ソフトウェア基礎論配布資料 (9)
オブジェクト計算 (3): Featherweight Java

五十嵐 淳
京都大学 大学院情報学研究科知能情報学専攻
e-mail: igarashi@kuis.kyoto-u.ac.jp

平成 16 年 12 月 15 日

1 クラスに基づくオブジェクト指向言語

1.1 クラス

多くのオブジェクト指向言語では，**クラス**(*class*) という仕組みによって，類似する構造を持つオブジェクトを簡潔に記述できるようになっている．例えば，点オブジェクトは，状態である座標値こそ各点毎に違うものの，メソッドに関しては同一であることが想定されるため，初期座標値から点オブジェクトを生成できる仕組みがあると便利である．クラスは，このような共通の構造を記述するためのものである．クラスには通常，

- 状態を表すデータ (の名前) の定義
- メソッド群の定義
- オブジェクトを生成する際の状態の初期化の方法 (**コンストラクタ**(*constructor*))

が記述される．とりあえず，例の記述のために Java 風の言語を用いると，(一次元) 点オブジェクトのクラス `Point` は

```
class Point {
  field x;
  Point(x') { this.x = x'; }
  method getx() { return this.x; }
  method setx(x') { return new Point(x'); }
  method move(dx) { return this.setx(this.getx() + dx); }
}
```

と記述できる．ここでは `this` を自分自身を表すための特別な変数として用いている．また, `new Point(...)` のような式で, `Point` クラスの新しいオブジェクトが生成される．この時, コンストラクタ (`Point(x') {...}`) が呼ばれて，フィールドの初期化 `this.x = x';` が行なわれる．メソッド名の直後の括弧はメソッドの引数の宣言である．

このようなクラスを中心としてプログラムを記述するオブジェクト指向言語を**クラスに基づく言語**(*class-based (object-oriented) language*) ということがある．

1.2 サブクラスと継承

またクラスに基づく言語では，類似するクラスの記述，特に，あるクラスが別のクラスの機能を拡張して (より具体的には既存のクラスのフィールド宣言・メソッド定義を丸ごと複製して) 得られるような場合の記述を助ける仕組みが備わっている．これが**継承**(*inheritance*) と呼ばれる仕組みである．例えば，色付きの点オブジェクトのためのクラス CPoint を継承を使って以下のように定義することができる．

```
class CPoint extends Point {
  field color;
  ColorPoint(color',x') { super(x'); this.color = color'; }
  method getc() { return this.color; }
  method setx(x') { return new CPoint(x',White); }
}
```

このクラス定義のポイントは以下の通りである．

- extends Point と書くことで，Point クラスのフィールド/メソッド/コンストラクタ定義を継承し，CPoint オブジェクトは x フィールド, getx, move メソッドを持つことになる．CPoint は Point の**サブクラス・派生クラス**(*subclass, derived class*) である，といったりする．逆に Point は CPoint の**親クラス**(*superclass*) である，ということがある．
- color, getc は追加されたフィールド/メソッドである．
- 追加されたフィールドの初期化コードがコンストラクタに記述されている．super(x') は，継承した x フィールドの初期化のために親クラスのコンストラクタを呼び出すための，特別な記法である．
- setx は親クラスの定義を使わずに独自のものをここで定義している．これをメソッド定義の**上書き**(*override*) と呼ぶ．CPoint クラスのオブジェクトに対して setx を呼んだ時には新しい定義が呼び出されるのは勿論，継承された move メソッドを呼んだ場合にも，その内部から呼び出される setx は新しい定義になる．

このようにして，クラスを拡張して新しい機能を追加 (場合によっては上書きによる変更) をして，新しいクラスを定義することができる．

ここまでの概念を形式化したような計算体系を ς 計算と同様に記述することは可能であるが，ここでは型システムのある体系を最初から考えよう．

1.3 クラスと型・サブクラスと部分型

クラスの備わった言語においても，ς 計算と同様な手法で，オブジェクトの型を，メソッド (フィールド) の名前とそれが返す値の型の集まりと考えることができる．しかし，大抵のクラスに基づく言語では，同一のクラスから生成されたオブジェクトは同一名のメソッド・フィールドを持つ，という性質を利用して，**クラスの名前を型として利用する**ことが多い．メソッド呼び出しの返り値の型に関しては，ς 計算では self パラメータの型として宣言されていたが，多くの言語では，代わりに各フィー

