A Ruby nyelv

Készítette: Erdélyi Zsolt

A Ruby nyelv

- Minden objektum
- Minden objektum rendelkezik egy egyedi azonosítóval
- Az objektumoknak definiálhatunk példányváltozókat (példányonként egyedi érték)
- Az osztályokban definiálhatunk példány metódusokat

Metódushívás

- az objektumnak küldött üzenetküldés segítségével valósul meg
- az üzenet tartalmazza a metódus nevét és a szükséges paramétereket
- üzenetküldés egy konkrét objektumnak:
- az objektum kikeresi a saját osztályából a megfelelő metódust
- ha megvan, akkor végrehajtja
- ha nincs, akkor NoMethodError
- első ránézésre bonyolultnak tűnhet, de a gyakorlatban nagyon természetesen használható

Eltérés más nyelvekhez viszonyítva

- number = Math.abs(number) // Java
- number = number.abs # Ruby

- strlen(name) // C
- name.length # Ruby

```ruby
### Ruby alapok:

def say_goodnight(name)
  result = "Good night, " + name
  return result
end
puts say_goodnight("John-Boy")   # => Good night, John-Boy

def say_goodnight name
  "Good night, #{name}"
end
puts say_goodnight("Mary-Ellen") # => Good night, Mary-Ellen

puts "And good night,\nGrandma"
# => And good night,
#    Grandma

p "And good night,\nGrandma" # => "And good night,\nGrandma"
```

```ruby
### Fixnum

5.times { puts "Hello\n" }
# => Hello
#    Hello
#    Hello
#    Hello
#    Hello

p 5.to_s # => "5"


### String, Symbol

string_1 = 'simple string object'
string_2 = 'simple string object'
puts string_1.object_id == string_2.object_id # => false

symbol_1 = :simple_symbol_object
symbol_2 = :simple_symbol_object
puts symbol_1.object_id == symbol_2.object_id # => true
```

```ruby
### Range

range = (0..10)
p range.to_a # => [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]

# NilClass
#   - nil (null)
#     - Míg más nyelvekben a jelentése az, hogy nem objektum, addig Ruby-ban a nil is egy objektum.

p nil.class # => NilClass
p nil.nil?  # => true


# Nincs Boolean osztály
# TrueClass
#   true

p true.class # => TrueClass

# FalseClass
#   false

p false.class # => FalseClass
```

```ruby
### Array

array = [ 1, 'cat', nil, 3.14 ]
puts array[0]    # => 1
puts array.first # => 1

### Hash
# - asszociatív tömb

hash = {
   :egy    => 1,
   :ketto => 2,
   :harom => 3
}
puts hash[:ketto] # => 2
```

```ruby
hash = {
   egy:    1,
   ketto: 2,
   harom: 3
}
puts hash[:ketto] # => 2
```

```ruby
### Regexp

p    %r{\d\d\d\d:\d\d:\d\d}.match('2014:03:27') # => #<MatchData "2014:03:27">
puts %r{\d\d\d\d:\d\d:\d\d}.match('2014:03:27') # => 2014:03:27

p    '2014:03:27'.match(/\d{4}:\d{2}:[0-9]{2}/) # => #<MatchData "2014:03:27">
puts '2014:03:27'.match(/\d{4}:\d{2}:[0-9]{2}/) # => 2014:03:27

p    '2014:03:27'.match(/(\d{4}):\d{2}:(\d{2})/) # => #<MatchData "2014:03:27" 1:"2014" 2:"27">
puts '2014:03:27'.match(/(\d{4}):\d{2}:(\d{2})/) # => 2014:03:27

p $1 # => "2014"
p $2 # => "27"
```

```ruby
# Elágazás:

def test_if_statement number
  current_number = number.to_i
  if current_number > 10
    puts 'Greater than 10.'
  elsif current_number < 3
    puts 'Less than 3.'
  else
    puts 'Between 3 and 10.'
  end
  puts 'It is 7.' if current_number == 7
end
test_if_statement 11 # => Greater than 10.
test_if_statement 2  # => Less than 3.
test_if_statement 7  # => Between 3 and 10.
                     #    It is 7.

def test_unless_statement maybe_nil
  puts 'It is not nil.' unless maybe_nil.nil?
end
puts test_unless_statement nil                # =>
puts test_unless_statement 'a string object' # => It is not nil.
```

```ruby
# Ciklus:

i = 0
while i < 3
  puts i
  i += 1
end
# => 0
#    1
#    2

# Egy kicsit "Ruby"-sabban:

(0..2).each do |i|
  puts i
end
# => 0
#    1
#    2
```

```ruby
i = 0
until i == 3
  puts i
  i += 1
end
# => 0
#    1
#    2
```

```ruby
### Paraméterezés

def example_method arg1, arg2, arg3={}
  m = arg3[:method] || :+
  arg1.send(m, arg2)
end
# example_method                        # => ArgumentError
p example_method 2, 3                   # => 5
p example_method 2, 3, :method => :*    # => 6

def example_method_2 *args
  args
end
p example_method_2          # => []
p example_method_2 1, 2, 3  # => [ 1, 2, 3 ]
```

