Ruby

Podstawy języka Ruby

Kraków 13-10-2015

Ruby jest prosty z wyglądu, ale bardzo skomplikowany w środku, tak jak ciało ludzkie.

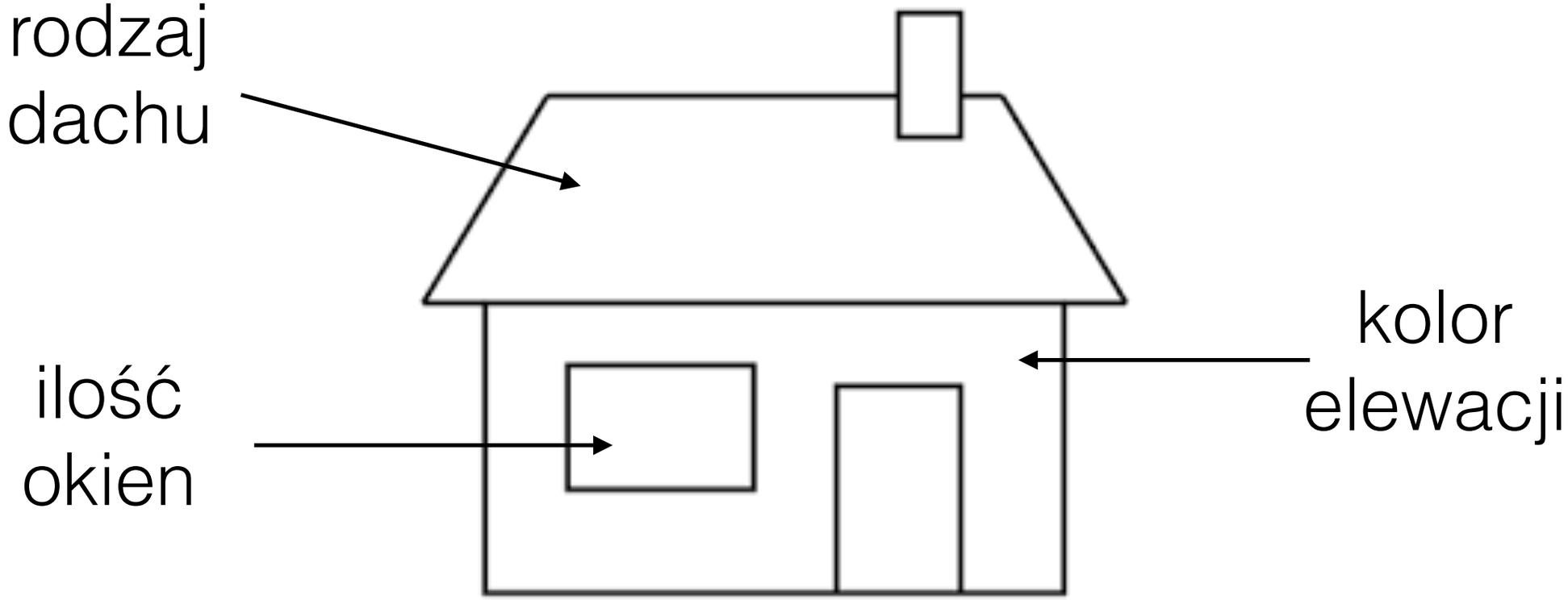
Yukihiro Matsumoto

Cechy

- dynamiczne typowanie - “Duck typing”
- wszystko jest obiektem
- bardzo wysoka elastyczność
- wysoką przenośnością pomiędzy platformami
- w pełni bezpłatny
- opcjonalne nawiasy, znaki końca linii i ‘return’

Obiektowość…

Klasa



Obiekt

spójrzmy za okno :)

Podstawowe klasy

STRING / SYMBOL

String.new(“Ala ma kota”)

“false”

“Ala ma kota”

“Adrian ma alergię”

“Dawid ma żółwia”

:nazwa

“908,87”

:ulica

:kot

“1” + “1” = “11”

:nazwisko

:email

“0987”

:imie

“”

“true”

FIXNUM / BIGNUM / FLOAT

1234567890

6%5 = 1

0.8767

77

0/3 = 0

3/2.0 = 1.5

34.98

3/2.0 = 1.5

3*2.0 = 6.0

1 + 1 = 2

3/0 = ?

3.0 / 2 = 1.5

3/2 = 1

99999999999

ARRAY / HASH

[1,3,4]

{“a” => 1, “b” => 2}

[nil, nil, nil, nil, 5]

[“tekst”, “987”, 987, nil]

{true => “OK”, false => “Błąd”}

[50, nil, true, false]

[“ala”, “ma”,”kota”]

[{ala: “kot”}, “1”, “3”]

{“ala” => “kot”}

{ala: “kot”}

[[:a, :b, :c], “a”, “b”, “c”]

