プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 10

前回の課題に関して

```
class Point { ...
  public String toString () {
    return "Point...";
  }
}
class ColoredPoint extends Point { ...
  public String toString () {
    return "ColoredPoint...";
  }
}
```

Override (上書き):

- 親クラス (Point) を継承した子クラス (ColoredPoint) では、メソッド toString の定義 (実装) をそのままもらうのではなく、違うものに書きかえている。
- toString の引数の個数、型、返すものの型は、まったく同じ。

Override in Java

```
class Test1 {
  public static void main(String args[]) {
  Point p = new Point(10.0, 20.0);
  ColoredPoint cp = new ColoredPoint(10.0, 20.0, 3);

  System.out.println(p.toString());  => 親の toString
  System.out.println(cp.toString()); => 子の toString
```


Override with Cast in Java

(1) 親クラスの変数に、子クラスのオブジェクトを代入してもよい。

```
class Test1 {
  public static void main(String args[]) {
  ColoredPoint cp = new ColoredPoint(10.0, 20.0, 3);
  Point p = cp;
  System.out.println(p.toString());  => 子の toString
```

cp.toString() について、動的にルックアップしている。

(2) 子クラスの変数に、親クラスのオブジェクトを代入するのはいけない。

```
class Test1 {
  public static void main(String args[]) {
  Point p = new Point(10.0, 20.0;
  ColoredPoint cp = p; => コンパイルエラー
```

Override in Java

- (型に関する) インタフェースは継承。
- 実装は書き換える。

Overload in Java

```
class Test1 {...
  public static void foo(Point p) {
   System.out.println("foo-1:" + p.toString());
  }
  public static void foo(ColoredPoint cp) {
   System.out.println("foo-2:" + cp.toString());
  }
  public static void foo(Point p, ColoredPoint cp) {
   System.out.println("foo-3:" + p.toString() + ":" + cp.toString()
  }
}
```

Overload:

- 1 つのメソッド名に複数の実装。
- 引数の個数、引数の型、返す値の型で区別。

Overload with Cast in Java

親クラスの変数に、子クラスのオブジェクトを代入してもよい。

```
Point p = new ColoredPoint(...);
foo(p);          ==> foo-1 が呼ばれる。
```

変数 p の中は、子クラスのオブジェクトであるが、どの foo が呼ばれるかは、静的に (変数の型等で) 決定されるため、ここでは foo-2 でなく foo-1 が呼ばれる。

(一方、foo の中で呼ばれる toString は動的に決定されるため、子クラスの toString が呼ばれる。

Override and Overload in Java

```
class Test4 {
  public static void foo(Point p) {
   System.out.println("foo-1:" + p.toString());
  }
}
class Test5 extends Test4 {
  public static void main(String args[]) {
    Point p = new Point(10.0, 20.0);
    ColoredPoint q = new ColoredPoint(10.0, 20.0, 5);
    Point r = q;
    foo(r);      ==> 何が返るか？
  }
  public static void foo(ColoredPoint cp) {
    System.out.println("foo-2:" + cp.toString());
  }
}
```

Override vs Overload in Java

Java は静的型付きオブジェクト指向言語

- Override: 親クラスと子クラスで、同じ名前・型で実装が異なるメソッド (この場合は toString メソッド) を持つこと。
  - 「どの型 (クラス) の変数か」ではなく、「実行時に、その変数にどのクラスのオブジェクトがはいっているか」によって、使われるメソッドが決まる (動的ルックアップ)。
- Overload: 同一クラス内で 1 つのメソッド名に、複数の定義を与えること。
  - 複数の定義は、引数パターン (型，個数，順番) が異なる。
  - どのメソッド定義が使われるかは、メソッド呼び出しの引数パターンにより (つまり静的に) 決定される。

スクリプト言語

Scripting Language (Language for Scripting)
歴史的には:

- スクリプト＝台本
- Computer Science では，(アプリケーションソフトウェアに対する)「指示や命令の列」
- スクリプト言語は，コマンドラインで実行されるような簡易な命令を記述できる言語で，インタープリタで実行されるものを意味することが多かった.
- 本格的なプログラム言語の対義語.
- 代表例: Unix の shell (shell のプログラムを shell script と言う)
- その他: Unix の awk, sed, ...

現在:

- PHP, Perl, Python, Ruby, JavaScript ... など多数の汎用の言語。
- スクリプト言語とは一体何だろう?

スクリプト言語

現在では,「スクリプト言語とは何か？」「この言語はスクリプト言語か否か」という問に正確に答えるのは難しい．その代わりに...

- 動的言語 (dynamic language): 通常は静的 (コンパイル時) に行われること (たとえば，関数やクラスの定義，型付けなど) の多くを動的に (実行時に) 行う言語.「この言語は動的言語か」を明確に決めることはできないが,「Ruby は Java に比べて動的である」ということは言える.
- 動的型付け言語: 型付けが静的に行われないで，実行時に行われる言語．(1+"abc" は実行時エラー)
- 領域特化言語 (domain specific language, DSL): 特定の領域 (domain) の問題を解くために適した (簡潔であり，汎用ではない) 言語．(例: Java で turtle graphics をするプログラムを書いたとき，そのプログラムのユーザは,「右へ 3cm 行け」「左に 90 度回転せよ」といった命令を発行して graphics を行う．後者の命令群から構成される言語が,この際の DSL である．)

スクリプト言語 (続き)

「スクリプト言語」の説明 (定義にはなっていない)：

- 「プログラムをすぐ書ける」言語 (生産性の高い言語，プロトタイピングに適した言語)
- インタープリタで動かすことが多い言語。
- 実行性能よりも、書きやすさ (速く書けること) を重視。

論理型プログラム言語とは

- 「推論＝計算」という考えで設計された言語.
- プログラムは，事実，あるいは，推論規則 (事実から事実を導く) として記述.
- どのように推論するかの手順は，記述しない.
- 宣言型プログラム言語 (Declarative Programming Language) の一種.

