

# Object Orientation, The Perl 6 Way



**Jonathan Worthington**  
Bratislava.pm

# Object Orientation, The Perl 6 Way

## This Talk

- Was originally given at this year's YAPC
- But nobody from Bratislava.pm came!
- So now I'm going to subject them to my talk anyways... :-)
- All of the code shown in this talk runs on Rakudo today
  - Rakudo = implementation of Perl 6 that I hack on

# Classes

## The class Keyword

- In Perl 5, we use **package** whether we are writing a class or not
- In Perl 6, we differentiate them
  - **class** = a class; can be instantiated and has instance data
  - **role** = re-usable unit of functionality that can be composed into a class
  - **module** = subs in a namespace

# Object Orientation, The Perl 6 Way

## Today's Examples

- I love to travel
- Going to implement a simple system to manage journeys, using the OO features of Perl 6
- To start off with, we'll introduce classes to represent places and journeys

```
class Place {  
}  
class Journey {  
}
```

# Object Orientation, The Perl 6 Way

## Attributes With Accessors

- Use the **has** keyword to introduce attributes

```
class Place {  
    has $.name;  
    has $.population is rw;  
}
```

- The `. twigil` states an accessor method should be generated
- The **rw** trait specifies that the accessor method should return an lvalue

# Object Orientation, The Perl 6 Way

## Attributes With Accessors

- Can also use the **!** twigil to declare a private attribute

```
class Journey {  
    has $.from;  
    has $.to;  
    has $!start_time;  
    has $!end_time;  
}
```

- Even public attributes have **\$!name** declared; it refers to the underlying storage location



# Object Orientation, The Perl 6 Way

## Methods

- Differentiated from subs in Perl 6; use the `method` keyword
- No need to list invocant in parameter list

```
method opinion() {  
    say "I luvs ma travelz.";  
}
```

- Aside: Perl 6 has parameter lists, so you can list the parameters taken, as in many other languages. To cover it in detail would take another 30 minutes...



# Object Orientation, The Perl 6 Way

## Some More Methods

- Methods that work with our private attributes

```
method start() { $!start_time = time(); }
method end()   { $!end_time = time(); }
method duration() {
    if !$!start_time {
        die "Journey not started yet.";
    } else {
        return $!end_time ??
            $!end_time - $!start_time !!
            time() - $!start_time;
    }
}
```

## Proto-Objects

- In Perl 6, there is no class object
- Instead, when you declare a class, a proto-object is installed in the namespace under the name of the class
  - An "empty instance" of the class
  - Can call any methods that do not access the state
  - This includes the **new** method

# Object Orientation, The Perl 6 Way

## Instantiation and Method Calls

- You can instantiate the class by calling the **new** method

```
my $city = Place.new();  
my $trip = Journey.new();
```

- Note the new syntax in Perl 6 for method calls; we now use `.`
- Can call the opinion method on the instance:

```
$trip.opinion(); # I luvz ma travelz.  
$trip.opinion;  # same
```

# Object Orientation, The Perl 6 Way

## Initializing Attributes

- Pass named parameters to **new**

```
my $lhasa = Place.new(  
    name => 'Lhasa',  
    population => 257400  
);  
my $xian = Place.new(  
    name => 'Xian',  
    population => 2670000  
);  
my $trip = Journey.new(  
    from => $lhasa,  
    to    => $xian  
);
```



# Object Orientation, The Perl 6 Way

## Inheritance

- There's More Than One Way To Travel
- Make subclasses of Journey for them

```
class TrainJourney is Journey {  
    has $.train_no;  
    has $.coach;  
    has $.place;  
}  
class Flight is Journey {  
    has $.flight_no;  
}  
class Walk is Journey {  
}
```

# Object Orientation, The Perl 6 Way

## Initializing Parent Attributes

- To initialize the attributes of a parent class, need slightly different syntax

```
my $trip = TrainJourney.new(  
    Journey{ from => $lhasa, to => $xian },  
    train_no => 'T28',  
    coach => '12',  
    place => '68'  
);
```

- You may find this messy; in that case you are free to define your own **new** method that does what you like

# Object Orientation, The Perl 6 Way

## Auto-vivification

- Doing hash-like indexing into a proto-object actually returns a copy of the proto-object with an auto-vivification closure attached

```
my $from_home = Journey{  
    from => $bratislava  
};  
my $to_yapc = $from_home.new(  
    to => $copenhagen  
);  
say $to_yapc.from.name; # Bratislava  
say $to_yapc.to.name;   # Copenhagen
```



# Object Orientation, The Perl 6 Way

## Delegation

- We might like to have `from_name` and `to_name` methods on our `Journey` class
- They just call the `name` method on the `Place` class
- Use **`handles`** to generate them

```
class Journey {  
    has $.from handles :from_name<name>;  
    has $.to handles :to_name<name>;  
    # ...rest of the class...  
}
```

# Object Orientation, The Perl 6 Way

## Delegation

- The handles trait verb doesn't just take a pair, but can also take
  - A single string, to delegate one method and not change the name
  - A list of strings and pairs to delegate without or with name changes (can mix them together in one list)
  - More things not yet implemented (including regex/substitutions)

# Roles

# Object Orientation, The Perl 6 Way

## Pollution

- We want to add pollution tracking functionality into our journeys



# Object Orientation, The Perl 6 Way

## Pollution

- Only want to apply it to some classes
  - A Flight and TrainJourney will pollute, but a Walk will not
- We'd also like to be able to re-use the functionality of calculating pollution on other things that are not Journeys

