetaB inherits from MetaA
- B has to has i-bar

Inheritance & metaclass compatibility

```
package MetaA;
.....
sub c_foo {
    my ( $self, $child ) = @_;
    $child->i_bar;
}
1;
```

Inheritance & metaclass compatibility

```
package MetaB;
use strict;
use warnings;

use parent 'MetaA';
1;
```

Inheritance & metaclass compatibility

```
MetaA->create_class(
    package => 'A',
    methods => {
        new => sub {
            my ($self)     = shift;
            my $attributes = { @_ };
            return bless $attributes, $self;
        },
        i_bar => sub {
            print "in i_bar\n";
        },
    },
);

MetaA->c_foo( 'A' );
```

Inheritance & metaclass compatibility

```
MetaB->create_class(
    package => 'B',
    methods => {
        new => sub {
            my ($self)     = shift;
            my $attributes = { @_ };
            return bless $attributes, $self;
        },
        # NO i_bar
    },
);

MetaB->c_foo( 'B' );
```

Inheritance & metaclass compatibility

```
Can't locate object method "i_bar" via package
"B" at MetaA.pm line 16.
```

Inheritance & metaclass compatibility

```
sub c_foo {
    my ( $self, $child ) = @_;
    $child->i_bar;
}
```

Metaclass Incompatibility

- Various ways of dealing with this

Metaclass compatibility (Moose)

- Does parent & child metaclasses have any common ancestors?
  - If yes, then \o/
  - else, die
- Moose::Exception::CannotFixMetaclassCompatibility

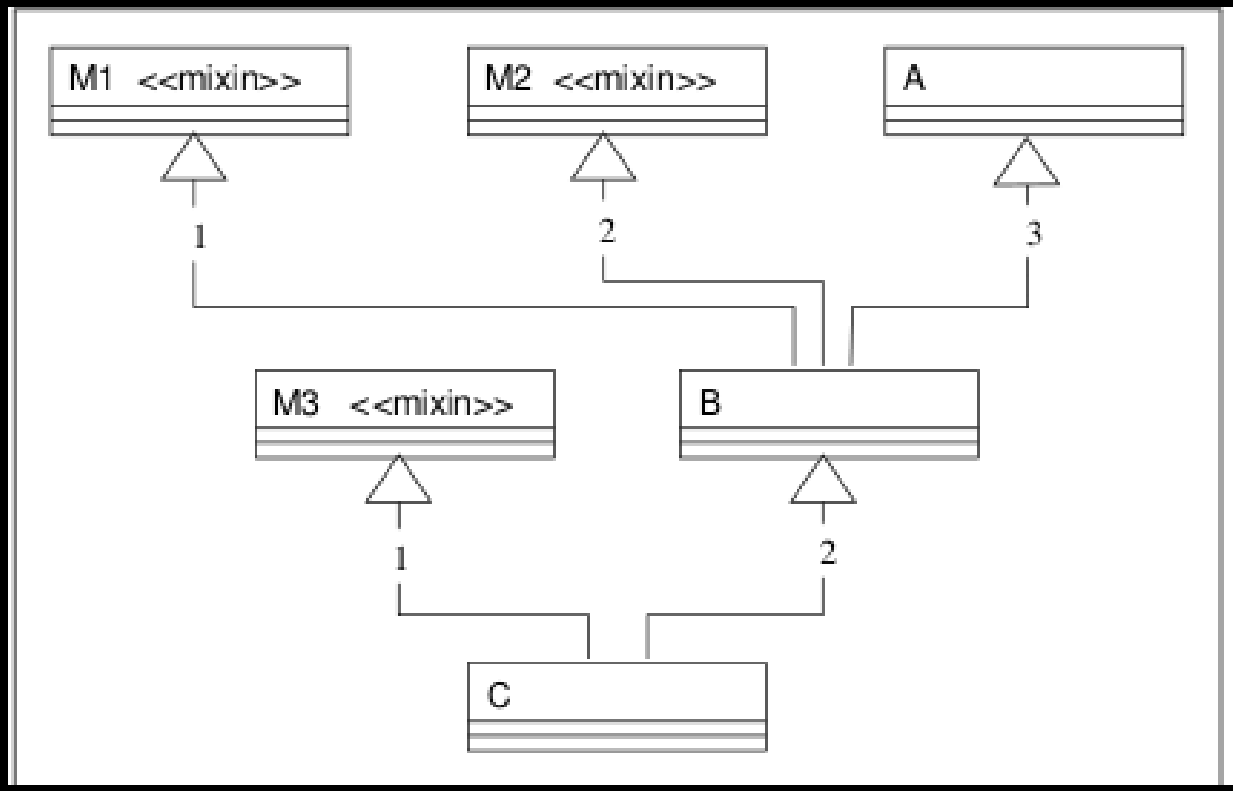
Mixins

- A class that contains a combination of methods from other classes
- 'Included' rather than 'inherited'
- Moose roles are similar to mixins

