

Moo in practice - App::Math::Tutor

Jens Rehsack

Niederrhein Perl Mongers

2014-2015

Part I

Introduction

- 1 Introduction
 - Motivation
 - Audience

Motivation

Moo and App::Math::Tutor

- real world examples over far-fetched conceptualls
- MooX::Cmd, MooX::Options and MooX::ConfigFromFile provide way more features and flexibility than either
 - ▶ App::Cmd with Getopt::Long::Descriptive
 - ▶ MooseX::App::Cmd along with corresponding MooseX wrappers around related stuff
- 2nd generation of modern OO in Perl5

Motivation

Moo and App::Math::Tutor

- real world examples over far-fetched conceptualls
- MooX::Cmd, MooX::Options and MooX::ConfigFromFile provide way more features and flexibility than either
 - ▶ App::Cmd with Getopt::Long::Descriptive
 - ▶ MooseX::App::Cmd along with corresponding MooseX wrappers around related stuff
- 2nd generation of modern OO in Perl5

App::Math::Tutor

- Allow parents help their children improving their mathematical skills
- Add support for exercise types as children require
- provide extensible design to allow easy augment of exercise

Motivation

Moo and App::Math::Tutor

- real world examples over far-fetched conceptualls
- MooX::Cmd, MooX::Options and MooX::ConfigFromFile provide way more features and flexibility than either
 - ▶ App::Cmd with Getopt::Long::Descriptive
 - ▶ MooseX::App::Cmd along with corresponding MooseX wrappers around related stuff
- 2nd generation of modern OO in Perl5

App::Math::Tutor

- Allow parents help their children improving their mathematical skills
- Add support for exercise types as children require
- provide extensible design to allow easy augment of exercise
- **Goal:** Improve to Web-Service, eg. by mapping MooX::Cmd to URI path and MooX::Options to GET parameters

Audience

Audience

- Developers who want to create or improve Perl5 software
- Developers who want to learn how to develop modern OO with Perl5
- Developers who are interested in developing mathematical exercises

Audience

Audience

- Developers who want to create or improve Perl5 software
- Developers who want to learn how to develop modern OO with Perl5
- Developers who are interested in developing mathematical exercises

Prerequisites of the Audience

Following knowledge is expected:

- General knowledge about object oriented programming or concepts like
 - ▶ classes
 - ▶ objects
 - ▶ polymorphism, inheritance and/or roles
 - ▶ methods, class functions
 - ▶ attributes, properties
- slightly above basic Perl experience

Audience

Audience

- Developers who want to create or improve Perl5 software
- Developers who want to learn how to develop modern OO with Perl5
- Developers who are interested in developing mathematical exercises

Prerequisites of the Audience

Following knowledge is expected:

- General knowledge about object oriented programming or concepts like
 - ▶ classes
 - ▶ objects
 - ▶ polymorphism, inheritance and/or roles
 - ▶ methods, class functions
 - ▶ attributes, properties
- slightly above basic Perl experience
- ever heard of Smalltalk and its OO-concept is a strong bonus

Part II

Moo basics

- 2 Modules
 - Classes and Roles
- 3 Attributes
 - Attributes in Moo
 - Attribute Options
 - Attribute Options
- 4 Methods
 - Method Examples
 - Method Modifiers

Classes in Moo

- classes can be instantiated

Classes in Moo

- classes can be instantiated
- one can inherit from classes

Classes in Moo

- classes can be instantiated
- one can inherit from classes
- one can aggregate classes

Classes in Moo

- classes can be instantiated
- one can inherit from classes
- one can aggregate classes
- distinguish naming conflicts is up to developer

Classes in Moo

- classes can be instantiated
- one can inherit from classes
- one can aggregate classes
- distinguish naming conflicts is up to developer

```

{
  package Natural;

  use Moo;

  sub _stringify { ... };
}
{
  package Roman;

  use Moo;
  extends "Natural";

  sub _stringify { ... };
}
my $natnum = Natural->new( value => 42 ); say $natnum->_stringify(); # 42
my $romnum = Roman->new( value => 42 ); say $romnum->_stringify(); # XLII

```

Roles in Moo

- roles describe a dedicated behavior (e.g. `printable`)

Roles in Moo

- roles describe a dedicated behavior (e.g. `printable`)
- roles can be composed into classes

Roles in Moo

- roles describe a dedicated behavior (e.g. `printable`)
- roles can be composed into classes
- one can't inherit from roles - only consume

Roles in Moo

- roles describe a dedicated behavior (e.g. `printable`)
- roles can be composed into classes
- one can't inherit from roles - only consume
- roles cannot exist stand-alone

Roles in Moo

- roles describe a dedicated behavior (e.g. `printable`)
- roles can be composed into classes
- one can't inherit from roles - only consume
- roles cannot exist stand-alone
- roles are consumed once

Roles in Moo

- roles describe a dedicated behavior (e.g. `printable`)
- roles can be composed into classes
- one can't inherit from roles - only consume
- roles cannot exist stand-alone
- roles are consumed once
- naming conflicts cause compile time error

Roles in Moo

- roles describe a dedicated behavior (e.g. printable)
- roles can be composed into classes
- one can't inherit from roles - only consume
- roles cannot exist stand-alone
- roles are consumed once
- naming conflicts cause compile time error

```
{ package Printable;

  use Moo::Role; # now it's a role - no 'is a' relationship anymore

  sub print { my $self = shift; say $self->_stringify }
}

{ package Natural;

  use Moo; # class
  with "Printable"; # consumes a role

  sub _stringify { ... };
}

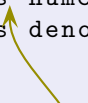
my $natnum = Natural->new( value => 42 ); $natnum->print; # 42
my $romnum = Roman->new( value => 42 ); $romnum->print; # XLII
```

Attributes in Moo

```
package VulFrac;  
  
use Moo;  
  
has numerator => ( is => "ro", required => 1 );  
has denominator => ( is => "ro", required => 1 );  
  
1;
```

Attributes in Moo

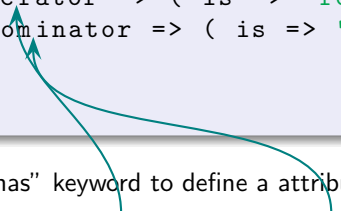
```
package VulFrac;  
  
use Moo;  
  
has numerator => ( is => "ro", required => 1 );  
has denominator => ( is => "ro", required => 1 );  
  
1;
```



- use "has" keyword to define a attribute

Attributes in Moo

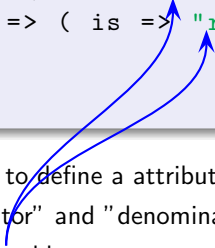
```
package VulFrac;  
  
use Moo;  
  
has numerator => ( is => "ro", required => 1 );  
has denominator => ( is => "ro", required => 1 );  
  
1;
```



- use "has" keyword to define a attribute
- attributes "numerator" and "denominator"

Attributes in Moo

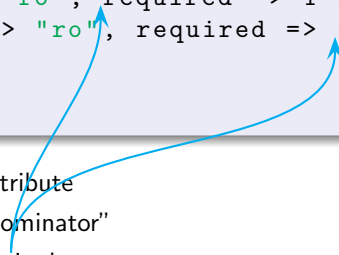
```
package VulFrac;  
  
use Moo;  
  
has numerator => ( is => "ro", required => 1 );  
has denominator => ( is => "ro", required => 1 );  
  
1;
```



- use "has" keyword to define a attribute
- attributes "numerator" and "denominator"
- attributes are immutable

Attributes in Moo

```
package VulFrac;  
  
use Moo;  
  
has numerator => ( is => "ro", required => 1 );  
has denominator => ( is => "ro", required => 1 );  
  
1;
```



- use "has" keyword to define a attribute
- attributes "numerator" and "denominator"
- attributes are immutable and required

Attribute options - Selection I

is **mandatory** behavior description

Attribute options - Selection I

is **mandatory** behavior description

ro defines the attribute is read-only

Attribute options - Selection I

is **mandatory** behavior description

ro defines the attribute is read-only

rw defined the attribute is read/writable

Attribute options - Selection I

is **mandatory** behavior description

ro defines the attribute is read-only

rw defined the attribute is read/writable

lazy defines the attribute is read-only with a lazy initialization, implies builder => 1

Attribute options - Selection I

is **mandatory** behavior description

ro defines the attribute is read-only

rw defined the attribute is read/writable

lazy defines the attribute is read-only with a lazy initialization, implies `builder => 1`

required when set to a true value, attribute must be passed on instantiation

Attribute options - Selection I

is **mandatory** behavior description

ro defines the attribute is read-only

rw defined the attribute is read/writable

lazy defines the attribute is read-only with a lazy initialization, implies `builder => 1`

required when set to a true value, attribute must be passed on instantiation

init_arg Takes the name of the key to look for at instantiation time of the object. A common use of this is to make an underscored attribute have a non-underscored initialization name. `undef` means that passing the value in on instantiation is ignored.

Attribute options - Selection I

is **mandatory** behavior description

ro defines the attribute is read-only

rw defined the attribute is read/writable

lazy defines the attribute is read-only with a lazy initialization, implies `builder => 1`

required when set to a true value, attribute must be passed on instantiation

init_arg Takes the name of the key to look for at instantiation time of the object. A common use of this is to make an underscored attribute have a non-underscored initialization name. `undef` means that passing the value in on instantiation is ignored.

isa defines a subroutine (`coderef`) which is called to validate values to set

Attribute options - Selection I

is **mandatory** behavior description

ro defines the attribute is read-only

rw defined the attribute is read/writable

lazy defines the attribute is read-only with a lazy initialization, implies `builder => 1`

required when set to a true value, attribute must be passed on instantiation

init_arg Takes the name of the key to look for at instantiation time of the object. A common use of this is to make an underscored attribute have a non-underscored initialization name. `undef` means that passing the value in on instantiation is ignored.

isa defines a subroutine (`coderef`) which is called to validate values to set

coerce defines a subroutine (`coderef`) which forces attribute values

Attribute options - Selection I

is **mandatory** behavior description

ro defines the attribute is read-only

rw defined the attribute is read/writable

lazy defines the attribute is read-only with a lazy initialization, implies `builder => 1`

required when set to a true value, attribute must be passed on instantiation

init_arg Takes the name of the key to look for at instantiation time of the object. A common use of this is to make an underscored attribute have a non-underscored initialization name. `undef` means that passing the value in on instantiation is ignored.

isa defines a subroutine (`coderef`) which is called to validate values to set

coerce defines a subroutine (`coderef`) which forces attribute values

default subroutine (`coderef`) which is called to initialize an attribute

Attribute options - Selection II

Following options can benefit from *attribute shortcuts*: the value of 1 instead of a method name means to generate a coderef which calls the method

`_${option_name}__${attr_name}`

`builder` takes a method name (string) which is called to initialize an attribute (calls `_build_${attr_name}` on *attribute shortcut*)

Attribute options - Selection II

Following options can benefit from *attribute shortcuts*: the value of 1 instead of a method name means to generate a coderef which calls the method

`_${option_name}_${attr_name}`

builder takes a method name (string) which is called to initialize an attribute (calls `_build_${attr_name}` on *attribute shortcut*)

trigger takes a subroutine (coderef) which is called anytime the attribute is set (calls `_trigger_${attr_name}` on *attribute shortcut*)

Attribute options - Selection II

Following options can benefit from *attribute shortcuts*: the value of 1 instead of a method name means to generate a coderef which calls the method

`_${option_name}_${attr_name}`

builder takes a method name (string) which is called to initialize an attribute (calls `_build_${attr_name}` on *attribute shortcut*)

trigger takes a subroutine (coderef) which is called anytime the attribute is set (calls `_trigger_${attr_name}` on *attribute shortcut*)

clearer takes a method name (string) which will clear the attribute (provides `clear_${attr_name}` on *attribute shortcut*)

Attribute options - Selection II

Following options can benefit from *attribute shortcuts*: the value of 1 instead of a method name means to generate a coderef which calls the method

