Ruby 初級者向けレッスン第 2 回

かずひこ@株式会社 ネットワーク応用通信研究所

2005 年 8 月 20 日

メニュー

- Ruby で学ぼうオブジェクト指向
  - オブジェクト
  - クラス
  - 継承
  - カプセル化
  - ポリモルフィズム

オブジェクトとは？

- 実際にある「もの」や「概念」
  - かずひこ、ネットワーク応用通信研究所、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、会社、などなど
- オブジェクト (インスタンス) を作る「雛型」
- クラスの例

```
class Person
  def initialize(myname)
    @name = myname
  end
  def eat
    puts "食べてます。"
  end
end
```


- インスタンスを作る

```
kaz  = Person.new('かずひこ')
matz = Person.new('まつもと')
```

- 変数やメソッドをまとめる

```
kaz.eat  # かずひこが「食べる」
matz.eat # まつもとさんも「食べる」
```

- インスタンス変数
  - '@' で始まる変数
  - グローバル変数と違ってクラスの外では参照できない
  - ローカル変数と違ってメソッドの外でも保持される
  - インスタンスごとに違う値を持てる

```
class Person
  def initialize(myname)
    @name = myname
  end
end
```

- インスタンスメソッド
  - class 文の中に定義したメソッド
  - インスタンス (オブジェクト) に対して呼べるメソッド

```
class Person
  def name #  ←インスタンスメソッド
    return @name # ←インスタンス変数
  end
end
```

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
  - 会社員 < 人間 < 動物、などなど
- 継承のもとになるクラスをスーパークラスという
- 継承により作られた新しいクラスをサブクラスという
- いろいろなクラス定義の共通化
  - クラス定義の共通部分をスーパークラスにまとめる
- スーパークラスで定義した性質は自動的にサブクラスに定義される
- 継承によってできること
  - 新しい機能を追加する
  - 既存の機能を置き換える
  - 既存の機能に処理を追加する
- 新しい機能を追加する

```
class Animal
  def eat # 動物は食べる
    puts "食べてます。"
  end
end

class Person < Animal # 継承して、人は食べる
  def walk # 人は歩く
    puts "歩いてます。"
  end
end
```

- 既存の機能を置き換える

```
class Shain < Person
  def eat
    puts "社員食堂で食べてます。"
  end
end
```

- 既存の機能に処理を追加する


```
class Shain < Person
  def walk
    print "セカセカと"
    super # ←スーパークラスの同名メソッドを呼び出す
  end
end

kaz = Shain.new('かずひこ')
kaz.walk #=> "セカセカと歩いてます。"
```

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
  - オブジェクトの操作はメソッド経由で行う
- データ構造が変わっても、仕事の頼み方は同じ

```
# 変更前
class Person
  def name
    return @name
  end
end

puts matz.name #=> "まつもと"

# 変更後
class Person
  def name
    return "#{@first_name} #{@last_name}"
  end
end

# 呼び出し側は同じ
puts matz.name #=> "まつもと ゆきひろ"
```

ポリモルフィズム

- 仕事を頼む側の種類が増えても、仕事の頼み方は同じ

- **呼び出し側を共通化する**

```
kaz  = Person.new
poti = Dog.new

kaz.eat  # かずひこは「食べる」
poti.eat # ポチも「食べる」
```

- **もしポリモルフィズムがなければ？**
  - **仕事を頼む側の種類が増えたら大変！**

```
def eat(obj)
  case obj
  when Person
    puts "箸でつかんで食べてます。"
  when Dog
    puts "立ちながら食べてます。"
  when Cow
    puts "吐きながら食べてます。"
  end
end
```

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。**
- **社長のコード** (shacho1.rb)

```
require 'shain1'

class Shacho
  def meirei(type)
    case type
    when "Tanto"
      shain = Tanto.new
    when "Shunin"
      shain = Shunin.new
    when "Bucho"
      shain = Bucho.new
    end

    shain.standup
  end
end

shacho = Shacho.new
shacho.meirei(ARGV[0])
```


- **動作例**

```
$ 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。

$ ruby shacho2.rb Bucho 100
部長がだるそうに立ちました。
給料は 300 円です。
```

演習 2 - 給料はいくら？

- 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
```

- **社長のコード** (shacho2.rb)

```
require 'shain2'

class Shacho
  def meirei(type, kihonkyu)
    case type
    when "Tanto"
      shain = Tanto.new
    when "Shunin"
      shain = Shunin.new
    when "Bucho"
      shain = Bucho.new
    end

    shain.standup
    puts "給料は #{shain.kyuryo(kihonkyu)} 円です。"
  end
end

shacho = Shacho.new
shacho.meirei(ARGV[0], ARGV[1].to_i)
```

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。
```

- **社長のコード** (shacho3.rb)

```
require 'shain3'

class Shacho
  def meirei(type, kihonkyu)
    case type
    when "Tanto"
      shain = Tanto.new
    when "Shunin"
```

```
      shain = Shunin.new
    when "Bucho"
      shain = Bucho.new
    when "Torishimariyaku"
      shain = Torishimariyaku.new
    end

    shain.standup
    puts "給料は #{shain.kyuryo(kihonkyu)} 円です。"
  end
end

shacho = Shacho.new
shacho.meirei(ARGV[0], ARGV[1].to_i)
```

演習 4 - ボーナスはいくら？

- ボーナスは社員だれでも「基本給 * 4」です。
- 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。
- 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。

$ ruby shacho4.rb Bucho 100
部長がだるそうに立ちました。
給料は 300 円です。
ボーナスは 400 円です。

$ ruby shacho4.rb Torishimariyaku 100
取締役がふんぞりかえって立ちました。
給料は 400 円です。
ボーナスは 400 円です。
```


- 社長のコード (shacho4.rb)

```
require 'shain4'

class Shacho
  def meirei(type, kihonkyu)
    case type
    when "Tanto"
      shain = Tanto.new
    when "Shunin"
      shain = Shunin.new
    when "Bucho"
      shain = Bucho.new
    when "Torishimariyaku"
      shain = Torishimariyaku.new
    end

    shain.standup
    shain.kihonkyu = kihonkyu
    puts "給料は #{shain.kyuryo} 円です。"
    puts "ボーナスは #{shain.bonus} 円です。"
  end
end

shacho = Shacho.new
shacho.meirei(ARGV[0], ARGV[1].to_i)
```

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
  - 仕事を頼む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**』 ISBN:4798104183
- 『**たのしい** Ruby』 ISBN:4797314087
- 『**プログラミング** Ruby』 ISBN:4894714531

今後の情報源

公式 Web サイト
: http://www.ruby-lang.org/

リファレンスマニュアル
: http://www.ruby-lang.org/ja/man/

日本 Ruby の会
: http://jp.rubyist.net/

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

ふぇみにん日記
: http://kazuhiko.tdiary.net/