

Ruby 講義

第11回 繙承、Module

五十嵐邦明

twitter : igaiga555

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



2013.6.27 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



クックパッド

食べ過ぎリセット

お知らせオブジェクト

1番人気のレシピが今だけ無料で見放題▼

ログイン



料理名・食材名



目的・用途

レシピ検索

父の日 夏 夏バテ



注目ワード



レシピをのせる

*【簡単】チーズケーキオブジェクト

レシピを保存

単純作業ばかりの簡単レアチーズケーキ！

母からのお墨付きをもらいました♪

※材料、倍量に変更しました。

※ゼラチン多め

kanan店長

材料 (18cm底の抜ける丸型)

この部分の表示が
意図通り出ない・・・

☆水きりヨーグルト	200g
☆砂糖	90g
☆レモン汁(ポッカレモン)	大2
☆蜂蜜	10g
☆牛乳	200ml
★ゼラチン	16g

注目ワード
オブジェクトこここのデータを管理す
る役割はRecipeクラス
が受け持ってるから、そ
こに何かバグがあるのか
も？各オブジェクトに役割と責任を持たせる設計指針を
「オブジェクト指向」といいます。かして、袋の中に入
れ揉んでなじませ
る。・ゼラチンをふやか
す。

レシピオブジェクトの役割と責任

レシピオブジェクト

```
class Recipe
```

```
def title          公開ゾーン  
  @title  
end  
def title=(t)  
  @title = t  
  @updated_at = Time.now  
end
```

```
...               非公開ゾーン
```

```
  @title = "cheese cake"  
  @author = "igarashi"
```

```
...  
end
```

データへのアクセスは所定
のメソッドを通じてお願い
します。

→ 公開ゾーンと
非公開ゾーンを分ける

データを外から勝手に書き換えるとレシピオブジェクトは責任を果たせない
= データ(変数)は外部からはアクセスできなくすべき(非公開ゾーンに置くべき)
= Rubyのclassは最初からそうなるように設計されています

public, private

メソッドの公開・非公開を制御できます。

```
class AccessTest
```

```
public
```

```
#これ以降に書いたメソッドはpublic
```

```
def show
```

```
  puts "public method"
```

```
end
```

```
#ここに書いたメソッドもpublic
```

```
private
```

```
#これ以降に書いたメソッドはprivate
```

```
def secret
```

```
  puts "private method"
```

```
end
```

```
end
```

```
obj = AccessTest.new
```

```
obj.show #=> "public method"
```

```
obj.secret
```

public

以降のメソッドを公開メソッドにします。 (または、何も書かないと publicになります)

private

以降のメソッドを非公開メソッドにします。 非公開メソッドは、オブジェクト内部からは呼び出せますが、外部からは呼び出せません。

※protectedというのもあるのですが、滅多に使わないので省略

オブジェクトの内部と外部

```
class AccessTest
  def show
    secret #ここは内部
  end

  private
  def secret
    puts "private method"
  end
end
```

```
obj = AccessTest.new
obj.secret #ここは外部 ✗
```

内部

内部

そのオブジェクトクラスのメソッド
の中。そのオブジェクトクラスの
class～endの間。

外部

それ以外

または、

secret のようにメソッド名だけで呼べ
るところが内部、

obj.secret のように
オブジェクト.メソッド名で
呼ぶところが外部です。

クラスについてよくわからない
場合は、前回、前々回の資料を
復習してみてください

第9回 クラス・インスタンス

<https://speakerdeck.com/igaiga/ruby09>

第10回 オブジェクト指向

<https://speakerdeck.com/igaiga/ruby10>

ここから今週の内容

A close-up photograph of a woman with blonde hair and green eyes, smiling gently at the camera while holding a baby. The woman is wearing a light-colored top and a necklace. The baby has dark hair and is looking towards the camera. The background is blurred.

