

Design Patterns in Ruby

→ Pisano



What's a design pattern ?



Who Invented Patterns ?



Why Should I Learn Patterns ?



Criticism of patterns



Classification of patterns

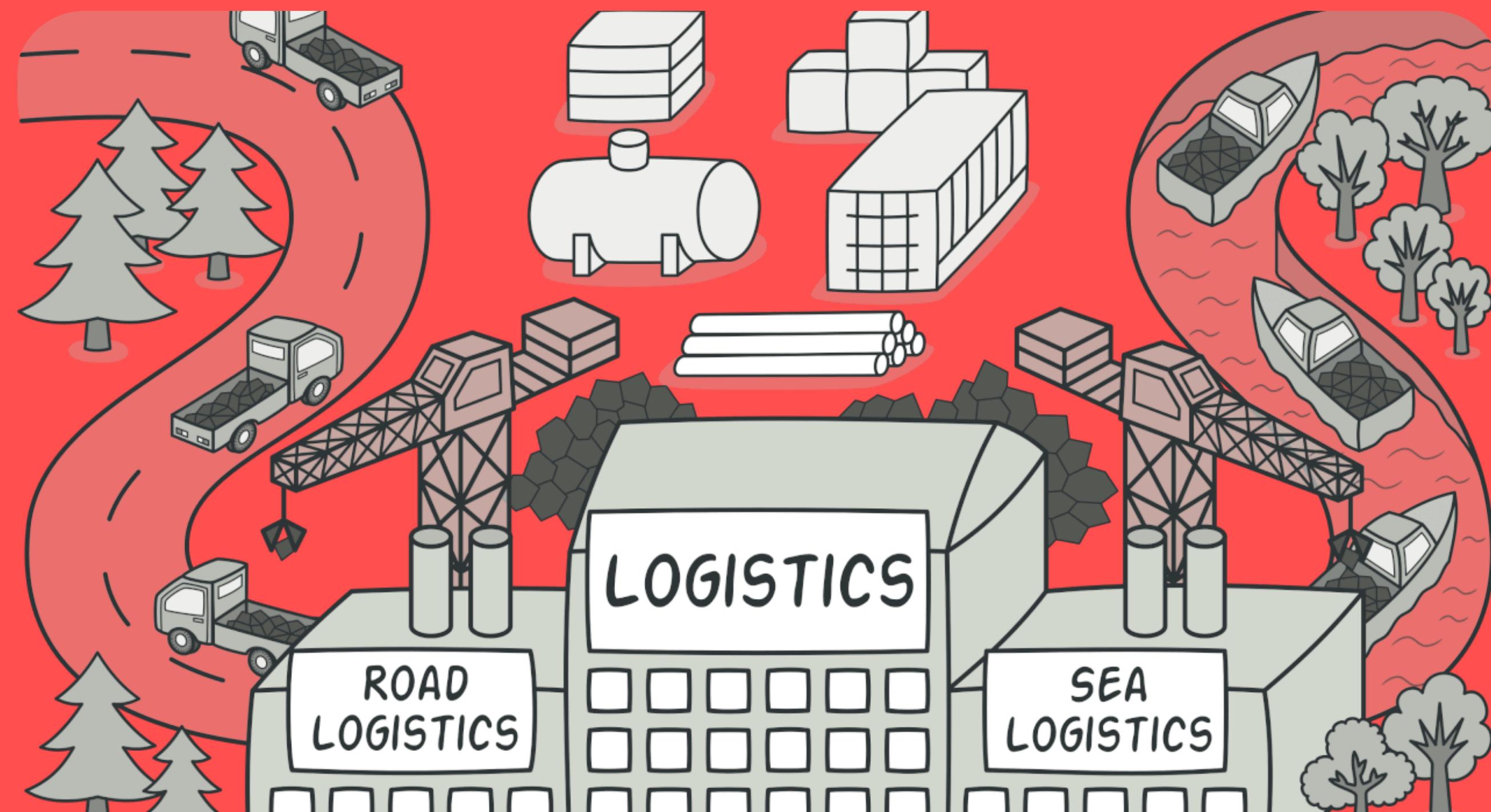


Creational Patterns

- 
- 1 Factory Method
 - 2 Abstract Factory
 - 3 Builder
 - 4 Prototype
 - 5 Singleton

