プログラミング学習の高度支援環境に関する研究

Enhanced Supporting Environment for Programming Learning

2005年4月

長 慎也

プログラミング学習の高度支援環境に関する研究

Enhanced Supporting Environment for Programming Learning

2005年4月

早稲田大学大学院理工学研究科
情報科学専攻 アルゴリズム設計論研究

長 慎也

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究の目的

本論文は，情報科学を専門分野とする，大学の学部学科（以下，CS と呼ぶ）における，プログラミング教育を対象として，特にその入門教育に主眼をおいて高度な支援を行う環境を提案する．支援環境は，適切な授業方法（コースデザイン）があってはじめて機能する．本論文でも，CS でのプログラミング入門教育のコースデザインを並行して議論しながら，高度支援環境のあり方を究めていく．

プログラミング教育において最も重要なことは，プログラミングとは楽しいものである，という認識を植え付ける，すなわち，プログラミングの楽しさを教えることであると考える．

楽しさを教えることの重要性は，プログラミングに限らずあらゆるものを教える場合に言えることではあるが，プログラミング学習は，他の多くの教科と異なり，それ以前に教育がなされていないことが多く，学習者がもつ予備知識の量，質が様々である．特に，初めてプログラミングを経験する学習者（プログラミング初心者）が多い．プログラミング初心者に対して，難しさ，大変さが先行し，楽しさがなかなか伝わらない教え方をしてしまうと，プログラミングが楽しくないという印象を与えてしまい，学習者はプログラミングの学習に対する意欲がなくなり，学習をそれ以降まったくしなくなる恐れさえある．CS において，プログラミングの概念はすべての教科の基盤となるものであるから，最初の段階での挫折は避けなければならない．

しかし，現状のプログラミング学習環境には，楽しさを阻害する要因が多い．CS の授業においては，学習に用いるプログラミング言語とその言語環境には，Java などの実用的な言語環境を利用することが多いが，これらは教育目的に設計されていないため，初心者にとってはわかりにくい概念が多く，学習を挫折させてしまう恐れがある．また，教室におけるインフラ整備が進み，Web 教材などのネットワークを活用した学習環境が発達したことなどにより，学習環境は複雑になってきている．そのため，様々な要因が絡み合って，学習（特に実習）を進める上でのトラブル（困難）が多発するようになった．学習者は，その困難の原因を突き止めるのは難しく，TA などの支援を得ることになるが，人数が多い教室では十分な支援が受けられないために学習が滞り，楽しさが減退してしまうことがある．

本研究では，このような問題を，大きく分けて 2 つの手法を用いて解決する．

まず，最近では情報科学以外の学部教育および初等中等教育（以下，Non-CS と呼ぶ）でもプログラミング教育を行う試みが始まっており，そのための学習環境やコースデザインも開発されている [1][2]．これらの環境は，学習者が楽しんで，興味がもてるような工夫がなされているものが多い．これら Non-CS 向けの学習環境に見受けられるような工夫を取り入れた学習環境を設計し，CS の授業に適用する手法を提案する．この場合，Non-CS でのプログラミング教育だけでは，プログラミングの楽しさを教えることができたとしても，情報科学で必要な知識を身につけさせるには至らないという点に注意しなくてはならない．楽しさを重視しつつも，CS のカリキュラムに合致するような形で利用できる言語環境と，それを効果的に利用するためのコースデザインが必要である．

次に，学習者・教師・TA の活動履歴を，学習者の困難解決の支援に利用する手法を提案する．教室内のネットワーク設備などを利用して，大量の活動履歴が収集できるようになった．これらの履歴からは，授業の支援に有益な知識を抽出することができ，その知識を活用して，学習者の学習支援を行うことはもちろん，教師・TA の活動を支援することが可能である．

本論文では，このような手法を取り入れた学習支援環境を提案，設計し，実際の授業で適用した結果から，学習を楽しく行うことができるようになったかどうかを評価し，その有効性を示す．

1.2 プログラミング教育の状況

情報科学において，プログラミングは基本の概念であり，古くから教えられてきたものであるが，最近はオブジェクト指向などの新しい概念が増えてきたことを受けて，CS での教育も大きく変化しようとしている．

例えば，ACM と IEEE が合同して提唱している CS のカリキュラム “Computer Curricula” で，その 2001 年版カリキュラム cc2001 [3] を 1991 年版 cc1991 [4] と対比してみると，CS に要求される知識項目（body of knowledge）として，“Object-oriented Programming” や，“Event-driven programming” といった，新しい概念が追加されていることを見てとることができる．

実務で使われるプログラミング言語の勢力分布にも大きな変化が出てきている．これに呼応して，CS でのプログラミング教育に使われる言語も，C，Pascal などの非オブジェクト指向言語から，Java や C++などのオブジェクト指向へと大勢は移行し，オブジェクト指向言語を用いてプログラミングを教えることが多くなっている．実際，米国での，最近の調査報告 [5] と 10 年前のもの [6] を比較すると，新しく Java が登場し，C++の比率が上昇していることがわかる．

加えて，新しいコースデザインも採り入れられるようになった．オブジェクト指向が何であるかを知る例を先に示し，代入，制御構造といった旧来の概念を後に習う，という指針に基づくコースデザインが出てきた [7]．この指針を “Object first” と呼ぶ．さらに，この Object first の指針に従ったコースを支援するための学習環境も作られている [8]．

現状は，プログラミング環境，カリキュラム，コースデザイン，すべてが変化している最中であるといえる．CS におけるプログラミング教育に最適な言語環境とコースデザインが何かということについて常に十分な配慮をしておくことが重要である．

コンピュータが普及したことに伴って，コンピュータ利用に対する教育が広く行われるようになってきた．日本でいえば，大学では，理系・文系の枠を超えて一般情報処理が基礎教育の必須項目となっている．高校以下でも，1999 年の学習指導要領の改訂に基づいて，小中学校では 2002 年度から，高等学校では 2003 年度から，それぞれ情報教育が始められている．

こうした中で，情報科学以外，理工系以外の学部教育や，大学以前の初等中等教育（Non-CS） においても，コンピュータのそのものを理解させるのに，プログラミングを使うという試みが，それぞれに行われるようになってきた [9][10][11] .

Non-CS において利用されるプログラミング学習環境は，実用言語（教育が主目的でない，実務用に使われている言語や環境）を用いて教えるのは困難であるという判断から，教育に特化したプログラミング環境を利用して学習したり [9]，実用言語を使ったとしても，既存のプログラムを「読める」「理解できる」「少しだけ改造できる」ことに焦点をあてたり [10],「文法の細かいことを長々と説明するより，プログラム例を実際に示して，プログラミングに慣れさせる」「文法や概念に対しては具体例の中で説明する」[11] ことを重点においたりしながら教える場合が多い．

小学校から大学まで，様々なプログラミング教育の試みがなされているが，いずれにせよ，Non-CS 向けのプログラミング環境は,「プログラミングとは（あるいは，コンピュータとは）何か」を大雑把に理解してもらうことに焦点を当てており，言語仕様などの細かいことをあまり意識させないようにしている．また，グラフィックスを多用するなど，学習者に興味を持たせるためのさまざまな趣向を凝らしたものが多い．

Non-CS のプログラミング教育は，プログラミングの楽しさを教えるという面では，CS よりも先進的な手法が開発されているといえる．これらの手法を CS の教育でも取り入れていくことが重要である．

1.3 プログラミング学習支援環境に求められる性質

CS のプログラミング入門授業において用いる，学習を支援する環境必要な性質を，現状の問題点を考察しながら議論する．

- 興味をもって学習させること

  CS におけるプログラミングの入門授業は難しいと言われている．例えば，プログラミングの入門授業の落第率は，30%から 40%と，他の教科より高いという報告 [7] [12] がある．また，プログラミングに関して,「基本的な概念もまったくつかめない学習者」や「どんな手段を使ってでも，プログラミングだけは避けて通ろうする学習者」などが例年いるという [13].

  こうしたプログラミング嫌いを生まないことが，特に CS においては重要である．なぜなら情報科学におけるプログラミングの知識や技術は必須のものであり，ここで学習者が挫折し，十分な学習ができなければ，CS での “body of knowledge”[3] を構成する “Architecture and Organization”，“Operating Systems”，“Software Engineering” などの学習もおぼつかなくなるからである．

  これは CS に限ったことではなく，Non-CS においても，プログラミングに興味をもたせることは重要である．しかし，CS では，興味を「もたせる」だけでなく，その後「持続させる」ための工夫が必須である．

- 情報科学に必要な素養を身につけられること

  CS において，Non-CS において用いられているプログラミング環境を用いれば「興味を惹く」という点では役に立つかもしれない．

  しかし，Non-CS でのプログラミング教育は，あくまでプログラミングに「少しだけ」触れさせる [14] ことに焦点を当てている場合が多い．そこで用いられる学習環境もその教え方に従って作られている．プロフェッショナル教育である CS の教育では，そのような教え方でとどまってしまうのは不十分である．したがって，Non-CS のためのプログラミング環境をのそのまま利用するのは，不適当である．

  学習環境は，学習者をプログラミング嫌いにさせない配慮がなされていることが必要だが，同時に「難しさ」や「厳しさ」をもった「現実」につながるものでなければならない．いいかえれば，学習者が，学習を楽しいと感じる理由が「内容が簡単だから」であってはならない．

1.4 研究の概要と本論文の構成

本論文は，まず，初心者向きプログラミング言語 “Nigari” と，その環境である “Nigari System” を提案する．その言語は，Java をベースにしながらも，プログラムとしては必要な動作指示だけを書けばよいように簡約された仕様となっている．その動作環境は，オブジェクトを自動的に可視化し，実行結果をアニメーションとして表示する機能をもつ．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，オブジェクト指向の基礎をも理解させることができる．

次に，初心者向けのプログラミング言語・環境での学習から，実用言語での学習へと段階的に進める方式のコースデザインを定め，その課程全体を支援する学習環境“ N-Java ”を提案する．実用言語の仕様が抱える，複雑な概念構成を階層分けし，順次高い階層の概念に呼応した言語仕様を必須とする環境へと移行させていく．

最後に，学習環境が引き起こす様々な困難に対する解決を補助するための手段として，学習者の学習履歴を分析し，その分析結果を学習環境の改善に活用するための枠組みと，その枠組みを利用したシステム “proGrep” を提案する．このシステムは，学習者の書いたプログラムの内容など，プログラミング環境に対して起こした行動を，専用のサーバに逐一送信・蓄積する機能をもつ．さらに，蓄積された履歴に対して “検索” をかけることで，学習者の起こしがちな典型的な行動，特に多くの学習者が直面しがちなを困難を抽出する．システムは，これらの困難の解決に役立つアドバイスを，学習者に自動的に提示することができる．

これらの内容を，つぎの章建てに従って展開する．

まず 2 章では，言語 Nigari とその環境 Nigari System の設計方針を議論し，仕様，実装を解説する．さらに，CS におけるプログラミング教育に用いるのに適した言語と言語環境に求められる要件を整理し，既存の言語および言語環境との比較を行う．

3 章では，Nigari System を早稲田大学コンピュータ・ネットワーク工学科（CS 学科）にて適用した結果を示す．アンケートやテスト結果から，Nigari System の有効性と問題点を評価する．

4 章では，Nigari System を用いた授業から，Java の習得へつなげるようなコースデザインと，それに準じた学習環境である N-Java を提案する．また，授業での実践結果をもとに，コースデザインおよび N-Java の評価を行う．

5 章では，proGrep を提案し，その設計と実装を示す．そして，proGrep を実際の授業で利用し，収集された学習履歴と，その解析結果から，proGrep の評価を行う．

6 章で，結論を記す．

第2章　初心者向きプログラミング環境「Nigari System」の設計

2.1　はしがき

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 Nigari System を開発した．

本章では，Nigari と Nigari System をどのように設計し，実装したかについて述べる．まず，初心者に興味をもってプログラミングを学習させることができるような言語環境が備えるべき性質を挙げ，それらを同時に満たすための設計の方針を考案する．続いて，その指針に従って設計したプログラミング言語と環境の仕様を概説する．

2.2　Nigari System の設計方針

2.2.1　おまじないを意識させない言語仕様

初心者に興味をもってプログラミングの学習を進めてもらうには，提示する例題や実習する問題のプログラムが，説明をきちんと与え働きを理解し終えた言語構成素だけで書き表せるようにしておくことが大切である．その段階でのプログラミングには必要のない機能に対応するために用意されている言語構成素を用いてはじめてプログラムが書き表るようになっているとすると,「いずれ意味を解説しますが，今の時点ではとにかくこのとおり書いておかなければならない “おまじない” だと思って丸覚えしてください.」といって強制するほかはない．しかも，その “おまじない” が何文字も続く長いものであり，1 字でも間違うとそのプログラムが誤りになってしまうというのでは，初心者にプログラミング嫌いを強制しているに等しい．

実用言語でのプログラミングを教育したいからといって，最初から実用言語を使い，テキストエディタを使ってプログラムを書き，それを実用コンパイラでそのまま翻訳する，という環境を使うのは，初心者に興味をもって学習してもらうという観点からすると，最悪の選択である．少なくとも，“おまじない” に当たる文言があらかじめ自動的に提示された状況からプログラムの編集が開始できるような支援を用意しておくべきである．それでもなお，編集の途中でこの “おまじない” の部分を意図せずに書き替えてしまわないとは限らない．そうなると，実用コンパイラはその “おまじない” がないことに起因する多くの誤り表示を行ってくる．初心者はまったく意識していないことがらであるから，戸惑うばかりでなんら得るところがない．

同じ支援を行うのなら，その “おまじない” そのものがいらない場を設け，その場に限定された範囲でプログラムの翻訳や検査が行われるように徹底するのが望ましい．まず，どこまでの範囲の言語機能を学習させようとするのかを明らかにし，それらの言語機能だけでプログラムが書き表せるように初心者教育用のプログラミング言語を設計する．このとき，プログラミング言語を独立に設計するのではなく，その言語でのプログラミングを支援する環境と一体として設計していくことが何より重要である．

2.2.2 実用言語に則した基本構文

ここでは，CS でのプログラミング教育をつぎの順序で進めるとして議論を進める．

1. 手続き型プログラミングの基本を学ばせる
2. その上でオブジェクト指向の基本を学ばせる

これは，従来からのプログラミング教育にオブジェクト指向に関する教育を上乗せした形であり，プログラミングの歴史的発展の順序に沿ってもいるので，教育するのに採用しやすい順序である．これに対して，オブジェクト指向を学ばせるには，まずオブジェクト指向の基本から教え始めるべきだ（“Object first”）とする試みが行われている [7][15]．Object first は最近になって提唱され始めたもので，確固たる定義がまだなく [16]，概念が複雑であるため学生の落第率が上昇する，といった問題点が指摘されている [7]．どちらの方式が有効であるかについては，まだ明確な結論が出てはいない．ここでは，歴史的順序に従って，手続き型プログラミングの基本を終えてからオブジェクト指向の基本に進む（“Object late”）という方式での教育を対象とする．

手続き型プログラミングの基本といえば，つぎのものを扱う必要があるのはいうまでもない．

- 変数，代入
- 順次処理
- 条件分岐
- 反復

CS でのプログラミング教育では，その教育が実用言語に自然につながっていくことも考慮しておくのがよい．幸い，手続き型言語でいえば，上に挙げた言語機能に対応する言語構成素は，C, C++, Java の間でほとんど差がない．したがって，構文としては，これらの言語の構文に則したものを採用することにする．

また，“おまじない” を書かせることなくプログラムを記述できることが重要である．上に挙げた言語構成素だけをプログラムに書かせるようにし，それ以外のメソッドやクラスの宣言といった言語構成素は書かない，という言語仕様にする．

2.2.3 プログラムの可視化

単純変数に対する値の代入，という手続き型プログラミングの基本中の基本ともいえる概念は，素直に理解し受け入れることができるものではない．一つには，高等学校までの数学教育の中で学んで

```
x=0; y=0;
while( x<300 ){
  x=x+1;
  y=y+1;
}
```

図 2.1: プログラム例

```
while( x<300 ){
  x=x+1;
  y=y+1;
}
```

図 2.2: プログラム例（初期設定なし）

きた，数学での “変数” と同じ名称でありながら，まったく違った概念であることが理解のしにくさを誘っているのだろうと思われる．

プログラミングでの変数の概念を学ばせるには，プログラムの実行に従って変数の値が変化していく，という状況を目の当たりに提示して見せるのが一番である．そこで，変数の値がプログラムの実行に従って変化していくことを，否が応でも目にすることになる仕組みを用意する．さらには，それを目にすること自体をプログラミングの課題対象とすることができるようにしたい．それには，画面上で “もの” が動くようにする，すなわち，アニメーションが起こるようにしておくのが一番である．

アニメーションが特に意図するまでもなく自動的に表示され，かつ，それ自身が変数の値の変化そのものを理解するのに役に立つようにするにはどうすればいいか．最も簡便な方式は，特定の絵柄（たとえば円，正方形などの図形）を，プログラム中の特定の 2 つの変数の値を座標とする位置に表示することである．高校までの数学での教育では，(x, y) 座標という表現が普通に採用されているから，これに合わせてプログラム中の変数 `x` と `y` との値を座標とすることにすれば，ことさらに説明するまでもなく理解してもらえる．例えば，図 2.1 のプログラムを書くだけで，絵柄が画面上を斜めに動くようになる．これを通じて，プログラムの実行にともなって変数の値が書き変わっていくことを如実に目にすることができる．

2.2.4 基礎的なオブジェクト指向

この研究では，“Object late” の方針でのプログラミング教育を考え，そこでの手続き型プログラミングの基本教育の支援を対象としている．手続型プログラミングを直接の支援対象とはしているものの，次に続くオブジェクト指向のプログラミングに自然につながっていく配慮も欠かすことはできない．

図 2.1 で示したプログラムによって絵柄が斜めに画面上を斜めに移動する．それなら，初期値設定の部分を除いた図 2.2 のプログラムが “斜めに移動する” 動作を表していると認識するのは，初心者にとってもたやすいことである．

さらに進んで，この動作をする “もの”（絵柄），すなわち “オブジェクト” を，1 個だけでなく 2 個，3 個と画面上に配置して動かしてみたい，と学習者が思っても不思議ではない．

そこで，このプログラムを収めたファイルを用意し，そのファイルを対象にウィンドウ上でのマウス操作によって，このプログラムに従った動作をする “オブジェクト” をいくつでも “生成” し “配置” できるようにしておく．画面上で配置することで，その “オブジェクト” の “インスタンス変数” である x と y の値が自動的に設定されるようにしておくのである．逆に，すでに配置されたオブジェクトが，どのような振舞いをするのかを確認したい時もある．画面上のオブジェクトを，ポインティングデバイス等を用いて選択することにより，その振舞いを表すプログラムをいつでも呼び出せるようにしておけば，オブジェクトとそのプログラムの関連を強く意識できるようになる．

学習者は，こうした操作を行いその結果を観測することを通じて，ごく自然に，動作内容を規定するプログラム（クラス）とその動作を行うオブジェクトという，オブジェクト指向の基本的な概念構成を体得することができる．ついでながらに，同じクラスのオブジェクトでも，画面上での絵柄を違えることができるようにしておくとより便利である．そこで，インスタンス変数として x, y に加えて p を設け，画面表示する絵柄の番号を値として保持するようにしておく．絵柄の選択は，オブジェクトの生成操作をする際に，用意された複数の絵柄の中から選択することができるようにしておく．

言語環境は，プログラムを実行する前の状態（設計時）にあるときに，利用者（学習者）がオブジェクトを配置したりその初期値を与えたりした最新の変更結果をすべて記録しておく．実行開始の指示が利用者から発行された場合は，その記録された初期値をそれぞれのオブジェクトのインスタンス変数に与えてから，実行を開始する．

2.3 プログラミング環境 Nigari System

Nigari System は，画面上にオブジェクトを配置し，そのオブジェクトがもつ変数の初期値を設定し，配置されたオブジェクトの動作を言語 Nigari で記述することにより，プログラムを作成する．これらの一連の作業を “設計” と呼び，設計を行っている状態を “設計時” と呼ぶ．この設計段階で配置されたオブジェクトを “設計時オブジェクト”，プログラムが実行されている時（“実行時”）に動的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．

Nigari System における “プログラム” は，クラスの定義を記述する “ソースファイル” および，設計時オブジェクトの，オブジェクト変数（2.4.8 参照）の初期値から構成される．1 つのプログラムには，設計時オブジェクトとソースファイルをともに複数個含めることができる．各オブジェクトは必ずただ 1 個のクラスに属するが，複数のオブジェクトが，同一のクラスに属することがある．

2.4.2 で後述するように，ソースファイルとクラスの定義は 1 対 1 に対応するため，各オブジェクトは 1 個のソースファイルに対応づけることができる．ユーザは，画面上に配置されたオブジェクトを選択することによって，そのオブジェクトに対応づけられたソースファイルをいつでも呼び出すことができる．

Nigari System がプログラムの実行を開始すると，まず，プログラムに含まれるすべての設計時オブジェクトが作成され，画面上に配置される．次に，それらのオブジェクト変数の初期値が設定される．そして，各オブジェクトに 1 個ずつスレッドが割り当てられ，各スレッドは，そのオブジェクトが属しているクラスのメイン文列（2.4.4 参照）をそれぞれ並行に実行する．

実行中の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の値を用いて行う（2.5.3 参照）．

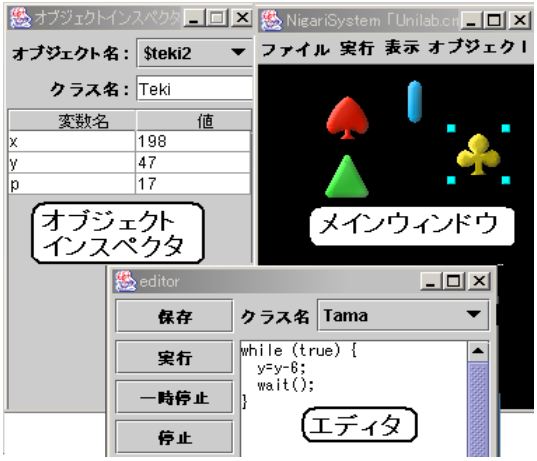


図 2.3: Nigari System の動作画面

図 2.3 に，Nigari System の動作画面を示す．それぞれの役割を示す．

- **メインウィンドウ**

  設計時には，設計時オブジェクトを配置する．実行時には，オブジェクトの状態に応じてオブジェクトに対応する絵柄を随時表示する．

- **エディタ**

  ソースファイルを編集する．

- **オブジェクトインスペクタ**

  設計時は，設計時オブジェクトのオブジェクト変数の初期値を設定する．実行時は，オブジェクトのその瞬間のオブジェクト変数をリアルタイムに表示する．

2.4 Nigari 言語仕様

2.4.1 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，代入式の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．

2.4.2 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構文上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．

2.4.3 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（付録 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

2.4.4 メイン文列

プログラム本体に直接[1] 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（2.5.2. 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.

2.4.5 文

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

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 [17] を参照されたい.

