

OO Systems and Roles

Curtis “Ovid” Poe

Senior Software Engineer

Copyright 2009-2010 by Curtis “Ovid” Poe.

This presentation is free and you may redistribute it and/or modify it under the terms of the GNU Free Documentation License.

Template Copyright 2009 by BBC.

<http://www.slideshare.net/Ovid/inheritance-versus-roles>

Not a Tutorial

- #ovidfail
- “How” is easy
- “Why” and “when” is not

A Brief History of Pain

- Simula 67
 - Classes
 - Polymorphism
 - Encapsulation
 - Inheritance

Multiple Inheritance

- C++
- Eiffel
- Perl
- CLOS

Single Inheritance

- C#
- Java
- BETA
- Ruby

Handling Inheritance

- Liskov
- Strict equivalence
- Interfaces
- Mixins
- C3

The Extremes

- BETA!

- <http://www.daimi.au.dk/~beta/>

- **Pattern**

```
employee:
(# name: @text;
  dept: ^Department;
  totalHours: @integer;
  registerWork:
    (# noOfHours: @integer
      enter noOfHours
      do noOfHours + totalhours -> totalHours
    #);
computeSalary:<
  (# salary: @integer
    do inner
    exit salary
  #);
#);
```

- **Subpatterns**

```
worker: employee
```

```
(# seniority: @integer;
```

```
  computeSalary: <
```

```
    (# do totalHours * 80 + seniority * 4  
      -> salary;
```

```
    0 -> totalHours #)
```

```
#);
```

```
salesman: employee
```

```
(# numSoldUnits: @integer;
```

```
  computeSalary: <
```

```
    (# do totalHours * 80 + numSoldUnits * 6  
      -> salary;
```

```
    0->numSoldUnits->totalHours
```

```
  #)
```

```
#)
```

BETA

- Inheritance Order

```
computeSalary:<          (* from pattern *)
  (# salary: @integer
    do inner
      exit salary
  #);
```

```
computeSalary::<        (* to subpattern *)
  (# do totalHours * 80 + seniority * 4
      -> salary;
    0 -> totalHours
  #)
```

BETA

- Inheritance Order

```
computeSalary:<          (* from employee *)
  (# salary: @integer
    do
      (* optionally do something here *)
      inner
      (# do (if salary < 0 then
              0 -> salary
              (* an exception is better *)
            if)#)
      exit salary
    #);
```

BETA

- Inverts inheritance order
- Almost automatic Liskov
- Can't forget parents
- (Moose's augment/inner)

Four Decades of Pain

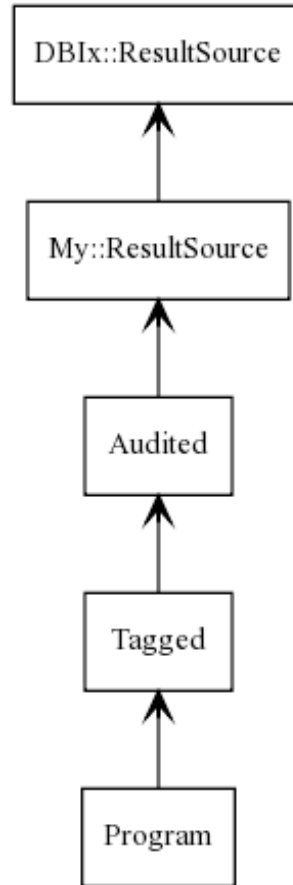
- Code smell
 - In the language!

The Problem Domain

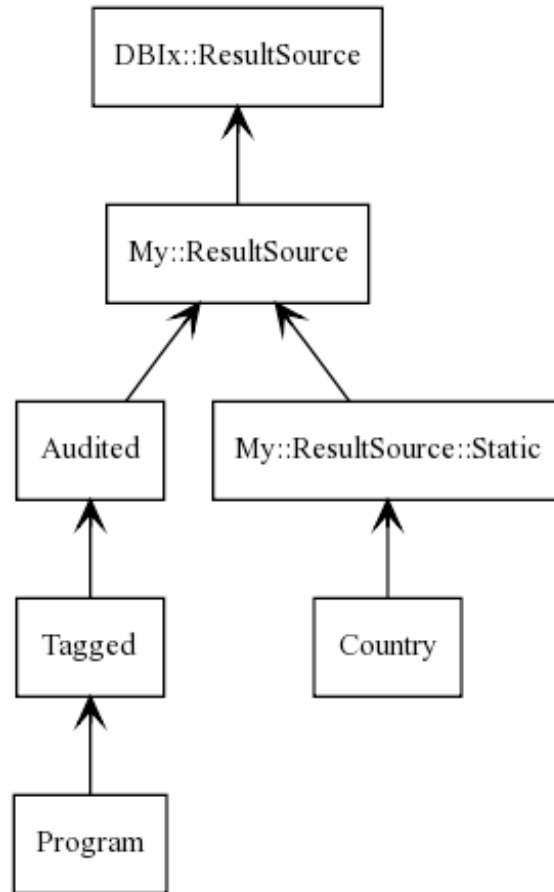
- 5,613 Brands
- 6,755 Series
- 386,943 Episodes
- 394,540 Versions
- 1,106,246 Broadcasts
- 1,701,309 On Demands

