型階層とクラス階層を分離したオブジェクト指向言語

1B－5

越田一郎

東京工科大学　情報工学科

1. はじめに

強く型付けされたオブジェクト指向言語では，型とクラスを同一視しているものが多い．その結果，型階層とクラス階層は同型となり，プログラムを作成する上ではクラス階層のみを意識していれば良かった．しかし，クラスと型を同一視することの問題点が多く指摘されており，型とクラスを独立に定義し，型階層とクラス階層を分離したプログラミング言語も作成されている[1]．

このような型階層とクラス階層を分離したプログラミング言語の検討を行うことを目的として，Lisp に基づく言語を作成した．ここでは，最初にこのような言語の利点について検討した後，本言語の仕様について述べる．

2. 型階層とクラス階層の分離

型階層とクラス階層を独立に定義することが有効な場合については，既にいくつかの議論がある．ここでは，オブジェクト指向プログラミングにおいて重要な2つの概念について，型とクラスという観点から検討してみたい．

2.1 情報隠蔽

オブジェクト指向言語では，オブジェクトのプロトコルを規定するものが型であり，オブジェクトのインプリメンテーションを規定するものがクラスであると考えられる．情報隠蔽およびカプセル化を主眼とした抽象的データ型の立場から考えれば，オブジェクトに対するプロトコルとその実装は分離されるのが望ましい．たとえば，クラスライブラリのユーザに対して提供しなければならないのは，そのライブラリが提供するオブジェクトのプロトコルだけである．その実装は隠蔽しておかなければならない．

しかし，従来のオブジェクト指向言語ではプロトコルと実装の分離が良くなされてきたとは必ずしも言えない（たとえば C++）．より良い情報隠蔽を達成するためには，オブジェクトのプロトコルを型として独自に扱う機能が要求されよう．

2.2 抽象クラス

ポリモーフィズムを活用したプログラムを作成する場合，複数のクラスに共通したプロトコルを記述するためにインスタンスを作成することを考えないクラスを定義することが多い．このようなクラスを「抽象クラス」と呼ぶ．抽象クラスをサポートしている言語は多い．しかし，インスタンスすなわち実装を考慮しないという点について考えると，抽象クラスはクラスとは言い難い．抽象クラスで規定するものは基本的にプロトコルであり，したがって通常のオブジェクト指向言語におけるクラスの持つ型の側面だけを利用していることになる．プロトコルと実装を独立に定義可能であれば，抽象クラスのように特殊なクラスを考える必要がなくなる．

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

3. 言語仕様

本言語は Macintosh Common Lisp 2.0 の上で実装された．多くの機能はは Common Lisp で既に定義されているものをそのまま用い，分離した型階層とクラス階層をサポートするのに必要な機能だけを追加している．以下，各フォームの形式を示す（記法は [2] に準拠）．なお，型で定義されるオブジェクトに適用可能な操作をメッセージ，クラスで定義されるメッセージの実装をメソッドと呼ぶ．Common Lisp の標準的なオブジェクト指向環境である CLOS と異なり，メッセージの第 1 引数は，そのメッセージが適用されるオブジェクトでなければならない．また，実行されるメソッドも第 1 引数だけで決定される．

3.1 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*
({ *super-class* }*) *type*
({ *slot* }*)
{ (*private-method* ({ *type* }*)
return-type) }*

メソッド定義：

define-method (*class method*)
({ *var* }+)
body

3.2 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*

3.3 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．

4. 例題

スタック型とそれをリストで実装した例を次に示す．なお，スペースの関係でメソッド定義は一部だけ示した．

stack 型

```
(define-type stack ()
     (init (stack) stack)
     (push (stack t) stack)
     (pop (stack) t)
     (emptyp (stack) boolean))
```

list-stack クラス

```
(define-class list-stack
     ()
     (contents))

(define-method (list-stack init)
     (self)
     (setf (contents self) '())
     self)

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

5. 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についてその利点を検討した．さらに，Lisp に基づいて，上記の性質を持つ言語を実現し，簡単な例題を示した．オブジェクト指向言語が多様な分野のアプリケーション作成に用いられるに伴い，このような型とクラスの整理が重要となろう．

＜参考文献＞

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

[2] Steele, G.L. : "Common Lisp the Language (2nd Ed.)", Digital Press, 1990