Programowanie w Ruby

Wykład 4

Marcin Młotkowski

28 października 2016

Plan wykładu

Deklaracja klasy

```
class Device

end
```

Konstruktor

```
class Device
    def initialize(name, type)
        @name = name
        @type = type
    end
end
```

Deklaracja metody

```
class Device
    def initialize(name, type)
        ...
    end

    def to_s
        @name + ' ' + @type
    end
end
```

Utworzenie obiektu

```
pendrive = Device.new('/dev/sdc', 'kingston')
puts pendrive.to_s
```

Dygresja

Konwencja

```
# równoważne: puts pendrive.to_s
puts pendrive
```

Zmienne obiektu (zmienne instancyjne)

Składnia

@nazwa

Zmienne instancyjne są prywatne.

Dostęp do zmiennych obiektu (akcesory)

```
class Device
    def name
        @name
    end

    def zmien_nazwe(nowa)
        @name = nowa
    end
end
```

Inny modyfikator

```
class Device
    def name=(nowa)
        @name = nowa
    end
end
```

pendrive.name = 'Corsair'

Deklaracja akcesorów

Deklaracja

```
class Device
    attr_reader :name
end
```

Równoważne

```
class Device
    def name
        @name
    end
end
```

Modyfikatory

Zamiast pisać tak

```
class Device
    def name=(nowa)
        @name = nowa
    end
end
```

Można napisać tak

```
class Device
    attr_writer :name
end
```

Akcesor odczyt/zapis

```
class Device
    attr_accessor :name
end
```

Atrybuty wirtualne

```
class Device
    attr_reader :name
    def long_name
        @type + '--->' + @name
    end
```

Użycie

```
puts pendrive.name
puts pendrive.long_name
```

Atrybuty wirtualne

Implementacja obiektu reprezentującego temperaturę

```
class Temperatura
    attr_reader :temperatura
end
```

Atrybuty wirtualne

Implementacja obiektu reprezentującego temperaturę

```
class Temperatura
    attr_reader :temperatura
end
```

Skale temperaturowe:

- Celsius
- Kelvin
- Fahrenheit
- Reamur

Skale temperaturowe

```
class Temperatura
    attr_accessor :kelvin
    def celsius=(c)
        @kelvin = 273.15 + c
    end
    def celsius
        return @kelvin - 273.15
    end
    def fahrenheit=(f)
        @kelvin = (f - 32) * (5/9) + 273.15
    end
    def fahrenheit
        return (@kelvin - 273.15)*(9/5) + 32
    end
end
```

Przykłady użycia

```
t = Temperatura.new
t.kelvin = 273.15
puts "Cels #{t.celsius}, fahr #{t.fahrenheit}"
```

```
Cels 0.0, fahr 32.0
```

Metody singletonowe

t = Temperatura.new
t.kelvin = 273.15

Skala Reaumura

```
def t.reaumur; (@kelvin - 273.15)*(5/4) end
```

puts t.reaumur

Zmienne i metody klasy (statyczne)

```
class Device
    @@liczba = 0
    def initialize
        @@liczba = @@liczba + 1
    end
```

Zmienne i metody klasy (statyczne)

```
class Device
    @@liczba = 0
    def initialize
        @@liczba = @@liczba + 1
    end
    def Device.licznik
        @@liczba.to_s
    end
end
```

Kontrola dostępu do pól i metod

- Zmienne klasy i obiektu są prywatne
- Metody klasy i obiektu są publiczne

Metody *protected*

- Można korzystać w podklasach
- Można korzystać w metodach obiektów tej samej klasy

Składnia

```
class Device
    protected
        def scan
        end
    def block
    end
    protected :block
end
```

Metody prywatne

Można je wywoływać wyłącznie w postaci

```
self.metoda_prywatna
```

Deklarowanie podklasy

```
class Printer < Device

end
```

Konstruktor w podklasie

```
class Printer < Device
    def initialize(name, type, port)
        super(name, type)
        @port = port
    end
end
```

Dziedziczenie i przykrywanie metod

Przykład

```
class Device
    def to_s
        ...
    end
end
class Printer < Device
    ...
end

iglowka = Printer.new('biurkowa', 'dot printer', 'PRN:')
puts iglowka # puts iglowka.to_s
```

Wynik

```
'biurkowa dot printer'
```

Ulepszenie metody to_s

Pierwsza wersja

```
class Printer < Device
    def to_s
        super.to_s + '#{@port}'
    end
end
```

Ulepszenie metody to_s

Pierwsza wersja

```
class Printer < Device
    def to_s
        super.to_s + '#{@port}'
    end
end
```

Druga wersja

```
class Printer < Device
    def to_s
        super + '#{@port}'
    end
end
```

Na koniec o metodach

Wszystkie metody są wirtualne.

Plan wykładu

Co to jest mixin

Mixin – mechanizm włączania kodu modułu do deklaracji klasy

Przykład

Zadanie

Zdefinowanie modułu do odczytywania stanu obiektu

Implementacja modułu

```
module Debugger
    def snapshot
        puts "Stan obiektu klasy #{self.class}"
        for iv in self.instance_variables
            puts "#{iv} = #{self.instance_variable_get(iv)}"
        end
    end
end
```

"Wmiksowanie" modułu do klasy

```
class Drukarka < Device
    include Debugger
        ...
end
```

Zastosowanie

```
lokalna = Drukarka.new('kuchenna', 'hp5000N', '/dev/')
lokalna.snapshot
```

Inne zastosowania

Moduł (mixin) Comparable

- implementuje operatory porównania $<$, $<=$, $==$, $>=$, $>$ i metodę between?
- wymaga implementacji operatora $<=>$

Zastosowanie modułu Comparable

```
class Wektor
    include Comparable
    def <=> (aWektor)
        # zwraca -1 gdy self< aWektor, 0 lub 1
    end
end
```

Zastosowanie

```
w1 = Wektor.new([3, -4, 5])
w2 = Wektor.new([-5, 12, -2])

w1 < w1
w1 >= w2
```

Plan wykładu

Wszystkie klasy są podklasami klasy Object

Metody klasy Object

- class - klasa obiektu
- instance_variables
- freeze
- to_s
- i inne...

Chwila rozrywki

```
(irb)> 5.class
Fixnum
```

Chwila rozrywki

```
(irb)> 5.class
Fixnum
(irb)> 5.class.class
Class
```

Chwila rozrywki

```
(irb)> 5.class
Fixnum
(irb)> 5.class.class
Class
(irb)> 5.class.class.class
Class
```

Zmiana implementacji istniejącej klasy

```
class Fixnum
    def to_s
        return "Niespodzianka!"
    end
end
```

```
puts 5
```

Zagnieżdżanie klasy w klasie

```
class A
    class InA
    end
end
```

```
x = A::InA.new
```

Dostęp indeksowany

```
class MyArray
    def [](i)
        return "Zażądano #{i}"
    end
    def []=(i, v)
        puts "[#{i}]= #{v}"
    end
end
```

```
arr = MyArray.new
arr[234]
arr[34] = 16
```