# プログラミング環境Nigari - 初学者がJavaを習うまでの案内役

長 慎也[†] 甲斐 宗徳[††] 川合 晶[†]
日野 孝昭[†] 前島 真一[†] 筧 捷彦[†]

初学者がプログラミングを学習する場合，一般に，最初は簡単な概念を学び，次第に高度な概念を習得するという順序を踏む．学習環境も，その進行に沿ったものが用意できるとよい．最初は，難しい概念を知らなくても使える，取っ付きやすい言語や環境を与え，学習が進むに従って，高度な概念も扱える，Java 等の実用的な言語へ移行させるとよい．これによって，初学者にプログラミングに対する興味をもたせ，学習意欲を継続させることが可能であると考えられる．そこで，プログラミング学習の導入部において用いるのに適した言語 Nigari とその環境を開発した．

Nigari の言語仕様は，Java のそれを簡素化したものになっており，クラスやメソッドの宣言など，初学者にとって理解が難しいものを書く必要がない．一方，基本的な制御構造などは，Java とほとんど同じ仕様である．

Nigari の実行環境は，オブジェクトを自動的に可視化する機能をもつ．これによって，学習者のプログラミングへの意欲を向上させるだけでなく，オブジェクトの概念をも理解させることができる．

早稲田大学コンピュータ・ネットワーク工学科 1 年前期のプログラミングの授業を実験の場とした．この授業は本来 Java を用いて実習を行うが，導入部に Nigari を用いた．実験では，オブジェクトの可視化機能について学生から高い評価を得られた．また，言語を簡素にすることについても，ある程度の評価を得られた．

# Nigari - a programming language and environment for the first stage, leading to Java world

Shinya CHO,[†] Munenori KAI,[††] Akira KAWAI,[†]
Takaaki HINO,[†] Shin'ichi MAESHIMA[†]
and Katsuhiko KAKEHI [†]

When novice students learn programming, they usually start from basic notions and proceed to higher-level concepts. Teachers should provide materials adapted for students' progress: first, they should provide simple and easy programming language environment, then, shift to a practical one such as of Java, which can handle more sophisticated concepts.

We developed a programming language Nigari and its environment. It is designed as a workbench at the first stage of programming lessons. The language specification of Nigari is a subset of Java: control structures are almost equal to Java, but no class declarations and no method declarations are required to write. It is intended that students can shift to Java ultimately without confusion.

Moreover, the environment of Nigari has visualization feature that shows objects on a screen automatically. This feature fuels students' motivations to learn programming and helps their grasping basic concepts of "Objects".

We applied Nigari, as an experiment, in a lesson of programming in Department of Computer Science, Waseda University. The course had been designed to use Java originally. We provided Nigari to students as their startup environment. Questionnaire in and after the course showed that many students favored its object visualization feature and some appreciated its language simplicity.

## 1. プログラミングの授業に適した言語と環境

情報科学の授業は，まずプログラミングを教えるこ

† 早稲田大学 理工学研究科
Graduate School of Science and Engineering, Waseda Uniersity
†† 成蹊大学工学部 経営・情報工学科
Department of Industrial Engineering and Information Sciences, Seikei Uniersity

とから始まるが，そこで使われるプログラミング言語およびその環境は，次のような性質をもつことが望ましい．

- **学習の進行に合わせた言語**
  初学者が最初に習う概念は，変数や制御構造といった基本の仕組みであり，関数・メソッドやクラスといった高度な概念は，後になって習うことが多い．
  学習に使用する言語は，学習の進行の度合いに合わせて，初学者が今理解できる範囲のことがらだけを書くだけで動作するようにしておくと，混乱なく学習ができると考えられる．
  逆に，その時点で習っていない概念，例えば，関数・メソッドやクラスなどについて，必ずそれらの宣言を書く必要があるような言語では，その部分の説明をすることが難しく「おまじない」である，として教えざるを得ないことが多い．
- **すぐに実行できる言語と環境**
  プログラムを実行するまでに，多くの手順を踏む必要がある環境では，プログラムの実行がなかなか行えず，意欲を失うことになる．
  例えば，コンパイルをコマンドライン上で行うという作業は初学者にとって負担である．さらに，コンパイルの際に，変数宣言の不備や初期化のし忘れなどで，たくさんのエラーが発生した場合も，適切な対処ができず，行き詰まってしまい挫折感を味わうことが多い．
  そこで，コンパイル環境と実行環境とを統合したり，プログラム上に不備があってもある程度実行ができるようにしたりすれば，実行結果をすぐに見ることができ，学習意欲の向上につながると考えられる．
- **魅力的な出力**
  プログラムの出力結果が，アニメーションなどのグラフィックスを出力すると，単にコンソールに文字を出力するのと比べて，学習者の興味を格段に惹くことができると考えられる．
  これは，単にグラフィックスのライブラリを用意すればすむというものではない．なぜなら，初学者にとっては，そのライブラリを使ってグラフィックス出力を行うこと自体が十分に複雑な作業であり，結局「おまじない」として教えざるを得なくなるからである．
  したがって，プログラム中にグラフィックスに関する命令を何も書かなくとも，自動的にグラフィックスの出力を行われるような仕組みを用意し，魅力的な出力が簡単に作り出せるようにしておくことが必要であるといえる．
- **オブジェクト指向の基礎の体得**
  オブジェクト指向は，継承，ポリモルフィズム，カプセル化といった高度な概念を含み，初学者にいきなり教えるのは難しい．加えて，変数や制御構造など，非オブジェクト指向プログラミング言語にも共通の概念について，まず初めに教えておく必要がある．
  しかしながら，最初の段階においても，プログラムの実行中に,「オブジェクト」というものが存在する，という感覚を自然に得られるような環境においてやるとよい．
  実行しているプログラムにおけるオブジェクトの状態を，グラフィックス画面へ自動的に表示する仕組みがあれば，オブジェクトの存在を確認したり，オブジェクトの振る舞いを視覚的に把握したりできると考えられる．この仕組みは，前項で述べた，プログラムから自動的にグラフィックスを生成するための仕組みとしても利用できる．
- **既存の言語との親和性**
  現状のプログラミングの授業では，世の中の実情に対応した教育を行うという目的から，Java 言語（以下，Java）などの既存の言語を使用することが多い．それら既存の言語の理解を助けるような言語と環境を用意する必要がある．

こうした性質をもつ言語として，Java を簡素化した言語仕様をもつプログラミング言語 Nigari とその環境 Nigari System[1)] を開発した．これを早稲田大学のコンピュータ・ネットワーク工学科 1 年生に対するプログラミングの授業で実際に用いて，プログラミング入門と Java 入門を行ってみた．

## 2. Nigari の概要

プログラム言語 Nigari とそのプログラミング環境である Nigari System は，次のような特徴をもっている．

- **簡素な言語仕様**
  Nigari は，クラスの宣言やメソッドの宣言を書かなくても，文だけを書けばプログラムが動作する．このため，学習の初期段階においても，学習者がプログラムのほとんどの箇所を理解することができ,「おまじない」として教えなければならない事項がほとんどない．
  また，Nigari の変数は，宣言をすることなく使え，どのような型の値でも代入できるため，コンパイ

ル時にエラーが出る頻度が少ない．ユーザ（学習者）は，自分の書いたプログラムをすぐに実行してみて，その動作を確認することができるので，意欲的に学習に取り組むことができる．

- **プログラムの可視化**
  Nigari System には，ユーザが，特別なグラフィックス命令を書かなくても自動的にプログラムの動きを可視化する機能が組み込まれている．ユーザは，プログラムの流れを直感的に理解することができるし，興味惹かれるアプリケーションを簡単に作ることができる．
  また，グラフィックス画面に現れる画像を「オブジェクト」と見なし，そのオブジェクトの振る舞いを記述する，という流れでプログラミングを行っていくので，学習者にオブジェクトという概念を強く印象づけることが可能である．
- **Java に近い言語仕様**
  学習が進行すると，実用的な言語の習得も必要となる．Nigari の言語仕様は，式や制御文の構造などをほとんど Java と同じにして，実用言語として学ぶ Java への移行の際に混乱を来たさないようにしている．

## 3. プログラミング環境 Nigari System

まず，言語 Nigari のプログラミング環境である Nigari System について述べる．**図 1** に，Nigari System のスクリーンショットを示す．

Nigari System を用いたプログラムは，次のような手順で作成する．

- **ページを作成する・開く**
  ページとは，プログラムを作成し，動作させるための作業領域である，1 個のページには，ソースファイルおよびオブジェクトの配置（後述）を保存するファイルが格納される．
- **オブジェクトを配置・編集する**
  ユーザは，実行時に出現するオブジェクトを，実行前に予め配置することが可能である．オブジェクトの配置は GUI を用いて行う．配置されたオブジェクトは，メインウィンドウに表示される．また，オブジェクトインスペクタを用いて，配置したオブジェクトのもつ変数を閲覧・変更することが可能である．
- **プログラムを編集する**
  配置した各オブジェクトの動作記述（正確にはそのオブジェクトが属するクラスの記述）であるプログラムを，エディタを用いて編集する．
  通常，各オブジェクトは互いに異なるクラスに属しているので，それぞれの動作を独立に記述可能である．
- **コンパイル・実行する**
  メニューから**実行**を選ぶか，エディタに設置されたショートカットボタンの**実行**を押すと，システムはプログラムをコンパイルし，コンパイルエラーがなければ実行する．エラーがある場合はエラーメッセージを表示する．
  コンパイルと実行は 1 つの指示でまとめて行うため，ユーザが別々に指示を出す必要はない．
  実行中は，各オブジェクトがそのプログラムに応じて並行に処理を行う．それにともなって，オブジェクトはメインウィンドウ上に可視化される．可視化は，`x`，`y`，`p` という 3 つのオブジェクト変数（4.8 参照）の値を用いて行う．ここで，`x` はオブジェクトの x 座標，`y` は y 座標，`p` は絵柄の番号である．絵柄は 40 種類のあり，0〜39 の番号で指定する（5.3. 参照）．
  また，オブジェクトインスペクタを用いて，オブジェクトの変数を閲覧・変更することが可能であり，さらに変数の値の変化はリアルタイムに表示される．
  実行中はその実行を一時停止させる（メニューまたはショートカットボタンの**一時停止**）ことも可能で，途中の経過をじっくり観察できるようにも配慮した．
- **停止（終了）する**
  メニューまたはショートカットボタンの**停止**を選ぶと，実行中のプログラムが終了する．各オブジェクトの配置が，実行する直前の状態に戻り，再びオブジェクトやプログラムの編集が可能になる．

なお，プログラムを実行している状態を「実行時」，それ以外の状態（プログラムの編集やオブジェクトの配置をしている状態）を「設計時」と呼ぶ．

## 4. Nigari 言語仕様

### 4.1 データ型

データ型には整数型，実数型，文字列型，オブジェクト型，配列型，真偽値型，null 型がある．

- **整数型，実数型，文字列型，真偽値型，null 型:** それぞれ，整数，浮動小数点数，文字列，真偽値，`null` を表現する．これらの型の値は，代入操作によって内容そのものがコピーされる．
- **オブジェクト型:** オブジェクトへの参照を表現する．変数にオブジェクト型の値を代入すると，変