Creational Patterns

Factory Method



Factory Method

```
1 class Creator
2   def factory_method
3     raise NotImplementedError, "#{self.class} has not implemented method '#{__method__}'"
4   end
5   def some_operation
6     product = factory_method
7     result = "Creator: The same creator's code has just worked with #{product.operation}"
8     result
9   end
10 end
11
12 class ConcreteCreator1 < Creator
13   def factory_method
14     ConcreteProduct1.new
15   end
16 end
17
18 class ConcreteCreator2 < Creator
19   # @return [ConcreteProduct2]
20   def factory_method
21     ConcreteProduct2.new
22   end
23 end
24
25 class Product
26   # return [String]
27   def operation
28     raise NotImplementedError, "#{self.class} has not implemented method '#{__method__}'"
29   end
30 end
31
32 class ConcreteProduct1 < Product
33   # @return [String]
34   def operation
35     '{Result of the ConcreteProduct1}'
36   end
37 end
38
39 class ConcreteProduct2 < Product
40   # @return [String]
41   def operation
42     '{Result of the ConcreteProduct2}'
43   end
44 end
45
46 def client_code(creator)
47   print "Client: I'm not aware of the creator's class, but it still works.\n"
48   "#{creator.some_operation}"
49 end
50
51 puts 'App: Launched with the ConcreteCreator1.'
52 client_code(ConcreteCreator1.new)
53 puts "\n\n"
54
55 puts 'App: Launched with the ConcreteCreator2.'
56 client_code(ConcreteCreator2.new)
```