`_${option_name}_${attr_name}`

builder takes a method name (string) which is called to initialize an attribute (calls `_build_${attr_name}` on *attribute shortcut*)

trigger takes a subroutine (coderef) which is called anytime the attribute is set (calls `_trigger_${attr_name}` on *attribute shortcut*)

clearer takes a method name (string) which will clear the attribute (provides `clear_${attr_name}` on *attribute shortcut*)

predicate takes a method name (string) which will return true if an attribute has a value (provides `has_${attr_name}` on *attribute shortcut*)

Methods in Moo I

```
package VulFrac;

use Moo;
use overload '""'    => "_stringify",
             '0+'    => "_numify",
             'bool'  => sub { 1 },
             '<=>'   => "_num_compare";

has numerator => ( is => "ro", required => 1 );
has denominator => ( is => "ro", required => 1,
                    isa => sub { $_[0] != 0 or die "Not != 0" } );

sub _stringify { my $self = shift;
                 return sprintf("\\frac{%s}{%s}",
                                $self->numerator, $self->denominator); }

sub _numify { $_[0]->numerator / $_[0]->denominator; }
...
```


Methods in Moo II

```
package Rationale;

use Moo;

extends "VulFrac";

has digits => ( is => "ro", required => 1 );

sub _stringify {
    my $digits = $_[0]->digits;
    sprintf("%.${digits}g", $_[0]->_numify); }
}
```

Methods in Moo II

```
package Rationale;

use Moo;

extends "VulFrac";

has digits => ( is => "ro", required => 1 );

sub _stringify {
    my $digits = $_[0]->digits;
    sprintf("%.${digits}g", $_[0]->_numify); }

```

- nothing like MooseX::Declare - pure Perl5 keywords are enough for plain methods

Method Modifiers

Method modifiers are a convenient feature from the CLOS (Common Lisp Object System) world:

Method Modifiers

Method modifiers are a convenient feature from the CLOS (Common Lisp Object System) world:

`before` `before method(s) => sub { ... }`

`before` is called before the method it is modifying. Its return value is totally ignored.

Method Modifiers

Method modifiers are a convenient feature from the CLOS (Common Lisp Object System) world:

before `before method(s) => sub { ... }`

`before` is called before the method it is modifying. Its return value is totally ignored.

after `after method(s) => sub { ... }`

`after` is called after the method it is modifying. Its return value is totally ignored.

Method Modifiers

Method modifiers are a convenient feature from the CLOS (Common Lisp Object System) world:

before `before method(s) => sub { ... }`

`before` is called before the method it is modifying. Its return value is totally ignored.

after `after method(s) => sub { ... }`

`after` is called after the method it is modifying. Its return value is totally ignored.

around `around method(s) => sub { ... }`

`around` is called instead of the method it is modifying. The method you're overriding is passed as `coderef` in the first argument.

Method Modifiers

Method modifiers are a convenient feature from the CLOS (Common Lisp Object System) world:

before `before method(s) => sub { ... }`

before is called before the method it is modifying. Its return value is totally ignored.

after `after method(s) => sub { ... }`

after is called after the method it is modifying. Its return value is totally ignored.

around `around method(s) => sub { ... }`

around is called instead of the method it is modifying. The method you're overriding is passed as `coderef` in the first argument.

- No support for `super`, `override`, `inner` or `augment`

Method Modifiers

Method modifiers are a convenient feature from the CLOS (Common Lisp Object System) world:

before `before method(s) => sub { ... }`

before is called before the method it is modifying. Its return value is totally ignored.

after `after method(s) => sub { ... }`

after is called after the method it is modifying. Its return value is totally ignored.

around `around method(s) => sub { ... }`

around is called instead of the method it is modifying. The method you're overriding is passed as `coderef` in the first argument.

- No support for `super`, `override`, `inner` or `augment`

In doubt `MooX::Override` provides `override` and `super`

Method Modifiers

Method modifiers are a convenient feature from the CLOS (Common Lisp Object System) world:

before `before method(s) => sub { ... }`

before is called before the method it is modifying. Its return value is totally ignored.

after `after method(s) => sub { ... }`

after is called after the method it is modifying. Its return value is totally ignored.

around `around method(s) => sub { ... }`

around is called instead of the method it is modifying. The method you're overriding is passed as `coderef` in the first argument.

- No support for `super`, `override`, `inner` or `augment`

In doubt `MooX::Override` provides `override` and `super` while `MooX::Augment` provides `augment` and `inner`

Method Modifiers - Advantages

- supersedes `$self->SUPER::foo(@_)` syntax

Method Modifiers - Advantages

- supersedes `$self->SUPER::foo(@_)` syntax
- cleaner interface than SUPER

Method Modifiers - Advantages

- supersedes `$self->SUPER::foo(@_)` syntax
- cleaner interface than SUPER
- allows multiple modifiers in single namespace

Method Modifiers - Advantages

- supersedes `$self->SUPER::foo(@_)` syntax
- cleaner interface than SUPER
- allows multiple modifiers in single namespace
- also possible from within roles and not restricted to inheritance

Method Modifiers - Advantages

- supersedes `$self->SUPER::foo(@_)` syntax
- cleaner interface than SUPER
- allows multiple modifiers in single namespace
- also possible from within roles and not restricted to inheritance
- ensures that inherited methods invocation happens right (mostly - remember around)

Method Modifiers - Advantages

- supersedes `$self->SUPER::foo(@_)` syntax
- cleaner interface than SUPER
- allows multiple modifiers in single namespace
- also possible from within roles and not restricted to inheritance
- ensures that inherited methods invocation happens right (mostly - remember around)
- no need to change packages

Methods Modifiers - around avoid calling \$orig

```
package App::Math::Tutor::Role::Roman;

use Moo::Role;

with "App::Math::Tutor::Role::Natural";

{ package RomanNum;
  use Moo;
  extends "NatNum"; # derives overloading!
  sub _stringify { ... } }

around "_guess_natural_number" => sub {
  my $orig = shift;
  my $max_val = $_[0]->format;
  my $value = int( rand( $max_val - 1 ) ) + 1;
  return RomanNum->new( value => $value );
};
```


Methods Modifiers - around avoid calling \$orig

```
package App::Math::Tutor::Role::Roman;

use Moo::Role;

with "App::Math::Tutor::Role::Natural";

{ package RomanNum;
  use Moo;
  extends "NatNum"; # derives overloading!
  sub _stringify { ... } }

around "_guess_natural_number" => sub {
  my $orig = shift;
  my $max_val = $_[0]->format;
  my $value = int( rand( $max_val - 1 ) ) + 1;
  return RomanNum->new( value => $value );
};
```

- captures control

Methods Modifiers - around avoid calling \$orig

```
package App::Math::Tutor::Role::Roman;

use Moo::Role;

with "App::Math::Tutor::Role::Natural";

{ package RomanNum;
  use Moo;
  extends "NatNum"; # derives overloading!
  sub _stringify { ... } }

around "_guess_natural_number" => sub {
  my $orig = shift;
  my $max_val = $_[0]->format;
  my $value = int( rand( $max_val - 1 ) ) + 1;
  return RomanNum->new( value => $value );
};
```

- captures control
- receives responsibility

Methods Modifiers - around avoid calling \$orig

```
package App::Math::Tutor::Role::Roman;

use Moo::Role;

with "App::Math::Tutor::Role::Natural";

{ package RomanNum;
  use Moo;
  extends "NatNum"; # derives overloading!
  sub _stringify { ... } }

around "_guess_natural_number" => sub {
  my $orig = shift;
  my $max_val = $_[0]->format;
  my $value = int( rand( $max_val - 1 ) ) + 1;
  return RomanNum->new( value => $value );
};
```

- captures control
- receives responsibility
- runtime of modified method completely eliminated

Methods Modifiers - around modifying \$orig return value

```
package App::Math::Tutor::Role::Roman;

use Moo::Role;

with "App::Math::Tutor::Role::Natural";

{ package RomanNum;
  use Moo;
  extends "NatNum"; # derives overloading!
  sub _stringify { ... } }

around "_guess_natural_number" => sub {
  my $orig = shift;
  my $num  = $orig->(@_);
  return RomanNum->new( value => $num->value );
};
```

Methods Modifiers - around modifying \$orig return value

```
package App::Math::Tutor::Role::Roman;

use Moo::Role;

with "App::Math::Tutor::Role::Natural";

{ package RomanNum;
  use Moo;
  extends "NatNum"; # derives overloading!
  sub _stringify { ... } }

around "_guess_natural_number" => sub {
  my $orig = shift;
  my $num  = $orig->(@_);
  return RomanNum->new( value => $num->value );
};
```

- modifies only required part

Methods Modifiers - around modifying \$orig return value

```
package App::Math::Tutor::Role::Roman;

use Moo::Role;

with "App::Math::Tutor::Role::Natural";

{ package RomanNum;
  use Moo;
  extends "NatNum"; # derives overloading!
  sub _stringify { ... } }

around "_guess_natural_number" => sub {
  my $orig = shift;
  my $num  = $orig->(@_);
  return RomanNum->new( value => $num->value );
};
```

- modifies only required part
- leaves most responsibility in modified method

Methods Modifiers - around modifying \$orig return value

```
package App::Math::Tutor::Role::Roman;

use Moo::Role;

with "App::Math::Tutor::Role::Natural";

{ package RomanNum;
  use Moo;
  extends "NatNum"; # derives overloading!
  sub _stringify { ... } }

around "_guess_natural_number" => sub {
  my $orig = shift;
  my $num  = $orig->(@_);
  return RomanNum->new( value => $num->value );
};
```

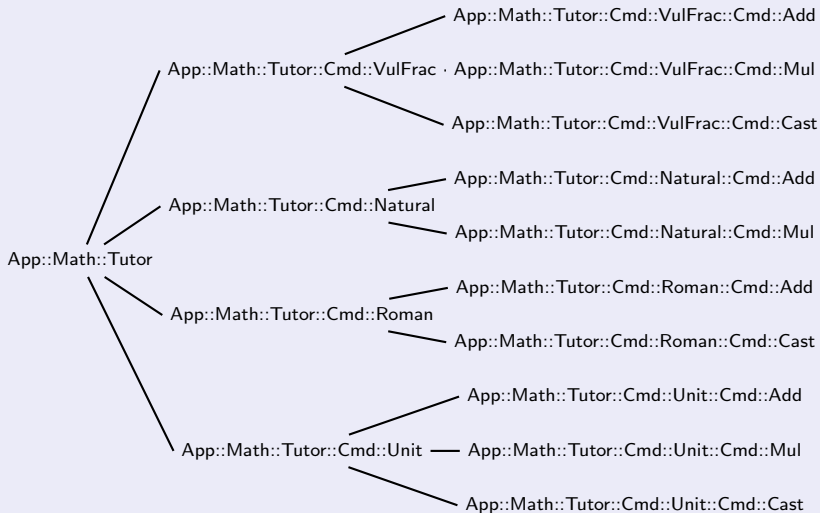
- modifies only required part
- leaves most responsibility in modified method
- runtime of modified method added to this method's runtime

Part III

Math Tutor

- 5 Structure
 - Frontend
 - CLI Concept
- 6 Design
 - Simple Exercise
 - Derived Exercise
- 7 Improvements
 - Design

Frontend (dictated by MooX::Cmd)



Exercise Groups

`App::Math::Tutor::Cmd::VulFrac` Exercises in vulgar fraction calculation

`App::Math::Tutor::Cmd::Natural` Exercises in calculations using natural numbers

`App::Math::Tutor::Cmd::Roman` Exercises in calculations using natural numbers, but show them using roman number encoding (exercises and solutions)

`App::Math::Tutor::Cmd::Unit` Exercises in calculations using units (times, currency, ...)

`App::Math::Tutor::Cmd::Power` Exercises in calculations of power mathematics

`App::Math::Tutor::Cmd::Polynom` Exercises for polynomial mathematics (Zero of a function, Vertex, ...)