目次

継承
Module

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 Magazine
  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。そんなときは、継承を使うとすっきり書けます。

継承

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。

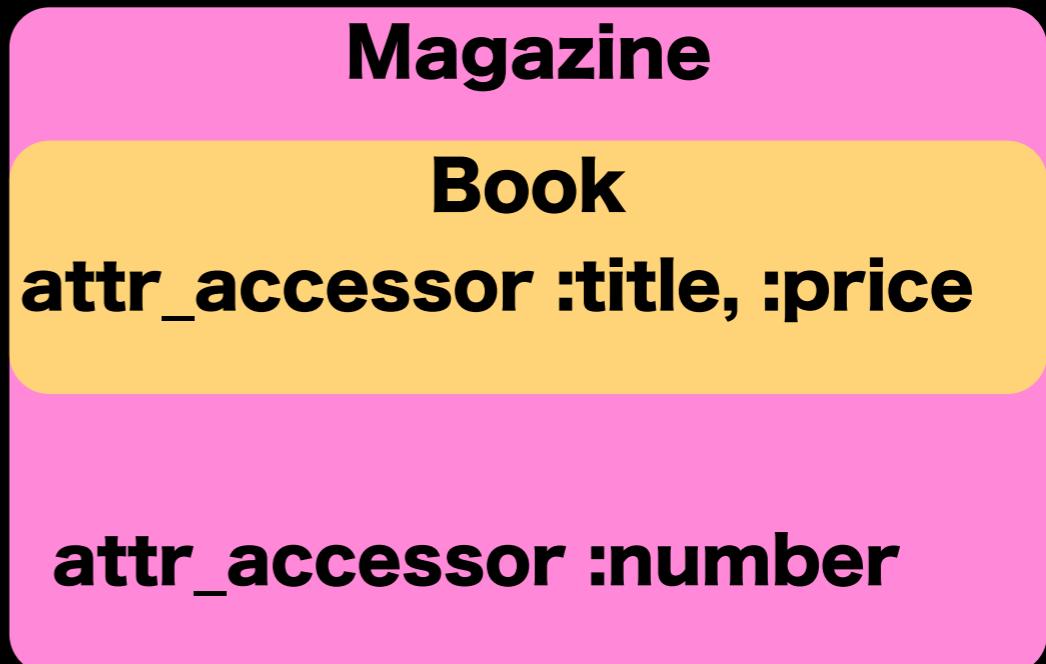
継承

継承を使った書き方

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

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

図解



Bookクラスを継承したMagazineクラスは、Bookクラスの持ち物を受け継ぎます。この例の場合は、Bookクラスの attr_accessor :title, :price をMagazineクラスでも使えます。加えて、Magazineクラスにある attr_accessor :number も利用できます。

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、親クラスの全てのメソッド、インスタンス変数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(a1と同じです)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a3(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 "Digital book infomation:"
  super
end
```

次のページに
解答があります

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、メソッド定義がオーバーライド（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a3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 "Digital book 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
    "こんにちは"
  end
end
```

```
class Bob
  def hi
    "こんにちは"
  end
end
```

"こんにちは"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

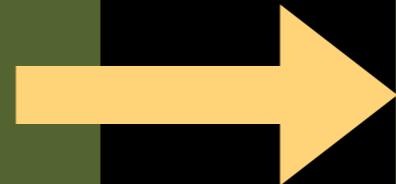
Module

メソッドを共同利用する仕組み

Module

moduleにメソッドを定義しておくと、 クラスにinclude文を書くことで、 moduleのメソッドを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。

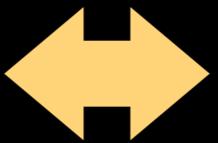
Module

以下の2つのAliceクラスは同じ動作になります。

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

class Alice
  include Greeting
end
```

同じ
動作



```
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を使うと重複なく書けるので便利です。

```
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

混同し易い2つの命令、それぞれの意味を復習しておきましょう。

include は**module**で定義されてい
るメソッドを追加する命令。
module 名を指定する。

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

```
class Human
  include Greeting
end

Human.new.hello #=> "hello!"
```

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。
ファイル名を指定する。