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，2.4.1 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. 2.7.2 参照）.

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 文の式や，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.

2.4.6 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**（2.4.8 参照）を並べる. var 定義は文に先だって並べる必要がある.

[1] 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

2.4.7　コンストラクタの定義

コンストラクタは，2.4.9 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．

2.4.8　変数

2.4.1 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（2.5.4 参照）．

- **ローカル変数**

  ローカル変数は，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**ブロック**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．

2.4.9　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．

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その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．

オブジェクトの動的生成

2.3 で示したように，オブジェクトには，設計時に（間接的に生成して）配置する設計時オブジェクトと，実行時に生成される実行時オブジェクトがある．

実行時オブジェクトの生成には,「“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（2.4.7 参照）を呼び出し，さらにそのオブジェクトをこの式の値とする．

実行時オブジェクトにも，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，並行にそのクラスのメイン文列を実行する．

2.4.10 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（2.4.11 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の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，まったくの別物である．

2.4.11 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[2] 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付録 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 2.5.2 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.

2.5 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.
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

2.5.1 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.

[2] 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.

2.5.2 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 [17] 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（2.4.9 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（2.5.3 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. スレッドの処理の実行」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 2.4 のようになる．

2.5.3 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，その値に基づいて，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．絵柄は環境に予め用意されており，それぞれの絵柄に番号が割り当てられている．

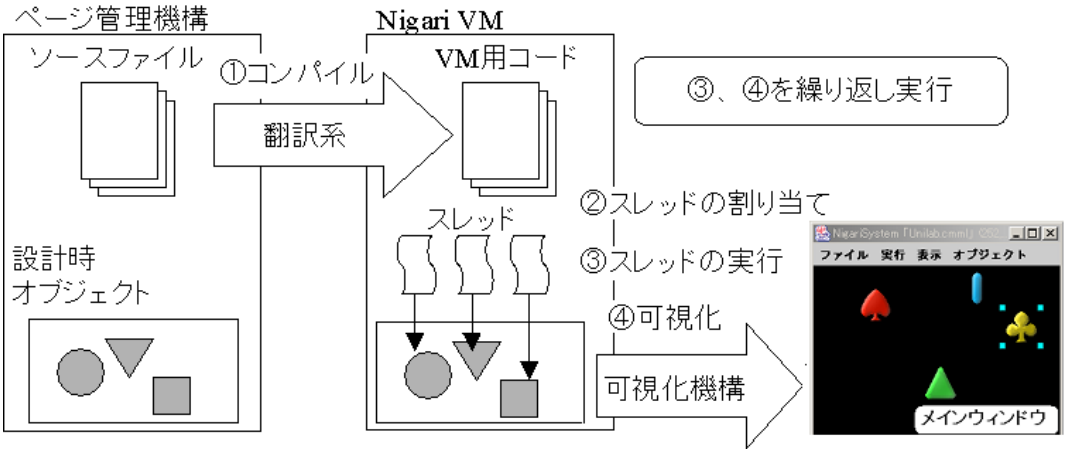


図 2.4: 翻訳系，実行系，可視化機構の動作

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（2.4.8）をもつ．

2.5.4 ページ管理機構

“ページ” とは，Nigari System のプログラム（ソースファイルと設計時オブジェクトの初期状態の集まり）のことである[3]．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 [17] 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）

[3] Nigari System のプログラムのことを，ユーザには “ページ” という名称で説明している．これは，“ソースファイル” と “プログラム” は混同しやすいことと，Nigari System のプログラムを読み込むと，設計時オブジェクトの初期配置が画面上に表示され，それがあたかも 1 枚分の画面を読み込んだように見てとれるため，一般的な「ページ」の概念によくあてはまることによる．

- 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．

2.5.5 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．

2.6 プログラム例

Nigari System によるプログラムの例を示す．なお，各プログラム例の図中には，ソースファイルだけを掲載することにし，原則として，そのソースファイルで定義されるクラスのオブジェクトが，設計時オブジェクトとして少なくとも 1 個は配置されているようなプログラムを想定している．

2.6.1 変数の値によるアニメーション

図 2.5 に示したソースファイルで定義されるクラスのオブジェクトは，オブジェクト変数 `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，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．

2.6.2 マウス入力と if 文

図 2.6 に示したソースファイルで定義されるクラスのオブジェクトは，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）．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．

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

図 2.5: Nigari のサンプルプログラム（1）

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

図 2.6: Nigari のサンプルプログラム（2）

2.6.3 他のオブジェクトの参照

図 2.7 に示したソースファイルで定義されるクラスのオブジェクトは，別のオブジェクト$playerを追跡する．

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（2.5.4 参照）．

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

2.6.4 メソッドの定義

図 2.8 に示したソースファイルで定義されるクラスのオブジェクトは，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．

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

図 2.7: Nigari のサンプルプログラム（3）

```
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 2.8: Nigari のサンプルプログラム（4）

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

図 2.9: Nigari のサンプルプログラム（5）

2.6.5 オブジェクトの動的生成

図 2.9 に示したソースファイルで定義されるクラスのオブジェクトは，マウスボタンが押されるたびに，新しく Tama 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．

2.7 言語仕様の考察

2.7.1 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．

Nigari では，wait メソッドを呼び出すことでスレッドの処理を他のスレッドに移す働きがある．す

べてのスレッドは，VM によって完全に制御が行われているため，wait メソッドの呼出しがない限り，他のスレッドに処理を移さないようにすることが可能である．よって，wait メソッドを呼ぶタイミングに注意しながら，Nigari のプログラムを書くことで，排他制御をすることは可能である．

しかし，2.5.2 で述べたように，現行の仕様においては，wait メソッドが呼ばれなくとも，ある一定数のコードを実行するだけでも他のスレッドに処理が移るようにしてある．この機能を敢えてつけた理由には，学習者が wait メソッドを書き忘れた場合に，他のスレッドの処理が行われなくなることが学習者の混乱を招くことと，初心者教育でのマルチスレッドの題材が,「複数のオブジェクトが同時に動くアニメーション」程度のものであり，厳密な排他制御を必要としないものである，ということが挙げられる．

Nigari で排他制御を扱いたい場合は，wait メソッドの呼出しがない限り他のスレッドに処理を移さないように，Nigari System の設定を変えられるようにしておけばよい．

2.7.2 真偽値の扱い

Nigari は，C 言語などと同様，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．これは，実際の授業において，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）を行うようにすれば問題はほとんどないと考えられる．

2.7.3 変数宣言と変数の型

Nigari では，プログラムがすぐに動作することを第一に考え，変数宣言が必要でなく，変数にあらゆる型の値が格納できるようにした．しかし，変数宣言が必要でないと，変数名の書き間違えや，型の誤りに起因して実行時に予期せぬ動作をした場合に，その原因をつきとめるのが困難である．

ただし，プログラムが小規模であるうちは，誤りの発見はそれほど困難でないと考えられる．実際，Nigari System を用いたプログラミングの演習では，あまり大きなプログラムの作成を想定していない．例えば，3 章で実践した授業で用いた教材 [18] も，多くのプログラムは 10 行未満に収まっている．

もちろん，変数の宣言，変数の型は重要な概念であるが，最初からそれを意識してプログラム作る必要は必ずしもない．最初の段階で教えるべきことは，変数という値を入れるための仕組みがある，ということであり，変数宣言，変数の型は，Java などの他の言語に移行してから習得すればよい．

2.7.4 メソッドの書き方

Nigari のメソッドの書き方を Java と比較すると，次のような点で異なっている．

- Java のメソッドの宣言の先頭部分は，戻り値の型が記述されるが，Nigari では，先頭部分は “`function`” である．
- Java のメソッドの宣言における仮引数列は，型と仮引数名の対を並べたものであるが，Nigari では型を記述せず，単に仮引数名を並べる．

これらの違いは，変数に型がないことから，メソッドの引数，戻り値にも特定の型を期待することができないために発生した違いである．

これらの違いによって，Nigari を習った学習者が，Java に移行したときに混乱を招く恐れがあるが，Nigari で学習している段階では，オブジェクト間でメッセージを送受信するための仕組みとして，メソッドというものがある，程度の認識を持たせるにとどめておき，メソッドが必要になるくらいの複雑さをもったプログラムを書くような状況になったら，Java に移行する，というコースデザインにしておけばよい．

2.8 既存のプログラミング環境との比較

既存の言語環境と，Nigari System の評価を行い，CS におけるプログラミング教育に適した言語環境について考察する．

2.8.1 調査対象と調査項目

既存のプログラミング言語環境として，次のものを取り上げる．括弧書きしたものは，その環境でのプログラミングに用いる言語である．

- JDK (Java)
- 若葉 （若葉）
- Squeak (Squeak)
- ドリトル（ドリトル）
- DrJava (Java)
- BlueJ (Java)
- Beanshell (Java)
- Nigari System(Nigari)

それぞれの言語環境について，特に次の点に重きを置いて見ていく．

1. 言語の簡単さ

   その言語環境で用いるプログラミング言語について，どれほどの複雑さをもったものになっているかどうかを見る．特に，最初の段階でのプログラミング例とするプログラムを書いたり読んだりするのに，その複雑さ故に難しくなることがないかどうかについて吟味する．

2. プログラムの可視化

   代入や制御構造などの振舞いを，画面上にグラフィックスやアニメーションを利用して可視化することにより，プログラムの理解を助けたり，興味を惹く題材を簡単に作成したりすることができる．

   プログラムの振舞いを可視化するのに，その言語環境がどれほどの知識を必要とするかどうかを吟味する．より少ない準備で済む方が学習者を引率するのに適している．

```
class Test {
  public static void main(String []args) {
    System.out.println(``Hello, World'');
  }
}
```

図 2.10: Java の Hello, World

3. オブジェクト指向（OO）の学習

   オブジェクト指向のもっとも基本の概念は，まず “オブジェクト” が存在すること，そしてオブジェクトの振舞いを規定する “クラス” が存在することである．

   その言語環境を利用することで,「プログラム全体をオブジェクトが動かしている」「その個々のオブジェクトの振舞いは，クラスによって定義される」という概念を，学習者に積極的に意識させることができるかどうかを評価する．

4. 実用言語との親和性

   入門時に習う言語といえど，何らかの実用言語に似た形式をもたせておいたほうが，CS を学ぶ上では後の学習に役に立つ場合が多い．その言語が，実用言語と共通する性質をどれだけもっているかどうかを評価する．（もちろん，入門の段階から実用言語を題材とする場合は，親和性についてはほとんど議論の余地はない．）

2.8.2 言語環境の評価

JDK

オブジェクト指向を教えることが多い中，代表的なオブジェクト指向言語である Java を，授業の題材として用いることが多くなってきている．ここでは，Java およびその開発実行環境である JDK を用いた場合のプログラミング授業に関して考察する．

まず，言語の簡単さという観点から考察すると，Java にはつぎのような初心者にとっての “おまじない” が多く，あまり簡単な言語ではないと考えられる．

- クラスとメソッドの宣言

  いわゆる “Hello, World” プログラム（図 2.10 参照）のような，最も単純なプログラムを書く場合でも，クラスの宣言と main メソッドの宣言を書くことが強制され，しかも main メソッドは `static` や `String[]` など，普通は最初には習わない概念がたくさんあり，それらを説明することが困難である，という指摘がされている [19][20].

- 複雑なライブラリ（特に I/O と GUI）

  JDK が用意しているライブラリを用いて，標準入力から文字列を入力するためのプログラムを図 2.11 に示す．このように，単純な文字入力を行わせるにも，たくさんのクラスと例外処理を用いる必要がある．このような標準入力関連のライブラリや例外処理の煩雑さは，Java の「弱点」として指摘されている [21].

```
class Test2 {
  public static void main(String []args) {
    try {
      InputStreamReader str=new InputStreamReader(System.in);
      BufferedReader br=new BufferedReader(str);
      String line;
      while((line=br.readLine())!=null) {
        System.out.println(line);
      }
    } catch (IOException e) {}
  }
}
```

図 2.11: Java における，標準入力からの読み込み

また，プログラムの振舞いを適切にグラフィックスとして可視化させるには，JDK でグラフィックスを扱うために必要な AWT ライブラリを，プログラムから明示的に呼び出す必要がある．しかし，AWT は「不必要に複雑な概念」を学習しなければ使えるようにならない，という指摘がある [22] ように，JDK でグラフィックスを初心者に使わせるのは難しい．このため，プログラムの振舞いを可視化することも容易ではない．

また，Java はオブジェクト指向言語であるにもかかわらず，オブジェクト指向に向かないという指摘もある．特に，Java アプリケーションの開始時に最初に呼ばれるメソッドである main メソッドが static なメソッドであるのは，オブジェクト指向を習う場合の障害であるという指摘がされている [19][23] .

若葉

若葉 [24] は，大学におけるプログラミング教育用に開発された言語環境である．

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，言語仕様を簡素なものにしてあり，言語の簡単さと，実用言語との親和性は考慮されているといえる．

しかし，これは非オブジェクト指向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．

また，作成されるアプリケーションもコンソールアプリケーションが主体であり，興味を惹かれる成果物はあまり作れない．プログラムの振舞いも，コンソールに明示的な出力しないかぎり確認することができないし，グラフィックスでの出力はできない．

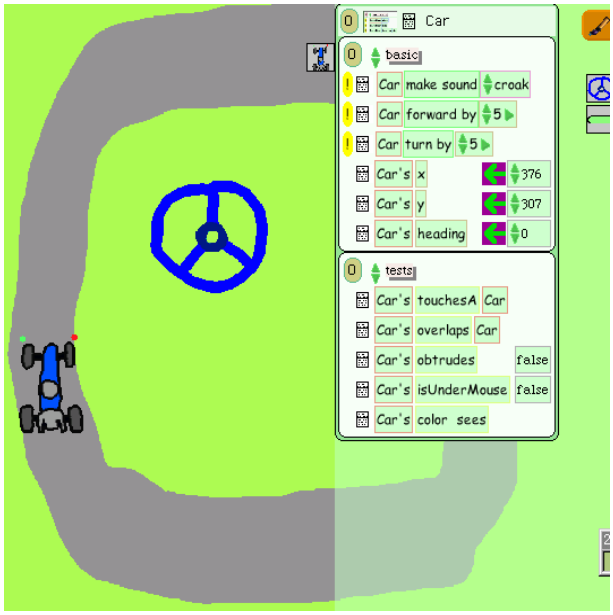


図 2.12: Squeak（全体画面）

Squeak

Squeak[25] は，Smalltalk の環境の 1 つで，プログラミングの機能をもち，学習者が自由に絵を描き，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．動作画面を図 2.12 に示す．

実際に絵をアニメーションとして動かすことができるため，プログラムの振舞いが可視化されているうえに，学習者は興味をもってプログラミングをすることが可能である．

また，学習者の描いた絵は “オブジェクト” として扱われ，そのオブジェクトがもつ変数やそのオブジェクトに対して呼び出せるメソッドの一覧が表示される．これらの部品をマウス操作で組み合わせることでプログラムの作成を行う（図 2.13）．このように，プログラミング方法は簡単で，しかもオブジェクト指向を強く意識したものになっている．

ただし，Squeak は，特定のオブジェクトに対して直接にプログラムを付加することで動作を記述するため，クラスの概念をあまり意識することができない，あるいは敢えて意識させないようにしている．

また，そのプログラミング環境自体が，マウス操作が主体であることは，実用言語のプログラミング方法とは大きく異なる．さらに，Squeak でのオブジェクトの動作はタイマー（一定時間ごとに特定の処理を行わせる仕組み）が主体になっており，繰り返し構造を使わないでも，擬似的に繰返しの概念が書けてしまう．このような性質から，実用言語との親和性はあまり期待できない．

ドリトル

ドリトル [26] 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て作られた，LOGO をベースにした言語環境である．動作画

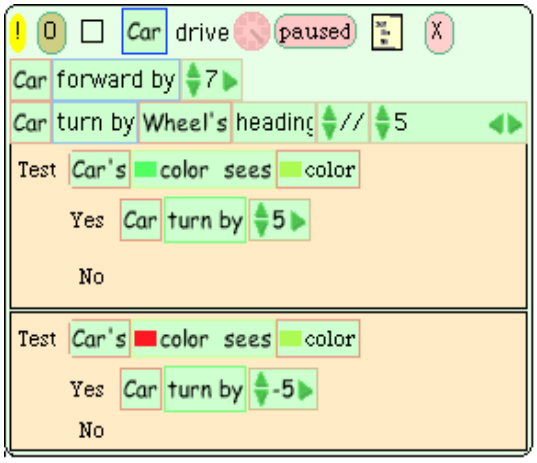


図 2.13: Squeak（スクリプト作成ウィンドウ）

面を図 2.14 に示す．この画面を描くためのプログラムを図 2.15 に示す．

実行結果は主にタートルグラフィックスの形で可視化される．他にも GUI 部品を操作したり，他の学習者とオブジェクトを共有したりする機能があり，学習者は興味のもてるプログラムを作れるように工夫してある．

プログラムの記述はテキストベースであり，メソッドやクラスの宣言をすることなく，命令だけを書けばプログラムが実行できる．命令やオブジェクトの名前はほとんどは日本語で記述するようになっており，文字種の細かい違い（全角，半角等）を吸収し，些細な間違いを最小限にとどめるようにしてあるなど，言語の簡単さへの配慮が払われている．

また，オブジェクト指向の習得を考慮しており,「タートル」などのオブジェクトに「歩く」「右回り」などのメッセージ送信を行う，という方法でプログラムを動かす．また，それらの振舞いが画面上に自動的に反映されるため，プログラムの可視化が容易である．

しかし，ドリトルは，プロトタイプベースのオブジェクト指向言語であるため，クラスの概念がない．オブジェクトとクラスの役割を理解させるように作られていない．

また，Squeak 同様，繰返しの処理にはタイマーを使うことが多い．Java の while 文に相当する繰返しの制御構造も備えてはいるが，学習者が実際に書いたプログラム [1] の中では，ほとんど使われていない．他にも，文法や命令が日本語を意識して作られていることもあり，実用言語との親和性はあまりないといえる．

そもそも，ドリトル，Squeak などは Non-CS におけるプログラミングの教材として設計されているため，CS 用の教材として直接利用するには適さない．

DrJava

DrJava[27] は，CS でのプログラミング学習での利用を想定した，Java の学習支援環境である．動作画面を図 2.16 に示す．DrJava には，対話実行機能があり，オブジェクトの生成や，生成したオブジェクトに対するメソッド呼出しなどを手軽に行うことができる．

この対話実行機能は，オブジェクト指向の学習を支援する効果の他に，main メソッド（2.8.2 参照）

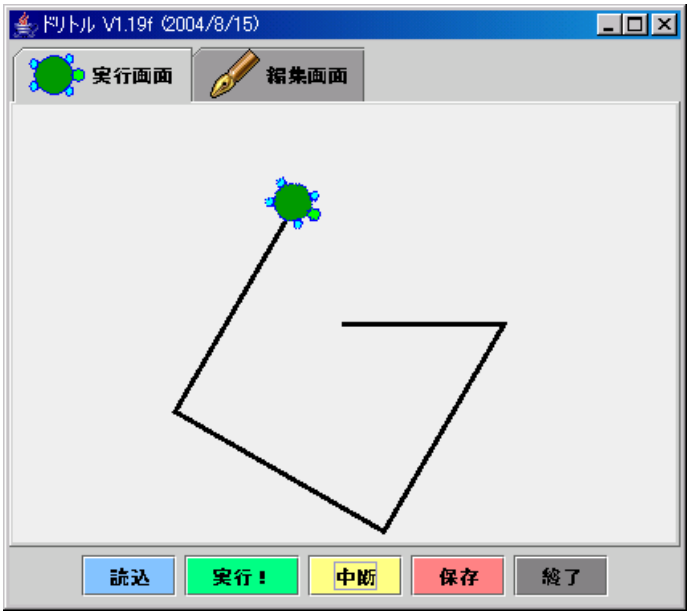


図 2.14: ドリトル

という “おまじない” を教えないで済む，といった効果もある [23].

しかし，DrJava にはグラフィックスを表示するための特別な機能が用意されていないため，可視化の困難さは JDK と何ら変わるところはない．オブジェクトやクラスの存在を，はっきり意識させるのも難しい．

また，対話実行機能に入力するプログラム以外は，標準的な Java のプログラムを書く必要がある．ただし，DrJava は，Object first に従ったコースデザインに基づいた授業で使うことを前提としているため，クラスやメソッドの宣言に関して初期の段階から説明を行う．このため，それらを “おまじない” である，という教え方をする必要はない．Object first の方針そのものが，言語の難しさを改善しているともいえるが，この指針に従った環境は，本研究が対象としている Object late に基づくコースデザインに直接使うのにはあまり適さない．

BlueJ

BlueJ[8] は，DrJava 同様，CS でのプログラミング学習での利用を想定した，Java の学習支援環境である．動作画面を図 2.17 に示す．

クラスや，オブジェクトなどが画面上に図形として表示され，それらに対して「オブジェクトの生成」や「メソッドの呼出し」などの命令を，マウス操作で対話的に発行することができる．オブジェクトとクラスの情報が画面上に可視化されるため，プログラムの振舞いや，オブジェクトとクラスの関係を掴むのに適している．

しかし，オブジェクトの情報が表示されるとはいえ，呼び出したメソッドが動作している間に何が起きているかは，明示的に処理を書かない限り表示されない．つまり，あるメソッドが全体としてどのような振舞いをしているかは理解できても，その内部で使われている代入，制御構造などの詳細な