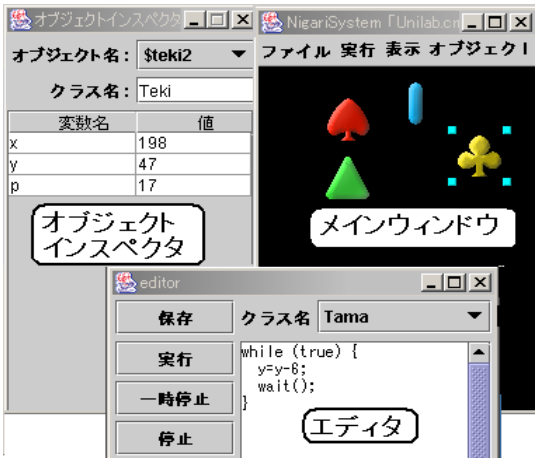



**図 1** Nigari System のスクリーンショット
Fig. 1 Screenshot of Nigari System

数にはオブジェクトへの参照がコピーされ，オブジェクトの内容そのものはコピーされない．

- **配列型:** 配列への参照を表現する．変数に配列型の値を代入すると，変数には配列への参照がコピーされ，配列の内容そのものはコピーされない．この特性はオブジェクト型と同じであるが，オブジェクト型と配列型（すなわち，オブジェクトと配列）は明確に区別される．

実装上は，ここに示したデータ型に加えて，左辺値型も使う．この型のデータは，左辺となった変数の記憶場所を示す．ユーザが直接意識して（例えば値の参照渡しをするなどの用途に）使用することはできない．

すべての値は，真または偽を表すものとして解釈することができる．整数または実数の 0，真偽値の `false`，および `null` は偽となり，それ以外は真となる．

また，変数には，あらゆる型の値が代入可能であり，配列の各要素には，任意の型の値を混在して格納できる．さらに，メソッドが返す値の型は呼出し毎に異なっていてもよい．

### 4.2 クラスの宣言とソースファイル

オブジェクトの動作を表すものをクラスと呼ぶ．クラスの宣言については，次のような規則を設けた．

- **1 個のソースファイルには，ただ 1 個のクラスの内容を記述する．**
- **記述されるクラスの名前はファイル名と同じである．**

この規則によれば，ファイル名からクラス名が一意に決定できるので，クラス名をファイル本体に書く必要がない．また 1 つのファイルに複数のクラスを書くこともないので，クラスの境界を示すための特別な区切りもいらない．このため，クラスの宣言であることを示す特別な構文を必要としない．

### 4.3 ソースファイルの構造

前述の通り，1 個のソースファイルには，ある 1 個のクラスを記述する．以下,「このクラス」とは，あるソースファイルが定義しようとしているクラスのことを指すものとする．

ソースファイルは，次の部分に分かれる．

- **extends 文**
  このクラスの親クラスを指定する．extends 文自体を省略すると `object` という名前のクラス（object クラス）を継承する．object クラスはシステムに組み込んであり，ユーザは作成できない．
  object クラスには，ユーザにとって必要な組み込みメソッド（A.2 参照）などがすべて定義されているため，継承の概念を習得するまでは，extends 文は特に意識する必要はなく，書く必要もない．
- **プログラム本体**
  本体には，次のいずれかを並べる
  - 文
  - メソッドの定義
  - コンストラクタの定義

### 4.4 メイン文列

プログラム本体に直接☆書かれた文の集まりをメイン文列と呼ぶ．

1 個のオブジェクトには，自動的に 1 個のスレッドが割り当てられ，それらのスレッドが並行に動作する（5.2. 参照）．このスレッドは，メイン文列を上から順に実行する．ただし文の途中に現れるメソッドやコンストラクタの定義は，無視される．

exntends 文，メソッド，コンストラクタを一切書かない場合，ソースファイル全体は，このメイン文列だけから構成される．すなわち，このオブジェクトの実質的な動作だけが書かれたソースファイルを記述することが可能である．

### 4.5 文

**文**には，**式文**，**if 文**，**while 文**，**do-while 文**，**for 文**，**break 文**，**return 文**がある．

#### 4.5.1 式 文

構文や（代入を含めた）演算子の意味は Java とほとんど同じであるが，変数にはあらゆる型の値が格納されている可能性があるため，式の値がどの型になるかは不定である．また，演算の振る舞いは，オペランドの型によって実行時に決定される．詳しくは仕様書[2)]

☆ メソッドの定義の内部に書かれているものではなく，という意味

を参照されたい．

#### 4.5.2 if 文，while 文，do-while 文，for 文，break 文

これらの制御文も Java とほぼ同じ意味をもつ．ただし，4.1 で述べたように，すべての値が真または偽として解釈できるので，条件式の値が真偽値型である必要はない．例えば，`while(1) ...`のような書き方も許容される（が，推奨されない．10.2.2 参照）．

#### 4.5.3 return 文

Java 同様，メソッドの戻り値を設定し，呼出し元に復帰する．

return 文はメイン文列に含めることはできず，メソッド内部でしか使えない．return 文の式を省略したり，return 文自体を省略すると，このメソッドの対象であるオブジェクト（`this`）が戻り値になる．

### 4.6 メソッドの定義

メソッドの定義では，このクラスがもつメソッドを定義する．

メソッドの定義は予約語 “`function`” で始まり，続いて，**識別子**（メソッド名），**仮引数リスト**，**ブロック**（処理内容）が続く（A.1 参照）．

**ブロック**の内部においては，**文**または **var 定義**（4.8 参照）を並べる．var 定義は文に先だって並べる必要がある．

### 4.7 コンストラクタの定義

コンストラクタは，4.9.2 で後述するように新しいオブジェクトが生成された場合に自動的に呼び出されるメソッドである．構文は，メソッドの定義における，予約語 “`function`” を “`constructor`” にしたものである．

### 4.8 変　　数

4.1 で述べたように，変数には，どの変数にもあらゆるデータ型の値を代入することができる．

変数は，次の 3 種類に分類される．

- **グローバル変数**
  どのオブジェクトからも共通して直接に参照できる共通な変数を，グローバル変数という．グローバル変数の名前は先頭に`$`がついたものに限る．
  グローバル変数は，変数宣言なしで使う．
  設計時（3. 参照）に配置したオブジェクトは，それぞれグローバル変数から自動的に参照される（5.4 参照）．
- **ローカル変数**
  ローカル変数は，あるメソッドにローカルな変数である．
  ローカル変数を使用するには，明示的に指定が必要となる．メソッドの**仮引数リスト**に記述されたもの，および，メソッドの**処理本体**内の，**var 定義**によって指定された変数がローカル変数になる．
  ローカル変数は，メソッド内でしか使えず，var 定義をメイン文列に書くこともできない．
- **オブジェクト変数**
  各オブジェクトに固有の変数を，オブジェクト変数という．オブジェクト変数は，そのオブジェクトからだけ直接に参照が可能である．
  オブジェクト変数は，宣言なしで使う．
  グローバル変数でなく，かつローカル変数でない（すなわち，先頭が`$`でなく，仮引数でもなく，var 定義で宣言されてもいない）変数はオブジェクト変数になる．
  ソースファイル中に出現したすべてのオブジェクト変数が，このクラスのオブジェクトがもつすべてのオブジェクト変数になる．このクラスのオブジェクトがもつべき変数の一覧は，コンパイル時に決定され，オブジェクト生成時には，その一覧に従ってオブジェクト変数が用意される．

### 4.9 オブジェクト指向の実現

Nigari には，オブジェクト指向プログラミングを可能にするための，次のような機能が用意してある．

#### 4.9.1 間 接 参 照

間接参照とは，任意のオブジェクトについて，そのオブジェクトの変数を参照すること，または任意のオブジェクトに対してメソッドを呼ぶことである．

任意のオブジェクトの変数を参照するには，「**式** “.” **変数名**」という式を用いる．**式**がオブジェクトを表す場合，そのオブジェクトの**変数名**で指定された名前をもつ変数を参照する．**式**がオブジェクトでない場合，または**変数名**で指定された変数がオブジェクトにない場合はエラーとなる．このエラーはコンパイル時には検出できないので，実行時に発生する．

任意のオブジェクトのメソッドを呼び出すには，「**式** “.” **メソッド名** “(” **引数リスト** “)”」という式を用いる．**式**がオブジェクトを表す場合，そのオブジェクトの**メソッド名**で指定された名前をもつメソッドを呼び出す．その際，**引数リスト**に指定した引数を渡す．**式**がオブジェクトでない場合，または**メソッド名**で指定されたメソッドがオブジェクトにない場合はエラーとなる．このエラーも実行時に発生する．

#### 4.9.2 オブジェクトの動的生成

オブジェクトは，設計時に（間接的に生成して）配置するほかに，実行時にプログラムで生成することもできる．

生成には,「“new” **クラス名** “(” **引数リスト** “)”」という式を用いる．この式は，**クラス名**で表されるクラスのオブジェクトを新規作成し，そのオブジェクトに対してコンストラクタ（4.7 参照）を呼び出し，さらにそのオブジェクトを式の値とする．

実行時に生成されたオブジェクトにも，その場でスレッドが自動的に割り当てられ，他のオブジェクトと一緒になって，並行にそのプログラムを実行する．

なお，以下では，設計時に生成されたオブジェクトを設計時オブジェクト，実行時に生成されたオブジェクトを実行時オブジェクトと呼ぶ．

### 4.10 配列の扱い

配列は，複数の変数を要素として格納可能なデータである．Nigari の配列は，実行時に動的に作成され，配列型の値によって参照される．

配列に格納できる要素の個数（要素数）は，配列を作成する際に任意に設定できるが，一度作成した配列の要素数は変更することができない．

- **配列の作成**
  作成には組み込みメソッド（4.11 参照）の `array` メソッドを利用する．第 1 引数に作成したい配列に必要な要素数を数型で指定する．戻り値として，新しく作成された配列を参照する配列型の値が返される．
- **要素への参照**
  配列への要素を参照するには,「**式**$_1$ “[” **式**$_2$ “]”」という式を用いる．
  この式は，**式**$_1$ の値が配列であり，**式**$_2$ で表される値が 0 以上要素数未満の整数である場合，**式**$_2$ の値をインデックスとした **式**$_1$ の配列の要素を表す．それ以外の場合は実行時にエラーとなる．
  多次元配列はなく，配列の要素に配列型の値を代入することで同等の機能を実現する．
- **要素数の取得**
  配列の要素数を得るには,「**式** “.” “`length`”」という式を用いる．**式**は配列でなければならない．要素数を表す整数値が評価結果になる．
  構文的にはオブジェクトに対する間接参照の形を借用しているが，全くの別物である．

### 4.11 組み込みメソッド

マウス入力･キーボード入力･グラフィックス描画☆を行うメソッドや，標準的な数学関数などが用意されている．その他にも次のようなメソッドがある．なお，組み込みメソッドのすべてを A.2 に示す．

- `wait`
  文字通りの処理を待機させるという働きの他に，他のオブジェクトに処理を譲るという働きを併せもっている．詳しい働きは 5.2 で述べる．
- `die`
  オブジェクトを画面から消去し，オブジェクトに割り当てられたスレッドを破棄する．
- `array`
  引数で指定された個数の要素数をもつ新しい配列を作成し，その配列への参照を表す配列型の値を返す．

