Ruby 初級者向けレッスン 37 回
— String クラスを改造しよう —

こなみひでお@Ruby 関西

2010 年 6 月 19 日

概要

クラスを新しく定義したり，既存のクラスを活用することを学びます。

Ruby のバージョンは 1.8.7 を想定しています.

1.9.x は文字列やイテレータまわりの仕様が大きく変更されています．新しいメソッドの一部は 1.8.7 にも取り込まれており，文字列関係では each_char が使えます[*1].

1 Ruby のクラスのいじり方

1.1 クラスを定義する

新しいクラスを定義することができます。

```
class Hoge
  def initialize(arg1,arg2,...)
    ...
  end

  def method1(arg)
    ...
  end
end
```

arg1, ... は仮引数です。

1.2 既存のクラスから継承する

```
class Animal
  ...
end
```

[*1] もともと 1.8 系では添付ライブラリの jcode を require しないと使えないのですが， 1.8.7 ではデフォルトで組み込まれているようです.

```
4
5  class Dog < Animal
6    ...
7  end
```

Dog クラスは Animal のメソッド等を継承します。このとき，既存のメソッドを Dog 固有の振る舞いをするように上書き（オーバライド）することができます。

1.3 既存のクラスの機能を追加する

クラスを新しく作るだけではなく，次のように既存のクラスをいじってしまうこともできます。

```
1  class String
2    def new_method(arg)
3    ...
4    end
5  end
```

1.4 基本的なお題

練習 1–1 Array クラスには push, pop というスタック操作のメソッドが用意されています。このことを利用して，超簡単なやり方で Stack クラスを作って下さい。

練習 1–2 練習 1–1 のままだと，スタックなのにキューとしても使えるオブジェクトができてしまいます．shift や unshift を使えば配列の先頭からのデータの出し入れができるからです．それを封じて Stack クラスのインスタンスが「勝手な」ことをしないようにしてください．

練習 1–3 String クラスに fun というメソッドを追加して下さい。このメソッドは String オブジェクトをレシーバとして，インスタンスの末尾に'---' という「金魚のフン」を追加します。

```
p "kingyo".fun   #=> "kingyo---"
```

self でレシーバの本体を参照

下に練習 1–3 の解答例を示します．間違いの例では，メソッドの使い方が puts fun("kingyo") というふうに，引数を与えたメソッド呼び出しになってしまいます。ここでは文字列を引数と渡すのではなく，文字列をレシーバとして，それに働きかけるメソッドをほしいのです。

メソッド中で，レシーバとなるインスタンスの本体は self で表現できます。この変数は**擬似変数**と呼びます。

間違ったソース

```
1  class String
2    def fun(str)
3      return str + "---"
4    end
5  end
```

正しいソースの例 (予定)

```
1  class String
2    def fun
3      ??????
4    end
5  end
```

2 String クラスの新メソッドを実装しよう

お題

Array クラスの次のメソッドを String クラスにも作って下さい。文字列はアスキー文字 (1 バイト文字) だけからなるものとします．`sort` については，非破壊的なメソッドと破壊的なメソッドを作ってください．

`push`, `pop`, `shift`, `unshift`, `first`, `last`, `sort`[a]

[a] `sort` は文字列をレシーバとしても一応動作しますが，結果はまったく意味のないものです．ですから，ここで `sort` についてやろうとしていることは既存メソッドのオーバライドです．

push の仕様

String クラスのインスタンスをレシーバとし，1 文字を引数として，レシーバの末尾に追加します．戻り値は追加された文字列自身とします．

```
str = "ABCD"
str.push("X")
p str   #=> "ABCDX"
```

pop の仕様

String クラスのインスタンスをレシーバとして，文字列の末尾の 1 文字を返す．返した文字はインスタンスから削られる．

```
str = "ABCD"
p str.pop    #=>  "D"
p str        #=>  "ABC"
```

shift,unshift の仕様

`shift`, `unshift` は `pop`, `push` と似た動作をしますが，対象となる文字は先頭の文字になります．

```
str = "ABCD"
p str.shift       #=>  "A"
p str             #=>  "BCD"
p str.unshift "X" #=>  "XBCD"
p str             #=>  "XBCD"
```

first, last の仕様

first, last は文字列の最初あるいは最後の文字を返します．

```
str = "ABCD"
p str.first       #=>  "A"
p str.last        #=>  "D"
p str             #=>  "ABCD"
```

2.1 破壊的，非破壊的？

レシーバにメソッドを作用させたときに，レシーバ自身を変更するようなメソッドは**破壊的なメソッド**，レシーバ自身はそのままであるようなメソッドは**非破壊的なメソッド**といいます．

上の例だと，push, pop, shift, unshift は破壊的なメソッド，first, last は非破壊的なメソッドです．

破壊的なメソッドを作る方法は後で解説します．

2.2 既存のクラスメソッドを確かめる

String クラスの新しいメソッドを実装する前に，ネタ取りに使っている Array クラスのメソッドを irb で確かめておきましょう．他のメソッドについてもどうぞ．

```
ar = ["O","R","A","N","G","E"]
ar.push("Q")
ar.pop
```

こんどは同じメソッドが String クラスで備わっていないかどうかも確認してください．あれ？ sort とか一応あるんですね・・・

2.3 String クラスのメソッドを活用

配列型のメソッド

リファレンスマニュアルで String クラスのメソッドを調べましょう．

self[nth], self[nth, len], self[substr], self[regexp], self[regexp, nth], self[first..last], self[first...last], self[nth]=val, self[nth, len]=val, ...