July 2, 2009

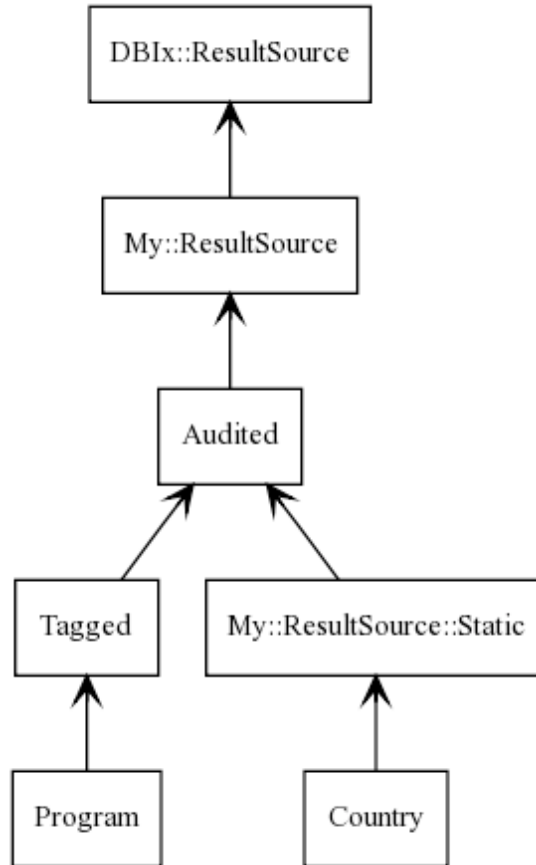
Real World Pain



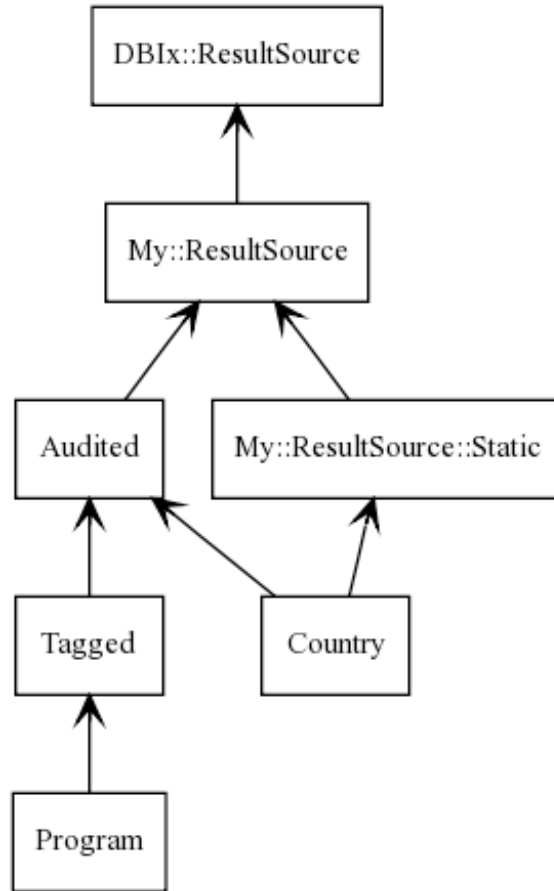
Real World Pain



Real World Pain



Real World Pain



The Problem

- Responsibility
 - Wants larger classes
- Reuse
 - Wants smaller classes

Solution

Decouple!

Solutions

- Interfaces
 - Reimplementing

Solutions

- Delegation
 - Scaffolding
 - Communication
 - Delegate to "Audited"?