Prolog プログラミング-1

プログラムは別紙参照.
Alice, Bob, Charlie, David, Eliza, Fritz, George, Hillary の 8 人の関係.

Prolog プログラミング-2

```
?- is_mother(X, charlie).
X = alice

?- is_mother(alice, X).
X = charlie n
X = eliza.

?- is_husband(alice, X).
false.

?- is_father(bob, X).
X = charlie n
X = eliza.
```


Prolog プログラミング-3

```
?- is_parent(X,eliza).
X = alice n
X = bob n
false.

?- is_grandparent(X,Y).
X = alice,
Y = george n
X = bob,
Y = george n
X = bob,
Y = george n
false.
```

Prolog プログラミング-4

```
?- is_relative(alice,X).
X = alice n
X = bob n
X = charlie n
X = eliza n
X = george n
false.

?- is_relative(eliza, X).
X = eliza n
X = fritz n
X = george n
X = alice n
X = bob n
false.
```


Prolog プログラミング-5

解が無限にたくさんある例。

```
?- is_superrelative(alice, X).
X = alice n
X = bob n
X = charlie n
X = eliza n
X = george n
X = alice n
X = bob n
X = charlie n
X = eliza n
X = george n
X = alice n
X = bob n
```


Prolog プログラミング-6

```
?- fact(10, X).
X = 3628800.

?-  append([a, b, c], [], X ).
X = [a, b, c].
?-  append([a, b, c], [d, e, f], X ).
X = [a, b, c, d, e, f].
?- append([a, b, c], X, [a, b, c, d, e, f]).
X = [d, e, f].
```


Prolog プログラミング-7

append(X, Y, [a, b, c, d, e, f]) とやると何が返ってくるだろうか？

Prolog プログラミング-8

```
?- append(X, Y, [a, b, c, d, e, f]).
X = [],
Y = [a, b, c, d, e, f] n
X = [a],
Y = [b, c, d, e, f] n
X = [a, b],
Y = [c, d, e, f] n
X = [a, b, c],
Y = [d, e, f] n
X = [a, b, c, d],
Y = [e, f] n
X = [a, b, c, d, e],
Y = [f] n
X = [a, b, c, d, e, f],
Y = [] n
false.
```


Prolog プログラミング

宣言型プログラム言語 (Declarative Programming Language) の一種.

- 推論＝計算.
- プログラム＝事実，あるいは，推論規則.
- ゴール＝その事実が成立するかどうかを質問.
- 推論をする手順は，記述しない.

特徴.

- プログラムの目的=ゴールが成立するかどうか，を求める.
- 求解＝ゴール中の変数のある値に対して，ゴールが成立するか？
- 複数の解があることもある.
- あらゆる可能性を網羅的に探索する．双方向計算が可能.
- 一階述語論理のサブセット (Horn 節論理) に対応.

論理型プログラム言語

- Kowalski 1974 が提唱.
- Colmerauer 1973 が Prolog の最初の処理系.
- 日本の ICOT(第５世代コンピュータ・プロジェクト) が大きな貢献.
- 現在でも，知識表現，帰納論理プログラミングなどの分野で活躍.

Prolog = Programming in Logic

Prolog 処理系

ここでは Linux 上の SWI-Prolog 処理系を使った。
(http://www.swi-prolog.org/)

```
% swipl
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.58)
Copyright (c) 1990-2008 University of Amsterdam.
...

For help, use ?- help(Topic). or ?- apropos(Word).

?- [test-prolog].   (←ここで test-prolog.pl ファイルの読み込み)
[test-prolog].
...
% test-prolog compiled 0.00 sec, 5,280 bytes
true.

?-  append(X, Y, [a, b, c, d, e, f]).
X = []
Y = [a, b, c, d, e, f]
```

まとめ

- 「推論」を計算過程と見なしたプログラミング言語.
- その他にも様々な (驚くような) 仕組みを計算の原理に使ったものがあり得る。
  - 量子コンピューティング, DNA コンピューティング, ...

Quiz

「事実やルール (推論規則) を記述するだけで，プログラムとして動く」という言語は，メリットもデメリットもある.

- メリット: どのルールをどう使ったら，ゴールとなる事実を導けるか (効率的に導くにはどうしたらよいか) を，プログラマは考えなくてよい．解決したい問題を正確に記述するだけでよい．(プログラミングが楽になる．)
- デメリット: 常に Prolog 処理系が決めた順序で解を探索するので，効率が必ずしも良くない．(効率を良くしようと思うと，処理系の動きを把握していないといけない．)

このような言語が，現実に有用な場面としてどんなものがあるか、想像して書きなさい.
参考: 「プログラムの仕様」がそのまま動くものを、executable specification(実行可能な仕様) という。