Object Oriented Programming Sucks

Grant McLean
<grantm@cpan.org>

Everyone “knows” Perl OO Sucks

Actually
Perl sucks less!

Smart people came up with a solution

Make your language suck less too!

OO's unfulfilled promise of code reuse

Could Do Better

Code Reuse is Hard

Essential Elements of OO?

What is an object?

- Object = Data + Behaviour

Jargon++

- Encapsulation
- Polymorphism

What's missing?

- Inheritance

What is a class?

Factories

Units of reuse

Reuse

- Inherit and override
- Aggregate and delegate

Conflict!

- Factory => Big
- Code reuse => Small

The big idea:

- *Wouldn't it be cool if there was some sort of container for reusable code that was smaller than a class*

And now there is

Traits Defined

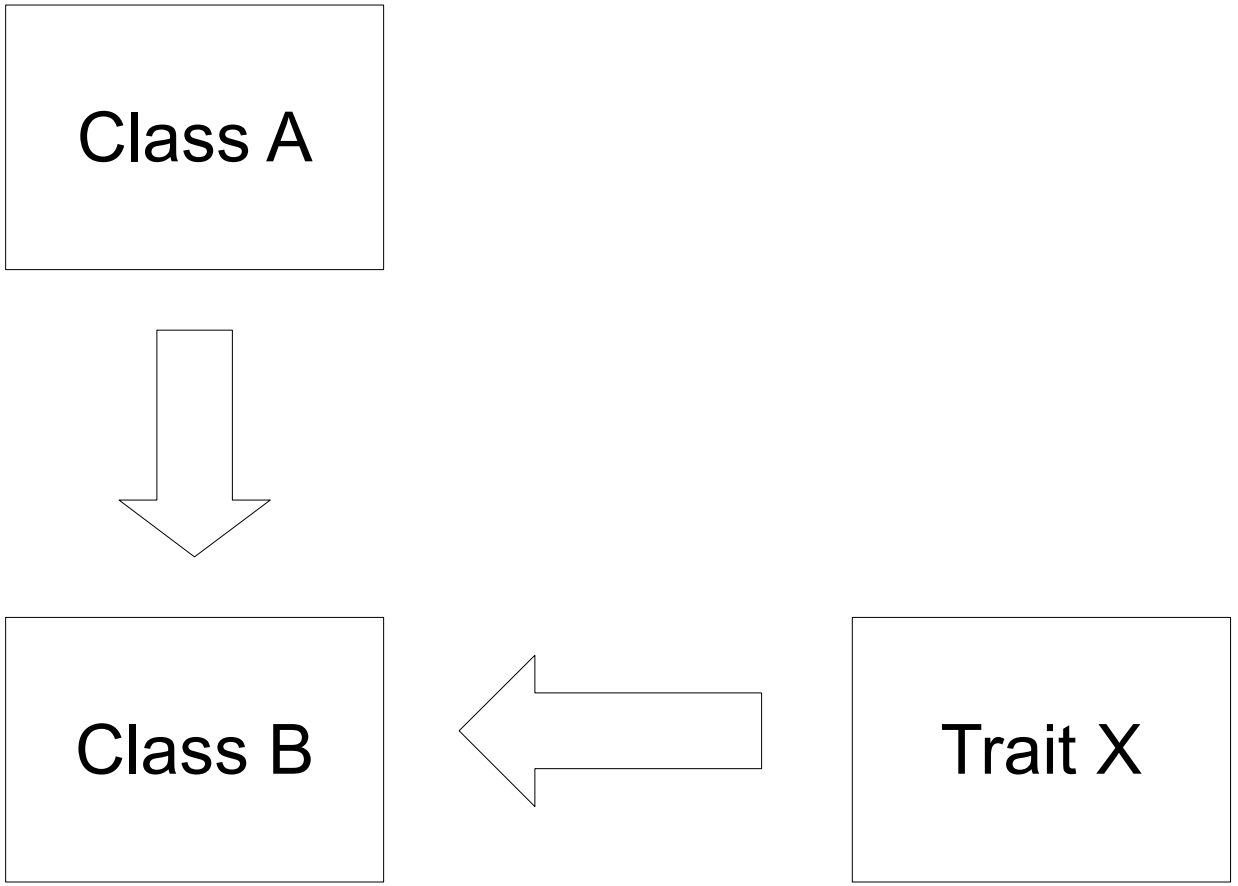
- *Traits: composable units of behaviour*
  - Schärli, Ducasse, Nierstrasz, and Black, 2002

Traits Applied

- *Applying Traits to the Smalltalk Collections Classes*
  - Black, Schärli, Ducasse, 2003

What are traits?

A Big Picture



1. Collection of methods

2. To be composed into classes

3. "Pure behaviour"