```
カメ太＝タートル！作る。
カメ太！１００歩 歩く。
カメ太！１２０度 右回り。
カメ太！１５０歩 歩く。
カメ太！９０度 右回り。
カメ太！１５０歩 歩く。
カメ太！９０度 右回り。
カメ太！１５０歩 歩く。
```

図 2.15: ドリトルのプログラム例

振舞いを把握する用途にはあまり適さない.

また，プログラム自体は標準的な Java で書く必要がある．Bluej は DrJava 同様，Object first の方針で教えるための学習環境であるため，プログラムを書く難しさは DrJava と同等であるといえる.

Beanshell

Beanshell[28] は，Java のインタプリタ環境で，DrJava 同様，Java の文を対話的に実行する機能をもつ．Beanshell のプログラムは，Java 同様クラスやメソッドの宣言を書くこともできるが，これらの宣言を省略して文を直接書くこともできる．また，変数宣言を省略することもできる．こうした言語仕様により，プログラムを簡単に書くことができる.

しかし，DrJava 同様，グラフィックスを表示するには，AWT など，JDK で用いられている標準的なライブラリが必要となるため，可視化を行ったり，オブジェクトやクラスの存在を意識させたりするのが難しい.

Nigari System

本章で取り上げた Nigari System は，手続きや制御構造の構文を Java，C，C++などの実用言語に近づけ，実用言語との親和性を重視している.

実用言語との親和性を考慮しつつも，クラスやメソッドなどの,（Object late の方針では）最初に “おまじない” としてしか教えられない部分は書く必要がなく，プログラムの記述が簡単である.

実行環境は，オブジェクトのもつ変数の値を利用して，オブジェクトやプログラムの振舞いを画面上に可視化する機能をもつ．学習者は，可視化のための特別なプログラムを書く必要がない.

オブジェクトの動作は，そのオブジェクトに関連づけられたソースファイルによって記述される．異なるオブジェクトが同じソースファイルに関連づけることも可能である．また，オブジェクトを画面上で選択するだけで，関連づけられたソースファイルをいつでも確認することができる．さらに，Nigari において，ソースファイルとはクラスに他ならない．これは，オブジェクトとその振舞いを表すクラスの関係を認識させるのに適している．また，オブジェクトは実行中に常に可視化されているため，プログラムの実行にオブジェクトが深く関与していることを印象づけることができる．これらの特徴は，オブジェクト指向の基本概念を学習するのに適しているといえる.

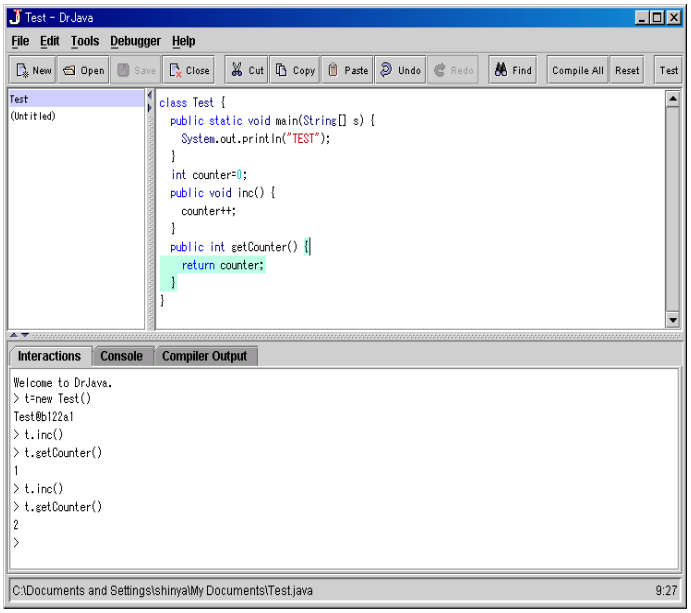


図 2.16: DrJava

2.8.3 言語環境の比較のまとめ

本章で取り上げた言語環境と要件との対応を表 2.1 に示す.

1. 言語の簡単さの面でいうと，JDK で用いられる言語 Java は，初心者にとって難しい言語である．若葉，Squeak，ドリトル，Beanshell，Nigari System は，その言語仕様をなるべく簡単にするための配慮がなされた環境であるといえる．DrJava，BlueJ は，プログラムそのものを Java で書く必要があるため，積極的に言語仕様を簡単にしようとはしていないが，Object first の方針に基づいたコースデザインを採用し，Java の難しさを緩和する工夫がなされている．

2. プログラムの振舞いの可視化に対する支援の面でいうと，Squeak，ドリトル，BlueJ，Nigari System は，特別なグラフィックス命令を使用せずとも，何らかの意味で可視化を行う機能（可視化機能）をもつ．ただし，BlueJ は，アニメーションを行う機能をもたないため，繰返しなどの制御構造の振舞いを理解させるのには向かない．JDK，若葉，DrJava，Beanshell は可視化機能が用意されていない．

3. オブジェクト指向（OO）の基本を学習する，という面では，まず，そもそもオブジェクト指向言語でない若葉は学習に適さない．さらに，オブジェクト指向言語であっても，可視化機能のない JDK，DrJava，Beanshell は，オブジェクトの存在をはっきりと意識させるのが難しい．また，Squeak やドリトルは，可視化機能があっても，オブジェクトとクラスをはっきりと区別させるようには設計されていない．BlueJ，Nigari System は，オブジェクトとクラスの概念を意識した可視化が行われている．

4. 実用言語との親和性の面では，そもそも実用言語を採用している JDK，DrJava，BlueJ，Beanshell は言うまでもない．若葉，Nigari System は実用言語の文法を借りてくることで親和性を

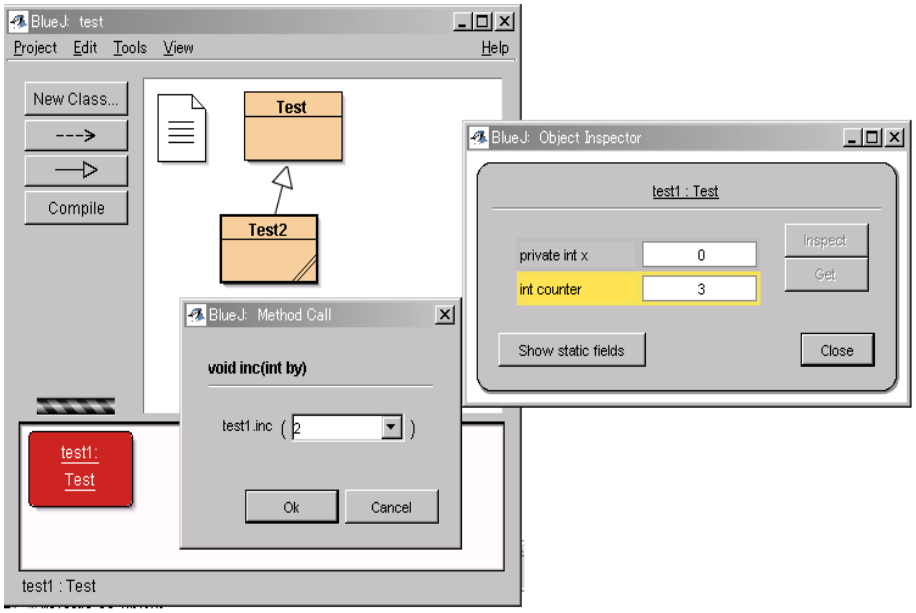


図 2.17: BlueJ

もたせている．Squeak，ドリトルは，Non-CS，特に初等中等教育を対象としているため，親和性をもたせようとはしていない．

これらのことから，1 から 4 として掲げたすべての点で，高い評点を得るような言語環境は Nigari System だけであることを確認した．

2.9 まとめ

本章では，CS におけるプログラミング入門授業に適した言語 Nigari とその環境 Nigari System を提案した．

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ため，簡単に記述することができる．一方，基本的な制御構造などは，Java とほとんど同じ仕様になっており，実用言語としての Java を理解するのに適している．

また，Nigari の実行環境は，オブジェクトを自動的に可視化し，プログラムの動作に従ってアニメーションを表示する機能をもつ．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，制御構造の振舞いや，オブジェクトの概念をも理解させることができる．

このような「簡単に記述できる」「プログラムが可視化される」「オブジェクト指向の学習に役立つ」「実用言語との親和性が高い」といった性質をすべて満たすような言語環境は他に存在しないことを確認した．

次章では，Nigari System を用いた授業を実践した結果から，Nigari System の有効性を評価する．

表 2.1: 言語環境の比較

	1. 簡単さ	2. 可視化	3. OO	4. 親和性
JDK	×	×	△	○
若葉	○	×	×	○
Squeak	○	○	△	×
ドリトル	○	○	△	×
DrJava	△	×	△	○
BlueJ	△	△	○	○
Beanshell	○	×	△	○
Nigari System	○	○	○	○

第3章　Nigari Systemの評価

3.1　はしがき

本章では，2 章で提案した言語 Nigari およびその環境 Nigari System の有効性を，実際の大学の授業で使用した結果を用いて確認する．

早稲田大学コンピュータ・ネットワーク工学科（CS 学科）1 年前期のプログラミングの授業の一部に適用した．この授業は本来 Java を用いた実習を伴う形に設計されている．並行して行われる 3 クラスの中の 1 つにおいて，その導入部の実習に Nigari System を用いた．

アンケート結果と試験の成績から，Nigari System は学習者の興味を惹き，学習意欲を高めることができるか，また，CS の必要なプログラミングの基礎知識を身につけることができるか，という点について評価する．

3.2　授業の進め方

3.2.1　授業の概要

- **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）」[29]
- **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．

表 3.1: 授業内容（第 1 クラス）

日付	内容
4/14	講義概要
4/21	PC の使い方とプログラミング
4/28	Nigari System 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 （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

3.2.2 学習環境

3 つのクラスのうち，第 1 クラスは Nigari System を利用し，第 2，第 3 クラスでは利用しなかった．第 1 クラスの授業内容を表 3.1 に示す．今後，特に断りがない限り，第 1 クラスの授業についてのみ述べる．

第 1 クラスは，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．

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 3.1 に動作画面を示す．

3.2.3 授業形態

1 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 人で，TA は学習者の挙手に応じて個別に対応を行った．

3.2.4 教本

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．

Nigari System 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 System を使った授業の実習において出題した演習問題の一部および，5 月 19 日の「if 文」の項目で使用したテキ

図 3.1: JavaEditor の動作画面

ストの一部を付録 A.3 に示す．なお，全授業のテキストは Web[18] にて参照されたい．

3.3 評価方法

授業の実施結果から，Nigari System の評価を行う．評価は，授業を楽しく受けることができたか，CS に必要なプログラミングの基礎を習得できたか，の 2 点について行う．

3.3.1 授業の楽しさ

第 1 クラスでは，毎回の授業の出席点呼の代わりにアンケートを実施した．4 月 28-6 月 30 日の間，「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．

この中の「授業が楽しさ」の回答結果から評価を行う．ただし，授業の楽しさが，単に授業が簡単だったから，という理由であってはならない．また，授業が難しいと感じたにもかかわらず，楽しいとも感じている場合,「課題を達成したときの喜びが大きい」[1] と考えることができるため,「授業の難しさ」もあわせて評価の対象にする．

3.3.2 プログラミングの基礎の習得

授業全体の目標は「Java 言語とプログラミングの基礎の習得」となっている．実用言語としての Java 言語の習得を通じて，CS のプログラミングの基礎の習得が可能である，という考えに基づき，

次の調査を実施した．

- **類推テスト**

  類推テストは，Java の構文に関する簡単な問題を，Nigari を学習した経験から類推して解かせることで，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．

  Nigari System での学習を終え，Java に移行した後の授業では，その日学ぶ Java の構文についての説明を行う前に，類推テストを行った．

  類推テストの内容は付録 A.5 に示す．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．

  また，類推テストをどのように解いたかを問うアンケートも実施した．これは，その学習者が予め Java の知識をもっているかどうかを調査する目的で実施した．Java の知識をもっていないにもかかわらず，問題を解くことができれば，Nigari の知識だけで Java の問題を解くことができたといえる．

- **共通試験**

  すべてのクラスに共通に実施された試験の結果を，クラス別に評価する．第 1 クラス以外のクラスは Nigari System を用いていない．Nigari System を用いた場合と用いない場合で，学習効果に違いがあるかを調べた．

- **理解度調査**

  7 月 7 日の最終授業で実施したアンケート（最終アンケート）において，授業で学習した項目についてどれほど理解ができたかを，自己申告で回答してもらった．これは，すべてのクラスの学習者を対象とした．この結果から，Nigari System を使った学習とそうでない学習で，理解に差が生じるかどうかを評価した．

3.3.3 総合評価

最終アンケートに，次のような質問を用意した．

- 「授業に望ましい授業スタイルはどれか（Nigari System を使うべきか）」
- 「Nigari System を使ったことで Java の理解が深まったか」
- 「Nigari System を利用した後，Java に移行したときにギャップを感じたか」
- 「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」

これらは，Nigari System が授業の楽しさや Java の学習にどのように貢献していたか，あるいはどのような問題点があったかを総合的に分析するために実施した．

なお，授業中に実施したすべてのアンケート内容（設問）を付録 A.4 に掲載する．

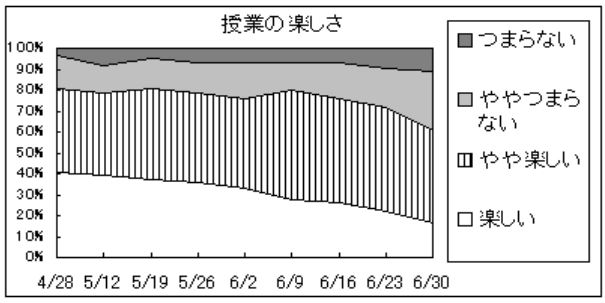


図 3.2: アンケート結果：授業の楽しさの変化（第 1 クラス）

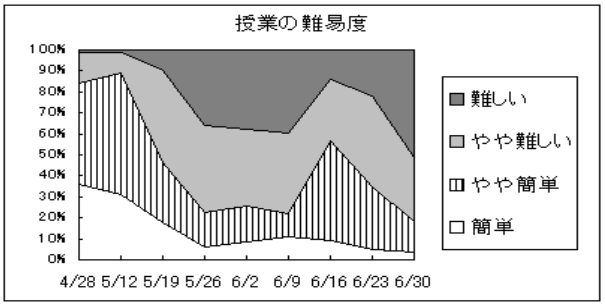


図 3.3: アンケート結果：授業の難しさの変化（第 1 クラス）

3.4 評価

3.4.1 授業の楽しさと難しさの変化

図 3.2 に，授業の楽しさの，授業ごとの変化を示す．図 3.3 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．

授業の楽しさは，Java に移行する前の 6 月 9 日までは,「楽しい」「やや楽しい」の割合の合計が，ほぼ 8 割を保っている．Nigari System による学習を楽しいと感じている学習者が多かったことがわかる．

また，Java に移行する前までは，授業が進むにつれて，授業の難しさは上昇していることがわかる．このことから，難易度は高いが楽しい，という授業を実践することができ，課題を達成したときの喜びも大きかったと予想できる．

一方，Nigari System での学習が終わり，Java の授業に入ると，楽しさの減少が目立つうえに，6 月 16 日頃は一時的ではあるが難易度が低下してしまっている．難易度が低下した原因は，Nigari System で習った変数や while 文の概念を Java でもう一度習ったことであると考えられる．楽しいと思った学習者の中でも，単に簡単だったからそう思っただけという学習者がいる恐れがある．こうした結果から，環境の移行時には若干の無駄があったと言わざるを得ない．

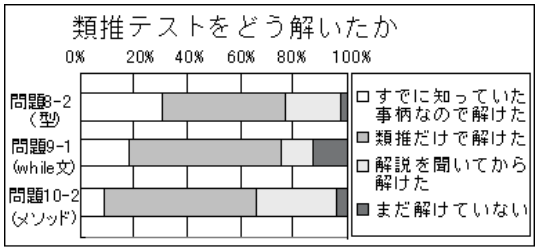


図 3.4: アンケート結果：類推テストをどのように解いたか（回答数:85）

3.4.2 類推テストの解き方

図 3.4 に，類推テストの解き方に関する回答を示す．半分強の学習者が,「類推だけで解けた」と答えた．つまり，半分強の学習者が，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習者も多かった．

3.4.3 総合評価

最終アンケートにおける,「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」（いずれも自由記述）の各設問の回答のうち，Nigari System に関する回答が 30 件含まれた．その詳細を表 3.2 に示す[1] ．

Nigari System 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．Nigari System のプログラムの可視化機能や，Nigari の簡素な言語仕様によって，興味ある出力が得られ，挫折することなく学習ができたといえる．

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」という意見もあり，CS で必要な知識を身につける効果もあったといえる．

一方，Nigari System 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．また，配列や文字入力など，Nigari System 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 System を用いることに疑問を感じる学習者もいた．

なお，Nigari System 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．

図 3.5 に,「Nigari System 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，半数強の学習者が「深まった」と答えており，Nigari System が，Java の学習に有効であったことを示している．一方，15%の学習者が「かえってわからなくなった」と答えている．

図 3.6 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．合計で約 7 割の学習者は，何らかの形で Nigari System を使いたいという意見が得られ，全体の約半数の学習者

[1] 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．

表 3.2: アンケート結果：自由意見

授業に対する意見総数	92 件
Nigari System に関する肯定的意見	19 件
Nigari System 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
プログラムが視覚的	7
オブジェクト指向の理解の助けになる	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
Java とのギャップを感じた	3
Nigari System ではカバーできない点が多い	1
簡単すぎる	1

表 3.3: Nigari System から Java へ移行したときにギャップを感じたか

回答	人数
あまりギャップはなかった。今もない。	15
最初のうちはギャップを感じたが、今はそうでもない。	25
最初は似ていると思っていたが、次第にギャップがでてきた。	22
最初からギャップがあった。今もある。	23

が，今回採用した Nigari System 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．一方，全体の 3 割の学習者は，Nigari System を使わずに Java を最初から学習するのがよいと答えた．

表 3.3 に,「Nigari System を利用した後，Java に移行したときにギャップを感じたか」，という質問に対する回答の割合を示す．約半数（47%）の学習者は，最終的に Nigari System と Java の違いに戸惑いをそれほど感じなかった．残りの半数（53%）の学習者は，最後まで Nigari System と Java の違いに戸惑いを感じていたといえる．

大方の学習者からは，Nigari System を用いることに対して賛同を得られたが，Nigari System から Java の移行に戸惑いを感じたり，Nigari System は用いないほうがよいと思ったりする学習者もいた．自由意見に書かれた意見によれば，その理由として，配列やメソッドの宣言方法など，Java と Nigari で仕様が異なっている部分に違和感を覚え，混乱してしまう，といったものが挙げられる．

3.4.4 共通試験

共通試験の結果を表 3.4 に示す．

第 1 クラスの成績が最も高い．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．トラブルは第 2 クラス で多発し，第 3 クラス で若干問題が発生したものの，第 1 クラスではほとんど発生なかった．

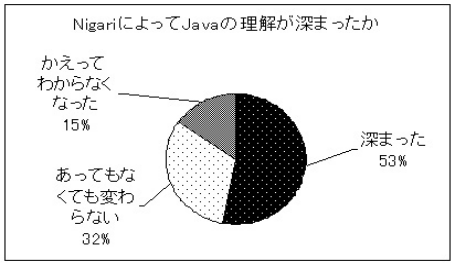


図 3.5: アンケート結果：Nigari System によって Java の理解が深まったか（回答数:85）

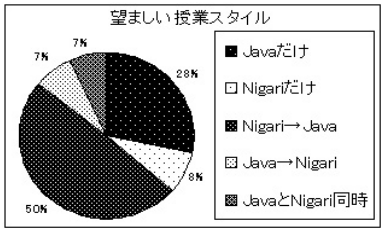


図 3.6: アンケート結果：望ましい授業スタイル（回答数:85）

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 3.7 に示し，問題内容を表 3.5 に示す．第 2，第 3 クラス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．

よって，Nigari System による Java の学習効果は，Java のみを使った場合とほぼ同等であるといえる．第 1 クラスは他のクラスより，Java そのものの学習時間は短いはずであるが，Nigari System を用いることで，Java の学習を行った場合とほぼ同等の効果を得た，ということもできる．

3.4.5 理解度調査

理解度調査の結果から，各学習者の各学習項目の理解度を得点化した．学習項目毎の選択肢には，「理解できた」「まあまあ理解できた」「あまり理解できなかった」「ほとんど理解できなかった」があり，それらに 3 点，2 点，1 点，0 点という得点を与えた．その得点の学習項目別，クラス別平均点を図 3.8 に示す．

第 1 クラスでは，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 を学習していたときにあまり扱ったことが少なかったこと，Java での宣言形式が，Nigari のそれと異なっていることなどから，理解しにくくなった．類推テスト（3.4.2 参照）の結果から見ても，メソッドの構文は Nigari と Java で違いを感じた学習者が多かったことがわかる．

- 配列については，Nigari System では学習せず，Java になって初めて学習した．さらに，学習

表 3.4: 共通試験結果

クラス	平均点	受験者数
1	85.8	81
2	75.4	80
3	81.1	80

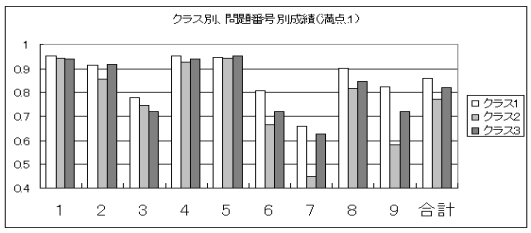


図 3.7: 共通試験結果：問題別

時間が少なかった.

3.5 考察

本研究の目標である「授業を楽しくする」「CS に必要なプログラミングの基礎を学ぶ」といったことが，Nigari System を用いた授業によっていかに達成できたかを考察する

3.5.1 授業を楽しくする

実験結果によれば，Nigari System を授業に用いることで次のような効果が得られた.

- **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.**

  WEB 教材の中に書いたプログラムの解説は，プログラムの一部分を “おまじない” として誤魔化すことはほとんどなく，プログラムの全体をもれなく説明することができた．学習者が理解できない部分を極力減らすことで，プログラミングは難しいと思わせない工夫ができたと言える.

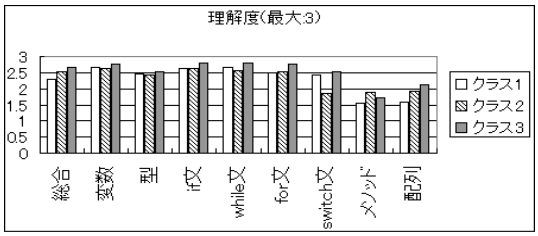
  また，最終アンケート結果によれば Nigari について「Nigari は変数の型などの細かいところを気にしなくて良いので、考えたことがすぐに表現しやすかった」「Nigari を使うことで、小難しいことを意識せずにプログラミングが理解できた。」など，Nigari は敷居が低くとっつきやすいと感じた学習者がある程度いたことがわかる.

- **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.**

  図 3.2 から，Nigari System を用いて授業を行っている間は，授業が楽しいと感じている学習者の割合が高いと読むことができる.

表 3.5: 共通試験問題内容

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



（回答数： 第 1 クラス 79 人，第 2 クラス 39 人，第 3 クラス 34 人）

図 3.8: アンケート結果：クラス別，分野別理解度

その理由として,「Nigari は実際オブジェクトが動いてるところが見えたりして興味を持ちやすかった」「アニメーションできるプログラムを、Nigari を用いて最初から作った点（が良かった)。無味乾燥な文法をコマンドライン上で学習するよりは、はるかに良い」といった，Nigari System によってオブジェクトが可視化され，プログラムの動きがアニメーションとして出力される点を指摘する意見が見られた.

これらの効果によって，Nigari System を用いたことで，授業を楽しく興味あるものにできたといえる.

3.5.2 CS に必要なプログラミングの基礎を学ぶ

類推テストの結果によれば，多くの学習者が，Nigari で習得した変数や制御構造の概念だけを使って，まだ習っていない Java のプログラムの振舞いを予測することができた．このことから，Nigari System 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 に応用できたといえる.

また，Nigari System を利用した学習したクラスの学習者は，共通試験において，Java のみを用いて学習した他のクラスと同じ内容の試験を受験し，他のクラスと遜色ない成績を修めている．このことからも，Nigari System は，実用言語である Java の概念を学ぶために十分な環境であるといえる.

しかし，Nigari System を用いて，その後 Java を習得する，というコースデザインについては改

善の余地があると考えられる．実験では，2 つの環境をたった半期で用いたため，両者の言語仕様の違いから戸惑った学習者が多かった．理解度調査においてメソッドや配列の学習の理解が進まなかったのも，言語仕様の違いが原因と考えられる．また，Nigari と Java でほぼ同じ構文であるはずの，代入文や制御構造の学習を 2 度行っており，難易度が一時的に低下していることなどから，授業計画に無駄があるといえる．