Rules of mixins-based inheritance

- Order of the mixins matter
- Mixins take precedence over non-mixins

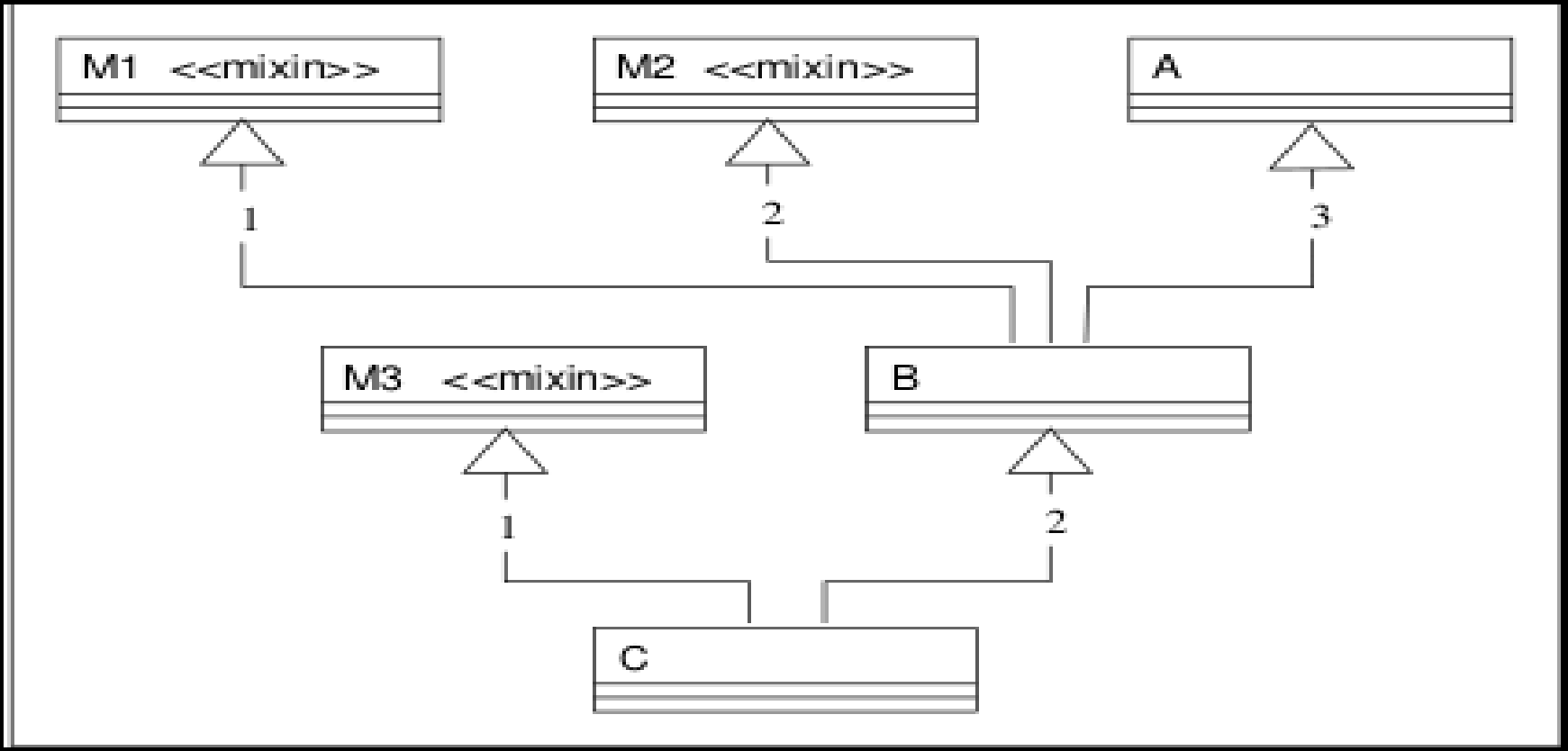
Mixins-based inheritance




```
A subclass: #B
    instanceVariableNames: 'u v'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Mixins–Example'
    metaclass: CompositeClass.
B mixins: {M1. M2}
```

```
B subclass: #C
    instanceVariableNames: 'w'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Mixins–Example'
    metaclass: CompositeClass.
C mixins: {M3}
```

