

Ruby 講義

第11回 繙承、Module

五十嵐邦明

twitter : igaiga555

<http://www.facebook.com/kuniaki.igarashi>



2013.6.13 at 一橋大学
ニフティ株式会社寄附講義
社会科学における情報技術と
コンテンツ作成III

五十嵐邦明 講師 株式会社万葉



twitter: igaiga555

<https://github.com/igaiga/>

<http://www.facebook.com/kuniaki.igarashi>

濱崎 健吾

Teaching Assistant
fluxflex, inc(米国法人)



twitter: hmsk

<https://github.com/hmsk/>

<http://www.facebook.com/hamachang>

先週の復習

★TODO

ここから今週の内容

A close-up photograph of a woman with blonde hair and green eyes, smiling warmly at the camera. She is wearing a light-colored top and a necklace. She is holding a baby in her arms. The baby has dark hair and is looking slightly to the side with a neutral expression. The background is blurred, showing some greenery.

目次

継承
Module

<http://www.flickr.com/photos/7378221@N03/672843312/>

13年5月15日水曜日

継承

既に定義されているクラスを拡張して新しいクラスを作ることを継承といいます。

(既にあるたい焼きの型を利用して、少し違う新しいたい焼きの型をつくるようなものです。)

継承

たとえばBookクラスとMagazineクラス
(雑誌)を作るとします。

```
class Book
  attr_accessor :title, :price
end
```

```
class Magazie
  attr_accessor :title, :price, :number
end
```

別々に定義を書いてもいいのですが、共通項
もたくさんあります。

継承

たとえばBookクラスとMagazineクラス
(雑誌)を作るとします。

```
class Book
  attr_accessor :title, :price
end
```

```
class Magazie
  attr_accessor :title, :price, :number
end
```

別々に定義を書いてもいいのですが、共通項
もたくさんあります。

継承

そんなときは、継承を使うとすっきり書けます。

継承を使った書き方

```
class Book  
  attr_accessor :title, :price  
end
```

```
class Magazine < Book  
  attr_accessor :number  
end
```

継承を使わない書き方

```
class Book  
  attr_accessor :title, :price  
end
```

```
class Magazine  
  attr_accessor :title, :price, :number  
end
```



`class Magazine < Book` と書くことで、Bookクラスを継承した Magazineクラスを定義できます。MagazineクラスはBookクラスの性質を受け継ぎます。何を受け継ぐかを次のページで解説します。

継承

継承する場合の書式

```
class クラス名 < スーパークラス名  
クラスの定義  
end
```

スーパークラスとは、継承元の
クラス(親クラス)です。

継承したクラスは、親クラスの全てのインスタンス変数、メソッドなどを受け継ぎます。

```
class Book  
attr_accessor :title, :price  
end
```

```
class Magazine < Book  
attr_accessor :number  
end
```

例えばBookクラスを継承した
Magazineクラスは、titleとpriceを
受け継いでいます。例えば、↓のような
コードを書くことができます。

```
magazine = Magazine.new  
magazine.title = "CanCam"  
p magazine.title #=> "CanCam"
```

継承演習問題a1

右のBook クラスを継承した
DigitalBook(電子書籍) ク
ラスを作ってください。

DigitalBookクラスに右のよ
うなfilenameメソッドを実
装してください。

DigitalBookクラスのインス
タンスオブジェクトを作成し
て、**@title**に"Momo"をセッ
トし、**filename**メソッドを
呼び出してください。

```
class Book
  def title
    @title
  end
  def title=(t)
    @title = t
  end
end
```

```
# ここにDigitalBookクラスを書く
# (同じファイルでBookの下)
# filenameメソッドは以下。
def filename
  @title + ".pdf"
end
```

継承演習問題a2

以下のBook クラスを継承した DigitalBook(電子書籍) クラスを作ってください。

DigitalBookクラスに親クラスにあるメソッドと同名のtitleメソッドを実装するとどうなるでしょうか。DigitalBookクラスのインスタンスオブジェクトを作成して、@titleに"Momo"をセットし、titleメソッドを呼び出してください。

```
class Book
  def title
    @title
  end
  def title=(t)
    @title = t
  end
end
```

```
#DigitalBookクラスのtitleメソッド
def title
  "[ebook]" + @title
end
```

