Liskov substitution principle

Good coding practice in real life

Vladimir Alekseichenko

Agenda

- LSP and OCP
- Formal and informal definitions LSP
- Some examples with violation LSP

LSP and OCP

Motivation for LSP comes from OCP (at least partly)

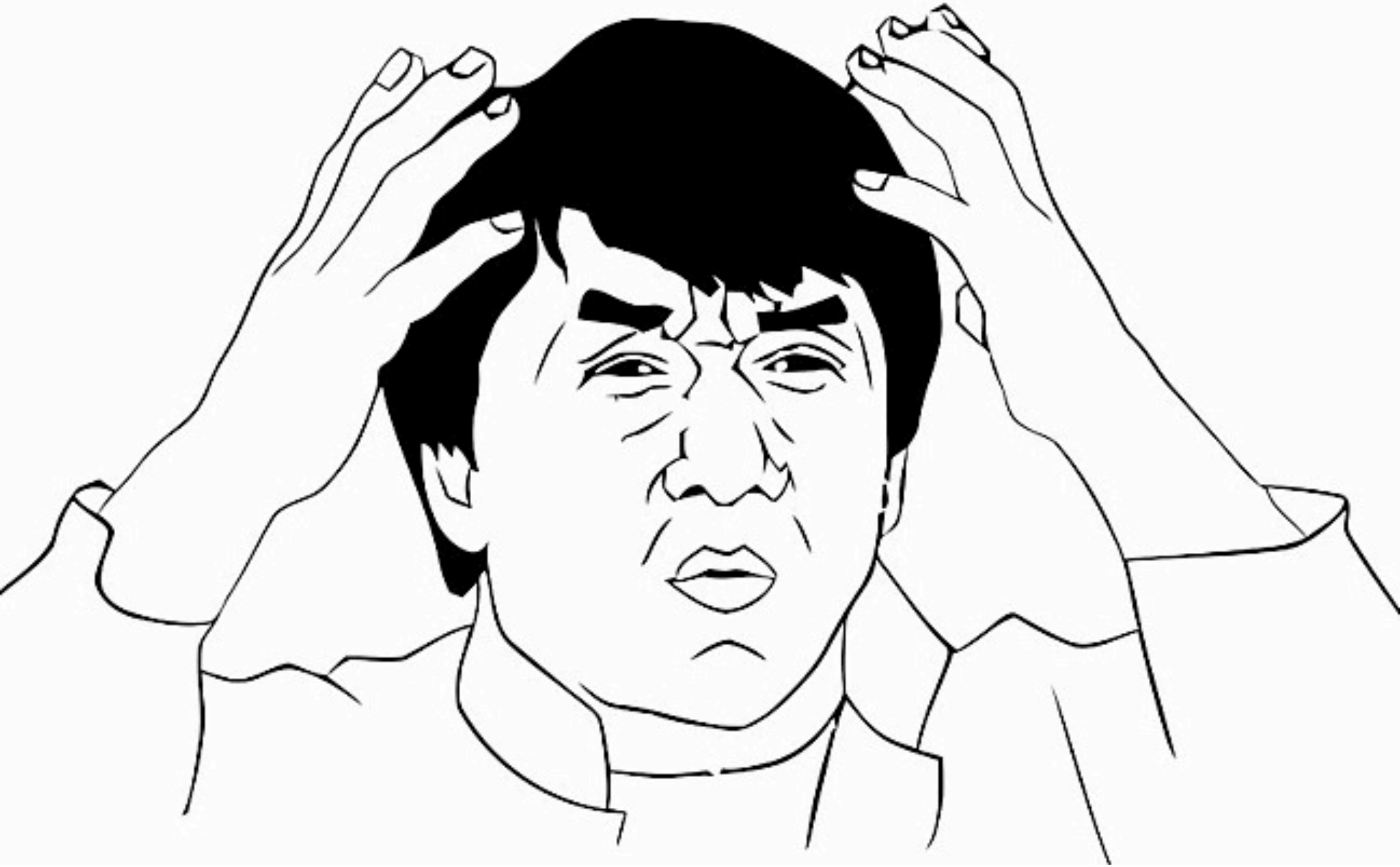
Barbara Liskov

http://spectrum.mit.edu/articles/top-prize/

1987

Definition

If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T.



Informal definition

Subtypes must be substitutable for their base types.

Examples

https://www.flickr.com/photos/doug88888

```ruby
class Rectangle
  attr_accessor :height, :width
end
```

```ruby
r = Rectangle.new
r.height = 111
r.width = 333
#r -> <Rectangle:... @height=111, @width=333>
```

```ruby
class Square < Rectangle
  def height=(value)
    @height = value
    @width = value
  end

  def width=(value)
    @height = value
    @width = value
  end
end
```

```
class Square < Rectangle
  def height=(value)
    @height = value
    @width = value
  end

  def width=(value)
    @height = value
    @width = value
  end
end
```

```
s = Square.new
s.height = 111
s.width = 333
#s -> <Square:... @height=333, @width=333>
```

