use Moose;
use Test::Routine;

$self->tdd;

Raf Gemmail
@fiqus
Wellington.pm
October 2014

use Moose;
use Test::Routine;

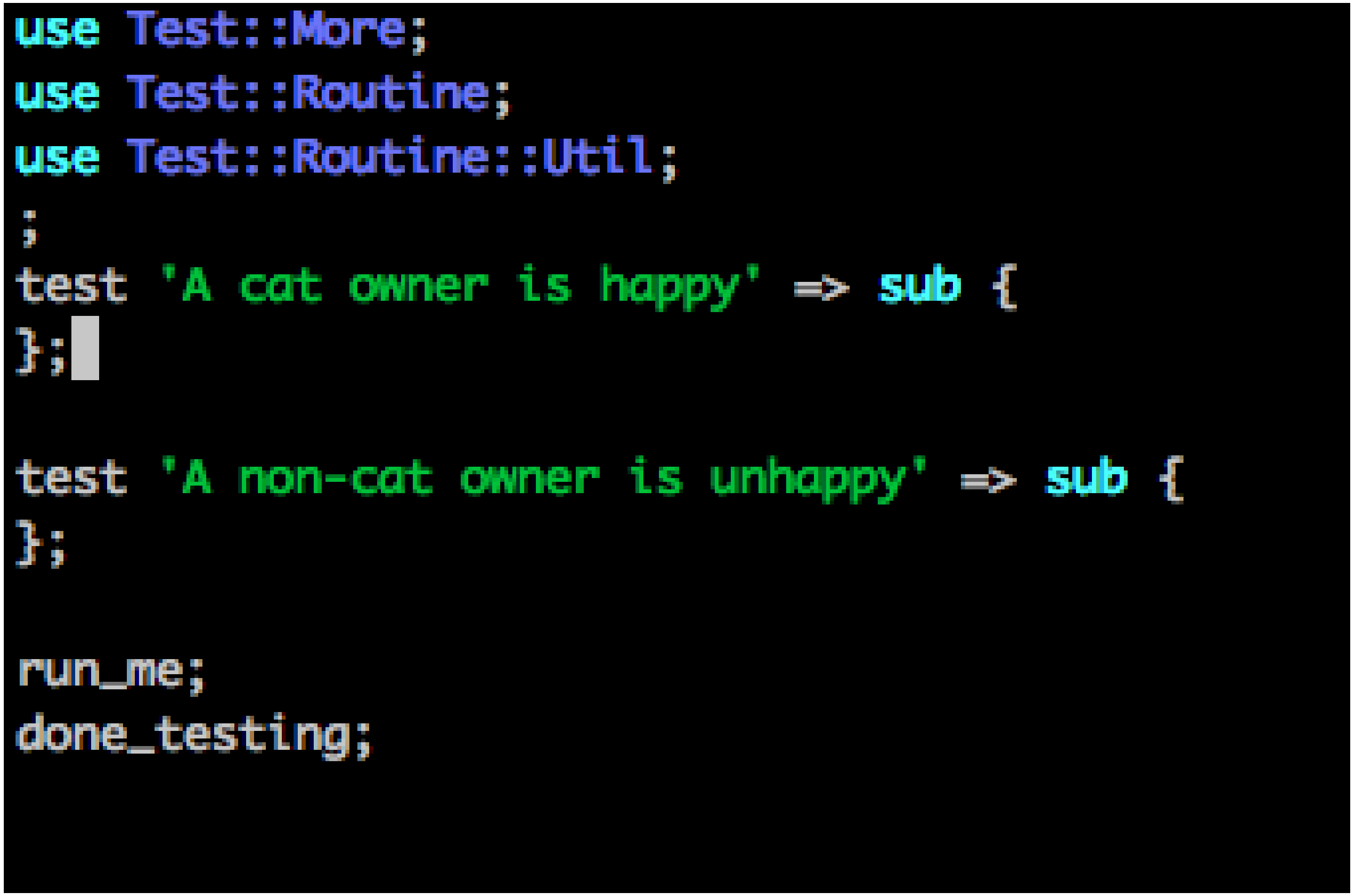
$self->td_design;

Raf Gemmail
@fiqus
Wellington.pm
October 2014

Story 1

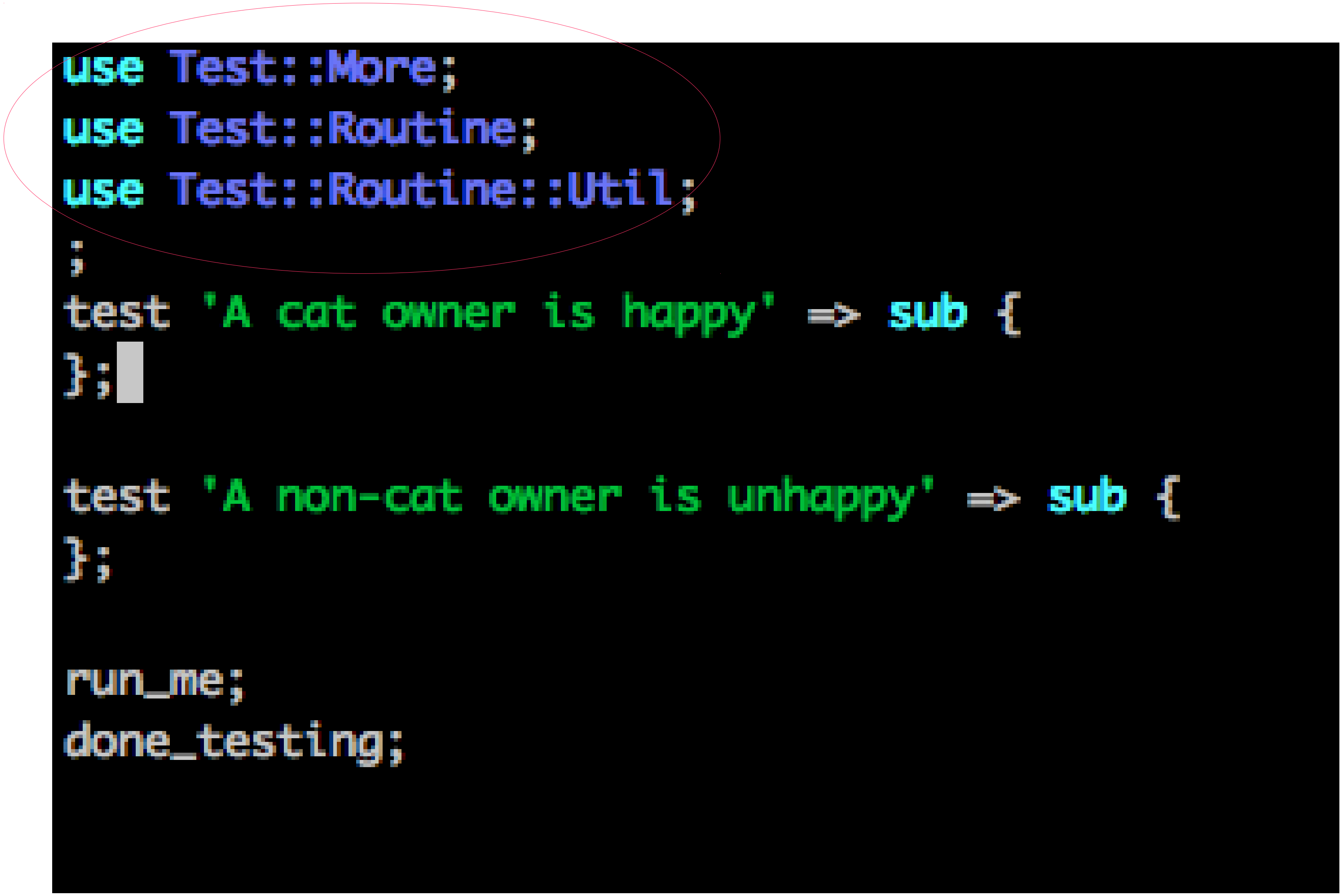
As an Animal Lover
I want a model of a Cat Owner
So that I can show that cats make you happy

Capture some AC's



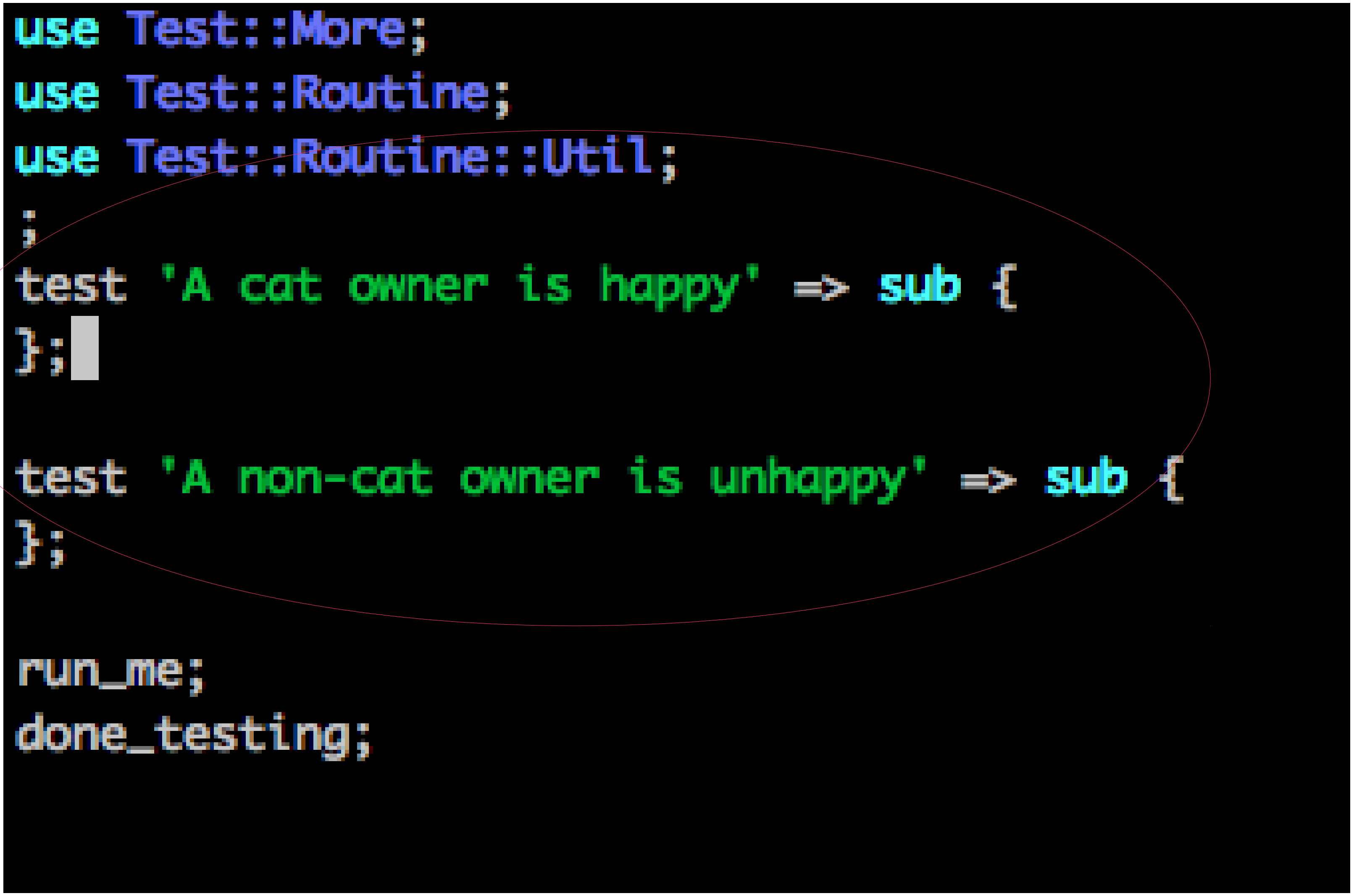
```
use Test::More;
use Test::Routine;
use Test::Routine::Util;
;
test 'A cat owner is happy' => sub {
};

test 'A non-cat owner is unhappy' => sub {
};


run_me;
done_testing;
```

Capture some AC's



```
use Test::More;
use Test::Routine;
use Test::Routine::Util;
;
test 'A cat owner is happy' => sub {
};

test 'A non-cat owner is unhappy' => sub {
};

run_me;
done_testing;
```

Capture some AC's



```
use Test::More;
use Test::Routine;
use Test::Routine::Util;
;
test 'A cat owner is happy' => sub {
};

test 'A non-cat owner is unhappy' => sub {
};

run_me;
done_testing;
```

Concepts

- Role – behavioural mixin

Concepts

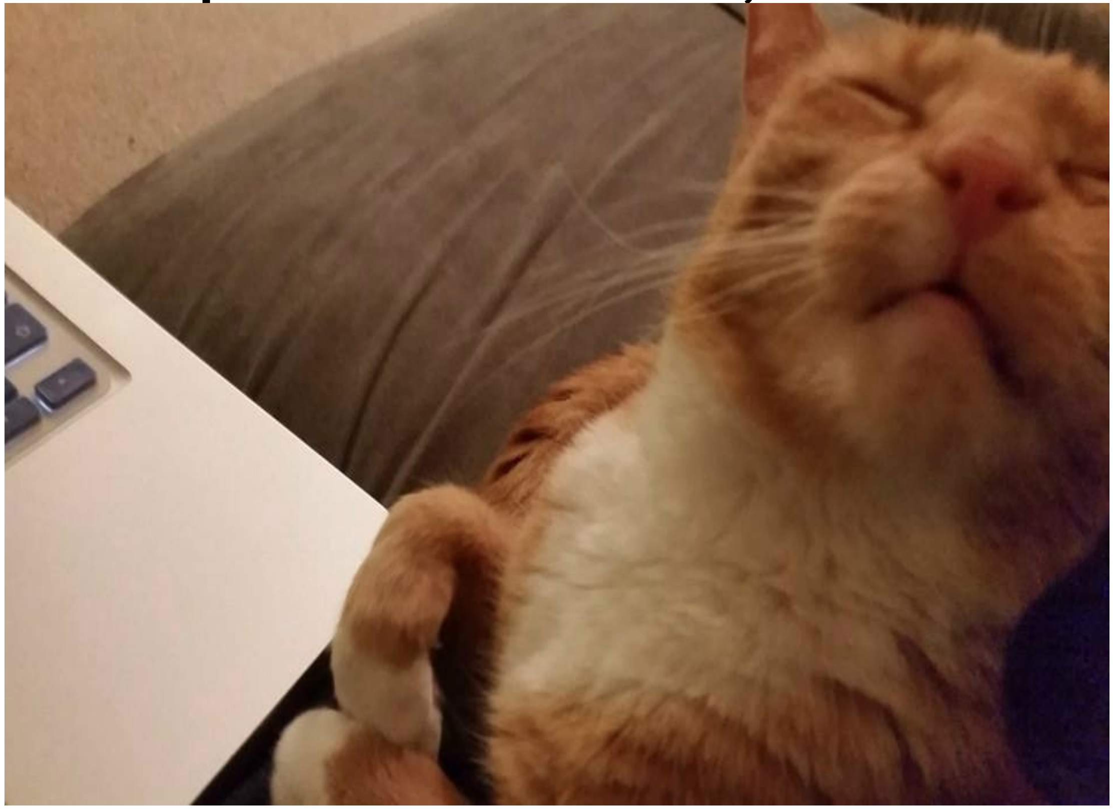
- Role – behavioural mixin
- Test::Routine → role with fixtures and/or tests

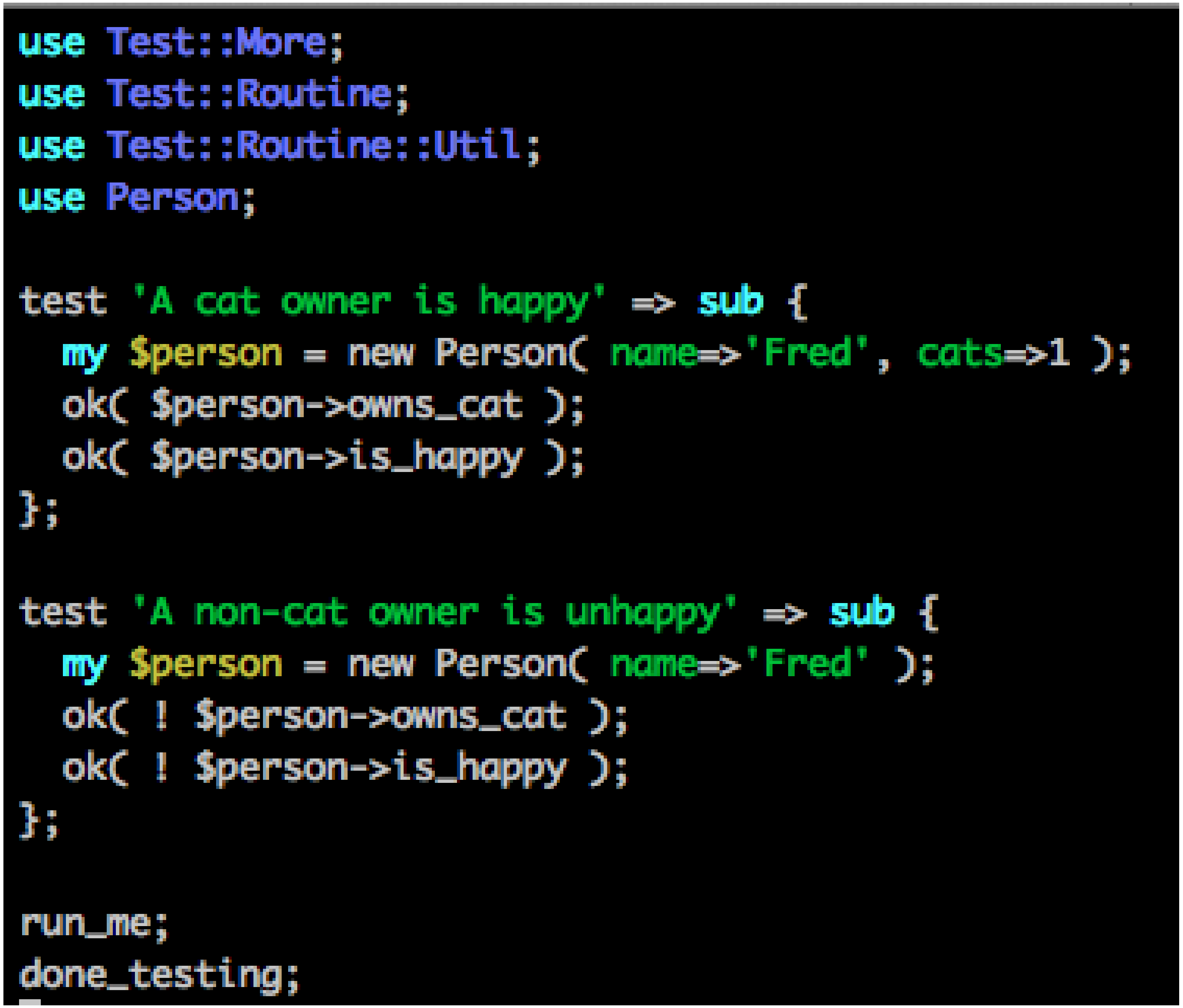
Concepts

- Role – behavioural mixin
- Test::Routine → role with fixtures and/or tests
- Test::Routine::Util exposes run_me, run_tests