継承演習問題Sa1(上級)

継承したクラスで親クラスと同名のメソッドがある場合、メソッドは継承したクラスのものが優先して呼ばれます。しかし、親クラスのメソッドを上書きしてしまうわけではなく、`super` という命令を使うと親クラスの同名メソッドを呼び出すことができます。

以下のBook クラスを継承した DigitalBook(電子書籍) クラスを作り、親クラスにあるメソッドと同名のメソッドを作り、メソッド内で `super` を呼び出して動作を確認してみてください。

```
class Book
  def information
    puts "Great book!"
  end
end
```

```
# DigitalBookクラスのinfomationメソッド
def information
  puts "Infomation:"
  super
end
```

継承演習問題a1 解答

右のBook クラスを継承した
DigitalBook(電子書籍) クラスを作つ
てください。
DigitalBookクラスに右のような
filenameメソッドを実装してください。
DigitalBookクラスのインスタン
スオブジェクトを作成して、**@title**
に"**Momo**"をセットし、**filename**メ
ソッドを呼び出してください。

```
class Book
  def title
    @title
  end
  def title=(t)
    @title = t
  end
end

class DigitalBook < Book
  def filename
    @title + ".pdf"
  end
end

ebook = DigitalBook.new
ebook.title = "Momo"
p ebook.filename #=> "Momo.pdf"
p ebook.title #=> "Momo"
```

継承演習問題a2解答

以下のBook クラスを継承した DigitalBook(電子書籍) クラスを作ってください。

DigitalBookクラスに親クラスにあるメソッドと同名のtitleメソッドを実装するとどうなるでしょうか。DigitalBookクラスのインスタンスオブジェクトを作成して、@titleに"Momo"をセットし、titleメソッドを呼び出してください。

継承したクラスで親クラスと同名のメソッドがある場合は、メソッド定義が上書きされます。以下のように、Bookクラスには影響は出ません。

Bookクラスのtitle=, titleを呼んだ場合

```
book = Book.new  
book.title = "Momo"  
p book.title #=> "Momo"
```

```
class Book  
  def title  
    @title  
  end  
  def title=(t)  
    @title = t  
  end  
end
```

```
class DigitalBook < Book  
  def title  
    "[ebook]" + @title  
  end  
end
```

```
ebook = DigitalBook.new  
ebook.title = "Momo"  
p ebook.title #=> "[ebook]Momo"
```

継承演習問題Sa1 解答

継承したクラスで親クラスと同名のメソッドがある場合、メソッドは継承したクラスのものが優先して呼ばれます。しかし、親クラスのメソッドを上書きしてしまうわけではなく、`super` という命令を使うと親クラスの同名メソッドを呼び出すことができます。

以下のBook クラスを継承した DigitalBook(電子書籍) クラスを作り、親クラスにあるメソッドと同名のメソッドを作り、メソッド内で `super` を呼び出して動作を確認してみてください。

```
class Book
  def information
    puts "Great Book!"
  end
end
```

```
class DigitalBook < Book
  def information
    puts "Information:"
    super
  end
end
```

```
digital_book = DigitalBook.new
digital_book.information
```

コーディングでは
重複を排除するのが
大切
なぜかというと・・・

重複が多いコード

重複が多いコードは、動作を変更したいときにたくさん書き換える必要があります。

```
class Alice
  def hi
    "こんちは"
  end
end
```

```
class Bob
  def hi
    "こんちは"
  end
end
```

"こんちは"を
"hi"に
変更したい

この場合は2箇所直さないといけないけど、1回で済ませられないか？

```
class Alice
  def hi
    "hi"
  end
end
```

```
class Bob
  def hi
    "hi"
  end
end
```

そこで

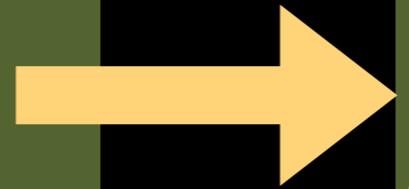
Module

メソッドを共同利用する仕組み

Module

クラスでモジュールをincludeすると、moduleに定義してあるメソッドをクラスへ追加することができます。

```
module Greeting
  def hello
    puts "Hello!"
  end
end
```



```
class Alice
  include Greeting
end

alice = Alice.new
alice.hello #=> "Hello!"
```

