OH HAI

Hi!

Hola!

Hej hej!

Ahoj!

Salut!

Olá!

Ciao!

Привет!
Me

- Programming Perl since 2001ish
Me

- Programming Perl since 2001ish
- Originally from UK, now live in Slovakia
Me

- Programming Perl since 2001ish
- Originally from UK, now live in Slovakia
- Rakudo Perl 6 and Parrot hacker
Me

- Programming Perl since 2001ish
- Originally from UK, now live in Slovakia
- Rakudo Perl 6 and Parrot hacker
- Travel all over the place
Me

- Programming Perl since 2001ish
- Originally from UK, now live in Slovakia
- Rakudo Perl 6 and Parrot hacker
- Travel all over the place
- Usually hack to heavy metal
Me

- Programming Perl since 2001ish
- Originally from UK, now live in Slovakia
- Rakudo Perl 6 and Parrot hacker
- Travel all over the place
- Usually hack to heavy metal
- Good beer makes me happy
Me

- Programming Perl since 2001ish
- Originally from UK, now live in Slovakia
- Rakudo Perl 6 and Parrot hacker
- Travel all over the place
- Usually hack to heavy metal
- Good beer makes me happy
- Acrostics are awesome
Me

- Programming Perl since 2001ish
- Originally from UK, now live in Slovakia
- Rakudo Perl 6 and Parrot hacker
- Travel all over the place
- Usually hack to heavy metal
- Good beer makes me happy
- Acrostics are awesome
- LOL
Roles
Roles

Right at the heart of Perl 6
Perl 6 Roles In Depth

Roles

- Composition and mix-ins
- Typed data structures
- Sigils
- Traits
Perl 6 Roles In Depth

Roles

Composition and mix-ins

Typed data structures

Sigils

Traits
Roles

Composition and mix-ins

Sigils

Traits

Typed data structures
Perl 6 Roles In Depth

Roles

- Composition and mix-ins
- Typed data structures
- Sigils
- Traits
Perl 6 Roles In Depth

Roles

Composition and mix-ins

Typed data structures

Sigils

Traits
Perl 6 Roles In Depth

Roles

Composition and mix-ins

Sigils

Traits

Typed data structures
So what is a role anyway?

- A collection of zero or more…
  - Methods
  - Attributes
- Unlike a class, cannot be instantiated (if you try, a class is generated for you)
- Classes in Perl 6 are mutable (with the right pragma in force, can be monkey-typed), whereas roles are immutable
What does a role look like?

- Introduced with the role keyword
- Methods and attributes declared just as they would be in a Perl 6 class
Perl 6 Roles In Depth

What does a role look like?

- Introduced with the role keyword
- Methods and attributes declared just as they would be in a Perl 6 class

```perl
role DebugLog { 
    ...
}
```
Perl 6 Roles In Depth

What does a role look like?

- Introduced with the role keyword
- Methods and attributes declared just as they would be in a Perl 6 class

```perl
role DebugLog {
    has @.log_lines;
    ...
}
```
What does a role look like?

- Introduced with the role keyword
- Methods and attributes declared just as they would be in a Perl 6 class

```perl
role DebugLog {
    has @.log_lines;
    has $.log_size is rw = 100;
    ...
}
```
What does a role look like?

- Introduced with the `role` keyword
- Methods and attributes declared just as they would be in a Perl 6 class

```perl
role DebugLog {
    has @.log_lines;
    has $.log_size is rw = 100;
    method log_message($message) {
        ...
    }
}
```
Perl 6 Roles In Depth

What does a role look like?

- Introduced with the role keyword
- Methods and attributes declared just as they would be in a Perl 6 class

```perl
role DebugLog {  
    has @$_log_lines;
    has $.log_size is rw = 100;
    method log_message($message) {
        @$_log_lines.shift if
        @$_log_lines.elems >= $!log_size;
        @$_log_lines.push($message);
    }
}
```
Role Composition

- A role is composed into a class using the does trait

```perl
class WebCrawler does DebugLog {
    ...
}
```

- This adds the methods and attributes to the class

- End result is as if they had been written inside the class in the first place
Mix-ins

- Allow the functionality of a role to be added on a per-object basis (whereas compile time composition works on a per-class basis)
- Does not affect any other instances of the class
- Methods from the role always override any existing methods the object has
Mix-ins Example

- Suppose we want to trace what happens to a certain object
- Mix in the DebugLog role

```perl
$account does DebugLog;

Later, we can output the lines that were logged

$account.log_lines>>.say;
```
Mix-ins Example