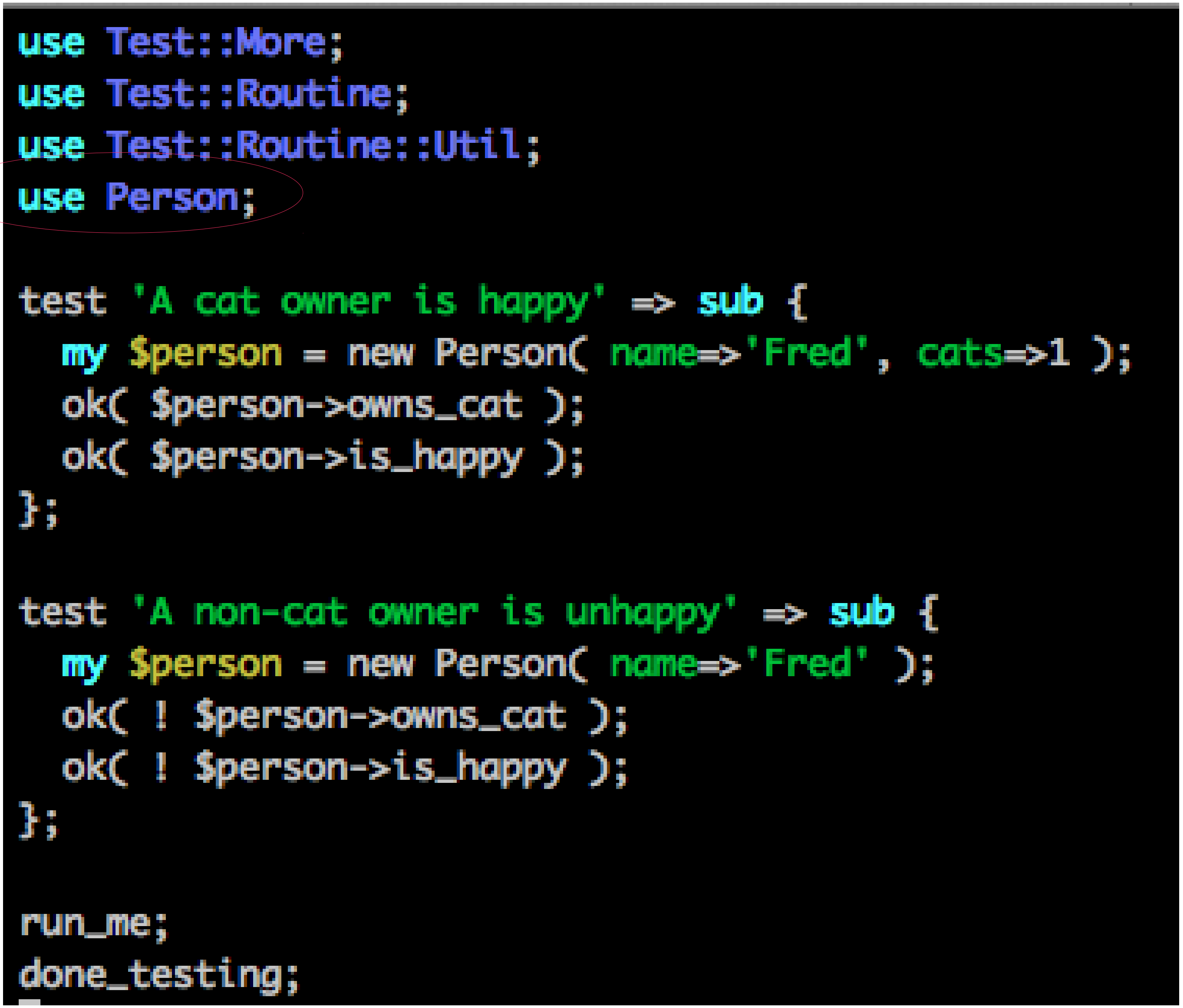
Concepts

- Role – behavioural mixin
- Test::Routine → role with fixtures and/or tests
- Test::Routine::Util exposes run_me, run_tests
- Cat

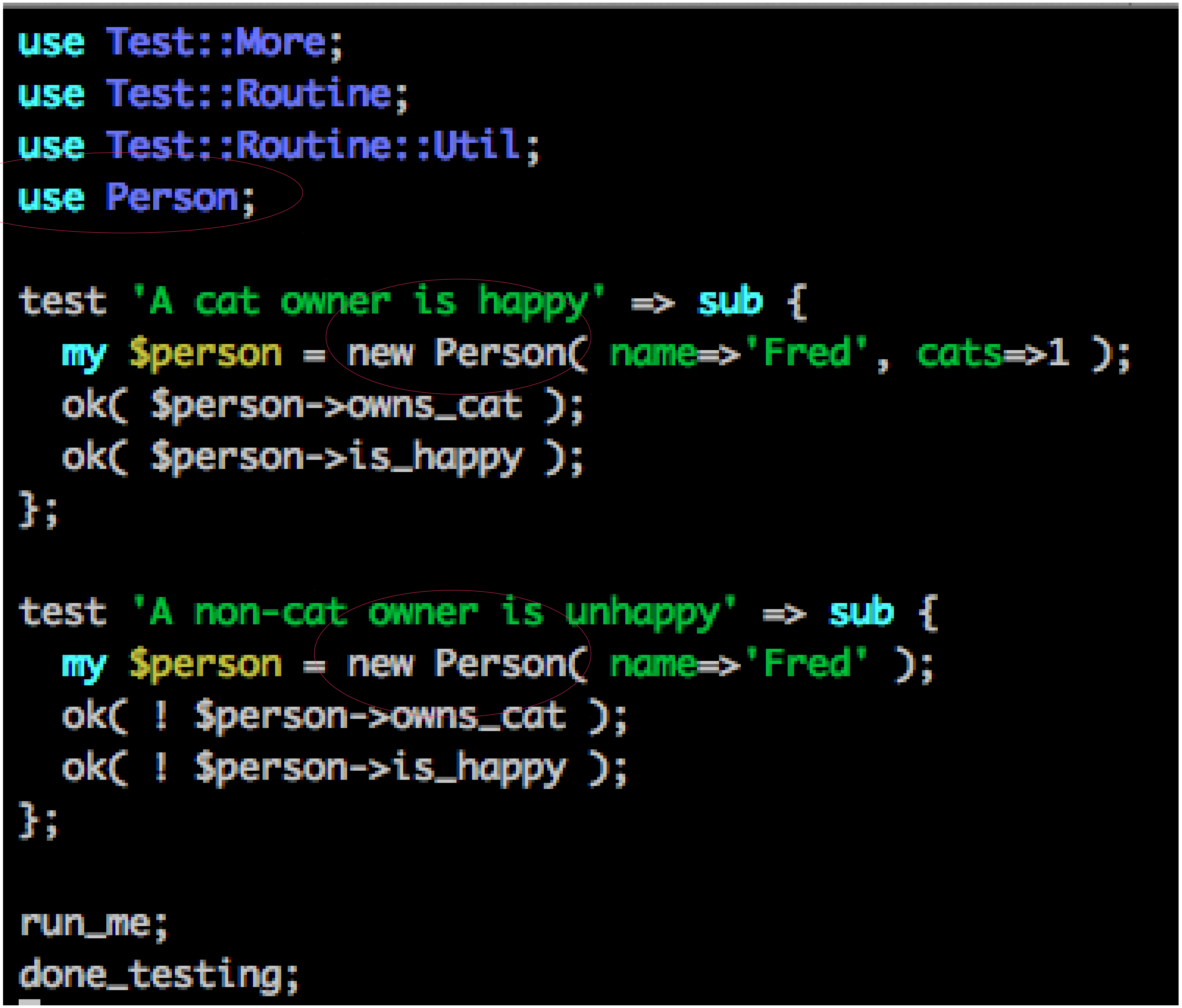




```
use Test::More;
use Test::Routine;
use Test::Routine::Util;
use Person;

test 'A cat owner is happy' => sub {
  my $person = new Person( name=>'Fred', cats=>1 );
  ok( $person->owns_cat );
  ok( $person->is_happy );
};

test 'A non-cat owner is unhappy' => sub {
  my $person = new Person( name=>'Fred' );
  ok( ! $person->owns_cat );
  ok( ! $person->is_happy );
};

run_me;
done_testing;
```



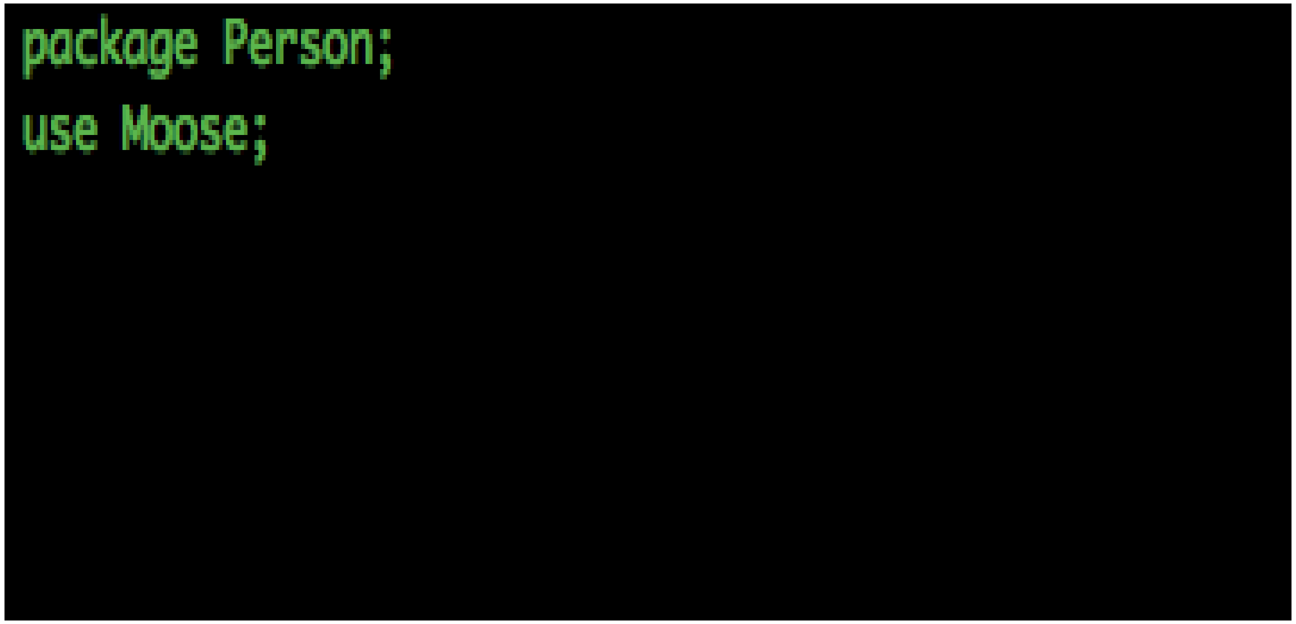
```
use Test::More;
use Test::Routine;
use Test::Routine::Util;
use Person;

test 'A cat owner is happy' => sub {
   my $person = new Person( name=>'Fred', cats=>1 );
   ok( $person->owns_cat );
   ok( $person->is_happy );
};

test 'A non-cat owner is unhappy' => sub {
   my $person = new Person( name=>'Fred' );
   ok( ! $person->owns_cat );
   ok( ! $person->is_happy );
};

run_me;
done_testing;
```



```
use Test::More;
use Test::Routine;
use Test::Routine::Util;
use Person;

test 'A cat owner is happy' => sub {
   my $person = new Person( name=>'Fred', cats=>1 );
   ok( $person->owns_cat );
   ok( $person->is_happy );
};

test 'A non-cat owner is unhappy' => sub {
   my $person = new Person( name=>'Fred' );
   ok( ! $person->owns_cat );
   ok( ! $person->is_happy );
};

run_me;
done_testing;
```

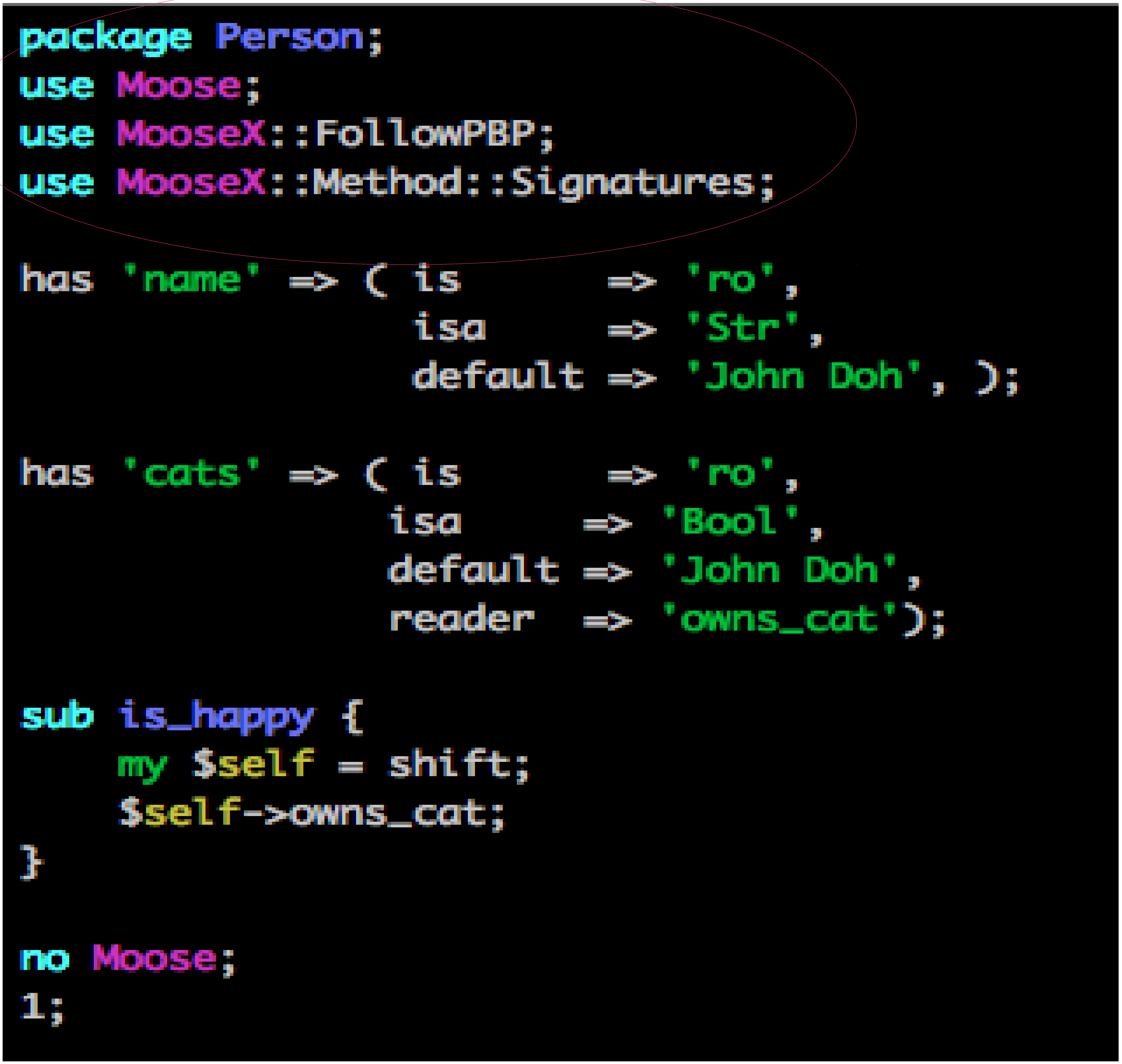
Real software is never perfect..

Lets make a person

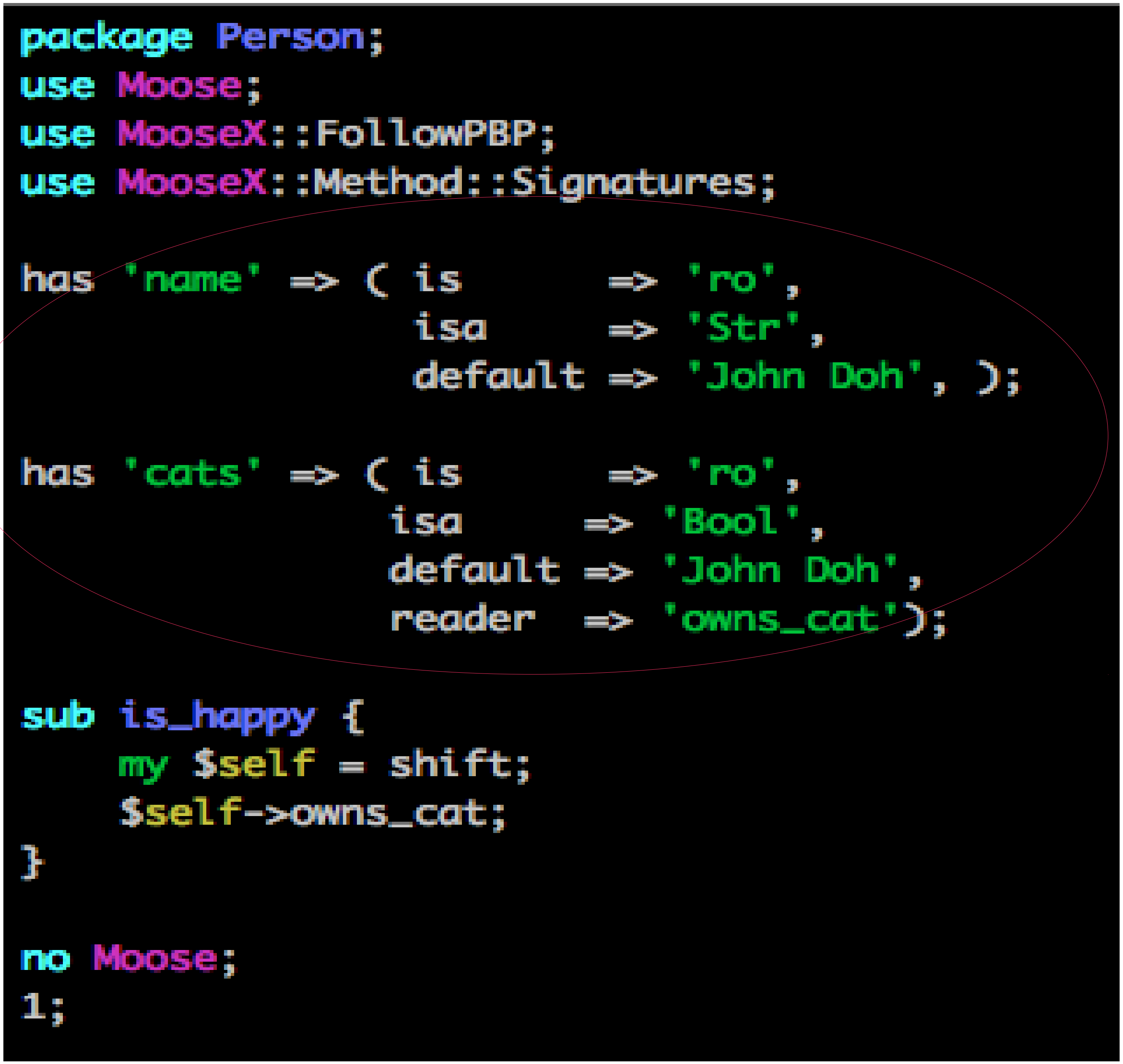


```
package Person;
use Moose;
```

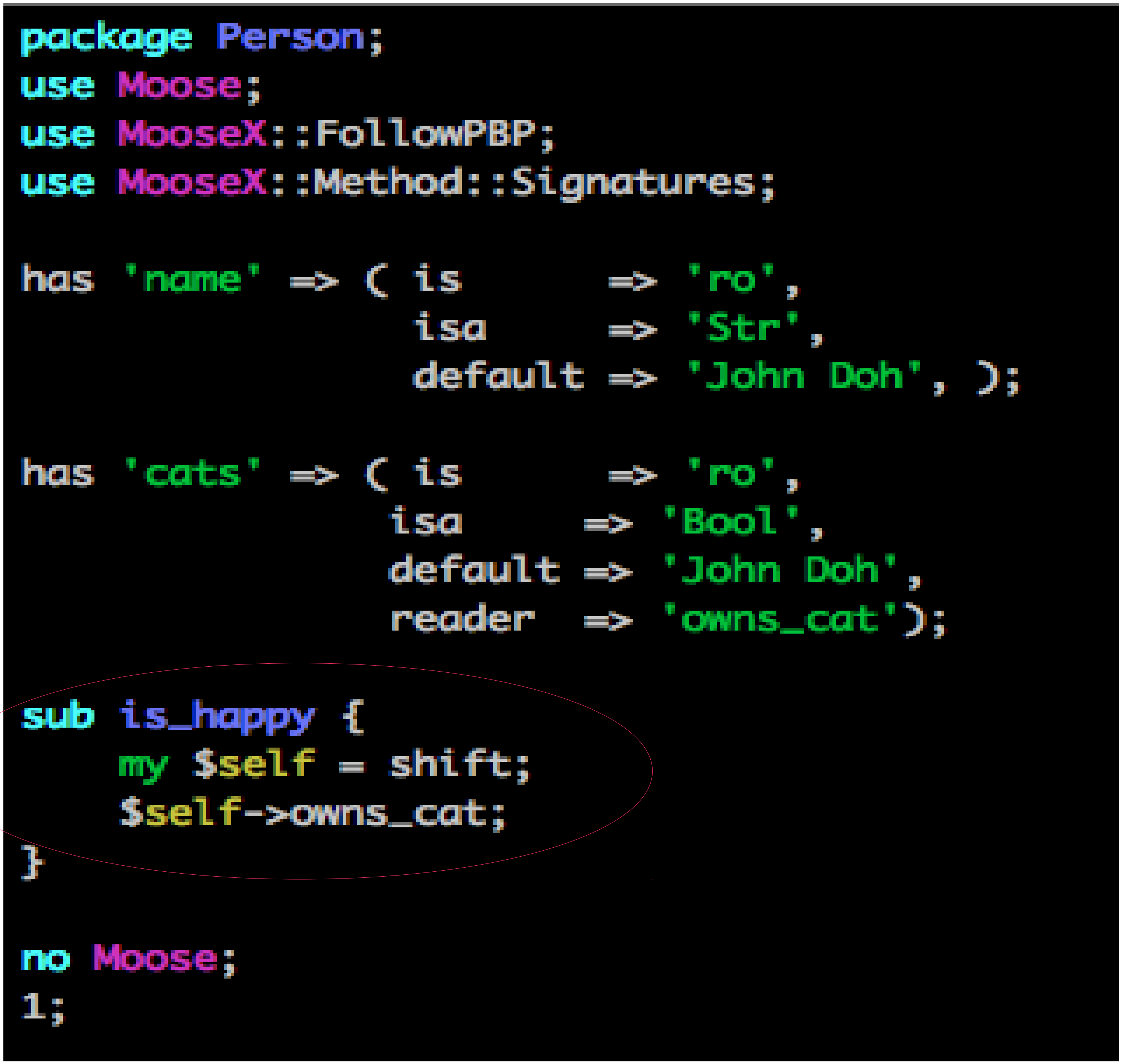
```perl
package Person;
use Moose;
use MooseX::FollowPBP;
use MooseX::Method::Signatures;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'John Doh', );

has 'cats' => ( is      => 'ro',
               isa     => 'Bool',
               default => 'John Doh',
               reader  => 'owns_cat');

sub is_happy {
    my $self = shift;
    $self->owns_cat;
}

no Moose;
1;
```



