Ruby 初級者向けレッスン第 36 回

okkez @ Ruby 関西

2010 年 05 月 15 日

今回の内容

- オブジェクト指向
  - 歴史
  - 使うオブジェクト指向
  - 作るオブジェクト指向

今回のゴール

- 使うオブジェクト指向を理解する
- 作るオブジェクト指向を知る

歴史

オブジェクト指向以前

- データ構造もへったくれも無い時代
  - 大昔
  - そこそこの規模 (複雑さ) のソフトウェアで破綻
- データと処理を分ける時代
  - 構造化プログラミング -> ルーチンの抽象化
  - 大規模なソフトウェアにも対応できる (と言われていた)

構造化プログラミングの限界

- さらなる大規模 (もっと複雑な) ソフトウェアの出現
  - データ構造と処理を分けるだけでは対応不可能に

オブジェクト指向言語の登場

- Simula, Smalltalk
- C++, Java
- Python, JavaScript
- Ruby

使うオブジェクト指向の具体例

いくつか具体例を見ていきます。
分からないメソッドがあったらリファレンスマニュアルを適宜参照してください。

文字列を扱いたい

```
01: str = "foobarbaz"
02: p str.include?('bar')  # => true
03: p str.include?('hoge') # => false
```

str という変数に、ある文字列が含まれているかどうか調べています。

```
01: str = "foobarbaz"
02: p str.upcase   # => "FOOBARBAZ"
03: p str          # => "foobarbaz"
04: p str.upcase!  # => "FOOBARBAZ"
05: p str          # => "FOOBARBAZ"
06: p str.upcase!  # => nil
```

str という変数に格納されている文字列を大文字にしています。

数値を扱いたい

```
01: 1 + 2  # => 3
02: 1.+(2) # => 3
```

数値の計算はこのように書くことができます。

```
01: 10.times do
02:   puts 'Hello, Ruby!'
03: end
```

10 回 "Hello, Ruby!" と表示します。

配列・ハッシュを扱いたい

```
01: obj = [1, 2, 3, 4, 5]
02: obj.each{|v| p v }
03: obj = { :a => 'aaa', :b => 'bbb', :c => 'ccc', :d => 'ddd'}
04: obj.each{|v| p v }
05: # >> 1
06: # >> 2
07: # >> 3
08: # >> 4
09: # >> 5
10: # >> [:a, "aaa"]
11: # >> [:b, "bbb"]
12: # >> [:c, "ccc"]
13: # >> [:d, "ddd"]
```

obj の内容が違いますが、同じメソッドを呼び出しています。

使うオブジェクト指向 - まとめ

使うオブジェクト指向とは、上記のようにあるオブジェクトに対して行ってほしい事を伝えることにより、目的を達成することです。

既にあるもの (組み込みクラス、標準添付ライブラリ) を活用して目的を達成するのがポイントです。

上で紹介した以外にも、組み込みクラスには様々なメソッドを定義してあるので、よく使う String, Array, Hash, Regexp, Fixnum, Float, Enumerable あたりのリファレンスマニュアルは一通り読んでサンプルを実行しておくと、実際に自分でプログラムを書くときに役立つでしょう。

作るオブジェクト指向

- 世間で言われてる方ののオブジェクト指向

オブジェクト指向とは？

ソフトウェアの設計や開発において、操作手順よりも操作対象に重点を置く考え方。

関連するデータの集合と、それに対する手続き (メソッド) を「オブジェクト」と呼ばれる一つのまとまりとして管理し、その組み合わせによってソフトウェアを構築する。

私的オブジェクト指向

- 汎用の整理術
- 脳内スタック節約術

- DRY 実現のための考え方
  - Don’t Repeat Yourself.

オブジェクトとは？

- 実際にある「もの」や「概念」
  - okkez, okkez のノート PC, などなど
- オブジェクトはデータと処理 (メソッド) をセットにしたもの
- オブジェクトは自分ができること (メソッド) を知っている

クラスとは？

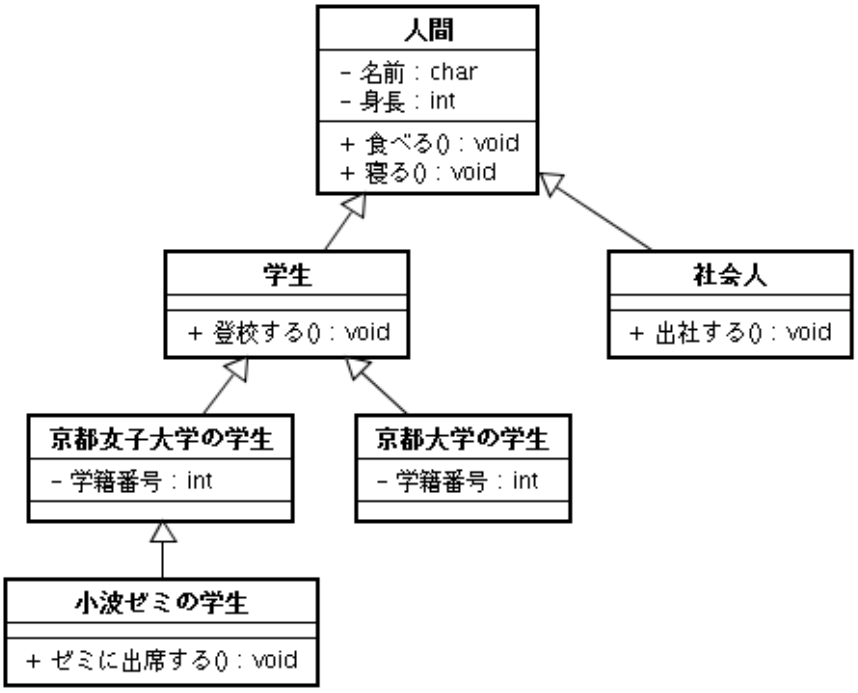
- オブジェクトのうち共通の性質を持った「くくり」
  - 人、大学、パソコン、などなど
- インスタンスを作る「雛型」

