項目

- 7.7 メソッドの公開レベル (p.246〜254)
- 7.8 定数についてもっと詳しく (p.254〜258)

WEF2105 髙木基寛 2021.07.07

7.7 メソッドの公開レベル

Ruby には以下3つの公開レベルがある。

- public
- protected
- private

まずは
publicメソッドについて

- public
- protected
- private

7.7 メソッドの公開レベル

┗7.7.1 publicメソッド

public

クラスの外から自由に呼び出せるメソッド

```ruby
class User
  def hello
    'Hello!'
  end
end
user = User.new
p user.hello
```

```
PROBLEMS  OUTPUT  ターミナル  デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
"Hello!"
```

Userクラスの外部から、helloメソッドをuser.helloで呼び出しが出来ている。

「public」の記述があるわけではない。

デフォルトでpublicだから。

次頁から
privateメソッドについて

- public
- protected
- private

7.7 メソッドの公開レベル

┗7.7.2 privateメソッド

private

クラスの外から自由に呼び出すことが出来ず、クラス内のみで使えるメソッド

厳密には、レシーバを指定して呼び出すことができないメソッド

```ruby
class User
  private
  def hello
    'Hello!'
  end
end
user = User.new
p user.hello
```

これ

```
PROBLEMS  OUTPUT  ターミナル  デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
Traceback (most recent call last):
lesson.rb:17:in `<main>': private method `hello' called for #<User:0x00007fa6f09e6698> (NoMethodError)
```

この例だと、userがhelloメソッドのレシーバ

helloメソッドがprivateメソッドのためuser.helloで呼び出すと

エラーになる

7.7 メソッドの公開レベル

┗7.7.2 privateメソッド

private

privateメソッドではself付きで呼び出すとエラーになる。

```
20  class User
21    def hello
22      "Hello, I am #{self.name}."
23    end
24    private
25    def name
26      'Alice'
27    end
28  end
29  user = User.new
30  p user.hello
```

これ

```
PROBLEMS    OUTPUT    ターミナル    デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
Traceback (most recent call last):
        1: from lesson.rb:30:in `<main>'
lesson.rb:22:in `hello': private method `name' called for #<User:0x00007f80bf8862c8> (NoMethodError)
```

クラス内で、他のメソッドを呼び出す時に使えるselfも
privateメソッドでは使えない。
selfというレシーバを指定してメソッドを呼び出したことになるから。

7.7 メソッドの公開レベル

┗7.7.3 privateメソッドはサブクラスでも呼び出せる

private

スーパークラスのprivateをサブクラスで呼び出せてしまう

```ruby
class Product
  private
  def name
    'A greate movie'
  end
end
class DVD < Product
  def to_s
    "name: #{name}"
  end
end
dvd = DVD.new
p dvd.to_s
```

スーパークラス

サブクラス

```
PROBLEMS  OUTPUT  ターミナル  デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
"name: A greate movie"
```

サブクラスDVDからスーパークラスProductのnameメソッドを呼び出せている。

private

7.7 メソッドの公開レベル

┗7.7.4 クラスメソッドをprivateにしたい場合

privateメソッドが使えるのはインスタンスメソッドだけ
クラスメソッドは使えない

```
48  class User
49    private
50    def self.hello
51      'Hello!'
52    end
53  end
54  p User.hello
55
```

```
PROBLEMS    OUTPUT    ターミナル    デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
"Hello!"
```

クラスメソッドUser.helloを呼び出すと、、

呼び出せてしまう❗つまりprivateが機能していない

クラスメソッドをprivateにする方法は次ページ➠

private

7.7 メソッドの公開レベル

┗7.7.4 クラスメソッドをprivateにしたい場合

クラスメソッドをprivateにする方法1（方法は２つ）
「class << selfを使う」

```
57  class User
58    class << self
59      private
60      def self.hello
61        'Hello!'
62      end
63    end
64  end
65  p User.hello
```

これ

```
PROBLEMS  OUTPUT  ターミナル  デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
Traceback (most recent call last):
lesson.rb:65:in `<main>': undefined method `hello' for User:Class (NoMethodError)
```

エラーになった。

クラスメソッドがちゃんとprivateになっている。

private

7.7 メソッドの公開レベル

┗7.7.4 クラスメソッドをprivateにしたい場合

クラスメソッドをprivateにする方法2
「private_class_methodというメソッドを使う」

```
68  class User
69    def self.hello
70      'Hello!'
71    end
72    private_class_method :hello
73  end
74  p User.hello
75
```

これ

```
PROBLEMS  OUTPUT  ターミナル  デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
Traceback (most recent call last):
lesson.rb:74:in `<main>': private method `hello' called for User:Class (NoMethodError)
```

エラーになった。privateになっている。

private

7.7 メソッドの公開レベル

┗7.7.5 privateメソッドから先に定義する場合

privateを先に定義して、以降でpublicを使用したい場合
以下のように記述する。

```
77    class User
78      private
79      def foo
80      end
81      public
82      def bar
83      end
84    end
```