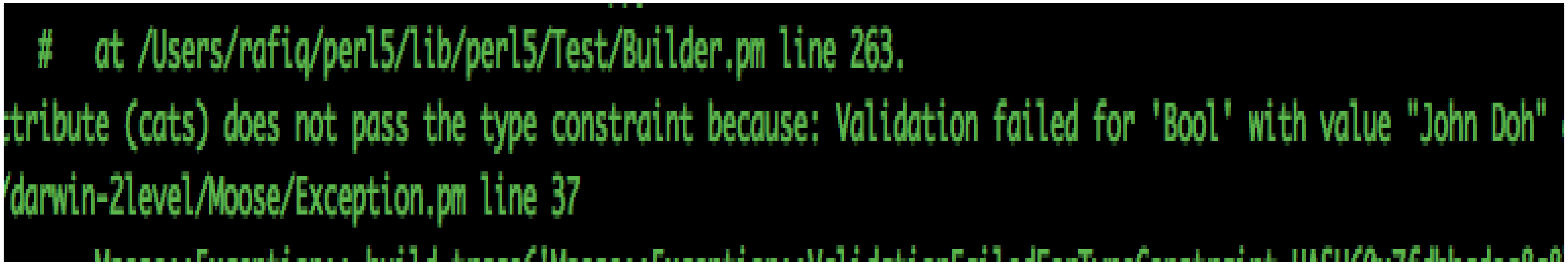
```
package Person;
use Moose;
use MooseX::FollowPBP;
use MooseX::Method::Signatures;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'John Doh', );

has 'cats' => ( is      => 'ro',
               isa     => 'Bool',
               default => 'John Doh',
               reader  => 'owns_cat');

sub is_happy {
    my $self = shift;
    $self->owns_cat;
}

no Moose;
1;
```



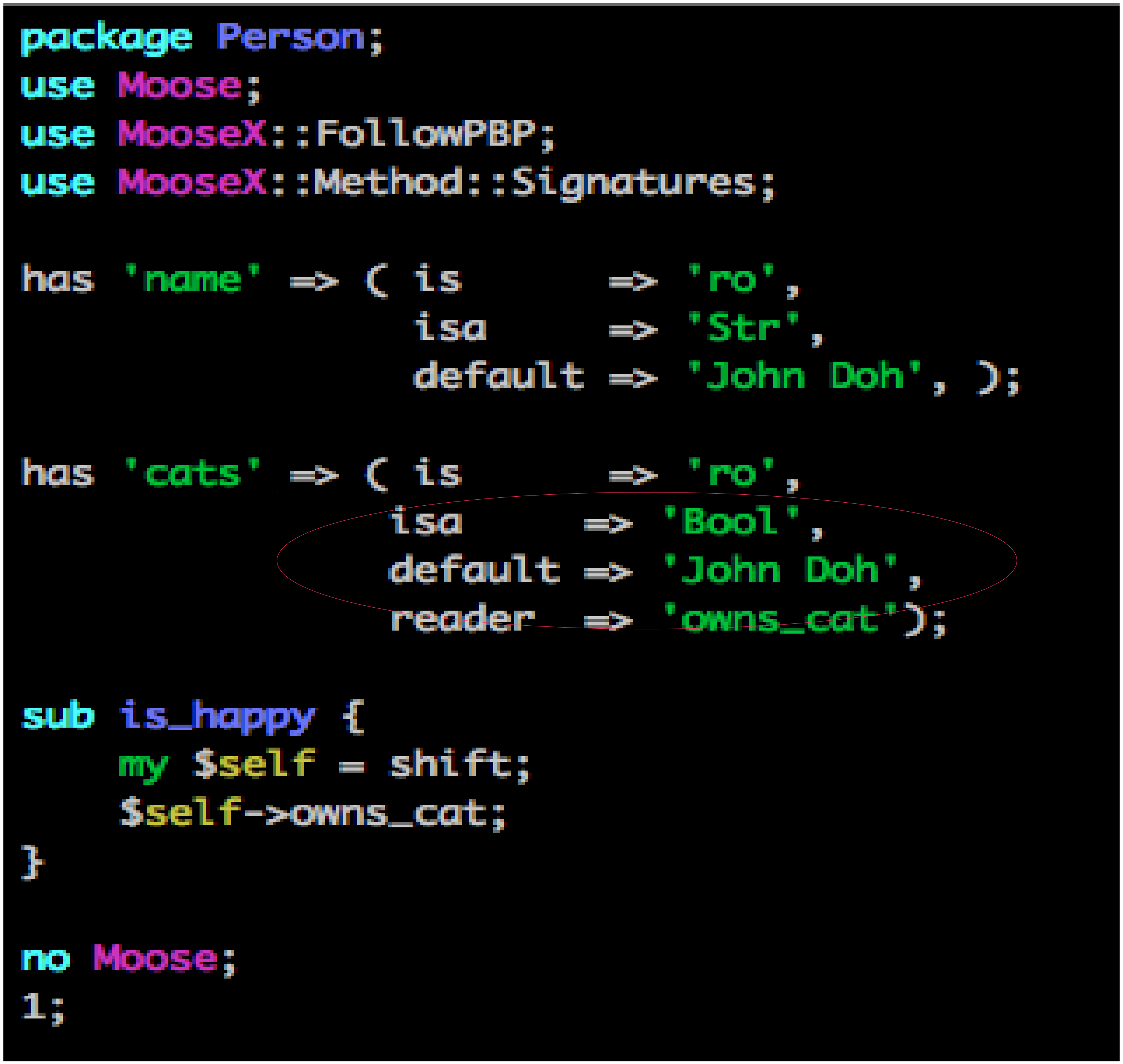
```
package Person;
use Moose;
use MooseX::FollowPBP;
use MooseX::Method::Signatures;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'John Doh', );

has 'cats' => ( is      => 'ro',
               isa     => 'Bool',
               default => 'John Doh',
               reader  => 'owns_cat');

sub is_happy {
    my $self = shift;
    $self->owns_cat;
}

no Moose;
1;
```



```perl
package Person;
use Moose;
use MooseX::FollowPBP;
use MooseX::Method::Signatures;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'John Doh', );

has 'cats' => ( is      => 'ro',
               isa     => 'Bool',
               default => 'John Doh',
               reader  => 'owns_cat');

sub is_happy {
    my $self = shift;
    $self->owns_cat;
}

no Moose;
1;
```

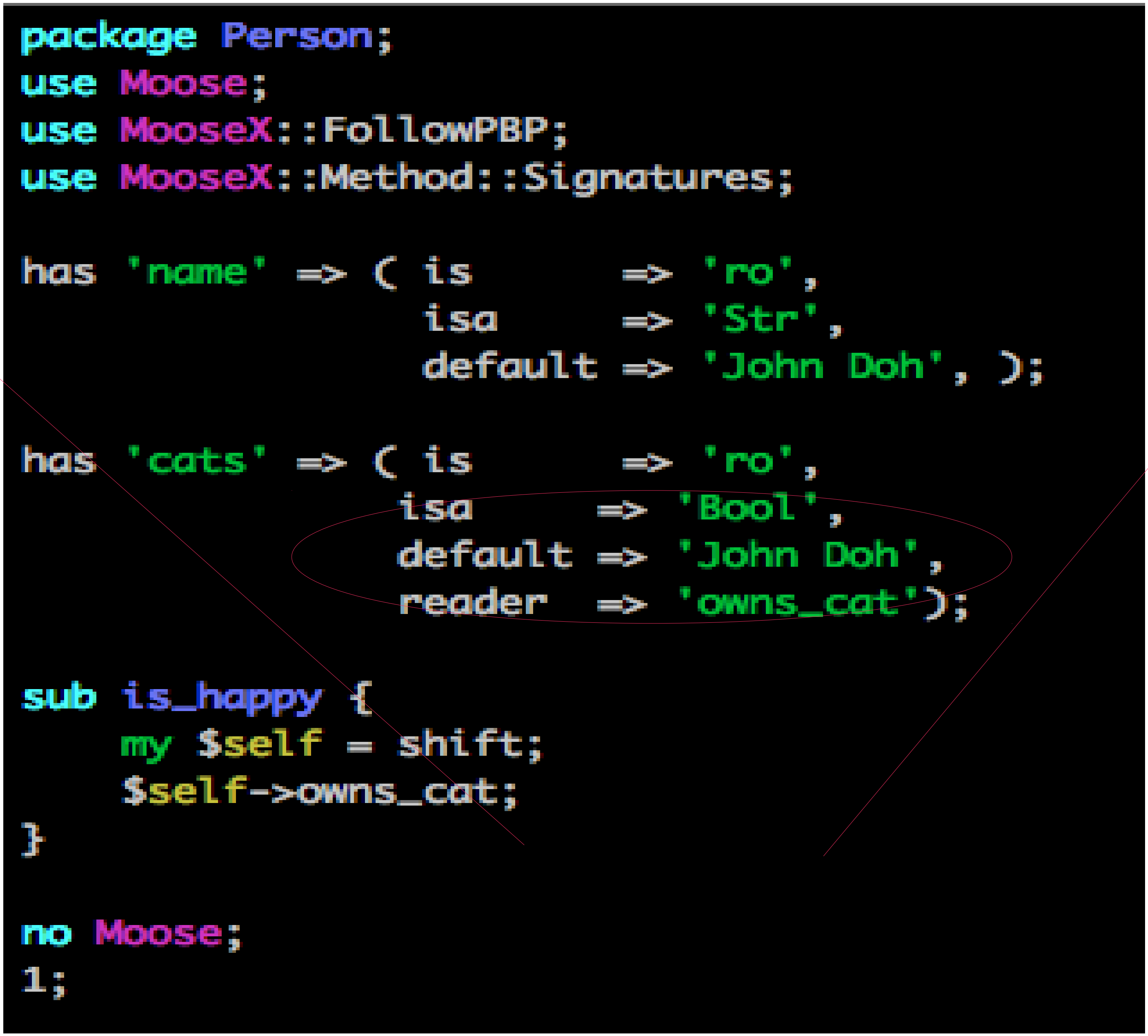


```
  #   at /Users/rafiq/perl5/lib/perl5/Test/Builder.pm line 263.
ttribute (cats) does not pass the type constraint because: Validation failed for 'Bool' with value "John Doh"
/darwin-2level/Moose/Exception.pm line 37
```

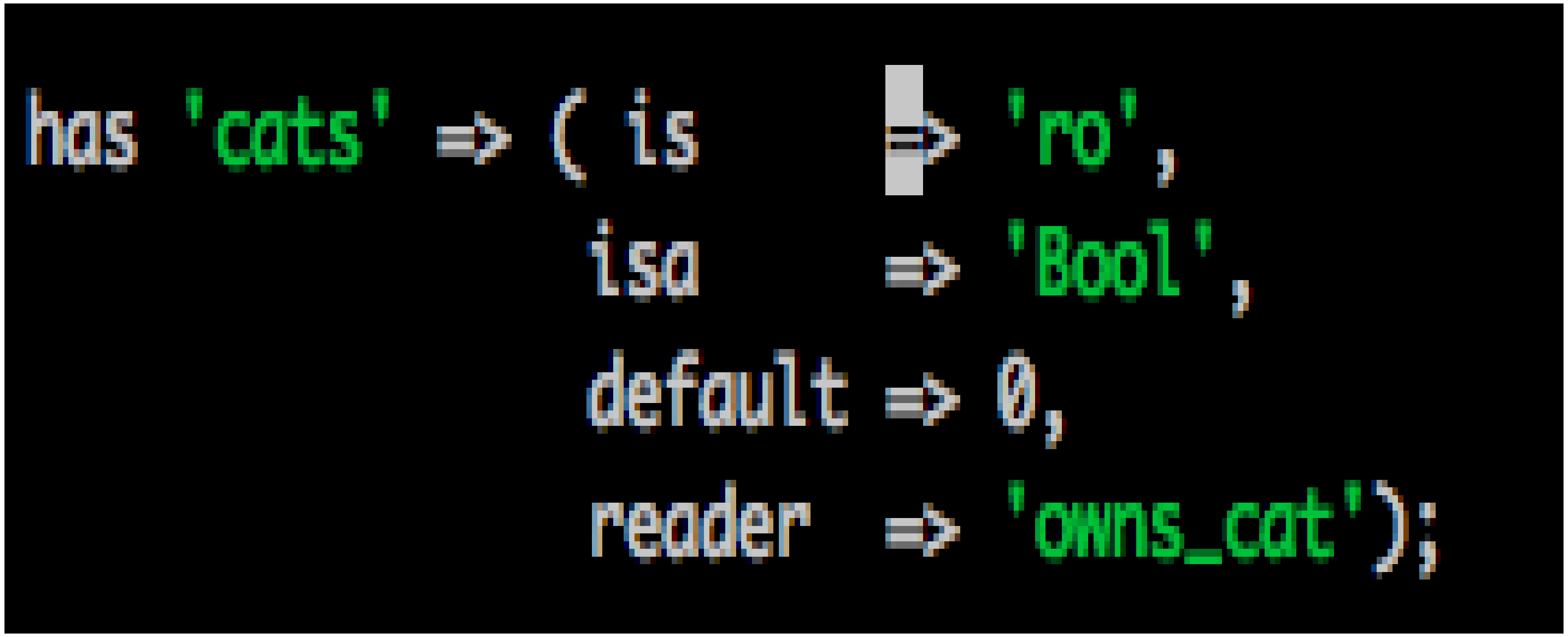


```
package Person;
use Moose;
use MooseX::FollowPBP;
use MooseX::Method::Signatures;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'John Doh', );

has 'cats' => ( is      => 'ro',
               isa     => 'Bool',
               default => 'John Doh',
               reader  => 'owns_cat');

sub is_happy {
    my $self = shift;
    $self->owns_cat;
}

no Moose;
1;
```

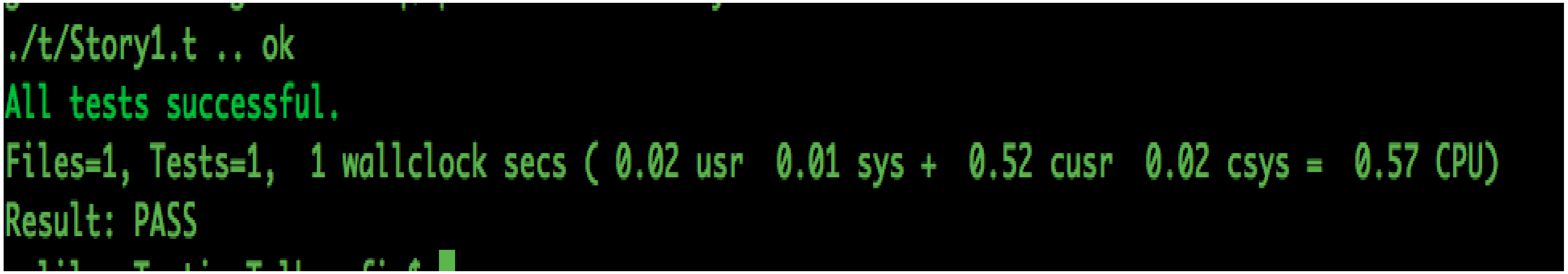
Copy Paste!



```
package Person;
use Moose;
use MooseX::FollowPBP;
use MooseX::Method::Signatures;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'John Doh', );

has 'cats' => ( is      => 'ro',
               isa     => 'Bool',
               default => 'John Doh',
               reader  => 'owns_cat');


sub is_happy {
    my $self = shift;
    $self->owns_cat;
}

no Moose;
1;
```



```
has 'cats' => ( is      => 'ro',
                isa     => 'Bool',
                default => 0,
                reader  => 'owns_cat');
```



```
./t/Story1.t .. ok
All tests successful.
Files=1, Tests=1,  1 wallclock secs ( 0.02 usr  0.01 sys +  0.52 cusr  0.02 csys =  0.57 CPU)
Result: PASS
```

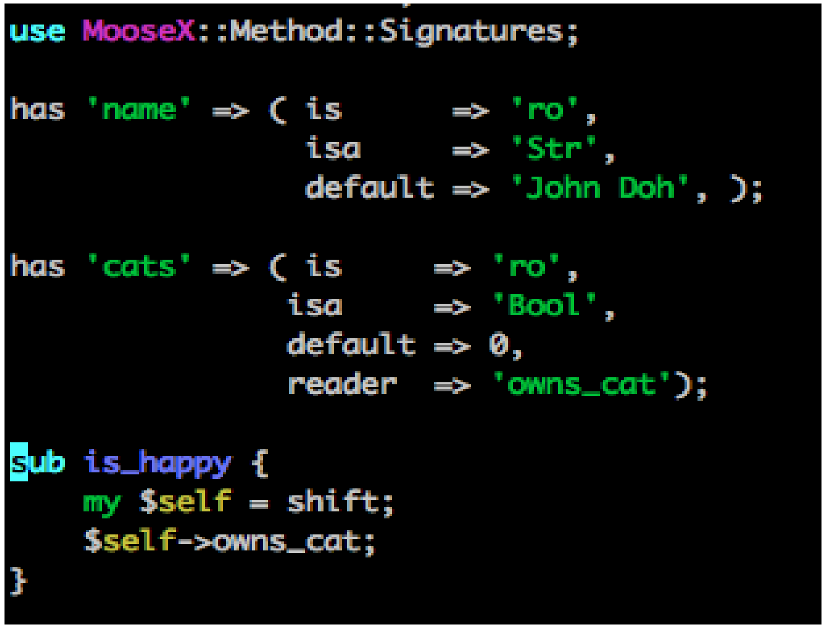
Story 2

As a cat
I want to be able to put on my grump
So that I can make my owner miserable

