社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2003－CE－72　(15)
2003／12／15

プログラミング入門教育におけるモデルによるプログラムの状態表現

青山　希[†]　松澤芳昭[†]　杉浦　学[†]
川村昌弘[†]　大岩　元[††]

プログラミングの導入教育において，学習者が自分でプログラムを作成することができるようになるためには，制御構造を理解し，プログラム実行時の動作を正しくイメージする必要がある．さらにプログラムの 実行時の状態を把握し，処理が実行された瞬間の状態遷移を正しくイメージできる必要がある．しかし，学習者がプログラムの状態を理解するための，簡潔で，わかりやすい記法はない．

これに対し，我々はプログラム実行時の，オブジェクトの状態を図示する 3 種類の説明モデル「表モデル」,「入れ子モデル」と「参照モデル」を提案し，実際にプログラミング教育の場で試用してみた．本稿ではこれらのモデルについて，その有効性について考察する．

Expression of State of Program with Models In Introduction to Programming Courses

Nozomu Aoyama,[†] Yoshiaki Matsuzawa,[†] Manabu Sugiura,[†]
Masahiro Kawamura[†] and Hajime Ohiwa[††]

In introduction to programming courses, a leaner must understand the control structure of programming and must be able to imagine what is done and how the state of the program is changed at the time of program execution. However, there is no appropriate notation for the learner to understand the state of a program.

For this problem, we propose three kinds of explanation models named "Table Model", "Nest Model", and "Reference Model" to illustrate the state of program at the time of execution. We have tried them for programming instruction, and describe how these models are useful.

1. はじめに

プログラミングの導入教育において，学習者が自分でプログラムを作成することができるようになるためには，まず制御構造を理解し，プログラム実行時の動作を正しくイメージできる必要がある.[6] しかし，それだけでは自分でプログラムを作成するためには不十分である．それに加え，プログラムの実行時の状態を把握し，処理が実行された瞬間の状態遷移を正しくイメージできる必要がある．

プログラミング教育の初期段階において，学習者が作成したプログラムが学習者の思うように動かず，なぜそのような結果がでるのかの検討がつかなくなることがままある．これはプログラム実行時の動作と，その時点での状態を正しくイメージできていないためであると考えられる．動作と状態を正しくイメージでき，プログラムの 1 行 1 行で何が起きているのかを把握できるようになって初めて，学習者はプログラムの実行とともにどのような処理が行われているのかを理解することができ，同時に，プログラムのデバッグを行うことができるようになる．

プログラムの処理の構造を理解するための記法としては，処理の流れを図解するフローチャートや，目的の視点でプログラムの構造を図解する HCP チャート[5] などがある．しかし，プログラムの実行時の状態を正しくイメージするための図解は十分であるとはいえない．

ある瞬間のプログラムの状態を図示するための手法として一般的によく用いられるものに，図 1 のよ

† 慶應義塾大学 政策・メディア研究科
Graduate School of Media and Governance, Keio University
†† 慶應義塾大学 環境情報学部
Faculty of Environmental Information, Keio University

番地	内容
100	101
101	'a'
102	
103	
104	

図 1　メモリイメージの図解サンプル

うなコンピュータのメモリイメージをモデル化したものがある．しかし，コンピュータの仕組みを意識したことのない学習者にとって，最初から実際のメモリの仕組みを想像するのは困難である．同時に，講師の側からしても，学習者にとって想像しにくいメモリイメージを用いた図解で，プログラムのデータの移り変わりを説明するのは困難である．

筆者らは，この問題を克服するために，3 種類の説明モデル「表モデル」「入れ子モデル」「参照モデル」を提案する．これらのモデルは，変数表を拡張した記法を用い，構造化プログラミングからオブジェクト指向プログラミングまで一貫した記法でプログラムの実行時の状態を図解することができる．

慶應義塾大学大岩研究室では，構造化プログラミングからオブジェクト指向プログラミングまでを教えるカリキュラムを作成し，大学でのプログラミング教育やシステムエンジニア育成のための新入社員教育などの場で実践している．今年度版のカリキュラムにおいてこれらのモデルを用いた教育を実践してみた．本稿ではこの実践の成果を報告する．以下，2 章では 3 種類のモデルの記法について具体例を使って述べ，3 章では試用した結果とモデルの有効性についての考察を行い，4 章で今後の課題について述べ，5 章でまとめを行う．