Solutions

- Mixins
 - Ordering

PracticalJoke

- Needs:
 - explode ()
 - fuse ()

Shared Behavior

	Method	Description
✓	<code>Bomb::fuse()</code>	Deterministic
	<code>Spouse::fuse()</code>	Non-deterministic
	<code>Bomb::explode()</code>	Lethal
✓	<code>Spouse::explode()</code>	Wish it was lethal

Ruby Mixins

```
module Bomb
  def explode
    puts "Bomb explode"
  end
  def fuse
    puts "Bomb fuse"
  end
end

module Spouse
  def explode
    puts "Spouse explode"
  end
  def fuse
    puts "Spouse fuse"
  end
end
```

Ruby Mixins

```
class PracticalJoke
  include Spouse
  include Bomb
end

joke = PracticalJoke.new()
joke.fuse
joke.explode
```

Prints out:

Ruby Mixins

```
class PracticalJoke
  include Spouse
  include Bomb
end

joke = PracticalJoke.new()
joke.fuse
joke.explode
```

Prints out:

```
Bomb fuse
Bomb explode
```

Moose Roles

```
{  
  package Bomb;  
  use Moose::Role;  
  sub fuse      { say "Bomb fuse" }  
  sub explode  { say "Bomb explode" }  
}  
  
{  
  package Spouse;  
  use Moose::Role;  
  sub fuse      { say "Spouse fuse" }  
  sub explode  { say "Spouse explode" }  
}
```

Moose Roles

```
{  
    package PracticalJoke;  
    use Moose;  
    with qw(Bomb Spouse);  
}  
my $joke = PracticalJoke->new;  
$joke->fuse;  
$joke->explode;
```

Prints out:

Moose Roles

```
{  
  package PracticalJoke;  
  use Moose;  
  with qw(Bomb Spouse);  
}  
my $joke = PracticalJoke->new;  
$joke->fuse;  
$joke->explode;
```

Prints out:

Due to a method name conflict in roles 'Bomb' and 'Spouse', the method 'fuse' must be implemented or excluded by 'PracticalJoke'

Moose Roles

```
{  
  package PracticalJoke;  
  use Moose;  
  with 'Bomb'    => { excludes => 'explode' },  
      'Spouse' => { excludes => 'fuse' };  
}  
my $joke = PracticalJoke->new;  
$joke->fuse;  
$joke->explode;
```

Prints out:

```
Bomb fuse  
Spouse explode
```

Moose Roles

```
package PracticalJoke;
use Moose;
with 'Bomb'    => { excludes => 'explode' },
    'Spouse' => { excludes => 'fuse',
                  alias    =>
                    { fuse => 'random_fuse' }
    };
```

And in your actual code:

```
$joke->fuse(14);      # timed fuse
# or
$joke->random_fuse; # who knows?
```

Role Example

```
package Does::Serialize::YAML
use Moose::Role;
use YAML::Syck;

requires 'as_hash';

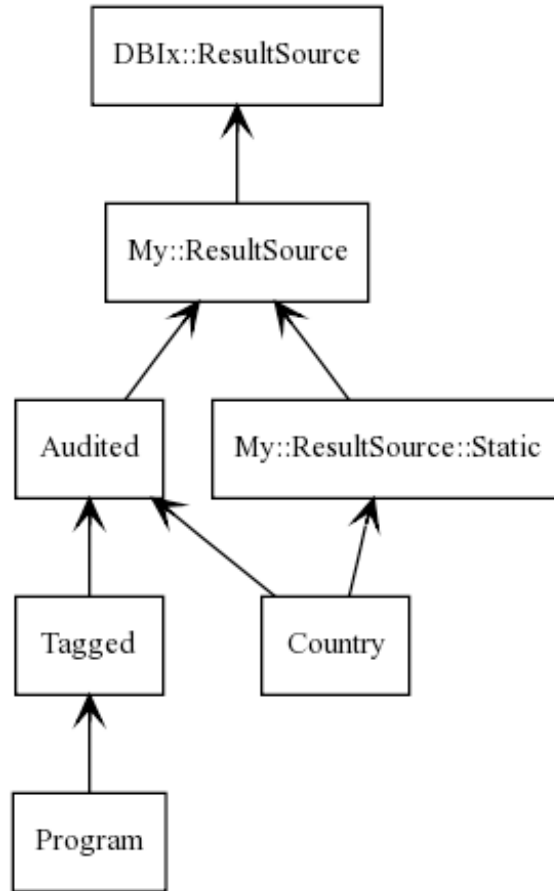
sub serialize {
    my $self = shift;
    return Dump($self->as_hash);
}

1;
```