オブジェクト指向の三大要素

- 継承
- カプセル化
- ポリモルフィズム

継承

- 具体的なクラスは一般的なクラスの性質を引き継いでいる
  - 世の中のものをより一般的にくくることを抽象化
  - 世の中のものをより具体的にくくることを具象化

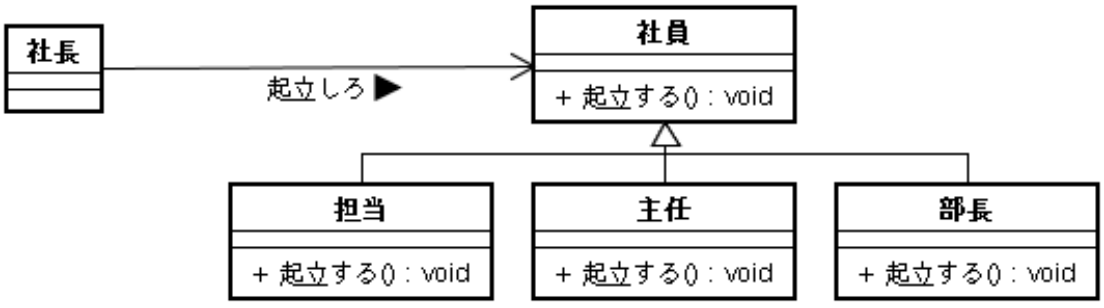


カプセル化

- 仕事 (メソッド) のやりかたは担当 (クラス) だけが知っていて、仕事を頼む人はその中身は知らない
  - オブジェクト内部の構造は外からはわからない -> 抽象データ型
  - オブジェクトの操作はメソッド経由で行う -> 変更に強くなる

ポリモルフィズム

- 究極奥儀
- 仕事を頼まれる側の種類が増えても、仕事を頼む方法は同じ



Ruby では？

コード例は最後にあります。

継承

- 使わなくても綺麗なプログラムは書ける
- 使った方がわかりやすい場合も多い

カプセル化

- インスタンス変数は外から変更できない
- 自動的にカプセル化は実現されている

コード例

```
01: class Person
02:   attr_reader :name, :birthday, :height
03:   def initialize(name, birthday, weight, height)
04:     @name = name
05:     @birthday = birthday
06:     @weight = weight
07:     @height = height
08:   end
09: end
10:
11: okkez = Person.new('okkez', '1979-09-23', 68, 168)
12: p okkez.name     # => "okkez"
13: p okkez.birthday # => "1979-09-23"
14: p okkez.height   # => 168
15: p okkez.weight   # => NoMethodError
```

ポリモルフィズム

- Duck Typing
  - アヒルのように歩き、アヒルのように鳴くものはアヒルだ
- IO, File, Tempfile, StringIO などなど
  - IO と File は継承関係にある -> File < IO
  - Tempfile は File クラスに処理を委譲している -> Delegate

  - StringIO は IO が持つメソッドを全て実装している -> Duck Typing

- 実例
  - logger -> 出力先が IO オブジェクト (標準出力、ファイルなど)

コード例

```
01: class Song
02:   def initialize(name, artist, duration)
03:     @name     = name
04:     @artist   = artist
05:     @duration = duration
06:   end
07:
08:   def to_s
09:     "#{@name} --  #{@artist} (#{@duration})"
10:   end
11: end
12:
13: class KaraokeSong < Song
14:   def initialize(name, artist, duration, lylic)
15:     super(name, artist, duration)
16:     @lylic = lylic
17:   end
18:
19:   def to_s
20:     super + " [#{@lylic}]"
21:   end
22: end
23:
24: song = Song.new('Ruby', 'Matz', '14 years')
25: karaoke_song = KaraokeSong.new('Ruby', 'Matz', '14 years', 'Hello, World!')
26:
27: puts song.to_s
28: puts karaoke_song.to_s
29:
30: if __FILE__ == $0 and ARGV.size > 0
31:   case ARGV.shift
32:   when /\Asong\z/i
33:     song = Song.new(*ARGV)
34:   when /\Akaraoke/i
35:     song = KaraokeSong.new(*ARGV)
36:   else
```

```
37:     puts 'must not happen!'
38:   end
39:   puts song.to_s
40: end
```

