オブジェクト指向を指向する初期アルゴリズム教育

林 雄二

オブジェクト指向が唱えられて久しいが，ソフトウエア開発の世界では，ようやく潜伏期を終え，蔓延期の入りつつあるように思われる．北海道情報大学では，1学年における科目〝プログラム設計論〟において，1994年度から，オブジェクト指向の内容を教育に盛り込むことにした．問題世界のモデル化と段階的詳細化を柱として，オブジェクト指向の精神を初心者にも解るように教育することが大切であり，可能であろうと考えられる．本稿では，この教育で用いたものを含む3題ほどの例題をもとにして，具体的にどのようにオブジェクト指向的なプログラムを開発するか，その考え方を述べる．

1．はじめに

オブジェクト指向によるソフトウエア開発が叫ばれて久しいが，ようやく最近になってツールの開発が進められ，本格的に取り組みがなされるようになってきた．本学（北海道情報大学）における1学年の科目「プログラム設計論」において，遅ればせながら1994年度からオブジェクト指向の内容を盛り込むことにした．本稿はオブジェクト指向に基づくソフトウエア設計について，講義の中で取り上げたものを含むいくつかの例題を解説したものである．プログラム設計論における講義だけではなく，プログラミングの基礎教育の段階からオブジェクト指向の考え方は取り入れられるべきであり，その考えの中で基礎になる‘現実界のモデル化’を，段階的詳細化と同時に教育に盛り込むことが可能であるというのが筆者の主張である．

2．オブジェクト指向とプログラム設計

現在のオブジェクト指向という言葉の含むすべての考え方が，今後もそのまま踏襲されていくとは限らないが，多くの考えは新しいソフトウエア開発方法論の基礎として普及していくものと思われる．オブジェクト指向という言葉が意図している主な概念を挙げてみると以下のようになろう．

(イ) プログラムは問題世界をモデル化したものであるべきという考え（例えばホテルの管理システムを開発するのであれば，ホテルの利用客，客室，カウンタなどがプログラム中に反映されるように作るべきである）．

(ロ) メッセージ伝達と受け手の責任におけるメッセージ処理（処理はオブジェクトにメッセージを送ることによって実行させる．しかも，処理の詳細はメッセージの送り手がいっさい知る必要はなく，メッセージの受け手がすべて責任をもって行うこと）．

(ハ) クラスとインスタンスによるプログラム要素の統一（データの型はクラスの1つであり，その場合個々のデータはそのインスタンスである．また，ホテルがクラスであれば，ライオンホテル，プリンセスホテルなどがそのインスタンスである．あらゆるオブジェク

Yuji HAYASHI 北海道情報大学経営情報学部

トはクラスとインスタンスによって表現可能).

(ニ) クラス階層はクラスの性質の継承を含むこと（クラスの設計は，上位クラスの持つ性質に追加する部分だけについて行えばよい．このことは差分プログラミングを用いたソフトウエアの再利用につながる).

これらの概念をすべて包含するプログラムは，オブジェクト指向向けプログラミング言語(Smalltalk, C++など，以下OOP言語と呼ぶ）によらなければならないが，(イ)(ロ)の考え方に関しては通常の手続き型プログラミング言語によっても教育が可能ではないかと考えられる．ただし，OOP言語で表現したプログラムでは個々の要素がオブジェクトであり，プログラムの段階的詳細化の各レベルがその表現に含まれている．しかし，一般の手続き型言語では，特に意識しない限り，最終段階レベルのプログラムのみが表現され，それに至る上位レベルの設計はプログラム中には残らない（コメントで残されるにすぎない)．そのような意味で，一般の手続き型言語で表現されたプログラムは，必ずしもオブジェクト指向にかなったものではないが，プログラミングをこのような考え方で進めること自体がオブジェクト指向の理解につながるものと考える．