本稿ではまず，3 種類のモデルの記法について述べる．次にそれらを実際に使用した結果を示し，その有効性と今後の課題について述べる．

2. モデルの概要

2.1 表 モ デ ル

表モデルはプログラム実行時の変数と値の関係を表すためのモデルである．学習者は，表モデルを通じて，変数が評価されて値になるという考え方を身につけることができる．

変数と値の関係を表すためのモデルとして一般的

```
void run(){
  int a;                    //(1)
  a = 5;                    //(2)

  int[] b = new int[3];     //(3)
  b[0] = 4;                 //(4)
}
```

図 2　表モデルのサンプルプログラム

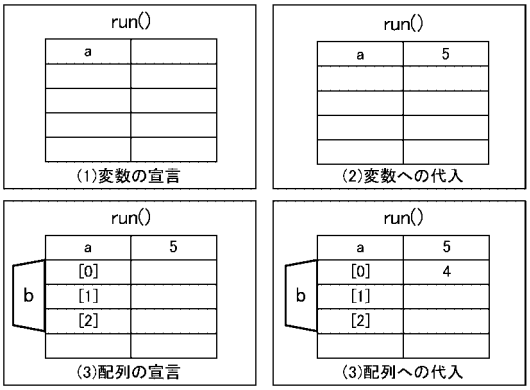

図 3　表モデルのサンプル

なものに，変数表がある，表モデルは変数表を拡張し，記法を定義したものである．表モデルでは，メソッドごとに変数の値が移り変わっていく様子を図に表す．通常の変数表と異なる点は，配列が変数のまとまりであることをわかりやすくするために，変数まとまりに配列が格納されている変数名のタブをつけて表すことである．

図 2 のようなプログラムがあったとき，(1) から (4) の各行が実行されたときのプログラムの状態を表す表モデルは図 3 の各状態に対応する．

2.2 入れ子モデル

オブジェクト指向プログラミングの導入教育において問題となるのが，オブジェクトがさまざまな値を持つ"もの"である，というイメージをつかむのが難しいことと，さらに，アドレス参照の考え方を身につけなければならないことの 2 つである．プログラミング言語として Java 言語を使った場合，学習者は 2 つの新しいことを同時に理解しなければならないため，個々の概念を余計に難しく感じてしまう．

入れ子モデルはオブジェクトの状態を図示するための，表モデルを拡張したモデルである．このモデ

```
void run(){
  Section section = new Section();     //(1)
  section.name = "sales";              //(2)

  Employee employeeA = new Employee();
  employee.name = "Yamada";            //(3)

  section.employees[0] = employeeA;    //(4)
}

class Section {

  String name;
  Employee[] employees;

  public Section(){
    employees = new Employee[3];
  }
}

class Employee {

  String name;
}
```

図 4　オブジェクトの構造をつくるサンプルプログラム

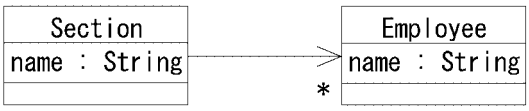

図 5　サンプルプログラムのクラス構造

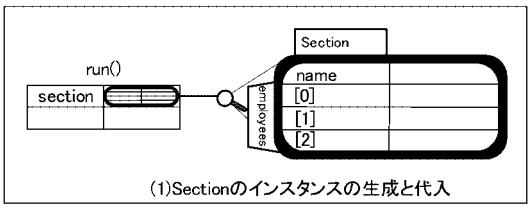

図 6　入れ子モデルサンプル (1)

ルは，参照の理解の問題を避けるため，参照の概念を理解しなくてもオブジェクトのイメージがつかめるよう考えられている．

入れ子モデルではオブジェクトは変数に代入されると，参照が渡されるのではなく，オブジェクトがコピーされて変数の値として格納されると考える．これは初学者にとって，オブジェクトがコピーされて変数の値として格納されるという考え方が，参照に比べて理解しやすいと考えたためである．記法も

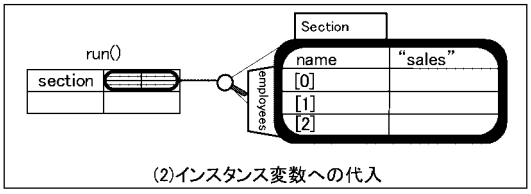

図 7　入れ子モデルサンプル (2)

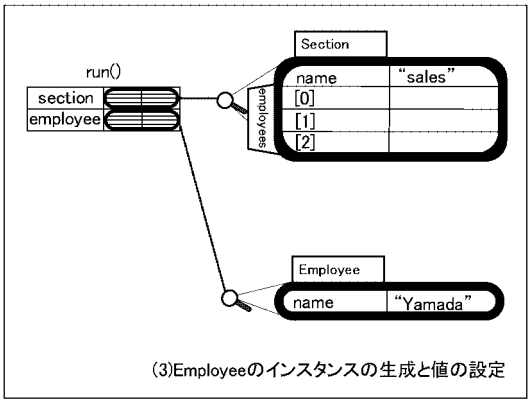

図 8　入れ子モデルサンプル (3)

