A first go at Perl 6 “Weather App”

Florent Mara
<florent.mara@gmail.com>

Introduction

What this presentation is

- A first attempt at a Perl 6 application
- Potentially solving a problem
- WARNING: Contains some Perl code!

What it is not

- Perl 5 versus Perl 6
- Me knowing everything about Perl 6 (or Perl 5)
- A review of Perl 6 features

Disclaimer

This presentation is largely based on "Perl 6 Hands-on workshop: Weatherapp" written by Zoffix Znet (aka Peter Evans)

Weather App Workshop - part 1 / 3

- Project purpose:

  "... build an application that talks to an API. The app will tell us weather at a location we provide."

- Preparation covers git (but not Perl6)
- Design doc: DESIGN.md
- Research and prior art:
  - Perl 6 returned no results
  - Multi-languages: C with NativeCall, one of the 8 Inline::* modules
  - Perl 5 Weather::Underground and Weather::OpenWeatherMap

Installing Perl 6

- Google first result: Rakudo.org
  - *git clone https://github.com/tadzik/rakudobrew ~/.rakudobrew*
    echo 'export Path=~/.rakudobrew/bin:$PATH >> ~/.bashrc
    source ~/.bashrc

    rakudobrew build moar
    rakudobrew build panda

    panda install Task::Star

- Instructions to install Perl 6 are different on Perl6.org

Weather App Workshop - part 2 / 3

- More about the benefits of writing design documentation
- … and talking to ‘the future boss’
- … and tests
- … and user documentation
- … and sound architecture

-> Adding ‘General Outline’ and ‘Interface Details’ sections into DESIGN.md

```
# INTERFACE DETAILS

## EXPORTED SUBROUTINES

### `weather-for`

    my $result = weather-for 'Brampton', 'ca';

    printf "Current weather is: %d°C, %dmm precip/3hr, wind %dm/s\n",
        .temp, .precip, .wind given $result;

Takes two positional arguments—name of the city and [ISO country
code](http://www.nationsonline.org/oneworld/country_code_list.htm)—to
provide weather information for. The country is optional and by default is
not specified.

Returns a `Weather::Result` object on success, otherwise returns
a `Failure`. The object provides these methods:

#### `.temp`

    say "Current temperature is $result.temp()°C"

Takes no arguments. Returns the `Numeric` temperature in degrees Celcius.

#### `.precip`

    say "Expected to receive $result.precip()mm/3hr of percipitation";

Takes no arguments. Returns the `Numeric` amount of precipitation in
millimeters per three hours.

#### `.wind`

    say "Wind speed is $result.wind()m/s";

Takes no arguments. Returns the `Numeric` wind speed in meters per second.
```

Weather App Workshop - part 3 / 3

- Test driven development
- Install testing modules: *panda install Test::When Test::Meta*
- Test files use .t extension and are located in t/ directory
- Documents a recommended folder structure

```
t
├── key
├── 01-use.t
├── author
│   └── 01-meta.t
└── online
    └── 01-weather-for.t
```

Weather App Workshop - part 3 / 3

- t/01-use.t, t/author/01-meta.t and

```
# t/01-use.t
use Test;

use-ok 'WebService::Weather';

done-testing;
```

```
# t/author/01-meta.t
use Test::When <author>;
use Test;
use Test::META;

meta-ok;

done-testing;
```

```perl
# t/online/01-weather-for.t
use Test::When <online>;
use Test;
use WebService::Weather;

for ('London'), ('London', 'ca') -> $args {
    subtest {
        my $result = weather-for |$args;

        isa-ok $result, 'WebService::Weather::Result',
            'result is of a correct data type';

        does-ok $result."$_"(), Numeric, "$_ is numerical"
            for <temp wind precip>;

        cmp-ok $result.temp,   &[<],  70,   'temperature is not too high';
        cmp-ok $result.temp,   &[>],  -100, 'temperature is not too low';
        cmp-ok $result.wind,   &[<],  120,  'wind speed is not too high';
        cmp-ok $result.wind,   &[>=], 0,    'wind speed is not too low';
        cmp-ok $result.precip, &[<],  3200, 'precipitation is not too high';
        cmp-ok $result.precip, &[>=], 0,    'precipitation is not too low';
    }, "Testing with args: $args";
}

isa-ok weather-for('blargs' x 12), Failure,
    'we get a Failure for unknown city';

done-testing;
```

Running tests

- *prove -e ‘perl6 -Ilib’ -vr t/*
- Or alias the command
  - alias prove6=”prove -e ‘perl6 -Ilib’ -vr”
  - *prove6 t/*

```
florent@bladeranger:~/workspace/Weatherapp$ prove6 t/
t/01-use.t ..................
not ok 1 - The module can be use-d ok

# Failed test 'The module can be use-d ok'
```

That’s it!