本学の情報学科では，1学年のプログラミング関係の講義として以下の2科目がある．

プログラミング言語1（Pascal)
必修 6単位（通年）
プログラム設計論
必修 2単位（後期）

プログラム設計論（2単位）について，本年度は以下の内容で行った．

(1) プログラム設計とアルゴリズム
(2) フローチャートによるアルゴリズム表現
(3) 構造化プログラミング
(4) 疑似言語によるアルゴリズム表現
(5) PADによるアルゴリズム表現
(6) 段階的詳細化
(7) ジャクソン法（JSP）による設計
(8) オブジェクト指向によるプログラム設計

最後の(8)オブジェクト指向によるプログラム設計の部分は，本年度初めて試みたものである．オブジェクト指向の考え方（第3章で示すものと同様な例をも含む）とSmalltalkの概要説明（含例題）がその主たる内容である．

段階的詳細化をプログラミング設計の基本的要請であるといいながら，その方法に対する具体的な指針がないことに筆者は長い間疑問を感じていたが，オブジェクトという問題世界の実体をモデル化することを詳細化の指針とすることによって（オブジェクト指向とは限らない)，一般的なプログラミングにおいて段階的詳細化を進めるための柱が得られたと考えられる．その意味で，プログラミング教育においても，可能な限り現実界のモデル化というオブジェクト指向の精神を身に付けるよう教育することが必要であろうと思われる．以下では，この講義で取り上げた例題を基に，アルゴリズム基礎教育でオブジェクト指向の考え方を盛り込むことへの試みを考察する．

3．例題によるオブジェクト指向的な設計

3つの例題をあげ，オブジェクト指向の考えを取り入れて設計する方法を考える．いずれも，問題から直接プログラミングをするのではなく，まず大まかな設計をして，それを詳細化することで，自然にオブジェクト指向の考えが取り入れられることを示している．すなわち，まず問題の世界からオブジェクトを抽出し，そのオブジェクトを含んだおおまかな設計をし，次にオブジェクトの

(a) データ部を作り

(b) メソッド部のアルゴリズムを作る

こうすることによって，オブジェクトの詳細化を表現することができる．ただし，本章の以下の記述ではメッセージ伝達を強調するために，オブジェクトの部分については，日本語表現を基にしたSmalltalk風の表現を用いている．

例1）多数の点数を入力し，最大値と2番目に大きな値を求めて出力する．入力のストッパは999とする．

以下では3通りの表現によってプログラミングを行ってみる．(a)はオブジェクト指向に徹したSmalltalk風のプログラム，(b)はオブジェクト指向的に設計したものを単純に詳細化したプログラム，そして(c)はオブジェクトをレコード型データや手続き(procedure)で表現することを試みたPascalのプログラムである．

(a) Smalltalk風のプログラミング

（a__1） 第1段階のプログラム(疑似言語)

```
上位2をクリア
read 点数
while 点数≠999 do
  if (上位2 の2位)<点数 then
      上位2 にセット：点数
  endif
  read 点数
endwhile
上位2 を出力
```

（a__2） オブジェクトの設計（Smalltalk風）

```
object 上位2

  data
    第1位
    第2位

  method
    をクリア
      第1位←－99
      第2位←－99

    の2位
      return 第2位

    にセット：点数
      if 点数＞第1位 then
        第2位←第1位
        第1位←点数
      else if 点数＞第2位 then
        第2位←点数
        endif
      endif

    を出力
      write 'max=', 第1位
      write 'second=', 第2位
```

第1段階のプログラム（a__1）は，1位と2位を一緒に扱っており，1位2位の決め方の詳細は後回しにして設計している．このとき1位2位を含めたデータとして，上位2というオブジェクトが必要になる．この第1段階のプログラムは必ずしもオブジェクト指向を意識しなくとも生み出される自然なプログラムであり，それを詳細化するときにデータと操作（メソッド）の両方を含んだものとしてのオブジェクトが具体化される（図1）．このオブジェクトは本来はクラスというべきものであるが，クラスに言及しないで解説するためにあえて，オブジェクトとしている．