Typical Exercise design

```
App::Math::Tutor::Cmd::VulFrac::Cmd::Add
```

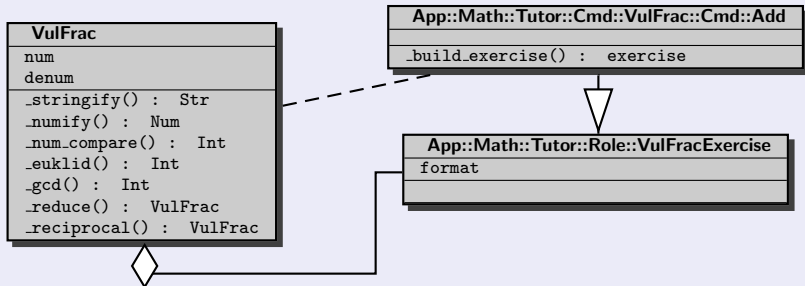
```
_build_exercise() : exercise
```

Typical Exercise design

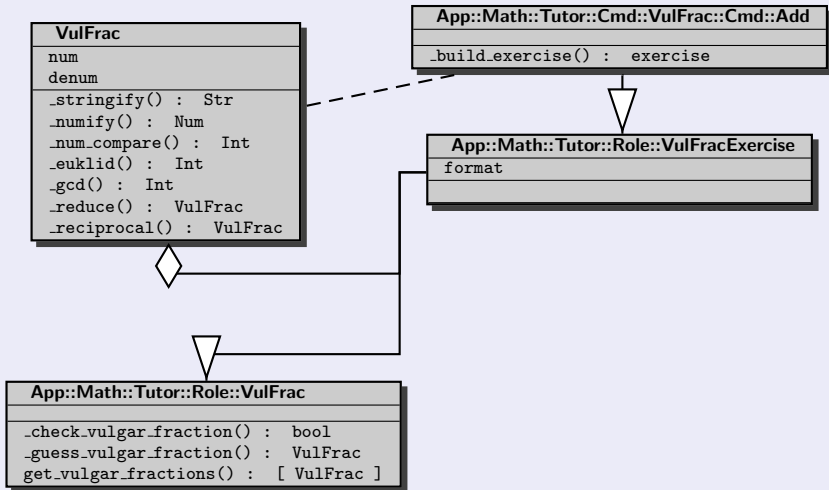
VulFrac
num
denum
<code>_stringify() : Str</code>
<code>_numify() : Num</code>
<code>_num_compare() : Int</code>
<code>_euklid() : Int</code>
<code>_gcd() : Int</code>
<code>_reduce() : VulFrac</code>
<code>_reciprocal() : VulFrac</code>

App::Math::Tutor::Cmd::VulFrac::Cmd::Add
<code>_build_exercise() : exercise</code>

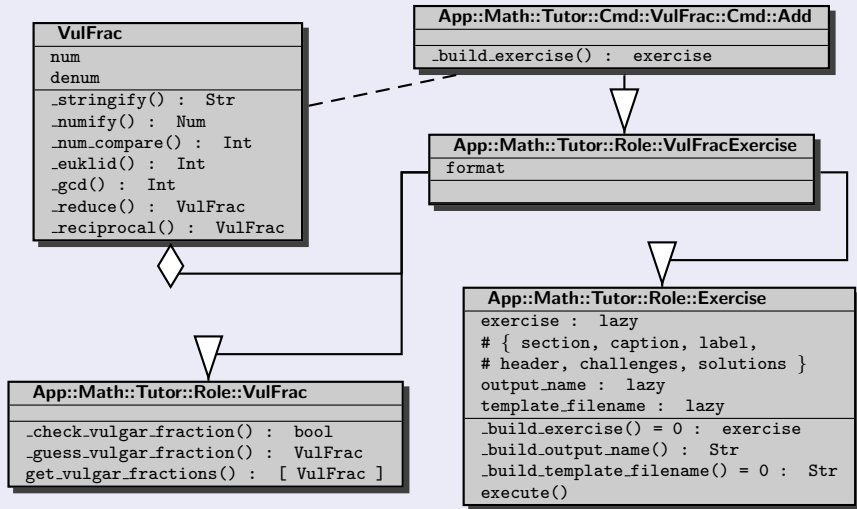
Typical Exercise design



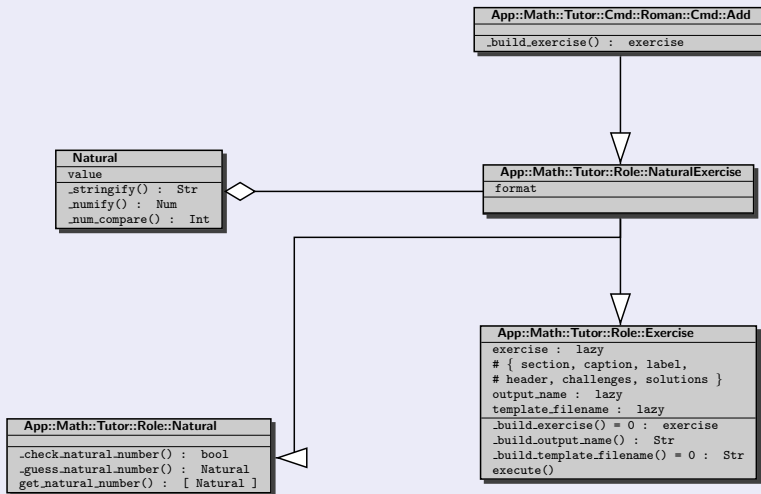
Typical Exercise design



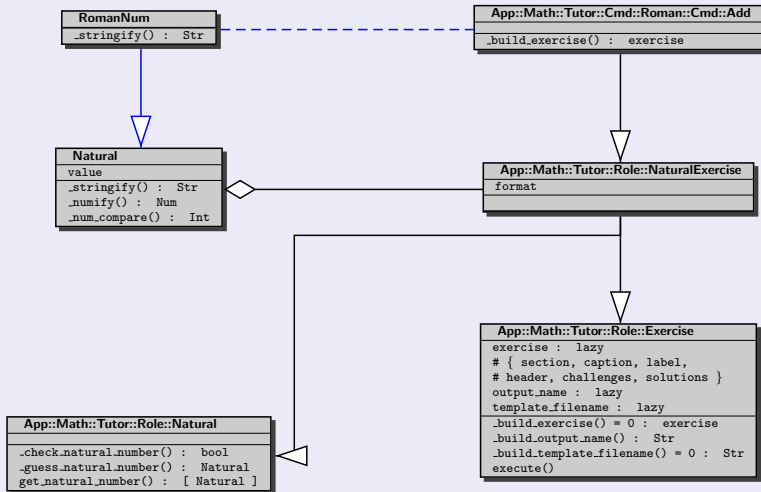
Typical Exercise design



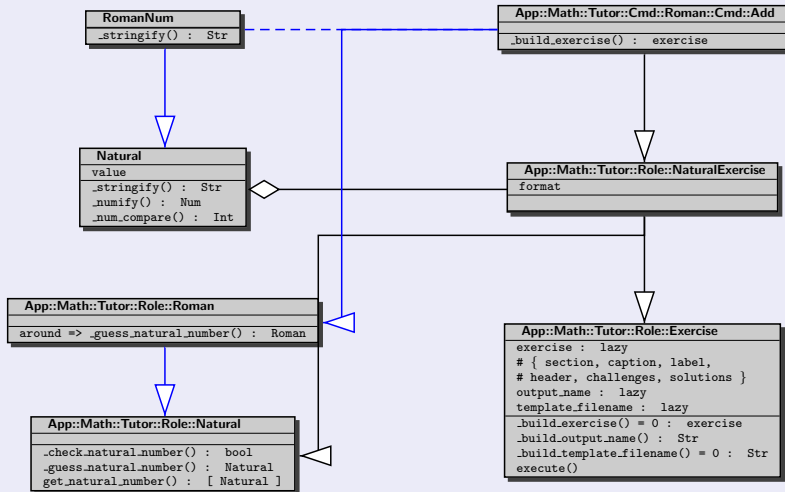
Derived Exercise design



Derived Exercise design



Derived Exercise design



Improving design

- introduce factories

Improving design

- introduce factories
- add lazy factory attribute

Improving design

- introduce factories
- add lazy factory attribute (allows overriding factory to use by around'ing builder)

Improving design

- introduce factories
- add lazy factory attribute (allows overriding factory to use by around'ing builder)
- separate number classes (type system, but **no** `MooseX::Types`)

Improving design

- introduce factories
- add lazy factory attribute (allows overriding factory to use by around'ing builder)
- separate number classes (type system, but **no** MooseX::Types)
- approval for reasonable exercise value should be part of factory

Improving design

- introduce factories
- add lazy factory attribute (allows overriding factory to use by around'ing builder)
- separate number classes (type system, but **no** MooseX::Types)
- approval for reasonable exercise value should be part of factory
- approval for valid number should be coerced (trigger?)

Improving design

- introduce factories
- add lazy factory attribute (allows overriding factory to use by around'ing builder)
- separate number classes (type system, but **no** MooseX::Types)
- approval for reasonable exercise value should be part of factory
- approval for valid number should be coerced (trigger?)

Design history

- I started with just vulgar fraction exercises

Improving design

- introduce factories
- add lazy factory attribute (allows overriding factory to use by around'ing builder)
- separate number classes (type system, but **no** `MooseX::Types`)
- approval for reasonable exercise value should be part of factory
- approval for valid number should be coerced (trigger?)

Design history

- I started with just vulgar fraction exercises
- Design covered exactly those needs

Improving design

- introduce factories
- add lazy factory attribute (allows overriding factory to use by around'ing builder)
- separate number classes (type system, but **no** MooseX::Types)
- approval for reasonable exercise value should be part of factory
- approval for valid number should be coerced (trigger?)

Design history

- I started with just vulgar fraction exercises
- Design covered exactly those needs

Design future

- Modern Perl-OO allows easy refactoring to apply above improvements

Part IV

Moo eXtensions

- 8 MooX
 - General purpose eXtensions
 - CLI related eXtensions

MooX in general

Some Moo extensions for getting a picture - the list is neither complete nor is it intended to be

MooX Distributions

`MooX::Log::Any` logging role building a very lightweight wrapper to `Log::Any`

MooX in general

Some Moo extensions for getting a picture - the list is neither complete nor is it intended to be

MooX Distributions

`MooX::Log::Any` logging role building a very lightweight wrapper to `Log::Any`

`MooX::Validate` minimalist Data Validation for Moo

MooX in general

Some Moo extensions for getting a picture - the list is neither complete nor is it intended to be

MooX Distributions

`MooX::Log::Any` logging role building a very lightweight wrapper to `Log::Any`

`MooX::Validate` minimalist Data Validation for Moo

`MooX::Types::MooseLike` Moose like types for Moo

MooX in general

Some Moo extensions for getting a picture - the list is neither complete nor is it intended to be

MooX Distributions

MooX::Log::Any logging role building a very lightweight wrapper to `Log::Any`

MooX::Validate minimalist Data Validation for Moo

MooX::Types::MooseLike Moose like types for Moo

MooX::Override adds "override method => sub ..." support to Moo

MooX in general

Some Moo extensions for getting a picture - the list is neither complete nor is it intended to be

MooX Distributions

MooX::Log::Any logging role building a very lightweight wrapper to Log::Any

MooX::Validate minimalist Data Validation for Moo

MooX::Types::MooseLike Moose like types for Moo

MooX::Override adds "override method => sub ..." support to Moo

MooX::Augment adds "augment method => sub ..." support to Moo

MooX in general

Some Moo extensions for getting a picture - the list is neither complete nor is it intended to be

MooX Distributions

MooX::Log::Any logging role building a very lightweight wrapper to `Log::Any`

MooX::Validate minimalist Data Validation for Moo

MooX::Types::MooseLike Moose like types for Moo

MooX::Override adds "override method => sub ..." support to Moo

MooX::Augment adds "augment method => sub ..." support to Moo

MooX::late easily translate Moose code to Moo

MooX in general

Some Moo extensions for getting a picture - the list is neither complete nor is it intended to be