☆ 可視化機能だけでは描画できない線や円などの図形を補助的に描画するためのもの．

## 5. 実　　装

Nigari System は，プラットフォームによらず使えることを目指して，Java で実装した．実際，Linux（Vine 2.5）でも Windows（2000/Me/XP）でも，JDK がインストールしてあれば動作する．

Nigari System の実装は，大きく分けて次の部分からなる．

- 翻訳系
- 実行系
- 可視化機構
- ページ管理機構
- コントローラ

Nigari System では，作られたアプリケーションをページと呼ぶ．ページは，初期画面上に配置されたオブジェクトと，それらに関連した一群のクラスから成る．

### 5.1 翻　訳　系

翻訳系は，ソースファイルを翻訳し，実行系（Nigari VM，以下単に VM）のコードからなる列（コード列）を生成して，VM がそのコード列を実行できるようにする働きをもつ．

コード列はクラス毎に生成され，コード列が表す処理内容はそのクラスのメイン文列を実行する，というものである．

### 5.2 実行系（VM）

VM は，翻訳系が生成したコードをもとに，Nigari のプログラムを実行する．

Nigari のプログラムは，各オブジェクトがそれぞれの処理を並行に実行するという仕様をもつ．それらの実行を管理するための仕組みをスレッドという．このスレッドは，自ら能動的に処理を実行するのではなく，VM からの指令によって受動的に処理を行う．このような制御を行うために，このスレッドは，Java の

Thread オブジェクトを用いず，独自に実装してある．

各スレッドは，一時的な計算領域としてスタックをもち，コードに従ってスタックに対する各種操作を行う．なお，コードの詳細は仕様書[2] を参照されたい．

VM の処理手順は次のようになる．

(1) **スレッドの割当て**
プログラムが実行されると，まず VM は，実行開始にあたって，各設計時オブジェクトにスレッドを 1 個ずつ割り当てる．また，実行時オブジェクトが生成された場合（4.9.2 参照）にもスレッドを 1 個割り当てる．

(2) **スレッドの処理の実行**
VM は，各スレッドを順繰りに呼び出して，割り当てられたオブジェクトのクラスのコード列を解釈実行させる．スレッドは，`wait` メソッドの呼出しが起きるか，ある一定数（1000 程度）のコードを実行するかしたときにその回の作業を終了する．VM は引き続いて次のスレッドに同様の処理をさせる．この結果，スレッドはそれぞれ並行に実行される．
スレッドは，コード列をすべて実行し終えるか，`die` メソッドの呼出しが起きるかしたときに，VM から削除される．なお，`die` メソッドが呼ばれない限り，削除されたスレッドが割り当てられていたオブジェクトは画面上に残ったままになる．

(3) **可視化**
すべてのスレッドに対して上の処理を 1 回ずつ行わせた後，可視化機構（5.3 参照）に，各オブジェクトの可視化を行うように指示する．

(4) **待機・繰り返し**
オブジェクトの動きが人間の目にもわかる程度の速さにするため，数十ミリ秒待機してから，(2) に戻る．この動作を，プログラムが停止するまで（ユーザが停止の指示を出すまで）繰り返す．

翻訳系，実行系，可視化機構の動作をまとめると，**図 2** のようになる．

### 5.3 可視化機構

可視化機構は，存在しているすべてのオブジェクト（設計時オブジェクトおよび，実行時オブジェクト）を，メインウィンドウ（3. 参照）に画像として表示する．

可視化機構は，各オブジェクトのオブジェクト変数 `x`，`y`，`p` の値を参照し，その値に基いて，メインウィンドウ上の位置 (`x`,`y`) に，`p` 番目の絵柄を表示する．

なお，すべてのクラスの親クラスである object クラスに，変数 `x`，`y`，`p` というオブジェクト変数をもたせるように実装してあるため，どのオブジェクトも `x`，`y`，`p` というオブジェクト変数（4.8）をもつ．

### 5.4 ページ管理機構

ページ管理機構は，次のような操作を担当し，ページに関連する設計時オブジェクトや，関連するクラスのソースファイルを管理する．

- ページの保存，読込み
- ソースファイルの保存，編集，読込み
- 設計時オブジェクトの追加
- 設計時オブジェクトの編集

ページの情報は，cmml と呼ばれる形式でファイルに格納される．cmml 形式[2] のファイルには，次の情報を格納する．

- ページが使用するクラスとそのソースファイルの一覧
- 各設計時オブジェクトの情報
  - 名前（詳細は後述）
  - 属するクラス
  - 変数名とその値の組

設計時オブジェクトには，それぞれに一意な名前をユーザが指定できる．さらに，実行時には，各オブジェクトがその名前と同じ名前のグローバル変数から自動的に参照されるようになる．このため，設計時オブジェクトの名前は，グローバル変数の命名規則に合致するものだけが指定できる．

### 5.5 コントローラ

コントローラは，上で挙げた各機構を，ユーザからの要求に応じて動作させる．その主な処理は次のようなものである．

- **ページ作成・読込み:** ページ管理機構を用いて，ページの作成や読込みを行う．
- **編集・保存:** ソースファイルの編集や，オブジェクトの編集，またそれらの保存をページ管理機構を介して行う．
- **コンパイル・実行:** 翻訳系を用いて，ソースファイルのコンパイルを行い，コンパイルエラーがなければ実行系を用いて実行を行う．エラーがある場合は実行せずユーザにエラーメッセージを表示する．
- **停止:** 実行系が実行しているプログラムを止める．

## 6. プログラム例

### 6.1 変数の値によるアニメーション

**図 3** に，Nigari のプログラムの一例を示す．このプログラムを実行したオブジェクトは，オブジェクト

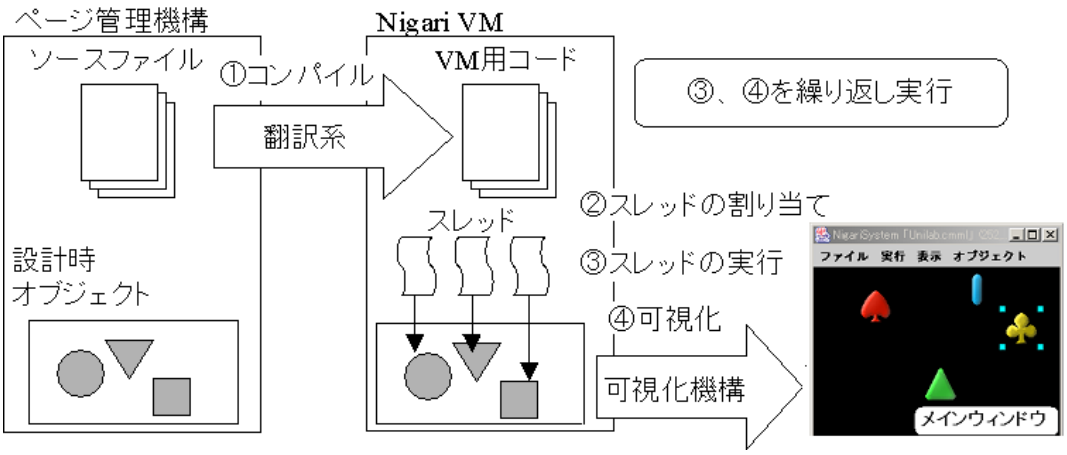



**図 2** 翻訳系，実行系，可視化機構の動作
Fig. 2 Mechanism of compiler, VM and visualizer

```
while(x<300) {
  x=x+5;
  wait(10);
}
```

**図 3** Nigari のサンプルプログラム（1）
Fig. 3 A sample program of Nigari (1).

変数 x の値を 5 ずつ増やし，組み込みメソッドである wait メソッドを呼ぶ，という動作を繰り返す．このオブジェクトは，Nigari System のメインウィンドウ上では，右に 5 ドット移動し，少し（約 10 ミリ秒）待つという動作の繰返しとして観察される．これは，3 のメインウィンドウの項目で説明したように，変数 x，y，p の値に応じてそのオブジェクトが自動的にメインウィンドウ上に表示されるからである．

### 6.2 マウス入力と if 文

**図 4** に示したプログラムは，オブジェクトが，自分自身と，マウスカーソルの x 座標の位置関係によって，左右に移動する（マウスカーソルのある方向に寄ってくる）プログラムである．組み込みメソッドの getMouseX を用いて，マウスの位置と自分の変数 x の値を大小比較し，比較結果に応じて動作を変化させている．

### 6.3 他のオブジェクトの参照

**図 5** に示したプログラムは，このプログラムで動作するオブジェクトとは別のオブジェクト $player を追跡するプログラムである．

このプログラムを正しく実行するには，追跡の対象となるオブジェクトを設計時に $player という名前で配置しておく（5.4 参照）．

```
while(true){
  if(x<getMouseX()) x=x+1;
  if(x>getMouseX()) x=x-1;
  wait(10);
}
```

**図 4** Nigari のサンプルプログラム（2）
Fig. 4 A sample program of Nigari (2).

```
while (y<180) {
  if (x>$player.x) x=x-1;
  if (x<$player.x) x=x+1;
  y=y+1;
  wait(10);
}
```

**図 5** Nigari のサンプルプログラム（3）
Fig. 5 A sample program of Nigari (3).

$player.x は，オブジェクト $player がもつ変数 x を間接参照している．

### 6.4 メソッドの定義

**図 6** に示したプログラムで動作しているオブジェクトは，別のオブジェクト $player との衝突を検知すると，自分自身を消去する．ここでは，diff というメソッドが定義されている，これは 2 つの引数の値の差（絶対値）を返すメソッドである．これを用いて，自分と $player の x,y 座標を比較し，x,y ともに差が 20 未満であれば die メソッドにより消去を行っている．

### 6.5 オブジェクトの動的生成

**図 7** に示したプログラムで動作しているオブジェクトは，マウスボタンが押されるたびに，新しく Tama

```
while(true) {
  x=x+1;
  if (diff(x,$player.x)<20 &&
      diff(y,$player.y)<20) {
    die();
  }
  wait();
}
function diff(a,b) {
  if (a>b) return a-b;
  return b-a;
}
```

**図 6** Nigari のサンプルプログラム（4）
Fig. 6 A sample program of Nigari (4).

```
while (true) {
  if(getKey(1)==1){
    t=new Tama();
    t.x=x;t.y=y;
  }
  wait(10);
}
```

**図 7** Nigari のサンプルプログラム（5）
Fig. 7 A sample program of Nigari (5).

クラスのオブジェクトを生成する．生成したオブジェクトを変数 t に代入し，間接参照を用いて，新しいオブジェクトの位置を，自分と同じ位置に設定している．

## 7. 実　　験

Nigari の有効性を確認するため，Nigari の特徴を最大限に生かしたコースデザインを行い，実際の大学の授業で使用した．その結果を報告する．

### 7.1 教材作成の方針

まず，Nigari に特化した形での，プログラミング学習のための教材[3] を用意した．

この教材では，他の多くの教材で見られるような，「代入」「分岐」「繰り返し」といった，プログラミングを構成する要素を個別に取り上げて説明することはなるべく避けた．大きな目的として「アニメーションを作る」というものをまず掲げ，その目的を実現するためにプログラムを作らせる，という方針をとった．例えば，繰返しを使えば，アニメーションを連続して行わせることが可能であり，分岐を使えば，マウスなどの入力に反応する動作ができるなど，学習者が作りたくなるような動きを実現するための手段として，プログラミングの様々な要素を自然に導入できるように配慮した．