Blokkok:

```ruby
def call_block
  puts "Start of method"
  if block_given?
    yield
    yield
  end
  puts "End of method"
end
call_block { puts "In the block" }
# => Start of method
#    In the block
#    In the block
#    End of method
```

```ruby
def call_block &block
  puts "Start of method"
  if block_given?
    block.call
    block.call
  end
  puts "End of method"
end
call_block { puts "In the block" }
# => Start of method
#    In the block
#    In the block
#    End of method
```

```ruby
def call_block number
  yield(number)
end
call_block(5) do |number|
  puts number * 2
end
# => 10

def call_block number, &block
  block.call(number)
end
call_block(5) do |number|
  puts number * 2
end
# => 10
```

```ruby
### Paraméterül blokkot is váró metódusok:
# Más nyelvekben a gyűjtemények nem tartalmazzák az iterátorukat, hanem külsö segéd objektumokat használnak.

array = [ 1, 2, 3, 4 ]

new_array = array.map do |item|
  item * 2
end
p new_array # => [ 2, 4, 6, 8 ]

array.each_with_index do |item, index|
  puts "#{index} * #{item} = #{index}"
end
# => 0 * 1 = 0
#    1 * 2 = 1
#    2 * 3 = 2
#    3 * 4 = 3
```

```ruby
hash = {
  one:   1,
  two:   2,
  three: 3
}

hash.each_with_index do |(key, value), index|
  puts "#{index}: #{key} => #{value}"
end
# => 0: one   => 1
#    1: two   => 2
#    2: three => 3

array = [ 1, 2, 3, 4 ]
sum = array.inject(0) do |s, item|
  s + item
end
p sum                    # => 10
p array.inject(&:+) # => 10
```

```ruby
### Objektum-orientált programozás:

class A
  #attr_reader :a
  #attr_writer :a
  attr_accessor :a

  def initialize
    @a = 1
  end

  def reset
    initialize
  end
end

x = A.new
p x.a          # => 1
x.a = 2
p x.a          # => 2
#x.initialize # nem hívható, mert alapértelmezetten privát
x.reset
p x.a          # => 1
```

```ruby
# Osztálymetódusok:

class A
  class << self
    def print_four
      p 4
    end
  end

  def self.print_five
    p 5
  end
end
A.print_four # => 4
A.print_five # => 5
```

```ruby
# Változók:
#   $var  => globális
#   @@var => osztályváltozó
#   @var  => példányváltozó

class A
  @@var = 0

  def var
    @@var
  end
end
a1 = A.new
a2 = A.new
p a1.var # => 0
p a2.var # => 0
```

```ruby
# Származtatás
#   - többszörös öröklődés Ruby-ban nem támogatott
class A
  attr_accessor :one

  def initialize one
    @one = one
  end
end
a = A.new 1
p a.one # => 1

class B < A
  attr_accessor :two

  def initialize numbers={ one: 1, two: 2 }
    super(numbers[:one])
    @two = numbers[:two]
  end
end
b = B.new
p b.one # => 1
p b.two # => 2
```

```ruby
# Mixin
#   - egyszeres öröklődés egyszerűsége
#   - többszörös öröklődés ereje

module C
  def random_number_between_0_and_10
    (rand * 10).to_i
  end
end

class D
  include C # Itt a C modulban definiált metódusok a D osztály példánymetódusai lesznek.
end

p D.new.random_number_between_0_and_10 # => Egy 0 és 10 közötti véletlen szám.

class E
  extend C # Itt a C modulban definiált metódusok az E osztály osztálymetódusai lesznek.
end
p E.random_number_between_0_and_10 # => Egy 0 és 10 közötti véletlen szám.
```

```ruby
# Virtuális attribútumok
#   Bertrand Meyer (Object-Oriented Software Construction ) => Uniform Access Principle
#   => a példányváltozók és a számított értékek közötti különbség elrejtése

class Song