MooX Distributions

MooX::Log::Any logging role building a very lightweight wrapper to `Log::Any`

MooX::Validate minimalist Data Validation for Moo

MooX::Types::MooseLike Moose like types for Moo

MooX::Override adds "override method => sub ..." support to Moo

MooX::Augment adds "augment method => sub ..." support to Moo

MooX::late easily translate Moose code to Moo

MooX::PrivateAttributes create attribute only usable inside your package

MooX in general

Some Moo extensions for getting a picture - the list is neither complete nor is it intended to be

MooX Distributions

MooX::Log::Any logging role building a very lightweight wrapper to Log::Any

MooX::Validate minimalist Data Validation for Moo

MooX::Types::MooseLike Moose like types for Moo

MooX::Override adds "override method => sub ..." support to Moo

MooX::Augment adds "augment method => sub ..." support to Moo

MooX::late easily translate Moose code to Moo

MooX::PrivateAttributes create attribute only usable inside your package

MooX::Singleton turn your Moo class into singleton

MooX in general

Some Moo extensions for getting a picture - the list is neither complete nor is it intended to be

MooX Distributions

MooX::Log::Any logging role building a very lightweight wrapper to `Log::Any`

MooX::Validate minimalist Data Validation for Moo

MooX::Types::MooseLike Moose like types for Moo

MooX::Override adds "override method => sub ..." support to Moo

MooX::Augment adds "augment method => sub ..." support to Moo

MooX::late easily translate Moose code to Moo

MooX::PrivateAttributes create attribute only usable inside your package

MooX::Singleton turn your Moo class into singleton

MooX::Aliases easy aliasing of methods and attributes in Moo

MooX Distributions for CLI

MooX::Cmd

- giving an easy Moo style way to make command organized CLI apps
- support sub-commands with separated options when used together with MooX::Options

MooX Distributions for CLI

MooX::Cmd

- giving an easy Moo style way to make command organized CLI apps
- support sub-commands with separated options when used together with MooX::Options

MooX::Options

- explicit Options eXtension for Object Class
- no mess with composed attributes

MooX Distributions for CLI

MooX::Cmd

- giving an easy Moo style way to make command organized CLI apps
- support sub-commands with separated options when used together with MooX::Options

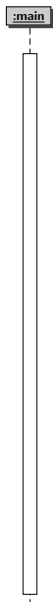
MooX::Options

- explicit Options eXtension for Object Class
- no mess with composed attributes

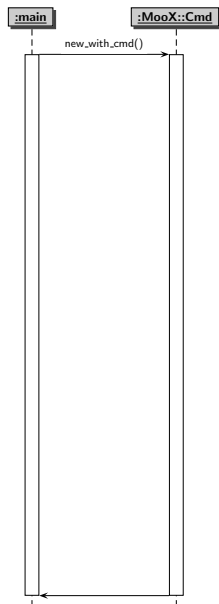
MooX::ConfigFromFile

- Moo eXtension for initializing objects from config file
- RC files with structures (JSON, INI, YAML, XML, ...)