- Now we just need to add calls to the log_message method
- We can do this with the .? operator, which calls the method if it exists

```perl
class Account {
    method change_password($new) {
        self.?log_message(
            "changing password to $new");
        ...
    }
}
```
Perl 6 Roles In Depth

Roles

Composition and mix-ins

Typed data structures

Sigils

Traits
Sigil = Interface Contract

- In Perl 6, a sigil implies an interface contract
- This interface contract is defined by a role
- You can only put things into a variable with that sigil if it does the required role
- Exception: variables with the $ sigil can store anything (if not type-constrained)
Perl 6 Roles In Depth

@ = Positional

- The @ sigil implies the Positional role
- Promises that there will be a method postcircumfix:<[ ]> that you can call
- This is that gets called when you do an index positionally into something

```
say @fact[1];
say @fact.postcircumfix:<[ ]>(1);
```

- Note: optimizer (when we have one) may emit something more lightweight
% = Associative

- The % sigil implies the Associative role
- Promises that there will be a method postcircumfix:{ } that you can call
- This is that gets called when you do an index associatively into something

```
say %price<Cheese>;
say %price.postcircumfix:{ }('Cheese');
```
& = Callable

- The & sigil implies the Callable role
- Promises that the thing can be invoked
- This role is done by things like Block, Sub, Method and so forth
- Will be able to do this role in your own types (not yet supported in Rakudo)
- Requires that the method postcircumfix:<( )> is implemented
Aside: Multiple Dispatch

Since a sigil implies the doing of a role, you can use them in the signature of a multi-sub

```perl
multi what_is($it) { say "It's a scalar" }  # It's a scalar
multi what_is(@it) { say "It's an array" }  # It's an array
multi what_is(%it) { say "It's a hash" }   # It's a hash
multi what_is(&it) { say "It's code" }     # It's code

what_is([1,2,3]);                         # It's an array
what_is({ x => 4, y => 2 });              # It's a hash
what_is(-> $x { 2 * $x });                # It's code
what_is(42);                              # It's a scalar
```
Parametric Roles

- So far, we have seen roles as units of functionality that we can compose into a class or mix in to an existing object.
- A role can also take parameters.
- Allow for customization of the role's behaviour on a per-use basis.
- In the problem space of C++ templates, C#/Java Generics, System F, etc.
Parametric Roles

- Role parameters go in square brackets after the role name

```
role Can[::Contents] {
    method top_up(Contents $substance) {
        say "Yay...more {Contents.perl}!";
    }
}
```

- What goes between the square brackets is a signature, just like with a sub/method.
Parametric Roles

- To do a parametric role, pass the parameters in square brackets

```perl
class Beer { }
class Coke { }
my Can[Beer] $starobrno .= new;
$starobrno.top_up(Beer.new);  # Works
$starobrno.top_up(Coke.new);  # Exception
```

- It's much like doing a sub call
- Part of the type name; Can[Beer] is a different type to Can[Coke].
Parametric Roles

* If a role takes just one positional parameter (like our current example), you can use the of keyword to specify the parameter

```perl
my Can of Beer $staroibrno .= new;
```

* Can nest these too

```perl
my Pack of Can of Beer $six_pack .= new;
```
Can define multiple variants of a role that take different parameters

Selected using the same mechanisms as multiple dispatch

```perl
role Can[:Contents] { # One parameter
  ...
}
role Can {  # No parameters
  ...
}
```
Typed Arrays

- Typed arrays restrict what may be stored inside them

```perl
my Str @langs = <Perl Ruby PHP Python>;    # Implemented as a parametric role
@langs = 1, 2, 3;                      # Error, Int
@langs[2] = 'Smalltalk';              # Fine, Str
push @langs, 4.2;                     # Error, Num

my @langs of Str = <Perl Ruby PHP Python>;
```

- Implemented as a parametric role
- Can also write it as:
Typed Hashes

- Typed hashes restrict what can be stored as the values

```perl
my Int %word_counts;
%word_counts<monkey> = 5;      # OK
%word_counts<badger> = 0;      # OK
%word_counts<monkey> = "none"; # Error
```

- Can build up nested typed data structures

```perl
my @doc_word_counts of Hash of Int;
```
A Common Fail

- Note that the sigil already implies one level of parametric type
- What does this declare?

```perl
my Array @walruses;
```
Perl 6 Roles In Depth

