プログラミング 13－6
（1997．5．21）

Javaヴァーチャルマシンをターゲットとした日本語オブジェクト指向言語の開発

中鉢　欣秀
慶應義塾大学政策・メディア研究科
大岩　元
慶應義塾大学環境情報学部

情報処理技術者不足が指摘されている今日、情報処理教育としてのプログラミング教育の重要性が高まっている。しかしながらプログラミング言語の高度化により、初心者にとって既存のプログラミング言語は学習しづらいものとなっている。今回、初心者に対する教育効果が高く、かつ全てのソフトウエア開発に対応できるようなプログラミング言語についての考察を行った。その結果、Java のヴァーチャルマシンに実装し、Java のライブラリを活用するような仕組みを備えた日本語によるオブジェクト指向言語が初心者教育から、本格的なソフトウエア開発まで対応できることがわかった。

Object-Oriented Japanese Programming Language On Java Virtual Machine

Yoshihide Chubachi
Graduate School of Media and Governance, Keio University
Hajime Ohiwa
Faculty of Environmental Information, Keio University

The shortage of Information Processing Engineer is severe in Japan, and the importance of teaching programming as a part of Information Processing is increasing. But, because of the technical advance of programming language, it is hard to study existing programming languages. We intended to develop a programming language that has high educational effect and also has a capability to develop any practical software. We have developed an Object Oriented Language, that can be written in Japanese and has a structure that can use Java Library, implemented on a Java Virtual Machine, and that can cover from educating beginner to a full-scale software development.

1 はじめに

コンピュータの歴史の中で、プログラミング言語は日々洗練され、急速に進化している。Smalltalk に始まったオブジェクト指向の技術は、perl や Tcl/Tk のようなスクリプト言語にもとりこまれ、また Java 言語のように、言語仕様もさることながら、インターネット環境での本格的使用に耐え得るよう、セキュリティやネットワークリソースに対するアクセスなどを考慮した、戦略的言語も登場している[1]。しかし、依然としてコンピュータにプログラマの意志を伝えるための手段として、「文字化された言語」を使っているという状況は変わっておらず、これ以外のより優れた方法についてのさらなる研究が望まれる。

だが、今後しばらくは、従来スタイルの「プログラミング言語」が主流であり、人間が、コンピュータが理解できる言語を学び、プログラミングするという状況は変わらないと考える。ところが、最新のプログラミング言語は機能的に高度化、複雑化され、初心者にとって学習しづらいものになっているといわざるを得ない。筆者が在籍する慶應義塾大学湘南藤沢キャンパスの大岩研究室では、昨年から、主に初心者を対象に、Java 言語を用いてのオブジェクト指向プログラミング、perl を用いての文字列処理プログラミングの教育を行っている。この際、もっとも難しく感じるのは、Java が備えているオブジェクト指向のフレームワークの仕組みや、perl の洗練された表記法など、上級者にとっては有り難い高度な機能である。前提となる知識の少ない初心者には「おまじない」として教えるしかなく、ひいては初心者の学習意欲をそぐこともある。

情報処理教育の重要性が指摘されている今日、初心者のプログラミング教育は重要なテーマであると考える。また、高度なソフトウエア技術者の不足も指摘されている。コンピュータが急激に普及し、より多くのソリューションがコンピュータに求められており、適切な情報処理教育を行い、問題を発見し、解決する人材の育成が急務である。

このような目的のために、教育効率が高く、かつ全てのアプリケーション開発に対応できるプログラミング言語について検討する必要があると考える。今回、こういった目的に対応するための、プログラミング言語開発のためのアプローチと、基本的な言語仕様をまとめたので、ここで報告する。

2 プログラム言語「ふじ」

2．1 日本語プログラミング言語

今回仕様の定義を行った言語はコードネームを「ふじ」とした。由来は、湘南藤沢キャンパスの「藤沢」の頭文字を取ったものである。

まず、初心者が抵抗なくプログラムの学習を始めるために、日本語による表記を採用することにした。日本語をベースにしたプログラミング言語で、実際に商用化された言語としては(株)リギーコーポレーションの片桐明が開発した日本語 Mind がある。片桐は、日本語によるプログラムの表現とスタック式言語の表現とが非常に相性の良いことに気づき、Forth をもとに日本語 Mind を開発した。日本語 Mind は、日本語によるプログラムの記述性、保守性の高さなどから、一部のソフトウエア技術者に根強く支持されている。しかしながら、このような試みはあるものの、プログラミングへの本格的な日本語の導入は一般的にはなっていない。