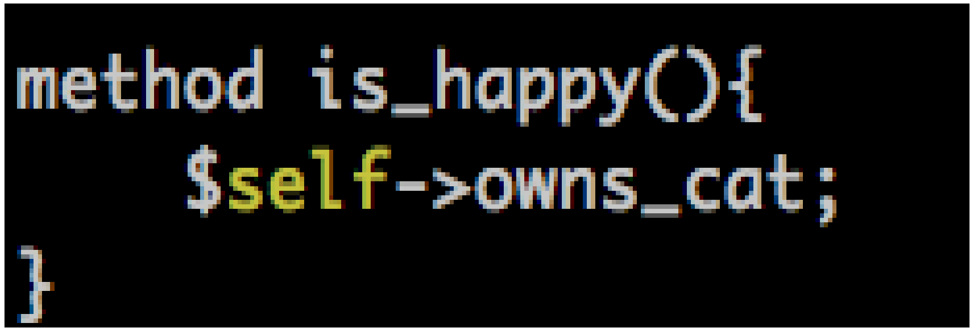
Story 2

I wa np
So th ble

Fear-free Refactor

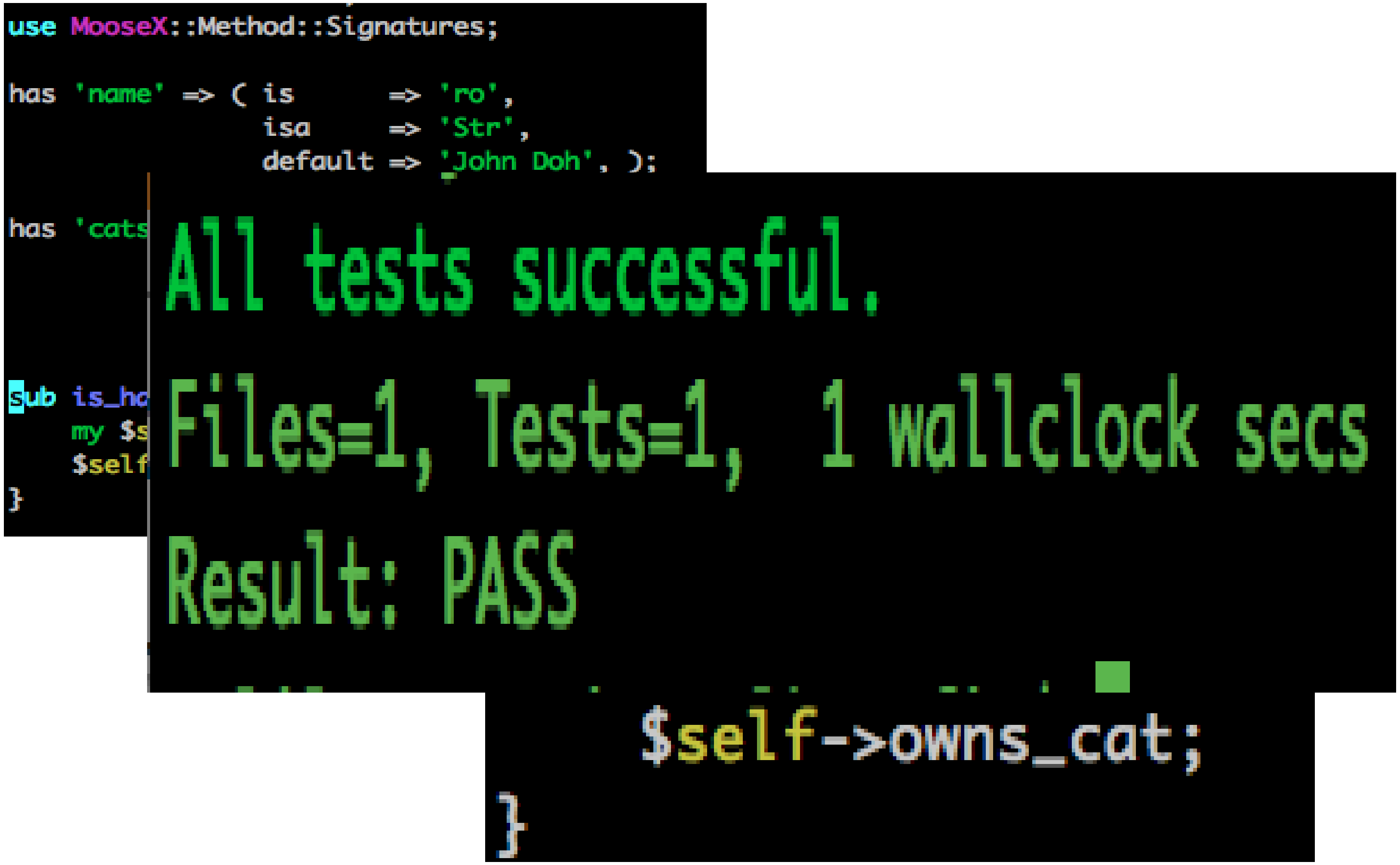


```
use MooseX::Method::Signatures;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'John Doh', );

has 'cats' => ( is      => 'ro',
               isa     => 'Bool',
               default => 0,
               reader  => 'owns_cat');

sub is_happy {
    my $self = shift;
    $self->owns_cat;
}
```



```
method is_happy(){
    $self->owns_cat;
}
```

Fear-free Refactor

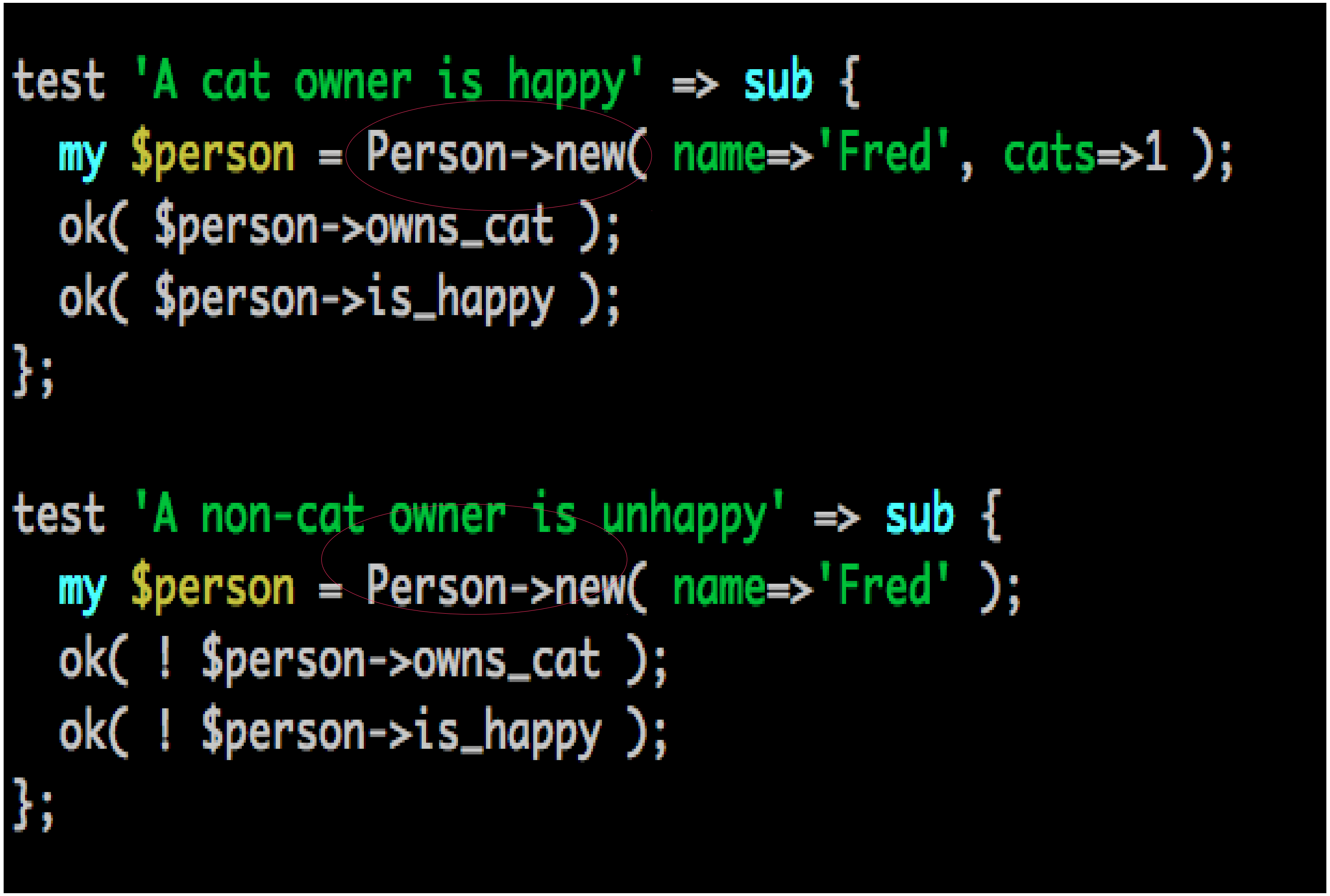


```
use MooseX::Method::Signatures;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'John Doh', );

has 'cats

sub is_ho
    my $s
    $self
}
```

```
All tests successful.
Files=1, Tests=1,  1 wallclock secs
Result: PASS
```

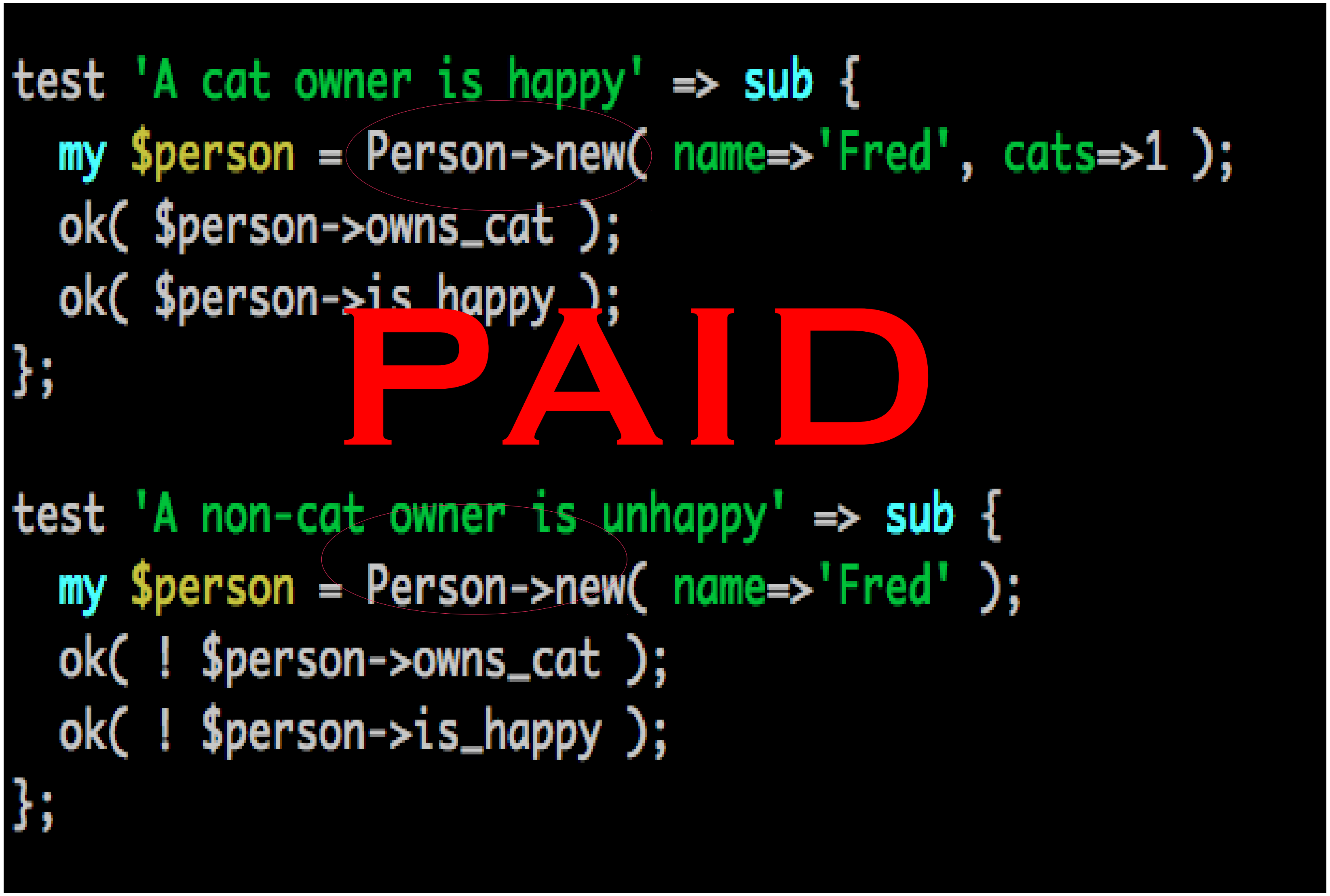
```
    $self->owns_cat;
}
```

It's OK to fix your tests..



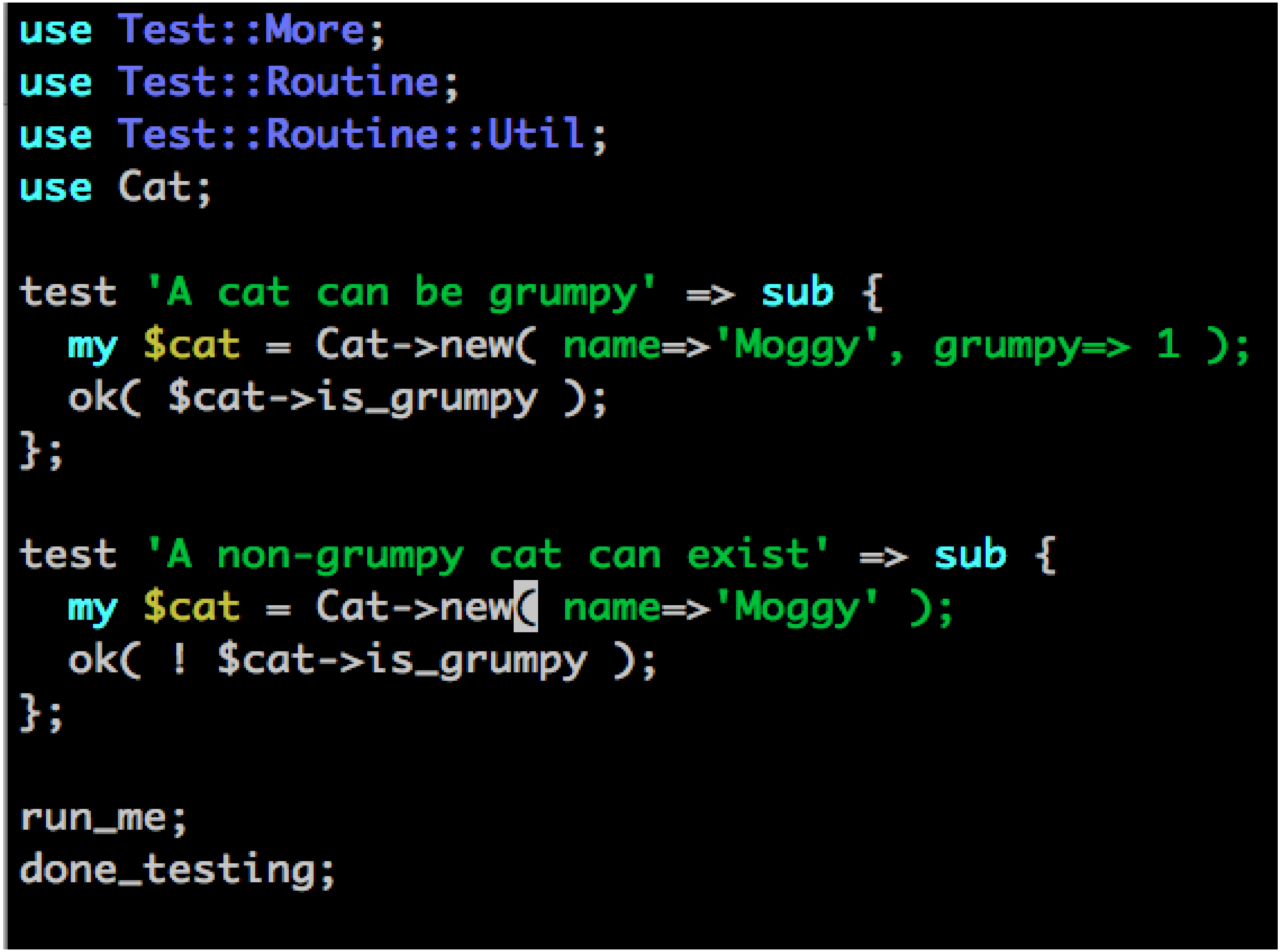
```
test 'A cat owner is happy' => sub {
  my $person = Person->new( name=>'Fred', cats=>1 );
  ok( $person->owns_cat );
  ok( $person->is_happy );
};

test 'A non-cat owner is unhappy' => sub {
  my $person = Person->new( name=>'Fred' );
  ok( ! $person->owns_cat );
  ok( ! $person->is_happy );
};
```

And pay off some debt



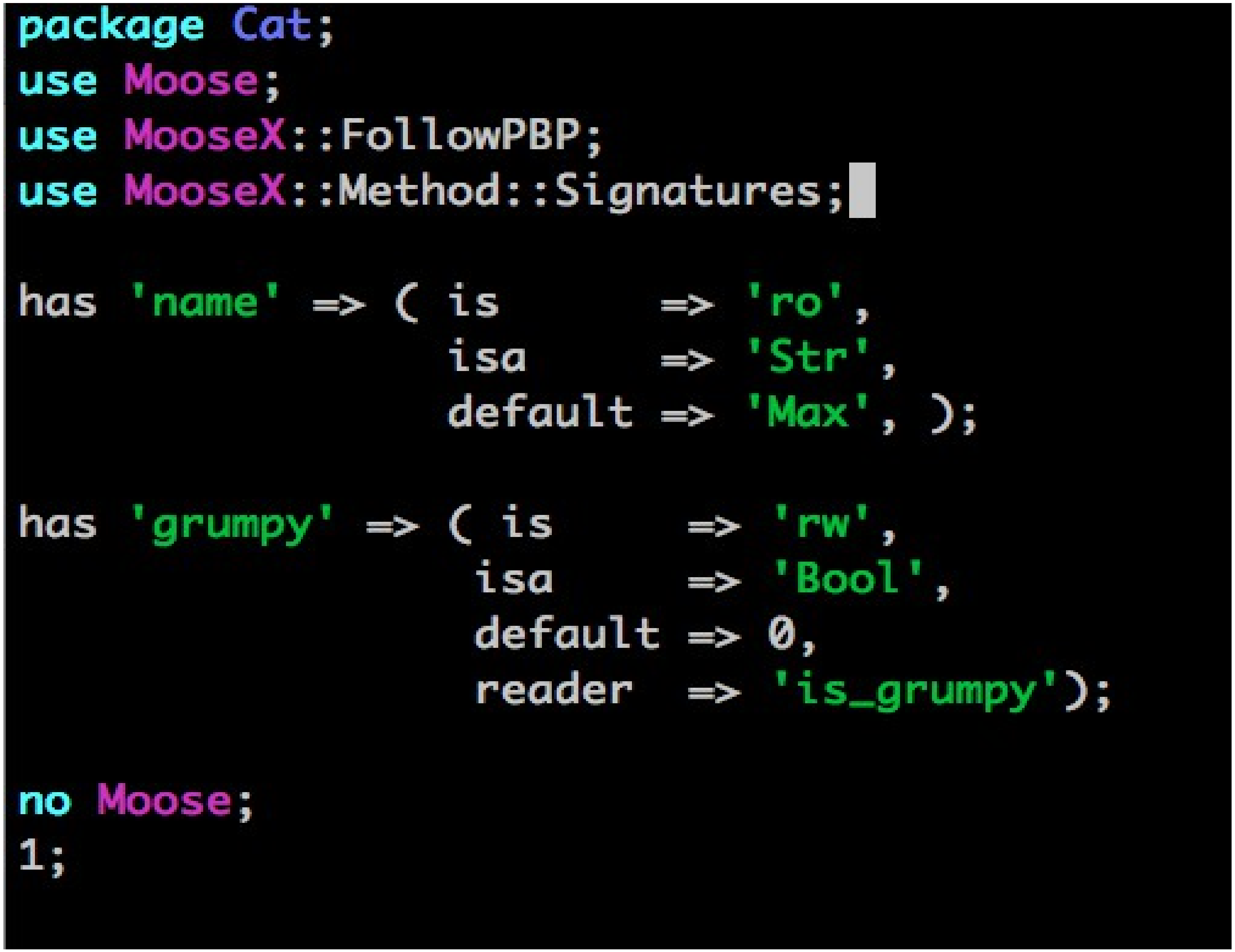
```
test 'A cat owner is happy' => sub {
  my $person = Person->new( name=>'Fred', cats=>1 );
  ok( $person->owns_cat );
  ok( $person->is_happy );
};

test 'A non-cat owner is unhappy' => sub {
  my $person = Person->new( name=>'Fred' );
  ok( ! $person->owns_cat );
  ok( ! $person->is_happy );
};
```

Understand our Cat



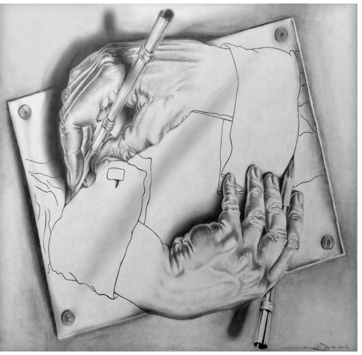
```
use Test::More;
use Test::Routine;
use Test::Routine::Util;
use Cat;

test 'A cat can be grumpy' => sub {
  my $cat = Cat->new( name=>'Moggy', grumpy=> 1 );
  ok( $cat->is_grumpy );
};

test 'A non-grumpy cat can exist' => sub {
  my $cat = Cat->new( name=>'Moggy' );
  ok( ! $cat->is_grumpy );
};

run_me;
done_testing;
```