これらの点についての改善策は， 4 章で述べることにする．

3.6 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 System 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ることを示した．これによって「授業を楽しくする」という目標を達成できた．

また，Nigari System で学習したことを，実用言語である Java にも適用できることを示し，「CS に必要なプログラミングの基礎を学ぶ」という目標もある程度達成されたが，Nigari System を用いた後で Java へ移行するというコースデザインには，両者の環境の違いに起因する問題点が見つかった．

実験結果を踏まえ，次章では，初心者向き コースデザインを見直し Nigari System から Java への無理のない移行を支援する環境，およびそれを用いたコースデザインを提案する．

第4章　円滑なレベル移行を支援する環境「N-Java」の設計と評価

4.1　はしがき

プログラムの構成要素のうち，まだ習っていない要素を読んだり書いたりするのは，プログラミング初心者にとっては負担であり，プログラミングの楽しさを半減させる恐れがある．また，これらの要素を初期の段階で説明するは難しく，“おまじない” として教えざるを得ない状況が多い．

本章では，プログラミング学習の入門段階において，“おまじない” を教えないで済むコースデザインを提案する．

2 章では，おまじないを教えないで済む環境として Nigari System を提案した．Nigari System で教えられる内容は，CS のプログラミングの基礎に求められるすべての内容を網羅していないため，Nigari System の後で，実用言語として Java も習うようなコースデザインを採用した．この場合，Java に移行した後で，おまじないを教えることなく最後まで学習ができるかどうかはわからない．

そこで，プログラミングの入門授業において，教育用言語環境としての Nigari System から，実用的な言語環境としての Java（JDK）に至るまでの過程において習うべき学習要素を，“レベル” という単位で分類したコースデザインを提案する．レベルの分類の際には，その学習要素と言語環境の関係に着目し，おまじないを教えずに済むように配慮した．また，このコースデザインに適した学習環境 N-Java の設計と実装を示し，実験によりその評価を行う．

4.2　プログラミング入門において習う要素

プログラミング入門授業において，学習すべき項目（学習項目）を挙げると，おおよそ次のようになる．

- 変数
- 型と変数宣言
- 制御構造（条件分岐・繰返し）
- 配列
- メソッド（サブルーチン）
- クラス（オブジェクト）

本章では，これらの項目をどのような順番で教え，その各項目でどんな学習環境を用意すべきかを “コースデザイン” として決めることにする．

4.3 コースデザインの方針

コースデザインは，おまじないを教えなくてすむような順序立てにしなければならない．つまり，あらゆる段階で，プログラムのすべての場所を理解できるように，学習順序および，その時点で用いる言語環境を選択する必要がある．

各学習項目を学習する場合，プログラムのすべての文法要素に着目することはまれで，その学習項目に関連する特徴的な文法要素だけを着目することが多い．例えば，変数の項目であれば代入文，分岐の項目であれば if 文に着目することが多い．このような各学習項目に強く関連した文法要素，いいかえればその学習項目を学習するのに必要十分な言語の文法要素を，その学習項目に対応する “着目要素” と呼ぶことにする．もちろん，ある学習項目に複数の着目要素が対応する場合もある．

学習環境は，今学習している学習項目に対応する着目要素，あるいはそれ以前に学習した学習項目に対応する着目要素だけを見せるようにする．まだ習っていない学習項目だけに対応するような着目要素は，おまじないとしてしか教えることができないので，適切に隠すのが望ましい．

4.4 レベルによる学習項目の分類

それぞれの学習項目に対して，それを学習する際に最適な学習環境を提供したい．そこで，各学習項目の着目要素をもとに，適切な学習環境を考察し，その学習環境の形態の違いにより，学習項目を分類することを考える．この分類された学習項目の集合を “レベル” と呼ぶことにする．

次に，レベルの分類と，そのレベルの学習に適した環境，適切な学習の順序について議論する．最後に，レベル分けと，学習順序を表 4.1 にまとめる．

4.4.1 レベル 1

Nigari は，Java の代入式や制御構造の書き方がまったく同じようになるように設計されているため，これらの概念は Nigari で教えるのが適している．Nigari で教えられる範囲の概念，いいかえれば着目要素の書き方が Nigari と Java で同じになるような学習項目は**レベル 1** に属するものとする．

レベル 1 においては，Nigari を言語として使った演習を行うのがよい．なぜなら，Nigari は，Java と比べて初心者にとってのおまじないが少なく，しかもプログラム内に本質的に書く内容は Java と同等であり，その後の Java の学習に支障を来さないからである．

4.4.2 レベル 2

Nigari と Java の言語としての大きな違いに，変数宣言の有無がある．Nigari は，すぐに実行ができることを目指して設計されているため，変数宣言が不要である．変数宣言が不要であることと関連して，変数に格納される値の型にも制限がなく，1 つの変数にあらゆる型の値が格納できる（変数に型がない）．よって，変数宣言の概念は Nigari では習うことができないため，Java で習う必要がある．

また，Nigari と Java では，メソッドやクラスの宣言，配列の初期化の方法が違う．これらの概念を Nigari で習うと，Nigari と Java で 2 通りの書き方を覚える必要がある．メソッド，配列，クラスは Nigari で習うよりも，最初から Java で習うほうが，混乱を引き起こさないで済む．

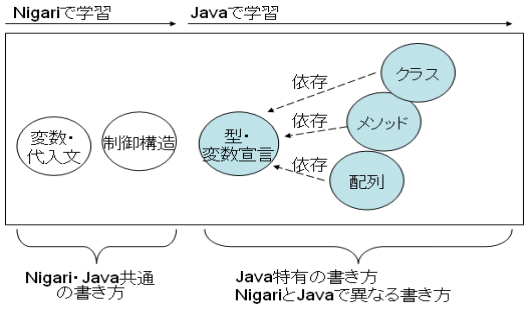


図 4.1: 学習順序と学習環境

型と変数宣言，メソッド，配列，クラスといった学習項目は Java で習うようにする．その中でも最初に習うべき概念は型と変数宣言であると考える．なぜなら，Java におけるメソッド，配列，クラスに関する文法には，変数の型の概念に強く依存しているからである．メソッドの宣言には引数や戻り値の型の指定が必要である．配列にも要素の型の指定が必要である．また，あるクラスのインスタンスを変数に格納するには，そのクラスの宣言において指定したクラス名と同じ名前の型を使って変数を宣言しなければならない．これらのことをまとめると図 4.1 のようになる．

レベル 1 での Nigari を使った学習を終えたら，Java へ移行し，まず型と変数宣言を習うことになる．しかし，型と変数宣言を習う時点では変数や制御構造しか習っていないため，クラスやメソッドの概念を学習者は知らない．しかし，Java はクラスやメソッドの宣言を書かないかぎりプログラムが実行できないため，これらの構文がおまじないになってしまう．

そこで，型と変数宣言を習う段階では，クラスやメソッドの宣言を隠して，型と変数宣言の概念だけに集中できるような学習環境を新たに用意する必要がある．このような学習環境下で学ぶ学習項目を，**レベル 2** に属する学習項目とする．具体的には，Nigari から Java に移行し，クラスやメソッドの概念を習う前までの学習項目がレベル 2 に属する．

4.4.3 レベル 3

レベル 2 で，型と変数宣言の概念を習うと，配列，メソッド，クラスなどの学習項目を習う準備ができたことになる．

これらの概念をどの順で習うかを考える．

- 配列は，クラスやメソッドの概念とは独立であるため，レベル 2 の学習環境でも習うことができる．配列は，レベル 2 に属するものとして扱う．
- メソッドはレベル 2 では習えない．レベル 2 では，学習環境によってメソッドの概念が隠されているためである．
- クラスは，メソッド同様レベル 2 では隠されているので習えない．さらに，クラスの宣言はメソッドの宣言を包含するため，メソッドの概念を先に習っておかなければならない．

よって，まずレベル 2 の段階で配列を習い，次のレベルでメソッドを習い，その後クラスを習う，という順番がよい．

表 4.1: 学習順序

内容	レベル
変数	1
制御構造	1
型と変数宣言	2
配列	2
メソッド	3
クラス	4

そこで，メソッドの概念を習う学習項目は**レベル 3** に含めることにする．このレベルで用いる学習環境は，メソッドの宣言を書くことができる．しかし，クラスの宣言はまだ習っていないので，隠す必要がある．

4.4.4 レベル 4

メソッドの学習が終わり，クラスに関する学習項目を習う段階になると，クラス宣言の構文を含めた，Java のほぼすべての構文に関して学習をする必要が出てくる．よって，この段階では Java をそのまま使った学習を行っても，ほとんどおまじないに煩わされることはない．この段階で学習する項目は**レベル 4** に属するものとする．

4.5 プログラミング学習環境 N-Java

本章では，各レベルにおいて使われるべき環境を統合し，教育用言語環境としての Nigari System から， 実用言語環境としての Java までを段階的に学習する環境 “N-Java” を提案する．N-Java の動作画面を図 4.2 に示す．

N-Java は，レベルごとに利用する言語（Nigari/Java）やその書き方を変化させながらプログラムを作成できる “レベル別環境” をもつ．また，Nigari System にも用意されていた “可視化機能” が拡張されている．ここでは，レベル別環境と可視化機能の概要を示す．N-Java についての詳細 [30][31] は別途参照されたい．

4.5.1 レベル別環境（4 レベルエディタ）

利用者（学習者）は，現在学習している学習項目のレベルに応じて，N-Java におけるプログラムの書き方，実行方法を任意に切り替えることができる．各レベルにおける N-Java の振舞いを示す．

- **レベル 1**

  このレベルでは，代入文，制御構造を学習するのが目的であるので，Nigari を使ったプログラムを記述する．

  図 4.3 のように，Nigari のソースファイルを記述する．ソースファイルには文だけを並べて書けばよく，変数宣言，クラスの宣言，メソッドの宣言などの構文は書く必要はない．クラス名

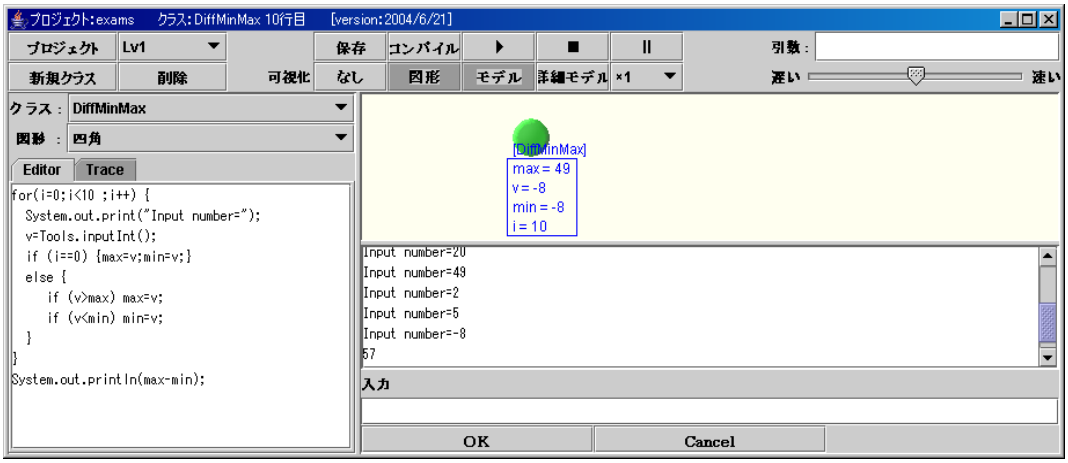


図 4.2: N-Java の動作画面

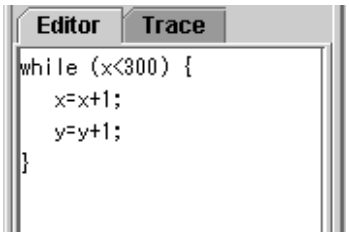


図 4.3: レベル 1

はこのソースファイルの名前で決定される．

プログラムは，ソースファイルによって定義されるクラスの集合である．N-Java からこのプログラムを実行する場合，“実行開始クラス” を指定する必要がある．プログラムが実行されると，実行開始クラスのオブジェクトが環境により自動的に生成され，そのクラスのメイン文列を実行する．

なお，レベル 1 に限らず，すべてのレベルにおいて，実行の際には，実行開始クラスを指定する必要がある．

- **レベル 2**

  レベル 2 では，変数宣言と変数の型の概念を学習するため，Java を使ったプログラムを記述する．これ以降のレベルはすべて Java を使う．

  図 4.4 のように，ソースファイルを記述するための欄として「変数宣言」「メインルーチン」の 2 つが設けられる．メインルーチンの欄には，文を並べて書き，変数宣言には，それら文の中に出現した変数を明示的に宣言する必要がある．宣言する場合には，その変数の型も決めなくてはならない．

  メインルーチンに書く内容を代入文や if 文，while 文などの制御文に限定すれば，レベル 1 で

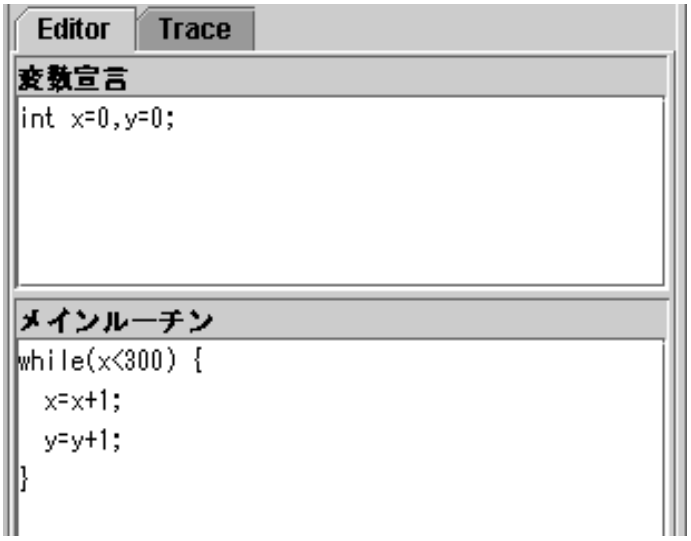


図 4.4: レベル 2

記述したプログラムは，レベル 2 で（変数宣言さえ追加すれば）そのまま実行できる．これは，Nigari と Java で代入文，制御文の文法を同じにしているからである．

N-Java 上でソースファイルを編集すると，図 4.5 の右側で示したような，Java のソースファイルが N-Java によって自動生成される．プログラム本体は，run という名前のメソッドの内部に挿入され，変数宣言は，このクラスのインスタンス変数として宣言される．

プログラムを実行すると，実行開始クラスのオブジェクトが環境により自動的に生成され，run メソッドが呼び出される．

- **レベル 3**

  レベル 3 は，メソッドの学習を行うため，図 4.6 のように，ソースファイルには，インスタンス変数と，メソッドの宣言を書く必要がある．クラスの宣言は書くことができない．

  N-Java 上でソースファイルを編集すると，図 4.7 の右側で示したような，Java のソースファイルが自動生成される．

  レベル 2 同様，プログラムを実行すると，実行開始クラスのオブジェクトが環境により自動的に生成され，run メソッドが呼び出される．

- **レベル 4**

  レベル 4 では，クラス・オブジェクトの学習をするため，図 4.8 のように，クラス宣言などを含めた，Java のソースファイル全体を編集できる．

  レベル 2，3 同様，プログラムを実行すると，実行開始クラスのオブジェクトが環境により自動的に生成され，run メソッドが呼び出される．

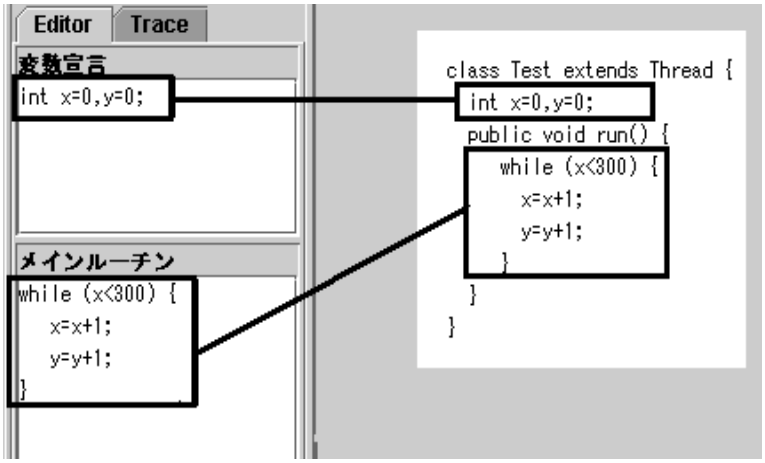


図 4.5: レベル 2 におけるソースファイルの自動生成

4.5.2 可視化機能

N-Java には，Nigari System 同様，プログラムの振舞いを可視化する機能がある．

Nigari System の可視化機能は，オブジェクトの存在を意識させ，オブジェクト指向の基礎を学ばせるという目的をもっていたが，単に画像が表示されただけでは，オブジェクトのもつ変数がどのような状態で，他のオブジェクトとどのような関係をもっているか，あるいは配列の要素にどのような値をもっているか，などということを詳しく把握できなかった．そこで，N-Java では，オブジェクトや配列のデータ構造（これらをモデルと呼ぶ）を表示するモデル表示機能を用意した．モデル表示の例を図 4.9 に示す．

- **変数表示**

  オブジェクトのもつすべての変数の内容を，文字情報として表示する．

- **配列表示**

  配列の中身をすべて表示する．

- **参照関係表示**

  あるオブジェクト（配列を含む）が他のオブジェクトを参照している場合は，それらを矢印で表示する．

また，従来の Nigari System と同様に画像のみを使ったアニメーションだけを表示させたい場合は，モデルを非表示にすることも可能である．

4.6 N-Java の評価

本章で提案したコースデザインと，そのための環境 N-Java の有効性を評価する．

評価は，実際の大学の授業で，このコースデザインと N-Java を適用した結果をもとに行う．

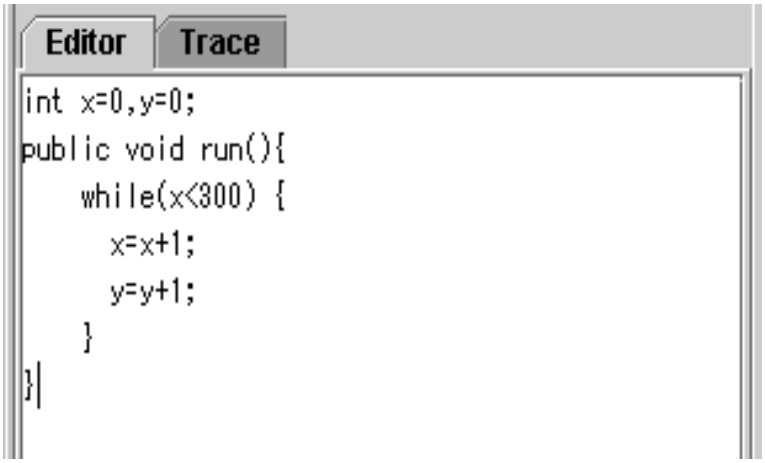


図 4.6: レベル 3

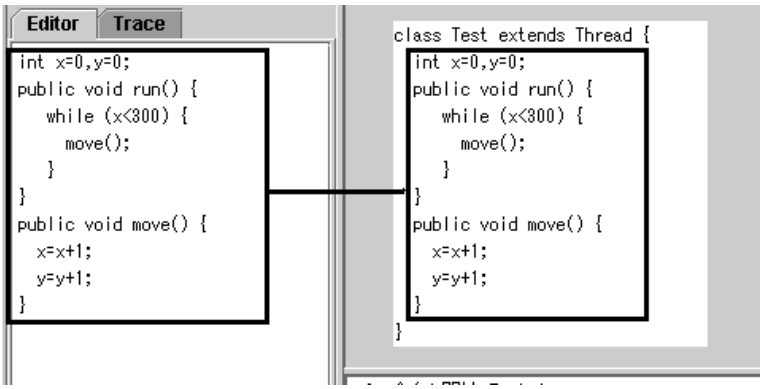


図 4.7: レベル 3 におけるソースファイルの自動生成

3 章で挙げたような，Nigari System から Java へ移行したときの，両者の環境の違いによる戸惑いや，2 つの環境を習得することから生じるコースデザイン上の無駄といったような問題を解消できたかどうかを評価する．

4.6.1 授業の概要

- **授業名:** 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 言語を用いた，変数，制御構造，配列など手続き型プログラミング（言語）の基本概念の習得
- **期間，回数:** 2004 年 4 月 12〜同年 7 月 5 日， 12 回

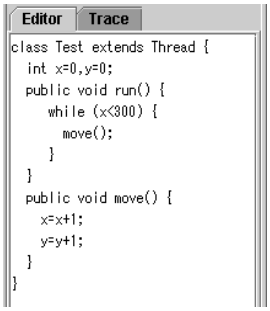


図 4.8: レベル 4

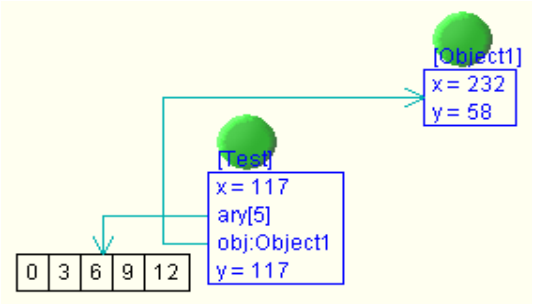


図 4.9: モデル表示機能

- **1 回あたりの授業時間:** 2 時限（約 3 時間）
- **人数:** 約 270 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．クラス分けは希望調査により決めた．希望もれはなかった．
- **教科書:** 「Java 言語プログラミングレッスン（上）」[29]
- **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．毎回 2，3 題程度（合計 22 題）のレポートを出題．

第 1，2 クラスは初心者クラス（ただし，プログラミング経験者も含む），第 3 クラスは中級者クラスとした．第 1 クラスでは，N-Java を利用した授業を行った．

4.6.2 コースデザインの評価

表 4.2 に，N-Java を適用した授業（2004 年度授業）における授業内容と，Nigari System を適用した授業（2003 年度授業）における授業内容とを示す．2004 年度の授業のテキスト [32] は Web に

表 4.2: 授業内容

回	2004 年度	2003 年度
1	講義概要	講義概要
2	N-Java 基本操作	PC 基本操作
3	変数/ いろいろな値	Nigari 基本操作
4	while 文/ if 文（1）	変数 /while 文（1）
5	if 文（2）/ 問題集	while 文（2）/if 文（1）
6	問題集 / 変数宣言と変数の型	if 文（2）
7	配列	複数のオブジェクト/マルチスレッド
8	オブジェクト	メソッド / オブジェクトの実行時生成
9	レポート解説	Java の概要 / 変数の宣言 / if 文（3） / while 文（3）
10	メソッドとローカル変数	制御構造/メソッド/文字列
11	メソッドの復習	配列 / コマンドライン引数 / 並べ換え
12	JDK/ デバッグ	並べ換え（つづき） / 文字入力

て参照されたい.

2003 年度においては，第 8 回までは Nigari System を使用し，第 9 回から Java へ移行した．その際に，制御構造，メソッドなどの概念を Java に合わせた格好で再度学習させたが，2004 年度においては，最初から最後まで N-Java を利用していたため，各概念を 1 回だけ教えればよくなり，授業の無駄を省くことができた．また，それにより余った時間帯を利用して，問題集や解説等の項目を増やすことが可能になった.

なお，レベルは次のように設定した.

- レベル 1 は,「問題集」まで使用．変数，制御構造の基礎を学ばせるのに使った.
- レベル 2 は,「変数の宣言と変数の型」から「レポート解説」まで使用．変数宣言を伴うプログラムを作成させるのに使った[1] .
- レベル 3 は，メソッドを学習させるのに使った.
- レベル 4 は，あまり使用しなかった．JDK を学習するにあたり，今まで N-Java のプログラムと，実際の Java プログラムとの対比をさせる程度にとどめた.

4.6.3 アンケート結果に基づく評価

この授業に対するアンケート調査を行った.

ここでは，次の各質問に対する回答結果をもとに，学習者が楽しく授業を受けられたか，また，レベルの変化によって戸惑を覚えることなく学習ができたかどうかを検証する.

なお，アンケートのすべての質問は付録 A.8 に掲載してある.

- 授業の難易度 （毎回のアンケート）

[1] 第 8 回の「オブジェクト」もレベル 2 で扱っているが，この段階ではクラス，メソッドの詳細な説明を避け,「オブジェクトを新しく生成することができる」「オブジェクトには値を複数もたせることができる」という概念だけを掴んでもらうにとどめた．具体的な演習内容 [32] は Web を参照のこと.

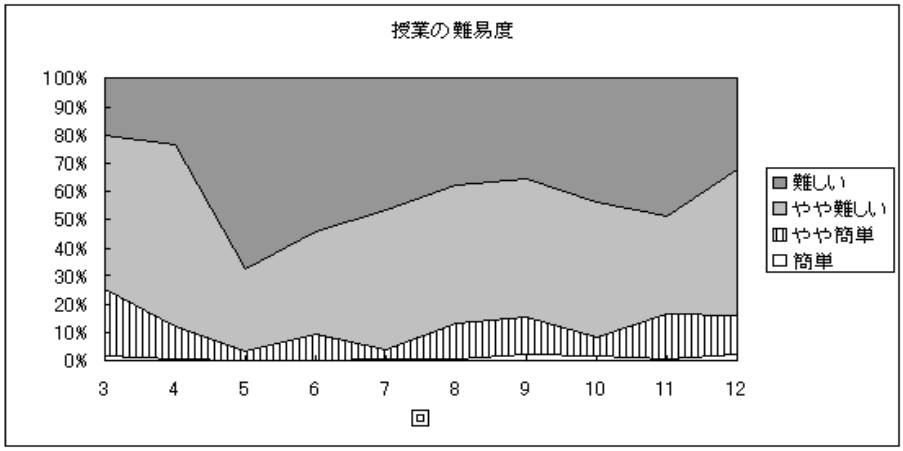


図 4.10: 授業の難易度の変化

- 授業の楽しさ （毎回のアンケート）
- N-Java のレベル変化時の戸惑い （最終アンケート）
- 授業の良かった点 （最終アンケート）
- 授業の悪かった点 （最終アンケート）

授業の難易度の変化

授業が難しかったかどうかのアンケートを毎回の授業で行った．その授業ごとの変化を図 4.10 に示す.

どの授業においても，全体の 9 割近くが授業を難しいと感じている．

3 章で示した，Nigari System を利用した授業における，授業の難易度の変化（図 3.3）と比べると，難易度は高いと感じた学習者が多く，難易度の日による変化はそれほどなかった．ただし，第 5 回，第 11 回は,「難しい」と答えた学習者が増えている．これらの回は，いずれも問題の演習を主体にした授業であったため，問題そのものが難しかったことに起因するものと考えられる．

いずれにせよ，Nigari System を適用した授業のように，環境の変化にともなって急激に授業が簡単になってしまう，といったことは防ぐことができた．

授業の楽しさの変化

授業が楽しかったかどうかのアンケートを毎回の授業で行った．その授業ごとの変化を図 4.11 に示す.

この結果によれば「楽しかった」「やや楽しかった」を合計した人数の割合は，どの授業でも 7 割前後を占めていることがわかる．

3 章で示した，Nigari System を利用した授業における，授業の楽しさの変化（図 3.2）と対比すると，Nigari System を利用した場合は，Java に移行したときに楽しさが減少したが，N-Java を利用

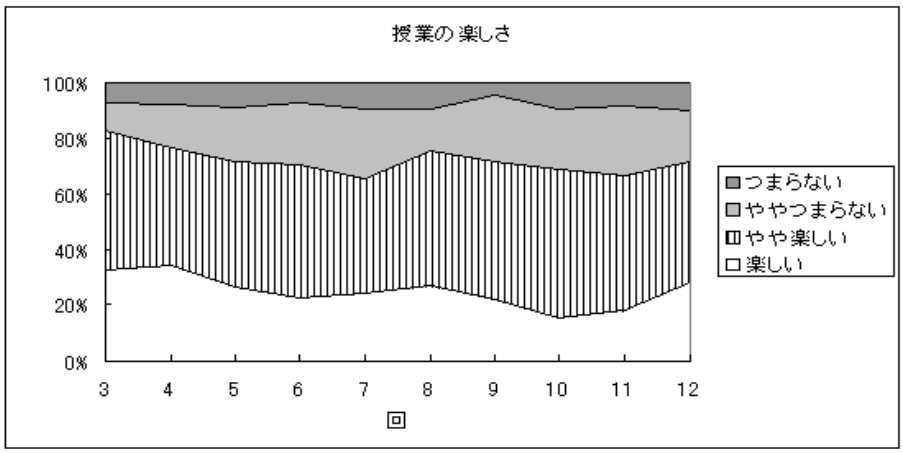


図 4.11: 授業の楽しさの変化

表 4.3: アンケート:移行したときに戸惑いがあったか（人数）

	レベル 1→ 2	レベル 2→3
戸惑いなし	43	23
あまり戸惑いなし	25	23
少し戸惑いあり	18	32
戸惑いあり	4	12

した場合は，楽しさを一定に保っていることがわかる．

レベルの変化にともなう戸惑い

「レベル 1 からレベル 2 に移行したときに戸惑いがあったか」「レベル 2 からレベル 3 に移行したときに戸惑いがあったか」という設問を設けた．これらの設問の結果を表 4.3 に示す．

この結果によれば，レベル 1 からレベル 2，つまり変数宣言を行うようになった時点では，大多数（76%）の学習者が，あまり戸惑いを感じることなく移行できた．

一方，レベル 2 からレベル 3，つまりメソッドを定義するようになった時点では，戸惑いを感じなかった学習者は約 51%に減った．3 章において示した「Nigari System を利用した後，Java に移行したときにギャップを感じたか」というアンケート結果（表 3.3）と比べると，レベル 2 からレベル 3 へ移行したときの戸惑いは，Nigari System から Java に移行した場合と同じ程度であったといえる．ただ，その原因がレベルの変化に起因するものなのかは一概には言えない．レベル 2 からレベル 3 に移行したときの授業では「メソッドとローカル変数」の概念を習った．この回のアンケートに寄せられた学習者の反応には「引数とか戻り値のあたりがよく分からなかった。」「メソッドの意味はある程度分かったが、いまいち使い方がわからなかった」「メソッドはまだ使いこなす自信がありません。」など，メソッドの難しさを主張するものが多かった．レベル 2 からレベル 3 への移行の際の戸惑いは，メソッドそのものの難しさに起因している可能性もある．

表 4.4: アンケート:この授業の良かった点

項目	人数
授業がわかりやすい	26
N-Java が良かった	16
（4 レベルエディタ）	（7）
（アニメーション表示）	（2）
（その他）	（7）
TA によるサポートが良かった	7
Web 教材が良かった	5
その他	8

表 4.5: アンケート:この授業の悪かった点

項目	人数
レポートが多い	10
レポートの解説が足りない	6
レポートが難しい	6
授業のスピードが速い	5
授業が難しい	4
N-Java より JDK を使いたかった	3
Web 教材に不満	3
その他	7

授業の良かった点

学習者が答えた，授業の良かった点を表 4.4 に挙げる．

まず，圧倒的に，授業のわかりやすさが支持されたが，N-Java を用いたことによる良さを評価した学習者も 16 人いた．そのうち， レベルに応じてプログラムを書き方が変わる機能（4 レベルエディタ）を使ったことが良かったという意見が 7 件，アニメーションが表示されて面白いという意見が 2 件であった．モデル表示機能に関する評価はまったく見られなかった．

授業の悪かった点

学習者が答えた，授業の悪かった点を表 4.5 に挙げる．

ほとんどの学習者が，授業や課題の難しさ，速さに関する不満を述べた．

2003 年では，Nigari から Java への移行が遅い，Nigari から Java になったときに戸惑ったなど，環境の移行に関する不満が 10 件ほどあったのに対し，2004 年度では，N-Java により，JDK を使う機会が減ったという意見がわずか 3 件あっただけで，環境の移行に関する不満はあまり見当たらなかった．

表 4.6: 共通試験問題内容

問題番号	内容
1	if 文（条件式）
2	while 文（同等の振舞いをする構文を見つける）
3	while 文・計算量（探索）
4	整数型の桁あふれ，コンピュータにおける数値表現
5	switch 文
6	オブジェクトと配列を利用した，探索，並べ替えなどのアルゴリズム
7	デバッグ（引数の型の違い，配列を走査方法の間違い，文法的間違い）

表 4.7: 共通試験のクラス別平均点

クラス	平均点	受験者数
1	47.2	97
2	58.8	98
3	61.9	48

4.6.4 成績に基づく評価

全体の成績は，平常点，クラス別期末試験，共通試験を用いて決めた．共通試験は，第 1，第 2，第 3 クラスすべての学習者に出題され，プログラムや文章の一部を選択肢または自由記入によって穴埋めする試験である．共通試験の問い別の内容を，表 4.6 に示す．また，すべての問題を付録 A.7 に示す．