7.7.1のコードでは「public」は記述していなかったが、

privateメソッド、publicメソッドをこのように好きな順番で使いたい場合は記述する。

だが、通常はprivateを使うのは1回だけ。

クラスの最後にprivateメソッドを定義するのがよい。

7.7 メソッドの公開レベル

└7.7.6 あとからメソッドの公開レベルを変更する場合

private

privateメソッドに、既存のメソッド名を引数として渡すと
そのメソッドがprivateメソッドになる。

```
77  class User
78    def foo
79      'foo'
80    end
81    def bar
82      'bar'
83    end
84    private :foo, :bar
85    def baz
86      'baz'
87    end
88  end
89  user = User.new
90  p user.foo
91  p user.bar
92  p user.baz
```

これ

privateメソッドに引数を渡した場合、

その下に定義したメソッドはprivateが適用されない。

出力結果は、fooとbarはエラー、bazはbazが出力される。

次頁から protectedメソッドについて

- public
- protected
- private

7.7 メソッドの公開レベル

┗7.7.7 protectedメソッド

protected

クラス外からは呼び出せないが、同じクラスやサブクラスの中であれば
レシーバ付きで呼び出せる。

```ruby
class User
  attr_reader :name
  def initialize(name, weight)
    @name = name
    @weight = weight
  end
  def heavier_than?(other_user)
    other_user.weight < @weight
  end
  protected
  def weight
    @weight
  end
end
alice = User.new('Alice', 50)
bob = User.new('Bob', 60)
p alice.heavier_than?(bob)
p bob.heavier_than?(alice)
p alice.weight
```

これ

```
PROBLEMS  OUTPUT  ターミナル  デバッグ コンソール  zsh

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
false
true
Traceback (most recent call last):
lesson.rb:113:in `<main>': protected method `weight' called for #<Use
r:0x00007f8b289e5290 @name="Alice", @weight=50> (NoMethodError)
```

・weightメソッドは、Userクラス内で定義されているから、Userクラス内では使える
・Userクラスの外では使えない
alice.weightでエラーになる
・weightは公開したくないけど、引数として使いたい！というときに使える。

Column

継承したら同名のインスタンス変数に注意

スーパークラスとサブクラスでインスタンス変数の名前がかぶった場合
スーパークラスのインスタンス変数が持つ値が入るなど、
意図しない結果が返ってくるため、クラス継承をする場合、
インスタンス変数の命名に注意が必要

項目

- 7.7 メソッドの公開レベル (p.246〜254)
- 7.8 定数についてもっと詳しく (p.254〜258)

7.8 定数についてもっと詳しく

定数は、クラスの外から直接参照することができる。

クラスの外から定数を参照する場合の書き方

クラス名::定数名

```
116    class Product
117      DEFAULT_PRICE = 0
118    end
119    p Product::DEFAULT_PRICE
```

```
PROBLEMS    OUTPUT    ターミナル    デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
0
```

定数に0を定義しているので0が返る

定数の記述位置は、クラス構文の直下にすること

```ruby
class Product
  def foo
  DEFAULT_PRICE = 0
  end
end
```

この位置は❌

メソッドの内部では作成できない。

7.8 定数についてもっと詳しく

└7.8.1 定数と再代入

定数には、再代入が可能。（定数の値を後から書き換えられる）
クラスの内部からも、外部からも可能

```
124    class Product
125      DEFAULT_PRICE = 0
126      DEFAULT_PRICE = 1000
127    end
128    p Product::DEFAULT_PRICE
```

これ

```
PROBLEMS    OUTPUT    ターミナル    デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
lesson.rb:126: warning: already initialized constant Product::DEFAULT_PRICE
lesson.rb:125: warning: previous definition of DEFAULT_PRICE was here
1000
```

上記は、クラスの内部で再代入

再代入した1000が返る

warningは出るが、再代入は成功している。

7.8 定数についてもっと詳しく

└7.8.1 定数と再代入

定数には、再代入が可能。（定数の値を後から書き換えられる）
クラスの内部からも、外部からも可能

```
124    class Product
125      DEFAULT_PRICE = 0
126      DEFAULT_PRICE = 1000
127    end
128    Product::DEFAULT_PRICE = 3000
129    p Product::DEFAULT_PRICE
```

これ

```
PROBLEMS    OUTPUT    ターミナル    デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
lesson.rb:126: warning: already initialized constant Product::DEFAULT_PRICE
lesson.rb:125: warning: previous definition of DEFAULT_PRICE was here
lesson.rb:128: warning: already initialized constant Product::DEFAULT_PRICE
lesson.rb:126: warning: previous definition of DEFAULT_PRICE was here
3000
```