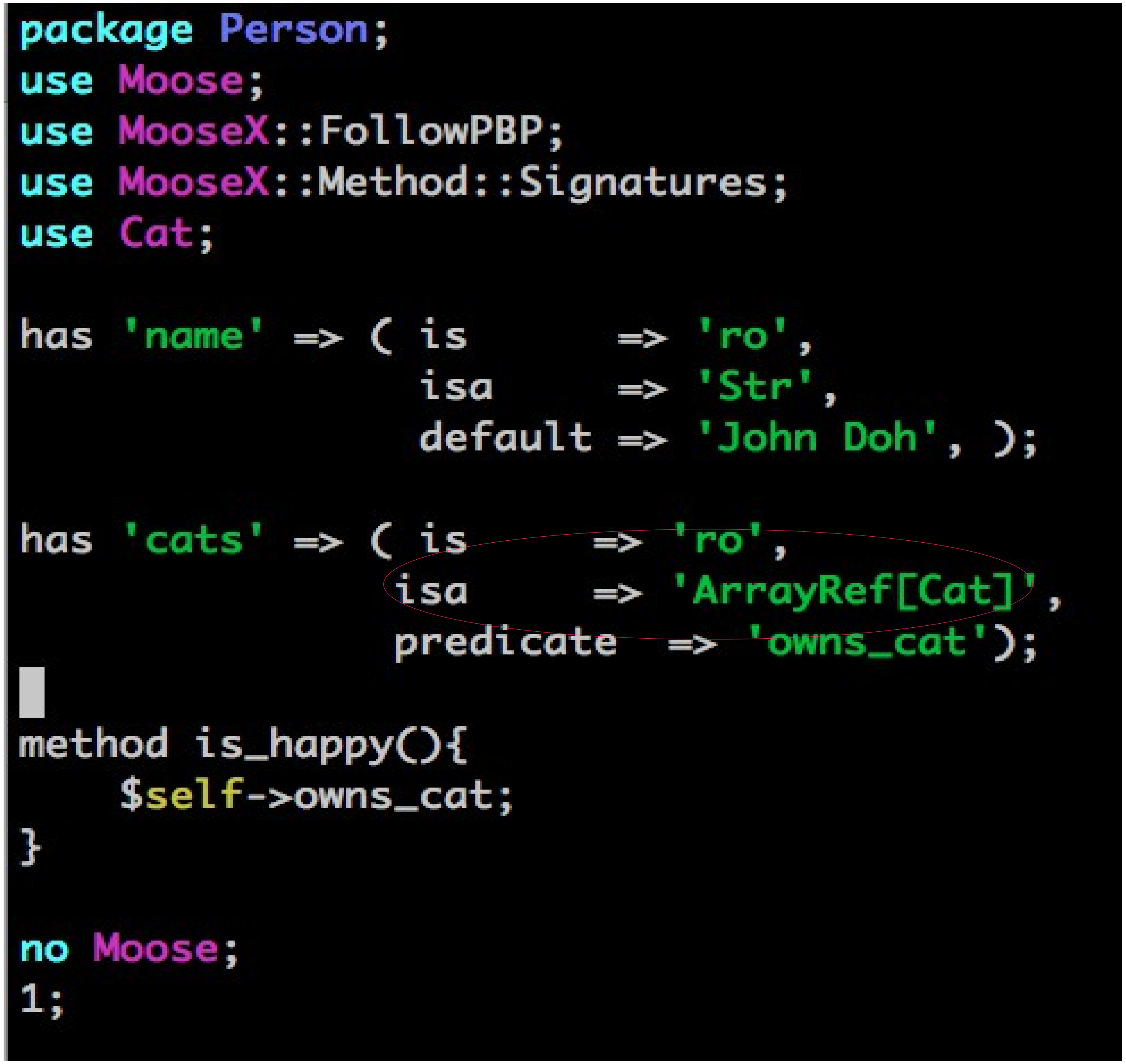
Factor out the Cat



```
package Cat;
use Moose;
use MooseX::FollowPBP;
use MooseX::Method::Signatures;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'Max', );

has 'grumpy' => ( is      => 'rw',
                  isa     => 'Bool',
                  default => 0,
                  reader  => 'is_grumpy');


no Moose;
1;
```

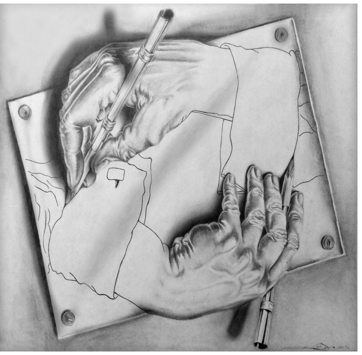
Person is a composition with a Cat

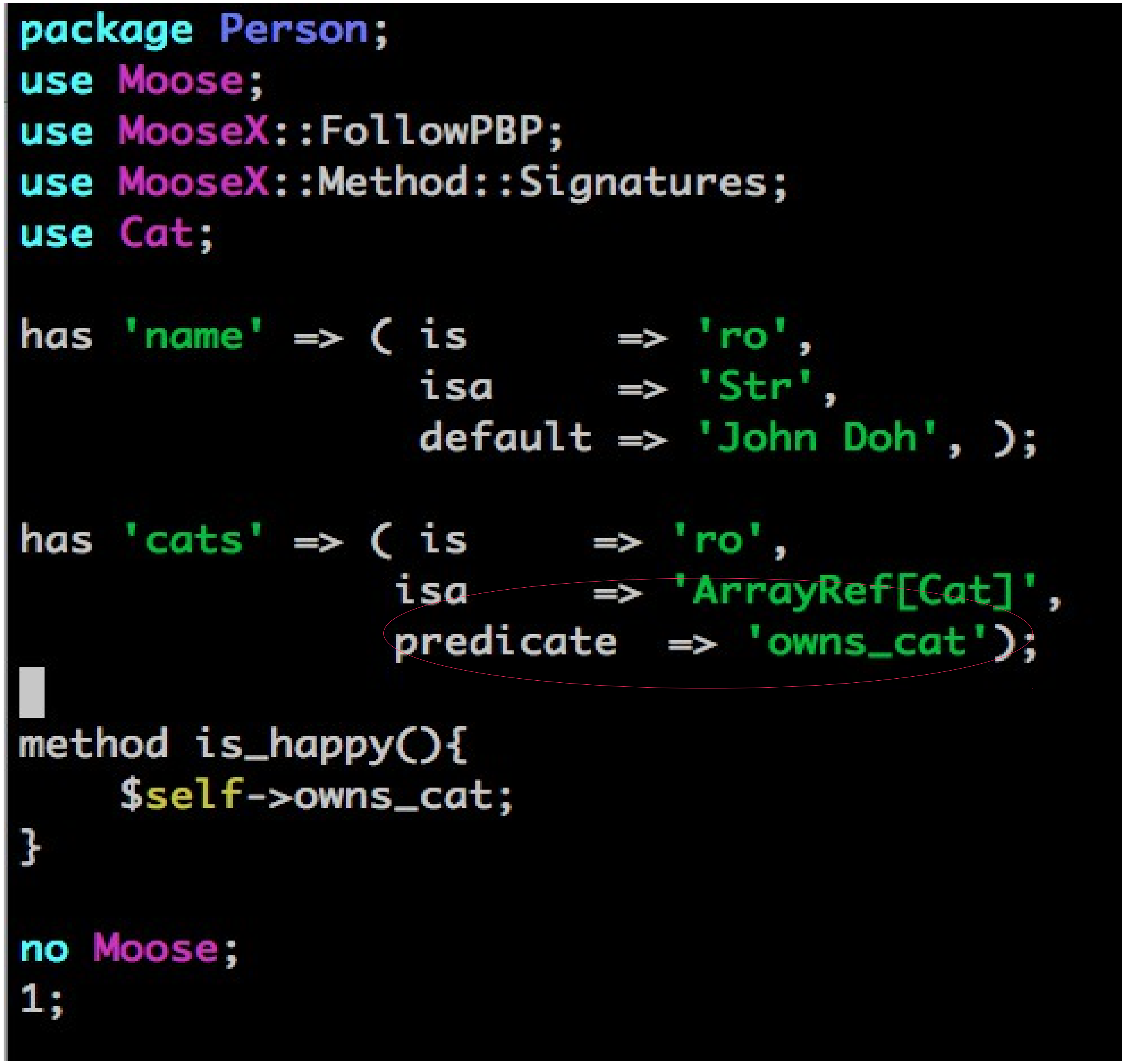




```
package Person;
use Moose;
use MooseX::FollowPBP;
use MooseX::Method::Signatures;
use Cat;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'John Doh', );

has 'cats' => ( is        => 'ro',
               isa       => 'ArrayRef[Cat]',
               predicate  => 'owns_cat');

method is_happy(){
    $self->owns_cat;
}

no Moose;
1;
```

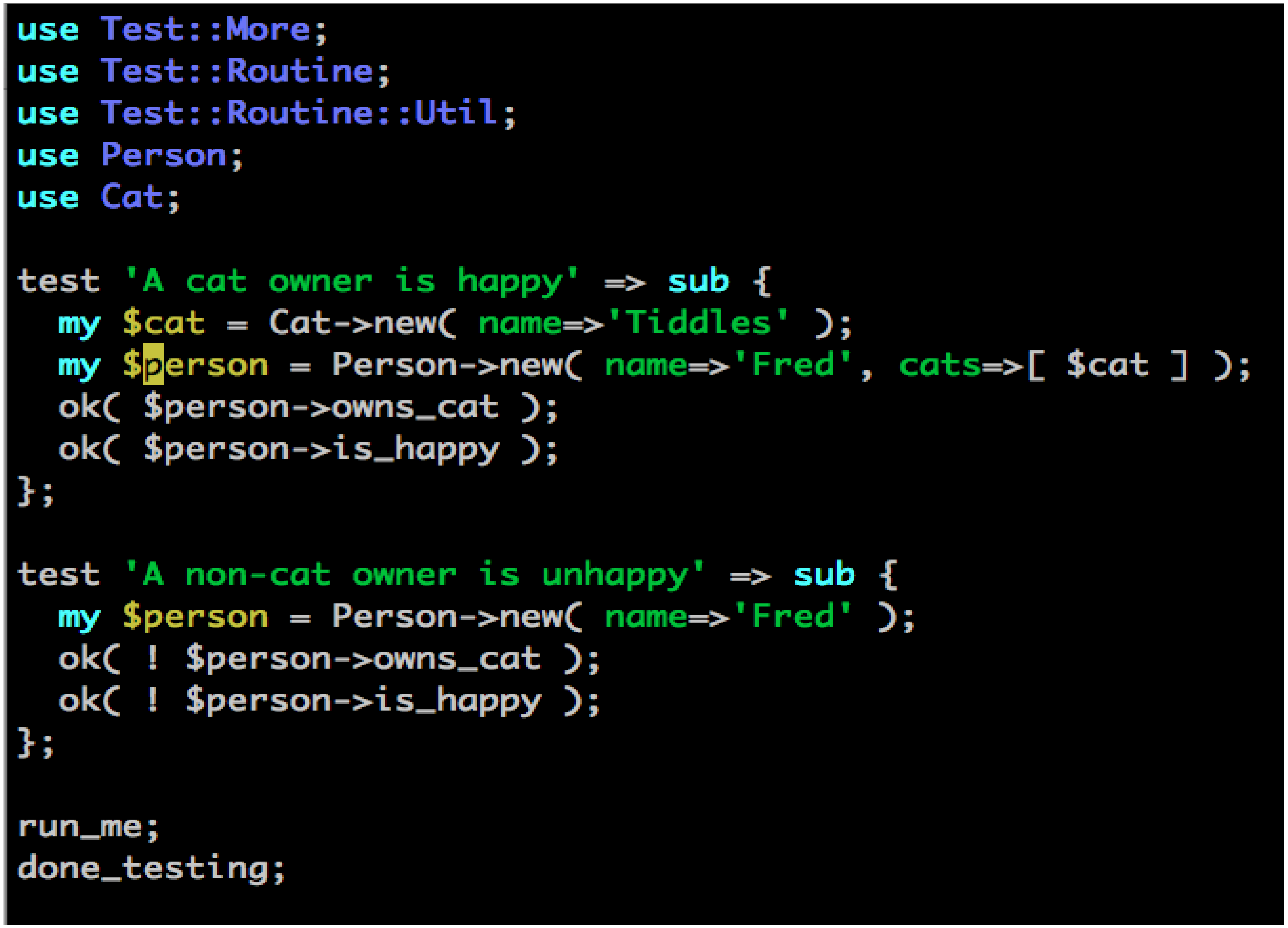
Person is a composition with a Cat





```
package Person;
use Moose;
use MooseX::FollowPBP;
use MooseX::Method::Signatures;
use Cat;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'John Doh', );

has 'cats' => ( is     => 'ro',
               isa     => 'ArrayRef[Cat]',
               predicate  => 'owns_cat');

method is_happy(){
    $self->owns_cat;
}

no Moose;
1;
```

You forgot to test!?
If the tests are broken fix them.



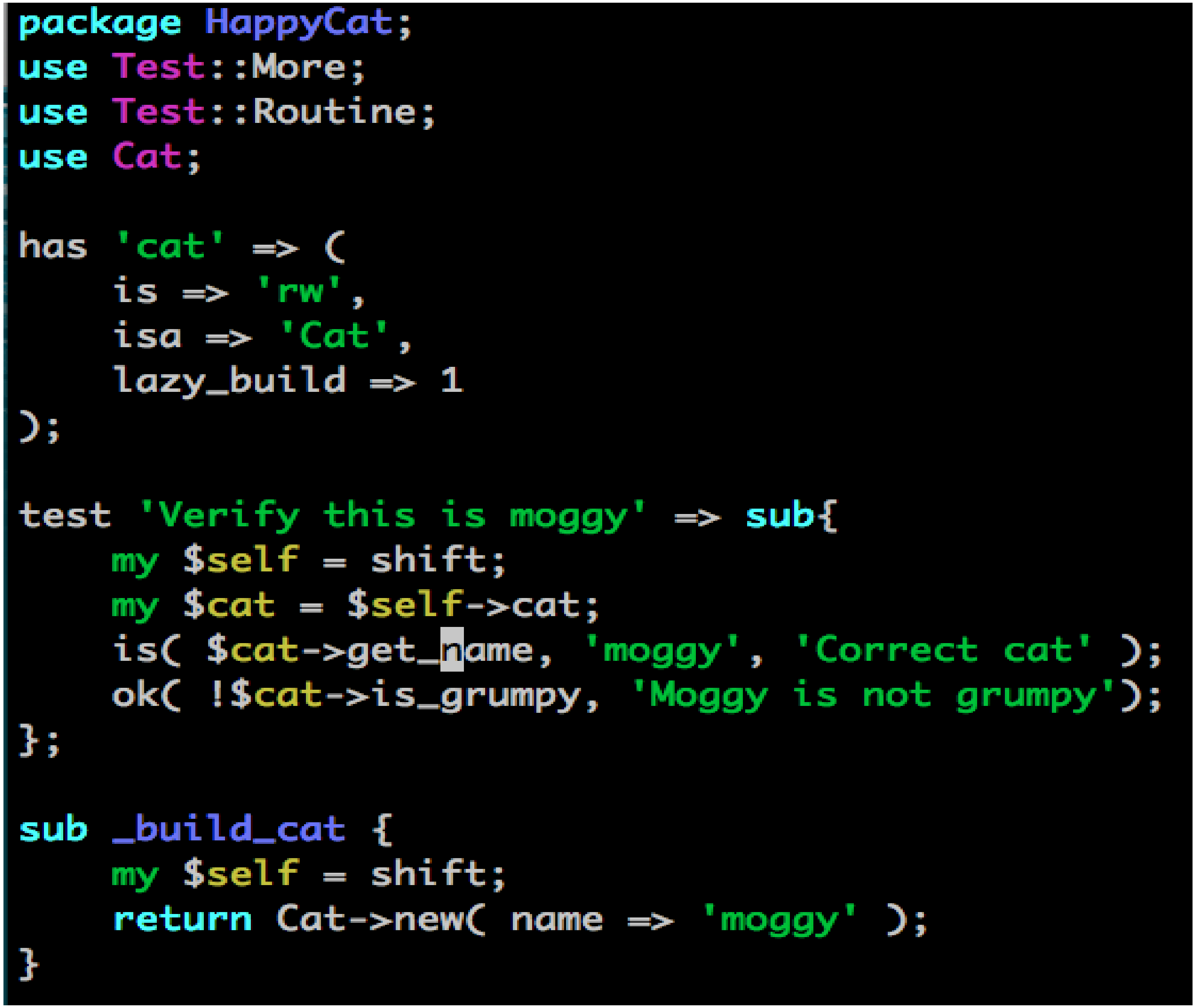
```perl
use Test::More;
use Test::Routine;
use Test::Routine::Util;
use Person;
use Cat;

test 'A cat owner is happy' => sub {
   my $cat = Cat->new( name=>'Tiddles' );
   my $person = Person->new( name=>'Fred', cats=>[ $cat ] );
   ok( $person->owns_cat );
   ok( $person->is_happy );
};

test 'A non-cat owner is unhappy' => sub {
   my $person = Person->new( name=>'Fred' );
   ok( ! $person->owns_cat );
   ok( ! $person->is_happy );
};

run_me;
done_testing;
```

AC

- Given Fred owns Moggy
- When Moggy is grumpy
- Then Fred is unhappy

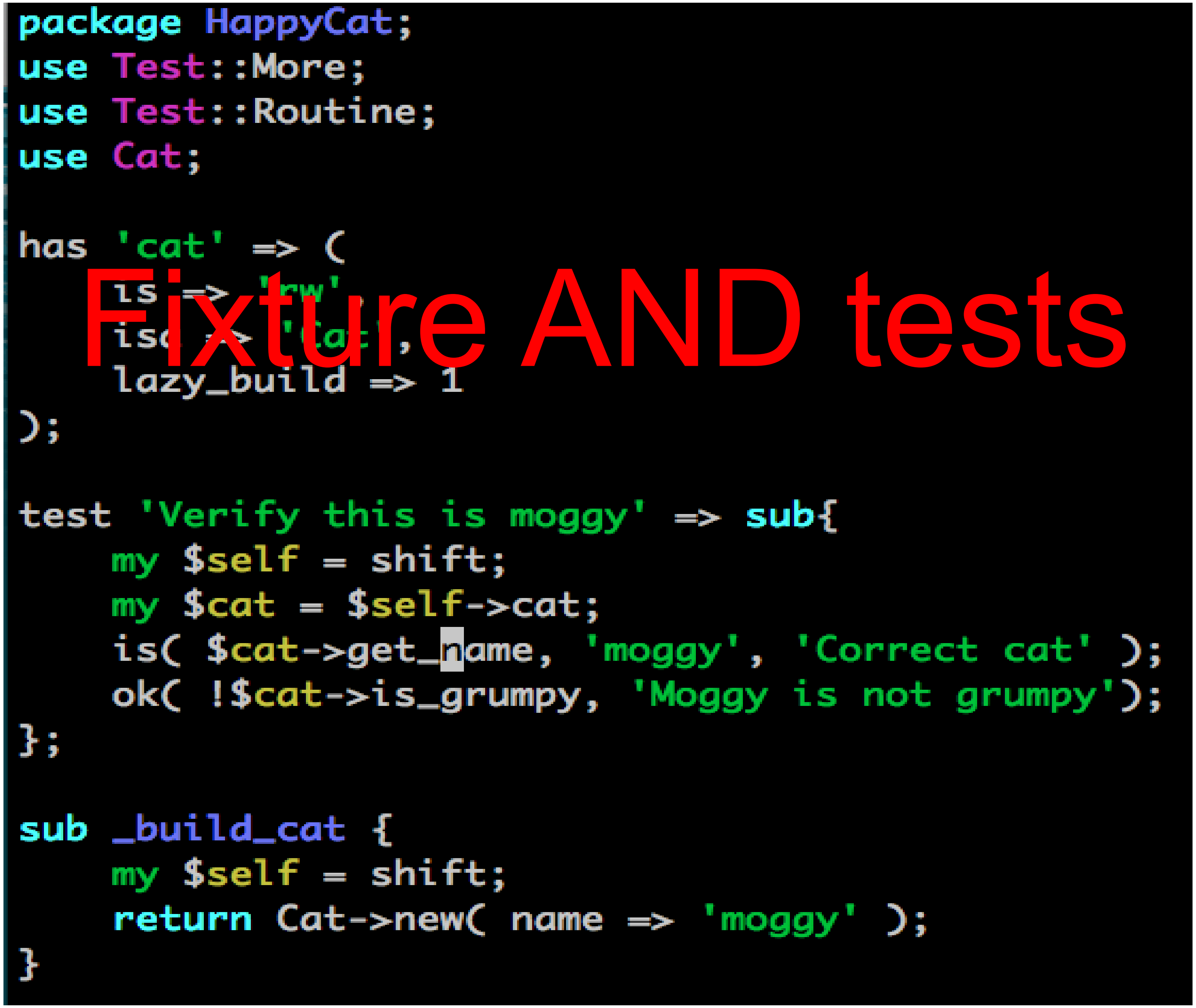
- Given Fred owns Moggy
- When Moggy is not grumpy
- Then Fred is happy

Create a happy cat fixture (Role)



