Open closed principle

Good coding practice in real life

Vladimir Alekseichenko

Agenda

- Related concepts
- Open closed principle
- Practical examples

Related concepts

- Protected variation
- Hide information

David Parnas

http://www.cs.hku.hk/gallery/20090331_Prof_Parnas/

David Parnas

http://www.cs.hku.hk/gallery/20090731_Prof_Parnas/

1972

Hide Information

... each module is then designed to hide such a decision from the others.

http://www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf

Alistair Cockburn

http://www.automatiseringgids.nl/

1996

Protected variation

Identify points of predicted variation and create a stable interface around them.

http://martinfowler.com/ieeeSoftware/protectedVariation.pdf

Bertrand Mayer

http://en.wikipedia.org/wiki/Bertrand_Meyer

1988

Open Closed Principle

Modules should be both open and closed

http://www.butunclebob.com/CleanCoder.UncleBob

1996

- **Open for extension**: this means that the behavior of the module can be extended.
- **Closed for modification**: the source code of such a module is inviolate. No one is allowed to make source code changes to it.

1996

- **Open for extension**: this means that the behavior of the module can be extended.
- **Closed for modification**: the source code of such a module is inviolate. No one is allowed to make source code changes to it.

2013

"no one is allowed..."?

"**I'm not sure why I phrased it that way** 17 years ago. I was young and impressionable back then, a mere 43 years old..."

-- Uncle Bob

http://blog.8thlight.com/uncle-bob/2013/03/08/AnOpenAndClosedCase.html

2003

- **Open for extension**: this means that the behavior of the module can be extended.
- **Closed for modification**: extending the behavior of a module does not result in changes to the source or binary code of the module.

Agile Software Development: Principles, Patterns and Practices

You should be able to change the environment surrounding a module without changing the module itself.

Examples


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

```ruby
class Rectangle
  attr_reader :height, :width
  def initialize(height, width)
    @height = height
    @width = width
  end
end
```

```ruby
class AreaCalculator
  def area(shapes)
    shapes.inject(0.0) do |sum, shape|
      sum += shape.width * shape.height
    end
  end
end
```

```
r_one = Rectangle.new(1, 2)
r_two = Rectangle.new(3, 5)
shapes = [r_one, r_two]
calculator = AreaCalculator.new

calculator.area(shapes) #> 17.0
```

We need add a circle

```ruby
class Circle
  attr_reader :radius
  def initialize(radius)
    @radius = radius
  end
end
```

```ruby
r_one = Rectangle.new(1, 2)
r_two = Rectangle.new(3, 5)
c_one = Circle.new(7)
shapes = [r_one, r_two, c_one]
calculator = AreaCalculator.new

calculator.area(shapes) # oops
```

```ruby
r_one = Rectangle.new(1, 2)
r_two = Rectangle.new(3, 5)
c_one = Circle.new(7)
shapes = [r_one, r_two, c_one]
calculator = AreaCalculator.new

calculator.area(shapes) # oops
```

It does not work

Fix it :)

```ruby
class AreaCalculator
  def area(shapes)
    shapes.inject(0.0) do |sum, shape|
      if shape.class == Rectangle
        sum += shape.width * shape.height
      elsif shape.class == Circle
        shape.radius * shape.radius * Math::PI
      end
    end
  end
end
```

```ruby
class AreaCalculator
  def area(shapes)
    shapes.inject(0.0) do |sum, shape|
      if shape.class == Rectangle
        sum += shape.width * shape.height
      elsif shape.class == Circle
        shape.radius * shape.radius * Math::PI
      end
    end
  end
end
```