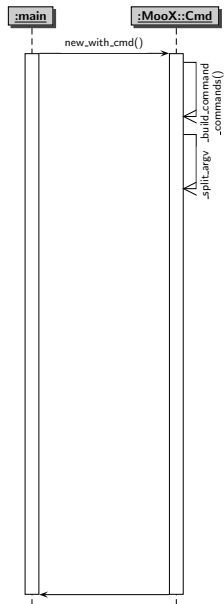
General CLI Construction Flow



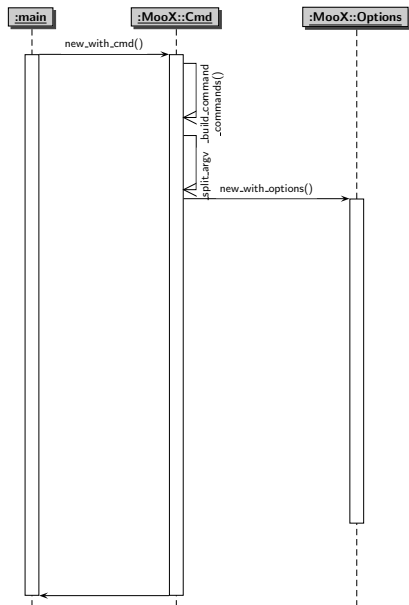
General CLI Construction Flow



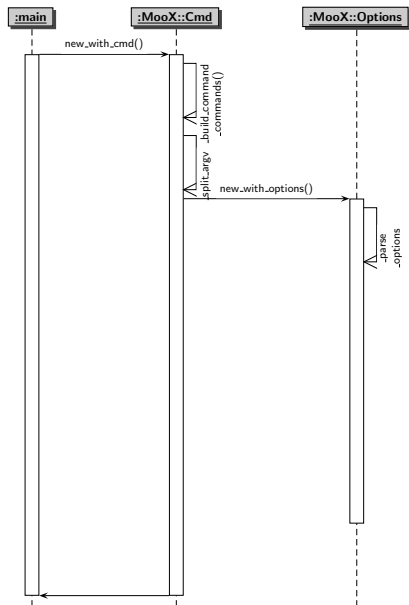
General CLI Construction Flow



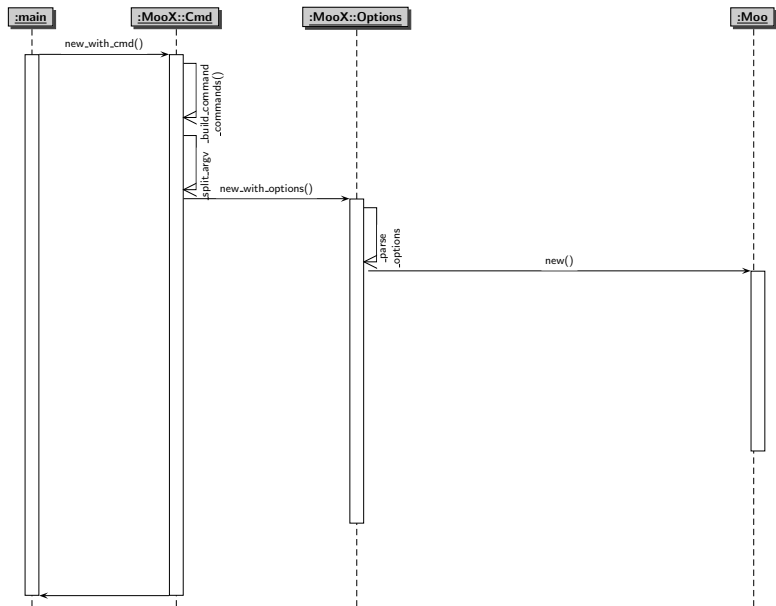
General CLI Construction Flow



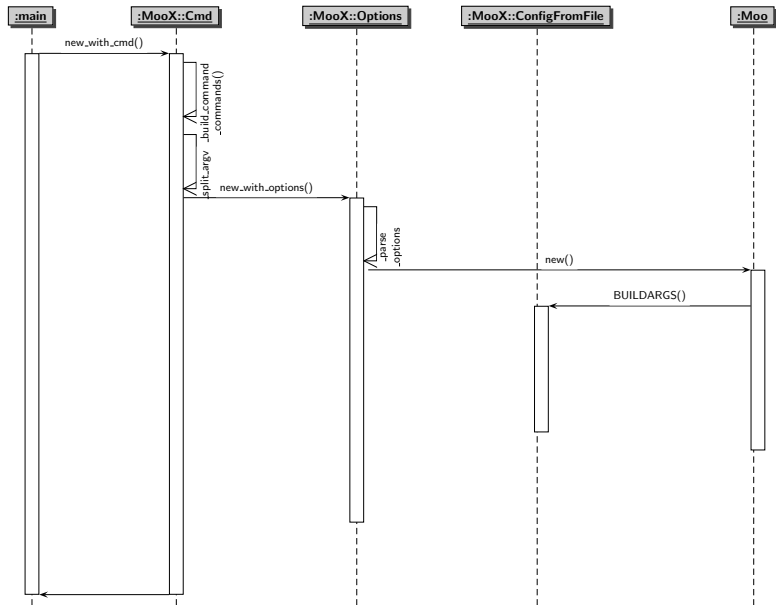
General CLI Construction Flow



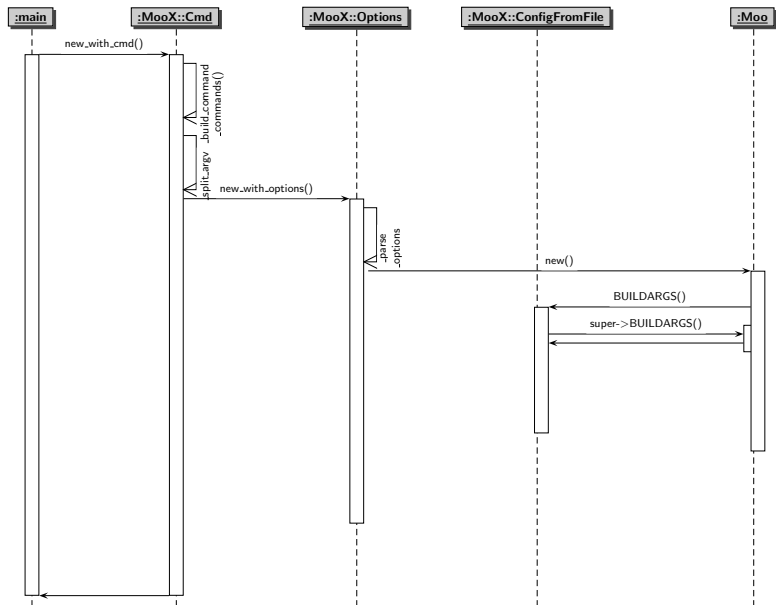
General CLI Construction Flow



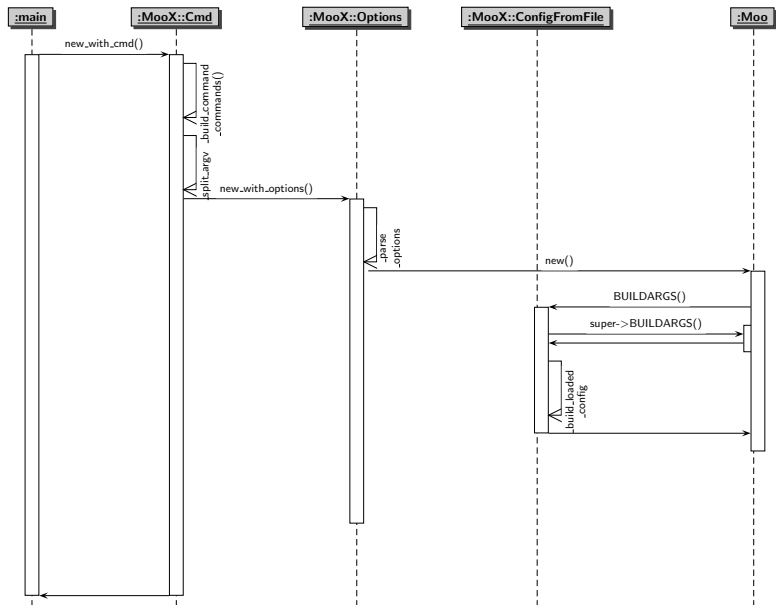
General CLI Construction Flow



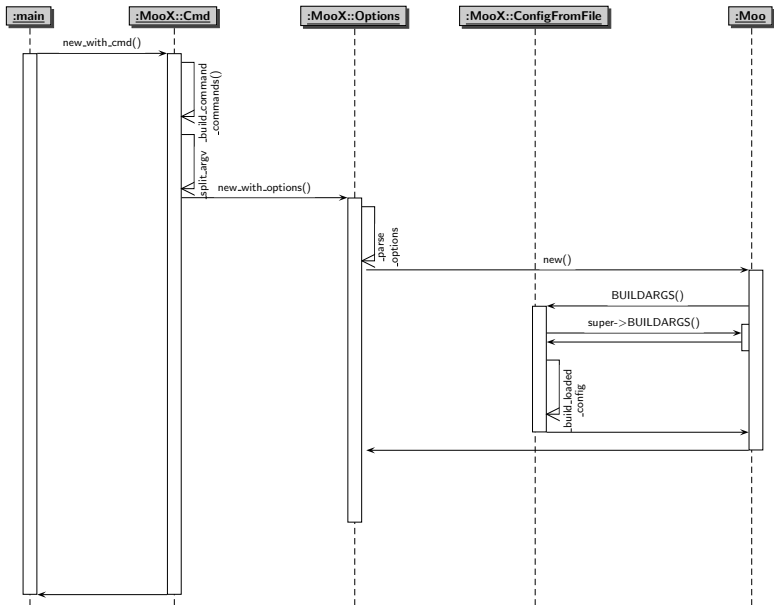
General CLI Construction Flow



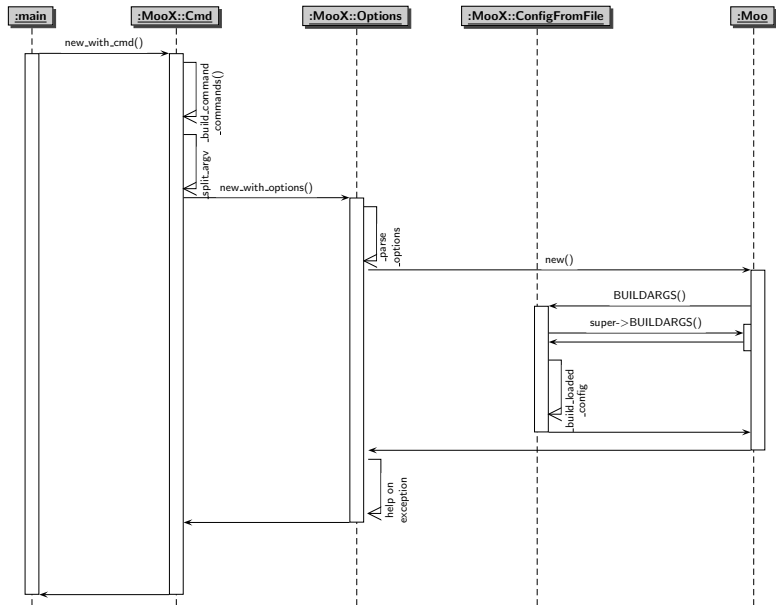
General CLI Construction Flow



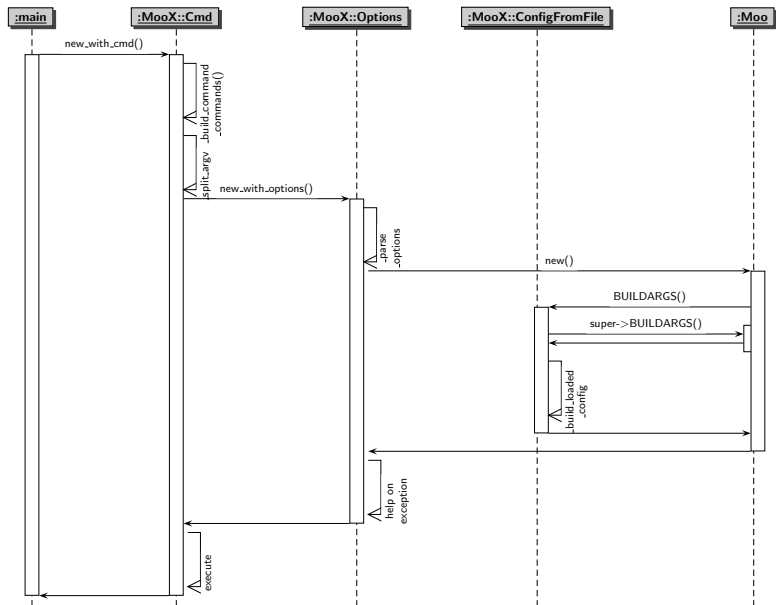
General CLI Construction Flow



General CLI Construction Flow



General CLI Construction Flow



Part V

Current Practice

- 9 Provided Framework
 - Exercise Role
 - Vulgar Fraction Exercise Role
- 10 Create Exercise
 - Preamble
 - Create Exercise Numbers
 - Format Challenge / Solution
 - Format Solutions
- 11 Created Exercise Output
 - Challenges
 - Solutions

```
package App::Math::Tutor::Role::Exercise;

use strictures; # instead of strict + warnings
use Moo::Role; use MooX::Options;

option quantity => (
    is          => "ro",
    doc         => "Specifies number of exercises to generate",
    long_doc   => "Specify number of exercises to generate. "
                . "In case of several kind of ...",
    format     => "i",
    short      => "n",
    default    => sub { 15 },
);

has output_name => ( is => "lazy" );
sub _build_output_name {
    join( "", map { $_->command_name || "" } @{$_[0]->command_chain} );
}
```

```
package App::Math::Tutor::Role::Exercise;

use strictures; # instead of strict + warnings
use Moo::Role; use MooX::Options;

option quantity => (
    is      => "ro",
    doc     => "Specifies number of exercises to generate",
    long_doc => "Specify number of exercises to generate. "
        . "In case of several kind of ...",
    format  => "i",
    short   => "n",
    default => sub { 15 },
);

has output_name => ( is => "lazy" );
sub _build_output_name {
    join( "", map { $_->command_name || "" } @{$_[0]->command_chain} );
}
```

- being a role (no "is a" relationship)

```
package App::Math::Tutor::Role::Exercise;

use strictures; # instead of strict + warnings
use Moo::Role; use MooX::Options;

option quantity => (
    is      => "ro",
    doc     => "Specifies number of exercises to generate",
    long_doc => "Specify number of exercises to generate. "
        . "In case of several kind of ...",
    format  => "i",
    short   => "n",
    default => sub { 15 },
);

has output_name => ( is => "lazy" );
sub _build_output_name {
    join( "", map { $_->command_name || "" } @{$_[0]->command_chain} );
}
```

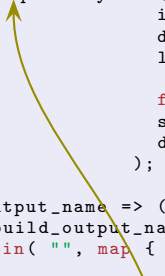
- being a role (no "is a" relationship) and add CLI options capabilities


```
package App::Math::Tutor::Role::Exercise;

use strictures; # instead of strict + warnings
use Moo::Role; use MooX::Options;

option quantity => (
    is      => "ro",
    doc     => "Specifies number of exercises to generate",
    long_doc => "Specify number of exercises to generate. "
        . "In case of several kind of ...",
    format  => "i",
    short   => "n",
    default => sub { 15 },
);

has output_name => ( is => "lazy" );
sub _build_output_name {
    join( "", map { $_->command_name || "" } @{$_[0]->command_chain} );
}
```



- being a role (no "is a" relationship) and add CLI options capabilities
- defining an attribute which will be initialized by command line

```
package App::Math::Tutor::Role::Exercise;

use strictures; # instead of strict + warnings
use Moo::Role; use MooX::Options;

option quantity => (
    is      => "ro",
    doc     => "Specifies number of exercises to generate",
    long_doc => "Specify number of exercises to generate. "
                . "In case of several kind of ...",
    format  => "i",
    short   => "n",
    default => sub { 15 },
);

has output_name => ( is => "lazy" );
sub _build_output_name {
    join( "", map { $_->command_name || "" } @{$_[0]->command_chain} );
}
```

- being a role (no "is a" relationship) and add CLI options capabilities
- defining an attribute which will be initialized by command line
- support MooX::Option's --man usage feature

```
package App::Math::Tutor::Role::Exercise;

use strictures; # instead of strict + warnings
use Moo::Role; use MooX::Options;

option quantity => (
    is      => "ro",
    doc     => "Specifies number of exercises to generate",
    long_doc => "Specify number of exercises to generate. "
        . "In case of several kind of ...",
    format  => "i",
    short   => "n",
    default => sub { 15 },
);

has output_name => ( is => "lazy" );
sub _build_output_name {
    join( "", map { $_->command_name || "" } @{ $_[0]->command_chain } );
}
```

- being a role (no "is a" relationship) and add CLI options capabilities
- defining an attribute which will be initialized by command line
- support MooX::Option's --man usage feature
- attribute for output file to generate

```
package App::Math::Tutor::Role::Exercise;

use strictures; # instead of strict + warnings
use Moo::Role; use MooX::Options;

option quantity => (
    is      => "ro",
    doc     => "Specifies number of exercises to generate",
    long_doc => "Specify number of exercises to generate. "
        . "In case of several kind of ...",
    format  => "i",
    short   => "n",
    default => sub { 15 },
);

has output_name => ( is => "lazy" );
sub _build_output_name {
    join( "", map { $_->command_name || "" } @{$_[0]->command_chain} );
}
```

- being a role (no "is a" relationship) and add CLI options capabilities
- defining an attribute which will be initialized by command line
- support MooX::Option's --man usage feature
- attribute for output file to generate — lazy to allow provide individual name builder

```
package App::Math::Tutor::Role::Exercise;

use strictures; # instead of strict + warnings
use Moo::Role; use MooX::Options;

option quantity => (
    is      => "ro",
    doc     => "Specifies number of exercises to generate",
    long_doc => "Specify number of exercises to generate. "
        . "In case of several kind of ...",
    format  => "i",
    short   => "n",
    default => sub { 15 },
);

has output_name => ( is => "lazy" );
sub _build_output_name {
    join( "", map { $_->command_name || "" } @{$_[0]->command_chain} );
}
```

- being a role (no "is a" relationship) and add CLI options capabilities
- defining an attribute which will be initialized by command line
- support MooX::Option's --man usage feature
- attribute for output file to generate — lazy to allow provide individual name builder

```
has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 });
  my $rc = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf');
  $rc or croak( $template->error() );
  return 0;
}
```

```
has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 });
  my $src = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf');
  $src or croak( $template->error() );
  return 0;
}
```

- attribute containing the exercises definitions, exercises itself and their solutions (depending on template)

```
has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 } );
  my $src = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf' );
  $src or croak( $template->error() );
  return 0;
}
```

- attribute containing the exercises definitions, exercises itself and their solutions (depending on template) — lazy implies requires `''_build_exercises''`


```
has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 });
  my $rc = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf');
  $rc or croak( $template->error() );
  return 0;
}
```

- attribute containing the exercises definitions, exercises itself and their solutions (depending on template) — lazy implies requires `''_build_exercises''`
- method to satisfy `MooX::Cmd`

```
has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 } );
  my $src = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf');
  $src or croak( $template->error() );
  return 0;
}
```

- attribute containing the exercises definitions, exercises itself and their solutions (depending on template) — lazy implies requires `'_build_exercises'`
- method to satisfy `MooX::Cmd`
- fetch exercises definition

```

has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 } );
  my $rc = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf');
  $rc or croak( $template->error() );
  return 0;
}

```

- attribute containing the exercises definitions, exercises itself and their solutions (depending on template) — lazy implies requires `'_build_exercises'`
- method to satisfy `MooX::Cmd`
- fetch exercises definition implies calling `_build_exercises`

```

has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 });
  my $src = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf');
  $src or croak( $template->error() );
  return 0;
}

```

- attribute containing the exercises definitions, exercises itself and their solutions (depending on template) — lazy implies requires `''_build_exercises''`
- method to satisfy `MooX::Cmd`
- fetch exercises definition implies calling `_build_exercises`
- build full qualified path of template file name residing in app's share directory,

```

has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 } );
  my $src = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf' );
  $src or croak( $template->error() );
  return 0;
}

```

- attribute containing the exercises definitions, exercises itself and their solutions (depending on template) — lazy implies requires `''_build_exercises''`
- method to satisfy `MooX::Cmd`
- fetch exercises definition implies calling `_build_exercises`
- build full qualified path of template file name residing in app's share directory,

```

has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 } );
  my $src = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf' );
  $src or croak( $template->error() );
  return 0;
}