なお，作成したテキスト（教本）はすべて Web 上で閲覧可能としておいた．演習問題やアンケートの解答も Web を利用した．また，授業のサポート用に BBS も設置した．

### 7.2 科目の概要

- **科目名:** プログラミング A
- **対象学科・学年:** 早稲田大学理工学部，コンピュータネットワーク工学科（CS 学科）1 年
- **目標:** Java 言語とプログラミングの基礎の習得
- **期間，回数:** 2003 年 4 月 14〜同年 7 月 7 日，12 回
- **1 回あたりの授業時間:** 2 時限（約 3 時間）
- **人数:** 約 240 人
- **クラス構成:** 3 クラスに分かれて授業．授業用資料や課題は担当教員により異なる．クラス分けは各学生の出席番号を 3 で割った余りで決めた．
- **教科書:** 「Java 言語プログラミングレッスン（上）」[4]
- **試験:** クラス試験，共通試験 1 回ずつ．クラス試験はクラスによって内容が異なる．共通試験は，Web を用いて行った．
- **授業形態:** 各自ノートパソコン持参，授業 1 回ごとに，説明を受け，演習を行う

### 7.3 授業の進め方

3 つのクラスのうち，クラス 1 は Nigari System を利用し，クラス 2，3 については利用しなかった．1 クラスの授業内容を**図 8** に示す．今後，特に断りがない限り，クラス 1 の授業についてのみ述べる．

クラス 1 は，6 月 9 日の授業までは Nigari System を利用し，6 月 16 日以降に Java の説明と実習を行った．

一回の授業は 5，6 個程度の節に分かれていた．各節については，説明を行い，演習問題（練習問題）を解かせ，その問題を解説する，という手順を，30 分から 40 分程度で行った．基本的には全員，一斉に同じ問題を解かせた．なお，プログラム作成を伴う問題を「演習問題」，それ以外の穴埋めなどの問題を「練習問題」と呼び分けた．

教室内にいるスタッフは，講師 1 人，TA4 人（いずれも本論文の著者または共著者）で，TA は学生の挙手に応じて個別に対応を行った．

#### 7.3.1 Nigari を使った授業

Nigari を使った授業では，変数，制御構造などの基礎を習得させた．Nigari を使った授業の実習において出題した演習問題の一部を**図 9** に示す．また，5 月 19 日の「if 文」の項目で使用したテキストの一部を**図 10**

| | |
|---|---|
| 4/14 | 講義概要 |
| 4/21 | PC の使い方とプログラミング |
| 4/28 | Nigari のインストールと試用 |
| 5/12 | 変数 / while 文 |
| 5/19 | while 文（つづき） / if 文 |
| 5/26 | if 文（つづき） / マウス入力 |
| 6/02 | 複数のオブジェクト/マルチスレッド |
| 6/09 | メソッド / オブジェクトの実行時生成<br>（6/16 から Java を使用） |
| 6/16 | Java の概要 / 変数の宣言 / while 文 |
| 6/23 | 制御構造/メソッド/文字列 |
| 6/30 | 配列 / コマンドライン引数 / 並べ換え |
| 7/07 | 並べ換え（つづき） / 文字入力 |
| 7/14 | （補講） クラス試験 |

**図 8** 授業内容（クラス 1）
Fig. 8 Lessons for classroom 1

に示す．なお，全授業のテキスト[3] は Web にて参照されたい．

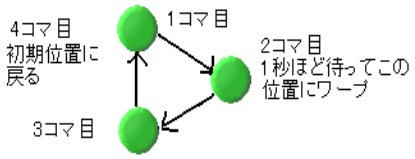
授業で用いた Nigari System には，学生が Nigari で作成したプログラムを，Nigari 専用のファイルサーバにアップロードする仕組みを追加し，演習問題の提出の手間を低減させ，学習履歴が追跡できるように配慮した．このアップロードの仕組みは，Web と連携して，教師側が解答状況をその場で閲覧できるようになっており，各学生の演習の進捗を把握したり，よくある間違いをその場で解説する，といった用途にも利用した．

#### 7.3.2 Java を使った授業

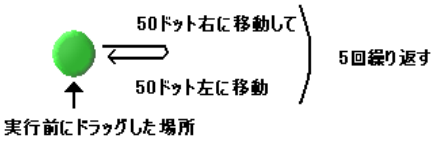
Java を使った授業では，JDK のコマンド（javac, java など）をそのまま利用させるのではなく，コンパイルや実行などを簡単に行えるアプリケーション（JavaEditor）を開発・提供した．**図 11** にスクリーンショットを示す．JavaEditor にも，Nigari 同様にファイルを自動的にアップロードする機能を付加した．

また，Java の授業では，その日学ぶ Java の構文についての説明を行う前に，**図 12** で示したような質問を行い，Nigari での経験から類推させてみる，ということを試みた．例えば，問題 8-2 は，型の概念や変数宣言を教える前に，問題 9-1 は，Java の while 文を教える前に，問題 10-2 は，Java のメソッドの書き方を教える前に出題したものである．以下，このような質問を類推テストと呼ぶ．この類推テストは，Nigari で習得したことが，そのまま Java に応用できるかどうかを試験し，Nigari の学習効果を測定する目的で行った．

**5/12（変数）** 次の図のように，三角形の頂点を右回りで移動して最初の場所に戻る 4 コマのアニメーションを作りましょう．




**5/19（while 文）** 次の図のように，左右に 5 往復するオブジェクトを作りましょう．




**5/26（if 文）** マウスカーソルを追いかけるオブジェクトを作りましょう．

**6/2（複数のオブジェクト）** ２つのオブジェクトが万有引力の法則に従って運動する様子をシミュレートしましょう．

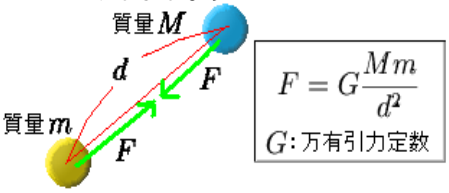



**図 9** 演習問題（抜粋）
Fig. 9 Practices

### 7.4 アンケート

次のような調査をアンケートにて行った．アンケートはすべて Web にて回答する方式をとった．

- **経験調査**
  コンピュータの用途，使用頻度，プログラミング経験の有無などを，自己申告で回答してもらった．クラス 1 は第 2 回（4 月 21 日），そのほかのクラスは 5 月中旬頃に回答した．これは，パソコンの操作方法を習う時期がクラスによって異なり，Web を使ってアンケートに答えられるまでの時間に差が出たためである．**表 1** に，クラス 1 におけるプログラミング経験の有無の結果を示す．

**マウスを使ってオブジェクトを動かす**

この章では if 文に関する学習を行いますが、その前の準備段階としてマウスカーソルの座標を得る方法をまず学習します。mouseCheck という名前の新しいページを作り、次のプログラムを実行してみましょう。マウスカーソルの位置にオブジェクトが表示されます。マウスを動かすと、マウスカーソルも動きオブジェクトも一緒に動きます。

```
while(true){
  x= getMouseX();
  y= getMouseY();
  wait(50);
 }
```

**if 文とは**

さて、本題の if 文についての説明に入ります。

if 文 (if statement) は、ある条件の時にだけこの処理をしたいといった、条件付きの動作を行うプログラムを作りたいときに if 文を用います。if 文は、つぎの形に書きます。

```
if( 条件１){
  処理１
}
```

条件１が成り立って場合にだけ処理１が行われ、条件１が成り立っていない場合には処理１は行われません。

**オブジェクトの動きに変化をつける。**

if という新しいページを作り、プログラムを実行してみましょう。

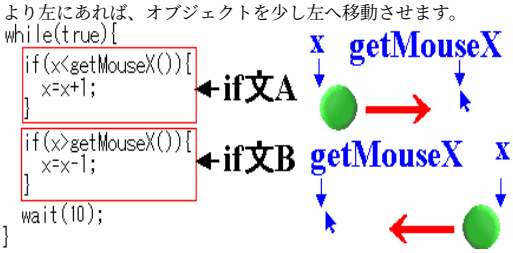
プログラム 4-3

```
while(true){
  if(x<getMouseX()){
    x=x+1;
  }
  if(x>getMouseX()){
    x=x-1;
  }
  wait(10);
}
```

このプログラムでは図のように二つの if 文が用いられています。一つ目の if 文を if 文 A、二つ目の if 文を if 文 B とします。

if 文 A では条件としてマウスカーソルがオブジェクトより右にあるかどうかを判定し、 図のようにマウスカーソルがオブジェクトより右にあれば、オブジェクトを少し右へ移動させます。

if 文 B では条件としてマウスカーソルがオブジェクトより左にあるかどうかを判定し、 図のようにマウスカーソルがオブジェクトより左にあれば、オブジェクトを少し左へ移動させます。




**演習問題**

getMouseY メソッドを使って、マウスカーソルがオブジェクトよりも上にあるとオブジェクトがゆっくりと上に進み、マウスカーソルがオブジェクトよりも下にあるとゆっくりと下に進むプログラムを作ってみましょう。

**マウスカーソルとオブジェクトの距離によってオブジェクトが動いたり、動かなかったりさせる。**

次のようなプログラムを実行させてみましょう。オブジェクトとマウスカーソルの横方向の距離が 100 以上ある場合、オブジェクトは移動しません。

```
while(true) {
  if(x<getMouseX()){
    if(getMouseX()-x<100){
      x=x+1;
    }
  }
  if(x>getMouseX()){
    if(x-getMouseX()<100){
      x=x-1;
    }
  }
  wait(10);
}
```

**条件と説明**

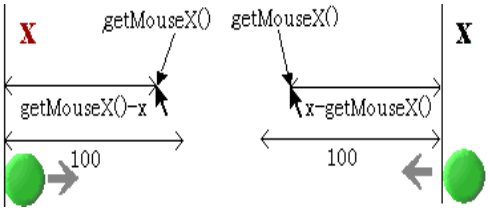
「オブジェクトとマウスカーソルの横方向の距離が 100 以上あるときは動かない」⇒「オブジェクトとマウスカーソルの横方向の距離が 100 未満のときだけ動く」

「x ＜ getMouseX」 マウスカーソルがオブジェクトより右側にあるかを判定

「getMauseX()-x ＜ 100」 マウスカーソルとオブジェクトの距離が 100 未満かを判定

「getMouseX ＜ x」 マウスカーソルがオブジェクトより左側にあるかを判定

「x-getMouseX() ＜ 100」 マウスカーソルとオブジェクトの距離が 100 未満かを判定




**演習問題**

逆に、マウスカーソルとオブジェクトが 100 以上離れていた場合だけ、動くようにするプログラムを書きましょう。

**図 10** 授業で用いたテキスト（抜粋）
Fig. 10 Textbook used in classrooms

- **毎回の授業評価（4 月 28-6 月 30 日）**
  クラス 1 では，毎回の授業の出席点呼の代わりに，「授業の分量」「授業の難しさ」「授業の楽しさ」「演習問題の達成度」の 4 項目を回答してもらった．

- **類推テストの結果（6 月 16 日，6 月 23 日）**
  類推テスト（7.3.2 参照）を行った日は，授業の最後に，これらをどのようにして解いたかを回答してもらった．選択肢には「今日学んだ項目はすでに知っていたので解けた」「知らなかったが類