Mixins-based inheritance

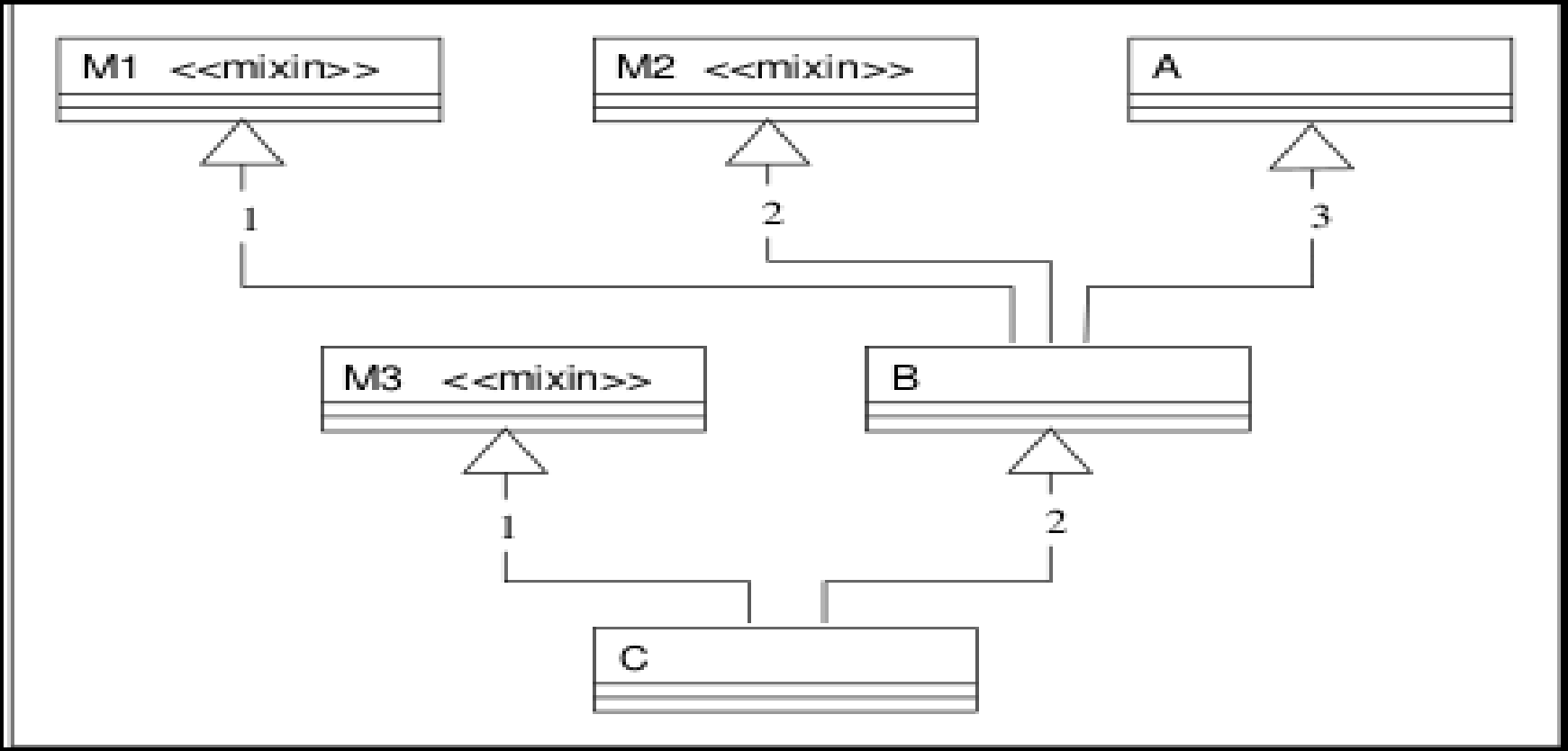


- B => {M1.M2.A}

Rules of mixins-based inheritance

- Methods in M2 will take precedence over A
- Methods in M1 will take precedence over M2

Mixins-based inheritance



- C => { M3.B.M1.M2.A }

Rules of mixins-based inheritance

- Methods in B will take precedence over M1
- Methods in M3 will take precedence over B

Moose provides a great MOP

Creating a class

```
Moose::Meta::Class->create(
    'Rectangle',
    attributes => {
        'height' => {
            is  => 'ro',
            isa => 'Int',
        },
        ...
    },
);
```

Introspection

- For getting attributes:
  `Rectangle->meta->get_attributes_list();`
- For getting methods:
  `Rectangle->meta->get_methods_list();`
- For getting superclasses:
  `Rectangle->meta->superclasses;`

Changing Class definition

- For adding a new attribute:

```
Rectangle->meta->add_attribute(...);
```

- For adding a new method:

```
Rectangle->meta->add_method(...);
```

Drawbacks of MOP

- Makes things slow
- While using Moose, don't forget to do:

```
__PACKAGE__->meta->make_immutable;
```

  - It tells Moose that you are not going to change your class at runtime

Bibliography

- The Art of the Metaobject Protocol
- Metaclass Composition Using Mixin-Based Inheritance by Noury Bouraqadi
- Wikipedia
- Moose documentation
- And lots of other random resources on the internet
- Stevan Little's awesome brain :)

Thank you for your time

Questions?