```

- attribute containing the exercises definitions, exercises itself and their solutions (depending on template) — lazy implies requires `'_build_exercises'`
- method to satisfy `MooX::Cmd`
- fetch exercises definition implies calling `_build_exercises`
- build full qualified path of template file name residing in app's share directory,

```
has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 });
  my $src = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf');
  $src or croak( $template->error() );
  return 0;
}
```

- attribute containing the exercises definitions, exercises itself and their solutions (depending on template) — lazy implies requires `''_build_exercises''`
- method to satisfy `MooX::Cmd`
- fetch exercises definition implies calling `_build_exercises`
- build full qualified path of template file name residing in app's share directory, instantiate template processor

```

has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 } );
  my $src = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf' );
  $src or croak( $template->error() );
  return 0;
}

```

- attribute containing the exercises definitions, exercises itself and their solutions (depending on template) — lazy implies requires `''_build_exercises''`
- method to satisfy `MooX::Cmd`
- fetch exercises definition implies calling `_build_exercises`
- build full qualified path of template file name residing in app's share directory, instantiate template processor and run it


```

has exercises => ( is => "lazy" );
sub execute {
  my $exercises = $_[0]->exercises;
  my $ttcpath = File::Spec->catfile(
    File::ShareDir::dist_dir("App-Math-Tutor"),
    $_[0]->template_filename . ".tt2" );
  my $template = Template->new( { ABSOLUTE => 1 } );
  my $src = $template->process( $ttcpath, {
    exercises => $exercises,
    output     => { format => 'pdf' },
  },
    $_[0]->output_name . '.pdf' );
  $src or croak( $template->error() );
  return 0;
}

```

- attribute containing the exercises definitions, exercises itself and their solutions (depending on template) — lazy implies requires `''_build_exercises''`
- method to satisfy `MooX::Cmd`
- fetch exercises definition implies calling `_build_exercises`
- build full qualified path of template file name residing in app's share directory, instantiate template processor and run it
- force output format "pdf" (`Template::LaTeX` will be instructed by template)

```

package App::Math::Tutor::Role::VulFracExercise;
use strictures; use Moo::Role; use MooX::Options;
with "App::Math::Tutor::Role::Exercise", "App::Math::Tutor::Role::VulFrac";

option format => (
  is      => "ro",
  doc     => "specifies format of numerator/denominator",
  long_doc => "Allow specifying the format of the numerator/denominator ...",
  isa => sub { defined( $_[0] ) and !ref $_[0]
    and $_[0] !~ m,^\d?n+(?:/\d?n+)?$, and die("Invalid format");
  },
  coerce => sub { ...
    my ( $fmta, $fmtb ) = ( $_[0] =~ m,^\d?n+(?:/\d?n+)?$, );
    $fmtb //= $fmta;
    my $starta = "1"; my $startb = "1";
    $fmta =~ s/^\d(.*)/$2/ and $starta = $1;
    $fmtb =~ s/^\d(.*)/$2/ and $startb = $1;
    [ $starta . "0" x length($fmta), $startb . "0" x length($fmtb) ]
  },
  default => sub { return [ 100, 100 ]; }, format => "s",
    );

```

```

package App::Math::Tutor::Role::VulFracExercise;
use strictures; use Moo::Role; use MooX::Options;
with "App::Math::Tutor::Role::Exercise", "App::Math::Tutor::Role::VulFrac";

option format => (
  is      => "ro",
  doc     => "specifies format of numerator/denominator",
  long_doc => "Allow specifying the format of the numerator/denominator ...",
  isa => sub { defined( $_[0] ) and !ref $_[0]
    and $_[0] !~ m,^\d?n+(?:/\d?n+)?$, and die("Invalid format");
  },
  coerce => sub { ...
    my ( $fmta, $fmtb ) = ( $_[0] =~ m,^\d?n+(?:/\d?n+)?$, );
    $fmtb //= $fmta;
    my $starta = "1"; my $startb = "1";
    $fmta =~ s/^\d(.*)/$2/ and $starta = $1;
    $fmtb =~ s/^\d(.*)/$2/ and $startb = $1;
    [ $starta . "0" x length($fmta), $startb . "0" x length($fmtb) ]
  },
  default => sub { return [ 100, 100 ]; }, format => "s",
);

```

- be a nice package

```

package App::Math::Tutor::Role::VulFracExercise;
use strictures; use Moo::Role; use MooX::Options;
with "App::Math::Tutor::Role::Exercise", "App::Math::Tutor::Role::VulFrac";

option format => (
  is      => "ro",
  doc     => "specifies format of numerator/denominator",
  long_doc => "Allow specifying the format of the numerator/denominator ...",
  isa => sub { defined( $_[0] ) and !ref $_[0]
    and $_[0] !~ m,^\d?n+(?:/\d?n+)?$, and die("Invalid format");
  },
  coerce => sub { ...
    my ( $fmta, $fmtb ) = ( $_[0] =~ m,^\d?n+(?:/\d?n+)?$, );
    $fmtb //= $fmta;
    my $starta = "1"; my $startb = "1";
    $fmta =~ s/^\d(.*)/$2/ and $starta = $1;
    $fmtb =~ s/^\d(.*)/$2/ and $startb = $1;
    [ $starta . "0" x length($fmta), $startb . "0" x length($fmtb) ]
  },
  default => sub { return [ 100, 100 ]; }, format => "s",
);

```

- be a nice package ^Wrole,

```

package App::Math::Tutor::Role::VulFracExercise;
use strictures; use Moo::Role; use MooX::Options;
with "App::Math::Tutor::Role::Exercise", "App::Math::Tutor::Role::VulFrac";

option format => (
  is      => "ro",
  doc     => "specifies format of numerator/denominator",
  long_doc => "Allow specifying the format of the numerator/denominator ...",
  isa => sub { defined( $_[0] ) and !ref $_[0]
    and $_[0] !~ m,^\d?n+(?:/\d?n+)?$, and die("Invalid format");
  },
  coerce => sub { ...
    my ( $fmta, $fmtb ) = ( $_[0] =~ m,^\d?n+(?:/\d?n+)?$, );
    $fmtb //= $fmta;
    my $starta = "1"; my $startb = "1";
    $fmta =~ s/^\d(.*)/$2/ and $starta = $1;
    $fmtb =~ s/^\d(.*)/$2/ and $startb = $1;
    [ $starta . "0" x length($fmta), $startb . "0" x length($fmtb) ]
  },
  default => sub { return [ 100, 100 ]; }, format => "s",
);

```

- be a nice package ^Wrole, intending to provide command line options for instantiation

```

package App::Math::Tutor::Role::VulFracExercise;
use strictures; use Moo::Role; use MooX::Options;
with "App::Math::Tutor::Role::Exercise", "App::Math::Tutor::Role::VulFrac";

option format => (
  is      => "ro",
  doc     => "specifies format of numerator/denominator",
  long_doc => "Allow specifying the format of the numerator/denominator ...",
  isa => sub { defined( $_[0] ) and !ref $_[0]
    and $_[0] !~ m,^\d?n+(?:/\d?n+)?$, and die("Invalid format");
  },
  coerce => sub { ...
    my ( $fmta, $fmtb ) = ( $_[0] =~ m,^\d?n+(?:/\d?n+)?$, );
    $fmtb //= $fmta;
    my $starta = "1"; my $startb = "1";
    $fmta =~ s/^\d(.*)/$2/ and $starta = $1;
    $fmtb =~ s/^\d(.*)/$2/ and $startb = $1;
    [ $starta . "0" x length($fmta), $startb . "0" x length($fmtb) ]
  },
  default => sub { return [ 100, 100 ]; }, format => "s",
);

```

- be a nice package \hat{W} role, intending to provide command line options for instantiation
- compose role behavior using Exercise and VulFrac

```

package App::Math::Tutor::Role::VulFracExercise;
use strictures; use Moo::Role; use MooX::Options;
with "App::Math::Tutor::Role::Exercise", "App::Math::Tutor::Role::VulFrac";

option format => (
  is      => "ro",
  doc     => "specifies format of numerator/denominator",
  long_doc => "Allow specifying the format of the numerator/denominator ...",
  isa => sub { defined( $_[0] ) and !ref $_[0]
             and $_[0] !~ m, ^\d?n+(?:/\d?n+)?$, and die("Invalid format");
  },
  coerce => sub { ...
    my ( $fmta, $fmtb ) = ( $_[0] =~ m, ^(\d?n+)(?:/(\d?n+))?$, );
    $fmtb //= $fmta;
    my $starta = "1"; my $startb = "1";
    $fmta =~ s/^(\\d)(.*)/$2/ and $starta = $1;
    $fmtb =~ s/^(\\d)(.*)/$2/ and $startb = $1;
    [ $starta . "0" x length($fmta), $startb . "0" x length($fmtb) ]
  },
  default => sub { return [ 100, 100 ]; }, format => "s",
);

```

- be a nice package ^Wrole, intending to provide command line options for instantiation
- compose role behavior using Exercise and VulFrac

```

package App::Math::Tutor::Role::VulFracExercise;
use strictures; use Moo::Role; use MooX::Options;
with "App::Math::Tutor::Role::Exercise", "App::Math::Tutor::Role::VulFrac";

option format => (
  is      => "ro",
  doc     => "specifies format of numerator/denominator",
  long_doc => "Allow specifying the format of the numerator/denominator ...",
  isa => sub { defined( $_[0] ) and !ref $_[0]
             and $_[0] !~ m,^\d?n+(?:/\d?n+)?$, and die("Invalid format");
  },
  coerce => sub { ...
    my ( $fmta, $fmtb ) = ( $_[0] =~ m,^\d?n+(?:/\d?n+)?$, );
    $fmtb //= $fmta;
    my $starta = "1"; my $startb = "1";
    $fmta =~ s/^\d(.*)/$2/ and $starta = $1;
    $fmtb =~ s/^\d(.*)/$2/ and $startb = $1;
    [ $starta . "0" x length($fmta), $startb . "0" x length($fmtb) ]
  },
  default => sub { return [ 100, 100 ]; }, format => "s",
);

```

- be a nice package ^Wrole, intending to provide command line options for instantiation
- compose role behavior using Exercise and VulFrac
- define option


```

package App::Math::Tutor::Role::VulFracExercise;
use strictures; use Moo::Role; use MooX::Options;
with "App::Math::Tutor::Role::Exercise", "App::Math::Tutor::Role::VulFrac";

option format => (
  is      => "ro",
  doc     => "specifies format of numerator/denominator",
  long_doc => "Allow specifying the format of the numerator/denominator ...",
  isa => sub { defined( $_[0] ) and !ref $_[0]
             and $_[0] !~ m,^\d?n+(?:/\d?n+)?$, and die("Invalid format");
  },
  coerce => sub { ...
    my ( $fmta, $fmtb ) = ( $_[0] =~ m,^\d?n+(?:/\d?n+)?$, );
    $fmtb //= $fmta;
    my $starta = "1"; my $startb = "1";
    $fmta =~ s/^\d(.*)/$2/ and $starta = $1;
    $fmtb =~ s/^\d(.*)/$2/ and $startb = $1;
    [ $starta . "0" x length($fmta), $startb . "0" x length($fmtb) ]
  },
  default => sub { return [ 100, 100 ]; }, format => "s",
);

```

- be a nice package ^Wrole, intending to provide command line options for instantiation
- compose role behavior using Exercise and VulFrac
- define option coercing option string into array

```

package App::Math::Tutor::Role::VulFracExercise;
use strictures; use Moo::Role; use MooX::Options;
with "App::Math::Tutor::Role::Exercise", "App::Math::Tutor::Role::VulFrac";

option format => (
  is      => "ro",
  doc     => "specifies format of numerator/denominator",
  long_doc => "Allow specifying the format of the numerator/denominator ...",
  isa => sub { defined( $_[0] ) and !ref $_[0]
    and $_[0] !~ m,^\d?n+(?:/\d?n+)?$, and die("Invalid format");
  },
  coerce => sub { ...
    my ( $fmta, $fmtb ) = ( $_[0] =~ m,^\d?n+(?:/\d?n+)?$, );
    $fmtb //= $fmta;
    my $starta = "1"; my $startb = "1";
    $fmta =~ s/^\d(.*)/$2/ and $starta = $1;
    $fmtb =~ s/^\d(.*)/$2/ and $startb = $1;
    [ $starta . "0" x length($fmta), $startb . "0" x length($fmtb) ]
  },
  default => sub { return [ 100, 100 ]; }, format => "s",
);

