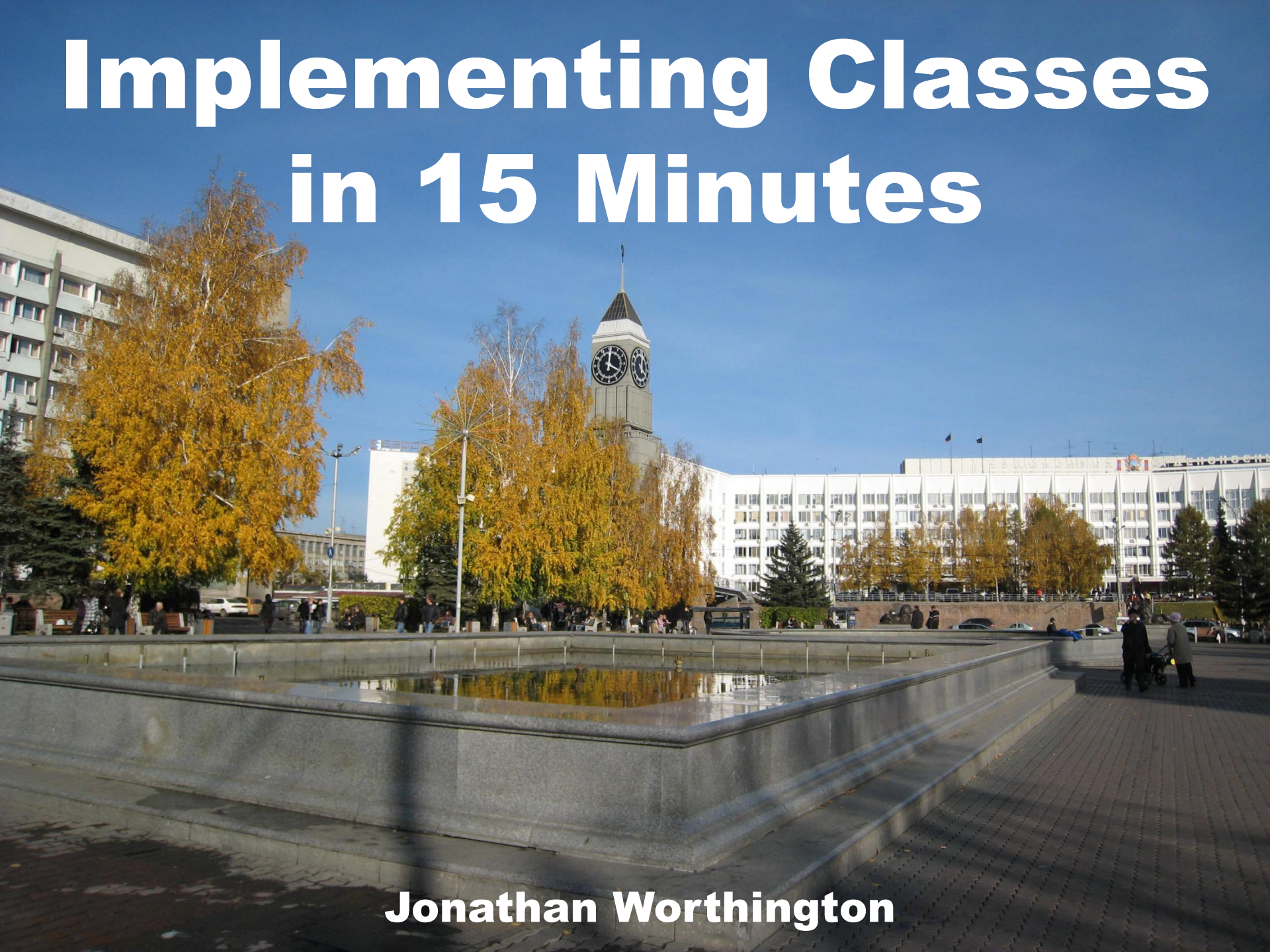
Implementing Classes in 15 Minutes



Jonathan Worthington

Reflection

Finding out what methods, parents etc an object has

Reflection

Finding out what methods, parents etc an object has

When it comes to meta-objects

that's only part of the story.