共通試験の結果を表 4.7 に示す．このように，第 1 クラスの平均点は，他のクラスより低い，という結果になった．しかし，各クラスにはプログラミング経験者（授業前にプログラミングをしたことがある）とプログラミング未経験者が混在しており，その比率がクラスごとに異なっていることを考慮する必要がある．そこで，各学習者のプログラミング経験や，設問別の得意不得意を考慮した詳しい解析を行うことにする．

授業開始前には，各学習者がプログラミング経験者であるかどうかを調べるための調査が行われていた．これをプレテストと呼ぶ．プレテストの問題は付録 A.6 に示す．プレテストへの解答は任意であった．プレテストの得点を，プログラミング経験者であるかどうかの指標に用いる．プレテストで 1 問でも正解を得た学習者を経験者とし，解答なしあるいは解答したが正解がなかった学習者を未経験者とする．

プログラミング経験者と未経験者の人数を表 4.8 に示す．このように，第 1 クラスは未経験者が圧倒的に多かったことがわかる．これは，共通試験の成績が低かった一因と考えることができる．

次に，共通試験の成績を，未経験者だけに限定した場合の成績を表 4.9 に，その成績を問い別に分類したものを表 4.10 に示す．

この結果から，第 1 クラスは，問 4，問 6，問 7 の成績が低いことがわかる．特に問 6 の成績は極端に低くなっている．これらの問題は，3 章の 2003 年度授業の最後に行った共通試験（2003 年度の試験）の内容（表 3.5）にはほとんど含まれていないものである（デバッグの問題はあったが，いずれも文法的な間違いを探させる問題であった），一方，問 4，問 6，問 7 以外の問題は，プログラムの文法，書き方などを問うもので，2003 年度の試験と似たような内容である．これらの得点は，他

表 4.8: クラス別プログラミング経験者と未経験者の人数

クラス	未経験者	経験者
1	91	6
2	67	31
3	13	35

表 4.9: 共通試験成績（プログラミング未経験者）

クラス	得点の平均
1	46.9
2	55.9
3	58.4

のクラスとあまり違いがないことがわかる．

よって， 授業の学習効果としては，2003 年度授業とほぼ同等の成果を挙げているということができる．

4.6.5 授業の楽しさと学習効果の関係に基づく評価

本論文は,「楽しく学習を続けることが，学習効果を上げるために不可欠である」という仮説のもとに議論を進めているが，この仮説が信用に足るものであるかを，授業の実践結果をもとに評価する．評価の対象になる学習者は，2004 年度に早稲田大学 CS 学科「プログラミング A（前期)」「プログラミング B（後期)」を受講した学習者である．

プログラミング A は，本章の N-Java の評価実験を行った授業である．その概要は 4.6.1 で示した通りである．また，プログラミング B は，5 章で後述する proGrep の評価実験を行った授業で，詳細は 5.8.2 に示してある．

プログラミング A，B ともに必修科目であり，3 つのクラスに分けて，別の教員が担当して行う．共通の達成目標と，標準教科書とが用意されており，共通試験を行い，その共通の達成目標に対しての達成度を評価している．ただし，その具体的講義の進め方は教員に任されている．また，プログラミング A, プログラミング B の間では，クラス替えが行われている．

授業の楽しさと学習評価の関係の評価には，次の結果を用いる．

- 年間の授業アンケート

  前期，後期ともに，毎回の授業の終わりに，授業アンケートを行った．その中にある「授業の楽しさ」という質問（４段階・選択式）に対する回答内容.

- 共通試験

  前後期，それぞれ最後に施行した共通試験の得点の偏差値.

前期に N-Java を使った授業，後期に proGrep を使った授業を行った対象は，ともに第 1 クラスであり，アンケートをとることができたのは，前後期ともに第 1 クラスの学習者だけである．

共通試験の偏差値は，前期後期それぞれの試験の，全受験者の偏差値とする．なお，前期共通試験の全受験者数は 252 人，後期は 251 人であった．

表 4.10: 共通試験問い別成績（プログラミング未経験者）

問	第 1 クラス	第 2 クラス	第 3 クラス
1	7.86	8.06	8.00
2	8.21	7.66	7.85
3	7.79	8.65	9.77
4	15.93	18.28	18.46
5	4.79	5.34	5.85
6	6.54	11.06	11.08
7	4.30	6.02	6.62

表 4.11: 後期共通試験成績

	楽しさ継続	楽しさ継続せず
学習者数	23	15
標準偏差	8.51	9.17
平均値	40.7	35.5

評価を行う対象は，次の条件にあてはまる学習者である．このような学習者は 38 人であった．

- 共通試験を前期後期ともに受験
- アンケート回答数が，前期（10 回）後期（9 回）あわせて 19 回中 15 回以上（もちろん，ともに第 1 クラスに所属）

「授業の楽しさが継続していた」ということを「前期，後期ともに，授業が楽しいと思った」と解釈することにして，次のように学習者を分類した．

- **楽しさが継続した学習者**

  前期，後期両方の科目において,「楽しい」あるいは「やや楽しい」と答えた回数が,「つまらない」「ややつまらない」と答えた回数を上回った学習者

- **楽しさが継続しなかった学習者**

  それ以外の学習者

後期共通試験得点の得点分布に，楽しさが継続した学習者とそうでない学習者の間で有意差があるかを，t 検定により検証すると，$t = 1.74$ であり，10% の有意水準で，両者の得点に差があることがいえる．検定に用いた値を表 4.11 に示す．

学習者の，前期偏差値と後期偏差値，楽しさの継続の有無の関係を図 4.12 に示す．

この図から，次のことがいえる．

- 後期成績が良い学習者のほとんどが，楽しさが継続していた．
- 後期成績が良くない学習者にも，楽しさが継続していた学習者はいる．

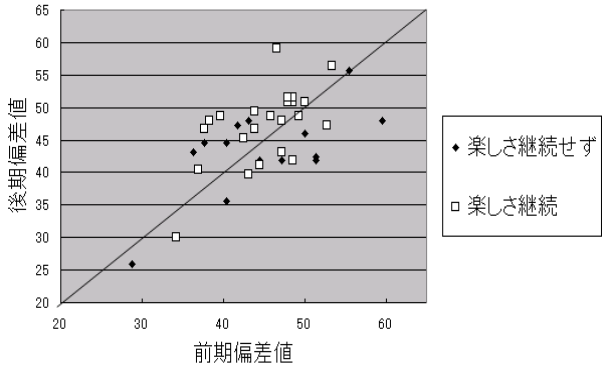


図 4.12: 共通試験偏差値と，授業の楽しさの関係

つまり，最終的によい成績を修めるためには，楽しさが継続することが必要条件である（が，十分条件ではない），ということができる．

これらのことから,「楽しく学習を続けることが，学習効果を上げるために不可欠である」という仮説には一定の信頼を置くことができるといえる．よって，本章で提案した N-Java のような，楽しさを継続させる学習環境は，学習効果を高める上で必要であるということができる．

4.7 関連研究

Java の学習の際に “language level” を設定し，最初のレベルでは，static，public といった文法要素が書けないような制限がかけられ，学習が進むにしたがい，制限をゆるめていくような学習環境 [23][33] が提案されている

これらの環境の主なねらいは，関数型言語である Scheme をすでに学習している学習者が，リスト構造などを扱うプログラムを Java で記述する場合に，難しい文法要素を意識しないで済むようにする，というものである．

Scheme との親和性を考慮しているためか，最初のレベルにおいては，すべてのインスタンス変数が強制的に final（1 回しか代入できない）として宣言される，while 文などの繰返し構造は発展レベルにならないと使えないといった制限がかけられている．一方，クラスやメソッドの宣言は最初のレベルから必須である．

このような仕組みは，本章で提案したような手続きを最初に学習するコースデザインには適用しづらいと考えられる．

4.8 まとめ

プログラミング学習において， 教育用言語環境である Nigari System から学習を始め，実用言語環境である Java に移行するまでの過程を段階的に支援するコースデザインと，そのための環境である N-Java を提案し，実際の授業で利用した事例をもとに評価した．

N-Java を用いた 2004 年度授業では，授業の進度（レベル）に合わせて，プログラムの書き方を変

化させることで，2003 年度授業に比べて，授業計画にゆとりをもたせることができた．これは，最初から最後まで N-Java という 1 つのアプリケーションを利用するため，学習者が途中で新たに操作方法を習得する必要がなく，4 レベルエディタのレベルの切り替えによるレベル間の違いにそれほど戸惑いがなかったためと考えられる．

共通試験結果からは，N-Java を用いた授業は，プログラミング言語の基本的な構成要素を学ぶ上で，Java だけを用いた授業と同等の成果を挙げられることがわかった．

また，アンケート結果によれば，授業の最初から最後まで，多くの学習者が興味を持続でき，レベル分けを行うことによって学習内容が理解しやすくなった，という意見が得られた．

これらのことにより，Nigari System を用いた 2003 年度授業において，2 つの環境を使うことによって起きた問題であったところの，途中から興味が持続できなくなる，難易度が突然下がるといった問題点を解決したといえる．

さらに，楽しさを継続したまま学習をすることが，最終的によい成績を修めるということを，アンケートと試験結果から確認し，楽しさを重視した学習環境には学習効果があることを示した．

第5章　学習履歴を利用した学習支援システム proGrep

5.1　はしがき

これまで見てきたように，プログラミングの学習においては，さまざまな学習環境を利用し，実習を主体にした授業を進めることが重要である．しかし，その学習環境が未成熟であるために生じる困難がある．このような困難のために挫折を味わわせてしまうと，授業が楽しいものであるという認識を与えるのは難しくなる．特に，プログラミングの本質と関係ない要因で発生する困難は学習意欲を著しく低下させる恐れがある．

通常，このような学習上の困難に対しては，サポートを行う TA が解決にあたるが，学習者が多い教室では全員の面倒を見ることができない．TA の手を介さずとも，学習環境そのものが困難の解決を支援するような仕組みがあるとよい．そのためには，困難が，いつ，どのように発生しているかを常に把握しておき，その情報から学習環境を改善するための指針を見つけ，それをすばやく学習環境へ還元することが重要であると考える．

本章では，学習者の行動を学習履歴として自動的に記録し，TA・教師がその記録をもとに学習環境の改善を行えるような学習支援システム “proGrep” を提案する．

5.2　学習環境に起因する困難

プログラミングの学習では，実際のコンピュータ上で実習することが不可欠である．その学習環境が未成熟であるために生じる困難がある．

- **不親切なメッセージ**

  言語処理系が出力するエラーメッセージが学習者にとって手助けとならないばかりでなく，かえって困惑させることがある．

- **システムに起因する各種の障害**

  ネットワーク，オペレーティングシステム，言語処理系などの設定の不備，不具合よって生じる障害によって，学習者の予期しない動作不全が起きる．ときにはシステムが停止してしまうことさえある．

こうした困難には，自力ではどうしても解決できないものと，自力で解決すべきものがあり，本来は自力で解決できないものだけを，TA などの補助により解決すべきである．しかし，プログラミング初心者は，今直面している困難が自力で解決できるものであるかどうかが判断できないことが多い．このため，ある学習者は，見境なく TA を呼びつけて自力では何も解決しようとせず，別の学習者は，すべて自力で解決しようとして行き詰まる，という状況が起きる．

こうした状況は，教師や TA にとって負担が増えるだけでなく，学習者が本来味わうべき達成感を奪い取り，プログラミングが楽しいものである，という認識から学習者を遠ざけるものである．

5.3 学習履歴の活用

学習環境に起因する困難を解消する方法として，その困難の対処の役に立つようなアドバイスを自動的に学習者に提示する手法を考える．

学習者へのアドバイスを自動化するための簡単な方法は，学習者が直面するであろう困難の典型的なパターンとその対処法（アドバイス）を予め作成しておき，学習環境がそのパターンに一致した状況になると，自動的にそのアドバイスを提示する，という方法である．

困難の典型的なパターンが何であるかを知るためには，学習者の学習履歴を参照することが不可欠である．ここでは，学習履歴を利用したアドバイスの自動化の手法を順を追って議論していく．

5.3.1 学習履歴の収集

授業で随時に行う小テストやアンケートといったものは，オンラインで行うようにする．そうすることで，その結果も学習履歴として記録しておくことができるようになる．

実習環境はネットワークに接続されている場合が多い．その場合には，学習者の実習履歴を逐一記録していくことも可能となる．例えば，Nigari System ，N-Java などのプログラミング環境に，その学習者の書いたプログラムの内容や，行った操作を専用のサーバに自動送信する仕組みを組み込むことができる．

5.3.2 学習履歴の解析

実習の場では，TA が学習者の支援にあたっている．TA が対処した困難も，TA の活動履歴として記録しておくことにする．TA が対処した（つまり，TA の活動記録に記載された）困難と同種の困難に直面していた学習者が多くいる可能性は高い．

しかし，TA が現実に対処できるのは，すべての学習者が直面した困難全体のごく一部である．したがって，TA が対処した困難と同様のものが，どのくらいの人数にわたって，どのくらいの頻度で発生しているかを知るには，収集した学習履歴を調べる必要がある．学習履歴の量は膨大であり，手作業ですべて調べ上げるのは難しいので，計算機による調べ上げの補助が必要である．これを学習履歴の “検索” と呼ぶことにする．

学習履歴の検索を行うには，TA の対処した困難と「同様」のものが何であるかを表す条件を定式化する必要がある．定式化された条件を “検索条件” と呼ぶ．また，その検索の結果発見された困難を “検索結果” と呼ぶ．検索結果の個数が多ければ多いほど，その困難はより典型的なパターンであるといえる．そのときに用いられた検索条件を記録しておけば，困難の典型的なパターンを蓄積していくことができる．

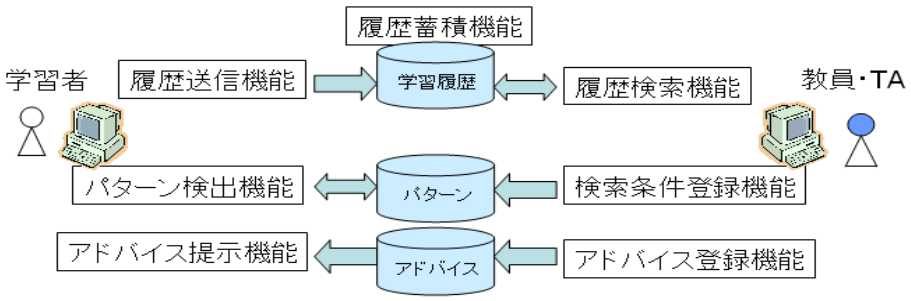


図 5.1: proGrep に必要な機能

5.3.3 解析結果の活用

困難の典型的なパターンが蓄積されると，今後，学習者が同種の困難に直面した瞬間に，その行動を典型的なパターンとして検出可能になる．さらに，そのパターンに対応したアドバイスを予め用意しておけば，そのアドバイスを学習者に自動的に提示できる．学習者はアドバイスをもとに，自らその困難を解決できるようになり，学習意欲を保ち自ら問題解決に向かう態度を育むことにもなる．教員や TA も，同様の質問に時間を費やさずに済み，個々人に応じた指導に当たることが可能となる．

5.4 システムに必要な機能

本章では，5.3 で述べた手法を実現したシステム “proGrep” を提案する．proGrep に必要な機能を図 5.1 に示し，次に詳細を述べる．

- **学習履歴の収集**

  学習履歴の収集は，学習環境によって行われる．学習環境は，本来の機能（プログラムの作成，実行など）に加えて次の機能が必要である．

  - **履歴送信機能** 学習環境に行われた操作（以下 “イベント” と呼ぶ）を，解析機構（後述）へ送信する．

- **学習履歴の解析**

  すべての学習者の学習履歴を横断的に解析する必要があるため，学習履歴を一ヶ所に集める仕組みと，集まった学習履歴を解析する機構が必要となる．これを “解析機構” と呼ぶ．5.3.2 で述べた手法を実現するには，解析機構に次の機能が必要である．

  - **履歴蓄積機能** 学習環境から送られてきたイベントを，学習履歴として蓄積する．“学習履歴” とは，イベントを時系列順に並べたものである．
  - **履歴検索機能** TA・教員が，蓄積された学習履歴からある特徴をもったイベントを検索する．
  - **検索条件登録機能** TA・教員が履歴の検索を行った場合に，特に多くのイベントが見つかったような検索条件を登録する．以下では，登録された検索条件を “パターン” と呼ぶ．

- **アドバイス登録機能** 登録されたパターンによって探される困難に対して，TA が，その解決を促すようなアドバイスを作成する．

- 解析結果の活用

  学習環境は，解析機構が生成した解析結果（パターンとアドバイス）を活用して，学習者の支援を行う．学習環境は,「学習履歴の収集」で挙げた機能にさらに加えて，次の機能が必要である．

  - **パターン検出機能** 学習環境に対して行われたイベントが，あるパターンに一致しているかどうかを検出する
  - **アドバイス提示機能** パターンが検出された場合で，そのパターンに対応するアドバイスが存在する場合は，そのアドバイスを学習者に提示する．

5.5 履歴収集・活用のための事前実験

前節で述べたような，ユーザのシステムの利用履歴を収集し，そこから有用な情報（知識）を引き出し，知識を活用してユーザに支援を行う，という手法自体がどれほど有効に働くかを調べるために，事前実験を行った．

事前実験においては，全文検索システムの検索語の履歴から，同義語の辞書を自動的に作成する手法を提案し，その辞書を用いて，検索時に用いる検索式を拡張する機能を追加した．

5.5.1 実験用システム Catwalk

実験のためのシステムは “Catwalk”[34] と呼ばれる．Catwalk は，全文検索の機能と，次のような検索履歴を収集・活用する機能をもったシステムである．

- **検索履歴の収集**

  全文検索システムは，本来の機能（文書の検索）に加えて，ユーザが入力した検索語の履歴を，解析機構（後述）へ送信する機能をもつ．

- **検索履歴の作成**

  全文検索システムから送られてきた検索語の履歴を蓄積し，同義語辞書を作成するための機構が “解析機構” である．同義語辞書は，互いに同義語とみなすことができるような単語の対を集めたものである．

  同義語辞書の作成にあたり，ユーザが入力した語の共起関係（同時に入力された検索語は互いに共起するという）から，“つながり” 呼ばれる検索語間の関係を求めた．ある 2 語がつながりの強い語で，しかも同時に入力されることは少ないものを同義語とみなした [35]．

- **解析結果の活用**

  全文検索システムは，解析機構が生成した解析結果（同義語辞書）を活用して，ユーザの検索支援を行う．

  全文検索システムは，ユーザが検索文字列を入力した場合，検索文字列に含まれる任意の単語 w_1 が，同義語辞書に含まれる単語の対のどちらかに一致すると，もう片方の単語 w_2 をユーザ

に提示する．ユーザが w_2 を選ぶと，w_1 と w_2 が同義語とみなされ，それらの単語のうち，どちらか少なくとも 1 つが含まれる文書を検索するように検索条件を変化させ，再検索を行う．これを検索条件の拡張という．

5.5.2 評価

実際に運用した Catwalk は，linux-users メーリングリスト [36] のアーカイブを対象にインデックスを作成し，検索サービスを提供している．検索対象となっている文書は 80388 個，その中に含まれる相異なる単語は 123939 個である．

2002 年 2 月 2 日（サービス開始）から同年 8 月 16 日までに，92485 回の検索要求があり，この履歴（検索履歴）を用いて同義語辞書の作成を行った．同義語辞書の作成にあたり，つながりを利用した手法 [35] を用いて，検索履歴から同義語を自動的に抽出した．その結果，818 個の同義語の対が同義語辞書に収録された．この同義語辞書の作成に用いた基準の詳細は付録 A.9 に示す．

実験では次の点を評価し，履歴の収集，知識の抽出，知識の活用という手法が正しく機能していることを確認した．

- **同義語辞書の作成手法の妥当性**

  同義語辞書の作成に，検索語のつながりを利用した手法を採用した．この手法が，同義語を適切に収録することができていたかどうかを評価した．

  検索語として入力された語の中から，人手により同義語を抽出してもらった．これを “正解” とする．つながりを利用した手法においては，正解を，既存の手法 [37][38] と比べて多く抽出できることが実験により確認できた [35]．このため，つながりを利用した手法の妥当性を示すことができた．

- **同義語辞書の利用**

  この同義語辞書を，2002 年 8 月 17 日から Catwalk に登録して実際に使用した．

  これを運用した結果，2004 年 12 月 22 日までに，検索条件の拡張が 2582 回行われた．

  このことから，ユーザの利用履歴をもとに作成された同義語辞書が，検索語の拡張という機能通じて，ユーザへ還元することができたことを確認した．

Catwalk で用いた手法と，proGrep で用いる手法とは，表 5.1 のように対応する．目的こそ違うが，Catwalk が採用した，履歴の収集，知識の抽出，知識の活用という枠組みは proGrep でも流用できると考えられる．

5.6 学習支援システム proGrep の設計

proGrep の構成要素と，それらのもつ機能を説明する．図 5.2 に proGrep の構成を示す．

5.6.1 学習履歴の送信

学習環境は，学習者ごとに用意し，それぞれのプログラムの翻訳（コンパイル）・実行を支援する．同時に，学習者の操作に基づく各種イベントの記録を履歴サーバに自動的に送信する．履歴サーバに

表 5.1: Catwalk と proGrep で用いられる手法

	Catwalk	proGrep
システムの目的	全文検索	プログラミング学習
履歴	検索履歴	学習履歴
	（検索語）	（作成したプログラムの内容など）
履歴の解析から得られる知識	同義語辞書	パターン・アドバイス
履歴の解析手法	検索語の “つながり” の解析	TA による学習履歴の検索
知識の活用	検索条件の拡張	自動アドバイス

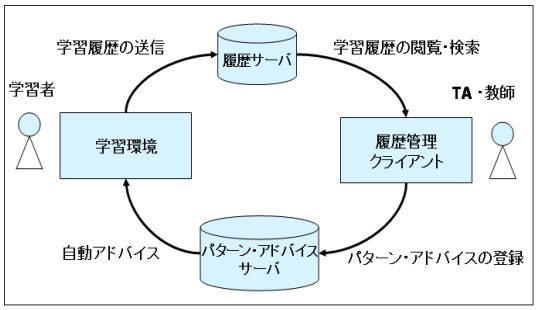


図 5.2: proGrep の構成

は，それらの記録を学習履歴として蓄積する．

イベントには次の種類がある．

- **作成イベント**

  学習者がソースファイルを書き，保存したときに発生する．保存時刻とファイルの名前・内容とを記録する．

- **翻訳イベント**

  学習者がプログラムを翻訳したときに発生する．翻訳時刻とコンパイラの出力（翻訳完了メッセージ，またはエラーメッセージ）とを記録する．

- **実行イベント**

  学習者がプログラムを実行したときに発生する．実行時刻を記録する．

5.6.2 学習履歴の検索

TA や教員は，履歴管理クライアントを通じて，履歴サーバの中に蓄積された履歴から，特定の特徴をもった “事例” を検索することができる．事例とは，ある困難を引き起こす原因となった一連のイベントのことを指す．一般的に，学習者が直面する困難には，ある単発の行動によるものもあれば，いくつかの行動の積み重ねによるものもあるので，事例は一般的に複数のイベントから成り立っているものと考える．

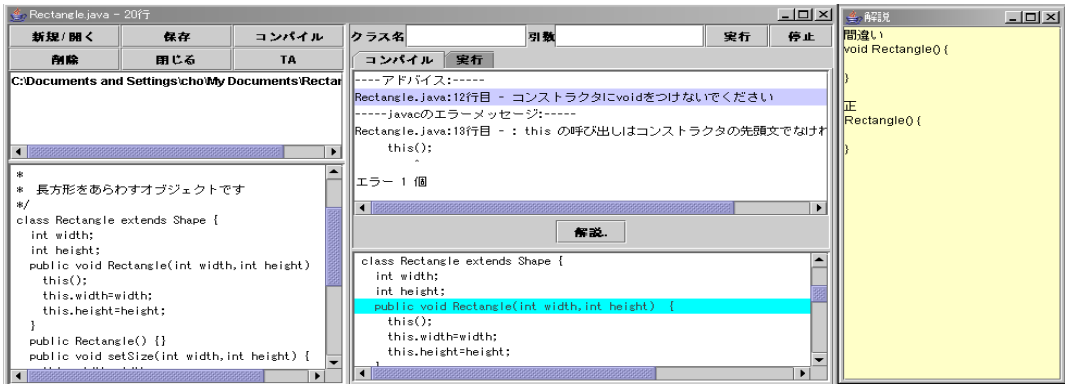


図 5.3: JavaEditor2

なお，検索条件の具体的な書き方について，proGrep 自体では規定しない．5.7 で示す実装にて一例を挙げる．

5.6.3 パターン・アドバイスの登録

TA や教員は，前述の学習履歴の検索において用いた検索条件を，履歴管理クライアントから，パターン・アドバイスサーバに登録することができる．このパターン・アドバイスサーバに登録された検索条件のことを，パターンと呼ぶ．さらに，パターンを登録する際に，そのパターンに関連するアドバイスも，パターン・アドバイスサーバに登録することもできる．

5.6.4 自動アドバイス

パターン・アドバイスサーバに登録されたパターンとアドバイスは，各学習者の学習環境に自動的にダウンロードされる．学習者が，ダウンロードされたパターンに一致する事例を学習環境上で起こすと，そのパターンに対応するアドバイスが表示される．

5.7 proGrep の実装

後述の実験で用いた，proGrep の実装を説明する．
なお，システムはすべて Java で実装した．

5.7.1 学習環境

学習環境として，“JavaEditor2” という統合開発環境を作成した．動作イメージを図 5.3 に示す．JavaEditor2 は，プログラムの作成，翻訳，実行が可能である．また，これらの操作を行うたびに，履歴サーバにイベントが送信される．

JavaEditor2 は，インストール時に，その環境固有の環境 ID を生成する．履歴サーバには，イベ

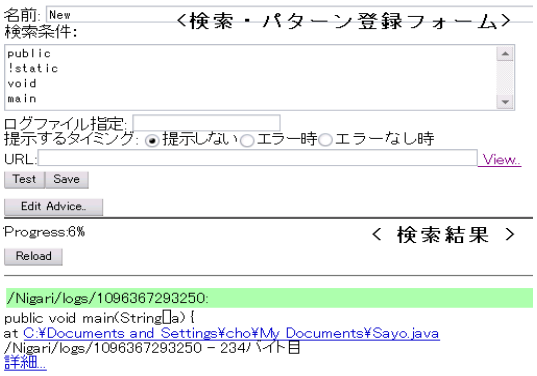


図 5.4: 履歴検索機能

ントにこの環境 ID が付加された形で送信を行う．ただし，環境 ID は，学籍番号などの個人情報とは無関係に生成される．

この他にも，JavaEditor2 にはアドバイス機能がある．JavaEditor2 を起動すると，パターン・アドバイスサーバに登録されているパターンとアドバイスを自動的にダウンロードする．最新のパターン・アドバイスを常に利用できるよう，起動した後も一定の間隔でダウンロードを自動的に行う．

JavaEditor2 は，イベントが発生するたびに，そのイベントがダウンロードされたパターンのどれかに一致する[1] かどうかを調べる．一致するパターンがあると，パターンの詳細を学習者に提示する（図 5.3 の中央部分）．さらに，このパターンのアドバイスを表示することもできる．（図 5.3 の右側のウィンドウ）

5.7.2 履歴管理クライアント

履歴管理クライアントは，履歴サーバに蓄積された履歴を検索する機能，パターンとアドバイスをパターン・アドバイスサーバに登録する機能をもつ．

教室での迅速な対応ができるように，Web アプリケーションとして実装した．これらの機能は，教室内の Web ブラウザが使える端末であれば，いつでも使うことができる．

履歴検索機能

履歴サーバに蓄積された履歴から，特定の作成イベントを検索する機能である．動作イメージを図 5.4 に示す．

現段階の実装では，作成イベントのみが検索可能である．また，単一のイベントに起因するような事例だけが検索できる．

検索条件は複数行にわたって書くことができる．各行が示す条件すべてを満たす作成イベントが探し出される．通常，1 行に書かれた文字列は「この文字列を含むソースファイル行がある」という条件を示す．また，次のような文法がある．

[1] 正確には，“パターン” に一致するのは “事例” であるが，実装では，単一のイベントに原因をもつ事例しか扱わないので，「イベントがパターンに一致する」という表現を用いる

