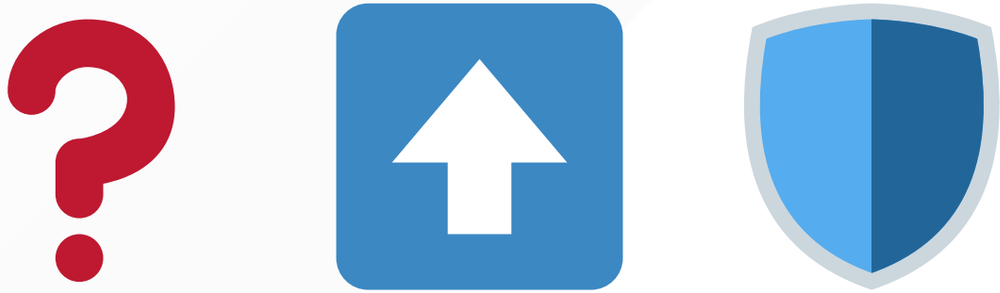
Programmation par contrat en ruby

Yves Brissaud

UT, TDD, BDD, *tests



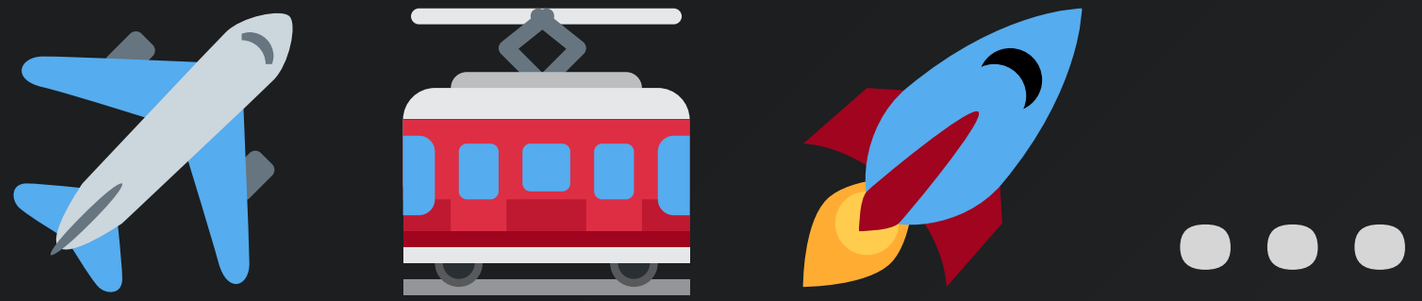
La programmation par contrat

“ est un **paradigme** de programmation dans lequel le déroulement des traitements est régi par de règles. ”

“ C'est une méthode de programmation **semi-formelle** dont le but principal est de réduire le nombre de bugs dans es programmes. ”

-- wikipedia

Eiffel, 1985



- précondition
- postcondition
- invariant

Précondition

Responsabilité sur l'appelant

Postcondition

Responsabilité sur l'appelé

Invariant

Toujours vrai

Examples

sqrt

```
def sqrt(x)
  Math.sqrt(x)
end
```

```
def sqrt(x)
  Math.sqrt(x)
end
```

```
> sqrt(2)
1.414213563730951
```

```
def sqrt(x)
  Math.sqrt(x)
end
```

```
> sqrt(-2)
?
```

```
def sqrt(x)
  Math.sqrt(x)
end
```

```
> sqrt(-2)
Math::DomainError: Numerical argument is out of domain
```

```ruby
def sqrt(x)
  unless x >= 0
    puts "x should be positive"
    nil
  end
  Math.sqrt(x)
end
```

```ruby
def sqrt(x)
  unless x >= 0
    puts "x should be positive"
    nil
  end
  Math.sqrt(x)
end
```

```
> sqrt(-2)
x should be positive
nil
```

```ruby
def sqrt(x)
  unless x >= 0
    puts "x should be positive"
    nil
  end
  Math.sqrt(x)
end
```

```
> sqrt(nil)
?
```

```ruby
def sqrt(x)
  unless x >= 0
    puts "x should be positive"
    0
  end
  Math.sqrt(x)
end
```