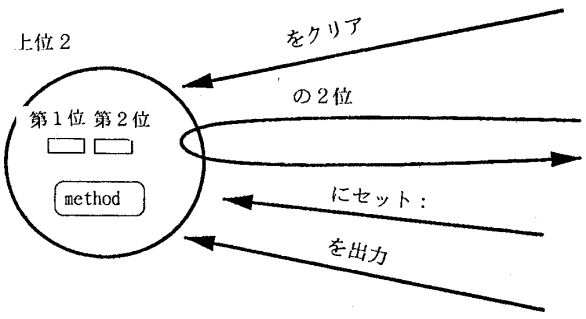

図1 オブジェクト「上位2」

大切な点は，（a＿1），（a＿2）の両方でプログラムを構成していることである．すなわち，プログラム自身に段階的詳細化が同時に表現されていることである．

この（a＿1）を第1段階の設計とみなして詳細化をして得られる疑似言語によるプログラムは以下(b)のようになる．この場合は，この(b)のみがプログラムであり，（a＿1）は(b)が作成された後は役目の大半を終えてしまう．そして，問題に変更が加えられたときには，せっかくの第1段階プログラム（a＿1）はプログラム修正作業に充分な威力を発揮してくれない．たとえば，'上位3位までを求める'というように問題が変更になったとき，結局この(b)プログラム全体を追跡して，修正個所を見いだしていかなければならないであろう．

(b)　（a＿1）から詳細化されたプログラム

```
{上位2をクリア}
第1位←－99
第2位←－99
read　点数

while　点数≠999　do
  if　第2位＜点数　then
    {上位2　にセット：点数}
    if　点数＞第1位　then
      第2位←第1位
      第1位←点数
    else
      第2位←点数
    endif
  endif
  read　点数
endwhile

{上位2　を出力}
write　'max＝', 第1位
write　'second＝', 第2位
```

(a)の表現では，個々のメソッドをSmalltalk風に記述したが，不慣れであればPascal風の記述になることも致し方ない．この場合は，1つひとつのメソッドが手続き(procedure)や関数(function)になる．例えば〝上位2をクリア〟の部分は，Pascal風に表現すると，以下のようになる．

```
procedure　上位2＿クリア；
  上位2．第1位←－99；
  上位2．第2位←－99；
end　上位2＿クリア
```

すなわち，第1位，第2位という変数は，レコード型変数〝上位2〟のメンバで，グローバル変数である．従ってデータがカプセル化されて保護されるというオブジェクト指向にはそぐわない表現になっている．しかし，標準のPascalで教育をしていく限りこのことは致し方ないものと思われる．

以下のプログラムはPascalによって記述したものである．ただし，変数名，手続き名などをあえて日本語を用いて表現している．

(c)　手続きを用いたPascalプログラム

```
program　上位2を求める；
var　上位2：record
          第1位：integer；
          第2位：integer
        end；

procedure　上位2＿クリア；
  begin
    上位2．第1位←－99；
    上位2．第2位←－99；
  end　上位2＿クリア；

procedure　上位2＿セット(k: integer)；
  begin
    if k＞上位2．第2位　then
      begin
        上位2．第2位：＝上位2．第
        1位；
        上位2．第1位：＝k
```

```
        end
      else
        上位2．第2位：＝k
    end　上位2__セット；

  procedure　上位2__出力；
    begin
      writeln（'最大値=', 上位2．第1位）；
      writeln('第2値=', 上位2．第2位)
    end　上位2__出力；

  begin

    上位2__クリア；
    read（点数）；
    while　点数≠999　do
      begin
        if　上位2．第2位＜点数　then
          上位2__セット（点数）；
        read（点数）
      end；
    上位2__出力；

  end.
```

(a), (b), (c)の3つを比較してみると，まず(b)はレコード型データや手続きなどを使用していないプログラムである．段階的詳細化によって得られたにもかかわらず，プログラムには第1段階の設計は表現されておらず，設計とプログラミングが別の産物を生み出している．一方，(a), (c)はプログラム中に段階的設計の表現がそのまま残されている点が(b)に勝っている．(c)はデータをグローバルにせざるを得ないことと，手続き呼び出しが(a)に比べて十分な表現力を持たないことが不満な点である．Smalltalkにおけるメッセージ伝達は，(a)のように自然言語（特に目的語をはじめに配置する日本語の用法）に近い表現をすることができるのも特徴であろう．