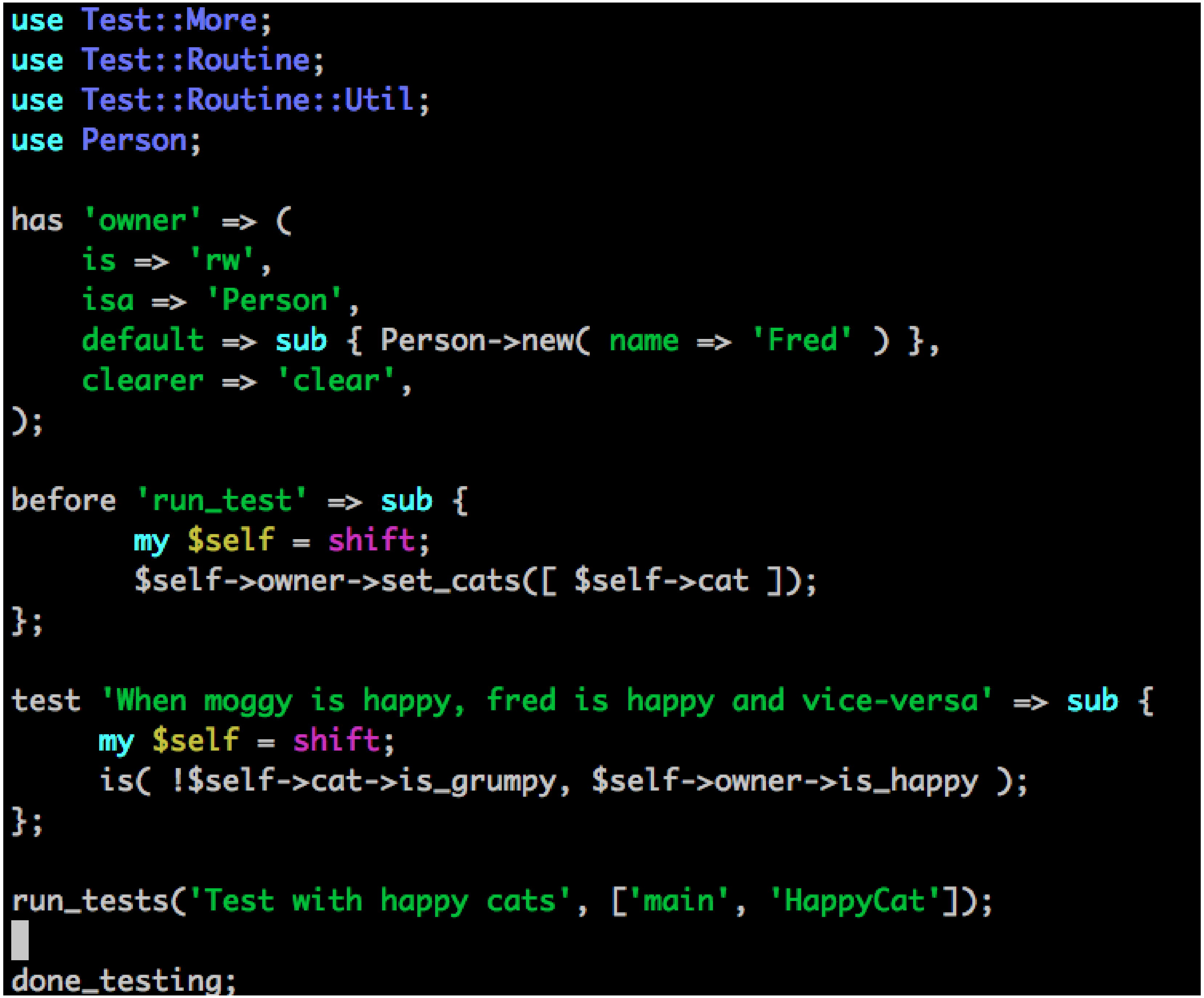
```perl
package HappyCat;
use Test::More;
use Test::Routine;
use Cat;

has 'cat' => (
    is => 'rw',
    isa => 'Cat',
    lazy_build => 1
);

test 'Verify this is moggy' => sub{
    my $self = shift;
    my $cat = $self->cat;
    is( $cat->get_name, 'moggy', 'Correct cat' );
    ok( !$cat->is_grumpy, 'Moggy is not grumpy');
};

sub _build_cat {
    my $self = shift;
    return Cat->new( name => 'moggy' );
}
```

Create a happy cat fixture (Role)



```
package HappyCat;
use Test::More;
use Test::Routine;
use Cat;

has 'cat' => (
    is => 'rw',
    isa => 'Cat',
    lazy_build => 1
);

test 'Verify this is moggy' => sub{
    my $self = shift;
    my $cat = $self->cat;
    is( $cat->get_name, 'moggy', 'Correct cat' );
    ok( !$cat->is_grumpy, 'Moggy is not grumpy');
};

sub _build_cat {
    my $self = shift;
    return Cat->new( name => 'moggy' );
}
```

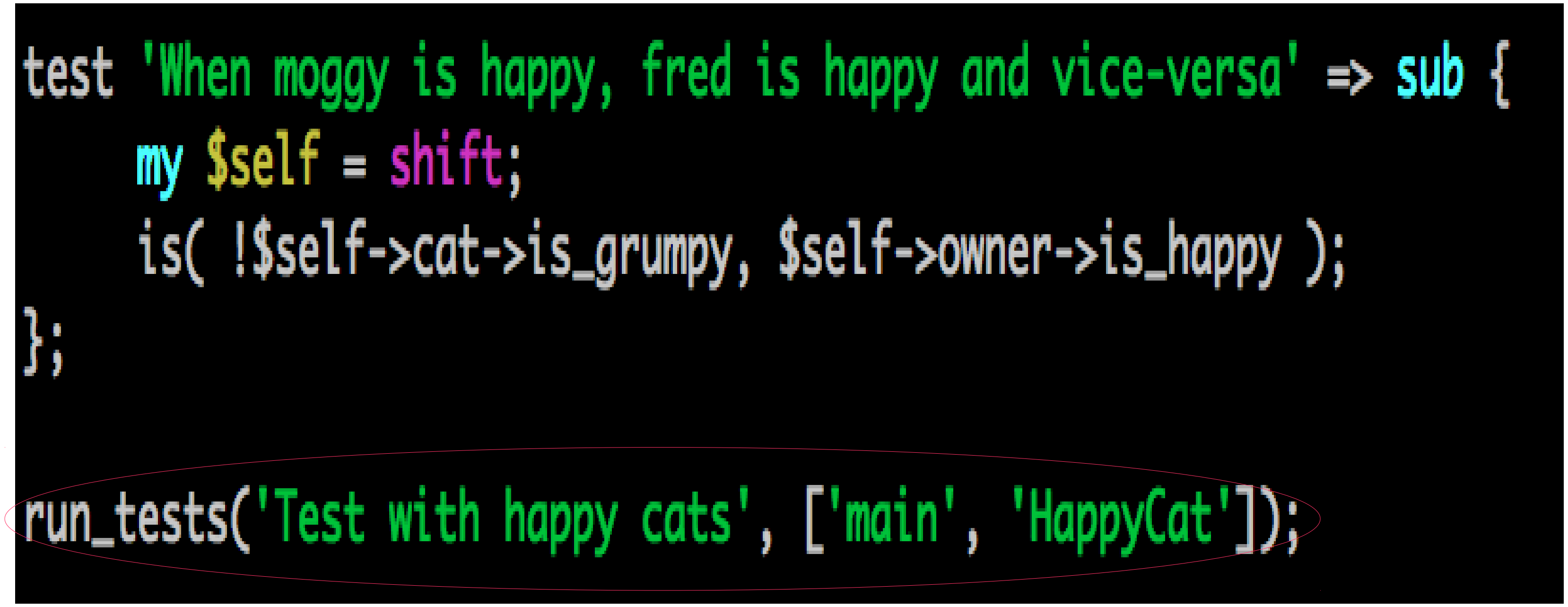
Fixture AND tests



```perl
use Test::More;
use Test::Routine;
use Test::Routine::Util;
use Person;

has 'owner' => (
    is => 'rw',
    isa => 'Person',
    default => sub { Person->new( name => 'Fred' ) },
    clearer => 'clear',
);

before 'run_test' => sub {
       my $self = shift;
       $self->owner->set_cats([ $self->cat ]);
};

test 'When moggy is happy, fred is happy and vice-versa' => sub {
     my $self = shift;
     is( !$self->cat->is_grumpy, $self->owner->is_happy );
};

run_tests('Test with happy cats', ['main', 'HappyCat']);

done_testing;
```

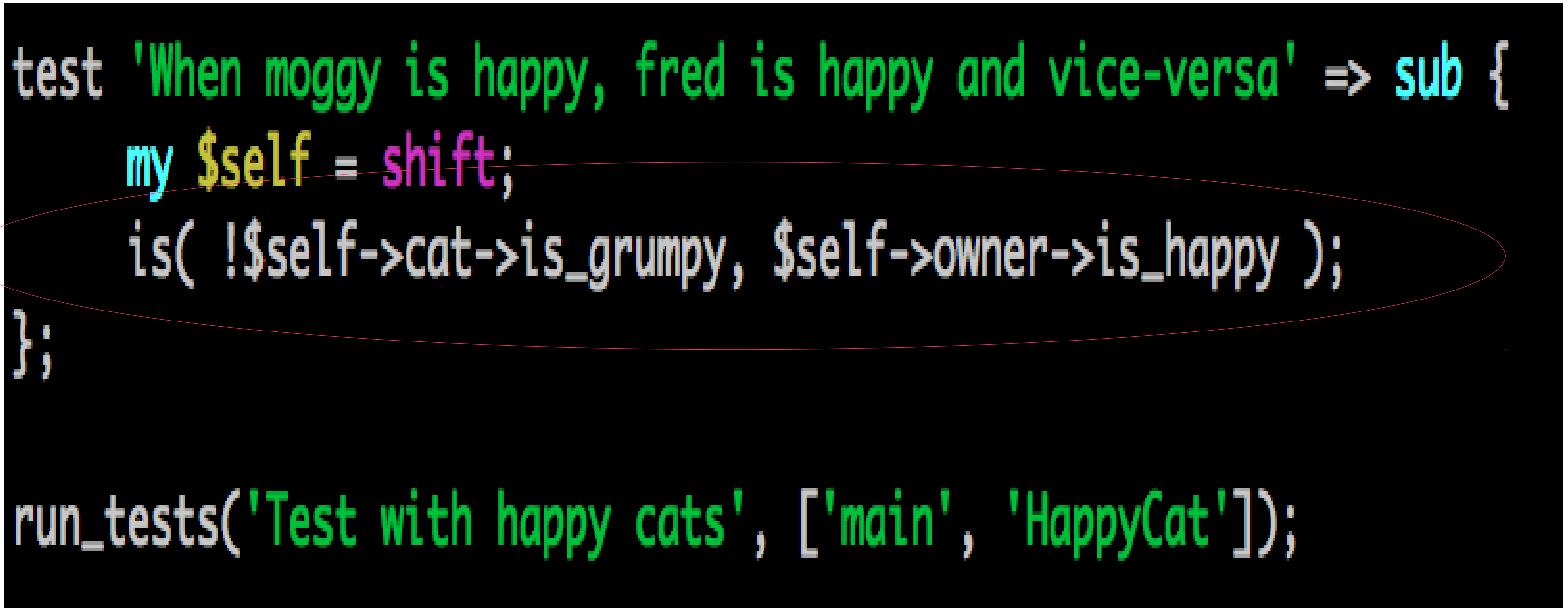
Implement tests for the User Story

$self->cat fixture consumed from HappyCat role



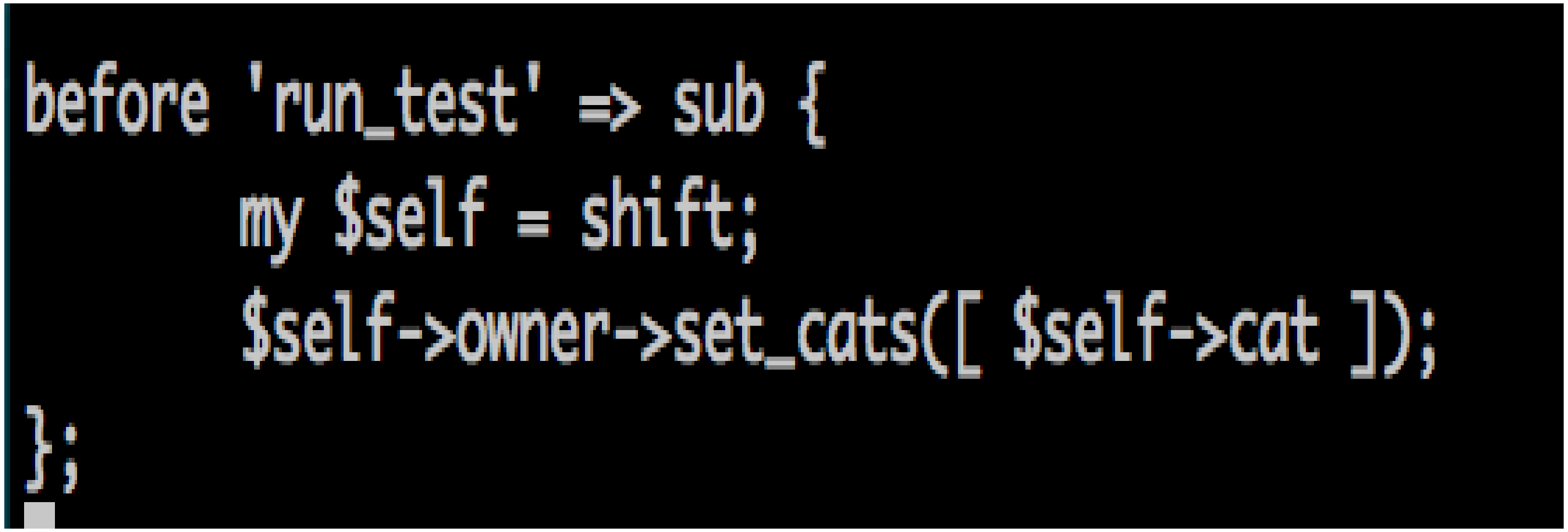
```
test 'When moggy is happy, fred is happy and vice-versa' => sub {
    my $self = shift;
    is( !$self->cat->is_grumpy, $self->owner->is_happy );
};

run_tests('Test with happy cats', ['main', 'HappyCat']);
```

$self->cat fixture is in scope



```
test 'When moggy is happy, fred is happy and vice-versa' => sub {
    my $self = shift;
    is( !$self->cat->is_grumpy, $self->owner->is_happy );
};

run_tests('Test with happy cats', ['main', 'HappyCat']);
```

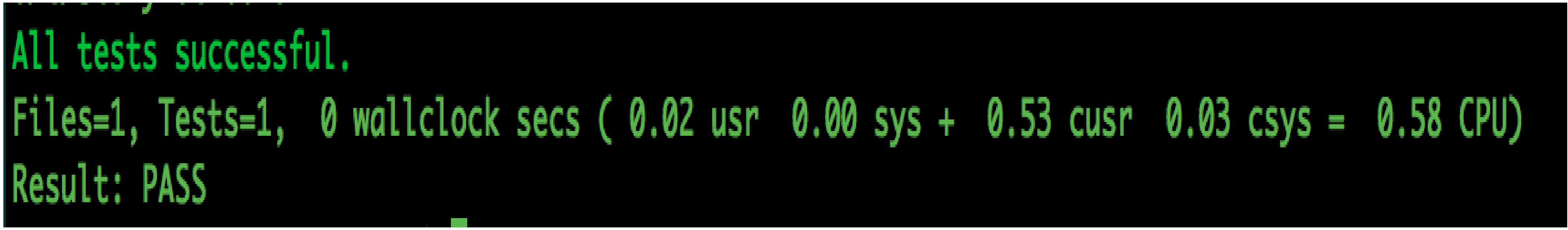
Setup / Tear Down with Modifiers



```
before 'run_test' => sub {
        my $self = shift;
        $self->owner->set_cats([ $self->cat ]);
};
```

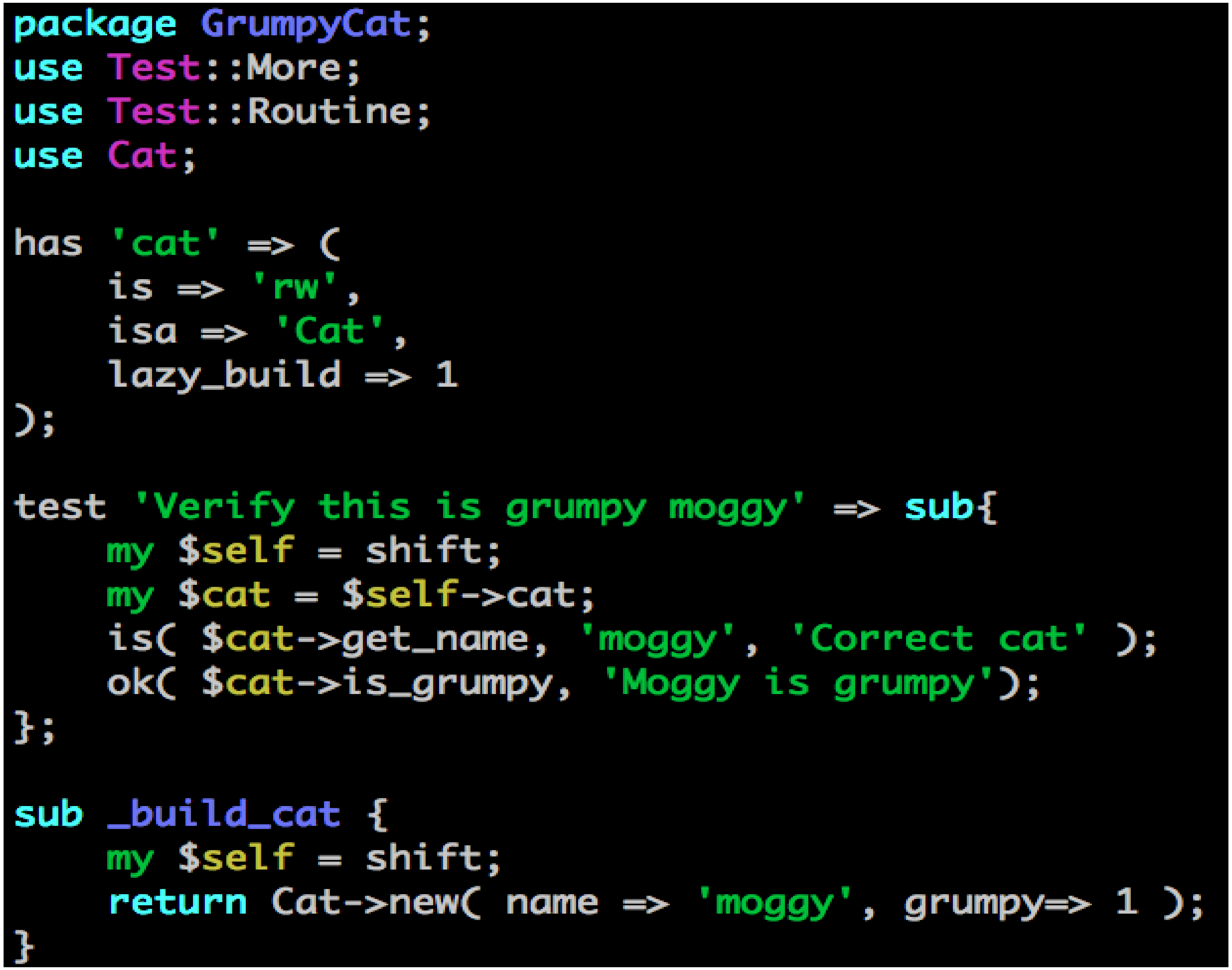
- before
- after
- around

\o/



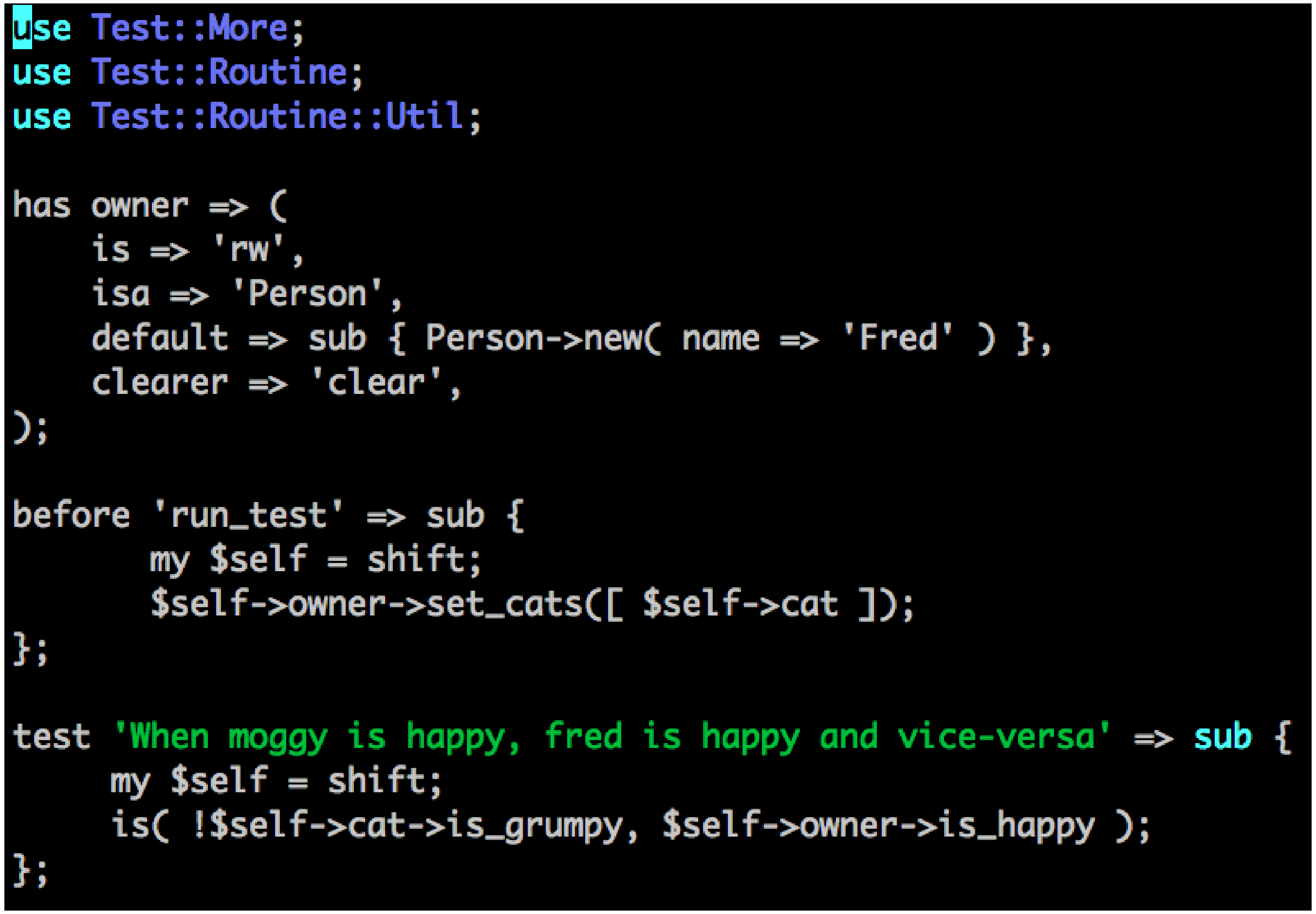
```
All tests successful.
Files=1, Tests=1,  0 wallclock secs ( 0.02 usr  0.00 sys +  0.53 cusr  0.03 csys =  0.58 CPU)
Result: PASS
```