**図 11** JavaEditor のスクリーンショット
Fig. 11 Screenshot of JavaEditor

推だけで解けた」「類推では解けなかったが，解説を聞いてから解けた」「まだ解けていない」の 4 種類を用意した．

- **総合評価（7 月 7 日）**
  授業全体を通して，理解度や感想を問うアンケートを授業の最終日（補講を除く）に実施した．全クラス実施したが，クラス 1 については，Java と Nigari の違いや，Java の学習における Nigari の効果などについて問う設問も用意した．

## 8. 実 験 結 果

クラス 1 のアンケートの集計結果を 8.1〜8.5 に示す．クラス 2，3 との比較結果を 8.6 に示す．

### 8.1 授業の難しさの変化

**図 13** に，アンケート結果に基づく，授業の難しさの，授業ごとの変化を示す．

5 月 12 日の時点では，変数の基礎的な概念の習得が内容であり，ほとんどの学生が問題なく理解していると考えられる．それ以降，`if` 文，`while` 文の概念を習得するにつれ，難しさが上がっていく．これは，`if` 文，`while` 文などの個々の概念の難しさを示すというよりは，プログラムが複雑になり仕上げるのが難しくなったことを示している．

6 月 16 日は最初の Java での授業であるが，難しさが低下している．これは Nigari で習得した変数や while 文の概念をそのまま Java に応用することができたからだと考えられる．

**問題 8-2：** 何と出力されるでしょう

```
public void printHello() {
  int x=2+3;
  int y=x*2;
  x=x+y*5;
  System.out.println(x);
}
```

**問題 9-1：** 同じ動作を while 文で書きましょう

```
int i=2;
System.out.println(i+"の 2 乗は"+i*i);i++;
System.out.println(i+"の 2 乗は"+i*i);i++;
System.out.println(i+"の 2 乗は"+i*i);i++;
System.out.println(i+"の 2 乗は"+i*i);i++;
```

**問題 10-2:** 何と出力されるでしょう

```
public void printResult() {
  int m=keisan2(-5);
  System.out.println(m);
}
public int keisan2(int a) {
  int c=keisan(a,a);
  if (c<0) return c*2;
  return c;
}
public int keisan(int x,int y) {
  return x*y;
}
```

**図 12** 類推テストの一部
Fig. 12 An example of “analogy tests”

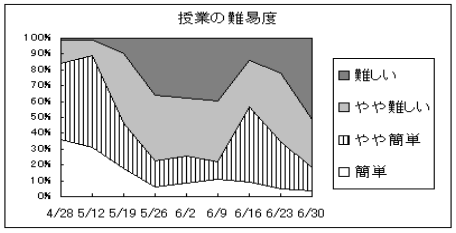



**図 13** アンケート結果：授業の難しさの変化（クラス 1）
Fig. 13 Changing of difficulty in classroom 1

### 8.2 類推テストの解き方

**図 14** に，類推テストの解き方に関する回答を示す．

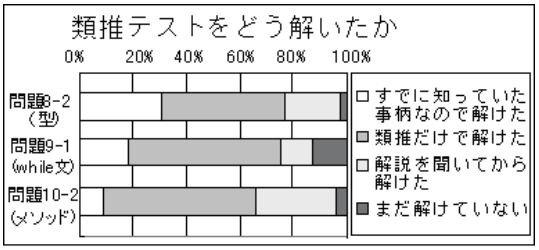



**図 14** アンケート結果：類推テストをどのように解いたか
Fig. 14 Questionnaire: How “analogy tests” are solved.

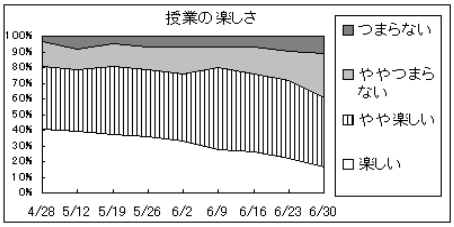



**図 15** アンケート結果：授業の楽しさの変化（クラス 1）
Fig. 15 Changing of interest in classroom 1

半分強の学生が，まだ習っていない Java の構文に関する問題を，Nigari の知識だけで解くことが可能であった．ただ，メソッドの書き方については，Java での構文を習って初めて解答できた学生も多かった．

### 8.3 授業の楽しさの変化

**図 15** に，授業の楽しさの，授業ごとの変化を示す．

授業の難しさが上がるにつれ，若干ながら楽しさは減少するが，難しさの上昇に比べて緩やかな減少で，「難しいけれどおもしろい」と感じている学生が多いと考えられる．Java の授業に入ると，楽しさの減少が目立つようになる．

### 8.4 Nigari を用いた授業の利点と欠点

7 月 7 日実施の 1 クラスのみ対象の最終アンケート（A.3 参照）に,「この授業の良かった点」「この授業の改善してほしい点」「その他，授業に関する意見をご自由に」の 3 つの自由記入欄を設置した．これらの欄に書かれた意見のうち，Nigari に関する意見が 30 件含まれた．その詳細を**表 2** に示す☆．なお,「肯定（否定）的意見の詳細」における括弧内は，7.4 で述べた能力調査のアンケートにて，プログラミングを事前に経験していた学生の内数を示す．

Nigari の利点として,「プログラムがアニメーションとして視覚的に表示されてわかりやすい・楽しい」「初心者にとってとっつきやすい」といった意見が多く見られた．また「オブジェクト指向，マルチスレッドなどの概念を理解する手助けとなった」という意見もあった．

一方，Nigari を学習する時間によって，本来の Java の学習ができる時間が減ったという意見が見られた．この意見はプログラミングの経験者から特に多く寄せられた．また，配列や文字入力など，Nigari では扱わなかった仕組みを Java で学習しなければならないので，Nigari を用いることに疑問を感じる学生もいた．

なお，Nigari 以外に関する自由意見には，肯定的な意見として「教師，TA によるサポートが充実していた」「Web 教材が使いやすかった」など，否定的な意見として「授業演習のみで，レポートがなかった」「授業の進み方が速すぎる」といったものがあった．

**表 1** アンケート結果：プログラミング経験（クラス 1）
Table 1 Questionnaire: Prior experience of programming

| | |
|---|---|
| プログラミング経験あり | 27 名 |
| プログラミング経験なし | 54 名 |

**表 2** アンケート結果：自由意見（クラス 1，括弧内はプログラミング経験者の内数）
Table 2 Questionnaire: Opinions (in classroom 1, parenthetic number indicates the number of experienced students in programming)

| | |
|---|---|
| 授業に対する意見総数 | 92 件 |
| Nigari に関する肯定的意見 | 19 件 |
| Nigari に関する否定的意見 | 11 件 |
| 肯定的意見の詳細: | （件） |
| 敷居が低くとっつきやすい | 7(3) |
| プログラムが視覚的 | 7(5) |
| オブジェクト指向の理解の助けになる | 2(2) |
| その他 | 5 |
| 否定的意見の詳細: | (件) |
| 早く Java に移ってほしかった | 6(4) |
| Java とのギャップを感じた | 3(2) |
| Nigari ではカバーできない点が多い | 1(0) |
| 簡単すぎる | 1(1) |

### 8.5 Java の学習に Nigari を用いることの是非

**図 16** に,「Nigari を使ったことで Java の理解が深まったか」という質問に対する回答の割合を示す．これについては，半数強の学生が「深まった」と答えた．一方，15%の学生が「かえってわからなくなった」と答えている．

**図 17** に,「望ましい授業スタイルはどれか」という質問に対する回答の割合を示す．約半数の学生が，今回採用した Nigari を導入に用いて，その後 Java に移行するというスタイルを支持したが，3 割の学生は，Nigari を使わずに Java を最初から学習するのがよいと答えた．

☆ 肯定的意見数に対して，その詳細の合計が 2 だけ多いのは，1 個の欄に 2 種類の特徴を挙げたものが 2 件あったため．

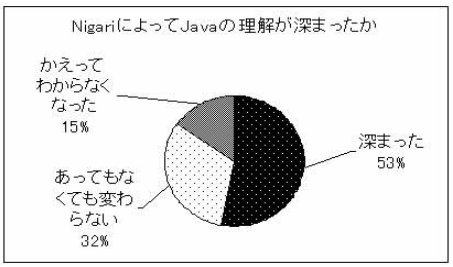



**図 16** アンケート結果：Nigari によって Java の理解が深まったか
Fig. 16 Questionnaire: Whether Nigari supported Java

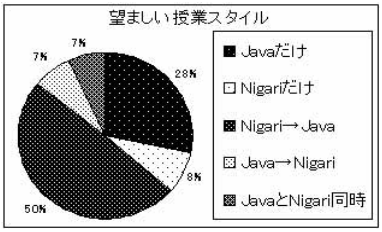



**図 17** アンケート結果：望ましい授業スタイル
Fig. 17 Questionnaire: Best style for the lesson

このように，大方の学生からは，Nigari を用いることに対して賛同を得られたが，Nigari は用いないほうがよいという意見も少なくなかった．その理由として，Java 特有のメソッドの宣言方法など，Nigari とは異なる仕様の部分に違和感を覚え，混乱してしまう，と述べたものがみられた．

### 8.6 クラス 2,3 との比較

#### 8.6.1 共 通 試 験

共通試験の結果を**図 3** に示す．

クラス 1 の成績が最も高い．ただし，共通試験を Web で実施した際に機器上のトラブルが起きたことを考慮して眺める必要がある．トラブルはクラス 2 で多発し，クラス 3 で若干問題が発生したものの，クラス 1 ではほとんど発生なかった．

各設問別の成績（各設問を 1 点満点とした場合の平均点）を**図 18** に示し，問題内容を**図 4** に示す．クラス 2，3 は後の問題になるほど点が低い．これは機器上のトラブルで，後の問題に手をつけられなかったのが原因と考えられる．それ以外に有意なクラス別の特徴を発見することは難しい．

#### 8.6.2 前期最終アンケート

授業で学習した項目についてどれほど理解ができたか（自己申告）を**図 19** に示す．

クラス 1 では，メソッド，配列の部分が特に落ち込んでいるが，その要因として次のようなことが考えられる．

- Java でのメソッドについては，Nigari で扱った例が少なかったこと，Java での宣言形式が Nigari のそれと比べて複雑であることなどから，理解しにくくなった．類推テスト（8.2 参照）の結果から見ても，メソッドの構文は Nigari と Java で違いを感じた学生が多かったことがわかる．
- 配列については，Nigari では学習せず，Java になって初めて学習した．さらに，学習時間が少なかった．

**表 3** 共通試験結果
Table 3 Results of common exam

| クラス | 平均点 |
|---|---|
| 1 | 85.8 |
| 2 | 75.4 |
| 3 | 81.1 |

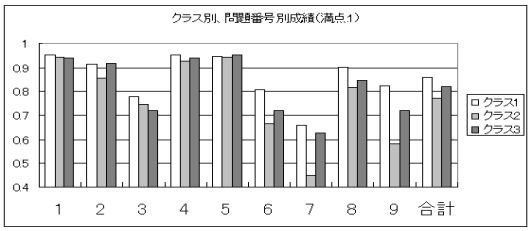



**図 18** 共通試験結果：問題別
Fig. 18 Results of common exam.(by categories)

**表 4** 共通試験問題内容
Table 4 Categories of common exam.

| 問題番号 | 内容 |
|---|---|
| 1 | if 文（プログラムの追跡） |
| 2 | if 文（条件式） |
| 3 | if 文（条件式） |
| 4 | switch 文 |
| 5 | while 文 |
| 6 | 型 |
| 7 | 配列 |
| 8 | オブジェクト指向 |
| 9 | 文法エラー訂正 |