コードの解説

- initialize はクラスを new したときに呼ばれる特別なメソッド
- Song.initialize は 引数を三つ受け取る
- KaraokeSong.initialize は引数を四つ受け取る
- super はスーパークラスの同名メソッドを呼び出す

演習問題

わからないことや知らないことがあったら、TA や周囲の人に相談してみましょう。

演習 0 - ウォーミングアップ

以下のプログラムを実行してみましょう。

```
01: p ARGV
02: p [ARGV[0], ARGV[0].class]
03: p [ARGV[1], ARGV[1].class]
04: p [ARGV[2], ARGV[2].class]
```

実行例

```
$ ruby sample07.rb aaa 100
```

演習 1 - 社長命令・起立！

- 社長の席のついたての向こうに誰か社員がいます。
- 社長は、社員なら誰でもいい用事を思いだし、声をかけます。
  - 「わしは社長や。誰か知らんけどそこにいる君、立ちなさい」
- 呼ばれた人はそれぞれなりに起立します。
  - 担当が普通に起立しました。
  - 主任がすばやく立ちました。
  - 部長がだるそうに立ちました。

実行例

```
$ ruby shacho1.rb Tanto
```
担当が普通に起立しました。

```
$ ruby shacho1.rb Shunin
```
主任がすばやく立ちました。

```
$ ruby shacho1.rb Bucho
```
部長がだるそうに立ちました。

- 社員のコード (shain1.rb) を書きましょう
  - Shain クラスを定義し、それを継承して Tanto, Shunin, Bucho クラスを作ります。

    class Shain

    ```
    def standup
    end
    ```

    end

    class Tanto < Shain

    ...

    end

演習 2 - 給料はいくら？

- 社長からさらに命令が出ました。
  - 「誰か知らんけど基本給を教えるから、そこから計算して君の給料がいくらか答えなさい」
- 給料計算のルール
  - 担当：基本給と同じ
  - 主任：基本給 * 2
  - 部長：基本給 * 3

実行例

```
$ ruby shacho2.rb Tanto 100
```
担当が普通に起立しました。
給料は 100 円です。

```
$ ruby shacho2.rb Shunin 100
```
主任がすばやく立ちました。
給料は 200 円です。

- shain2.rb の Shain クラスに、基本給から給料を計算するメソッドを追加します。

```
class Shain
  def standup
  end

  def kyuryo(kihonkyu)
  end
end
```

- Tanto, Shunin, Bucho クラスの kyuryo メソッドを定義しましょう。

```
class Tanto < Shain
  def kyuryo(kihonkyu)
    return ...
  end
end
```

演習 3 - 取締役を追加

- shain3.rb に取締役を追加しましょう。
  - 取締役はふんぞりかえって立ちました。
  - 取締役の給料は「基本給 * 4」です。

実行例

```
$ ruby shacho3.rb Torishimariyaku 100
取締役はふんぞりかえって立ちました。
給料は 400 円です。
```

演習 4 - ボーナスはいくら？

- 基本給をセットするメソッド kihonkyu= を定義しましょう。
- ボーナスを返すメソッド bonus を定義しましょう。
- ボーナスは社員だれでも給料の 4 倍です。

実行例

```
$ ruby shacho4.rb Tanto 100
担当が普通に起立しました。
給料は 100 円です。
ボーナスは 400 円です。

$ ruby shacho4.rb Shunin 100
主任がすばやく立ちました。
給料は 200 円です。
ボーナスは 400 円です。
```

ヒント

- コマンドライン引数は配列 ARGV に文字列として格納されています。
- 文字列を数値に変換するには String#to_i というメソッドを使用します。

まとめ

- オブジェクト
  - 実際にある「もの」や「概念」
- クラス
  - オブジェクト共通の性質を持った「くくり」
- 継承
  - いろいろなクラス定義の共通化
- カプセル化
  - データ構造が変わっても、仕事の頼み方は同じ
- ポリモルフィズム
  - 仕事を頼まれる側の種類が増えても、仕事の頼み方は同じ

参考文献

オブジェクト脳のつくり方

http://www.seshop.com/detail.asp?pid=4115

初めてのプログラミング

http://www.oreilly.co.jp/books/4873112923/

プログラミング Ruby 第 2 版 言語編

http://ssl.ohmsha.co.jp/cgi-bin/menu.cgi?ISBN=4-274-06642-8

ソフトバンク クリエイティブの本：たのしい Ruby 第 3 版

http://www.sbcr.jp/books/products/detail.asp?sku=4797357400

今後の情報源

公式 Web サイト

http://www.ruby-lang.org/

ReferenceManualRenewalProject - 概要 - Ruby Issue Tracking System

http://redmine.ruby-lang.org/projects/show/rurema

Ruby reference manual (beta)

http://doc.okkez.net/

日本 Ruby の会

http://jp.rubyist.net/

Rubyist Magazine

http://jp.rubyist.net/magazine/

okkez weblog

http://typo.okkez.net/