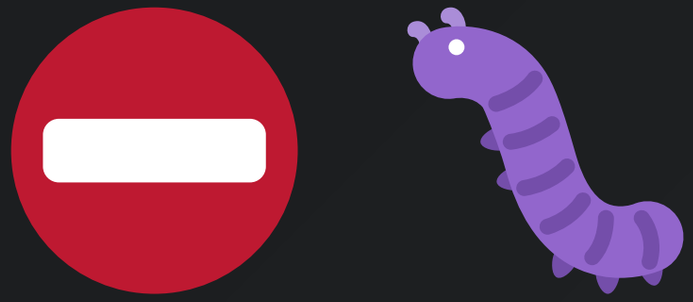
```
> sqrt(nil)
NoMethodError: undefined method `>=' for nil:NilClass
```

- Qui gère la bonne valeur en entrée ?

- Qui gère la bonne valeur en sortie ?

- entrée gérée par l'appelé

- sortie gérée par l'appelant



```
Contract Pos => Pos
def sqrt(x)
  Math.sqrt(x)
end
```

- définition claire de l'entrée
  - Pos => `Numérique >= 0`
- définition claire de la sortie
  - Pos => `Numérique >= 0`

```
Contract Pos => Pos
def sqrt(x)
  Math.sqrt(x)
end
```

```
> sqrt(-2)
?
```

```
Contract Pos => Pos
def sqrt(x)
  Math.sqrt(x)
end
```

```
> sqrt(-2)
ParamContractError: Contract violation for argument 1 of 1:
        Expected: Pos,
        Actual: -2
        Value guarded in: Object::sqrt
        With Contract: Pos => Pos
```

```
Contract Pos => Pos
def sqrt(x)
  Math.sqrt(x)
end
```

```
> sqrt(nil)
?
```

```
Contract Pos => Pos
def sqrt(x)
  Math.sqrt(x)
end
```

```
> sqrt(-2)
ParamContractError: Contract violation for argument 1 of 1:
        Expected: Pos,
        Actual: nil
        Value guarded in: Object::sqrt
        With Contract: Pos => Pos
```

- entrée gérée par ~~l'appelé~~ l'appelant

```
Contract Pos => Pos
def sqrt(x)
  "haha"
end
```

```
> sqrt(2)
?
```

```
Contract Pos => Pos
def sqrt(x)
  "haha"
end
```

```
> sqrt(2)
ReturnContractError: Contract violation for return value:
        Expected: Pos,
        Actual: "haha"
        Value guarded in: Object::sqrt
        With Contract: Pos => Pos
```

- sortie gérée par ~~l'appelant~~ l'appelé

Et si on veut `sqrt(nil) == 0` ?

Classique

```
Contract Maybe[Pos] => Pos
def sqrt(x)
  return 0 if x.nil?
  Math.sqrt(x)
end
```

Classique

```
Contract Maybe[Pos] => Pos
def sqrt(x)
  return 0 if x.nil?
  Math.sqrt(x)
end
```

Method overloading

```
Contract Pos => Pos
def sqrt(x)
  Math.sqrt(x)
end

Contract nil => 0
def sqrt(_)
  0
end
```

```
> sqrt(2)
1.414213563730951
> sqrt(nil)
0
```

Aucun, n'importe quoi

```
Contract None => Any
```

Hash, Array

```
Contract ArrayOf[String] => Any
```

```
Contract HashOf[Symbol => Date] => ArrayOf[{ :date => Num }]
```

Keyword arguments

```
Contract String, KeywordArgs[
  port: Optional[Num],
  user: String,
  password: And[String, -> (p) p.length >= 8]
] => Connection
def connect(host, port: 5000, user:, password:)
```

Functions

```
Contract Func[RangeOf[Date] => Enum[:workweek, :weekend]] => Any
def set_callback(range_to_day_type)
```

Duck typing

```
Contract RespondTo[:parent_user] => User
def get_parent(obj)
  obj.parent_user
end

Contract Any => NoParentError
def get_parent(obj)
  NoParentError.new obj
end
```

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

https://github.com/egonSchiele/contracts.ruby

\+ code auto documenté (à jour)

\+ responsabilités claires

\+ moins de surprise à l'exécution

\+ 💗 method overloading

\- coût à l'exécution (désactivable)

\- idiomatique

Thanks 🙏

SQUARESCALE

https://www.squarescale.com