Otherwise return nil

How to do it better?

```ruby
class Rectangle
  attr_reader :height, :width
  def initialize(height, width)
    @height = height
    @width = width
  end

  def area
    height * width
  end
end
```

```ruby
class Rectangle
  attr_reader :height, :width
  def initialize(height, width)
    @height = height
    @width = width
  end

  def area
    height * width
  end
end
```

```ruby
class Circle
  attr_reader :radius
  def initialize(radius)
    @radius = radius
  end

  def area
    radius * radius * Math::PI
  end
end
```

```ruby
class AreaCalculator
  def area(shapes)
    shapes.inject(0.0) do |sum, shape|
      sum += shape.area
    end
  end
end
```

```
class AreaCalculator
  def area(shapes)
    shapes.inject(0.0) do |sum, shape|
      sum += shape.area
    end
  end
end
```

```ruby
r_one = Rectangle.new(1, 2)
r_two = Rectangle.new(3, 5)
c_one = Circle.new(7)
shapes = [r_one, r_two, c_one]
calculator = AreaCalculator.new

calculator.area(shapes) #>170.9380
```

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

```ruby
class Product
  attr_reader :name, :category, :price
  def initialize(name, category, price)
    @name = name
    @category = category
    @price = price
  end
end
```

```ruby
class ShoppingCart
  def initialize
    @items = []
  end

  def add_item(item)
    @items << item
  end

  def total_price
    @items.inject(0.0) do |sum, item|
      sum += item.price
    end
  end
end
```

Calculate the price according to category

```ruby
def total_price
  @items.inject(0.0) do |sum, item|
    case item.category.start_with?
    when 'A'
      sum += item.price * 1.2
    when 'B'
      sum += item.price * 2.5
    else
      sum += item.price
    end
  end
end
```

```ruby
def total_price
  @items.inject(0.0) do |sum, item|
    case item.category.start_with?
    when 'A'
      sum += item.price * 1.2
    when 'B'
      sum += item.price * 2.5
    else
      sum += item.price
    end
  end
end
```

How to do it better?

```ruby
class PriceCategoryA
  def is_match?(item)
    item.category.start_with? 'A'
  end

  def calculate(item)
    item.price * 1.2
  end
end
```

```ruby
class PriceCategoryA
  def is_match?(item)
    item.category.start_with? 'A'
  end

  def calculate(item)
    item.price * 1.2
  end
end
```

```ruby
class PriceCategoryB
  def is_match?(item)
    item.category.start_with? 'B'
  end

  def calculate(item)
    item.price * 2.5
  end
end
```

```ruby
class PriceCategoryB
  def is_match?(item)
    item.category.start_with? 'B'
  end

  def calculate(item)
    item.price * 2.5
  end
end
```

```ruby
class PriceCalculator
  def initialize
    @strategies = []
  end
  def add_price(strategy)
    @strategies << strategy
  end

  def calculate(item)
    @strategies.each do |strategy|
      if strategy.is_match?(item)
        return strategy.calculate(item)
      end
    end

    item.price
  end
end
```

```ruby
class PriceCalculator
  def initialize
    @strategies = []
  end
  def add_price(strategy)
    @strategies << strategy
  end

  def calculate(item)
    @strategies.each do |strategy|
      if strategy.is_match?(item)
        return strategy.calculate(item)
      end
    end

    item.price
  end
end
```

```ruby
class ShoppingCart
  def initialize(price_calculator)
    @items = []
    @price_calculator = price_calculator
  end

  def add_item(item)
    @items << item
  end

  def total_price
    @items.inject(0.0) do |sum, item|
      sum += @price_calculator.calculate(item)
    end
  end
end
```

```ruby
class ShoppingCart
  def initialize(price_calculator)
    @items = []
    @price_calculator = price_calculator
  end

  def add_item(item)
    @items << item
  end

  def total_price
    @items.inject(0.0) do |sum, item|
      sum += @price_calculator.calculate(item)
    end
  end
end
```

```ruby
p_one = Product.new('apple', 'A', 20)
p_two = Product.new('pear', 'B', 15)

calculator = PriceCalculator.new
calculator.add_price(PriceCategoryA.new)
calculator.add_price(PriceCategoryB.new)

cart = ShoppingCart.new(calculator)
cart.add_item(p_one)
cart.add_item(p_two)
cart.total_price #> 61.5
```



```ruby
class Item
  attr_reader :value
  def initialize(value)
    @value = value
  end
end
```

```ruby
class SearchSetting
  attr_reader :value, :type
  def initialize(value, type)
    @value = value
    @type = type
  end
end
```

```
module MatchType
    EQUALS          = 1
    GREATER_THAN    = 2
    LESSER_THAN     = 4
end
```

```ruby
class Search
  def initialize(item)
    @item = item
  end

  def condition(value)
    #...
  end

  def operator_description
    #...
  end
end
```

```ruby
def condition(value)
  case @item.type
  when MatchType::EQUALS
    return @item.value == value
  when MatchType::GREATER_THAN
    return @item.value > value
  when MatchType::LESSER_THAN
    return @item.value < value
  else
    raise ArgumentError, 'invalid type'
  end
end
```

```ruby
def operator_description
  case @item.type
  when MatchType::EQUALS
    return 'equals'
  when MatchType::GREATER_THAN
    return 'greater than'
  when MatchType::LESSER_THAN
    return 'lesser than'
  else
    raise ArgumentError, 'invalid type'
  end
end
```

```ruby
items = [Item.new(1), Item.new(10), Item.new(11)]

setting = SearchSetting.new(10, MatchType::EQUALS)
search = Search.new(setting)

finded_items = items.select do |item|
  search.condition(item.value)
end
finded_items.to_s #> [#<Item:0x... @value=10>]
```

```ruby
def condition(value)
  case @item.type
  when MatchType::EQUALS
    return @item.value == value
  when MatchType::GREATER_THAN
    return @item.value > value
  when MatchType::LESSER_THAN
    return @item.value < value
  else
    raise ArgumentError, 'invalid type'
  end
end
```

```ruby
def operator_description
  case @item.type
  when MatchType::EQUALS
    return 'equals'
  when MatchType::GREATER_THAN
    return 'greater than'
  when MatchType::LESSER_THAN
    return 'lesser than'
  else
    raise ArgumentError, 'invalid type'
  end
end
```