上記は、クラスの外部で再代入

再代入した3000が返る

warningは出るが、再代入は成功している。

7.8 定数についてもっと詳しく

└7.8.1 定数と再代入

freezeにより再代入を防ぐことが可能

```
124    class Product
125      DEFAULT_PRICE = 0
126      Product.freeze
127      DEFAULT_PRICE = 1000
128    end
129    p Product::DEFAULT_PRICE
```

これ

```
PROBLEMS    OUTPUT    ターミナル    デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
Traceback (most recent call last):
        1: from lesson.rb:124:in `<main>'
lesson.rb:127:in `<class:Product>': can't modify frozen #<Class:Product> (FrozenError)
motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
```

freezeを入れると、クラスは変更を受け付けなくなる

7.8 定数についてもっと詳しく

└7.8.2 定数はミュータブルなオブジェクトに注意する

ミュータブルなオブジェクトは、再代入をしなくても定数の値を変えられる

ミュータブルなオブジェクトとは、、文字列、配列、ハッシュなど

```
132  class Product
133    NAME = 'A product'
134    SOME_NAMES = ['Foo', 'Bar', 'Baz']
135    SOME_PRICES = { 'Foo' => 1000, 'Bar' => 2000, 'Baz' => 3000 }
136  end
137  Product::NAME.upcase!
138  p Product::NAME
139  Product::SOME_NAMES << 'Hoge'
140  p Product::SOME_NAMES
141  Product::SOME_PRICES['Hoge'] = 4000
142  p Product::SOME_PRICES
```

```
PROBLEMS    OUTPUT    ターミナル    デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
"A PRODUCT"
["Foo", "Bar", "Baz", "Hoge"]
{"Foo"=>1000, "Bar"=>2000, "Baz"=>3000, "Hoge"=>4000}
```

文字列は、upcase!により大文字に

配列は、Hogeが追加される

ハッシュは、”Hoge”=>4000が追加される

7.8 定数についてもっと詳しく

└7.8.2 定数はミュータブルなオブジェクトに注意する

破壊的な変更が出来ないようにするためにはfreezeを使う

配列をfreezeする

```
145    class Product
146      SOME_NAMES = ['Foo', 'Bar', 'Baz'].freeze
147      def self.names_without_foo(name = SOME_NAMES)
148        name.delete('Foo')
149        names
150      end
151    end
152    p Product.names_without_foo
```

これ

```
PROBLEMS    OUTPUT    ターミナル    デバッグ コンソール

motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
Traceback (most recent call last):
        2: from lesson.rb:152:in `<main>'
        1: from lesson.rb:148:in `names_without_foo'
lesson.rb:148:in `delete': can't modify frozen Array (FrozenError)
```

Fooをdeleteしようとしても、freezeにより、配列の値の変更ができずエラー

7.8 定数についてもっと詳しく

┗7.8.2 定数はミュータブルなオブジェクトに注意する

（続き）破壊的な変更が出来ないようにするためにはfreezeを使う

配列の中の各要素もfreezeする

```ruby
SOME_NAMES = ['Foo'.feeze, 'Bar'.freeze, 'Baz'.freeze].freeze
```

これ

しかし、各要素全部にfreeze付けるのかっこ悪い！
だから、mapを使う↓

```ruby
SOME_NAMES = ['Foo', 'Bar', 'Baz'].map(&:freeze).freeze
```

7.8 定数についてもっと詳しく

┗7.8.2 定数はミュータブルなオブジェクトに注意する

イミュータブルなオブジェクトは破壊的に変更されることはない。

イミュータブルとは
p.124　**数値、シンボル、true/false、nil**の4種類

この4種類においては破壊的に変更されないため、freeze不要

Q & A

- Q1：privateメソッドによる非公開とは、何に対する非公開なのか？
  A1：github等で公開されないというわけではない。
  　　クラス外に対するアクセス制限をかけている。

- Q2：そもそもなぜprivateを使うのか。何から守っているのか。
  　　クラス外からアクセス出来ないようにして何がいいのか？
  A2：-書き換えが出来ないようにする。
  　　外からアクセスができるということは書き換えができてしまうということ
  　　-オブジェクト指向の3つの原則カプセル化、継承、ポロモーフィズムの中の、
  　　カプセル化の概念。privateを使うことでカプセル化つまり隠す。

- Q3：freezeは何に対して機能しているのか（スライド22）
  A3：freezeの位置を後にするとエラーにならなかった。
  　　クラス内で破壊的変更が起きた場合にエラーとなると思ったが、
  　　freezeの記述以降で破壊的変更があった場合にエラーとなる。が正しいみたいです。

```
137  class Product
138    DEFAULT_PRICE = 0
139    DEFAULT_PRICE = 1000
140    Product.freeze
141  end
142  p Product::DEFAULT_PRICE
```

問題　出力　ターミナル　デバッグ コンソール

```
motohirotakagi@MotohironoAir Ruby-book % ruby lesson.rb
lesson.rb:139: warning: already initialized constant Product::DEFAULT_PRICE
lesson.rb:138: warning: previous definition of DEFAULT_PRICE was here
1000
```