4. Flattened into the class

5. No superclass

6. Composition operators

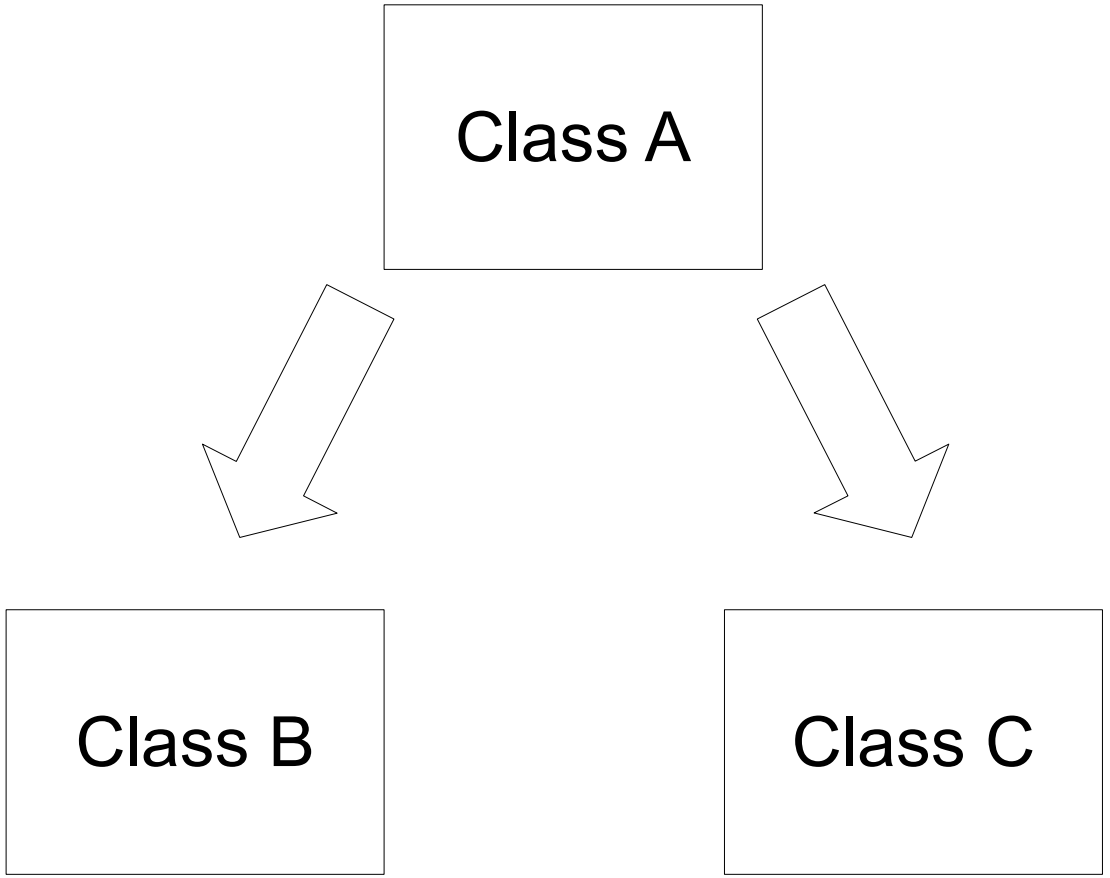
7. Dependencies

How do Traits stack up?

Reuse through Inheritance

- Single inheritance
- Multiple inheritance
- Mixin inheritance

Single Inheritance



Multiple Inheritance (MI)

- *"Multiple inheritance is good, but there's no good way to do it"*
  - Steve Cook
    (paraphrasing Alan Snyder)

Too evil for:

- Smalltalk, Java, C# ...
- And PHP!!!!

Of course it's allowed in Perl

Method collisions

- Not always obvious
- Not easy to fix

Mixin Inheritance

Mixin Problem #1

- Order of class composition matters

Mixin Problem #2

- Conflicts resolved silently

Mixin Problem #3

- What does SUPER mean?

Mixins just don't scale

Traits trump Mixins

- Conflicts – compile errors*

Trait Conflict Resolution:

- Class methods win
- Exclude trait methods
- Alias trait methods

Dependencies

- Mixins assume
- Traits require

Using traits in Perl

- Called 'Roles'
- Implemented in Moose
- Also in Perl 6

Creating a class with Moose:

```
package Car;

use Moose;

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

1;
```

Creating a role:

```
package Breakable;

use Moose::Role;

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

sub break {
    my $self = shift;

    print "I broke\n";

    $self->is_broken(1);
}
```

Composing a Role into a Class

```
package Car;

use Moose;

with 'Breakable';

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

1;
```

Move 'break' to class

```
package Car;

use Moose;

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

sub break {
    my $self = shift;

    $self->is_moving(0) if $self->is_moving;
}
```

Use a "method modifier" hook

```
package Breakable;

use Moose::Role;

requires 'break';

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

after 'break' => sub {
    my $self = shift;

    $self->is_broken(1);
};

1;
```

Lets get 'meta'

```
has 'is_moving' => (
    is      => 'rw',
    isa     => 'Bool',
    default => 0,
    traits  => ['Bool'],
    handles => {
        start      => 'set',
        stop       => 'unset',
        is_stopped => 'not',
    },
);

sub break {
    my $self = shift;
    $self->stop if $self->is_moving;
}
```

Example usage:

```
use Car;

my $car = Car->new(colour => 'red');

$car->start;
printf("Car %s moving\n", $car->is_moving ? "is" : "isn't"); # is
printf("Car %s broken\n", $car->is_broken ? "is" : "isn't"); # isn't

$car->break;
printf("Car %s moving\n", $car->is_moving ? "is" : "isn't"); # isn't
printf("Car %s broken\n", $car->is_broken ? "is" : "isn't"); # is

print "Car does 'Breakable'\n" if $car->does('Breakable');   # true
print "Car is a 'Breakable'\n" if $car->isa('Breakable');    # false
```

Method Aliasing

```
with 'Breakable'   => {
        -alias => { break => 'break_bone'  }
    },
    'Breakdancer' => {
        -alias => { break => 'break_dance' }
    };
```

Method Exclusion:

```
with 'Breakable' => {
        -alias    => { break => 'break_bone' },
        -excludes => 'break',
    },
    'Breakdancer' => {
        -alias    => { break => 'break_dance' },
        -excludes => 'break',
    };

sub break {
    # perhaps call $self->break_bone
    # and $self->break_dance
}
```

Python

- Experimental implementation: 'Strait'
- http://pypi.python.org/pypi/strait

Acknowledgements:

- http://scg.unibe.ch/research/traits/
- The authors of the original traits papers from which I have 'borrowed' liberally
- Perl Bloggers: Curtis 'Ovid' Poe, and chromatic