How to do it better?

```ruby
class EqualsOperator
  def condition(first_value, second_value)
    first_value == second_value
  end

  def description
    'equals'
  end
end
```

```ruby
class GreaterThanOperator
  def condition(first_value, second_value)
    first_value > second_value
  end

  def description
    'greater than'
  end
end
```

```ruby
class LesserThanOperator
  def condition(first_value, second_value)
    first_value < second_value
  end

  def description
    'lesser than'
  end
end
```

```ruby
class SearchSetting
  attr_reader :value, :operator
  def initialize(value, operator)
    @value = value
    @operator = operator
  end
end
```

```ruby
class Search
  def initialize(setting)
    @setting = setting
  end
  def first_value
    @setting.value
  end

  def operator
    @setting.operator
  end

  def condition(second_value)
    operator.condition(first_value, second_value)
  end
end
```

```ruby
items = [Item.new(1), Item.new(10), Item.new(11)]

setting = SearchSetting.new(10, EqualsOperator.new)
search = Search.new(setting)

finded_items = items.select do |item|
  search.condition(item.value)
end
finded_items.to_s #> [#<Item:0x... @value=10>]
```

```ruby
items = [Item.new(1), Item.new(10), Item.new(11)]

setting = SearchSetting.new(10, EqualsOperator.new)
search = Search.new(setting)
```

```ruby
finded_items = items.select do |item|
  search.condition(item.value)
end
finded_items.to_s #> [#<Item:0x... @value=10>]
```

Books

- **Object Oriented Software Construction** by *Bertrand Meyer*.
- **Object Oriented Software Engineering: A Use Case Driven Approach** by *Ivar Jacobson*.
- **Agile Software Development, Principles, Patterns, and Practices** by *Robert C. Martin*.
- **Clean Code: A Handbook of Agile Software Craftsmanship** by *Robert C. Martin*.

Paper/Article

- **Protected Variation: The Importance of Being Closed** by *Craig Larman*.
- **The Open-Closed Principle, in review** by *Jon Skeet*.
- **An Open and Closed Case** by *Robert C. Martin*.
- **The Open-Closed Principle** by *Robert C. Martin*.