4G－5

型階層とクラス階層を分離したオブジェクト指向言語 CNT-L の拡張

越田一郎

東京工科大学　情報工学科

1．はじめに

近年，オブジェクト指向プログラミング言語（以下，OOPLと略す)において，型とクラスを同一視することの問題点が多く指摘されている[1]．型の継承関係とクラスの継承関係は必ずしも同型ではなく，それを前提とすることはプログラミングにおける柔軟性を損なうことになる，というのがその主張である．

筆者は，独立した型階層とクラス階層を持つ言語の，実際のプログラミングにおける有効性を検証するため，Lisp に基づくプログラミング言語 CTN-L を開発中であり，その基本的な仕様については，既に報告している[2]．今回は，その後行われた CTN-L の仕様変更と拡張について述べる．変更点は，CTN-L によるプログラミングを容易にするためのものが主である．

2．独立した型階層とクラス階層

型階層とクラス階層が同型でない例として図形の階層を考える（図1）．正方形は is-a 関係で見た場合には長方形であるから，正方形は長方形の下位型だと言える．しかし実装を考えた場合，正方形を表すには1辺の長さだけを与えれば良いのに対して，長方形では2辺の長さを必要とする．

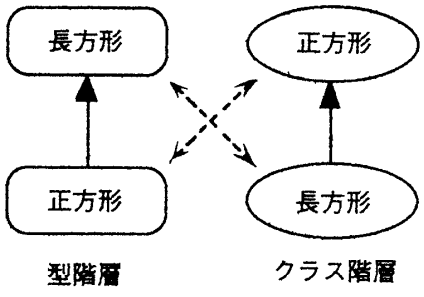

図1 型階層とクラス階層の逆転

したがって，正方形クラスにも辺の長さを表す状態を付け加えて長方形クラスを作ることが望ましい．

また，型階層とクラス階層を独立に扱うことにより，OOPL で用いられる概念を整理することができる．抽象クラスの場合を考えよう．抽象クラスはオブジェクトに対するインタフェースを定義するものであると考えることができる．これは，型の機能そのものである．共通のインタフェースをある型で定義しておき，その下位に，インスタンスを生成するクラスを対応させれば良い．

しかし，抽象クラスには，共通なインタフェースを定義するだけでなく，共通な処理を記述するという機能もある．この機能を実現するには，対応する型を持たないクラスを定義すれば良い．CNT-L では，変数はすべて型を持ち，変数に束縛されるオブジェクトは，変数の型に適合する型を持たなければならない．したがって，型を持たないオブジェクトは操作できないことになる．

3．CNT-L の追加機能

3.1 型宣言の省略

以上のように，型階層とクラス階層を分離することにより，多くの利点が得られる．しかし，型とクラスの各々について継承関係を考えなければならないのは，プログラマにより多くの負担をかけることになる可能性がある．また，多くの場合には型とクラスを同一視することができるのも事実である．そこで，CNT-Lでは型とクラスを同一視して良い場合には，それらをまとめて記述することを可能にした．

従来の仕様では，クラス定義が評価される以前に対応する型を定義しておかなければならなかった．新しい仕様ではこの制限を外し，対応する型

CNT-L : An Object-Oriented Programming Language with Separated Type and Class Hierarchy
Ichiro KOSHIDA
Tokyo Engineering University

が存在しないクラスが定義されると，自動的に対応する型が生成されるようにした．具体的には，クラス定義中に書かれたメソッドをメッセージとして持つ型を生成する．このようにして作られた型の継承関係は，クラスの継承関係に従う．すなわち，定義されるクラスがクラスAの小クラスならば，生成される型はクラスAに対応する型TAの下位型となる．

3.2 self 引数の省略

メソッド記述中で，メッセージを受け取ったオブジェクト自身を参照する際に self という特別な変数を用いる．CNT-L では CLOS と異なり，メッセージの第1引数のオブジェクトだけがメソッド選択に有効なので，self 引数はどのメソッドでも第1引数に存在する必要がある．従来の仕様では，このような self 引数もメソッド定義，型のメッセージ定義で記述しなければならなかったが，それを省略した．

4．言語仕様

本言語は Macintosh Common Lisp 2.0 の上に実装した．各フォームの形式は次のとおりである．

4.1 特殊フォーム

型定義：

define-type *name*
({ *super-type* }*)
{ (*message* ({ *type* }*)
{*return-type*) } }*

クラス定義：

define-class *name type*
({ *super-class* }*)
({ *slot* }*)
{ (*private-method* ({ *type* }*)
{*return-type* }) }*

メソッド定義：

define-method *class method*
({ *arg* }*)
body

4.2 オブジェクト生成

定義したクラスのインスタンスは次の関数で生成される

make-object *class-name*

4.3 メッセージ適用

メッセージ適用は Lisp の関数適用と同じ形式である．最初にメッセージの引数を評価し，動的に型チェックを行う．次に第 1 引数のクラスによって実行するメソッドを決定し，それを実行する．型の適合性判定には，型階層が，メソッド決定には，クラス階層が用いられる．

5．例題

スタッククラスをクラス定義無しで定義する例を示す．

list-stack クラス

```
(define-class list-stack stack
     ()
     (contents)
     (init ())
     (push (t))
     (pop () t))


(define-method list-stack push
     (val)
     (setf (contents self)
           (cons val
                 (contents self))))

(define-method list-stack pop
     ()
     (let ((rv (car (contens self))))
       (setf (contents self)
             (cdr (contens self)))
       rv))
```

6．まとめ

型階層とクラス階層を分離したオブジェクト指向言語CNT-L のプログラミングを容易にするための拡張機能について述べた．

＜参考文献＞

[1] Porter, H. H.： "Separating the subtype hirerarchy form the inheritance of implementation", 1992, JOOP, vol.4, no.9

[2] 越田：「型階層とクラス階層を分離したオブジェクト指向言語」，情報処理学会第47回全国大会，1B-5，1993