Factory Method

Abstract Factory

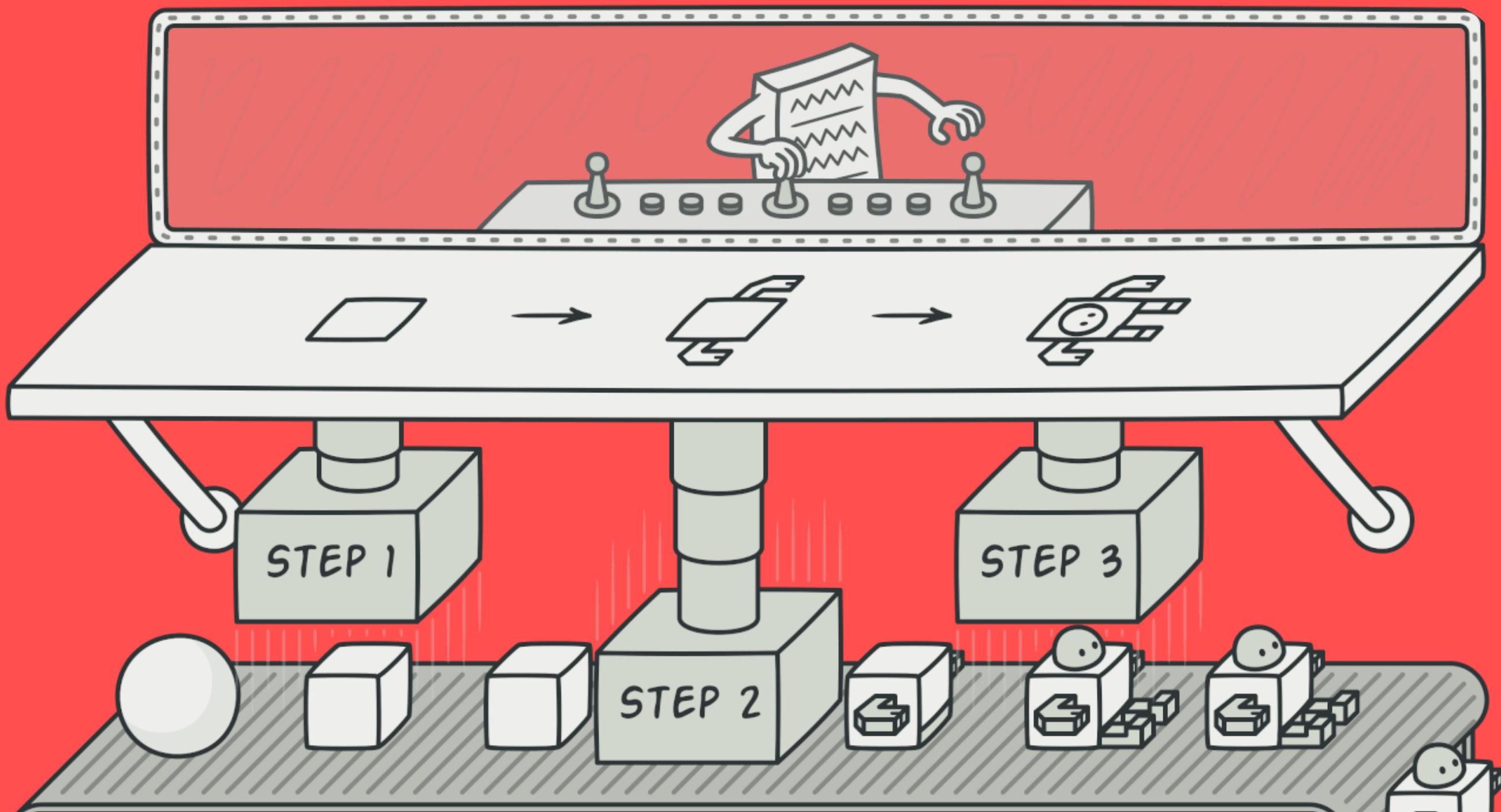


Abstract Factory Method

```
1 class AbstractFactory
2   def create_product_a
3     raise NotImplementedError, "#{self.class} has not implemented method '#{__method__}'"
4   end
5
6   def create_product_b
7     raise NotImplementedError, "#{self.class} has not implemented method '#{__method__}'"
8   end
9 end
10
11 class ConcreteFactory1 < AbstractFactory
12   def create_product_a
13     ConcreteProductA1.new
14   end
15
16   def create_product_b
17     ConcreteProductB1.new
18   end
19 end
20
21 class ConcreteFactory2 < AbstractFactory
22   def create_product_a
23     ConcreteProductA2.new
24   end
25
26   def create_product_b
27     ConcreteProductB2.new
28   end
29 end
30
31 class AbstractProductA
32   def useful_function_a
33     raise NotImplementedError, "#{self.class} has not implemented method '#{__method__}'"
34   end
35 end
36
37 class ConcreteProductA1 < AbstractProductA
38   def useful_function_a
39     'The result of the product A1.'
40   end
41 end
42
43 class ConcreteProductA2 < AbstractProductA
44   def useful_function_a
45     'The result of the product A2.'
46   end
47 end
48
49 class AbstractProductB
50   def useful_function_b
51     raise NotImplementedError, "#{self.class} has not implemented method '#{__method__}'"
52   end
```