```

- be a nice package ^Wrole, intending to provide command line options for instantiation
- compose role behavior using Exercise and VulFrac
- define option coercing option string into array ensuring the array ref as value

```
package App::Math::Tutor::Cmd::VulFrac::Cmd::Add;

use strictures;

use Moo;
use MooX::Cmd;
use MooX::Options;

has template_filename => ( is          => "ro",
                          default => "twocols");

with "App::Math::Tutor::Role::VulFracExercise";
```

```
package App::Math::Tutor::Cmd::VulFrac::Cmd::Add;

use strictures;

use Moo;
use MooX::Cmd;
use MooX::Options;

has template_filename => ( is      => "ro",
                          default => "twocols");

with "App::Math::Tutor::Role::VulFracExercise";
```

• \$ mtut

```
package App::Math::Tutor::Cmd::VulFrac::Cmd::Add;

use strictures;

use Moo;
use MooX::Cmd;
use MooX::Options;

has template_filename => ( is      => "ro",
                          default => "twocols");

with "App::Math::Tutor::Role::VulFracExercise";
```

- \$ mtut vulfrac

```
package App::Math::Tutor::Cmd::VulFrac::Cmd::Add;

use strictures;

use Moo;
use MooX::Cmd;
use MooX::Options;

has template_filename => ( is      => "ro",
                          default => "twocols");

with "App::Math::Tutor::Role::VulFracExercise";
```

• \$ mtut vulfrac add

```
package App::Math::Tutor::Cmd::VulFrac::Cmd::Add;

use strictures;

use Moo;
use MooX::Cmd;
use MooX::Options;

has template_filename => ( is          => "ro",
                          default => "twocols");

with "App::Math::Tutor::Role::VulFracExercise";
```

- \$ mtut vulfrac add
- we're of course a nice class

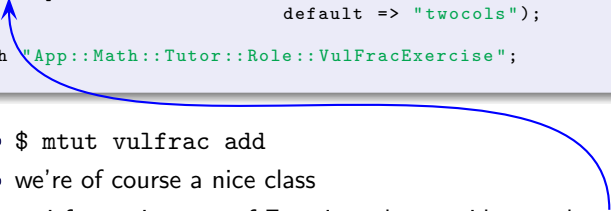
```
package App::Math::Tutor::Cmd::VulFrac::Cmd::Add;

use strictures;

use Moo;
use MooX::Cmd;
use MooX::Options;

has template_filename => ( is          => "ro",
                          default => "twocols");

with "App::Math::Tutor::Role::VulFracExercise";
```



- \$ mtut vulfrac add
- we're of course a nice class
- satisfy requirement of Exercise role: provide template_filename


```
package App::Math::Tutor::Cmd::VulFrac::Cmd::Add;

use strictures;

use Moo;
use MooX::Cmd;
use MooX::Options;

has template_filename => ( is      => "ro",
                          default => "twocols");

with "App::Math::Tutor::Role::VulFracExercise";
```

- `$ mtut vulfrac add`
- we're of course a nice class
- satisfy requirement of Exercise role: provide `template_filename` – two column template (for addition and subtraction)

```
package App::Math::Tutor::Cmd::VulFrac::Cmd::Add;

use strictures;

use Moo;
use MooX::Cmd;
use MooX::Options;

has template_filename => ( is      => "ro",
                          default => "twocols");

with "App::Math::Tutor::Role::VulFracExercise";
```

- \$ mtut vulfrac add
- we're of course a nice class
- satisfy requirement of Exercise role: provide template_filename – two column template (for addition and subtraction)
- compose role ...VulFracExercise

```
sub _build_exercises {
    my ($self) = @_;

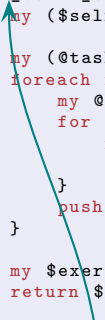
    my (@tasks);
    foreach my $i ( 1 .. $self->amount ) {
        my @line;
        for ( 0 .. 1 ) { # 0: +, 1: -
            my ( $a, $b ) = $self->get_vulgar_fractions(2);
            push @line, [ $a, $b ];
        }
        push @tasks, \@line;
    }

    my $exercises = ...;
    return $exercises;
}
```

```
sub _build_exercises {
  my ($self) = @_;

  my (@tasks);
  foreach my $i ( 1 .. $self->amount ) {
    my @line;
    for ( 0 .. 1 ) { # 0: +, 1: -
      my ( $a, $b ) = $self->get_vulgar_fractions(2);
      push @line, [ $a, $b ];
    }
    push @tasks, \@line;
  }

  my $exercises = ...;
  return $exercises;
}
```



- exercise builder has to be provided by individual exercise

```
sub _build_exercises {
    my ($self) = @_;

    my (@tasks);
    foreach my $i ( 1 .. $self->amount ) {
        my @line;
        for ( 0 .. 1 ) { # 0: +, 1: -
            my ( $a, $b ) = $self->get_vulgar_fractions(2);
            push @line, [ $a, $b ];
        }
        push @tasks, \@line;
    }

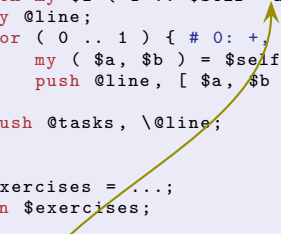
    my $exercises = ...;
    return $exercises;
}
```

- exercise builder has to be provided by individual exercise
- hold tasks of the exercise sheet

```
sub _build_exercises {
  my ($self) = @_;

  my (@tasks);
  foreach my $i ( 1 .. $self->amount ) {
    my @line;
    for ( 0 .. 1 ) { # 0: +, 1: -
      my ( $a, $b ) = $self->get_vulgar_fractions(2);
      push @line, [ $a, $b ];
    }
    push @tasks, \@line;
  }

  my $exercises = ...;
  return $exercises;
}
```



- exercise builder has to be provided by individual exercise
- hold tasks of the exercise sheet
- how many tasks per sheet?

```
sub _build_exercises {
  my ($self) = @_;

  my (@tasks);
  foreach my $i ( 1 .. $self->amount ) {
    my @line;
    for ( 0 .. 1 ) { # 0: +, 1: -
      my ( $a, $b ) = $self->get_vulgar_fractions(2);
      push @line, [ $a, $b ];
    }
    push @tasks, \@line;
  }

  my $exercises = ...;
  return $exercises;
}
```

- exercise builder has to be provided by individual exercise
- hold tasks of the exercise sheet
- how many tasks per sheet? (remember the option in ...Role::Exercise)

```
sub _build_exercises {
    my ($self) = @_;

    my (@tasks);
    foreach my $i ( 1 .. $self->amount ) {
        my @line;
        for ( 0 .. 1 ) { # 0: +, 1: -
            my ( $a, $b ) = $self->get_vulgar_fractions(2);
            push @line, [ $a, $b ];
        }
        push @tasks, \@line;
    }

    my $exercises = ...;
    return $exercises;
}
```

- exercise builder has to be provided by individual exercise
- hold tasks of the exercise sheet
- how many tasks per sheet? (remember the option in ...Role::Exercise)
- a "+" and a "-" exercise per line


```
sub _build_exercises {
  my ($self) = @_;

  my (@tasks);
  foreach my $i ( 1 .. $self->amount ) {
    my @line;
    for ( 0 .. 1 ) { # 0: +, 1: -
      my ( $a, $b ) = $self->get_vulgar_fractions(2);
      push @line, [ $a, $b ];
    }
    push @tasks, \@line;
  }

  my $exercises = ...;
  return $exercises;
}
```

- exercise builder has to be provided by individual exercise
- hold tasks of the exercise sheet
- how many tasks per sheet? (remember the option in ...Role::Exercise)
- a "+" and a "-" exercise per line
- invoke factory per task

```
sub _build_exercises {
    my ($self) = @_;

    my (@tasks);
    foreach my $i ( 1 .. $self->amount ) {
        my @line;
        for ( 0 .. 1 ) { # 0: +, 1: -
            my ( $a, $b ) = $self->get_vulgar_fractions(2);
            push @line, [ $a, $b ];
        }
        push @tasks, \@line;
    }

    my $exercises = ...;
    return $exercises;
}
```

- exercise builder has to be provided by individual exercise
- hold tasks of the exercise sheet
- how many tasks per sheet? (remember the option in ...Role::Exercise)
- a "+" and a "-" exercise per line
- invoke factory per task
- save each line for processing

```
my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my ( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my ( $a, $b, @way ) = @{$ $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @{$ $exercises->{challenges} }, \@challenge );
    push( @{$ $exercises->{solutions} }, \@solution );
}
```

```
my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my ( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my ( $a, $b, @way ) = @{ $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @{ $exercises->{challenges} }, \@challenge );
    push( @{ $exercises->{solutions} }, \@solution );
}
```

- create exercise structure containing

```
my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my ( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my ( $a, $b, @way ) = @{ $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @{ $exercises->{challenges} }, \@challenge );
    push( @{ $exercises->{solutions} }, \@solution );
}
```

- create exercise structure containing challenges and solutions

```
my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my ( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my ( $a, $b, @way ) = @{$ $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @{$ $exercises->{challenges} }, \@challenge );
    push( @{$ $exercises->{solutions} }, \@solution );
}
```

- create exercise structure containing challenges and solutions

```
my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my ( $a, $b, @way ) = @{ $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @{ $exercises->{challenges} }, \@challenge );
    push( @{ $exercises->{solutions} }, \@solution );
}
```

- create exercise structure containing challenges and solutions
- loop over created tasks

```
my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my( $a, $b, @way ) = @{ $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @{ $exercises->{challenges} }, \@challenge );
    push( @{ $exercises->{solutions} }, \@solution );
}
```

- create exercise structure containing challenges and solutions
- loop over created tasks and exercises per line


```

my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my ( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my ( $a, $b, @way ) = @{$ $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @{$ $exercises->{challenges} }, \@challenge );
    push( @{$ $exercises->{solutions} }, \@solution );
}

```

- create exercise structure containing challenges and solutions
- loop over created tasks and exercises per line
- format challenge using operator `'"` of VulFrac objects

```

my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my ( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my ( $a, $b, @way ) = @ { $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @ { $exercises->{challenges} }, \@challenge );
    push( @ { $exercises->{solutions} }, \@solution );
}

```

- create exercise structure containing challenges and solutions
- loop over created tasks and exercises per line
- format challenge using operator `'"` of VulFrac objects

```

my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my ( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my ( $a, $b, @way ) = @ { $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @ { $exercises->{challenges} }, \@challenge );
    push( @ { $exercises->{solutions} }, \@solution );
}

```

- create exercise structure containing challenges and solutions
- loop over created tasks and exercises per line
- format challenge using operator `'"` of VulFrac objects

```

my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my ( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my ( $a, $b, @way ) = @{ $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @{ $exercises->{challenges} }, \@challenge );
    push( @{ $exercises->{solutions} }, \@solution );
}

```

- create exercise structure containing challenges and solutions
- loop over created tasks and exercises per line
- format challenge using operator '""' of VulFrac objects
- same (but different) opening "way"

```

my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my ( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my ( $a, $b, @way ) = @{ $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @{ $exercises->{challenges} }, \@challenge );
    push( @{ $exercises->{solutions} }, \@solution );
}