```
public
!static
void
main
```

図 5.5: 検索条件の例（1）

```
!~;$
!~{$
!for
!while
!if
```

図 5.6: 検索条件の例（2）

- 行頭が‘!’の場合は,「それに続く文字列を含まない」という条件になる.
- 行頭が‘~’の場合は,「それに続く文字列を正規表現とみなして，そのパターンに一致する」という条件になる．‘!’と組み合わせて,「パターンに一致しない」という条件も書ける.
- 行頭に‘FILENAME’と書き，次の行から字下げを行って条件を書くと，それらの条件に一致するファイル名をもつ作成イベントに限定して検索する.
- 行頭に‘NOTEXIST’と書き，次の行から字下げを行って条件を書くと，ソースファイルに，それらの条件のどれか 1 つにでも一致する文字列が出てくるような作成イベントを，検索対象から除外する.

検索条件の例をいくつか示す.

図 5.5 は,「public と void と main がすべて含まれているが，static が含まれない行」を含むソースファイルに一致する．これは，Java のプログラムを書くときに典型的に用いられる main メソッドにおいて，static の書き忘れを修正させるのに用いる.

図 5.6 は,「セミコロンで終わっていなくて，かつ { でも終わっていなくて，さらに for も while も if も含まない行」を含むソースファイルに一致する．これは Java の式文において，セミコロンがない間違いを検出する目的の条件である.

図 5.7 は,「ファイル名に Rectangle を含むが，ソースファイル中に class Rectangle という文字列がないファイル」に一致する．これは，クラス名とソースファイルの名前が一致しないプログラムを探すのが目的である.

パターン・アドバイス登録機能

履歴検索機能で使った検索条件（パターン）と，そのアドバイスをパターン・アドバイスサーバに登録することができる.

登録の際は，次の情報を入力する必要がある.

```
FILENAME
 Rectangle
NOTEXIST
 class Rectangle
```

図 5.7: 検索条件の例（3）

- **名前**

  このパターンを表す適切な名前を指定する．この名前が学習環境においてパターンを提示する際の見出しになる．

- **提示タイミング**

  このパターンを学習者にどのように見せるかを指定する．

  - **表示なし** 学習者には提示しない．学習履歴の解析のためだけに用いる．
  - **エラー時** 翻訳上のエラーの発生時に，学習者にエラーを修正させる目的で提示する．
  - **エラーなし時** 翻訳が成功した後に提示する．翻訳上問題はないが，実行時に不具合が起りうる場合や，書き方が好ましくない場合などに，学習者に注意を促す目的で提示する．

- **アドバイス**

  このパターンのアドバイスを文章で記述する．ここに書かれた文章がそのまま学習環境を通じて，学習者に表示される．

5.8 proGrep の評価

proGrep の有効性を評価するため，実際のプログラミングの授業で proGrep を利用した．

5.8.1 評価方法

パターンの作成能力

多くのパターンが登録され，それぞれのパターンに該当するイベントが多いほど，学習者が起こしがちな典型的な事例を，システムが網羅的に把握できていることになる．

そこで，登録されたパターンの数や，代表的なパターンの詳細を示し，そのパターンの検索結果の個数を示し，典型的な事例を網羅できていることを示す．

しかし，パターンの作成は TA による手作業で行う上，パターンの記述能力にも限界があるため，すべての事例を網羅的に把握することは不可能である．このため，人手による学習履歴の調査を行い，学習履歴を順番に見ていき，特徴的な事例を見つけ出してもらった．どのパターンにも該当しない事例，すなわちシステムが把握できなかった事例についてはその原因を調べ，今後，パターンの記述能力を上げる手段を検討するための材料とする．

自動アドバイスの頻度

JavaEditor2 は，アドバイスを表示した場合に，その旨も履歴サーバに送信するようにした．これにより，実際に，学習者に対して自動的にアドバイスを行うことができたかどうかを調べられる．この自動アドバイスの履歴を用いて，学習者が直面した困難の解決を支援できたかどうかを評価する．

5.8.2 授業概要

- **科目名:** プログラミング B
- **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 言語を用いた，オブジェクト指向の入門レベルまでの機能の習得
- **期間，回数:** 2003 年 10 月 4 日〜2005 年 1 月 17 日， 11 回
- **1 回あたりの授業時間:** 2 時限（約 3 時間）
- **人数:** 約 280 人
- **クラス構成:** 3 クラスに分かれて授業．proGrep は第 1 クラスでのみ使用
- **TA:** 5 人 （第 1 クラス）
- **教科書:** 「Java 言語プログラミングレッスン（下）」[39]
- **授業形態:** 各自ノートパソコン持参，教室からネットワークに接続可能．授業 1 回ごとに，説明を受け，演習を行う．

第 1 クラスでは，学習環境である JavaEditor2 を学習者にダウンロードさせた．

TA は，学習者の事例の対応にあたり，事例の記録を行った．それとともに，履歴管理クライアントを用いて，対応した事例と同じような事例があるかないか調べ，多かったものについてはパターンとして登録する作業を行った．

5.8.3 登録されたパターン

授業中に学習者が起こした事例をもとに，TA によって作成されたパターンのうち，代表的なものを示す．

各見出しは，そのパターンの名前を表す．「意味：」は，そのパターンが適用されるような学習者の行動，「原因：」は TA の意見をもとに推測した，その行動を起こした原因を表す．

- **「NamedRectangle.java に，Rectangle クラスを書かないでください．」**

  意味：Rectangle クラスを Rectangle.java に定義する演習に続き，その子クラスである NamedRectangle クラスを NamedRectangle.java に定義する演習を行ったが，NamedRectangle.java に，NamedRectangle クラスだけでなく，改めて Rectangle クラスも定義してしまっている．本来は Rectangle.java に定義されている Rectangle クラスをそのまま使うべきである．

  原因：あるソースファイルに書かれたクラスから，他のソースファイルに書かれたクラスを参照できることを知らない．

- **「ファイル名は PBWindow.java です.」**

  意味：PBWindow というクラス（Frame クラスを継承）を定義するソースファイルの名前を，“PBWindow extends Frame.java” にしている．

  原因：ソースファイルの命名規則の誤解.

- **「displayInfo メソッドの戻り値は void です」**

  意味：displayInfo メソッドという，オブジェクトのもつ変数の内容を標準出力に表示するだけの機能をもつメソッドを作成する演習を行った．このメソッドに何らかの戻り値をもたせようとしている．

  原因：値を表示することと値を返すこととの区別がついていない.

- **「System.out.println() は return 文で返すことはできません.」**

  意味：「return System.out.println(. . .)」という書き方をした

  原因：「displayInfo メソッドの戻り値は void です」に該当する行動をした学習者が，そのプログラムを翻訳した際に「戻り値が必要」というエラーメッセージを見て，無理やり修正した.

- **「コンストラクタに void をつけないでください」**

  意味：図 5.8 のようにコンストラクタの宣言に void を書いた

  原因：何度も間違いを繰り返す学習者は，コンストラクタの書き方について不勉強．そうでなければケアレスミス.

- **「i の範囲に注意してください」**

  意味：図 5.9 のように，配列を後ろの要素から前の要素へ辿るループの書き方を間違えている．

  原因：何度も間違いを繰り返す学習者は，配列の扱いを習熟していないか，デバッグの方法がわかっていない．そうでなければケアレスミス.

- **「Copy の課題で StringTokenizer を使っている例」**

  （このパターンは学習者には提示しなかった）

  意味：「引数が 2 つ与えられ，第 1 引数で指定されたファイルの内容を，第 2 引数で指定されたファイルにそのままコピーする」プログラムを作らせたところ，使う必要のない StringTokenizer クラスを利用した.

  原因：ヒントとして，以前に演習した「ファイルを読み込み，それを StringTokenizer を使って単語ごとに切り分けて表示する」プログラムを示したが,「ファイルを読み込む」部分だけを利用すればいいことに気づかず，ヒントのプログラムをまるごとコピーした.

これらのパターンが，2004 年 12 月 30 日現在までの学習履歴中にどのくらい出現したかを表 5.2 に示す.

この表のうち,「人数」は，このパターンに該当するイベントを 1 回でも起こした学習者の人数,「合計回数」は，このパターンに該当するイベントが，全学習者の履歴で起きた回数の合計,「最大回数」は，このパターンに該当するイベントを最も多く起こした学習者の，起こした回数を表す.

なお，全学習者の学習履歴中に含まれる作成イベントの数は，約 50000 件，翻訳イベントのうち，エラーの起きたもの（エラーイベント）は約 23000 件であった.