筆者は、プログラムの学習の第一歩は、ソースコードの読解から始まると考えている。初心者がソースコードを読むとき、一見して何が書いてあるのか分かるというのは、教育の入り口として効果的である。このことと、後述する日本語の「助詞」の役割がオブジェクト指向的な表現によくなじむということで、「ふじ」は日本語をもとにしたプログラミング言語とすることにした。

2．2 教育効果と実用性

「ふじ」は初心者に教育しやすく、かつ実用的な言語とすることが目標である。本言語のターゲットマシンは Java のヴァーチャルマシンとした。このことで、「ふじ」で開発されたプログラムは、多くのプラットフォームでの利用が可能となる。

また、オブジェクト指向を導入し、Java のライブラリを活用できるような仕組みを取り入れる必要がある。これには、言語に動的なデータベースを導入し、Java が持つライブラリを日本語の名前でアクセスできるよう登録することにした。このデータベースは、単なる「和英辞書」ではなく、簡単な名前付けの規則を導入することによって、初心者にとって煩わしい変数宣言の簡略化を支援することができることが分かった。このデータベースを「ふじ」の「支援データベース」と呼び、あらかじめライブラリのクラスやクラス変数、メソッドを登録しておく。また、プログラム中で動的に登録することも可能とする。

2．3　変数とクラスの名前付け

Java 言語など静的型宣言のプログラム言語では、変数は使う前に型を宣言する必要がある。変数の宣言は、宣言されていない変数の使用を防ぐ、関数の引数の型の不一致をチェックするなどの意味を持つが、それを知らない初心者にとっては煩わしい不可解な行為に見えるようだ。しかしながら、オブジェクト指向言語では、静的型宣言の言語であっても動的型宣言の言語であっても、メッセージ送信の際、インスタンスの型を判断することが必要で[2]、本言語においても「型」の概念を無くすことはできない。そこで、初心者にとって煩わしくなく、かつ適切に型を指定する方法について検討した。

その結果、変数名をあらかじめ支援データベースに登録された型名とするか、あるいは変数の末尾を登録されている型名とすることで、変数の宣言を記述しない方法が効果的であることが分かった。

たとえば、支援データベースに「名」という型が String 型であると登録しておく。このようにすると、プログラム中にある「名」、「国名」、「書籍名」という名前の変数は「String 型」だと判断するというものだ。

このデータベースへの登録は、プログラム中で動的に記述できる。プログラム中での記述は次のように行う。「文字列」はあらかじめ String 型として登録してあるものとする。

```
姓は文字列。
名は文字列。
```

このように登録すると、次のようなプログラムが有効になる。ここで、「代入」は「ふじ」の基本語である。

```
姓に「中鉢」を代入する。
名に「欣秀」を代入する。
```

プログラム中でのクラスの定義も同様に、支援データベースに登録される。「姓」と「名」をクラス変数に持つ「氏名」というクラスを定義するには次のように記述する。

```
氏名とは｛
  姓。名。
｝
```

これ以降、「氏名」、「学生氏名」などという変数は、氏名クラスのインスタンスとして扱われる。なお、本言語でもブロック構造の記述には「{」と「}」を用い、文の終わりは「；」ではなく「。」を用いることにした。

また、この型の判断は、既に支援データベースに登録されている型名をもとに、変数名の末尾から「後方最長一致方式」で判断する。例外として変数名がアルファベットや数字の列で終わっているとき、その部分は無視することにした。「名」とは別の「氏名」というクラスが支援データベースに登録されていれば、「氏名１」、「氏名 A」という変数も氏名型だと判断する。

変数名に型名を含ませるという考えは、特に目新しいアイディアではない。昔のパソコン用

BASIC でも、「defint　i」と書くことで、i で始まる変数名は int 型とすると指定できた。また、C++言語などでは、変数名の頭に型を表すハンガリアン記法が普及している。本言語では、このような規則を言語仕様に取り込み、登録と使用を自然な表現で行えるようにしたものである。

同様なことはクラスの継承関係にもあてはまる。「アプレット」という名前は Java のライブラリの java. applet. Applet であると登録しておけば、「時計アプレット」「計算機アプレット」などという名前で新しいクラスを定義するだけで、それらは全てアプレットクラスから継承されたクラスだと判断され、かつ支援データベースに動的に登録される。