これらは文字列を配列のように扱っている「配列型」のメソッドです[*2]．これらを中心に，その他の String クラスのメソッドと組み合わせて使えば，お題はほとんど実装できるはずです．

破壊的なメソッドはどうやって作る？

擬似変数 self への代入は禁止されています．したがって，次のようなコードでレシーバ本体を変更することはできません．

[*2] C の場合，文字列はそのまま配列ですね．

```
class String
  def fun!
    self = self + "---"
  end
end
str = "Kingyo"
str.fun!
puts str
```

実行結果:

```
self.rb:3: Can't change the value of self
    self = self + "---"
           ^
```

こんなときには String クラスのメソッド `replace` を使います.

```
class String
  def fun!
    self.replace(self + "---")
  end
end
puts "Kingyo".fun!
```

2.4 Array クラスのメソッドを活用できないか？

文字列に配列と同じような動作をさせるのであれば，次の方針も使えそうです.

1. 文字列を配列に変換する
2. 配列に対して Array クラスのメソッドを働かせる.
3. 配列を元の文字列に戻す.

つまり次のようなメソッドのペアがあればいいのですね．探して使ってみましょう.

文字列 ⇔ 配列

2.5 日本語を使えるようにするには？

ここまでの練習では，アスキー文字，つまり 1 バイト文字だけからなる文字列を扱っていました．しかしそれでは実用にはなりません．日本語などのマルチバイト文字を扱えるようにするにはどうしたらよいでしょうか？

- ソースプログラムが日本語を含んでいる場合，ソースの文字コードを確かめましょう．Emacs 系エディタなら画面の左下に表示があります.
  あなたの使っているプラットフォームが UNIX/MacOS 系であれば，文字コードは UTF-8, Windows

であれば Shift-JIS と考えてよいでしょう[*3]の中の文字データには UTF 系，Shift-JIS 系以外に iso-2022-jp, EUC-JP などがあります．

- 文字コードに合わせて， `$KCODE="u"`, `$KCODE="s"` のような文字コード指定をプログラムの先頭に入れます．u, s はそれぞれ UTF-8, Shift-JIS の頭文字です．
- Ruby の起動につぎのようなオプションを付けてください．

  `ruby -Ku（または-Ks）`

  このオプションをスクリプトの先頭の shebang に入れてもよいでしょう．

  ```
  #!/usr/local/bin/ruby -Ku
  ```

- 必要なら添付ライブラリとして `jcode` を読み込みます．プログラムの `$KCODE` 指定の後ろに次の行を入れてください．

  ```
  require 'jcode'
  ```

[*3] 世

3 Ruby TIPS

3.1 サンプルコードを書くとき

クラスやメソッドを定義するためのソースは，他のプログラムから require で読み込んで走らせることが多いでしょう。一方，開発中のソースを走らせるためのサンプルプログラムを別に用意するのも面倒です。

そこで次のような工夫をします。hogehoge.rb というファイル名で次のソースを保存して下さい。if 以下の部分がサンプルコードです。

```
1  class Hoge
2    def honya
3        puts "Hogehoge"
4    end
5  end
6
7  # puts __FILE__
8  # puts $0
9  if __FILE__ == $0
10   h = Hoge.new
11   h.honya
12 end
```

ここで __FILE__ という疑似変数には，この行を含むファイルの hogehoge.rb が代入され， $0 には，実行しているプログラム名が代入されます。7,8 行目をアンコメントして確認して下さい。

このソースを次のようにして直接実行して下さい。> はシェルのプロンプトです。

```
>  ruby hogehoge.rb
```

ちゃんと走って，サンプルコードが実行されていますね。

次に irb を起動して，hogehoge.rb を読み込んで下さい。

```
irb(main):001:0> require "hogehoge"
```

require によって hogehoge.rb が irb 上で実行され，クラス定義が読み込まれます。その一方，Hoge クラスのインスタンス生成も， honya というメソッドの実行もスキップされています。

そこで，次のようにクラスインスタンスを生成して，メソッドを実行するとちゃんと動きます。

```
irb(main):002:0> a = Hoge.new
irb(main):003:0> a.hoge
```

つまり require で読み込まれるクラス定義ファイルとしてはちゃんと働いていて，しかも間接実行なので $0 には hogehoge.rb ではなく呼び出したプログラムの名前が代入されているために， サンプルコードは実行されないのです。