

Ruby 講義

第11回 Ruby入門

Kuniaki IGARASHI/igaiga

2012.6.21 at 一橋大学

社会科学における情報技術とコンテンツ作成III
(ニフティ株式会社寄附講義)

○ 剰余金の配当に関するお知らせ

○ ニフティ、「@nifty EMOBILE LTE 定額にねんプラン」の提供を開始

○ 「@nifty温泉」で「母の日 全国一斉！100のありがとう風呂」特設サイト公開

○ 「スマブレ！」のサービス停止について

○ ニフティとサンリオウェーブ、iOS向けアプリ「Hello Kitty Worl...」

○ 平成24年3月期 決算短信

○ 特別損失の計上に関するお知らせ

○ 「シユフモ」登録会員数150万人を突破、「2012年主婦の全国節電調査（冬季...」

ニフティとなら、きっとかなう。
With Us, You Can.

ニフティ株式会社

アット・ニフティ
楽しいサービスがいっぱい

アクセスマップ
大森から西新宿へ移転いたしました

@nifty Web募金
東日本大震災復興支援
募金受付中

- 2012年4月25日 IR [特別損失の計上に関するお知らせ](#)
- 2012年4月25日 IR [剰余金の配当に関するお知らせ](#)
- 2012年4月19日 IR [「@nifty EMOBILE LTE 定額にねんプラン」の提供を開始](#)
- 2012年4月19日 IR [ニフティとサンリオウェーブ、iOS向けアプリ『Hello Kitty World』を台湾で提供開始](#)
- 2012年4月10日 お知らせ [「@nifty温泉」で「母の日 全国一斉！100のありがとう風呂」特設サイト公開](#)

提供

講師

五十嵐邦明

株式会社万葉

エンジニア



GARASHI

.9.25 at 高専カンファ

Teaching Assistant 演習 健二
クックパッド株式会社 エンジニア



先週の
おさらい

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
```

レシピに関するデータと手続きを持ちます。

レシピに関するデータを管理します。

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。

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
= 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
```

```
def show
```

```
  puts "public method"
```

```
end
```

```
#ここに書いたメソッドはpublic
```

```
private
```

```
def secret
```

```
  puts "private method"
```

```
end
```

```
# ここに書いたメソッドはprivate
```

```
end
```

```
obj = AccessTest.new
```

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

```
obj.secret X
```

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 X
```

↑
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、

priv のようにメソッド名だけで呼べると
ころが内部、

obj.priv 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。

今週

ここから

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

そんなときは、継承を使うと便利です。

書き方A

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

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

書き方B

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

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

←→
同じ動作

`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1

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の下に
# filenameメソッドは以下。
def filename
  @title + ".pdf"
end
```

継承演習問題2

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
```

```
def title
  "[ebook]" + @title
end
```

継承演習問題S1

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
```

```
def information
  puts "Infomation:"
  super
end
```

継承演習問題1 解答

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
```

```
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2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 = 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S1 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
```

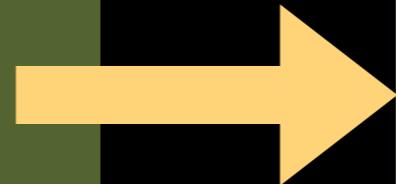
Module

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。

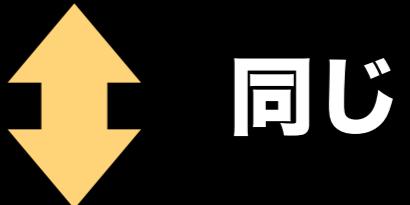
include したAliceクラスに
はhelloメソッドが追加され
ます。

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

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、モジュールを使えるようにする命令

Module演習問題1

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S1

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てずにmodule.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・・・

講義資料置き場

講義資料置き場をつくりました。
過去の資料がDLできます。

<https://github.com/hitotsubashi-ruby/lecture2012>

or

<http://bit.ly/ruby-lecture>

雑談・質問用facebookグループ

facebookグループを作りました

<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
- ・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