```
class Rectangle {
  public void Rectangle(int w,int h){
    :
  }
  :
}
```

図 5.8: 「コンストラクタに void をつけないでください」に該当する事例

```
for (int i=words.size() ; i>=0 ;i--) {
  System.out.println(words.get(i));
}
```

図 5.9: 「i の範囲に注意してください」に該当する事例

表 5.2: パターンの出現回数

名前	人数	合計回数	最大回数
i の範囲に注意してください	43	98	8
コンストラクタに void をつけないでください	34	101	19
Copy の課題で StringTokenizer を使っている例	33	212	30
displayInfo メソッドの戻り値は void です	31	179	20
NamedRectangle.java に，Rectangle クラスを書かないでください.	17	72	12
System.out.println() は return 文で返すことはできません.	9	26	7
ファイル名は PBWindow.java です.	7	18	4

この結果から，proGrep によって，TA が対応した事例と同様の事例に，他の学習者も直面しているかどうかを検索できるようになった．また，実際に複数の学習者が，同様の事例に直面していたことが確認できるようになった．

5.8.4 パターンに登録できなかった事例

事例の中で，多くの学習者がたびたび起こしていたにもかかわらず，該当するパターンを登録できなかったものを見つけ出すため，次の要領で，人手により，学習履歴の中から特徴的な事例を抽出してもらった．

- **調査対象：** 履歴サーバに蓄積された学習者 66 人分の学習履歴．
  - 作成イベント数： 約 11000 件
  - エラーイベントの数： 約 5000 件
- **調査時期：** 2004 年 11 月中旬
- **事例の抽出方法：** 学習履歴中のエラーイベントの直前に起きた作成イベントの内容（つまり，翻訳エラーを引き起こしたであろうプログラム）を見て，登録済みのパターンに該当するものや，スペルミスなどの単純な間違い，あるいはすでにこの作業で抽出したものと同種の事例を除いた事例を “特徴的な事例” として抽出した．

この結果，特徴的な事例が 66 個[2] 抽出された．一部を次に示す．

図 5.10 の例では，本来は IOException の例外処理を先に書き，Exception の例外処理を後に書かなければならない．また，図 5.11 の例では，return 文がメソッドの定義部分の外に書かれてしまっている．そして，図 5.12 の例では，コンストラクタと draw メソッドに書くべき内容を，それぞれ逆の場所に書いてしまっている．さらに，図 5.13 の例では，メソッドの中にメソッドの定義を書いてしまっている．

これらはいずれも，1 行を単独で見ても間違いはわからず，前後にどんな行があるか，あるいは，どんなブロック（メソッドの定義や，クラスの定義）の内部にその行が書かれたか，といった条件が書けないと検索できない．

図 5.14 の例では，同じ仮引数をもつコンストラクタを 2 つ定義している．これは，あるパターンが現れた行がいくつあったか，という条件が書けないと検索できない．

図 5.15 の例では，コンストラクタで宣言された仮引数を，他のメソッドで使おうとしている．これは，ある行に出現した部分文字列が，他の行で出現しているか否かを調べるような条件が書けないと検索できない．

今後，検索条件の記述能力を高め，様々な状況に対応できる柔軟なパターンが書けるように今後工夫する必要がある．

5.8.5 自動アドバイスの頻度

登録されたパターンに該当するイベントを学習者が起こしていたとしても，そのイベントが起きた時期が，パターンが登録される以前であれば，アドバイスは提示されない．このため実際にアドバ

[2] 履歴を調べる対象となった学習者の人数と同じであるが，これは偶然

```
import java.io.*;
class Kadai21{
  public static void main(String[] args){
    try{
      throw new IOException();
    } catch (Exception e) {
      System.out.println("Exception:"+e);
    } catch (IOException e){
      System.out.println("IOException:"+e);
    }
  }
}
```

図 5.10: 特徴的な事例（1）

```
public class Elephant {
  String name;
  double weight;
  public Elephant(String name,
    double weight){
    this.name=name;
    this.weight=weight;
  }
  int displayInfo(){
    System.out.println("名前："+name
      +"体重:"+weight);
  }return displayInfo;
}
```

図 5.11: 特徴的な事例（2）

```
class Oval {
  int width,height;
  public Oval(int width,int height) {
    window.drawOval (x,y,width,height);
  }
  public void draw(PBWindow window
    ,int x,int y){
    this.width=width;
    this.height=height;
  }
}
```

図 5.12: 特徴的な事例（3）

```
public class RectTest {
  public static void main(String[]a) {
    Rectangle r=new Rectangle(200,100);
    void RectTest(int w,int h){
      width=w;
      height=h;
    }
    System.out.println("面
積:"+r.getArea());
  }
}
```

図 5.13: 特徴的な事例（4）

```
class NamedRectangle extends Rectangle
{
  String neme;
  NamedRectangle(String name) {
      neme = "NO NAME";
  }
  NamedRectangle(String name) {
      this.name = name;
  }
}
```

図 5.14: 特徴的な事例（5）

```
public class Player {
  String name;
  double average;
  int age;
  Player (String n,int a,int daritu){
    this.age=a;
    this.name=n;
    this.average=daritu;
  }
  public void showStatus{

System.out.println(n+''+a+''+daritu);
  }
}
```

図 5.15: 特徴的な事例（6）

表 5.3: アドバイス表示回数

名前	人数	合計回数	最大回数
コンストラクタに void をつけないでください	21	33	4
i の範囲に注意してください	15	64	13
System.out.println() は return 文で返すことはできません.	7	23	8
ファイル名は PBWindow.java です.	4	8	2
displayInfo メソッドの戻り値は void です	3	7	3
NamedRectangle.java に，Rectangle クラスを書かないでください.	3	9	4

イスが提示された回数は表 5.2 の値より少なくなる．表 5.3 に，各パターンに関連するアドバイスが実際に学習者に提示された回数を示す.（「Copy の課題で StringTokenizer を使っている例」は，学習者に提示しないパターンなので提示は 0 回）

これらのパターンのほとんどは，ある特定の日の授業内容に関連するものであるので，短い時間にまとまって，同様な事例が発生している．しかしながら，システムを Web アプリケーションとして実装し，授業中に事例の検索やパターンの登録が速やかにできるようにしたため，何件かの事例には自動的にアドバイスを提示することができた.

5.9 関連研究

5.9.1 学習履歴の活用

学習履歴の利用方法として代表的なものに，項目応答理論 [40] が挙げられる．これは，試験の難易度と学習者の能力の関係を表すモデルと，学習者の正解・不正解に関する履歴を利用して，学習者の能力を推定するための手法である.

しかし，そこでの学習履歴とは，試験（授業中の小テストなども含む）の解答状況や，正解，不正解といった情報だけに限られ，試験と直接関係ない情報は扱われていない.

学習履歴として，単なる試験の解答結果にとどまらず，学習にかかる時間や，学習者の表情などといった，多様なデータも学習履歴として活用することの重要性が指摘され始めている [41] が，具体的な手法はあまり提案されていない.

試験以外の学習履歴として，学習者が作成したプログラムを「ポートフォリオ」という形で提出させる試みが行われている [42][43]．しかし，これらの仕組みが扱うのは，人に見てもらう形に整えたプログラムだけであるため，それを作り上げるまでの道程を捉えることは難しい．また量的に，半期に 5 個程度のプログラムを提出させるに留まっている.

本研究では，試験にとどまらず，学習者のあらゆる行動が，その学習者の能力を表しうる，という観点で，なるべくたくさんの履歴を集め，活用する手法を提案している．学習者のプログラム全体を逐一すべて記録し，それらを利用した支援を行うという試みは，調査した範囲ではこれまで行われていない.

5.9.2 プログラム学習のアドバイス

プログラミングの実習を行う上で，典型的な間違いを直す支援を行う環境はすでに研究が行われている．CAP[44] や Style[45] は，Pascal のプログラムの文法的，論理的エラーを，学習者にとってわかりやすい形で指摘するシステムである．Java の典型的な間違いを検出するシステムとしては Findbugs[46] が挙げられる．

これらのシステムは，Pascal など，学習の対象となっている言語の文法や動作に関する知識を予め組み込んでおき，それをもとに間違いを指摘する．学習履歴を用いているわけではないので，学習者がどのような間違いをしたかによって，柔軟な支援を行うのが難しい．また，論理的な間違いの検出ができるといっても，特定の課題に対する “考え方” の間違いなどは指摘できない．例えば，実験で検出された「displayInfo の戻り値は void です」や「Copy の課題で StringTokenizer を使っている例」といった事例は検出することはできない．

さらに，proGrep は，間違いを指摘する対象をプログラム本体に限定していない（実装上はプログラムの検索しか実現しなかったが）．コンパイルや，実行など，学習環境の操作にかかわるイベントをも検索可能なように設計してある．これにより，操作ミスによってプログラムが正しく動かない，といった事態にも対処できる作りになっている．

5.10 まとめ

本章では，プログラミングの実習を行う際に直面する，様々な困難の解決を支援するための手法とそのためのシステム proGrep を提案した．

本手法の特色は，学習履歴や，TA の活動履歴などを収集，解析し，そこから困難の解決に役立つ情報を引き出す点にある．

proGrep は，学習者の作成したすべてのプログラムを記録し，TA の活動履歴をもとに学習者のプログラムを検索する機能をもつ．プログラムを検索した際の検索条件をパターンとして登録すると，学習者がそのパターンと同じ状況に遭遇した場合に，自動的にアドバイスが提示できる．

実験では，実際の授業においてパターンの登録が行われ，それらのパターンが多くの学習者で共通に現れるものであることを確認し，さらにそのパターンにあてはまる状況に遭遇した学習者に対するアドバイスも可能になった．

これにより，学習環境がもとで引き起こす様々な困難に，学習者自身が対処できるような支援を行うことが可能となり，学習の行き詰まりや，TA に頼りきりの習慣を取り除き，問題解決の達成感を味わわせ，ひいてはプログラミングの楽しさを増長させるような環境づくりができたといえる．

第6章　結論

本論文では，CS におけるプログラミング学習のための支援環境として，楽しく学習でき，しかも CS に必要なプログラミングの知識を身につけられる学習環境を提案し，授業を通じた評価を通じて有効性を示した．

2 章では，初心者向きプログラミング言語 “Nigari” と，その環境である “Nigari System” について提案し，3 章で実験により評価を行った．実験の場となった授業では，本来 Java を用いた実習を行うが，導入部に Nigari System を用いた．実験では，オブジェクトの可視化機能や，簡素な言語によって学習者が楽しく，挫折することなく学習できたことを確認した．また，試験結果や，授業中の小テストなどから，Nigari System を使って，Java の基本的なプログラムの書き方を学習できることを示した．これによって，Nigari System は実用言語を習うための導入に有用であることを示した．

4 章では，教育用言語環境である Nigari System から，実用言語である Java へ，戸惑いを感じさせることなく移行できるようなコースデザインを提案し，そのコースデザインに適した学習環境 “N-Java” を提案し，授業による実験を通じて評価を行った．N-Java を利用した学習では，Nigari System，Java という 2 つの学習環境を用いて学習する場合に比べ，コースデザイン上の無駄や，環境の違いによる戸惑いを取り除くことができ，学習者の興味を持続させることができた． また，試験結果から，Java の基本的なプログラムの書き方についての学習効果については，ほぼ問題ないことを確認した．

5 章では，学習環境を使う上で直面するであろう困難（事例）の解決を補助し，プログラミング学習の挫折を防ぐ仕組みとして，学習履歴活用システム proGrep を提案した．このシステムは，学習者の用いる学習環境の使用履歴を収集・解析し，多くの学習者が直面している困難に対して自動的にアドバイスを提示する仕組みをもつ．授業で proGrep を用いた結果から，proGrep の有効性を示した．実験では，複数の学習者の学習履歴の中から，共通する事例を取り出すことができた．さらに，それらの事例に対するアドバイスの作成を迅速に行うことができ，いくつかの事例に対しては，学習者に実際にアドバイスを提示することができた．学習者が直面した事例に対する支援ができるようになることで，学習者の学習意欲の低下を防ぐことが可能となった．

本研究は，“楽しさ” に重点を置いたプログラミング学習支援環境を提案した．すぐに実行することができ，魅力的な出力が得られるような言語環境により，楽しさを増長させることができた．また，プログラミング言語に含まれるおまじないや，学習環境が引き起こす解決不能な困難などを取り除き，楽しさの増長を阻害する要因をなくすことが可能になった．さらに，楽しいだけでなく，CS におけるプログラミング教育に必要な素養を身につけるために，実用言語への応用を考慮した新しいコースデザインを提案し，楽しさと学習効果を両立させることも可能になった．

今回の研究では，楽しさを持続させるための工夫として，プログラムがすぐに動いたり，プログラムが可視化されたりすることを取り入れたが，筆者が実際の授業の現場にて観察した限り，学習者が楽しく取り組んでいたものはこれにとどまらなかった．例えば，お互いのプログラムを批評したり，プログラムを協力して作成したりする課題などは，多くの学習者が積極的に取り組んでいた．プログラムは他のユーザに使ってもらうことによりその価値が発揮されるものであるから，他の学習者と

の協調したプログラムの作成は大事な学習項目であるといえる．こうした，学習者が楽しいと感じ，しかも学習効果の高い要素を授業に多く取り込み，その授業内容に最適な環境も今後とも提供していきたい．

付録

A.1 EBNF による、Nigari の言語仕様

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 Nigari System 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 2003 年度 プログラミング A 教材

A.3.1 授業ページの一例

(5 月 19 日の「if 文」の項目で使用)

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

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。

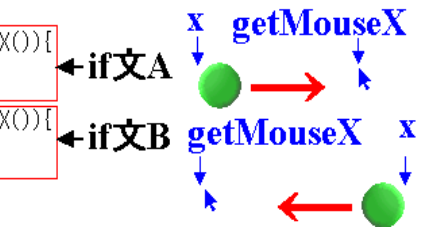
if という新しいページを作り、プログラムを実行してみましょう。

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

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。

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



(次ページへ)

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

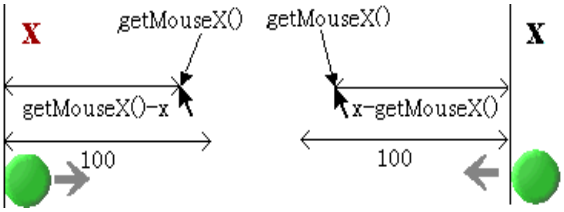
「オブジェクトとマウスカーソルの横方向の距離が 100 以上あるときは動かない」⇒「オブジェクトとマウスカーソルの横方向の距離が 100 未満のときだけ動く」

「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

「getMouseX()-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

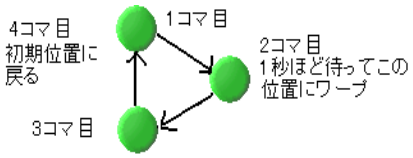


演習問題

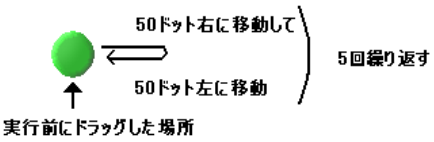
逆に、マウスカーソルとオブジェクトが 100 以上離れていた場合だけ、動くようにするプログラムを書きましょう。

A.3.2　練習問題の一例

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．

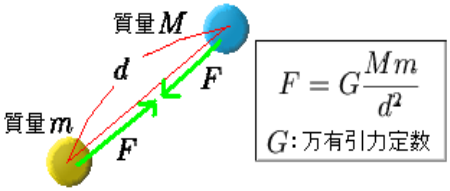


5/19（while 文） 次の図のように，左右に 5 往復するオブジェクトを作りましょう．



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．



A.4　2003 年度 プログラミング A 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（第 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ことが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)

A.5 類推テスト

問題 8-2 : 何が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: printResult メソッドを呼び出すと，何が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;
}
```

A.6　2004 年度プログラミング A プレテスト

このテストに出てくるプログラムは、Java、C、C++いずれでも動作するようになっています。また、変数宣言は適切になされているものとします。

問題 1

次の文が正しいか、間違っているかを○×で答えなさい。

プログラムが 2 行以上ある場合、すべての行が同時に実行される。
選択肢:　○　×

x=200;　という代入を行うことによって変数 x の値が 200 に等しくなる。
選択肢:　○　×

x=x+100;　という代入を行うことによって、x と x+100 が等しくなる。つまり 0=100 になる。
選択肢:　○　×

問題 2

2-1．次のプログラムを実行した場合の r の値はどれか

```
x=20;
y=5;
r=x+y*x-y;
```

選択肢:　45　50　115　375

2-1．次のプログラムを実行した場合の r の値はどれか

```
x=30;
y=2+x/10;
x=x+5;
r=x/y;
```

選択肢:　5　7　30　35

問題 3

3-1．次のプログラムを実行した場合の r の値はどれか

```
r=15;
if (r<5) {
   r=r*5;
} else {
   r=r/5;
}
```

選択肢: 3 15 25 75

3-2．次のプログラムを実行した場合の r の値はどれか

```
r=1;
if (r<5) {
   r=r*5;
   if (r<5) {
      r=r*5;
   } else {
      r=r-1;
   }
} else {
   r=r-1;
   if (r>0) {
      r=r*5;
   } else {
      r=r+1;
   }
}
```

選択肢: 4 5 20 25

問題 4

4-1．次のプログラムを実行した場合の r の値はどれか

```
x=5;
r=3;
while(x<9) {
   r=r+x;
   x=x+1;
}
```

選択肢: 3 8 29 38

4-2．次のプログラムを実行した場合の r の値はどれか

```
x=0;
r=0;
while (x<5) {
  if (x<3) {
     r=r+1;
  }
  x=x+1;
}
```

選択肢: 2　3　4　5

A.7　2004 年度プログラミング A 共通試験問題と正解

【】で囲まれた部分は解答欄およびその名前（ひらがな），その正解を表す.

1．配列（int[]）a にクラスの各人がとった試験の点数（0〜100 の整数）が入っている。60 点〜69 点のとき評価はDとなるとして，評価が D となる学生の人数を変数 count に求めるプログラムである。空欄【あ】を埋めるのに正しいものすべてを下の解答群から選び，その左端の欄に○印をつけよ。

```
count= 0;
for(int i=0; i!=a.length; i++){
   if( 【あ】 ) count++;
}
```

解答群

【　】 60<a[i] && a[i]<70
【○】 !(a[i]>=70 || a[i]<60)
【○】 59<a[i] && a[i]<70
【　】 !(69<a[i]) || !(a[i]<60)
【○】 60<=a[i] && !(a[i]>=70)

2．下の解答群のプログラムをつぎのプログラム A と比較し，プログラム A と同じ出力が得られるものすべてを下の解答群から選び，その左端の欄に○印をつけよ。

プログラム A

```
int i= 0;
while( i<10 ){
    i++;
    System.out.println(i);
}
```

解答群

【○】
```
int i= 1;
while( i<=10 ){
    System.out.println(i);
    i++;
}
```

【○】
```
int i= 1;
do{
    System.out.println(i);
    i++;
}while( i!=11 );
```

【　】
```
int i= 0;
while( true ){
    System.out.println(i+1);
    if( i==10 ) break;
    i++;
}
```

【　】
```
for(int i= 0; i<10; i++){
    System.out.println(i);
}
```

【○】
```
for(int i= 10; i>0; i--){
    System.out.println(11-i);
}
```

3．int f(int x, int y) というメソッドには，関数としてつぎの性質があるとする。

```
f(a, b) < f(a, b')     (b<b')
f(a, b) > f(a', b)     (a<a')
```

与えられた整数値 k と N (N>0) に対して，k=f(a,b) となる整数の組 (a,b) (0<=a,b<N) が存

在するかどうかを調べるプログラム B を作った。つぎの文およびプログラムの中の空欄【あ】・【い】を埋めるのに最も適当なものを，それぞれ下の解答群から選び，その左端の欄に○印をつけよ。
プログラム B 中の while 文の反復回数は，最大【あ：2*N】回である。

プログラム B （アルゴリズム部分だけを示す）

```
int a= 0; int b= 0;
while( a<N && b<N ){
    if( f(a,b)==k )  break;
    if( f(a,b)>k ) a++;  else b++;
}
if( 【い: !(a<N && b<N)】)
    System.out.println("f(a,b)!=k (0<=a,b<N)");
else
    System.out.println("f("+a+","+b+")="+k);
```

【あ】の解答群
【 】 log N
【 】 N
【○】 2*N
【 】 N*N

【い】の解答群
【 】 k==f(a,b)
【△】 k!=f(a,b)　※部分点
【 】 N=>a || N=>b
【○】 !(a<N && b<N)

４．2，3，5 のべき乗を，プログラム C を作って計算させてみた。その出力結果は，結果 C のようであった。プログラム C と結果 C を見て，後ろに続く文章の空欄【あ】～【お】に埋めるのに適切な整数値を解答欄に記せ。なお，文章中の x^y は x の y 乗を表す。

プログラム C

```
class IntOverflow{
    public static void main(String[] args){
        int p2, p3, p5;
        p2= p3= p5= 1;
        for(int i= 0; i!=35; i++){
            pn(i, 3); pn(p2,12); pn(p3,12); pn(p5,12);
            System.out.println();
            p2*= 2; p3*= 3; p5*= 5;
```

```
        }
    }
    static void pn(int n, int w){
        String str= Integer.toString(n);
        for(int i= str.length(); i<w; i++)
            System.out.print(" ");
        System.out.print(str);
    }
}
```