このことで、初心者がアプレットを作るときに最初にとまどう、

```
public class myClass　extends java. applet. Applet
```

というような「おまじない」が必要なくなる。自分のアプレットクラスを定義するにはクラス名の末尾に「アプレット」とつけておくだけでよい。

2．4　日本語プログラムと助詞

日本語 Mind では、分かち書きで語と語を区切っている。「ふじ」では、できるだけ自然な日本語の書き言葉を模するため、分かち書きを避け、その代わりに日本語の「助詞」を解析する事にした。検討の結果、日本語の助詞がプログラム中で現れるとき、特に日本語の文法でいう格助詞が、オブジェクトに対する操作、あるいは条件文の比較対象などの指定に一意に使われることが分かった。

前述の支援データベースに「システム」クラスと、その変数とメソッド「出力」、「表示」がそれぞれ「System」、「out」、「print」に対応すると登録されているとき、次の例は「ふじ」の有効なプログラムとなる。

```
国名に「日本」を代入する。
システムの出力に国名を表示する。
```

アンダーラインを引いた助詞「に」は操作の対象となるオブジェクトを示し、「を」は操作に引き渡される情報を示す。助詞「の」は、Java 言語の「.」に相当し、オブジェクトを示し、クラス変数を導く。他に、「に」と同様、操作の対象を示す助詞には「へ、から、より」があり、「を」の他には複数の情報を引き渡すときに使用する「と」と「で」がある。

これらの助詞の特性を解析することで、プログラム中の語順の変化にも対応できる。上の２行目を「国名をシステムの出力に表示する。」としても、「表示する」という操作の対象は「システム」のクラス変数である「出力」であり、引き渡すべき情報は「国名」であると判断できる。

この他に、助詞「が」は条件の比較対象を示す。「ふじ」の条件文は次のようになる。

```
国名が「日本」ならば
    システムの出力に「こんにちは」を表示する。
```

繰り返し文も同様に次のようになる。

```
値は整数。
数値に０を代入する。
数値が１０より小さい間｛
    システムの出力に「こんにちは」を表示する。
    数値を１増やす。
｝
```

2．5　操作の定義

本言語でクラスの操作（メソッド）の定義は次のように記述する。ここでは、前述の氏名クラスに、「表示」と、「代入」、「メイン」の２つの操作を追加し、実行可能なプログラムにした。（図１）

引数を取る操作の定義において、操作に与えられる仮引数には変数名さえ明記すればよい。型の判断は型埋め込み式変数名の仕組みでチェックが可能である。操作「代入」は仮引数「姓１」と「名１」をとる。これらは、型埋め込み式変数名の例外が適用され、ともに文字列型と

```
氏名とは {
  姓。
  名。
  表示するとは {
    システムの出力に姓を表示する。
    システムの出力に名を表示改行する。
  }
  代入するとは姓１と名１をとり {
    姓に姓１を代入する。
    名に名１を代入する。
  }
  メインとは {
    学生氏名を生成する。
    学生氏名に「中鉢」と「欣秀」を代入する。
    学生氏名を表示する
  }
}
```

図１ 「ふじ」による氏名クラスの定義

```
public class 氏名{
  String 姓;
  String 名;
  public void 表示(){
    System.out.print(姓);
    System.out.println(名);
  }
  public void 代入(String 姓１, String 名１){
    姓 = 姓１;
    名 = 名１;
  }
  public static void main(String arg[]){
    氏名 学生氏名 = new 氏名();
    学生氏名.代入("中鉢","欣秀");
    学生氏名.表示();
  }
}
```

図２ 「Java言語」による氏名クラスの定義

判断されるのは前述のとおりである。

3. Java との親和性

今回、「ふじ」のターゲットマシンとして、Java のヴァーチャルマシンを選んだ。それは次のような理由による。

1 仕様が完全に公開されていること。
2 バイトコードがスタック式であること。
3 バイトコードレベルでオブジェクト指向であること。
4 様々なプラットフォームに実装され、多くのライブラリが整備されつつあること。

日本語 Mind の実装から、日本語の表記とスタック言語は相性がよいとされている[3]。このことを検証するため、図１とほぼ等しい Java 言語のコード（図２）を作成した。これを、Java のリファレンス実行環境となっているジャバソフトの JDK1.1.1に付属の javac コマンドでコンパイルし、javap コマンドで逆アセンブルしたものから、メソッドの部分だけを取り出した結果を図３に示す。