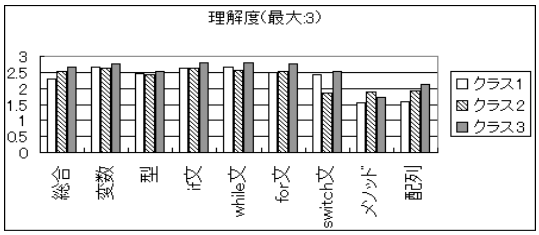



**図 19** アンケート結果：クラス別，分野別理解度
Fig. 19 Questionnaire: Understandings by classes and categories

#### 8.6.3 後期最終アンケート

後期には，前期のプログラミング A を引き継ぐ形

で「プログラミング B」が実施された．後期終了後の 2004 年 3 月上旬，メールで事後アンケートを実施した際に，他のクラスの学生からも感想を聞く機会を得た．

その中でも，クラス 3 の学生からは「教科書に沿って学習したのがよかった」「アプレットのデモなどがあって動機づけになった」など好意的な回答が多かった．クラス 3 の授業は，教科書[5] の内容にほぼ沿った形で，開発環境に JDK を用い，コンソールアプリケーションの作成を主な演習の内容としていた．さらに，アプレット等，GUI やアニメーションを用いたアプリケーションも，少しだけ触らせ，それらのデモを見せる機会を設けていた．

しかし，クラス 3 では，GUI やアニメーションの作成を本格的に実習させてはいない．そうした機会が増えると，さらなる動機づけになるという要望もあった．これは，限られた授業期間で Java の重要事項を漏れなくに教えるには，複雑な GUI などは実習を省略して，デモを見せる程度にしないと間に合わないからである．

これらの事柄をまとめると，次のようなことがいえる．

- 決められた教本に従って教えることが望ましい．
- アニメーションなどをサンプルとして見せるだけでも，動機付けの効果があるが，実際に学生自身に作らせるほうが望ましい．

これらの事柄を踏まえて，前期のクラス 1 の授業を見直してみると，クラス 1 は演習問題としてアニメーションの作成自体を盛り込むことができたため，上で述べた意味での動機付けの効果は十分にあったと考えられるが，反面，実行環境の違いから特殊な教材にする必要があり，教科書に沿った授業を進めづらくなったという課題が残ったといえる．

## 9. 授業を通じて得た知見

著者らが 1 クラスの授業の TA 業務を行った際に得られた知見を述べる．

### 9.1 学生からの質問

Nigari System を用いていたので，学生からの質問の多くは,「このようにプログラムを作ったのだけれど，期待通りに動作しない」という類いの，プログラミングの本質にかかわるものであった．一方，JDK を直接利用していた 2002 年度のプログラミングの授業においては，コンパイルエラー，実行時エラーの頻発に悩まされた．しかもその原因は，作業ディレクトリとソースファイルのあるディレクトリが違う，コンパイルしていないために古いクラスファイルの内容で実行しているといった，プログラミングの本質とは直接関係ないところに起因するものが大半であった．

### 9.2 学生が起こしがちな間違い

プログラム上のミスでもっとも多かったのは，“{” と “}” の対応が合っていないことによる文法エラーであった．これは，学生にインデントを行わせる習慣をつけさせなかったためだと考えられる．

今後の授業では，学生に対して，インデントを行うように働き掛ける予定である．もちろん，エディタが自動インデントを行うようにしてもよいが，プログラムの構造を理解させる意味では手動で行ったほうがよい．Nigari は，プログラムの構造が単純であるので，必要なインデントもあまり深くなく，初心者にとっても大きな負担ではないと考えられる．

### 9.3 授 業 態 度

実習中，学生は非常に大人しく黙々と課題をこなしていた．アニメーションが簡単に作れるということを，面白がっているような反応は直接には見られなかったが，毎回のアンケートからはプログラミングを楽しんでいることが見受けられた．その中でも，万有引力のシミュレーションを行う課題（図 9 参照）は興味を惹かれた学生が多く，授業の休み時間中に積極的に質問をしてくる学生もいた．

### 9.4 Java への移行

Java へ移行した後も，JavaEditor という専用の環境を用意したため，2002 年度の授業に発生したような問題はあまり発生しなかった．また，Nigari は Java と違って，すべてのオブジェクトに自動的にスレッドを割り振るが，これが原因で引き起こされた混乱はほとんど見られなかった．前期の授業では，Java でオブジェクトを複数作らせるような場面が少なく，使用した教科書にもそのような例があまりなかったためと考えられる．

ただ，後期の授業であるプログラミング B においては複数のオブジェクトを積極的に扱うような局面があり，その時点で何らかの問題が起きたかもしれないが，残念ながら調査を実施するには至らなかった．

## 10. 考 察

### 10.1 実験で得られた効果と問題点

実験結果から，Nigari には次のような効果があると言える．

- **簡単な言語仕様をもつ Nigari を用いて，プログラミング学習の敷居を低くすることができる．**
- **Nigari System によるプログラムの視覚化によって，学習者の興味を惹くことができる．**

- **Nigari を用いて，Java の基本的な仕組み，特に変数や制御構造，オブジェクト指向の概念を習得させることができ，それがそのまま Java に応用できる．**

一方，現在の Nigari には次のような問題点があると言える．

- **Java を詳しく学習したい学生にとっては，Java を学習する時間が減ってしまう．**
  この問題は，学習者の能力がすでに高い場合に顕著になる．そこで，能力別にクラス分けを行い，例えば，プログラミング経験がない学生だけに Nigari を使わせるといった改善案が考えられる．
- **クラスやメソッドの宣言など，Java と Nigari で仕様が違っている部分について，混乱を来たす可能性がある．**
  学生の意見によれば，Nigari から Java へ移行したときに，両者の言語仕様の違いから戸惑った学生が多かった．この場合，次のような方法で両者の違いを埋め合わせるような処置を行うとよいと考えられる．
  (1) Nigari の仕様を，Java に近づける
  (2) Java のプログラムを，Nigari のように簡素に記述できる環境を作る
  現在，(2) の方法での改善を進めている．その詳細は 11.2 で後述する．
- **Java の入門として教えるべき内容のすべてを網羅できているわけではない．特に配列や入出力についての学習を補助する仕組みがない．**
  Nigari でのプログラムの可視化機能は，変数 `x`，`y`，`p` を用いた画像表示にとどまっている．この仕組みだけでは，配列のようなデータ構造を視覚的に表示することはできない．これについての改善案は，11.1 で示す．

### 10.2 言語仕様の再検討

#### 10.2.1 排他制御の扱い

一般に，複数のスレッドが同時に同じデータにアクセスした場合には様々な問題が生じうる．Nigari では，オブジェクトがそれぞれ独立したスレッドをもって並行に動作するため，グローバル変数への同時アクセスや，他オブジェクトのメソッドを呼び出すことを介しての，オブジェクト内の変数への同時アクセスなど，原理的には競合が生じる場面がたくさんある．

しかし，Nigari には，このような問題を回避するための排他制御機構を特に用意していない．これは，初心者にはマルチスレッドを体感させることが第一であり，アニメーションという題材では，厳密な排他制御がなくてもそれなりの動作結果が得られるからである．排他制御を扱うのは，Java に移行したからで十分であると考えている．

#### 10.2.2 真偽値の扱い

Nigari は，すべての値が真または偽としても解釈可能になっているが，これは Java とは異なる仕様である．実際の授業では，分岐や反復の判定には必ず真偽値を用いた説明（例：`while(1)` ではなく `while(true)` と説明）を行っているので問題はほとんどないが，思いがけない動作につながる可能性もあるので，真偽値判定を真偽値型に限るように仕様を変更する予定である．

#### 10.2.3 変数の型と実行時エラー

Nigari は変数に型がない言語であるので，実行前に型のチェックをすることができない．そのため，型の誤りに起因して実行時エラー起こりうる．ただ，授業で用いたケースではほとんど実行時エラーは発生しなかった．その理由に，扱う型のほとんどが数値型であったこと，他のオブジェクトを参照する場合でも，変数 `x` や `y` を触る程度であることなど，あまり複雑なデータを扱っていないことが挙げられる．

Nigari で扱うデータをこのような単純なものに限定すれば，型の概念についてはあまり意識させる必要はない．複雑なデータの扱いは，11.2 で示すように Java に移行してから習得すればよいと考える．

## 11. 今後の予定

今後は，Nigari そのものの改良はもとより，プログラミングの授業全体を支援するための，コースウェアの開発に取り組む方針である．

例えば，今回の実験では，Java の授業を行う際に用意した環境は前述の JavaEditor だけであったが，Java に移行してからも，1. で述べたような特徴をもった環境を取り入れていく必要がある．

現在は，前節で挙げた問題点を踏まえて，次のような改善を進めている．

### 11.1 可視化機能の強化

従来の，変数 `x`，`y`，`p` を用いた画像の表示に加えて，次のような可視化を行うように機能を強化する

- すべてのオブジェクトの，すべての値を画面上に表示する
- オブジェクト間の参照関係を矢印で表示する
- 配列の要素を，文字またはグラフで表示する

この機能は，例えば，**図 21** で表されるようなグラフィックスを，実行とともに自動的に表示して，プログラム実行中におけるオブジェクトの状態表現[6] を動的に行うことを可能にしている．

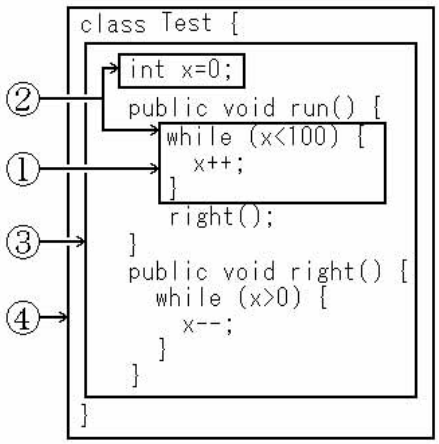



**図 20** 段階的にソースを見せる例（Java）
Fig. 20 Step-by-step code presentation (in Java)

また，この可視化機能は，Nigari のプログラムだけでなく，Java 等他の言語のプログラムを実行している場合でも，同様に可視化が行えるような枠組みで作成してある．現在，Java のプログラムを，この可視化機能を用いて可視化する仕組みも並行して開発している．

### 11.2 段階的にソースを見せる機能

学習対象のプログラミング言語が，クラスの宣言など，まだ習っていない概念についての記述を要求している場合（1. 参照），その部分を意識させないようにしておくことで現在学習すべき部分だけに注意を向けることができる．

Nigari では，言語仕様を簡素化することでその仕組みを実現し，実際に一部の学生から評価が得られた．しかし，現在の方法では授業が Java ベースになった瞬間にすべての部分が見えるようになるため，戸惑う学生も多かった．

そこで，いくつかの段階を作り，最初の段階では Nigari を用いたプログラムを作成し，Java に移行した後は，Java プログラムのごく限られた範囲だけを学習者に提示し，学習が進むにつれて，提示する範囲を徐々拡大していくような仕組みを作ればよい．この場合，提示されていない部分のコードは，環境によって自動生成されるようにしておく．