Grumpy Cat Fixture



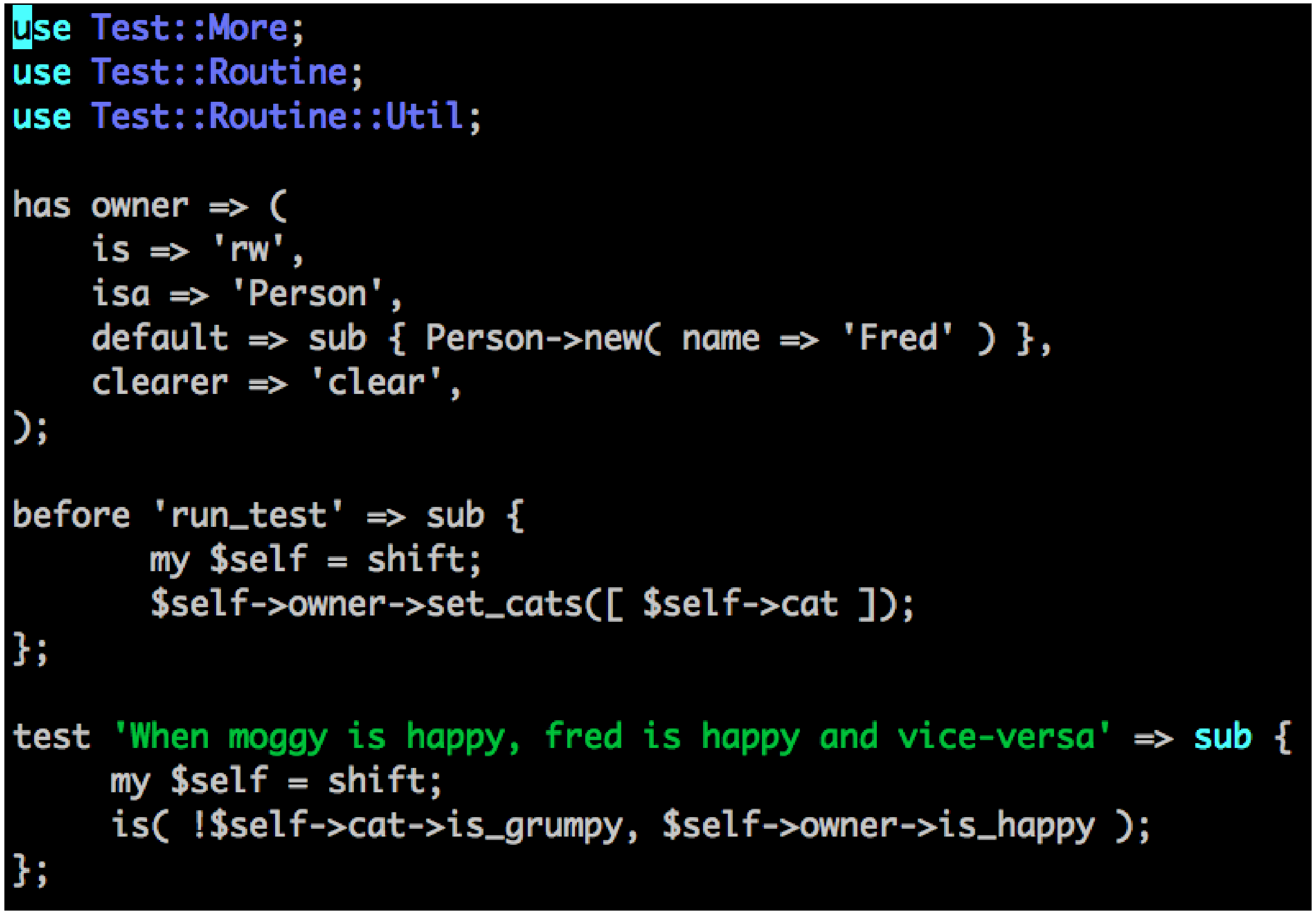
```perl
package GrumpyCat;
use Test::More;
use Test::Routine;
use Cat;

has 'cat' => (
    is => 'rw',
    isa => 'Cat',
    lazy_build => 1
);

test 'Verify this is grumpy moggy' => sub{
    my $self = shift;
    my $cat = $self->cat;
    is( $cat->get_name, 'moggy', 'Correct cat' );
    ok( $cat->is_grumpy, 'Moggy is grumpy');
};

sub _build_cat {
    my $self = shift;
    return Cat->new( name => 'moggy', grumpy=> 1 );
}
```

Story2.t

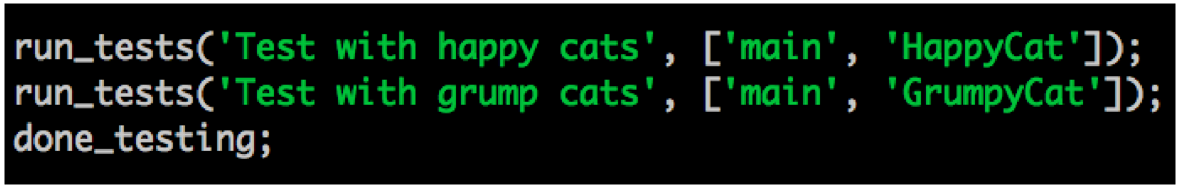


```perl
use Test::More;
use Test::Routine;
use Test::Routine::Util;

has owner => (
    is => 'rw',
    isa => 'Person',
    default => sub { Person->new( name => 'Fred' ) },
    clearer => 'clear',
);

before 'run_test' => sub {
       my $self = shift;
       $self->owner->set_cats([ $self->cat ]);
};

test 'When moggy is happy, fred is happy and vice-versa' => sub {
     my $self = shift;
     is( !$self->cat->is_grumpy, $self->owner->is_happy );
};
```

Story2.t

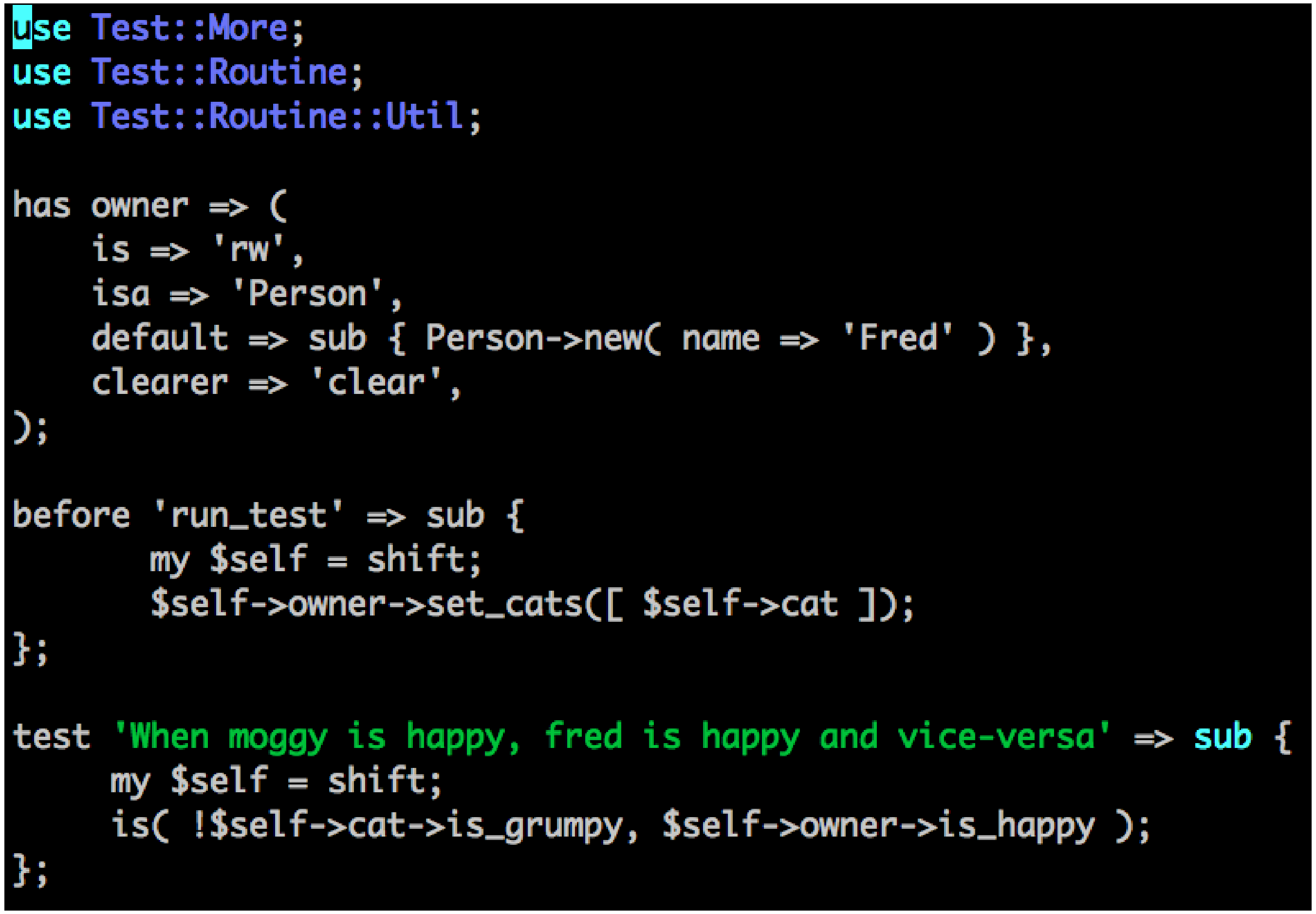


```
use Test::More;
use Test::Routine;
use Test::Routine::Util;

has owner => (
    is => 'rw',
    isa => 'Person',
    default => sub { Person->new( name => 'Fred' ) },
    clearer => 'clear',
);

before 'run_test' => sub {
       my $self = shift;
       $self->owner->set_cats([ $self->cat ]);
};

test 'When moggy is happy, fred is happy and vice-versa' => sub {
     my $self = shift;
     is( !$self->cat->is_grumpy, $self->owner->is_happy );
};
```

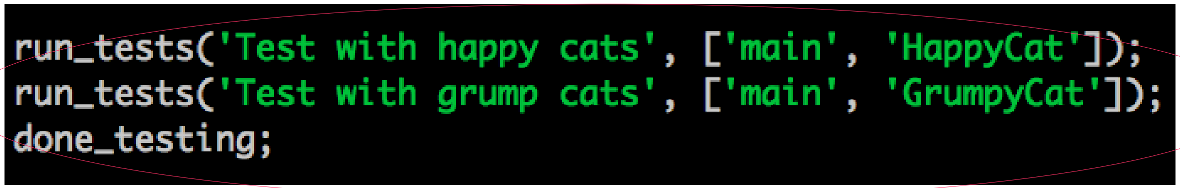


```
run_tests('Test with happy cats', ['main', 'HappyCat']);
run_tests('Test with grump cats', ['main', 'GrumpyCat']);
done_testing;
```

Story2.t

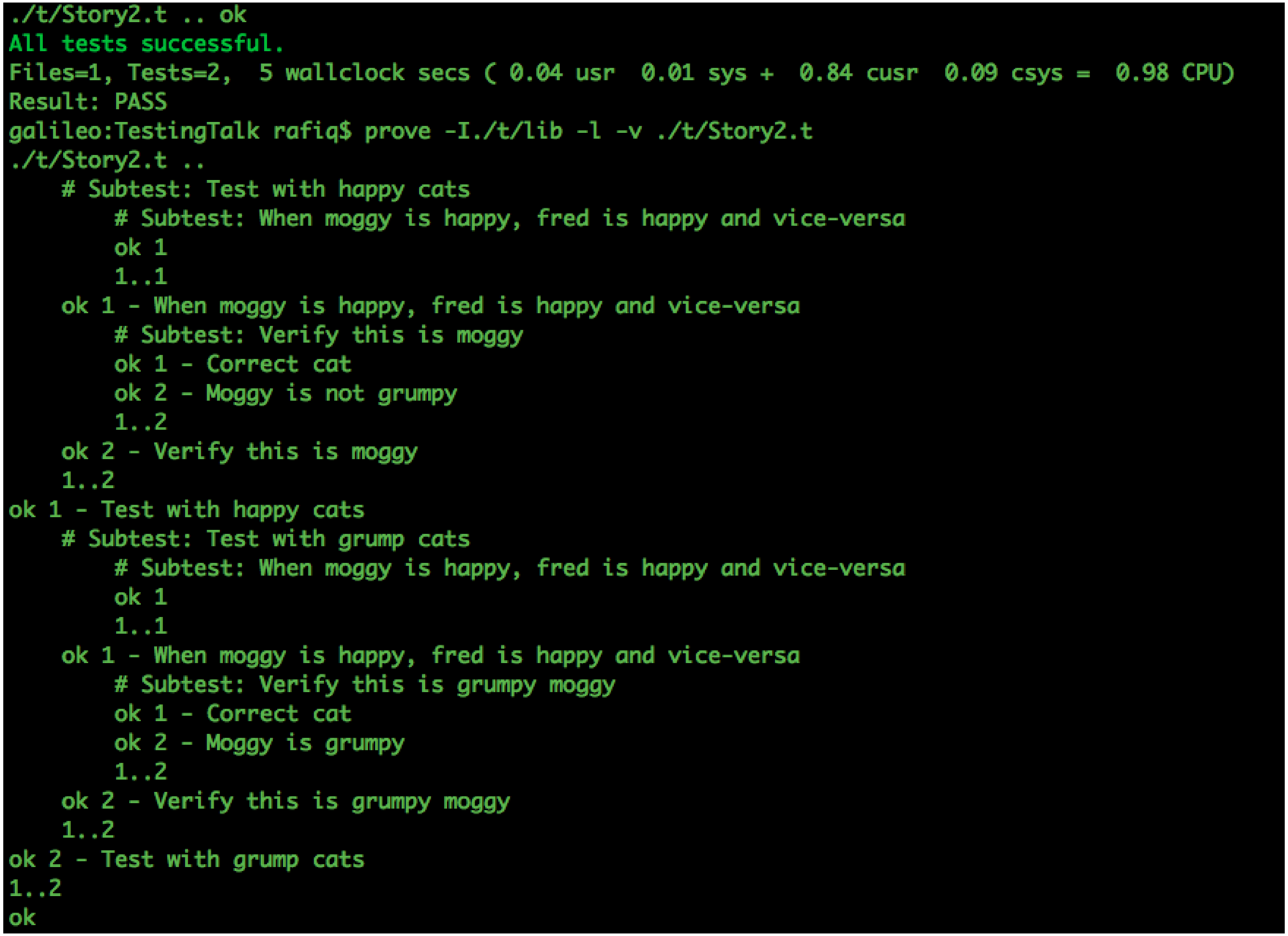


```
use Test::More;
use Test::Routine;
use Test::Routine::Util;

has owner => (
    is => 'rw',
    isa => 'Person',
    default => sub { Person->new( name => 'Fred' ) },
    clearer => 'clear',
);

before 'run_test' => sub {
       my $self = shift;
       $self->owner->set_cats([ $self->cat ]);
};

test 'When moggy is happy, fred is happy and vice-versa' => sub {
     my $self = shift;
     is( !$self->cat->is_grumpy, $self->owner->is_happy );
};
```



```
run_tests('Test with happy cats', ['main', 'HappyCat']);
run_tests('Test with grump cats', ['main', 'GrumpyCat']);
done_testing;
```

Tests Pass



```
./t/Story2.t .. ok
All tests successful.
Files=1, Tests=2,  5 wallclock secs ( 0.04 usr  0.01 sys +  0.84 cusr  0.09 csys =  0.98 CPU)
Result: PASS
galileo:TestingTalk rafiq$ prove -I./t/lib -l -v ./t/Story2.t
./t/Story2.t ..
    # Subtest: Test with happy cats
        # Subtest: When moggy is happy, fred is happy and vice-versa
        ok 1
        1..1
    ok 1 - When moggy is happy, fred is happy and vice-versa
        # Subtest: Verify this is moggy
        ok 1 - Correct cat
        ok 2 - Moggy is not grumpy
        1..2
    ok 2 - Verify this is moggy
    1..2
ok 1 - Test with happy cats
    # Subtest: Test with grump cats
        # Subtest: When moggy is happy, fred is happy and vice-versa
        ok 1
        1..1
    ok 1 - When moggy is happy, fred is happy and vice-versa
        # Subtest: Verify this is grumpy moggy
        ok 1 - Correct cat
        ok 2 - Moggy is grumpy
        1..2
    ok 2 - Verify this is grumpy moggy
    1..2
ok 2 - Test with grump cats
1..2
ok
```

Story 3

As a sadist data modeller
I want to make allergic people die by giving them Cats
So that I can have a laugh when they stroke them

AC

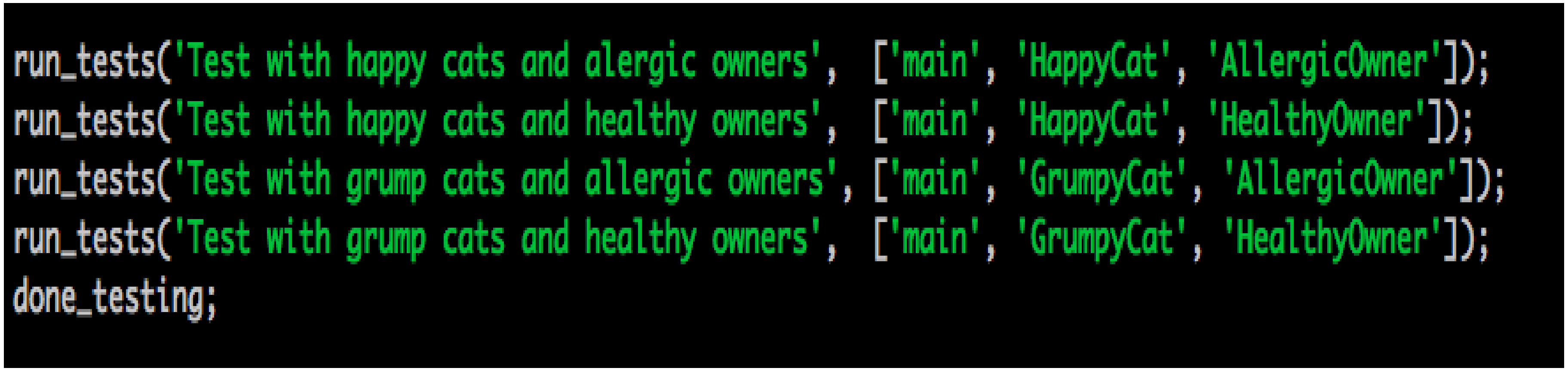
- GIVEN Fred has cat allergies
- AND Fred owns Moggy
- WHEN Fred strokes moggy
- THEN Fred croaks

- GIVEN Peter does not have cat allergies
- AND Peter owns Moggy
- AND Moggy is not grumpy
- WHEN Peter strokes Moggy
- THEN Fred lives

