Ruby 初級者向けレッスン

okkez@Ruby関西, ナツマ@小波ゼミ

自己紹介

- okkez
- 読み方は「おっきー」
- 所属はRuby関西
- Ruby歴はたぶん二年くらい

自己紹介

- ナツマ
- 小波ゼミの4回生
- 旅行が好きです
- 主に飛行機に乗るのが目的
- 最近 return 文を覚えました

Rubyistの基礎知識

- クラスとは
- 各種メソッドの使い分け
- アクセス制御
- 各種変数の使い分け
- モジュール

クラスとは

- Rubyではクラスも Classクラスのインスタンスです

クラスとは

```
# new の引数には継承元親クラスを指定します
Book = Class.new{
  attr_reader :isbn, :title, :author, :price
  define_method(:initialize){|isbn,title,author,price|
    @isbn   = isbn
    # 略
  }
  define_method(:spec){
    puts "#{isbn}, #{title}, #{author}, #{price}円"
  }
}
```

クラスとは

```
class Book
    attr_reader :isbn, :title, :author, :price
    def initialize(isbn, title, author, price)
      # 略
    end
    def spec
      puts "#{isbn}, #{title}, #{author}, #{price}円"
    end
end
```

クラスとは

- どちらがわかりやすいかは一目瞭然ですね
- def ～ end を受け付けない体質の友人が居たら前者の書き方を教えてあげてください

で、クラスって何？

- イデア
- 設計書
- あるいは、たい焼きの型

各種メソッド

- インスタンスメソッド
- クラスメソッド
- 関数的メソッド

インスタンスメソッド

- あるオブジェクトをレシーバとするメソッド

インスタンスメソッド

```
class Taiyaki
  attr_reader :created_at, :cream
  def initialize(cream)
    @created_at = Time.now
    @cream = cream
  end
  # 焼き上がってから5分以内なら焼きたて
  def yakitate? # インスタンスメソッド
    (Time.now.to_i - created_at.to_i) <= 300
  end
end
```

インスタンスメソッド

```
# 実行
obj = Taiyaki.new('つぶあん')
sleep(10)
p obj.yakitate? #=> true
sleep(300)
p obj.yakitate? #=> false
```

クラスメソッド

- クラスそのものをレシーバとするメソッド

クラスメソッド

```
class Taiyaki
  @@temperature = 0
  # 型の温度が150度より高ければ準備OK
  def self.prepared? # クラスメソッド
    @@temperature > 150
  end
  def self.heating # クラスメソッド
    @@temperature += 10
  end
end
```

クラスメソッド

```
# 実行
p Taiyaki.prepared? #=> false
16.times{ Taiyaki.heating }
p Taiyaki.prepared? #=> true
```

関数的メソッド

- レシーバがないメソッド

関数的メソッド

- 大抵、 Kernelモジュールの private instance method として定義されている
- Objectクラスが Kernelモジュールを include しているので使えると覚えておけばよい

関数的メソッド

- クラス定義内でレシーバを書かずに、メソッド呼出しできるのはselfというレシーバが省略されているから

アクセス制御

- public
- private
- protected

アクセス制御 public

- メソッドをインスタンスメソッドとして使えるように公開する

アクセス制御 private

- メソッドをクラスの内部だけで使えるようにする
- レシーバを指定して呼び出せないようにする

アクセス制御 protected

- メソッドをクラスの内部から使えるようにする
- 同一クラス内ではインスタンスメソッドとしても使えるようにする

publicメソッドとprivateメソッドの例

```
class AccTest
  def pub_method
    puts 'これはpublicなメソッドです'
  end
  def priv_method
    puts 'これはprivateなメソッドです'
  end
  private :priv_method
end
```

publicメソッドとprivateメソッドの例

```
# 呼び出してみる
obj = AccTest.new
obj.pub_method  #=> これはpublicなメソッドです
obj.priv_method
#=> sample2.rb:13: private method `priv_method' called
#    for #<AccTest:0x2a955a8cd0> (NoMethodError)
```

protectedメソッドの例

```
class Point
  attr_accessor :x, :y    # アクセスメソッドを定義する
  protected :x=, :y=      # x=, y= を protected にする
  def initialize(x=0.0, y=0.0)
    @x = x
    @y = y
  end
```

protectedメソッドの例

```
    def swap(other)
      x_tmp, y_tmp = [@x, @y]
      @x, @y = [other.x, other.y]
      other.x, other.y = [x_tmp, y_tmp]
    end
  end
```

protectedメソッドの例

```
obj1 = Point.new
obj2 = Point.new(1.0, 2.0)
p obj1 #<Point:0x2a955a9bd0 @y=0.0, @x=0.0>
p obj2 #<Point:0x2a955a99a0 @y=2.0, @x=1.0>
obj1.swap(obj2)
p obj1 #<Point:0x2a955a9bd0 @y=2.0, @x=1.0>
p obj2 #<Point:0x2a955a99a0 @y=0.0, @x=0.0>
obj1.x = 10.0
#=> sample3.rb:30: protected method `x=' called for
#   #<Point:0x2a955a9a68 @y=2.0, @x=1.0>
  (NoMethodError)
```

各種変数について

- ローカル変数
- グローバル変数
- クラス変数
- インスタンス変数

各種変数について ローカル変数

- 先頭がアルファベットの小文字か”_”ではじまるもの
- 最もよく使うが、最も有効範囲が狭い

各種変数について　グローバル変数

- 先頭が〝$〟で始まるもの
- 最も有効範囲が広いが、ユーザーが定義して使うのは嫌われる
- 定義済みのものはリファレンスマニュアル参照

各種変数について クラス変数

- 先頭が〝@@〟ではじまるもの
- そのクラスの全てのインスタンスで共有できる変数

各種変数について インスタンス変数

- 先頭が〝@〟ではじまるもの
- クラスを作るとほぼ必ず使う

クラス変数の例

```
class Taiyaki
  @@count = 0
  def initialize
    if @@count > 30
      raise 'たい焼きは30個以上同時に作れません'
    end
    @@count += 1
  end
end
# 実行
31.times{ Taiyaki.new }
#=> RuntimeError: たい焼きは30個以上同時に作れません
```

モジュールとクラスの違い

- クラスは継承することができるがモジュールは継承することができない
- クラスはインスタンス化できるがモジュールはインスタンス化できない

モジュールの使いどころ/使われどころ

- よく似た処理をまとめるのに使われる
- 名前空間を提供する

モジュールの使われどころの例

- Enumerable,Comparableなど
- 自作ライブラリでクラスやモジュールの名前に「ありそうな」名前を付けるとき

```
# 最近はまった例
require 'rubygems'
class Config
end
#=> sample5.rb:3: Config is not a class (TypeError)
# rubygems では module Config が定義されているため
```

まとめ

- RubyではクラスもClassクラスのインスタンス
- 各種メソッドの違いを理解しているとマニュアルも読みやすくなります
- アクセス制御を上手く使ってこそ、上品プログラマ
- 各種変数の違いを理解して適材適所で使いこなそう
- クラスとモジュールの違いを理解して使いこなそう

参考文献

- 初めてのプログラミング
- プログラミングRuby 第2版 言語編
- たのしいRuby 第2版

今後の情報源

- 公式Webサイト
  - http://www.ruby-lang.org/
- リファレンスマニュアル
  - http://www.ruby-lang.org/ja/man/
- 日本Rubyの会
  - http://jp.rubyist.net/
- Rubyist Magazine
  - http://jp.rubyist.net/magazine/