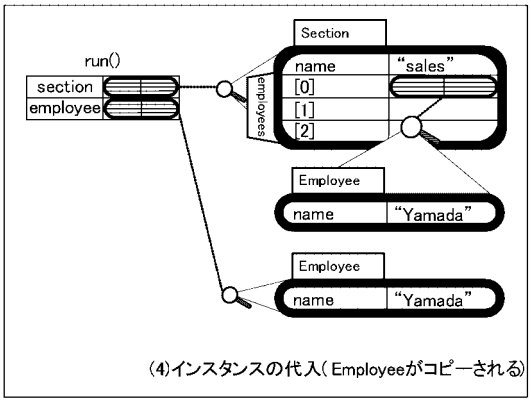

図 9　入れ子モデルサンプル (4)

それに合わせて，変数に対応するオブジェクトが値として中に入っているようなイメージのものになっている．(図 6)

会社組織における部署と社員を管理するサンプルプログラム図 4 が，図 5 のようなクラス構造を持つとき，(1) から (4) の各行が実行されたときのプログラムの状態を表す入れ子モデルは図 6 から図 9 のようになる．

入れ子モデルでは代入式が評価されるとオブジェクトがコピーされると考えるので，同じ参照を持つ

図 10　オブジェクト図のサンプル

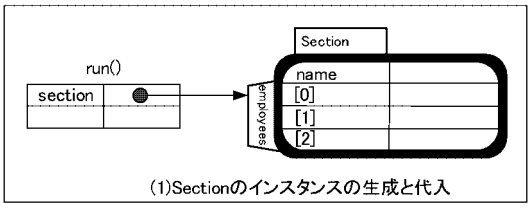

図 11　参照モデルサンプル (1)

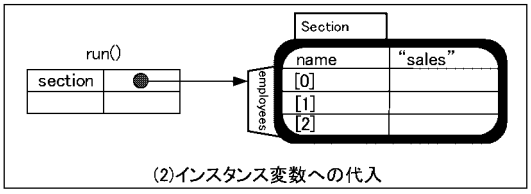

図 12　参照モデルサンプル (2)

2 つの変数があったとき，その関係を説明することができない．そのため，入れ子モデルを用いて説明する段階では参照の理解が必要な例題や課題は教育カリキュラムの中で扱わない．

2.3　参照モデル

参照モデルは，入れ子モデルを通じてオブジェクトのイメージがつかめた学習者に参照の考え方を教えるためのモデルである．参照モデルでは，変数に格納されている他のオブジェクトへの参照を丸と矢印で表す．

オブジェクト間の参照を表すための記法として一般的なものに，UML[3] に定義されているオブジェクト図がある．図 4 の (4) が実行されたときのオブジェクト図は図 10 のようになる．参照モデルがオブジェクト図と異なる点は，これまでの表モデルや入れ子モデルの記法を踏襲することで，変数と，変数に格納されるオブジェクトへの参照の関係を明確にしている点である．

入れ子モデルのときと同じプログラム (図 4)(図 5) を用いて参照モデルでの表現を説明する．(1) から (4) の各行が実行されたときのプログラムの状態を表す参照モデルは図 11 から図 14 のようになる．

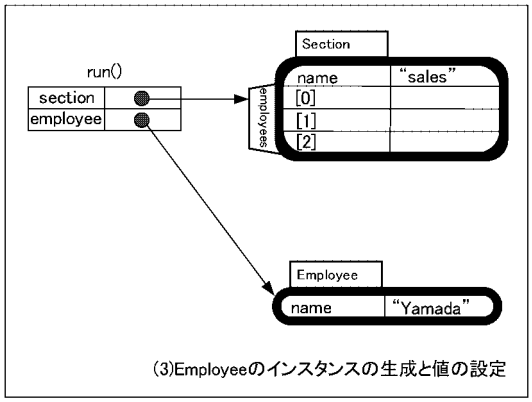

図 13　参照モデルサンプル (3)