Meta-objects

Objects that describe the way other objects work

Not just for introspection, but for implementation

Class behaviour is just defined in the class meta-object

It's easy!

Meta-objects have methods that respond to various "events" that occur as we compile a package

Implementing classes

=

Writing methods

Example

```
class Stroopwafel is Cake {
    has $!stroop;
    method eat() {
        say “om nom nom”;
    }
}
```

Example

```
class Stroopwafel is Cake {
    has $!stroop;
    method eat() {
        say "om nom nom";
    }
}
```

new_type

Example

```
class Stroopwafel is Cake {
    has $!stroop;
    method eat() {
        say "om nom nom";
    }
}
```

add_parent

Example

```
class Stroopwafel is Cake {
    has $!stroop;
    method eat() {
        say "om nom nom";
    }
}
```

add_attribute

Example

```
class Stroopwafel is Cake {
    has $!stroop;
    method eat() {
        say "om nom nom";
    }
}
```

add_method

Example

```
class Stroopwafel is Cake {
    has $!stroop;
    method eat() {
        say "om nom nom";
    }
}
```

compose

New Type

Create a type object

Provide default name and default representation

```
method new_type(:$name = '<anon>',
                :$repr = 'P6opaque') {
    my $metaclass := self.new(:name($name));
    nqp::type_object_for($metaclass, $repr);
}
```

Methods

Need a place to store them…

```
has %!methods;
```

And a way to add them…

```
method add_method($obj, $name, $code) {
    if %!methods{$name} {
        die("Duplicate method $name");
    }
    %!methods{$name} := $code_obj;
}
```

Attributes

Need a place to store them...

```
has %!attributes;
```

And a way to add them...

```
method add_attribute($obj, $attr) {
    if %!attributes{$attr.name} {
        die("Duplicate attribute " ~
            $attr.name);
    }
    %!attributes{$attr.name} := $attr;
}
```

Inheritance

A place to store it

```
has $!parent;
has $!parent_set;
```

And a way to add it...

```
method add_parent($obj, $parent) {
    if $!parent_set {
        die("Can only have one parent");
    }
    $!parent := $parent;
    $!parent_set := 1;
}
```

Composition

When we're finished declaring the class, need to compute MRO and compose attributes

```
has @!mro;

method compose($obj) {
    @!mro := self.compute_mro($obj);
    for %!attributes.values {
        $_.compose($obj);
    }
}
```

MRO

Method Resolution Order

The order we walk classes when looking for methods

Easy for single inheritance; just walk up the list of parents

MRO

Protocol wants us to have a parents method that returns a list of parents (will be 0 or 1 items for single inheritance)

```
method parents($obj) {
    $!parent_set ?? [$!parent] !! []
}
```

MRO

Then use that to compute the method resolution order

```
method compute_mro($obj) {
    my @mro;
    my @cur_parent := [self];
    while @cur_parents {
        my $p := @cur_parents[0]
        @mro.push($p);
        @cur_parents := $p.HOW.parents($p);
    }
    return @mro;
}
```

Dispatch

Meta-object should expose the methods it knows about

```
method method_table($obj) {
    %!methods
}
```

Dispatch

Implement method dispatch with MRO and method_table

```
method find_method($obj, $name) {
    for @!mro {
        my %meths := $_.HOW.method_table($obj);
        my $found := %meths{$name};
        if defined($found) {
            return $found;
        }
    }
    nqp::null() # As not found sentinel
}
```

That's it!

We've now implemented all that we need to have classes that support:

Methods and dispatch
Attributes
Single inheritance

Well, nearly...

We've missed various things out of this...

Various bits of introspection

Publishing method cache

isa and can methods

NQPClassHOW

Implementation of a subset of Perl 6's class support, so far as NQP needs to have it

Goal is that this will look identical for running on Parrot, .Net CLR, JVM, etc.

Dank je wel!

Questions?