```

- create exercise structure containing challenges and solutions
- loop over created tasks and exercises per line
- format challenge using operator `'"` of VulFrac objects
- same (but different) opening `"way"`

```

my $exercises = {
    section => "Vulgar fraction addition / subtraction",
    caption => 'Fractions',
    label   => 'vulgar_fractions_addition',
    header  => [ [ 'Vulgar Fraction Addition', 'Vulgar Fraction S
    challenges => [], solutions => [],
};

foreach my $line (@tasks) {
    my ( @solution, @challenge );
    foreach my $i ( 0 .. 1 ) {
        my ( $a, $b, @way ) = @{ $line->[$i] }; my $op = $i ? '-' : '+';
        $op eq '-' and $a < $b and ( $b, $a ) = ( $a, $b );
        push @challenge, sprintf( '$ %s %s %s = $', $a, $op, $b );
        push @way, sprintf( '%s %s %s', $a, $op, $b );
        ...
    }
    push( @{ $exercises->{challenges} }, \@challenge );
    push( @{ $exercises->{solutions} }, \@solution );
}

```

- create exercise structure containing challenges and solutions
- loop over created tasks and exercises per line
- format challenge using operator `'"` of VulFrac objects
- same (but different) opening `"way"` and remember the little thingsies ...

```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, "" . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num > $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$ ' . join( " = ", @way ) . ' $' );
}

```

```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, "" . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num > $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$' . join( " = ", @way ) . '$' );
}

```

- try to reduce operands


```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, "" . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num > $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$' . join( " = ", @way ) . '$' );
}

```

- try to reduce operands and add them to opening

```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, "" . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num > $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$' . join( " = ", @way ) . '$' );
}

```

- try to reduce operands and add them to opening when successful

```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, "" . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num > $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$ ' . join( " = ", @way ) . ' $' );
}

```

- try to reduce operands and add them to opening when successful
- World of Handcraft: show calculation method by

```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, " " . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num > $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$ ' . join( " = ", @way ) . ' $' );
}

```

- try to reduce operands and add them to opening when successful
- World of Handcraft: show calculation method by determine geatest common divisor of operands denominator,

```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, " . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num > $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$ ' . join( " = ", @way ) . '$ ' );
}

```

- try to reduce operands and add them to opening when successful
- World of Handcraft: show calculation method by determine greatest common divisor of operands denominator, format subsequent

```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, "" . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num > $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$' . join( " = ", @way ) . '$' );
}

```

- try to reduce operands and add them to opening when successful
- World of Handcraft: show calculation method by determine greatest common divisor of operands denominator, format subsequent steps to reach

```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, "" . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num / $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$ ' . join( " = ", @way ) . '$' );
}

```

- try to reduce operands and add them to opening when successful
- World of Handcraft: show calculation method by determine greatest common divisor of operands denominator, format subsequent steps to reach the solution

```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, "" . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num > $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$ ' . join( " = ", @way ) . ' $' );
}

```

- try to reduce operands and add them to opening when successful
- World of Handcraft: show calculation method by determine greatest common divisor of operands denominator, format subsequent steps to reach the solution
- remark possible reducing,


```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, "" . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num > $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$ ' . join( " = ", @way ) . ' $' );
}

```

- try to reduce operands and add them to opening when successful
- World of Handcraft: show calculation method by determine greatest common divisor of operands denominator, format subsequent steps to reach the solution
- remark possible reducing, mixed fraction,

```

sub _build_exercises {
  ( $a, $b ) = ( $a->_reduce, $b = $b->_reduce );
  push @way, sprintf( '%s %s %s', $a, $op, $b )
    if ( $a->num != $line->[$i]->[0]->num or $b->num != ... );

  my $gcd = VulFrac->new( num => $a->denum, denum => $b->denum )->_gcd;
  my ( $fa, $fb ) = ( $b->{denum} / $gcd, $a->{denum} / $gcd );
  push @way, sprintf( '\frac{%d \cdot %d}{%d \cdot %d} %s ...',
    $a->num, $fa, $a->denum, $fa, $op, $b->num, $fb, ... );
  push @way, sprintf( '\frac{%d}{%d} %s \frac{%d}{%d}',
    $a->num*$fa, $a->denum*$fa, $op, $b->num*$fb, $b->denum*$fb );
  push @way, sprintf( '\frac{%d %s %d}{%d}',
    $a->num * $fa, $op, $b->num * $fb, $a->denum * $fa );
  my $s = VulFrac->new( denum => $a->denum * $fa,
    num => $i ? $a->num * $fa - $b->num * $fb : ...s/-/+);
  push @way, "" . $s; my $c = $s->_reduce; $c->num != $s->num and push @way,
  $c->num > $c->denum and $c->denum > 1 and push @way, $c->_stringify(1);

  push( @solution, '$ ' . join( " = ", @way ) . ' $' );
}

```

- try to reduce operands and add them to opening when successful
- World of Handcraft: show calculation method by determine geatest common divisor of operands denominator, format subsequent steps to reach the solution
- remark possible reducing, mixed fraction, cord it and go ahead

Vulgar Fraction Addition

$$\frac{99}{67} + \frac{43}{84} =$$

$$\frac{19}{7} + \frac{16}{54} =$$

$$\frac{38}{82} + \frac{38}{99} =$$

$$\frac{96}{80} + \frac{46}{39} =$$

$$\frac{72}{68} + \frac{46}{99} =$$

Vulgar Fraction Subtraction

$$\frac{45}{41} - \frac{32}{48} =$$

$$\frac{51}{47} - \frac{49}{65} =$$

$$\frac{56}{33} - \frac{30}{32} =$$

$$\frac{49}{79} - \frac{29}{82} =$$

$$\frac{42}{49} - \frac{38}{59} =$$

Vulgar Fraction Addition

$$\frac{99}{67} + \frac{43}{84} = \frac{99 \cdot 84}{67 \cdot 84} + \frac{43 \cdot 67}{84 \cdot 67} = \frac{8316}{5628} + \frac{2881}{5628} =$$

$$\frac{8316+2881}{5628} = \frac{11197}{5628} = 1 \frac{5569}{5628}$$

$$\frac{19}{7} + \frac{16}{54} = \frac{19}{7} + \frac{8}{27} = \frac{19 \cdot 27}{7 \cdot 27} + \frac{8 \cdot 7}{27 \cdot 7} = \frac{513}{189} +$$

$$\frac{56}{189} = \frac{513+56}{189} = \frac{569}{189} = 3 \frac{2}{189}$$

$$\frac{38}{82} + \frac{38}{99} = \frac{19}{41} + \frac{38}{99} = \frac{19 \cdot 99}{41 \cdot 99} + \frac{38 \cdot 41}{99 \cdot 41} = \frac{1881}{4059} +$$

$$\frac{1558}{4059} = \frac{1881+1558}{4059} = \frac{3439}{4059}$$

$$\frac{96}{80} + \frac{46}{39} = \frac{6}{5} + \frac{46}{39} = \frac{6 \cdot 39}{5 \cdot 39} + \frac{46 \cdot 5}{39 \cdot 5} = \frac{234}{195} + \frac{230}{195} =$$

$$\frac{234+230}{195} = \frac{464}{195} = 2 \frac{74}{195}$$

$$\frac{72}{68} + \frac{46}{99} = \frac{18}{17} + \frac{46}{99} = \frac{18 \cdot 99}{17 \cdot 99} + \frac{46 \cdot 17}{99 \cdot 17} = \frac{1782}{1683} +$$

$$\frac{782}{1683} = \frac{1782+782}{1683} = \frac{2564}{1683} = 1 \frac{881}{1683}$$

Vulgar Fraction Subtraction

$$\frac{45}{41} - \frac{32}{48} = \frac{45}{41} - \frac{2}{3} = \frac{45 \cdot 3}{41 \cdot 3} - \frac{2 \cdot 41}{3 \cdot 41} = \frac{135}{123} - \frac{82}{123} =$$

$$\frac{135-82}{123} = \frac{53}{123}$$

$$\frac{51}{47} - \frac{49}{65} = \frac{51}{47} - \frac{49}{65} = \frac{51 \cdot 65}{47 \cdot 65} - \frac{49 \cdot 47}{65 \cdot 47} = \frac{3315}{3055} -$$

$$\frac{2303}{3055} = \frac{3315-2303}{3055} = \frac{1012}{3055}$$

$$\frac{56}{33} - \frac{30}{32} = \frac{56}{33} - \frac{15}{16} = \frac{56 \cdot 16}{33 \cdot 16} - \frac{15 \cdot 33}{33 \cdot 16} = \frac{896}{528} -$$

$$\frac{495}{528} = \frac{896-495}{528} = \frac{401}{528}$$

$$\frac{49}{79} - \frac{29}{82} = \frac{49 \cdot 82}{79 \cdot 82} - \frac{29 \cdot 79}{82 \cdot 79} = \frac{4018}{6478} - \frac{2291}{6478} =$$

$$\frac{4018-2291}{6478} = \frac{1727}{6478}$$

$$\frac{42}{49} - \frac{38}{59} = \frac{6}{7} - \frac{38}{59} = \frac{6 \cdot 59}{7 \cdot 59} - \frac{38 \cdot 7}{59 \cdot 7} = \frac{354}{413} - \frac{266}{413} =$$

$$\frac{354-266}{413} = \frac{88}{413}$$

Part VI

Finish

12 Conclusion

13 Resources

- Resources
- Thank you

Conclusion

- lazy attributes allow designing a multi-stage initialization phase

Conclusion

- lazy attributes allow designing a multi-stage initialization phase
- benefit of common runtime (faster load) when using

Conclusion

- lazy attributes allow designing a multi-stage initialization phase
- benefit of common runtime (faster load) when using
 - ▶ DBIx::Class
 - ▶ Web::Simple (Jedi)

Conclusion

- lazy attributes allow designing a multi-stage initialization phase
- benefit of common runtime (faster load) when using
 - ▶ DBIx::Class
 - ▶ Web::Simple (Jedi)
- improve design by

Conclusion

- lazy attributes allow designing a multi-stage initialization phase
- benefit of common runtime (faster load) when using
 - ▶ DBIx::Class
 - ▶ Web::Simple (Jedi)
- improve design by
 - ▶ using roles for behavioral design (avoid duck typing)

Conclusion

- lazy attributes allow designing a multi-stage initialization phase
- benefit of common runtime (faster load) when using
 - ▶ DBIx::Class
 - ▶ Web::Simple (Jedi)
- improve design by
 - ▶ using roles for behavioral design (avoid duck typing)
 - ▶ using explicit patterns for clear separation of concerns
 - ▶ express intensions clearer for method overloading by using *method modifiers*

Resources

Software

<https://metacpan.org/pod/Moo/>

<https://github.com/moose/Moo>

<https://metacpan.org/search?q=MooX>

<https://metacpan.org/pod/MooX::Cmd>

<https://metacpan.org/pod/MooX::Options>

<https://metacpan.org/pod/MooX::ConfigFromFile>

<https://github.com/rehsack/App-Math-Tutor>

Resources

Software

<https://metacpan.org/pod/Moo/>

<https://github.com/moose/Moo>

<https://metacpan.org/search?q=MooX>

<https://metacpan.org/pod/MooX::Cmd>

<https://metacpan.org/pod/MooX::Options>

<https://metacpan.org/pod/MooX::ConfigFromFile>

<https://github.com/rehsack/App-Math-Tutor>

IRC

<irc://irc.perl.org/#moose>

<irc://irc.perl.org/#web-simple>

Thank You For Contributing

Thank You For Contributing

Graham "haarg" Knop Found a lot of spelling errors and first suggestions

Thank You For Contributing

Graham "haarg" Knop Found a lot of spelling errors and first suggestions

Vincent "celogeek" Bachelier Suggested some real examples

Thank You For Contributing

Graham "haarg" Knop Found a lot of spelling errors and first suggestions

Vincent "celogeek" Bachelier Suggested some real examples

Fedor Babkin UML diagram improvements

Thank You For Contributing

Graham "haarg" Knop Found a lot of spelling errors and first suggestions

Vincent "celogeek" Bachelier Suggested some real examples

Fedor Babkin UML diagram improvements

Curtis "Ovid" Poe Final review and figured out missing fundamentals

Thank You For Listening

Questions?

Jens Rehsack <rehsack@cpan.org>