例2） 英字，スペースおよびピリオドからなる文字列を入力し，1行30文字以内に収まるようにそのまま出力する．ただし単語(スペースやピリオドすなわち区切り記号の間にある文字列が単語である）が2行にまたがるときは次の行に出力すること．30文字を越える文字列はないものとし，入力のストッパは〝$〟とする．入力例とそれに対する出力をあげておく．

〈入力例（ただし改行は入力されない）〉

```
Ever since computers were used to solve
application problems, it was the task of
software to bridge the gap between con-
cepts in an application and computer
concepts.$
```

〈出力〉

```
Ever since computers were
used to solve application
problems, it was the task of
software to bridge the gap
between concepts in an
application and computer
concepts.
```

(1) 第1段階のプログラム

```
行　をクリア
read　文字
while　文字≠〝$〟　do

  単語　を入力
  if（行　に収まらない：単語）　then
    行　改
  endif
  単語　を出力

  区切り　を入力
  if（行　の位置＝行　の最後）　then
    行　改
  endif
  区切り　を出力
```

```
    endwhile
(2) オブジェクトの設計（Smalltalk 風）
    object　単語
      data
        word　.....array [1..30] of char
        pos　..... 1..30

      method
        を入力
          pos ← 0
          while　文字　が英字　do
            word [pos＋1] ←文字
            pos ← pos＋1
            文字　を入力
          endwhile

        を出力
          for i：＝1　to pos do
            write word [ i ]
            行　位置増
          endfor

        の字数
          return pos

    object　行
      data
        字数 in 行　....　30
        col　.... 行内の現在位置

      method
        をクリア
            col ← 0

        の位置
            return col

        の最後
            return　字数 in 行

        に収まらない：単語
          if（単語　の字数）＋（行　の位置）
          ＞（行　の最後）　then
            return true
          else
            return false
          endif

        改
          write crlf
          col ← 0
        位置増
          col ← col＋1

    object　区切り
      data
        ch　.... 入力した区切り記号
        有無　.... 入力が区切りか否か

      method
        を入力
          有無←'無'
          if（文字　が　英字でない）
            and（文字≠'$'）　then
              ch ←文字
              有無←'有'
              文字　入力
          endif

        を出力
          if　有無＝'有'　then
            write ch
          endif
```

問題の世界に現れる言葉として，単語，行，区切り記号などがあるが，このような言葉をオブジェクトとして，まず抽象的なデータとしておく．そのデータの詳細（図2，3，4）は次の段階で定めればよい．このように，問題の中に現れる言葉や概念をデータとして第1段階のプログラム作成をすることがオブ