ルド・メソッド宣言に型の情報を付加する．例えば，上の Point クラスに，型情報を加えるとすると，以下のような定義になる．(int は整数を表す型とする．)

```
class Point {
  int x;
  Point(int x') { this.x = x'; }
  int getx() { return this.x; }
  Point setx(int x') { return new Point(x'); }
  Point move(int dx) { return this.setx(this.getx() + dx); }
}
```

x フィールドは整数を保持するものであり，setx は int を引数として受け取って Point を返すことが記述されている．

また，より多くのメソッドを持つオブジェクトが，少ないメソッドを持つオブジェクトの代わりができるように，クラスに基づく言語では，サブクラスから生成されるオブジェクトは親クラスのオブジェクトの代わりができる．これを型の立場から捉えると，自然に，親クラス・子クラスの関係を部分型関係として捉えることができる．つまり，CPoint クラスのオブジェクトを Point が必要な関数に渡したりすることが許されそうである．

ただし，継承関係を部分型関係として考えるためには，継承/上書きの仕方に一定の制限が必要である．例えば，親クラスでは，整数を引数に取ったメソッドを，上書きする時に文字列を受け取るものとすると，おかしなことがおこるので，上書きするメソッドは元の定義とある程度「マッチする」型である必要がある．(ここでは一番制限のきつい，同一の型で上書きすべし，という規則を設けることにする．この制限は ς 計算の部分型規則に対応している．)

2 Featherweight Java

Featherweight Java [1, 参考図書 1 の 19 章] (以下 FJ) は，上のようなクラス機構のエッセンスを Java 風の文法で形式化した計算体系である．

2.1 文法

B, C, D はクラス名を表すメタ変数，f, m, x はフィールド・メソッド・メソッドパラメータの名前を表すメタ変数，L, K, M, e はそれぞれクラス，コンストラクタ，メソッド，式を表すメタ変数，v, w は値を表すメタ変数である．上付き線は列を表し，$\overline{C}$ は (適当な n に対して)$C_1, \ldots, C_n$ の省略である．($n = 0$ の場合もある．) 列に現れる名前には重複を認めない．($\overline{C}\ \overline{f}$; は，$C_1\ f_1; \cdots C_n\ f_n$; の略であり，フィールド名 f_i にのみ重複を認めない．)

$$
\begin{array}{lcl}
L & ::= & \texttt{class}\ C\ \texttt{extends}\ C\ \texttt{\{}\overline{C}\ \overline{f}\texttt{;}\ K\ \overline{M}\texttt{\}} \\
K & ::= & C\texttt{(}\overline{C}\ \overline{f}\texttt{)\{super(}\overline{f}\texttt{);}\ \texttt{this.}\overline{f}\texttt{=}\overline{f}\texttt{;\}} \\
M & ::= & C\ m\texttt{(}\overline{C}\ \overline{x}\texttt{)\{}\ \texttt{return}\ e\texttt{;}\ \texttt{\}} \\
e & ::= & x \mid \texttt{this} \mid e.f \mid e.m\texttt{(}\overline{e}\texttt{)} \mid \texttt{new}\ C\texttt{(}\overline{e}\texttt{)} \\
v & ::= & \texttt{new}\ C\texttt{(}\overline{v}\texttt{)}
\end{array}
$$

Featherweight Java のプログラムは，クラスの集合と式の組 $(\overline{L}, e)$ である．この式は初めに実行される，いわゆる `main` メソッドに相当する．

式に対する代入は $[\overline{x} \mapsto \overline{e}]e_0$ と表記し，同時に複数の変数を置き換える．FJ の式には束縛変数がないため定義は非常に単純なものになる．よって定義の詳細は省略する．

Java では，全てのクラスの先祖のクラスとして `Object` というクラスが仮定されている．FJ では，`Object` をフィールドもメソッドも何も持たず，クラスの集合にも含まれない特別なクラスとして扱う．

以下，プログラムとして，同一名のクラスは複数存在しない，`Object` は明示的には定義されていない，`extends` 関係は循環していない，プログラムに現われるクラス名は `Object` であるかクラス集合で定義されている，という条件を満たすものだけを考える．