段階の例を**図 20** に示す．その概要は次のようになる．

- **段階 1** では，Nigari を使って，具体的な手続きだけを書く
- **段階 2** で Java に移行し，変数宣言の概念を追加する．なお，図 20 のように，提示すべき部分が分割される可能性があるので，エディタ等に工夫を凝らす必要があるかもしれない
- **段階 3** では，手続き自体がメソッドに囲まれていることを示し，メソッドを他にも定義したり，メソッド呼出しを行ったりする
- **段階 4** では，クラスの定義の構文（ファイル全体）を示す

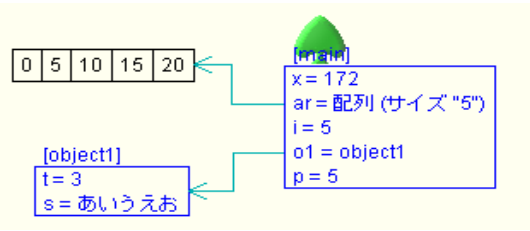



**図 21** Nigari の可視化機能の拡張
Fig. 21 An enhanced visualization feature

このような，学習の進行に合わせて，プログラミングスタイルに制約をもたせる仕組みをもったシステムとして，ProfessorJ[7] や，DrScheme[8] などが挙げられる．それらは，特定の構文の記述を禁止することで，不用意な操作を防止するのが主な目的であり，ユーザは依然としてプログラムの全体を書く必要がある．一方，ここで提案した手法は，プログラムの一部を書くだけで，全体を自動的に生成する仕組みをもっており，ユーザの負担は軽いと考えられる．

### 11.3 教材の作成

8.6.3 で述べたように，Nigari に特化した教材と，実際に用いられている Java の教材との違いをうめ，円滑な授業が進められるようなコースデザインをする必要がある．それには，11.2 で示したような，環境を Java に合わせた形に改良することの他に，Nigari 向けの教科書やカリキュラムを整備することも重要である．

今回の教材は，クラス 1 の授業では，Nigari を使った演習問題を作成し，授業中に演習問題として提示したものの，量的に不足が目立ち，授業以外での実習課題（レポート）が用意できなかった．事前に問題集を数多く作成しておき，授業中の演習問題や課題の充実させる必要がある．

## 12. 関 連 研 究

ここでは，言語の簡素さ，可視化機構，Java の学習という 3 つの観点から，Nigari に類似した環境やその適用事例について，本研究と比較検討する．

### 12.1 簡素な仕様をもつプログラミング環境

言語仕様を簡素にすることで，プログラミングに対する敷居を低くする試みは多く行われている．典型的

な例として，日本語でプログラムを記述する言語[9)10)]が挙げられる．しかしながら，日本語で書くプログラミング言語は，Java のような英語を基本にした言語とは書式に大きな差が出てしまうため，Java に移行した際に逆に負担を与える可能性がある．

一方，なるべく C 言語などの一般的な言語の書式を保ったまま，言語仕様を簡素化した言語に，若葉[11)]がある．しかし，これは手続き型言語の学習を目標として作られており，その目標に合わせて徹底した簡素化が行われている．そのため，構造体をあえて使えなくしているなどの特徴があり，オブジェクト指向の習得につながるような教材ではないと考えられる．

### 12.2 可視化機能をもつプログラミング環境

Squeak[12)] は，Smalltalk の環境の一つで，プログラミングの機能をもち，ユーザが自由に絵を描き，その絵に対して，プログラムを付加することで絵を動かすことができる．

日本では主に，小学校，中学校などの授業で実験的に利用されている[13)] が，授業目的としては，学生が自ら作品を作りだすための総合的な力を養う，といった性格が強く，プログラミングの学習を主眼においているわけではない．

また，そのプログラミング環境自体も，マウス操作が主体であり Java におけるプログラミングスタイルと大きく違い，直接 Java への応用が期待 できる可能性は低い．

Dolittle[14)] は，主に中学，高校の情報教育において，コンピュータの動作原理をプログラミングの体験を通じて学ぶことを主眼とした教材である．LOGO をベースにしたプログラミング環境をもち，実行結果がタートルグラフィックスとしてすぐに可視化される．

Dolittle を授業に適用した事例[15)] によれば，学生の反応，プログラミング に対する理解度はともに良好であったという． その主な要因は，Nigari 同様，アニメーション機能，言語の簡素さであった．

しかし，Dolittle は Nigari と違って，Java などの実用的な言語へ移行していくことについて，特に考慮していない．実際，LOGO のような初心者向けの言語を習得させても，C 言語の理解の助けには必ずしもならなかった，という報告[16)] もある．LOGO による入門から，実用的な言語への移行が容易であるかどうかについて，さらに研究が必要である．

### 12.3 Java の学習支援環境

DrJava[17)] は，Java の学習支援環境の一つであり，対話的に命令を実行する機能を備える．この機能を用いれば，値の参照や変更を任意のタイミングで行え，プログラムの振る舞いをいつでも確認することが可能である．

ただ，この機能はあくまで補助的なもので，学生が積極的に使おうとしなければ，まったく使われない可能性もある．実際，この DrJava は，今回の実験の対象者であった学生が後期の授業にて用いたが，この対話的実行の機能はほとんど活用されていなかった．学生の多くは,「動作しているプログラムの状態を確認してみよう」ということを明確に意識することがないのが原因と思われる．

一方，Nigari はオブジェクトの内部状態を常に自動的に表示しており，ユーザが特に意識をすることなく，プログラムの振る舞いを自然に理解できると考えられる．

BlueJ[18)] はまずクラスからオブジェクト作り，そのオブジェクトに対してメソッドを呼ぶ，という操作を真っ先に教える，”Object first”というアプローチに基づいて設計された Java の学習環境である．BlueJ には，オブジェクトの作成や，メソッドの呼び出しを対話的に行う機能があり，メッセージパッシングの機構を学生に強く印象づけられると期待される．

実験結果[19)] によると，大多数の学生が，BlueJ を用いることで楽しく，わかりやすく学習できたという．Nigari 同様，一部の上級者には，簡単すぎるという不満もみられたが，これをもって，BlueJ は初心者に適切な環境であったとしている．

一方，本研究では，オブジェクト指向プログラミングにおいても，基本となる概念は手続きであるという考えに基づき，まず手続きの基本概念を学び，その手続きを 行う主体がオブジェクトである，といったアプローチで授業を行った．そのため Nigari では，手続きの流れを可視化することによってわかりやすく表示する機能をもつ．BlueJ にもオブジェクトの可視化機構があるが，Nigari と比べて，手続きの動作中に何が起きているか，ということを把握する用途にはあまり適さない．

## 13. ま と め

Java 言語への導入として，Java に似ていて，かつ簡素な言語を用いて Java の基礎を学習するための言語 Nigari とその環境 Nigari System を提案した．

実験では，最初に Nigari を用いて，その後 Java へ移行するという授業を実施し，この方法によって学習者への負担を減らし，学習者の興味を持続させることが可能であり，しかも Nigari で学習したことが Java にも適用できることを示した．

しかしながら，学生の意見などを考慮すると，Nigari の言語としての簡素性が，その後の Java の学習に良い影響を与えているとばかりは言えない面がある．一方，Nigari の可視化の仕組みは多くの学生から歓迎されたと考えてよい．そこで，Java のプログラムそのものを可視化する仕組みや，Java のプログラムを簡単に書けるような仕組みを開発して，Nigari と Java の移行をスムーズに行えるような環境を構築する予定である．

## 参 考 文 献


1) 長慎也, 川合晶, 日野孝昭, 前島真一: Nigari System (2003). http://taurus.kake.info.waseda.ac.jp/Nigari/.
2) 長慎也, 川合晶, 日野孝昭, 前島真一: Nigari System 仕様書 (2003). http://taurus.kake.info.waseda.ac.jp/nspec/.
3) 長慎也, 川合晶, 日野孝昭, 前島真一: 早稲田大学 CS 学科プログラミング A 教材 2003 年度版 (2003). http://taurus.kake.info.waseda.ac.jp/03pa/.
4) 結城浩: Java 言語プログラミングレッスン (上), ソフトバンク パブリッシング (1999).
5) 結城浩: Java 言語プログラミングレッスン (下), ソフトバンク パブリッシング (1999).
6) 青山希, 松澤芳昭, 松浦学, 川村昌弘, 大岩元: 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, 情報処理学会研究会報告 (CE-72-15), pp. 109–114 (2003).
7) Gray, K. E. and Flatt, M.: ProfessorJ: a gradual introduction to Java through language levels, *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ACM Press, pp. 170–177 (2003).
8) Felleisen, M., Findler, R. B., Flatt, M. and Krishnamurthi, S.: The DrScheme project: an overview, *SIGPLAN Not.*, Vol. 33, No. 6, pp. 17–23 (1998).
9) 中鉢欣秀, 大岩元: 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, 情報処理学会研究報告「プログラミング」No.013 - 006, pp. 31–36 (1997).
10) 橋本裕, 早川栄一, 並木美太郎, 高橋延匡: プログラミング学習を支援する言語処理系「NB2」の設計, 情報処理学会研究報告「コンピュータと教育」, Vol. 47, No. 5, pp. 33–40 (1998).
11) 吉良智樹, 並木美太郎, 岩崎英哉: 初心者入門用言語「若葉」の言語仕様と処理系の実装, 情報処理学会トランザクション「プログラミング」, Vol. 40, No. SIG10, pp. 28–38 (1999).
12) Ingalls, D., Kaehler, T., Maloney, J., Wallace, S. and Kay, A.: Back to the future: the story of Squeak, a practical Smalltalk written in itself, *Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ACM Press, pp. 318–326 (1997).
13) 軽野宏樹, 木實新一, 上林弥彦: ALAN-K プロジェクト：Squeak を活用した創造的な情報教育の試み, 情報処理学会研究報告「コンピュータと教育」, Vol. 69, No. 1, pp. 1–8 (2003).
14) 兼宗進, 御手洗理英, 中谷多哉子, 福井眞吾, 久野靖: 学校教育用オブジェクト指向言語「ドリトル」の設計と実装, 情報処理学会トランザクション「プログラミング」, Vol. 42, No. SIG11, pp. 78–90 (2001).
15) 兼宗進, 中谷多哉子, 御手洗理英, 福井眞吾, 久野靖: 初中等教育におけるオブジェクト指向プログラミングの実践と評価, 情報処理学会トランザクション「プログラミング」, Vol. 44, No. SIG13, pp. 58–71 (2003).
16) 和田勉: LOGO を用いた「プログラミングの世界」への導入教育の経験, 情報処理学会研究報告「コンピュータと教育」, Vol. 92, No. 77, pp. 9–18 (1992).
17) Allen, E., Cartwright, R. and Stoler, B.: DrJava: a lightweight pedagogic environment for Java, *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, ACM Press, pp. 137–141 (2002).
18) Kölling, M. and Rosenberg, J.: Guidelines for teaching object orientation with Java, *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, ACM Press, pp. 33–36 (2001).
19) Smith, P. A. and Boyd, G.: Introducing OO concepts from a class user perspective, *J. Comput. Small Coll.*, Vol. 17, No. 2, pp. 152–158 (2001).