{1 => {“ala” => “kot”}}

{“a” => 1, “a” => 2}

True, False, Nil

TrueClass

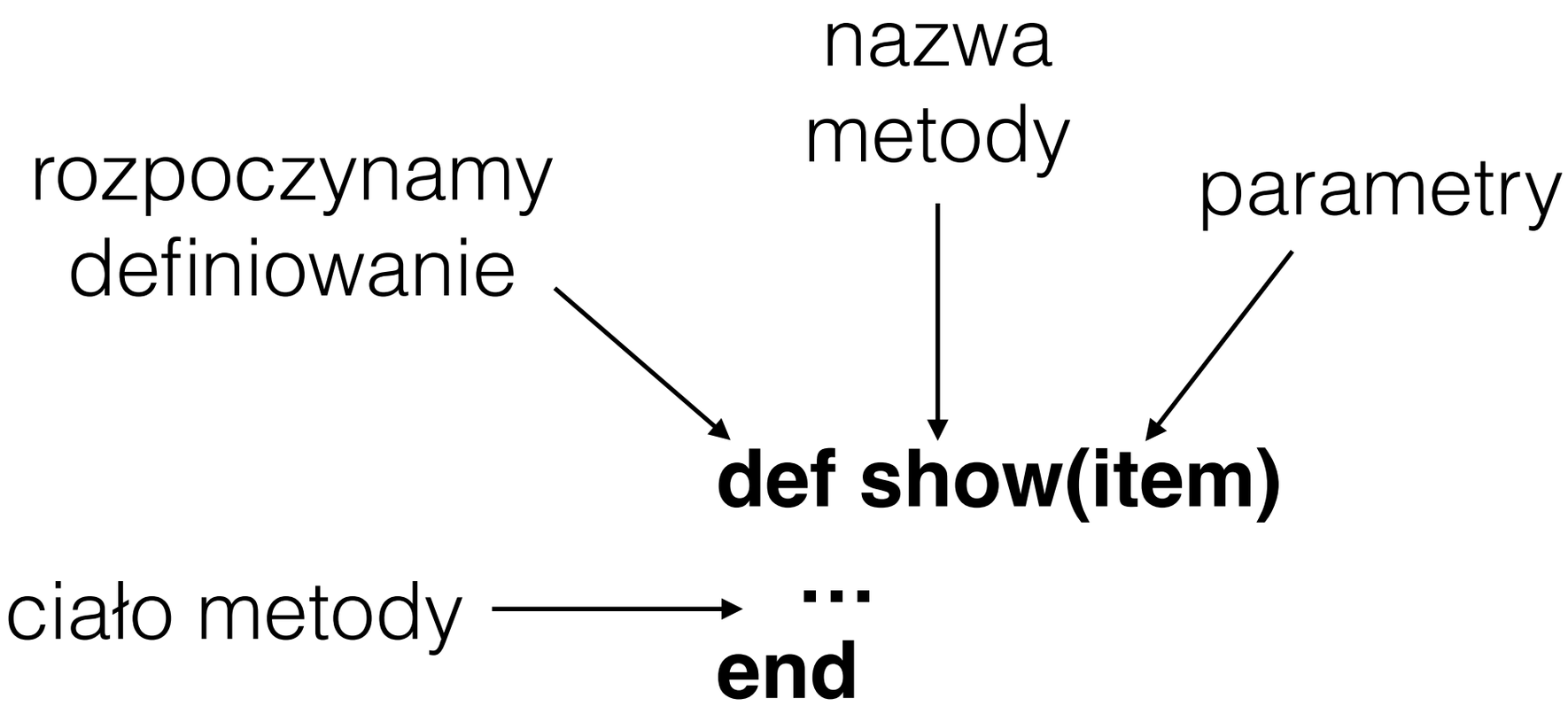
FalseClass

NilClass

Zadania

Tworzenie metod i klas

Metoda


rozpoczynamy definiowanie
nazwa metody
parametry
def show(item)
ciało metody
...
end

Klasa

rozpoczynamy definiowanie

nazwa klasy

po czym dziedziczy

```
class User < SuperUser
  def name
  end
```

melody obiektowe

```
  def self.name
  end
end
```

melody klasowe

Meteda konstruktora, instancyjna i klasowa

```
class User
    def initialize
    end

    def login
    end

    def self.stats
    end
end
```

```
user = User.new()

user.login

User.stats
```

Zmienne i stałe

```
class User
    @@count = 0
    TMP_PASSWD = 123                    User::TMP_PASSWD

    def initialize(name)
      @name = name
    end

    def user_count
        @@count
    end
end
```

Zadania

Flow Control

```
if a > 0
  “Więcej niż zero”
end
```

```
if a < 0
    “Mniej niż zero”
end
```

```
if a == 0
  “Równe zeru”
end
```

```
if a < 0
    “Mniej niż zero”
elsif a > 0
    “Więcej niż zero”
else
  “Równe zeru”
end
```

Flow Control

```
case a
when 0
  “Równa się 0”
when 1
  “Równa się 1”
when 2
  “Równa się 2”
else
  “Jest różne od 0, 1 i 2”
end
```

```
case
when a < 0
    “Mniej niż zero”
when a > 0
    “Więcej niż zero”
else
  “Równe zeru”
end
```

Flow Control

```
[1, 2, 3, 4, 5].each do |element|
    puts "—> #{element}"
end
```

```
('A'..'Z').each do |litera|
    puts "—> #{litera}"
end
```

```
(-10…10).each do |litera|
    puts "—> #{litera}"
end
```

Zadania

Moduły

```
class User
  ...
end
```

```
class User
  include Authorize
  ...
end
```

```
class User
  extend Authorize
  ...
end
```

```
module Authorize
  def auth
    ...
  end
end
```

Wyjątki

```
def read(file_name)
    File.open(file_name)
end
```

```
def divide(a, b)
    begin
        File.open(file_name)
    rescue Errno::ENOENT
        “Plik nie istnieje.”
    end
end
```

Zadania

alicja.cyganiewicz@infakt.pl
krzysztof.hostynski@infakt.pl
adrian.zieba@infakt.pl

inFakt

KONIEC

Adrian Zięba
inFakt