以下で定義される規則は，厳密にいうと，何らかのクラス集合を固定することで定義されるものだが，表記を単純にするために省略している．

2.2 簡約規則

簡約は，式の二項関係 $e \longrightarrow_v e'$ として定義するが，そのために必要な補助関数として，クラス名からそのフィールド名 (とその型) を問い合わせるための $\mathit{fields}(C)$ とクラス名とメソッド名からその定義を問い合わせるための $\mathit{mbody}(m, C)$ を以下の規則で定義する．これらの関数がフィールド・メソッドの継承を表現している．

$$\mathit{fields}(\texttt{Object}) = \bullet$$

$$\frac{\texttt{class}\ C\ \texttt{extends}\ D\ \texttt{\{}\overline{C}\ \overline{f}\texttt{;}\ K\ \overline{M}\texttt{\}} \qquad \mathit{fields}(D) = \overline{D}\ \overline{g}}{\mathit{fields}(C) = \overline{D}\ \overline{g}, \overline{C}\ \overline{f}}$$

$$\frac{\texttt{class}\ C\ \texttt{extends}\ D\ \texttt{\{}\overline{C}\ \overline{f}\texttt{;}\ K\ \overline{M}\texttt{\}} \qquad B\ m\texttt{(}\overline{B}\ \overline{x}\texttt{)\{ return}\ e\texttt{; \}} \in \overline{M}}{\mathit{mbody}(m, C) = \lambda\overline{x}.e}$$

$$\frac{\texttt{class}\ C\ \texttt{extends}\ D\ \texttt{\{}\overline{C}\ \overline{f}\texttt{;}\ K\ \overline{M}\texttt{\}} \qquad m \notin \overline{M} \qquad \mathit{mbody}(m, D) = \lambda\overline{x}.e}{\mathit{mbody}(m, C) = \lambda\overline{x}.e}$$

$m \notin \overline{M}$ は m という名前を持つメソッドが $\overline{M}$ の中にない，ということを示す記法である．

2.2.1 定義: 簡約関係 $e \longrightarrow_v e'$ を以下の規則で定義する．

$$\frac{\mathit{fields}(C) = \overline{C}\ \overline{f}}{\texttt{new}\ C\texttt{(}\overline{v}\texttt{)}\texttt{.}f_i \longrightarrow_v v_i} \qquad \text{(R-Field)}$$

$$\frac{\mathit{mbody}(m, C) = \lambda\overline{x}.e_0}{\texttt{new}\ C\texttt{(}\overline{v}\texttt{)}\texttt{.}m\texttt{(}\overline{w}\texttt{)} \longrightarrow_v [\overline{x} \mapsto \overline{w}, \texttt{this} \mapsto \texttt{new}\ C\texttt{(}\overline{v}\texttt{)}]e_0} \qquad \text{(R-Invk)}$$

$$\frac{e_0 \longrightarrow_v e_0'}{e_0.f \longrightarrow_v e_0'.f} \qquad \text{(RC-Field)}$$

$$\frac{e_0 \longrightarrow_v e_0'}{e_0.m(\overline{e}) \longrightarrow_v e_0'.m(\overline{e})} \quad \text{(RC-Invk-Recv)}$$

$$\frac{e_i \longrightarrow_v e_i'}{v_0.m(v_1, \ldots, v_{i-1}, e_i, \ldots, e_n) \longrightarrow_v v_0.m(v_1, \ldots, v_{i-1}, e_i', e_{i+1}, \ldots, e_n)} \quad \text{(RC-Invk-Arg)}$$

$$\frac{e_i \longrightarrow_v e_i'}{\texttt{new } C(v_1, \ldots, v_{i-1}, e_i, \ldots, e_n) \longrightarrow_v \texttt{new } C(v_1, \ldots, v_{i-1}, e_i', e_{i+1}, \ldots, e_n)} \quad \text{(RC-New-Arg)}$$

オブジェクト `new` C`(...)` の括弧内には，C(とその先祖クラス) で宣言されたフィールドの順にそれらの値が並ぶように体系 (特に型システム) が作られている．また R-Invk 規則にみられる `this` への代入は，ς 計算のメソッド呼び出しの規則と類似のものである．

2.3 型システム

既に述べたように，型としてクラス名を用い，クラス名の間に部分型関係 $C \leq D$ を導入する．また，型から，それが属するオブジェクトが持つ，メソッドの型を問い合わせる関数 $\mathit{mtype}(m, C)$ を定義する．(フィールドの型情報に関しては既に定義した $\mathit{fields}(C)$ で得られる．)

$$C \leq C \qquad \frac{C \leq D \qquad D \leq E}{C \leq E} \qquad \frac{\texttt{class } C \texttt{ extends } D \texttt{ \{...\}}}{C \leq D}$$

$$\frac{\texttt{class } C \texttt{ extends } D \texttt{ \{}\overline{C}\ \overline{f}\texttt{; } K\ \overline{M}\texttt{\}} \qquad B_0\ m\texttt{(}\overline{B}\ \overline{x}\texttt{)\{ return } e\texttt{; \}} \in \overline{M}}{\mathit{mtype}(m, C) = \overline{B} \to B_0}$$

$$\frac{\texttt{class } C \texttt{ extends } D \texttt{ \{}\overline{C}\ \overline{f}\texttt{; } K\ \overline{M}\texttt{\}} \qquad m \notin \overline{M} \qquad \mathit{mtype}(m, D) = \overline{B} \to B_0}{\mathit{mtype}(m, C) = \overline{B} \to B_0}$$