モジュールを定義します。
モジュールにはメソッドを
定義します。

include したAliceクラスに
はhelloメソッドが追加され
ます。

Module

include したAliceクラスは右のように書いたことと同じになります。

```
module Greeting
  def hello
    puts "Hello!"
  end
end
```

```
class Alice
  include Greeting
end
```



```
class Alice
  def hello
    puts "Hello!"
  end
end
```

Module

複数のクラスで同じメソッドを利用したいときにmoduleを使うと重複をなくせるので便利です。

```
module Greeting
  def hello
    puts "Hello!"
  end
end
```

もしもhelloメソッドで表示する"Hello!"を"こんにちは"に変えたい場合はこのモジュールだけ変更すれば良い

```
class Alice
  include Greeting
end

alice = Alice.new
alice.hello #=> "Hello!"
```

```
class Bob
  include Greeting
end
```

```
bob = Bob.new
bob.hello #=> "Hello!"
```



Moduleの文法

```
module モジュール名  
#メソッド定義  
end
```

```
class Sample  
include モジュール名  
end
```

module 定義

include(読み込み)

includeとrequireの違い

```
class Sample  
  include Greeting  
end
```

```
require "filename"
```

include は Sample クラスに Greeting モジュールで定義されているメソッドを追加する命令

require は他のファイル(.rb)で定義されているメソッドやクラス、モジュールを読めるようにする命令

Module演習問題b1

以下のコードが動作するようにGreetingモジュールを書いてください。

alice.rb

```
#ここにGreetingモジュールを書いてください
```

```
class Alice
  include Greeting
end
```

```
alice = Alice.new
alice.hi #=> "hi!"
```

Module演習問題Sb1

演習問題1で書いたGreetingモジュールを別のファイルへ保存してください。require文を使って読み込み、同様の動作をするコードを書いてください。

※require 文をここに書いてください。

```
class Alice  
  include Greeting  
end
```

```
alice = Alice.new  
alice.hi #=> "hi!"
```

※Greetingモジュールは別のファイルに保存してください。

Module演習問題1 解答

以下のコードが動作するよ
うにGreetingモジュール
を書いてください。

```
module Greeting
  def hi
    puts "hi!"
  end
end

class Alice
  include Greeting
end

alice = Alice.new
alice.hi #=> "hi!"
```

Module演習問題S1 解答

演習問題1で書いたGreetingモジュールを別のファイルへ保存してください。require文を使って読み込み、同様の動作をするコードを書いてください。

alice.rb \$ ruby alice.rb

```
require "./module_greeting"
```

↑ requireはファイル名

```
class Alice
```

```
  include Greeting
```

```
end
```

↑ includeはモジュール名

```
alice = Alice.new
```

```
alice.hi #=> "hi!"
```

module_greeting.rb

```
module Greeting
```

```
  def hi
```

```
    puts "hi!"
```

```
  end
```

```
end
```

※普通はこんな長いファイル名を付け
ずに例えばgreeting.rbとします。
ここではファイル名とモジュール名を
分けて説明したかったので・・・

參考資料

書式

Rubyコード

`puts "abc"`

実行結果

"abc"