A Common Fail

• Note that the sigil already implies one level of parametric type

• What does this declare?

```perl
my Array @walruses;
```

• What does this signature accept?

```perl
sub herd(Array @cats) {
    ...
}
```
A Common Fail

- Note that the sigil already implies one level of parametric type
- What does this declare?

```perl
my Array @walruses;
```

- What does this signature accept?

```perl
sub herd(Array @cats) { ... }
```

- Answer for both: an Array of Arrays.
A Common Fail

- Note that the sigil already implies one level of parametric type
- What does this declare?
  ```perl
  my Array @walruses;
  ```
- What does this signature accept?
  ```perl
  sub herd(Array @cats) { ... }
  ```
- Answer for both: an Array of Array.
- (Well, really a Positional of Array)
Perl 6 Roles In Depth

- Roles
  - Composition and mix-ins
  - Typed data structures
  - Sigils
  - Traits
So what are traits anyway?

- A Perl 6 trait is something applied to a declarand
  - A class that is currently being declared
  - A routine that is currently being declared
  - A variable that is currently being declared
Some Built-in Traits

- A method or sub is marked as being exported using a trait

```perl
module Walrus {
    sub lose_bukit() is export { ... }
}
```

- Inheritance works through trait application too

```perl
class PolarBear is Bear {
    ...
}
```
Trait Dispatch

- Which trait to do is decided by a multiple dispatch
- If the trait name is a type name (e.g. class or role), then the type is looked up and passed as the second positional argument
- Otherwise, a pair of the given name is passed
Implementing A Trait Handler

- Inside the trait implementation you can do pretty much whatever you like.
- However, often a well-behaved trait will mix in a role that provides an attribute of the same name.
- Basic example: a `doc` trait

```perl
sub answer() is doc('Compute the answer') {
    return 42;
}
say &answer.doc;
```
Implementing A Trait Handler

- Declare a role to hold the documentation string

```perl
role doc {
    has $.doc is rw;
}
```

- Then implement a trait mod to apply it to our routine

```perl
multi trait_mod:<is>(Routine $r, doc, $text) {
    $r does doc($text);
}
```
Traits On Variables

- Can also apply a trait to a container
- Here's how we write the handler...

```perl
multi trait_mod:<is>(Container $c, doc, $text) {
    $c does doc($text);
}
```

- ...and how we use it.

```perl
my %counts is doc('Count of each word');
say %counts.doc;
```
Here be dragons: for classes, the jury is still out on what you get as the declarand (the meta-class or some under-construction type object)

```perl
multi trait_mod:<is>(Class $c, doc, $text) {
    $c does doc($text);
}

class Bar is doc('Serves beer') { }
say Bar.HOW.doc;
```
Another Routine Trait Example

- Goal: install a wrapper on a routine that calls log_message on any parameter that does DebugLog

```perl
multi trait_mod:<is>(Routine $r is rw, :$logging!) { 
  ...
}
```
Another Routine Trait Example

- Goal: install a wrapper on a routine that calls log_message on any parameter that does DebugLog

```perl
multi trait_mod:<is>(Routine $r is rw, :$logging!) {
    $r.wrap(sub (*@pos, *%named) {
        ...
    });
}
```
Another Routine Trait Example

- Goal: install a wrapper on a routine that calls log_message on any parameter that does DebugLog

```perl
multi trait_mod:<is>(Routine $r is rw, :$logging!) {
    $r.wrap(sub (*@pos, *%named) {
        for @pos, %named.values -> $param {
            ...
        }
    ...
});
}
```
Perl 6 Roles In Depth

Another Routine Trait Example

- Goal: install a wrapper on a routine that calls log_message on any parameter that does DebugLog

```perl
multi trait_mod:<is>(Routine $r is rw, :$logging!) {
    $r.wrap(sub (*@pos, *%named) {
        for @pos, %named.values -> $param {
            if $param ~~ DebugLog {
                $param.log_message("Passed to " ~ $r.name);
            }
        }
    });
}
```

Another Routine Trait Example

- Goal: install a wrapper on a routine that calls log_message on any parameter that does DebugLog

```perl
multi trait_mod:<is>(Routine $r is rw, :$logging!) {
  $r.wrap(sub (*@pos, *%named) {
    for @pos, %named.values -> $param {
      if $param ~~ DebugLog {
        $param.log_message("Passed to ", $r.name);
      }
    }
  }nextsame;
});
```
That's All!
Thank You!
Questions?