型環境 Γ は変数と型の組の (変数名に重複のない) 列である．式・メソッド・クラスに関して型付け関係はそれぞれ $\Gamma \vdash e : C$, M ok in C, L ok と表記する．型付け規則は以下の通り．

$$\Gamma, x : C \vdash x : C \quad \text{(T-Var)}$$

$$\frac{\Gamma \vdash x : C \qquad x \neq y}{\Gamma, y : D \vdash x : C} \quad \text{(T-Weak)}$$

$$\frac{\Gamma \vdash e_0 : C_0 \qquad \mathit{fields}(C_0) = \overline{C}\ \overline{f}}{\Gamma \vdash e_0.f_i : C_i} \quad \text{(T-Field)}$$

$$\frac{\Gamma \vdash e_0 : C_0 \qquad \mathit{mtype}(m, C_0) = \overline{D} \to C \qquad \Gamma \vdash \overline{e} : \overline{C} \qquad \overline{C} \leq \overline{D}}{\Gamma \vdash e_0.m(\overline{e}) : C} \quad \text{(T-Invk)}$$

$$\frac{\mathit{fields}(C) = \overline{D}\ \overline{f} \qquad \Gamma \vdash \overline{e} : \overline{C} \qquad \overline{C} \leq \overline{D}}{\Gamma \vdash \texttt{new}\ C(\overline{e}) : C} \tag{T-New}$$

$$\frac{\begin{array}{c}\overline{x} : \overline{C}, \texttt{this} : C \vdash e_0 : E_0 \qquad E_0 \leq C_0 \\ \texttt{class}\ C\ \texttt{extends}\ D\ \texttt{\{...\}} \\ \text{if } \mathit{mtype}(m, D) = \overline{D} \to D_0, \text{ then } \overline{C} = \overline{D} \text{ and } C_0 = D_0\end{array}}{C_0\ m(\overline{C}\ \overline{x})\texttt{\{ return } e_0\texttt{; \}} \text{ ok in } C} \tag{T-Method}$$

$$\frac{K = C(\overline{D}\ \overline{g},\ \overline{C}\ \overline{f})\texttt{\{super(}\overline{g}\texttt{); this.}\overline{f}\texttt{=}\overline{f}\texttt{;\}} \qquad \mathit{fields}(D) = \overline{D}\ \overline{g} \qquad \overline{M} \text{ ok in } C}{\texttt{class}\ C\ \texttt{extends}\ D\ \texttt{\{}\overline{C}\ \overline{f}\texttt{;}\ K\ \overline{M}\texttt{\}} \text{ ok}} \tag{T-Class}$$

この体系には subsumption のための規則がないが，その代わりに，メソッド呼び出し・オブジェクト生成時に subsumption と同等のことができるように作られている．規則 T-Method は ς 計算の T-Object と比べてみると面白い．最後の前提条件は，上書きメソッドの型が親クラスのものと同一であることを示している．T-Class のコンストラクタに関する条件で，`new(...)` の引数は，各フィールドの値を示すようにしている．

2.4 型システムの性質

2.4.1 定理 [Type Preservation]: プログラム $(\overline{L}, e)$ に対し，$\overline{L}$ ok かつ $\Gamma \vdash e : C$ かつ $e \longrightarrow_v e'$ ならば，ある D が存在して $\Gamma \vdash e' : D$ かつ $D \leq C$ である．

2.4.2 定理 [Progress]: プログラム $(\overline{L}, e)$ に対し，$\overline{L}$ ok かつ $\bullet \vdash e : C$ かつ t が値でなければ，ある項 e' が存在して $e \longrightarrow_v e'$ である．

2.5 追記

元の Featherweight Java には，あるオブジェクトの属するクラスがある型の部分型かどうかを実行時に検査するための機構 (キャスト: $(C)e$ と記述する) も形式化されている．

参考文献

[1] Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight Java: A minimal core calculus for Java and GJ. *ACM Transactions on Programming Languages and Systems*, Vol. 23, No. 3, pp. 396–450, May 2001.