結果 C

0	1	1	1
1	2	3	5
2	4	9	25
3	8	27	125
4	16	81	625
5	32	243	3125
6	64	729	15625
7	128	2187	78125
8	256	6561	390625
9	512	19683	1953125
10	1024	59049	9765625
11	2048	177147	48828125
12	4096	531441	244140625
13	8192	1594323	1220703125
14	16384	4782969	1808548329
15	32768	14348907	452807053
16	65536	43046721	-2030932031
17	131072	129140163	-1564725563
18	262144	387420489	766306777
19	524288	1162261467	-463433411
20	1048576	-808182895	1977800241
21	2097152	1870418611	1299066613
22	4194304	1316288537	-2094601527
23	8388608	-346101685	-1883073043
24	16777216	-1038305055	-825430623
25	33554432	1180052131	167814181
26	67108864	-754810903	839070905
27	134217728	2030534587	-99612771
28	268435456	1796636465	-498063855
29	536870912	1094942099	1804648021
30	1073741824	-1010140999	433305513

```
 31 -2147483648  1264544299 -2128439731
 32           0  -501334399 -2052264063
 33           0 -1504003197 -1671385723
 34           0  -217042295   233005977
```

正整数 n に対して，つぎの関係を満たすビット列 bm-1 bm-2...b0 (m > 1, bm-1=1) を n の二進表現といい，m を n の二進表現の長さという。

```
n = Σ bi*2^i
   0<=i<m
```

一方，正負の整数値を表す int 型では 32 ビットのビット列を使い，ビット列 b31 b30...b1 b0 でつぎのとおりの整数 n を表す。

```
n = - b31*2^31 + Σ bi*2^i
               0<=i<31
```

結果 C 中の 2^i の計算結果から，2^31 = 【あ:2147483648】であることがわかる。さて， 2^i を表すビット列の中に 1 は【い：1】個ある。 2^(i+1) を表すビット列は， 2^i を表すビット列の左端のビットを取り除き右端に 0 を追加したものになる。毎回，0 の個数が 1 ずつ増えていくから， 2^32 以降の計算結果はすべて 0 ばかりからなるビット列になってしまう。
結果 C 中の 3^20 の計算結果が -808182895 となっているが， 3^20 の正確な値は【う:3486784401】であり,【う:3486784401】=(-808182895) + 2^32 となっている。 3^21 の計算結果は 1870418611 となっているが，正確には 3^21 = 1870418611 + 【え：2】* 2^32 である。
5^14 の計算結果が 1808548329 となっているが，正確には， 5^14 = 1808548329 + 2^32 である。同様に， 5^15 = 452807053 + 【お：7】* 2^32 である。

５．今学期に受講した必修科目の成績（ A，B，C，D，F ）から評定平均値を計算したい。評定平均値は，それぞれの科目の成績を数値化した値の平均をいう。成績の数値化は，つぎの通りとする。

```
A 9
B 8
C 7
D 6
F 0
```

プログラム D は，つぎのようにコマンドラインの引数に成績を並べて与えると，その評定平均を小数点以下まで出力する形に仕上げる。成績は，大文字で入力しても小文字で入力してもよいようにしておく。

```
[user@myPC ~] java Average C b b A c
```

7.8

プログラム D を完成するのに，空欄【あ】〜【お】を埋めるに適切な語句を解答欄に記せ。

プログラム D

```
class 【あ：Average】{
    public static void main(String[] args){
        int kamokusu= 0;
        int sotokuten= 0;
        for(int i= 0; i!= args.length; i++){
            switch( args[i].charAt(0) ){
            case 'A': case 'a':
                sotokuten【い：+=】 9; 【う：break】;
            case 'B': case 'b':
                sotokuten【い：+=】 8; 【う：break】;
            case 'C': case 'c':
                sotokuten【い：+=】 7; 【う：break】;
            case 'D': case 'd':
                sotokuten【い：+=】 6; 【う：break】;
            case 'F': case 'f':
            default:
                sotokuten【い：+= または =sotokuten+】 0;
           }
            kamokusu【え：++ または  +=1 または  =kamokusu+1】;
        }
        System.out.println(【お:(double)sotokuten または  (sotokuten+0.0)】 / kamokusu);
    }
}
```

６．簡単なくじを企画した。くじを引くたびに，1000 円を拠出しなければならない。一人何回くじを引いてもよい。それぞれのくじには，０〜1 の実数値が振られている。くじが終わったとき，くじを引いた人の「得点」は，自分が引いたくじに振られていた値の最大値とする。拠出金をこの「得点」で比例配分して賞金とする。ただし，100 円未満の端数は寄付してもらい，集めて UNICEF に送ることになっている。
そこで，このくじを実施し，各人の賞金と，UNICEF に送ることになる寄付金の総額とを計算するプログラム E を作った。プログラム E は，つぎのようにコマンドラインから実行し，くじを引いた人の名前をくじを引いた順に引数に並べて与える。

```
[user@myPC ~] java Lotto ito ue ito sato ue kato yamada
ito: amount 2600
kato: amount 2200
```

```
ue: amount 1200
sato: amount 800
yamada: amount 0
donation: amount 200
```

空欄【あ】〜【こ】に適切な語句を埋めてプログラム E を完成せよ。
プログラム E では，くじを引いた人の名前とその得点とを記録しておくためのクラス Record を定義して使っている。
まず，コマンドラインの引数に与えられた名前の順に取り出して，くじを引いて 0〜1 の値を得，その名前と値とを記録しておく。すでに出てきた名前なら，登録してある記録の値と比較して，大きい方の値を記録として残す。初めての名前なら，新たに記録を付け加える。こうした記録は，配列 result を使って行う。変数 applicants は，記録されている人数を示している。
なお， String 型のデータである二つの名前が等しいかどうかを判定するには， String クラスのメソッド equals を使う必要があることに注意する。また，プログラム E では， 0 〜 1 の乱数を得るのに Math.random() ではなく， Random クラスのオブジェクトを作ってその nextDouble() メソッドを呼び出している。これは，実行日時に応じて別の乱数列が得られるようにするためである。
すべての入力を処理し終わったら，記録されている参加者すべてについて，その得点の総和を totalScore に求める。同時に，記録を並べ替えて，得点の高い参加者の順に result[0], result[1], ... と並ぶようにする。
最後に，得点の高い人から順に，賞金額を計算し，その名前と一緒に賞金額を出力していく。賞金額は，拠出金総額に，その人の得点と全員の得点総計との比を乗じて円単位で切り捨てて計算上の金額を求める。100 円未満の端数は寄付してもらう約束だから，100 で割った整数商を求めて 100 倍すれば，寄付後の賞金額を求めることができる。終わりに拠出金総額から賞金総額を引いて寄付金額を算出して出力する。

プログラム E

```
import java.util.*;
class Record{
    String name;
    double score;
    Record(String name, double score){
        【あ：this.name】= name;
        【い：this.score】= score;
    }
}

class Lotto{
    public static void main(String[] args){
        Record[] result= new Record[args.length];
        int applicants= 0;
        Random rand= new Random();
```

```
        for(int i= 0; i!=args.length; i++){
            String key= args[i];
            double score= rand.nextDouble();
            result[applicants]= 【う：new Record(key, score)】;
            int k= 0;
            while( !key.equals(result[k].name) ) k++;
            if( k!=applicants ){
                if( score>result[k].score ) 【え：result[k].score= score】;
            }else{
                 【お：applicants++】;
            }
        }
        double totalScore= 0.0;
        for(int i= 0; i!= applicants; i++){
            totalScore【か：+= result[i].score】;
            for(int k= i; 【き:k!=0】  && result[k-1].score<result[k].score; k--){
                Record w= result[k-1];
                result[k-1]= result[k]; result[k]= w;
            }
        }

        int totalFund= args.length*1000;
        int totalAmount= 0;
        for(int i= 0; 【く：i!= applicants】; i++){
            int amount= (int)(totalFund*result[i].score/totalScore);
            amount= (amount/100)*100;
            System.out.println(result[i].name+": amount "+amount);
             【け：totalAmount+= amount】;
        }
        int totalDonation= 【こ：totalFund-totalAmount】;
        System.out.println("donation: amount "+totalDonation);
    }
}
```

７．メソッド longestEquals は，引数に与えられた double 型データの配列を，同じ値が続いている部分部分に分けていったとき，もっとも長い部分（複数あるときはその最も若い添字から始まるもの）を見つけ出して，その部分が始まる添字を結果として返して来る。そのアルゴリズムのあらましはつぎの通りである。

同じ値が続いている部分の左端の添字を h ，右端の添字を t で表すことにする。また，それまでに見つけたもっとも長い部分の長さを m に記録し，その左端の添字を mh に記録しておくことにする。 t を 1 ずつ増やしながら，これらの値を適切に調整していく。a[h], ..., a[t] がすべて同

じなら，h を変更する必要はない。そうでなければ， h を t に重ねればよい。その上で，a[t] を右端として同じ値が並ぶ部分の長さ t-h+1 が m を超えていれば， m と mh とを適切な値に書き換えればよい。
下に示す longestEquals の static メソッドとしての宣言には誤りが 3ヶ所ある。誤りを見つけ出して訂正を施したい。

```
 1: static int longestEquals(double a){
 2:     int m= 0, mh= 0;
 3:     int h= 0;
 4:     for(int t= 0; t<=a.length; t++){
 5:         if( a[h]!=a[t] ) h= t;
 6:         if( m < t-h+1 ){
 7:             m= t-h+1; mh= h;
 8:         },
 9:     }
10:     return mh;
11: }
```

誤り個所を指摘するとともに，その誤りに対する訂正内容を記せ.

解答
1 行目 double a -> double[]a
4 行目 t<=a.length -> t<a.length
8 行目 , を消去

A.8 2004 年度 プログラミング A 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？
- 今日の授業の難易度は？
- 今日の授業は楽しかったか ？

- 今日の問題は自力で解いたか？
- 自由記入欄 (F)

能力調査（4/19）：

- 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/F)
- パソコンを自宅で使っていますか (Y/F)
- 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/F)
- 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)
- CS 学科の科目で，一番面白いと思った科目は何ですか (S/F)
- 一番興味のある分野はどれですか (S/F)
- 将来の進路は (S/F)
  - 大学院に進学したい
  - 大学院に進学せず就職したい
  - その他・未定
- 将来どんな仕事をしたいと思いますか (M/F)
  - 実際にプログラミングを行うような仕事をしたい
  - 実際にプログラミングはしないまでも、プログラムの設計や管理など コンピュータのソフトウェアに関連する仕事をしたい
  - コンピュータのハードウェアに関連する仕事がしたい
  - コンピュータとあまり関係ない仕事をしたい
- その他，ご自由に記入してください (F)

理解度に関するアンケート (06/09)：
つぎの概念についてどの程度理解しているか (S)

- 変数
- 型
- while 文
- if 文
- インデント
- 変数の宣言

- 配列
- オブジェクト

最終アンケート（第 1 クラスのみ）(7/5)：

- レベル 1 から レベル 2(変数宣言が必要) へは、スムーズに移行できましたか (S/F)
- レベル 2 から レベル 3(メソッドの宣言が必要) へは、スムーズに移行できましたか (S/F)
- N-Java で面白い・便利だと思った機能は (S/F)
- N-Java を使うことで Java への理解が深まったと思いますか (S/F)
- 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/F)
- BBS を使いましたか (S/F)
- BBS を読んだ、書いた方は、BBS は役に立ちましたか (S/F)
- 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。 2つまで挙げてください (M/F)
- 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) (7/05)：

- 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/F)
- 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ことができるようになりましたか (S/F)
- 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/F)
- 各項目について理解度を答えてください (S)
  - 変数への代入／変数を使った計算
  - 変数の型
  - if 文 (条件に従って処理内容を変える)
  - while 文 (条件によって繰返しを行う)

- for 文 (while 文との使い分け)
- 配列
- クラスとオブジェクト
- メソッドを使う
- メソッドを作る

- 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/F)
- その他、自由にご意見をご記入ください。(F)

A.9 同義語辞書の作成に用いた評価基準

5.5.2 にて，Catwalk の検索履歴から同義語辞書を作成する際に用いた基準を示す.

Catwalk における “検索履歴” は，“問合わせ” の列からなり，問合わせは，“単語” の集合である [35].

ある単語の対 (w_1, w_2) が同義語であるかどうかを判定するための尺度 $S(w_1, w_2)$ を次の式で与える.

$$S(w_1, w_2) := S_1(w_1, w_2)(1 + \alpha S_2(w_1, w_2)) \tag{A.1}$$

ただし，$S_1(w_1, w_2)$ を，次の式で与える.

$$S_1(w_1, w_2) = \frac{M df(w_1 \cap w_2) - df(w_1)df(w_2)}{\sqrt{(M - df(w_1))df(w_2)(M - df(w_2))df(w_1)}} \tag{A.2}$$

M は Catwalk に収録された文書総数 (=80388)，$df(w)$ は，単語 w が出現する文書数，$df(w_1 \cap w_2)$ は，単語 w_1, w_2 がともに出現する文書数を，それぞれ表す.

さらに，$S_2(w_1, w_2)$ を，次の式で与える.

$$S_2(w_1, w_2) := |K(w_1, w_2)| - \beta|E(w_1, w_2)| \tag{A.3}$$

ここで $|E(w_1, w_2)|$ は，検索履歴中，w_1, w_2 をともに含んでいた問合わせの数を表し，$|K(w_1, w_2)|$ は，w_1, w_2 のつながりの数 [35] を表す.

また，式 A.1 において，$\alpha = 1$ とし，式 A.3 において，$\beta = 3$ とした.

単語の対 (w_1, w_2) について，$S(w_1, w_2)$ の値が 0.5 以上であるようなもの 818 個を，同義語辞書に収録した．(0.5 という数値には特に意味はない)

参考文献

[1] 兼宗進. 教育利用を目的としたオブジェクト指向言語の研究. 博士論文, 筑波大学大学院, 2003.

[2] 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.

[3] Computing curricula 2001. *J. Educ. Resour. Comput.*, Vol. 1, No. 3es, p. 1, 2001.

[4] Computing curricula 1991. *Commun. ACM*, Vol. 34, No. 6, pp. 68–84, 1991.

[5] John T. Minor and Laxmi P. Gewali. Pedagogical issues in programming languages. Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC '04), 2004.

[6] Suzanne Pawlan Levy. Computer language usage in CS1: survey results. *SIGCSE Bull.*, Vol. 27, No. 3, pp. 21–26, 1995.

[7] Kim B. Bruce. Controversy on how to teach CS 1: a discussion on the sigcse-members mailing list. In *ITiCSE-WGR '04: Working group reports from ITiCSE on Innovation and technology in computer science education*, pp. 29–34. ACM Press, 2004.

[8] Michael Kölling and John Rosenberg. Guidelines for teaching object orientation with Java. In *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pp. 33–36. ACM Press, 2001.

[9] 佐藤和浩. 小学校におけるプログラミング活用の現状と課題. 情報処理学会研究報告 「コンピュータと教育」, Vol. 78, No. 9, pp. 57–63, 2005.

[10] Mark Guzdial. A media computation course for non-majors. In *ITiCSE '03: Proceedings of the 8th annual conference on Innovation and technology in computer science education*, pp. 104–108. ACM Press, 2003.

[11] 伊知地宏. 大学１年生へのプログラミング教育の試み. 情報処理学会 2002 年度 夏のプログラミングシンポジウム, 2000.

[12] Vennila Ramalingam, Deborah LaBelle, and Susan Wiedenbeck. Self-efficacy and mental models in learning to program. In *ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pp. 171–175. ACM Press, 2004.

[13] Tony Jenkins and John Davy. Diversity and motivation in introductory programming. *The e-Journal of the LTSN-ICS*, Vol. 1, No. issue1-3, 2001.

[14] 大岩元, 半田亭, 辰巳丈夫, 橘孝博, 久野靖. 情報科教育法. オーム社, 2001.

[15] Kim B. Bruce, Andrea Pohoreckyj Danyluk, and Thomas P. Murtagh. Materials for Java: An eventful approach, 2004.
http://eventfuljava.cs.williams.edu/.

[16] John Lewis. Myths about object-orientation and its pedagogy. In *SIGCSE '00: Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pp. 245–249. ACM Press, 2000.

[17] 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/.

[18] 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/.

[19] Ralph Westfall. Technical opinion: Hello, world considered harmful. *Commun. ACM*, Vol. 44, No. 10, pp. 129–130, 2001.

[20] Qusay H. Mahmoud, Wlodek Dobosiewicz, and David Swayne. Redesigning introductory computer programming with HTML, JavaScript, and Java. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pp. 120–124. ACM Press, 2004.

[21] Eric Roberts. An overview of minijava. In *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pp. 1–5. ACM Press, 2001.

[22] Jason Hong. The use of Java as an introductory programming language. *Crossroads*, Vol. 4, No. 4, pp. 8–13, 1998.

[23] James I. Hsia, Elspeth Simpson, Daniel Smith, and Robert Cartwright. Taming Java for the classroom. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pp. 327–331. ACM Press, 2005.

[24] 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.

[25] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the future: the story of squeak, a practical smalltalk written in itself. In *Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 318–326. ACM Press, 1997.

[26] 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.

[27] Eric Allen, Robert Cartwright, and Brian Stoler. DrJava: a lightweight pedagogic environment for Java. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pp. 137–141. ACM Press, 2002.

[28] P. Niemeyer. Beanshell - lightweight scripting for Java. http://www.beanshell.org/.

[29] 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.

[30] 日野孝昭. 初心者プログラミング学習補助法〜段階的プログラミング学習〜. 修士論文, 早稲田大学大学院理工学研究科情報ネットワーク専攻, 2005.
http://taurus.kake.info.waseda.ac.jp/nspec/hino.pdf.

[31] 鈴木健司, 小田嶋祐介, 長慎也, 甲斐宗徳, 筧捷彦. オブジェクト指向プログラミングにおけるオブジェクトの自動可視化 - Nigari への実装を例として. 第 3 回 情報科学技術フォーラム (FIT), No. A-031, 2004.

[32] 長慎也, 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 2004 年度版, 2004.
http://pionero.kake.info.waseda.ac.jp/prog2004/frame.html.

[33] Kathryn E. Gray and Matthew Flatt. ProfessorJ: a gradual introduction to Java through language levels. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 170–177. ACM Press, 2003.

[34] 長慎也. Catwalk, 2002.
http://taurus.kake.info.waseda.ac.jp/catwalk/.

[35] 長慎也, 筧捷彦. 「つながり」を利用した検索語からの同義語の抽出. ソフトウェア科学会「コンピュータ ソフトウェア」, Vol. 20, No. 4, pp. 49–55, 2003.

[36] linux-users メーリングリスト, 1998.
http://www.linux.or.jp/community/ml/linux-users/.

[37] P. A. W. Lewis, P. B. Baxendale, and J. L. Bennett. Statistical discrimination of the synonymy/antonymy relationship between words. *Journal of the ACM (JACM)*, Vol. 14, No. 1, pp. 20–44, 1967.

[38] 山本英子, 梅村恭司. 辞書を用いない関連語リストの構築方法. 情報処理学会研究報告, Vol. NL-148, No. 12, pp. 81–88, 2002.

[39] 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.

[40] 大友賢二. 項目応答理論入門. 大修館書店, 1996.

[41] 原田康也ら. 学習履歴の双対性再考-英語語彙学習履歴のマイニングに向けて-. 情報処理学会研究報告, Vol. CE-75, No. 7, pp. 49–56, 2004.

[42] Christopher J. Van Wyk. Programming as writing: using portfolios. *SIGCSE Bull.*, Vol. 27, No. 4, pp. 39–42, 1995.

[43] John K. Estell. Ipp: a web-based interactive programming portfolio. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pp. 149–153. ACM Press, 2001.

[44] Tom Schorsch. Cap: an automated self-assessment tool to check pascal programs for syntax, logic and style errors. In *SIGCSE '95: Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education*, pp. 168–172. ACM Press, 1995.

[45] Al Lake and Curtis Cook. Style: an automated program style analyzer. *SIGCSE Bull.*, Vol. 22, No. 3, pp. 29–33, 1990.

[46] Findbugs.
http://findbugs.sourceforge.net/.

謝辞

本論文をまとめるにあたり，早稲田大学理工学研究科 筧 捷彦教授にご助言，ご指導をいただきました．また，早稲田大学理工学研究科 上田和紀教授，深澤良彰教授，山名早人助教授には，ご教示，ご助言をいただきました．感謝いたします．

本論文中の 2 章，4 章にて開発した Nigari System および N-Java を用いた授業の実施にあたっては，成蹊大学の甲斐宗徳教授に多大なご協力をいただきました．感謝いたします．

Nigari System および N-Java の開発，およびそれらを用いた教材の作成には，早稲田大学理工学研究科の小田嶋祐介君，川合晶君，佐々木康太朗君，鈴木健司君，日野孝昭君，前島真一君に協力をいただきました．感謝いたします．

また，Nigari System，N-Java，および 5 章にて開発した proGrep を用いた授業を受講し，システムや授業方法に対する率直な意見を寄せて頂いた，早稲田大学 CS 学科の学生諸君にも感謝いたします．

本研究は，ここに示した方々をはじめ，多くの人々からのご指導，ご協力のもとに成し遂げることができました．ここに謹んで感謝の意を表します．

研究業績

査読付き学術論文

- 長慎也, 甲斐宗徳, 川合晶, 日野孝昭, 前島真一, 筧捷彦. プログラミング環境 Nigari - 初学者が Java を習うまでの案内役. 情報処理学会論文誌：プログラミング, Vol. 45, No. SIG9(PRO22), pp. 25–46, 2004.
- 長慎也, 筧捷彦. 「つながり」を利用した検索語からの同義語の抽出. ソフトウェア科学会「コンピュータ ソフトウェア」, Vol. 20, No. 4, pp. 49–55, 2003.

講演

- 長慎也. proGrep - プログラミング学習履歴検索システム. 情報処理学会コンピュータと教育研究会 第 78 研究会, pp. 29–36, 2005.
- 長慎也, 日野孝昭, 前島真一, 小田嶋祐介, 佐々木康太朗, 筧捷彦. プログラミングの授業における，支援システムの開発と実践例. 情報処理学会コンピュータと教育研究会 第 76 回研究会, pp. 9–16, 2004.
- Shinya Cho, Katsuhiko Kakehi, Akira Kawai, Shinichi Maeshima, and Takaaki Hino. Nigari system - Stairway to Java. In *International Association for Development of the Information Society (IADIS) e-Society 2004*, pp. 960–965, 2004.
- 長慎也, 甲斐宗徳, 川合晶, 日野孝昭, 前島真一, 筧捷彦. Nigari - 他の言語への応用を考慮した初学者向けプログラミング言語. 情報処理学会 コンピュータと教育研究会 第 71 回研究会, pp. 13–20, 2003.

その他

- 長慎也. Tonyu ーアクションゲーム製作に特化した開発環境. 情報処理学会第 38 回プログラミング研究会, Vol.43, No. SIG8-022 pp. 121, 2002.
- 長慎也, 松田能成, 筧捷彦. ユーザの行動に基づく文書の自動分類. 電子情報通信学会 2001 年度情報システムソサエティ大会, D-5-3, pp. 35, 2001.
- 長慎也, 栗栖将也, 筧捷彦. エージェントを用いた自動情報配信システムの設計. 電子情報通信学会 2001 年度総合大会, D-8-19, pp. 118, 2001.

- 長慎也. 知的な自動情報配信システムの設計. 情報処理学会第６１回（平成１２年後期）全国大会, pp. 2–107–108, 2000.