## 付 録

### A.1 EBNF による言語仕様

**字句要素:**

- **予約語** := “`while`” | “`if`” | “`is`” | “`else`” | “`null`” | “`for`” | “`do`” | “`function`” | “`constructor`” | “`extends`” | “`native`” | “`new`” | “`return`” | “`this`” | “`var`” | “`true`” | “`false`”
- **区切り記号** := “`+`” | “`-`” | “`*`” | “`/`” | “`%`” | “`=`” | “`&&`” | “`||`” | “`<`” | “`>`” | “`.`” | “`,`” | “`(`” | “`)`” | “{” | “}” | “`[`” | “`]`” | “`++`” | “`--`” | “`+=`” | “`-=`” | “`*=`” | “`/=`” | “`%=`” | “`==`” | “`!=`” | “`<=`” | “`>=`”
- **識別子** := 英字 { 英字 | 数字 } （注：予約語を

除いたもの）
- **グローバル** := “$” { 英字 | 数字 }
- **文字列** := “"” { “"” 以外の文字列 | “\” 任意の文字列 } “"”
- **注釈** := “/*” で始め，0 個以上の文字を並べ（“*/” という文字の並びを含まないこと），“*/” で終える．または，“//” で始め，行末で終える．
- **数字列** := { 数字 }
- 英字 := “a”-“z” | “A”-“Z” | “_”
- 数字 := “0”-“9”

**構文:**
- ソースファイル := extends 文　内容
- extends 文 := [“extends” **識別子** “;”]
- 内容 := { メソッド定義 | コンストラクタ定義 | 文列 }
- メソッド定義 := “function” **識別子** “(” 仮引数リスト “)” ブロック
- コンストラクタ定義 := “constructor” **識別子** “(” 仮引数リスト “)” ブロック
- 仮引数リスト := **識別子** { “,” **識別子** }
- 文列 := { 文 }
- 文 := if 文 | while 文 | for 文 | do_while 文 | 式文 | return 文 | native 文
- if 文 := “if” “(” 式 “)” 文 [ “else” 文 ]
- while 文 := “while” “(” 式 “)” 文
- for 文 := “for” “(” 式 “;” 式 “;” 式 “)” 文
- do_while 文 := “do” 文 “while” “(” 式 “)” “;”
- return 文 := “return” [ 式 ] “;”
- native 文 := “native” **識別子** “;”
- ブロック := “{” { 区画 } “}”
- 区画 := var 定義 | 文列
- var 定義 := “var” ローカル変数宣言列 “;”
- ローカル変数宣言列 := ローカル変数宣言 { “,” ローカル変数宣言 }
- ローカル変数宣言 := **識別子** [ “=” 論理和 ]
- 式文 := 式 “;”
- 式 := 代入式 | 論理和
- 代入式 := 論理和 ( ( “=” | “+=” | “-=” | “*=” | “/=” | “%=”) 論理和 | “++” | “--” )
- 論理和 := 論理積 { “||” 論理積 }
- 論理積 := 判定式 { “&&” 判定式 }
- 判定式 := 加減式 [ ( “==” | “!=” | “>=” | “<=” | “>” | “<” ) 加減式 | “is” **識別子** ]
- 加減式 := 乗除式 { ( “+” | “-” ) 乗除式 }
- 乗除式 := 要素 { ( “*” | “/” | “%” ) 要素 }
- 要素 := [ “-” | “!” ] 素
- 素 := 素頭 { 作用 }
- 素頭 := 数 | 真偽 | **文字列** | new 素 | **グローバル** | “this” | “null” | “(” 式 “)” | 呼出し
- 作用 := 間接参照 | 添字指定
- 間接参照 := “.” 呼出し
- 呼出し := **識別子** [ “(” 引数リスト “)” ]
- 添字指定 := “[” 式 “]”
- 数 := **数字列** [ “.” **数字列** ]
- 真偽 := “true” | “false”
- new 素 := “new” **識別子** “(” 引数リスト “)”
- 引数リスト := [ 式 { “,” 式 } ]

### A.2 組み込みメソッドの一覧
- print 文字列や数値をウィンドウ（コメントボード）に出力する
- getkey キーボードのキーやマウスボタンが押されているかどうか判定する
- getMouseX マウスカーソルの x 座標を返す
- getMouseY マウスカーソルの y 座標を返す
- die オブジェクトを画面上から消す
- wait スレッドを待機させ，他のスレッドに処理を譲る
- drawString 文字列を描画する
- drawLine 直線を描画する
- drawRect 中空の長方形 (長方形枠) を描画する
- drawOval 中空の楕円を描画する
- fillRect 中身の詰まった長方形を描画する
- fillOval 中身の詰まった楕円を描画する
- setColor グラフィックスの描画色を設定する
- sin 正弦を求める
- cos 余弦を求める
- sqrt 平方根を求める
- rnd 乱数を返す
- abs 絶対値を求める
- array 配列を作成する

### A.3 アンケート設問
各設問の最後の記号は，回答形式を表す：
- **(Y) :** はい／いいえ
- **(S) :** 択一
- **(M) :** 複数選択
- **(F) :** 自由記入

**毎回の授業におけるアンケート :**
- 今日の授業の分量は？ (S)
- 今日の授業の難易度は？ (S)
- 今日の問題は自力で解きましたか (S)
- 今日の授業は楽しかったでしょうか (S)
- 自由記入欄 (F)

**能力調査（4/21）：**

- コンピュータネットワーク工学科 (CS) に入ったきっかけは何ですか (M)
- パソコンを自宅で使っていますか (Y)
- 自宅学校問わず，パソコンの利用頻度はどれくらいですか (S)
- Web ページ閲覧とメール以外にパソコンを使っていますか (Y)
- 「使っている」と答えた人は，Web ページ閲覧とメール以外にパソコンをどのように使っていますか (F)
- プログラミング経験はありますか (Y)
- プログラミング経験がある人への質問です．どんなプログラミング言語で何を作りましたか． (F)
- プログラミング経験がない人への質問です．プログラミングに興味はありますか (S)
- CS 学科の科目で，一番面白いと思った科目は何ですか (S)
- 一番興味のある分野はどれですか (S)
- 今日の授業は大変だったでしょうか (S)
- この授業は難しそうか，簡単そうか，今の段階でどう思いますか (S)
- その他，ご自由に記入してください (F)

**Nigari に関するアンケート (06/09)：**

Nigari に関するいくつかの質問にお答えください．

- 処理速度 (S)
- プログラムの読みやすさ (S)
- プログラムの書きやすさ (S)
- インターフェース (エディタウィンドウ，演習問題ウィンドウなど) の使いやすさ (S)
- 授業や演習問題を通じて自分が作ってみたプログラムの面白さ（プログラムの出来栄え） (S)
- 自由記入欄 (Nigari に関すること，今日の授業についてなど） (F)

**類推テストについて (06/16):**

- 問題 8-2 をどのように解きましたか
- 問題 9-1 をどのように解きましたか

**類推テストについて (06/23):**

- 問題 10-2 をどのように解きましたか

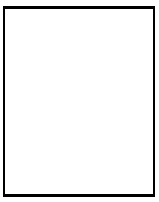
**最終アンケート（1 クラスのみ） (7/07)：**

- Nigari と比べて，Java のプログラムは読みやすいでしょうか (S)
- Nigari と比べて，Java のプログラムは書きやすいでしょうか (S)
- Nigari から Java に移行したとき，両者の違いによるギャップはありましたか (S)
- Java に移行したときに戸惑った点／つまずいた点を 3 つまで挙げてください (M)
- Nigari を使うことで Java への理解が深まったと思いますか (S)
- この授業のスタイルとして望ましいと思うものを選んでください (S)
- BBS を使いましたか（最初の自己紹介は除く） (Y)
- BBS を読んだ，書いた方は，BBS は役に立ちましたか (S)
- BBS に書き込みをしなかった方（読んだだけも含む）は，その理由は何でしょうか． (S)
- この授業の良かった点 (F)
- この授業の改善してほしい点 (F)
- その他，授業に関する意見をご自由に (F)

**最終アンケート (全クラス共通) (7/07)：**

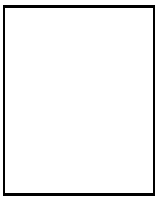
- 授業全体の分量は (S)
- 全体的な授業の難易度は (S)
- 全体的に，この授業は楽しかったでしょうか (S)
- Java のプログラムを読んだとき，何をやっているプログラムが理解することかできるようになりましたか (S)
- この授業を通じて Java を用いて望み通りのプログラムを書くできるようになりましたか． (S)
- 自分で作ってみた Java のプログラムは面白い（興味を持てる）プログラムでしたか (S)
- 各項目について理解度を答えてください
  - 変数への代入／変数を使った計算 (S)
  - 変数の型 (S)
  - if 文 (S)
  - while 文 (S)
  - for 文 (S)
  - switch 文 (S)
  - 配列 (S)
  - メソッド (S)
- この授業を通じてプログラミングに興味を持つことができるようになりましたか． (S)
- その他，自由にご意見をご記入ください．(F)

(平成 16 年 5 月 14 日受付)
(平成 16 年 5 月 14 日採録)



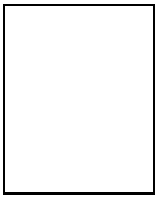
**長　　慎也**（正会員）

1976 年生．1999 年早稲田大学理工学部卒業．2001 年早稲田大学大学院理工学研究科博士前期課程修了．現在，同研究科博士後期課程在学中．情報教育及び文書の検索・分類に関する研究に従事．情報処理学会，電気情報通信学会，ACM 各会員．



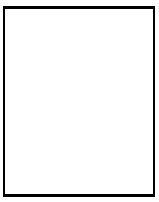
**甲斐　宗徳**（正会員）

1983 年早稲田大学理工学部電気工学科卒，1988 年早稲田大学大学院理工学研究科電気工学専攻博士後期課程修了，工学博士．1988 年成蹊大学工学部助手・講師を経て 1991 年同助教授．電気学会，電子情報通信学会，情報処理学会，IEEE 各会員．プログラミング環境，並列・分散処理システム，最適化問題の探索解法などに興味を持つ．



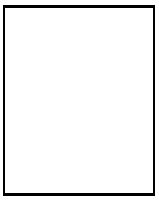
**川合　　晶**

1980 年生．2003 年早稲田大学理工学部卒業．現在，早稲田大学大学院理工学研究科博士前期課程在学中．プログラミング言語の設計，DTP などに興味を持つ．



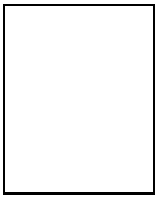
**日野　孝昭**

1979 年生．2003 年早稲田大学理工学部卒業．現在，早稲田大学大学院理工学研究科博士前期課程在学中．プログラミング言語の設計，多言語環境などに興味を持つ．



**前島　真一**

1980 年生．2003 年早稲田大学理工学部卒業．現在，早稲田大学大学院理工学研究科博士前期課程在学中．プログラミング言語の設計，教育工学に関する研究に従事．



**筧　　捷彦**（正会員）

1968 年東京大学計数工学卒，1970 年同修士課程修了．東京大学工学部助手，立教大学理学部講師・助教授を経て，1986 年から現職．日本ソフトウェア科学会，情報処理学会（フェロー），日本数学会，日本応用数理学会，ACM 各会員．ACM 日本支部副会長（ICPC Board 議長），情報処理学会情報規格調査会 SC22 専門委員会委員長．プログラミングの言語，方法，環境に興味をもつ．