```
other.rb
def hi
  puts "hi!"
end
```

```
main.rb
require "./other"
hi #=> "hi!"
```

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b2(上級)

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。

次のページに
解答があります

Module演習問題b1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b2解答（上級）

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`

課題チェック用の付箋の書き方

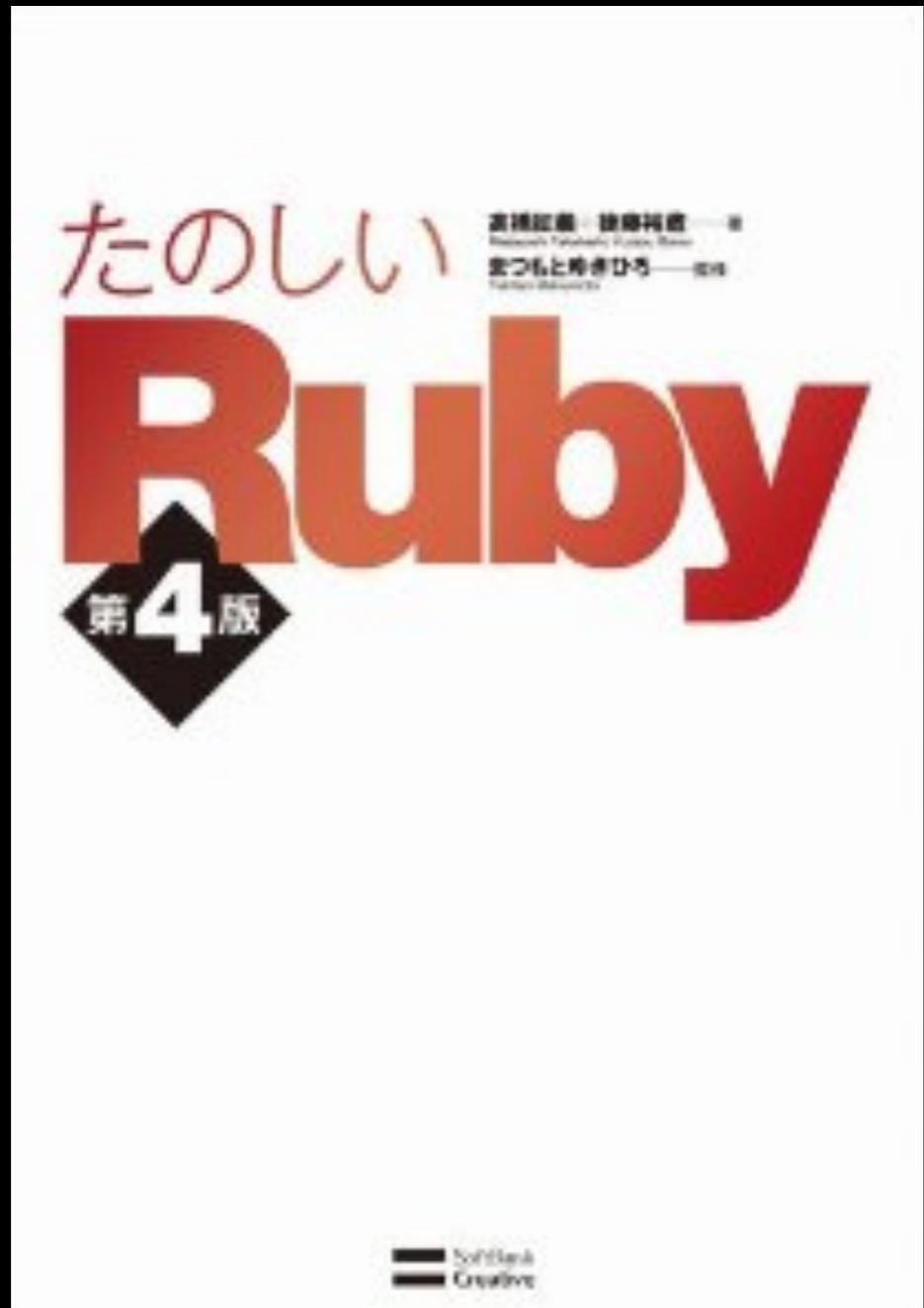
上の方に学籍番号と名前を書いてください

学籍番号

名前

※この辺に講師陣がクリアした課題番号を書いていきます。

教科書：たのしいRuby



<http://www.amazon.co.jp/dp/4797372273/>



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
- 雑談、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