## Introducing Roles

- Allow us to implement a piece of functionality (methods and attributes) that can be composed into a class
- Composition is flattening
  - Conflicts between methods of the same name from different roles will be flagged up at compile time
  - Class gets last say in resolving the conflict

# Object Orientation, The Perl 6 Way

## Introducing Roles

- Implement a role with two attributes and a method

```
role Pollute {  
    has $.carbon_per_unit;  
    has $.unit;  
    method carbon_footprint($units) {  
        return $units * $!carbon_per_unit;  
    }  
}
```

- Attributes declared with **has** as if they were declared in the class



# Object Orientation, The Perl 6 Way

## Composing Roles

- We compose roles into classes using the **does** keyword

```
class TrainJourney is Journey does Pollute {  
    has $.train_no;  
    has $.coach;  
    has $.place;  
}  
class Flight is Journey does Pollute {  
    has $.flight_no;  
}
```

- Use multiple **does** before each role name to compose many roles

## Roles As Mix-ins

- As well as composing roles at compile time, we can also treat them as mix-ins at runtime
- This derives a new anonymous class containing the methods and attributes provided by the role
- Note: methods in mixed-in role override those in the class; no collision detection here

# Object Orientation, The Perl 6 Way

## Roles As Mix-ins

- Useful for adding on extra things that we weren't expecting...

# Object Orientation, The Perl 6 Way

## Roles As Mix-ins

- Useful for adding on extra things that we weren't expecting...
- ...like delays...



# Object Orientation, The Perl 6 Way

## Roles As Mix-ins

```
role Delay {  
    has $.duration is rw;  
    method opinion() {  
        if $.duration <= 5 {  
            say "I luvs ma travelz.";  
        } elsif $.duration < 30 {  
            say "It's fine.";  
        } elsif $.duration < 60 {  
            say "*sigh*";  
        } else {  
            say "AAAARRRRRRRGGGGHHHH!!!";  
        }  
    }  
}
```

# Object Orientation, The Perl 6 Way

## Roles As Mix-ins

- We use the **does** infix operator to mix a role in at runtime

```
$journey does Delay;  
$journey.duration = 70;  
$journey.opinion; # AAAARRRRRRRGGGGGHHHH!!!
```

- If we have just one attribute, we have some special syntax to initialize it in one go (it's not actually a sub call)

```
$journey does Delay(40);  
$journey.opinion; # *sigh*
```

# Enumerations



# Object Orientation, The Perl 6 Way

## Enumerations

- The enum keyword allows you to introduce an enumeration type

```
enum Purpose <BusinessTrip Vacation>;
```

- By default, the values map to Int values starting at 0

```
say BusinessTrip;    # 0  
say Vacation;        # 1
```

- But you can use strings too...

```
enum Phonetic [ :Alpha<A>, Bravo, Charlie,  
                Delta, Echo, ..., Zulu ];
```

# Object Orientation, The Perl 6 Way

## Enumerations

- You can use an enumeration as a role and mix in into an existing object

```
$journey does Purpose(Vacation);
```

- Additionally, there is the but operator, which makes a copy of the value and then operates on that; it also knows how to generalize an enum value to its type

```
sub make_vacation($trip) {  
    return $trip but Vacation;  
}
```

# Object Orientation, The Perl 6 Way

## Enumerations

- After mixing in with the `but` or `does` operator, you get a method of the same name as the enum, returning the current value

```
$journey does Purpose(BusinessTrip);  
say $journey.Purpose; # 0
```

- As well as methods for each of members of the enum returning a Bool

```
say $journey.BusinessTrip; # 1  
say $journey.Vacation;      # 0
```

# Other Bits In Rakudo

# Object Orientation, The Perl 6 Way

## Meta-classes (incomplete)

- Each class has a meta-class, which can be retrieved using the .HOW macro

```
my $meta = $trip.HOW;
```

- Will provide a way to get a list of methods, attributes, parents and roles that a class does
- Use .^ to call methods on meta-class

```
my @methods = $trip.HOW.methods($trip);  
my @methdos = $trip.^methods(); # same
```

# Object Orientation, The Perl 6 Way

## Calling Sets Of Methods

- Not sure if a class has a method, and don't want an exception, but an undef back instead?

```
$fp = $trip.?carbon_footprint($kms) // 0;
```

- Can also use .\* to call all methods of the name (including those in super-classes) and .+ to enforce that at least one method will be called

```
my @captures = $trip.+opinion;
```

## More Attribute Stuff

- I showed role attributes declared with `has`, which are as if they were declared in the class
- You can also declare role-private attributes, invisible inside the class

```
my $!guts;
```

- There are also class attributes – essentially lexicals with accessors

```
my @.instances;
```



# **Rakudo OO Implementation Status**

## Object Orientation, The Perl 6 Way

### Probably Not Half Way Yet

- Much progress has been made in implementing the features shown today
- However, the Perl 6 object model is pretty rich, so there's probably about this much again worth of work to get the rest of the features in
- Once we've got those features in, there will also be some work to do on feature interaction and edge cases

# Object Orientation, The Perl 6 Way

## Still Lots To Play With

- With many of the common things implemented, there's plenty to play with today
- Downloading and building Rakudo, playing with it, breaking it and reporting bugs helps
- Sending in a test case we can add to the specification tests helps even more ;-)

**D'akujem**

# Questions?