実行結果

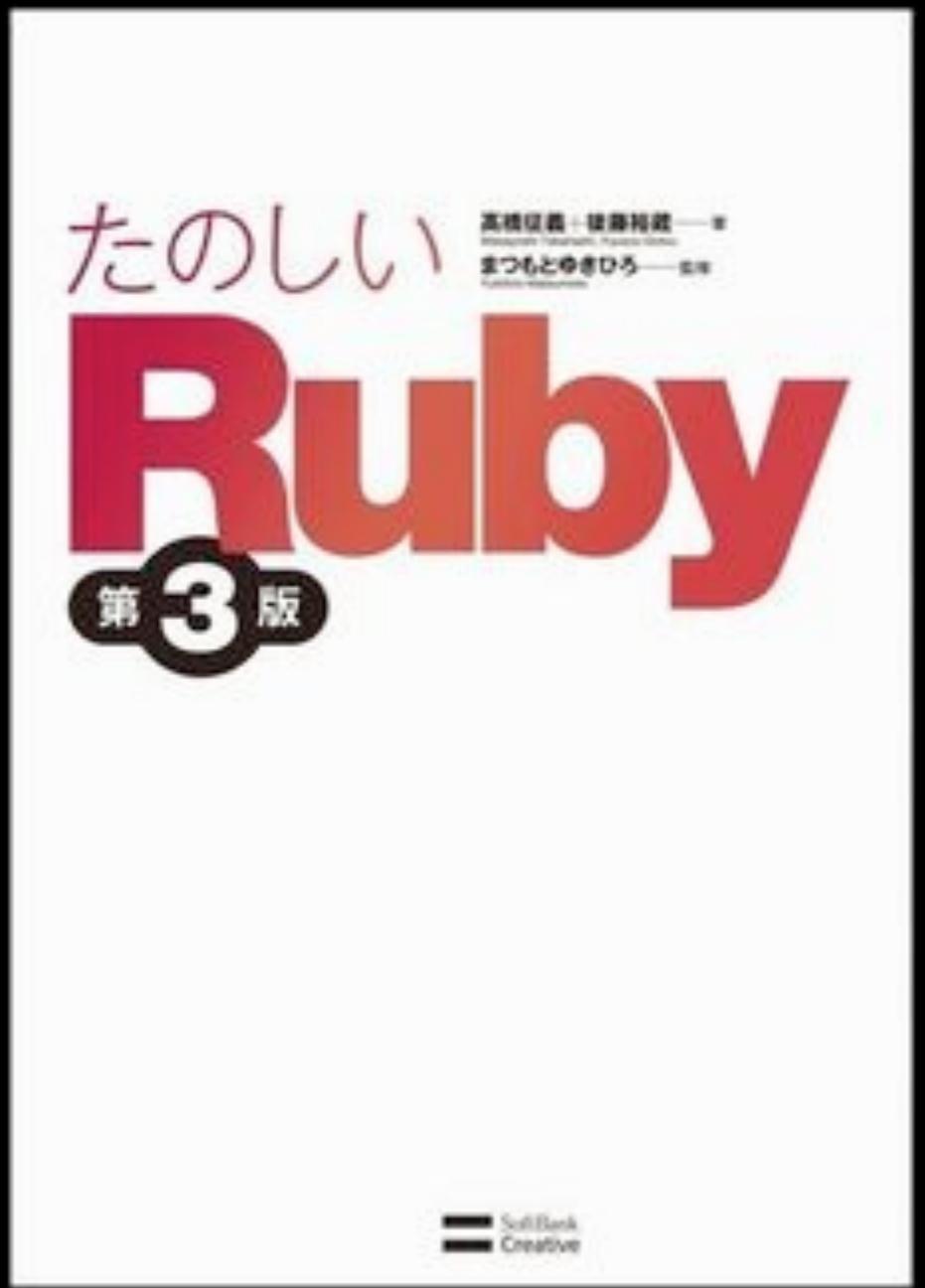
文中では `#=>` で書きます

`p 1+2 #=> 3`

shellコマンド

`$ ls`

教科書：たのしいRuby



<http://www.amazon.co.jp/dp/4797357401/>



お買い求めは
大学生協または
ジュンク堂池袋店で

講義資料置き場

過去の資料がDLできます。

<https://github.com/igaiga/hitotsubashi-ruby-2013>

雑談・質問用facebookグループ

<https://www.facebook.com/groups/hitotsubashi.rb>

- 加入/非加入は自由です
- 加入/非加入は成績に関係しません
- 参加者一覧は公開されます
- 参加者はスタッフ(講師・TA)と昨年、今年の受講者です
- 書き込みは参加者のみ見えます
- 希望者はアクセスして参加申請してください
- 雜談、質問、議論など何でも気にせずどうぞ~
- 質問に答えられる人は答えてあげてください
- 講師陣もお答えします
- 入ったら軽く自己紹介おねがいします