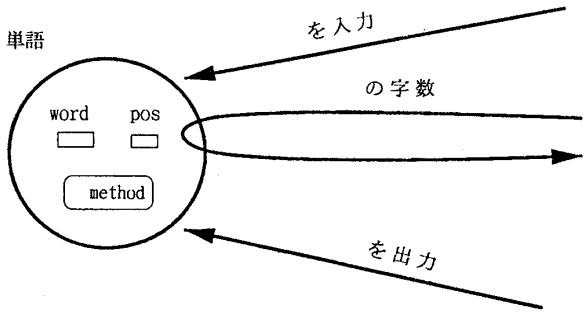

図2 オブジェクト「単語」

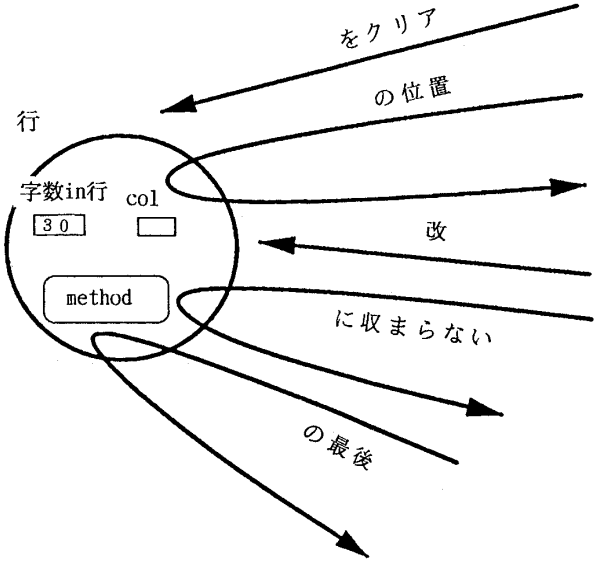

図3 オブジェクト「行」

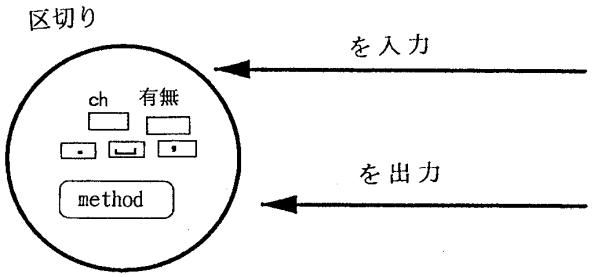

図4 オブジェクト「区切り」

ジェクト指向にかなった設計に結びつく.

例3）図書館の利用者に関するデータ，すなわち利用者番号，入館時刻(時，分)，退館時刻（時，分）の組を多数入力し，個々の利用者が図書館に居た時間を，～時間～分として出力する．時刻は0～23の表示で，入力のストッパは利用者番号＝0である.

これはオブジェクト指向とは無関係に疑似言語による設計の練習問題として学生に出題したものである．筆者は当日限りの入退館を想定していたが，この問題に取り組んだ多くの学生が，翌日の退館（ただし滞在時間は24時間以内に制限）を認めることにして設計した．ここでは，そのような条件の下で設計を考えることにする.

(1) 第1段階のプログラム

以下は，アルゴリズムを作る能力があまり備わっていないと思われる某君の解答を，少し手直しをして載せたものてある.

```
入館時刻　入力
退館時刻　入力
時刻差←退館時刻－入館時刻
if　時刻差＜0　then
  （時刻差＋時　24）出力
else
  時刻差　出力
endif
```

これは時間を時と分に分けて計算していないので，このままでは，通常のプログラミングの解答としては正しくないが，段階的に詳細化する最初の段階では十分な設計であり，しかも，オブジェクト指向の下でのプログラミングになっている．実際クラスとして，〝時刻〞を次のように設計すれば，入館時刻，退館時刻，時刻差などはそのインスタンスになり，立派なオブジェクト指向のプログラムに仕上げられる.

(2) オブジェクトの設計

```
class　時刻
  data
    分
    時

  method
    の分
```

```
  return　分
の時
  return　時

入力
  read　時，分

－　他時刻
  if　分＞（他時刻　の分）then
    temp←時刻　new
    temp　時セット：
      （他時刻　の時－時－１）
    temp　分セット：
      （他時刻　の分－分＋60）
  else
    temp←時刻　new
    temp　時セット：
      （他時刻　の時－時）
    temp　分セット：
      （他時刻　の分－分）
  endif
  return temp

＜数
  if　時＜数　then return true
              else return false
  endif

＋時　数
    時←時＋数
時セット：数
    時←数

分セット：数
    分←数
```

ただし，プログラム(1)における入館時刻，退館時刻，時刻差などが，(2)におけるクラス時刻のインスタンスであることを表す文がプログラム中には存在しないが，それを表現するには特定のOOP言語によらなければならない．