AC

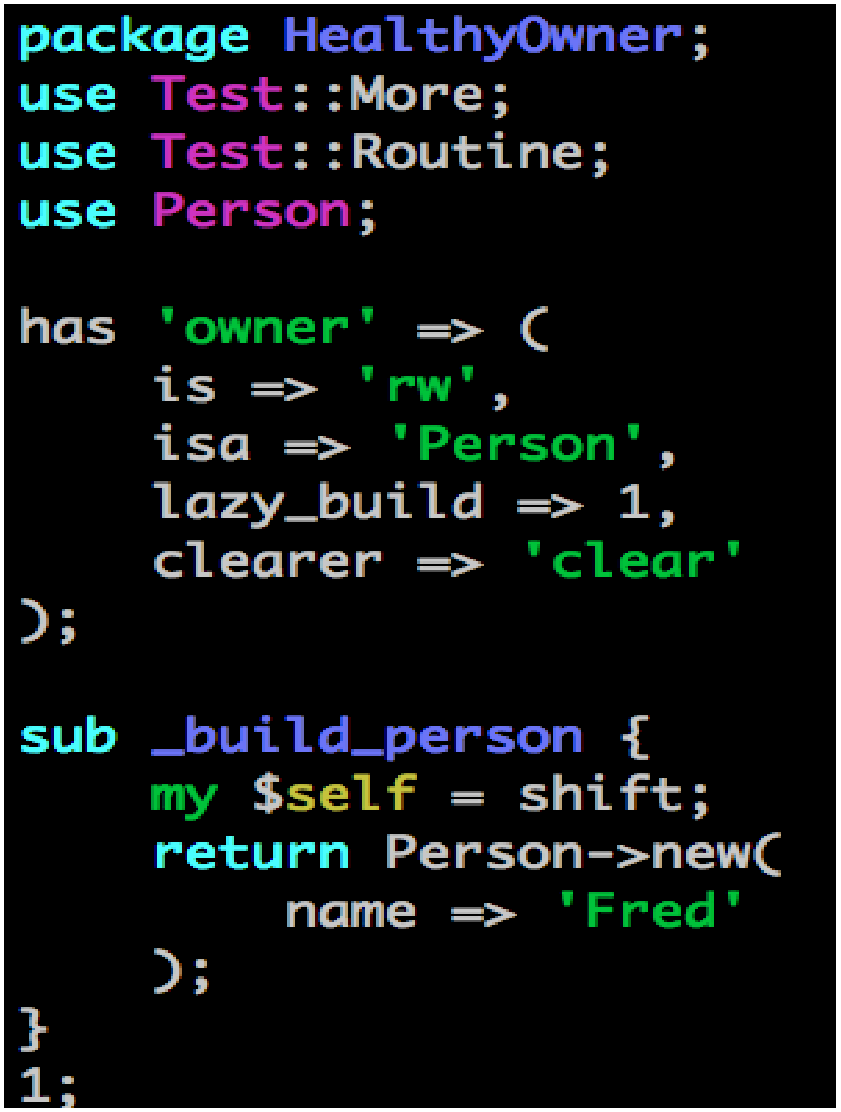
- GIVEN Peter does not have cat allergies
- AND Peter owns Moggy
- AND Moggy IS grumpy
- WHEN Peter strokes Moggy
- THEN Fred lives

Let's define our Owner Fixtures

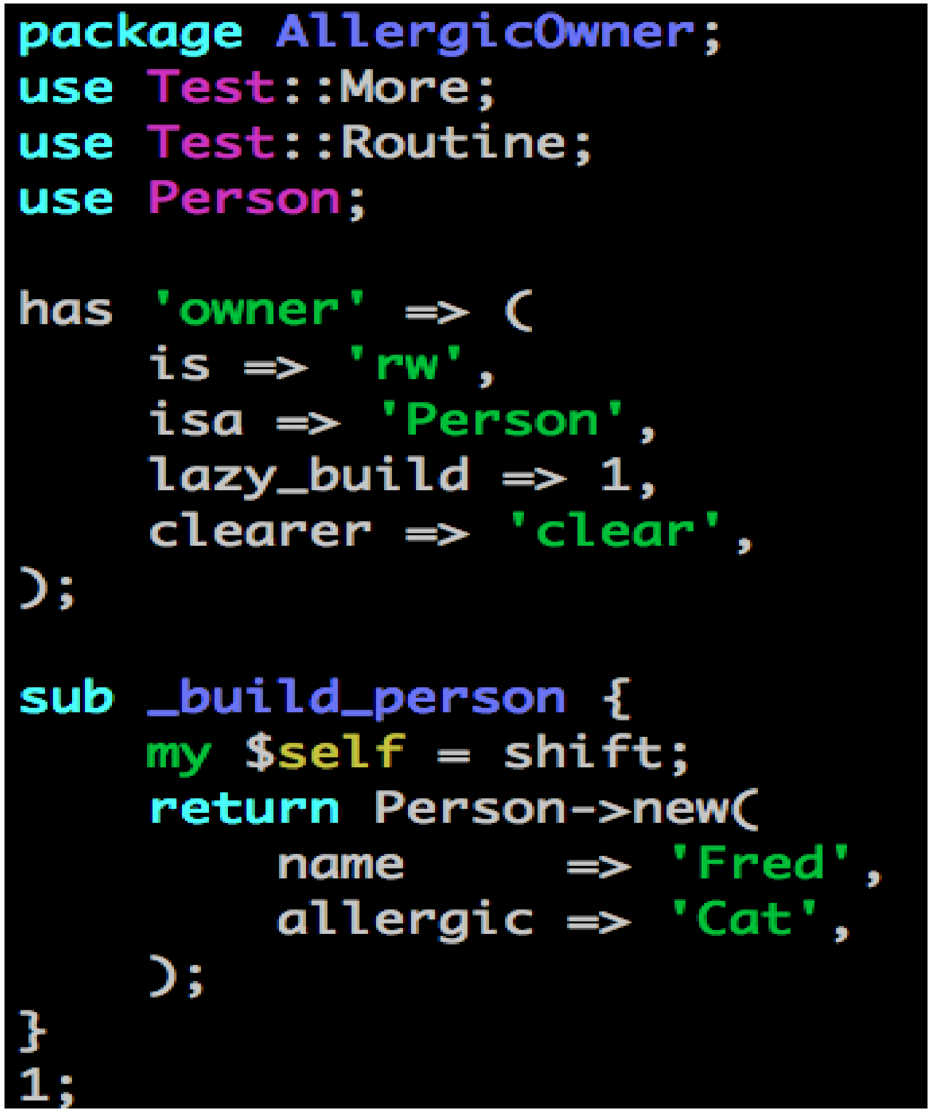


```
run_tests('Test with happy cats and alergic owners',  ['main', 'HappyCat', 'AllergicOwner']);
run_tests('Test with happy cats and healthy owners',  ['main', 'HappyCat', 'HealthyOwner']);
run_tests('Test with grump cats and allergic owners', ['main', 'GrumpyCat', 'AllergicOwner']);
run_tests('Test with grump cats and healthy owners',  ['main', 'GrumpyCat', 'HealthyOwner']);
done_testing;
```

Some owner fixtures

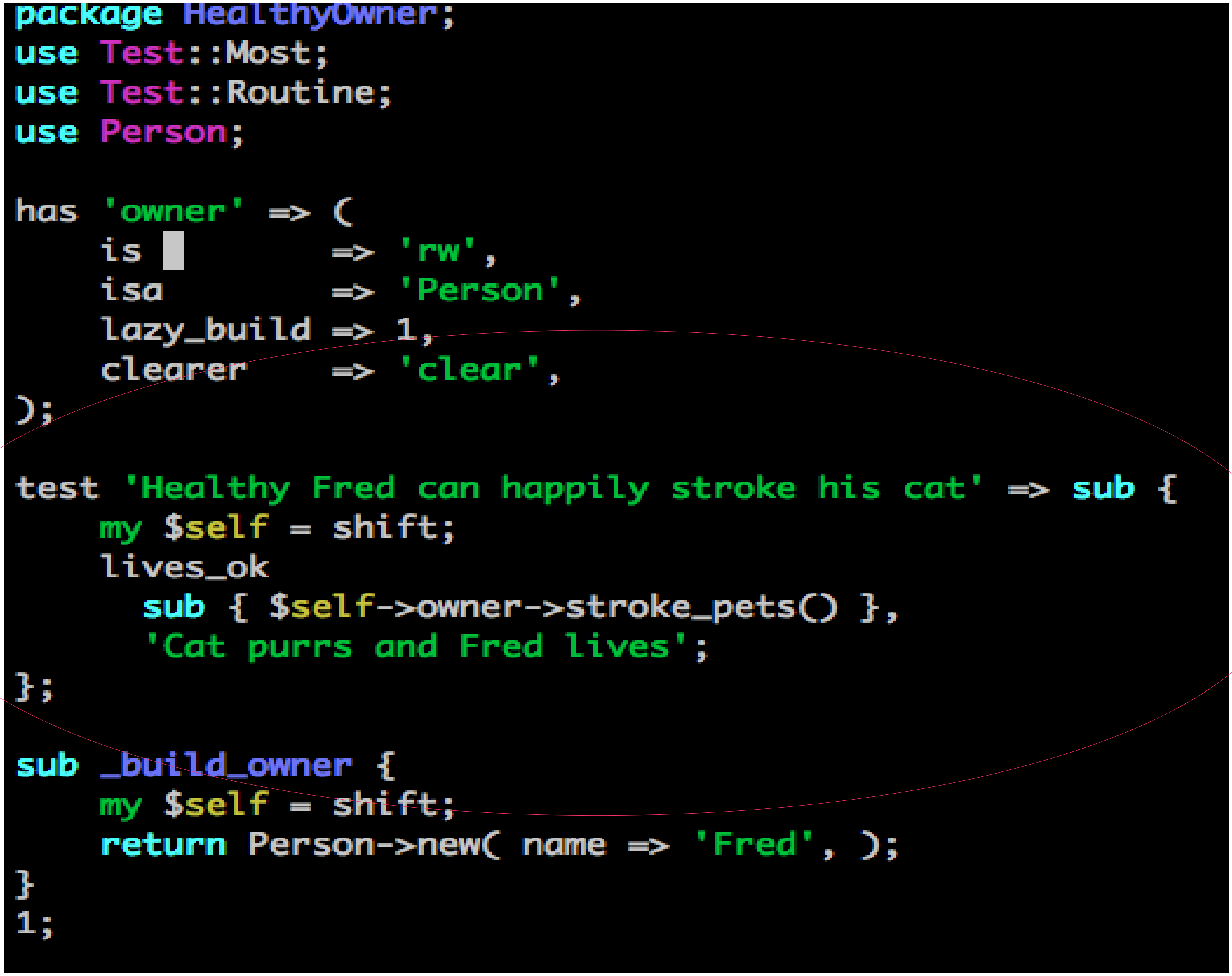


```perl
package HealthyOwner;
use Test::More;
use Test::Routine;
use Person;

has 'owner' => (
    is => 'rw',
    isa => 'Person',
    lazy_build => 1,
    clearer => 'clear'
);

sub _build_person {
    my $self = shift;
    return Person->new(
        name => 'Fred'
    );
}
1;
```



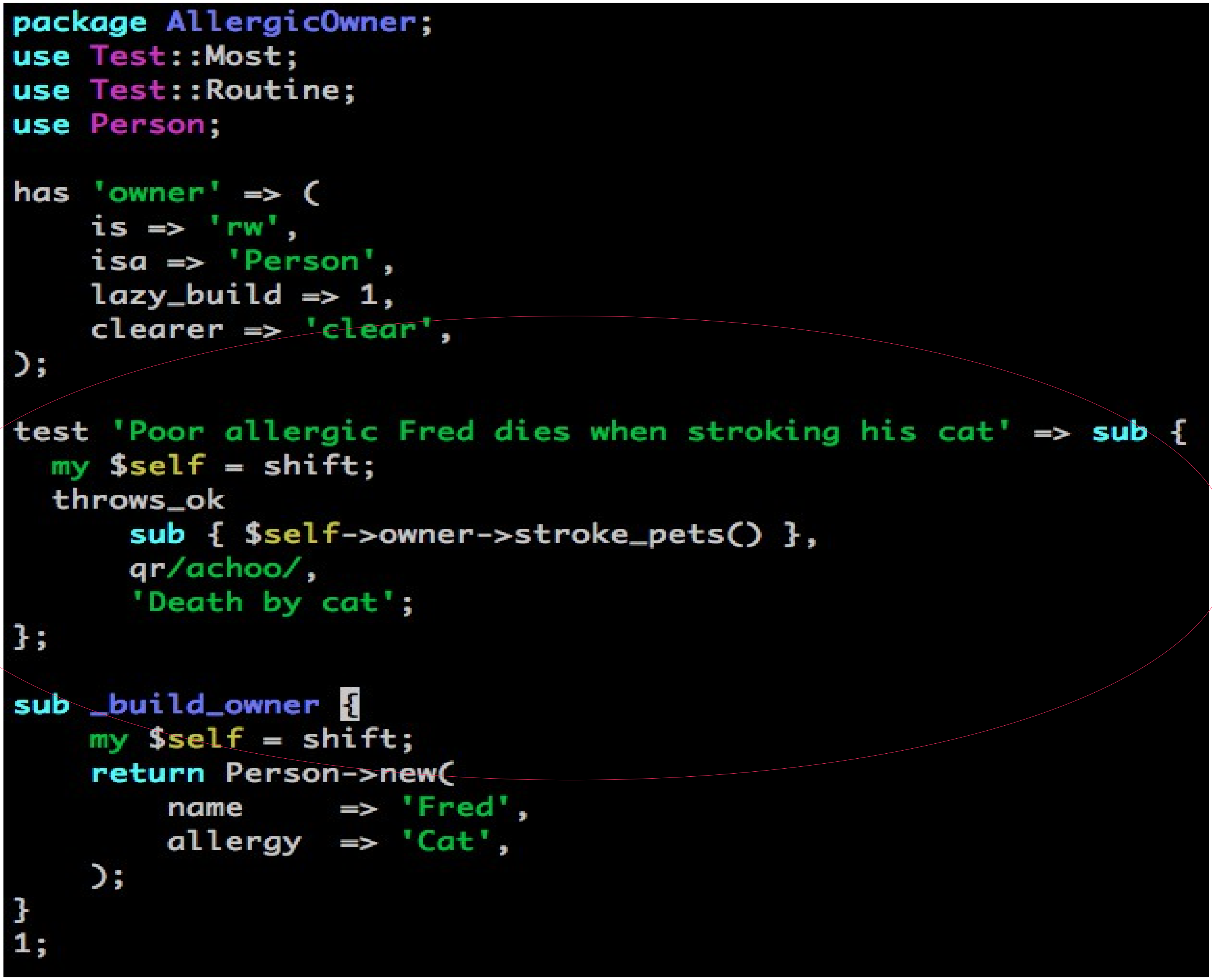
```perl
package AllergicOwner;
use Test::More;
use Test::Routine;
use Person;

has 'owner' => (
    is => 'rw',
    isa => 'Person',
    lazy_build => 1,
    clearer => 'clear',
);

sub _build_person {
    my $self = shift;
    return Person->new(
        name     => 'Fred',
        allergic => 'Cat',
    );
}
1;
```

Now add the AC's

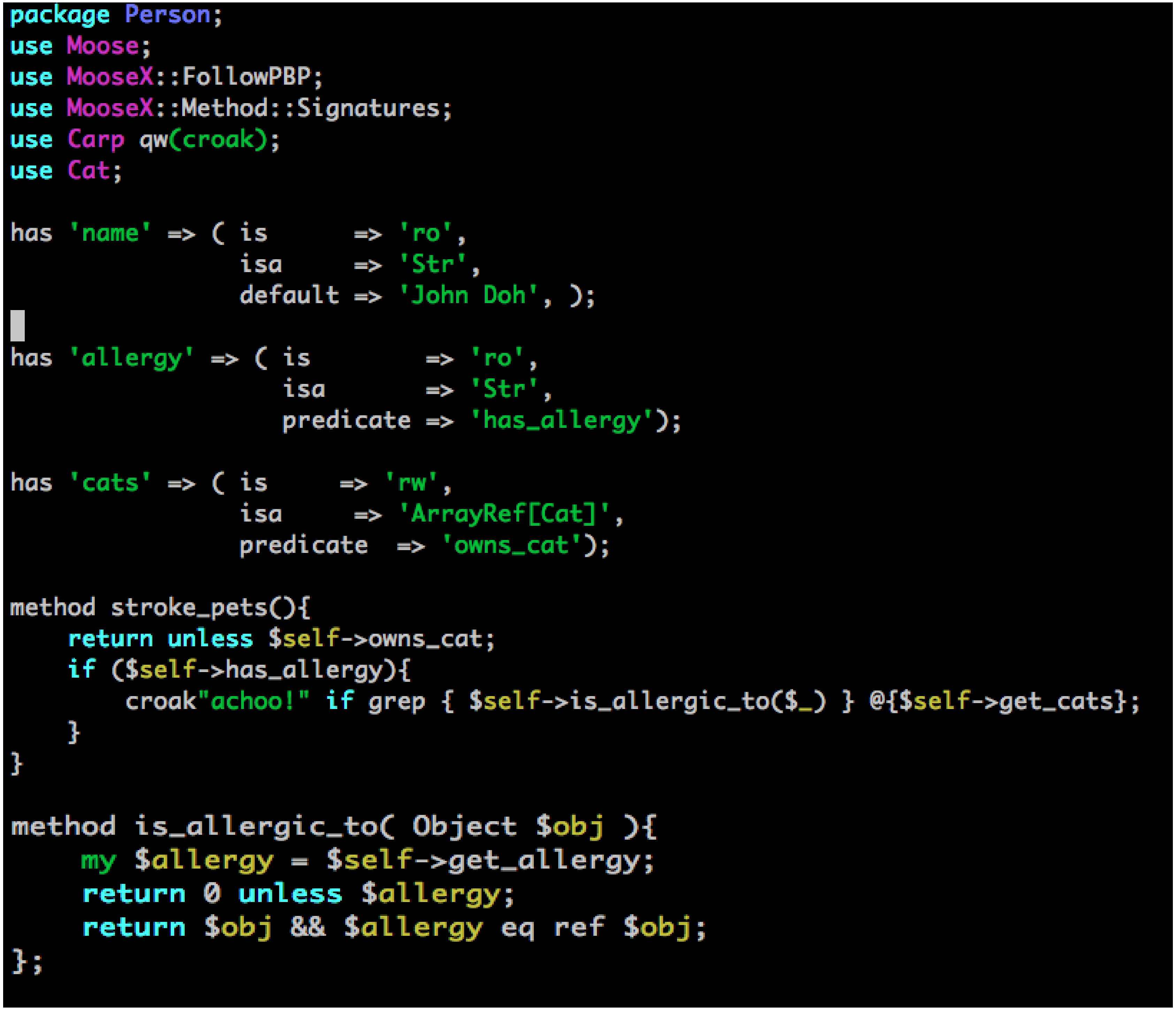


```perl
package HealthyOwner;
use Test::Most;
use Test::Routine;
use Person;

has 'owner' => (
    is         => 'rw',
    isa        => 'Person',
    lazy_build => 1,
    clearer    => 'clear',
);

test 'Healthy Fred can happily stroke his cat' => sub {
    my $self = shift;
    lives_ok
      sub { $self->owner->stroke_pets() },
      'Cat purrs and Fred lives';
};

sub _build_owner {
    my $self = shift;
    return Person->new( name => 'Fred', );
}
1;
```

Now add the AC's



```
package AllergicOwner;
use Test::Most;
use Test::Routine;
use Person;

has 'owner' => (
    is => 'rw',
    isa => 'Person',
    lazy_build => 1,
    clearer => 'clear',
);

test 'Poor allergic Fred dies when stroking his cat' => sub {
  my $self = shift;
  throws_ok
      sub { $self->owner->stroke_pets() },
      qr/achoo/,
      'Death by cat';
};

sub _build_owner {
    my $self = shift;
    return Person->new(
        name     => 'Fred',
        allergy  => 'Cat',
    );
}
1;
```



```
package Person;
use Moose;
use MooseX::FollowPBP;
use MooseX::Method::Signatures;
use Carp qw(croak);
use Cat;

has 'name' => ( is      => 'ro',
                isa     => 'Str',
                default => 'John Doh', );

has 'allergy' => ( is        => 'ro',
                   isa       => 'Str',
                   predicate => 'has_allergy');

has 'cats' => ( is      => 'rw',
                isa     => 'ArrayRef[Cat]',
                predicate  => 'owns_cat');

method stroke_pets(){
    return unless $self->owns_cat;
    if ($self->has_allergy){
        croak"achoo!" if grep { $self->is_allergic_to($_) } @{$self->get_cats};
    }
}

method is_allergic_to( Object $obj ){
     my $allergy = $self->get_allergy;
     return 0 unless $allergy;
     return $obj && $allergy eq ref $obj;
};
```

And this is what it looks like..

Questions?

https://github.com/fiqwrk/wellypm-moose-talk/

no Moose;