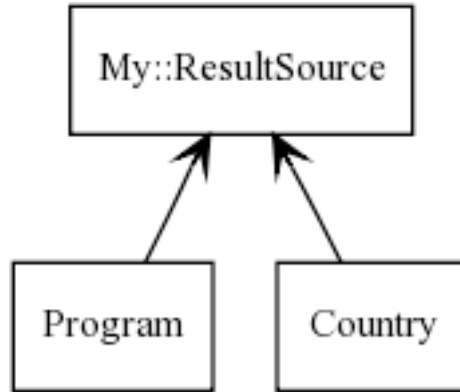
And in your actual code:

```
package My::Object;
use Moose;
with 'Does::Serialize::YAML';
```

Back To Work



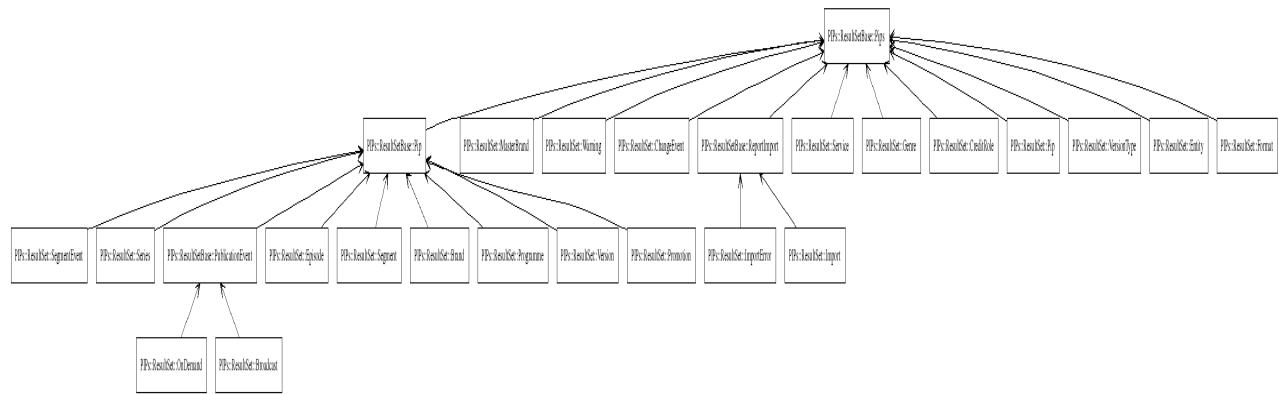
Work + Roles



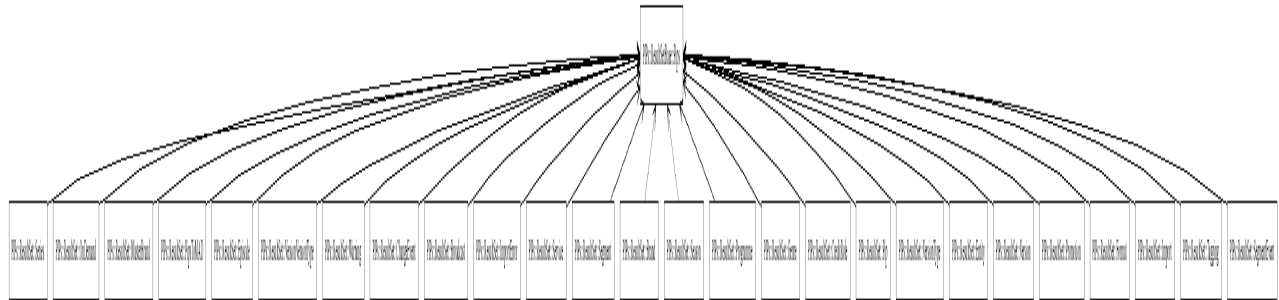
Work + Roles

```
package Country;  
use Moose;  
extends "My::ResultSource";  
with qw(DoesStatic DoesAuditing);
```

Old BBC ResultSet Classes



New ResultSet Classes



Work + Roles

```
package BBC::Programme::Episode;  
use Moose;  
extends 'BBC::ResultSet';  
with qw(  
    DoesSearch::Broadcasts  
    DoesSearch::Tags  
    DoesSearch::Titles  
    DoesSearch::Promotions  
    DoesIdentifier::Universal  
);
```

Conclusion

- Separates reuse and responsibility
- Increases comprehension
- Increases safety
- Decreases complexity

Extra!

- **Conflicting methods in roles**
- **State in roles**
- **Tool support**

Solutions

- AOP
 - Augment, not add
 - Abused
 - Terminology wankery
 - In joinpoints, no one can hear you scream