なお，この問題はレポートの課題として，オブジェクト指向に言及せずに出題したものであり，様々な解答が得られた．そのなかの１つを挙げてみる．

```
{学生の解答例}
if　退館時＜入館時　then
    滞在時←退館時－入館時＋24
    滞在分←退館分－入館分
else
    滞在時←退館時－入館時
    滞在分←退館分－入館分
endif

if　滞在分＜0　then
    滞在時←滞在時－1
    滞在分←滞在分＋60
    if　滞在時＜0　then
    滞在時←滞在時＋24
  endif
endif
```

これは間違いではないが，分かりやすいプログラムとはいい難い．オブジェクト指向の考え方にある現実界のモデル化という観点に立てば，プログラムに，当日退館の場合，翌日退館の場合という問題の世界が反映されるように作るべきであろう．

```
{望ましい設計　1}
分単位←（退館時－入館時）＊60
        ＋（退館分－入館分）
if　分単位＜0　then｛翌日退館｝
  分単位←分単位＋24＊60
endif
滞在時←分単位÷60　の商
滞在分←分単位÷60　の余り
```

あるいは

```
{望ましい設計　2}
if（退館時＜入館時）or
  （退館時＝入館時　and
```

```
    退館分＜入館分)
  then 〔翌日退館〕
    滞在時←退館時－入館時＋24
    滞在分←退館分－入館分
  else 〔当日退館〕
    滞在時←退館時－入館時
    滞在分←退館分－入館分
  endif

  if 滞在分＜0 then〔繰り下がり〕
    滞在時←滞在時－1
    滞在分←滞在分＋60
  endif
```

問題の世界を出来るだけ忠実にモデル化し,段階的にデータとメソッドを詳細化する.この基本的な考えが,オブジェクト指向に結びつく設計になる.コンピュータに縁の薄い科学者にオブジェクト指向の発想法を説明したところ,'そんなことは当然の考えでないか.いったい今までどうやってソフトウエアを設計してきたのだい!'という答えが返ってきたという[3].オブジェクト指向は原点に帰った発想によってソフトウエア開発を進めようという,コンピューター界のルネッサンスをもたらしているのではなかろうか.

4. まとめ

段階的詳細化の方法として,オブジェクトという問題世界の実体をモデル化すれば,自然に詳細化を進めることが今や可能になった.逆に,オブジェクト指向の考えを身につけるには,段階的詳細化を心掛けることが手始めである.プログラミング教育においても,まず

(1) 問題の記述に現れる単語などから,オブジェクトを抽出する.

(2) オブジェクトをデータ部分,メッソド部分にわけ,それぞれを詳細化する.

このようなことを実践することによって,オブジェクト指向の精神は少なからず身に付くのではないかと思われる.さらに,ほかのオブジェクト指向の機能(クラス,継承など)については,SmalltalkやC＋＋などのOOP言語の教育において取り入れることにすればよいと考える.

オブジェクト指向に基づいたソフトウエア開発は,今後基本的な設計概念として生き続けるものと思われる.オブジェクト指向についてのお話をするだけでは教育をしたことにはならず,初期プログラミング教育において,可能な限り演習などを盛り込んでオブジェクト指向の精神を身に付けるよう教育することが必要であろうと思われる.大学における情報処理教育は,ソフトウエア開発の現場に新風を吹き込むような学生を送り出すことが期待されているのではなかろうか.

謝辞:アルゴリズム教育ワークショップにおいて有益なご意見を寄せられた参加者の皆様,並びに本ワークショップの企画,運営にご尽力された札幌学院大学社会情報学部の皆様に御礼を申しあげます.

参考文献

1) Badd, T.A.: *An Introduction to Object-Oriented Programming*, Addison-Wesley (1991).
邦訳 羽部訳:オブジェクト指向プログラミング入門,トッパン (1992).

2) 鈴木則久:オブジェクト指向,共立出版 (1985).

3) Entsmeinger, G.: *The Tao of Objects*, M&T Books (1991).
邦訳 吉田訳:タオ・オブ・オブジェクト,技術評論社 (1992).

4) 林 雄二:プログラム設計の基礎,森北出版 (1993).