Why look at Perl6?

- Math that works
- Useful types and simpler sigils
- Function signatures
- Optional gradual typing
- Vector operators
- Parallelism, concurrency and asynchrony
- …
- Something called OO (?!?)

Some Perl6 code - fibonacci

```
my $addition = 0.1 + 0.2;
say $addition;                # 0.3

subset NonNegative of Int where * >= 0;

sub fib(NonNegative $nth) returns UInt {
    given $nth {
        when 0  { 0 }
        when 1  { 1 }
        default { fib($nth-1)+fib($nth-2) }
    }
}

say fib(8);                   # 21
say fib(-3)                   # Constraint type check failed for parameter '$nth'
```

More Perl6 code - A Class

```
class Point {
    subset PointLimit of Rat where -10 .. 10;
    has PointLimit $.x is rw is required;
    has PointLimit $.y is rw is required;

    method gist { "[$.x,$.y]" }
}

my $point = Point.new( x => 5.0, y => 3.0 );
say $point.x;            # 5
say $point;              # [5,3]
```

More Perl6 code - Parallel Computing

```
my $nths = 4000, 5, 100, 2000;

my @primes = gather for $nths -> $n {
    take start {
        (1 .. *).grep(*.is_prime)[$n]
    };
}
.say for await @primes;
```

Found bug #123072

Random Perl 6 features

```
say chars("nyanCat");        # 7
say uc("nyanCat");           # NYANCAT
say lc("nyanCat");           # nyancat
say tclc("nyanCat");         # Nyancat

say (+"42").WHAT;            # (Int) `+` coerce value to a Int.

loop ( my $i = 10; $i > 0; $i-- ) { say $i; }
say "LIFTOFF";

say 3 <=> 5;                 # Less. Using 'spaceship' operator
say (3 <=> 3).WHAT;          # type: (Order)

my %capitals = UK => 'London', Slovakia => 'Bratislava';
for %capitals.kv -> $country, $city {
    say "$city is the capital of $country";
}

say (1..6).roll(8); # Same as dice.
say (1..6).pick(8); # each element of the list is only picked once.
```

Conclusion

- No weather app

- Easy to get started
- Powerful features
- Familiar technology for Perl developers
- Most of it works already
- What next? Complete the weather app

Addendum - Webservice::Weather

- Perl 6 Weather module

```
unit package WebService::Weather;

use Weather;

my $p = Weather.new(apikey => '50a                                    04f');

sub weather-for( $capital, $country) {

    my $result = $p.get-weather( $capital, $country);

    return $result;
}

say weather-for('London', 'ca');
```

Bonus slide 1 - some Perl 5

Which is slowest?

```
my $id    = 1;
my $value = 5;
for my $loop ( 0 ... 100000 ) {
        my $rec = {
            id    => $id,
            value => $value,
            atime => $loop,
          };
}
```

```
my $hash = { id => 1, value => 2};
for my $loop2 ( 0 ... 100000 ) {
    my $rec = { %$hash };
    $rec->{atime} = $loop2;
}
```

Neither (~100ms)

Bonus slide 2

Which is slowest?

```
my $var4 = 0;
for my $loop4 ( 0 ... 1000000 ) {
    my $modulo = $loop4 % 5;

    given($modulo) {
        when( 0 ) { $var4 += $modulo; }
        when( 1 ) { $var4 += $modulo; }
        when( 2 ) { $var4 += $modulo; }
        when( 3 ) { $var4 += $modulo; }
        when( 4 ) { $var4 += $modulo; }
    }
}
```

~ 600 ms

```
my $var3 = 0;
for my $loop3 ( 0 ... 1000000 ) {
    my $modulo = $loop3 % 5;

    if( $modulo == 0 ) {
        $var3 += $modulo;
    } elsif( $modulo == 1 ) {
        $var3 += $modulo;
    } elsif( $modulo == 2 ) {
        $var3 += $modulo;
    } elsif( $modulo == 3 ) {
        $var3 += $modulo;
    } elsif( $modulo == 4 ) {
        $var3 += $modulo;
    }
}
```

> 300 ms

Questions

Florent Mara
<florent.mara@gmail.com>

RE GEN KNOW WHAT'S RIGHT