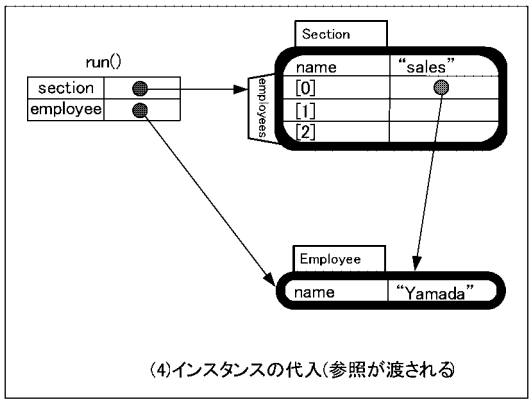

図 14　参照モデルサンプル (4)

表 1　カリキュラム実践環境

新入社員教育	プログラミング初心者	60 名
大学における授業	プログラミング初心者	約 80 名
社会人向けプログラミング講座	実務経験者 (オブジェクト指向は未修得)	31 名

3.　モデルの評価

筆者らは，本稿で提案する 3 種類のモデルを導入したプログラミング教育カリキュラム「人にやさしいプログラミングの哲学」を大学での授業，新入社員教育，社会人向けプログラミング講座の 3 つの場で実践した．実践を行った環境の詳細を表 1 に示す．本章では，その評価を行い，理解を助けるツール，コミュニケーションのためのツール，評価のためのツール，の 3 種類の観点からその有効性を考察する．

3.1　理解を助けるツールとしてのモデルの有効性

昨年度までに実施していたプログラミング教育カリキュラムでは，多くの学習者が概念の理解に苦し

む項目がいくつかあった．具体的には戻り値のある手続き，クラスとインスタンスの関係，オブジェクトのネットワーク構造の構築などである．今年度実践したカリキュラムでは，これらの概念の理解について昨年に比べ学習者の苦労が軽減されたという感触があった．理由としてはカリキュラム自体の改善もあるが，概念の説明にモデルを使用したことも理由のひとつである．

モデルの導入が学習者の理解を助けた理由として，表モデルをベースとする一貫した記法により，式が評価されて値になるという概念が学習者に定着したことが挙げられる．例えば，戻り値のある手続きに関して言えば，直接はモデルと関係がないが，表モデルで変数が評価されて値になったように手続きが評価されて値になる，という説明で昨年に比べ多くの学習者が概念を理解することができた．このことは，オブジェクトの構造を構築するという項目を説明する際にも同様の現象がみられた．

また，オブジェクト指向の理解についても，入れ子モデルと参照モデルの導入により，実体であるインスタンスが明瞭に理解され，それを定義するクラスの役割をはっきりさせることができた．その結果，両者の違いが理解できない学習者は昨年に比べ少なかった．

上に述べたようなことは，学習者を対象とするアンケートやヒアリング，課題提出の際の感想からも見てとることができた．

3.2 コミュニケーションツールとしてのモデルの有効性

通常，学習者がプログラミングの実習を行う中で何らかの問題につきあたっているとき，その原因がどこにあるのかを講師が瞬時に判断するのは難しい．また学習者自身にとっても，問題の原因に気づくのは難しい．例えば，学習者がオブジェクトを追加するアルゴリズムがわからないと言ったとしても，その原因が，アルゴリズムが理解できていないことなのか，クラスとインスタンスの関係がわからないことなのか，手続き，式と評価などより基本的な項目が理解できていないことなのかは瞬時にはわからない．

そのようなときでも，モデルを用い，プログラムの 1 行ごとの状態のイメージを学習者に書かせることで，どの段階で理解が曖昧になっているのかを把握することができる．例えば，入れ子モデルを正しく書くことができているかどうかで，クラスとインスタンスの関係まで理解できているかどうかがわかるといった具合である．

また，原因を特定した後の，学習者への説明のツールとしても，これらのモデルは有効である．これは，3 つのモデルが，1 行ごとの処理によってプログラムの状態がどのように変化するのを明確に示すことができるためである．例えば，new 演算子が評価されてインスタンスが生成される，というのは学習者にとっては抽象的で理解しにくい事柄である．しかし，入れ子モデルや参照モデルを使えば，インスタンスが生成されるということがどのようなことであるのかを，図で明確に示すことができる．

以上のことから，3 つのモデルは，講師が学習者が抱える問題に対して個別に対応する際のコミュニケーションのツールとして有用である．

3.3 評価ツールとしてのモデルの有効性

プログラミング教育の場において常に問題となることのひとつに，プログラミングに関する概念の，理解度の測定が挙げられる．一般的にプログラミングの概念理解を測るための試験では，概念の説明を文章で記述させる，実際に概念を使ったプログラムを書かせる，などの方法がとられている．