Geometry

A Square **is** a Rectangle

Behaviorally

A Square **is not** a Rectangle

How to do it better?

```ruby
class Rectangle < Struct.new(:height, :width)
end
r = Rectangle.new
r.height = 111
r.width = 333
#r -> <struct Rectangle height=111, width=333>
```

```ruby
class Square < Struct.new(:length)
end
s = Square.new
s.length = 333
#s -> <struct Square length=333>
```

https://www.flickr.com/photos/doug88888

```
class Bird
  def fly
    #...
  end
end
```

```ruby
class Parrot < Bird
  def fly
    #...
  end
  def speak
    #...
  end
end
```

```ruby
class Pengiun < Bird
  def fly
    raise NotImplementedError
  end
end
```

Penguins - BBC

http://www.youtube.com/watch?v=9dfWzp7rYR4

```ruby
birds = [Parrot.new, Pengiun.new]
birds.each do |bird|
  #...
  bird.fly
  #...
end
```

```ruby
birds = [Parrot.new, Pengiun.new]
birds.each do |bird|
  #...
  bird.fly
  #...
end
```

It raise

← **NotImplementedError** for a **Penguin**

How to do it better?

```ruby
class Bird
  def eat
    #...
  end
end
```

```ruby
class FlyingBird < Bird
  def fly
    #...
  end
end
```

```ruby
class Bird
  def eat
    #...
  end
end
```

```ruby
class FlyingBird < Bird
  def fly
    #...
  end
end
```

```ruby
class Parrot < FlyingBird
  def eat
    #...
  end

  def fly
    #...
  end

  def speak
    #...
  end
end
```

```ruby
class Parrot < FlyingBird
  def eat
    #...
  end

  def fly
    #...
  end

  def speak
    #...
  end
end
```

```
class Pengiun < Bird
  def eat
    #...
  end
end
```

```ruby
class Pengiun < Bird
  def eat
    #...
  end
end
```



```
class Employee
  def calc_pay
    #...
  end
end
```

```ruby
class Employee
  def calc_pay
    #...
  end
end
```

```ruby
class SalariedEmployee < Employee
  def calc_pay
    #...
  end
end
```

```ruby
class HourlyEmployee < Employee
  def initialize
    @time_cards = []
  end

  def calc_pay
    #...
  end
end
```

We need to add a

VolunteerEmployee

How to do it?

```ruby
class VolunteerEmployee < Employee
  def calc_pay
    0
  end
end
```

```
class VolunteerEmployee < Employee
  def calc_pay
    0 ←—— just always return zero
  end
end
```

Does it make any sense **calc_pay** in a VolunteerEmployee?

Return zero

That implying **calc_pay**
is a reasonable

```ruby
avg_salary =
  employees.inject(0.0) do |sum, employee|
    sum += employee.calc_pay
end / employees.length
```

Volunteers will be decrease the average

```
avg_salary =
employees.inject(0.0) do |sum, employee|
  sum += employee.calc_pay
end / employees.length
```

OK, fix it :)

```ruby
class VolunteerEmployee < Employee
  def calc_pay
    raise NotImplementedError
  end
end
```

```
total = 0.0
employees.each do |employee|
  total += employee.calc_pay #oops :)
  #..
end
```

```ruby
total = 0.0
employees.each do |employee|
  total += employee.calc_pay #oops :)
  #..
end
```

VolunteerEmployee raise Error.

Who will have to catch it?

OK, fix it :)

```ruby
total = 0.0
employees.each do |employee|
  begin
    total += employee.calc_pay
  rescue NotImplementedError
    #if we ignore it,
    #why we raised it? :)
  end

  #..
end
```

Stop hiding problems.

Let's start by participating the 25, 26 and 27 of September in **Puliamo il Mondo.**

Live sustainably.

Puliamo il Mondo

LEGAMBIENTE

Sweep under the rug

OK, fix it :)

What principle is violated?

```
total = 0.0
employees.each do |employee|
  next if employee.class == VolunteerEmployee

  total += employee.calc_pay
  #..
end
```

What principle is violated?

```
total = 0.0
employees.each do |employee|
  next if employee.class == VolunteerEmployee

  total += employee.calc_pay
  #..
end
```

What principle is violated?

```
total = 0.0
employees.each do |employee|
  next if employee.class == VolunteerEmployee

  total += employee.calc_pay
  #..
end
```

It violates the **Open Closed Principle**

How to do it better?

```ruby
class Employee
  #...
end

class EmployeePayable
  def calc_pay
    #...
  end
end

class SalariedEmployee < EmployeePayable
  def calc_pay
    #...
  end
end
```

```ruby
class HourlyEmployee < EmployeePayable
  def initialize
    @time_cards = []
  end

  def calc_pay
    #...
  end
end

class VolunteerEmployee < Employee
  #...
end
```

Sometimes inheritance just isn't the right thing to do.

Alternatives to inheritance

- Delegation
- Composition
- Agregation

Test your knowledge

Is it violation of LSP?

Is it violation of LSP?

Next question

Is it violation of LSP?

```ruby
class DoubleArray
  include Enumerable

  def intiialize
    @elements = []
  end

  def push(element)
    @elements << element
    @elements << element
  end

  # ...
end
```

Resource

- **Data abstraction and hierarchy** by *Barbara Liskov*.
- **Object-Oriented Software Construction** by *Bertrand Meyer*.
- **The Liskov Substitution Principle** by *Robert C. Martin*.
- **Design Principles and Patterns** by *Robert C. Martin*.