CONTRACTS.RUBY

assert on steroids

http://egonschiele.github.io/contracts.ruby

PRESENTATION PLAN:

1. What is it and what it offers?
2. Basic usage
3. Custom contracts
4. Performance
5. Summary

1. WHAT IS IT AND WHAT IT OFFERS

- Provides assertions on arguments and return values
- Forces the proper usage of an API
- Introduces method overloading (like in C++ and Java)
- Adds invariants to objects

What does it look like?

```
Contract String => nil
def greeting(name)
  puts "Hello, #{name}!"
end

greeting('Szymon') # => Hello, Szymon!
```

When argument or return value doesn't match...

```
Contract String => nil
def greeting(name)
  puts "Hello, #{name}!"
end

greeting(1)
```

... it raises an exception

```
Contract violation for argument 1 of 1: (ParamContractError)
        Expected: String,
        Actual: 1
        Value guarded in: Object::greeting
        With Contract: String => NilClass
        At: snippet2.rb:5
```

2. BASIC USAGE

What sort of contracts do we have out of the box?

- Primitives
- Boolean operation
- Arrays and Hashes
- Special cases
- Invariants
- Method overloading

PRIMITIVES

```
Contract Num, Num => Num
def add(x, y)
  x + y
end
add(1, 2) # => 3

Contract Pos, Neg => Num
def add_positive_to_negative(x, y)
  x + y
end
add_positive_to_negative(1, -1) # => 0
add_positive_to_negative(1, 1) # => Contract violation
```

We can also use literals like 1, "a", {} or nil

BOOLEAN OPERATIONS

```
Contract Or[Fixnum, Float] => Or[Fixnum, Float]
def double(x)
  2 * x
end
double(1) # => 2
double(1.5) # => 3
double("1") # => Contract violation
```

ARRAYS AND HASHES

```
Contract HashOf[Symbol, String] => String
def serialize(params)
  return JSON.dump(params)
end
serialize(foo: 'bar') # => {"foo":"bar"}
serialize(foo: 10) # => Contract violation

Contract ArrayOf[Num] => Num
def sum(numbers)
  numbers.reduce(:+)
end
sum(1.upto(10).to_a) # => 55
sum([1, 2, 3, '4']) # => Contract violation
```

SPECIAL CASES

- `Any` - passes for any argument (no constraint)
- `None` - when a method takes no arguments
- `Maybe(Num)` - argument can be nil or given contract
- `RespondTo[:foo, :bar]` - must respond to the given methods
- `Exactly(Numeric)` - won't pass for subclasses
- `Send[:valid?]` - the method must return true

INVARIANTS

```
class Order
  include Contracts, Contracts::Invariants
  attr_accessor :client, :products
  invariant(:client) { client.nil? == false }
  invariant(:products) { Array(products).any? }

  Contract Any, Any => Order
  def initialize(client, product)
    self.client = client
    self.products = products
    self
  end
end
Order.new(nil, nil)
```

```
failure_callback': Invariant violation: (InvariantError)
            Expected: client condition to be true
            Actual: false
            Value guarded in: Order::initialize
            At: snippet6.rb:13
```

METHOD OVERLOADING

```ruby
Contract ->(n) { n < 12 } => Hash
def get_ticket(age)
  { ticket_type: :child }
end

Contract ->(n) { n >= 12 } => Hash
def get_ticket(age)
  { ticket_type: :adult }
end

p get_ticket(11) # => {:ticket_type=>:child}
p get_ticket(12) # => {:ticket_type=>:adult}
```

What is this lambda doing there... ?

3. CUSTOM CONTRACTS AKA WHERE THE FUN BEGINS

There are 3 ways to define a custom contract

- A proc
- A class with a `valid?` class method
- A class with a `valid?` instance method

PROCS

```
is_even = ->(num) { num % 2 == 0 }

Contract is_even => String
def check(num)
  "Yay!"
end
```

Produces the following error when fails

```
Contract violation for argument 1 of 1: (ParamContractError)
        Expected: #<proc:0x007fe455849ec8@snippet8.rb:4 (lambda)="">,
        Actual: 1
            </proc:0x007fe455849ec8@snippet8.rb:4>
```

CLASS WITH `valid?` CLASS METHOD

```
class EvenNumber
  def self.valid?(num)
    num % 2 == 0
  end
end

Contract EvenNumber => String
def check(num)
  "Yay!"
end
```

Produces the following error when fails

```
Contract violation for argument 1 of 1: (ParamContractError)
        Expected: EvenNumber,
        Actual: 1
```

CLASS WITH `valid?` INSTANCE METHOD

```
class Or < CallableClass
  def initialize(*vals)
    @vals = vals
  end

  def valid?(val)
    @vals.any? do |contract|
      res, _ = Contract.valid?(val, contract)
      res
    end
  end
end
```

PERFORMANCE

http://adit.io/posts/2013-03-04-How-I-Made-My-Ruby-Project-10x-Faster.html

IO - opening websites and reading their body 100 times

```
contracts.ruby|master ⇒ RUBYLIB=./lib ruby benchmarks/io.rb
                                  user     system      total
testing download                3.970000   0.360000   4.330000 ( 48.
testing contracts download      3.810000   0.290000   4.100000 ( 48.
```

Invariants - ran 1 mil times

```
contracts.ruby|master ⇒ RUBYLIB=./lib ruby benchmarks/invariants.rb
                                        user     system      total
testing contracts add                 4.850000   0.040000   4.890000
testing contracts add with invariants  6.180000   0.030000   6.210000
```

For a real world usage (making requests)

the performance drop is barely visible

SUMMARY

Pros

- Very good in the development process
- Explicit dependency and results declaration
- You can disable contracts through `NO_CONTRACTS` env var
- Method overloading is sweet
- Invariants are also sweet

Cons

- No way to change failure handling
- It does degrade the performance a bit
- Without a proper failure handling it can't be used in code that is facing the end user

THANK YOU FOR YOUR ATTENTION.
ANY QUESTIONS?