Ruby 初級者向けレッスン（解答）

okkez@Ruby 関西,ナツマ@小波ゼミ
2006年12月16日

耳の遠いおばあちゃん Part1

- grandma.rb

```
class Grandma

  def talk_with_you
    until /\Abye/i =~ @line || @line.nil?
      hear
      speak
    end
    puts 'そうかい。さようなら'
  end

  def hear
    print 'あなたの発言 > '
    @line = STDIN.gets.chomp!
  end

  def speak
    puts @line + ' だって？知らないねー。'
  end

  private :hear, :speak
end

if __FILE__ == $0
  Grandma.new.talk_with_you
end
```

  - 普通のおばあちゃんクラスです。叫ばなくても"bye"とタイプすれば開放してくれます。
  - /\Abye/i は大文字小文字を区別せずにbyeで始まる単語にマッチする正規表現です。
  - untilは継続条件、whileは終了条件と考えるとわかりやすいと思います。

- deaf_grandma.rb

```
require 'grandma'

class DeafGrandma < Grandma

  BASE_YEAR = 1930

  def talk_with_you
    until 'BYE' == @line || @line.nil?
      hear
      speak
    end
    puts 'そうかい。さようなら'
  end
```

```
  def speak
    if @line.upcase == @line
      puts "いやー、#{year}年以来ないねー!!" unless 'BYE' == @line
    else
      puts 'は?! もっと大きな声で話しておくれ、坊や!!'
    end
  end

  def year
    BASE_YEAR + rand(21)
  end
end

if __FILE__ == $0
  DeafGrandma.new.talk_with_you
end
```

○　Grandma クラスを継承して DeafGrandma クラスを作成しています。

耳の遠いおばあちゃん Part2

- lonley_deaf_grandma.rb

```
require 'deaf_grandma'

class LonelyDeafGrandma < DeafGrandma
  attr_reader :name
  @@history = []
  def initialize(name)
    @name = name
    @@history << name
    @bye_count = 0
  end

  def talk_with_you
    if 1 < @@history.size && @@history.size < 4
      puts "この間、#{@@history[0...(@@history.size-1)].join('と')}が話していた子だね。"
    elsif @@history.size >= 4
      puts "ああ、あんたかい。話は聞いてるよ"
    else
      puts "おや、こんにちは。"
    end
    until 3 <= @bye_count
      hear
      speak
    end
    puts 'そうかい、さようなら。'
  end

  def speak
    if 'BYE' == @line
      @bye_count += 1
      super if @bye_count < 3
```

```
    else
      super
    end
  end

  def self.show_history
    puts "あなたは、#{@@history.join('、')}と話しました。"
  end
end

if __FILE__ == $0
  gm1 = LonelyDeafGrandma.new('とめ')
  gm1.talk_with_you
  gm2 = LonelyDeafGrandma.new('うめ')
  gm2.talk_with_you
  gm3 = LonelyDeafGrandma.new('キャシー')
  gm3.talk_with_you
  LonelyDeafGrandma.show_history
end
```

- ○ DeafGrandma を継承して LonleyDeafGrandma を作成しています。
- ○ クラス変数を使って、複数インスタンス間で情報を共有しています。
- ○ talk_with_you メソッド内の until 文の継続条件に@line.nil?がないのは、初回実行時は必ず、@count==0 だからです。

耳の遠いおばあちゃん Part3

- deaf_grandma2.rb

```
require 'deaf_grandma'
module HearingAid
  def self.included(klass)
    klass.instance_eval{|obj|
      alias_method :talk_with_you_orig, :talk_with_you
      alias_method :speak_orig, :speak
      define_method(:talk_with_you){
        hear
        speak
        until /\Abye/i =~ @line
          hear
          speak
        end
        puts 'そう。さようなら'
      }
      define_method(:speak){
        puts "いやー、#{year}年以来ないねー!!!" unless /\Abye/i =~ @line
      }
      private :speak
    }
  end
end

class DeafGrandma
  include HearingAid
```

```
end

if __FILE__ == $0
  DeafGrandma.new.talk_with_you
end
```

- ○ DeafGrandma クラスに HearingAid モジュールをインクルードして既存のメソッドを上書きしています。
- ○ Ruby ではメソッドを探索する順序が、そのメソッドが呼ばれたクラス→インクルードしているモジュール→スーパークラス→スーパークラスでインクルードしているモジュールという順番なので、今回の場合通常のモジュールでは上手くいきません。
- ○ シンプルなのはサブクラスを定義して、メソッドを再定義してやる方法です。

ローマ数字 Part1

- numeric.rb

```
# ローマ数字に変換
class Numeric
  OLD_ROMAN_UNIT = [1000, 500, 100, 50, 10, 5, 1].zip(%w[M D C L X V I])
  def to_old_roman
    number = self
    OLD_ROMAN_UNIT.inject('') do |r, elm|
      digit, number = number.divmod(elm[0])
      r + elm[1] * digit
    end
  end
end

1.step(1000) do |n|
 p [n, n.to_old_roman]
end
```

- ○ Array#zip と Array#inject を上手く使った例。
- ○ 数値と文字列の対応表を作っておいて、大きい数字から順番に変換していく。

- numeric.rb（別解）

```
class Numeric

  alias to_old_roman_orig to_old_roman

  OLD_ROMAN_UNIT2 = {1000 => 'M', 500 => 'D', 100 => 'C',
                     50 => 'L', 10 => 'X',  5 => 'V', 1 => 'I'}

  def to_old_roman

    convert_num2roman(self, [1000, 500, 100, 50, 10, 5, 1])

  end

  private

  def convert_num2roman(n, m, str = '')

    return str if n == 0

    key = m.shift

    digit, n = n.divmod(key)
```

```
    convert_num2roman(n, m, str<<(OLD_ROMAN_UNIT2[key] * digit))
  end
end
```

- 再帰で書いてみました。
- Array#inject 版と再帰版の結果を比較するには、それぞれの結果を一度配列にいれてから Array#- で差を取ってみるといいでしょう。

```
arr1 = []; arr2 = []
1.step(100){|n| arr1 << n.to_old_roman } # 再帰版
1.step(100){|n| arr2 << n.to_old_roman_orig } # Array#inject 版
p arr1 - arr2 #=> []
```

ローマ数字 Part2

- numeric.rb （Tahnks 南木さん）

```
class Numeric
 ROMAN_UNIT = [1000,900,500,400,100,90,50,40,10,9,5,4,1].zip(%w(M CM D CD C XC L XL X IX V IV I))
 def to_roman
   number = self
   ROMAN_UNIT.inject("") do |r, elm|
     digit, number = number.divmod(elm[0])
     r + elm[1] * digit
   end
 end
end

1.step(1000) do |n|
 p [n, n.to_roman]
end
```