Abstract Factory Method

Builder



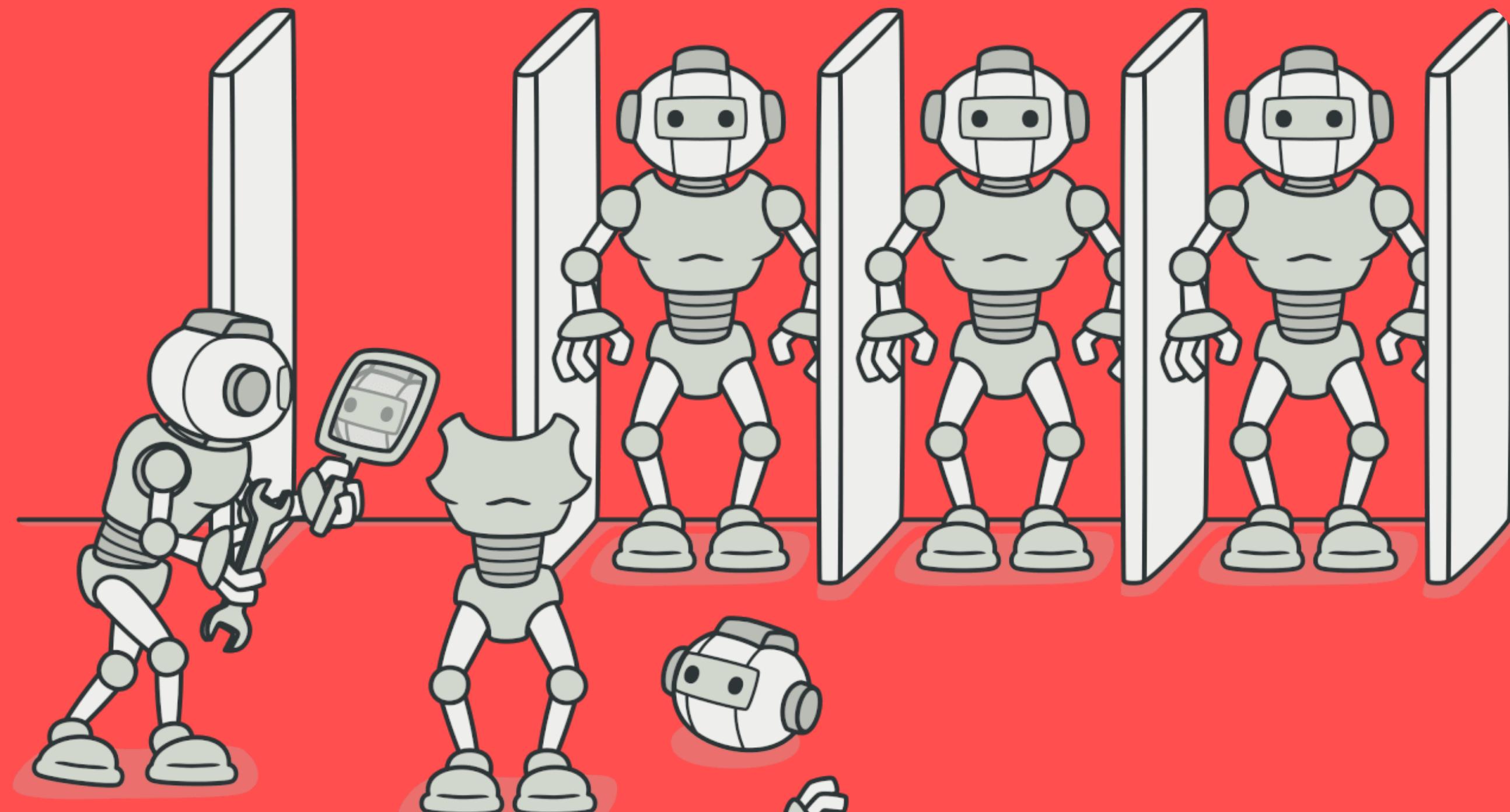


Builder

```
1 class Builder
2   # @abstract
3   def produce_part_a
4     raise NotImplementedError, "#{self.class} has not implemented method '#{__method__}'"
5   end
6
7   # @abstract
8   def produce_part_b
9     raise NotImplementedError, "#{self.class} has not implemented method '#{__method__}'"
10  end
11
12  # @abstract
13  def produce_part_c
14    raise NotImplementedError, "#{self.class} has not implemented method '#{__method__}'"
15  end
16 end
17
18 class ConcreteBuilder1 < Builder
19   def initialize
20     reset
21   end
22
23   def reset
24     @product = Product1.new
25   end
26
27   def product
28     product = @product
29     reset
30     product
31   end
32
33   def produce_part_a
34     @product.add('PartA1')
35   end
36
37   def produce_part_b
38     @product.add('PartB1')
39   end
40
41   def produce_part_c
42     @product.add('PartC1')
43   end
44 end
45
46 class Product1
47   def initialize
48     @parts = []
49   end
50
51   # @param [String] part
52   def add(part)
```

Builder in Ruby

Prototype



Prototype

```
Prototype

1 class Prototype
2   attr_accessor :primitive, :component, :circular_reference
3
4   def initialize
5     @primitive = nil
6     @component = nil
7     @circular_reference = nil
8   end
9
10 # @return [Prototype]
11 def clone
12   @component = deep_copy(@component)
13   @circular_reference = deep_copy(@circular_reference)
14   @circular_reference.prototype = self
15   deep_copy(self)
16 end
17
18 private def deep_copy(object)
19   Marshal.load(Marshal.dump(object))
20 end
21 end
22
23 class ComponentWithBackReference
24   attr_accessor :prototype
25
26   # @param [Prototype] prototype
27   def initialize(prototype)
28     @prototype = prototype
29   end
30 end
31
32 # The client code.
33 p1 = Prototype.new
34 p1.primitive = 245
35 p1.component = Time.now
36 p1.circular_reference = ComponentWithBackReference.new(p1)
37
38 p2 = p1.clone
39
40 if p1.primitive == p2.primitive
41   puts 'Primitive field values have been carried over to a clone. Yay!'
42 else
43   puts 'Primitive field values have not been copied. Booo!'
44 end
45
46 if p1.component.equal?(p2.component)
47   puts 'Simple component has not been cloned. Booo!'
48 else
49   puts 'Simple component has been cloned. Yay!'
50 end
51
52 if p1.circular_reference.equal?(p2.circular_reference)
53   puts 'Component with back reference has not been cloned. Booo!'
54 else
55   puts 'Component with back reference has been cloned. Yay!'
56 end
57
58 if p1.circular_reference.prototype.equal?(p2.circular_reference.prototype)
59   print 'Component with back reference is linked to original object. Booo!'
60 else
61   print 'Component with back reference is linked to the clone. Yay!'
62 end
```

Singleton





Singleton

```
1 class Singleton
2   attr_reader :value
3
4   @instance_mutex = Mutex.new
5
6   private_class_method :new
7
8   def initialize(value)
9     @value = value
10  end
11
12  def self.instance(value)
13    return @instance if @instance
14
15    @instance_mutex.synchronize do
16      @instance ||= new(value)
17    end
18
19    @instance
20  end
21
22  def some_business_logic
23    # ...
24  end
25 end
26
27 def test_singleton(value)
28   singleton = Singleton.instance(value)
29   puts singleton.value
30 end
31
32 puts "If you see the same value, then singleton was reused (yay!)\n"
33 puts "If you see different values, then 2 singletons were created (booo!!)\n\n"
34 puts "RESULT:\n\n"
35
36 process1 = Thread.new { test_singleton('FOO') }
37 process2 = Thread.new { test_singleton('BAR') }
38 process1.join
39 process2.join
```

Singleton

Thanks