  attr_accessor :duration

  def duration_in_minutes
    @duration / 60.0
  end

  def duration_in_minutes= new_duration
    @duration = (new_duration * 60).to_i
  end
end
s = Song.new
s.duration = 270
p s.duration            # => 270
p s.duration_in_minutes # => 4.5
```

```ruby
# Kivételkezelés:
begin
  # A folyamat.
rescue
  # Itt valósítható meg a kivételkezelés.
else
  # Ez a rész akkor kerül végrehajtásra, ha a folyamat futása nem váltott ki kivételt.
ensure
  # Ez a rész mindenképp lefut.
end
```

```ruby
# Ruby program írásakor a feladat megoldására koncentrálhatunk.
# Nem a nyelvet támogató keretek írásával kell foglalkuznunk.
#
# Ruby-ban nincs a megszokott értelemben vett generikus programozásra lehetőség,
# de ez nem okoz gondot, mivel Ruby-ban amúgy sem szükséges előre megadni a típusokat,
# így gyakorlatilag bármely metódus tekinthető bizonyos értelemben generikusnak,
# másrészt a metaprogramozás hatalmas lehetőségeket rejt magában.
#
# Metaprogramming, reflections
#
# Operátorok, operátor kifejezések
#   - metódusként valósulnak meg
class Fixnum
  #alias old_plus +
  alias_method :old_plus, :+

  def +(other)
    old_plus(other).succ
  end
end
p 1.old_plus(2) # => 3
p 1 + 2         # => 4
```

```ruby
module MyModule
  MY_CONST = [ 1, 2, 3 ]

  class A
    def initialize
      @a = :a
    end

    { one: 1, two: 2, three: 3 }.each do |method_name, number|
      define_method(:"increase_#{method_name}_times") do |arg|
        number.times { arg = arg.succ }
        arg
      end
    end
  end
end
```

```ruby
p MyModule.constants                    # => [ :MY_CONST, :A ]
p MyModule::MY_CONST                    # => [ 1, 2, 3  ]
a = MyModule.const_get(:A)
p a                                     # => MyModule::A
p a.instance_methods                    # => [ :increase_one_times,
                                        #      :increase_two_times,
                                        #      :increase_three_times,
                                        #      ... ]
p a.new.methods                         # => [ :increase_one_times,
                                        #      :increase_two_times,
                                        #      :increase_three_times,
p a.new.increase_two_times(5)           # => 7
p a.new.class                           # => MyModule::A
p a.new.is_a?(MyModule::A)              # => true
p a.new.instance_variables              # => [ :@a ]
p a.new.instance_variable_get(:@a)      # => :a
```

```ruby
a = MyModule::A.new
p a.respond_to?(:increase_two_times) # => true
m = a.method(:increase_two_times)
p m                                  # => #<Method: MyModule::A#increase_two_times>
p m.receiver                         # => #<MyModule::A:0x000000011c74a0 @a=:a>
p m.name                             # => :increase_two_times
p m.arity                            # => 1

a = MyModule::A.new
p ObjectSpace._id2ref(a.object_id).object_id == a.object_id # => true
```

Kérdések

Mivel pótolja a Ruby nyelv a többszörös öröklődés lehetőségének hiányát?

- Forrás: Programming Ruby (Dave Thomas, Chad Fowler, Andy Hunt)