しかし，プログラミングに関する概念は，言葉で説明できるだけでは理解として不十分であり，実際にプログラムが実行されていく中でどのような処理が行われ，どのように状態が遷移するのかを理解している必要がある．またプログラムを実際に書かせる試験には問題に対する普遍的な正解がない．そのため学習者の解答が多種多様なものとなり，必ずしも解答が理解を測りたい概念を利用したものだけとは限らないので評価が困難である．

この問題を解決するために，理解を測りたい概念を使ったソースコードを学習者にみせ，概念と関係がある段階のプログラムの状態をモデルとして記述させるという方法が有効であると考えられる．概念を理解していない学習者は，ソースコードからプログラムの実行時の状態をイメージすることができないためである．例えば，クラスとインスタンスの関係を理解しているかを測るために，オブジェクトを生成し，操作するサンプルプログラムの参照モデルを書かせる．このような方法により，概念の理解度を測る試験に正解を設定することができるので，容易に概念を理解しているかを測ることができる．

モデルを書かせることによって，概念の理解を測るという試みは，前述した実践の場で実際に行われた．モデルを書かせると，概念の理解ができていない学習者は一目瞭然であった．それゆえ，3 つのモデルは評価のためのツールとしても有用であると考えられる．

4. 今後の課題

この章では実際にモデルを使ったカリキュラムを実践し，明らかになった問題点と，それに関する考察を行う．

まず，入れ子モデルについて考察する．入れ子モデルはオブジェクトのイメージをつかむという点では，学習者の理解を助けたが，一方で，オブジェクトが代入の際にコピーされるという入れ子モデルの考え方は，実際に Java 言語のプログラムで起こる動作とは異なるため，学習者を混乱させた面もあった．

教育カリキュラムでは，入れ子モデルでオブジェクトがコピーされると教えたすぐ後に，参照モデルが登場し，実はオブジェクトは入れ子になっているのではなく参照を持っているという説明を行う．結果としてまだ入れ子モデルの理解が完全ではない状態で，実は参照を持っているという説明をすることになり，理解に不安を感じている学習者を混乱させる原因となった．また，どの実践の場においても，学習者の中に若干他よりもプログラミング経験の多い学習者がいた．そのような学習者から本当はコピーされないのではないかというような質問がでると，実は入れ子モデルは事実と違うということを説明せざるを得ず，これも学習者を混乱させる原因となった．

しかし，クラスとインスタンスの関係と参照の概念を同時に学ぶことが学習者にとって難易度が高いのは事実である．そのため，今回実践した場よりプログラミングに関する知識の少ない学習者が多いようなときには，入れ子モデルはやはり必要であると思われる．さらなる実践を重ね，このような入れ子モデルの有効性と弊害について，検証することは今後の課題である．

実際のプログラムの動きと異なることの弊害は，表モデルにおける配列に関しても同様のことが言える．Java 言語における配列は参照型であるため，表モデルで説明できない場合がある．また，表モデルは二次元配列については考慮されていない．これらの点に関しても今後検討が必要である．

5. おわりに

本稿ではプログラム実行時の状態を図示するモデルの提案とその評価について述べた．プログラム実行時の状態を図示する試みとして，現在，Kent Beck らが開発中の Object Spider や，SmallTalk ベースのオブジェクト指向プログラミング教育環境である Squeak[2] などがあり，今後このような試みは広まっていくと思われる．本稿で述べた 3 種類のモデルも今後改良を加え，モデル作成用のツールなどの環境も整えたいと考えている．

謝辞

本研究を進めるにあたり (株)EXA の児玉公信様，井関知文様を始めとする技術部の皆様には大変お世話になりました．この場を借りて感謝いたします．

参 考 文 献

1) Gerald Jay Sussman,Harold Abeson,Julie Sussman. 計算機プログラムの構造と解釈 第二版. ピアソン・エデュケーション, 1999.
2) Mark J. Guzdial,Kimberly M. Rose. Squeak 入門. エスアイビー・アクセス, 2003.
3) Object Management Group. UML 仕様書. ASCII, 2001.
4) 松澤芳昭，岡田健，中鉢欣秀，大岩元．オブジェクト指向技術者養成のためのカリキュラム．情報処理学会研究会報告 (CE-64-1), pp. 1–8, 2002.
5) 竹田尚彦，大岩元. プログラム開発経験に基づくソフトウェア技術者育成カリキュラム．情報処理学会論文誌, Vol. 33, No. 7, pp. 944–954, 1992.
6) 長谷川聡, 山住富也, 小池慎一．プログラミング教育における制御構造のイメージと理解度について．情報処理学会論文誌, pp. 1180–1183, 1998.