行頭の数字はメソッドの先頭からのバイト数を表し、コメントの形で記述したのは、対応する「ふじ」のコードである。また、「(自分の)」とある部分は、メソッドにおける自分のクラスの変数へのアクセスの際、省略されている Java 言語の「this」に相当し、「ふじ」でも省略されている。

これを見ると、バイトコードの１つのステップがほぼ「ふじ」のコードに対応していることが分かる。

ここでは、メソッドの呼び出しの際のバイトコード部分を検証する。図３の main メソッドの８～１５バイト目は次のようになっている。

```
 8 aload_1
 9 ldc #1 <String "中鉢">
11 ldc #2 <String "欣秀">
13 invokevirtual #12
    <Method void 代入(java.lang.String, java.lang.String)>
```

最初のコード aload_1で局所変数「学生氏名」への参照をスタックに積み、２つの ldc 命令で「中鉢」と「欣秀」の２つの文字定数をスタックに積んで invokevirtual 命令で氏名クラスの代入メソッドをコールしている。

対応する「ふじ」のコードを示す。

```
学生氏名に「中鉢」と「欣秀」を代入する。
```

前述の助詞「に」、「と」、「を」の性質を利用すれば、助詞で区切られた部分をそれぞれバイトコードで置き換えることで、コンパイルできる。ただし、「ふじ」では語順は必ず上のように現れるとは限らないため、順序関係の整合性を取る必要がある。たとえば、次のように記述される可能性がある。

「中鉢」と「欣秀」を学生氏名に代入する。

このように語順を柔軟にするためには、コンパイラの手間は増えることになるが、日本語で自然にプログラムをするということは、逆にいえば、日本語として正しいならばなるべくそれで動作するようにしたいということになる。助詞の使われ型は明確に定義しているため、混乱は少ないと思われるが、今後の実装に向けての大きなテーマとなるだろう。

4．まとめ

今回、Java のヴァーチャルマシンをターゲットとした日本語プログラミング言語「ふじ」の基本的なアイディアを紹介した。この言語の実装は現在行われている途中であり、今後仕様が変更される可能性はある。

「ふじ」は Java との整合性をとることを目標に設計したが、今後日本語によるプログラミングそのものをより実用的なものに洗練させ、向上させることは興味深いテーマである。今回の試みはそのための礎としたい。

5．参考文献

1) リメイ, パーキンス:Java 言語入門, トッパン (1996)

2) 小野寺:オブジェクト指向言語におけるメッセージ送信の高速化技法, 情報処理, Vol. 38, No 4, pp301-310 (Apr. 1997)

3) 片桐:Mind 基本文法, リギーコーポレーション (1995)


```
Method void 表示()
   0 getstatic #9 <Field java. io. PrintStream out>                    // システムの出力に
   3 aload_0                                                           // (自分の)
   4 getfield #14 <Field java. lang. String 姓>                        // 姓を
   7 invokevirtual #10 <Method void print(java. lang. String)>         // 表示する。
  10 getstatic #9 <Field java. io. PrintStream out>                    // システムの出力に
  13 aload_0                                                           // (自分の)
  14 getfield #13 <Field java. lang. String 名>                        // 名を
  17 invokevirtual #11 <Method void println(java. lang. String)>       // 表示する。
  20 return

Method void 代入(java. lang. String,  java. lang. String)
   0 aload_0                                                           // (自分の)
   1 aload_1                                                           // 姓1を
   2 putfield #14 <Field java. lang. String 姓>                        // 姓に代入する。
   5 aload_0                                                           // (自分の)
   6 aload_2                                                           // 名1を
   7 putfield #13 <Field java. lang. String 名>                        // 名に代入する。
  10 return

Method void main(java. lang. String[])
   0 new #6 <Class 氏名>
   3 dup
   4 invokespecial #8 <Method 氏名()>
   7 astore_1                                                          // 学生氏名を生成する。
   8 aload_1                                                           // 学生氏名に
   9 ldc #1 <String "中鉢">                                            // 「中鉢」
  11 ldc #2 <String "欣秀">                                            // 「欣秀」を
  13 invokevirtual #12 <Method void 代入(java. lang. String,  java. lang. String)> // 代入する。
  16 aload_1                                                           // 学生氏名を
  17 invokevirtual #15 <Method void 表示()>                            // 表示する。
  20 return
```

図 3　Java のオブジェクトコードと「ふじ」との対比