

proGrep - プログラミング学習履歴検索システム

長 慎也[†] 筧 捷彦[†]

教室におけるネットワーク環境の整備が進み、プログラミングの授業において、学習者の書いたプログラムや、学習者が出したエラーなどの詳細な学習履歴を収集することが可能となった。この収集された情報から、学習者の特徴的な行動を抽出して、学習者の出来を判断したり、学習者に自動的にアドバイスを与えたりできるシステム“proGrep”を開発した。今回は、このシステムの機能と、授業での活用事例について報告する。

proGrep - a programming education support system with pattern matching over study logs.

SHINYA CHO[†] and KATSUHIKO KAKEHI[†]

We propose a system to support programming education with pattern matching mechanism over students study logs and implement a prototype called “proGrep”. Study log records various events of a programming learner and information related to them, i.e., creation and/or modification, compilation and execution of programs and system messages at those times. We apply proGrep to a programming class and get effective results with primitive matching mechanism over log patterns.

1. はじめに

プログラミングの授業において、各学習者が1台ずつコンピュータを使用し、プログラムを作成しながら演習を行うことが多い。さらに、最近ではほとんどのコンピュータがネットワークに接続されている。これらのことから、学習履歴を詳細に集めることができるようになってきた。

本研究では、豊富な学習履歴を用いて、プログラミングの授業にまつわる問題点を解決することを試みる。

2. プログラミング教育での困難

プログラミングを教えるにあたっては、つぎの二つの困難に直面する。

2.1 実習環境に起因する困難

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

● 学習環境が不親切

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ず、かえって困惑させるこ

とさえある。

● 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、不具合に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。

学習者は、自力で解決しようがなく、TAや教員の助けを借りるほかない。これを何度も経験すると、自ずから考え対処することをしなくなりかねない。

2.2 大きな能力差に基づく困難

プログラミング能力は、習得できるレベルにも、習得していく速度にも、個人的な差が大きい。能力別クラス編成は、授業開始時の既習レベルへの対処法になっても、その後に生じる差には対応しきれない。学習者の能力を常に計測し、その状況に応じて教材や教え方を変えていく必要がある。

3. 学習履歴の収集活用

プログラミング教育での困難を解消する方法として、学習者の学習履歴を収集し解析してその教育に活用することを考える。

3.1 学習履歴の収集

実習環境がネットワークに接続されている場合が多

[†] 早稲田大学 理工学研究科

Graduate School of Science and Engineering, Waseda University

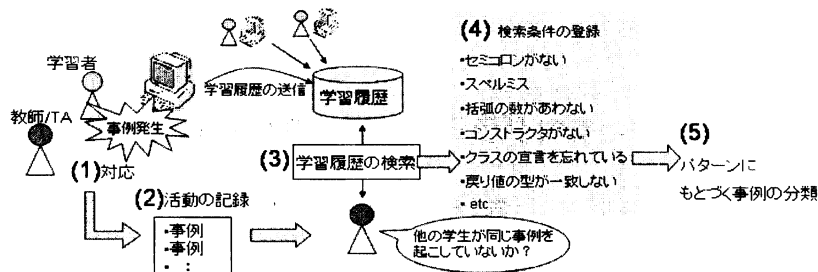


図1 学習履歴の分類

い。このため、学習者の実習履歴を逐一記録していくことが可能となる。例えば、Nigari¹⁾、N-Java²⁾などの初学者向けのプログラミング環境に、その学習者の行った操作をサーバに自動送信する仕組みを組み込むことができる。授業で随時に行う小テストやアンケートも、オンラインで行うようにすることで、その結果も学習履歴として記録しておくことができる。

3.2 学習履歴の活用

収集された学習履歴をプログラミング学習に活用する方法として、いくつかのものが考えられる。

● 困難解決の補助

収集した多量の学習履歴を解析することで、学習者に生じる困難を類型化することが可能になる。その結果を学習環境に組み込み、学習者が困難に直面したときに、その類型を検出し、それに対するアドバイスを自動提示できるようにしておく。アドバイスを適切に用意しておけば、学習者が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。

● 学習状況の把握と対応

それぞれの実習課題に対する解答状況や、小テスト・アンケートの結果などがすぐに解析できる。学習者全体の学習状況を把握することで、次の課題に進むことを促したり、臨時に解説を補ったりすることが可能となる。学習者集団に合わせて、次の講義内容を調節したり、教材を改良したり、授業計画を作り直したりすることも可能となる。

4. 学習履歴の解析

学習者に生じる困難を類型化するために、収集した学習履歴を解析したい。実習の場合は、TAが学習者の支援にあたっている。それぞれの支援事例にあたってTAが下した判断をこの解析に利用する。すなわち、

TAの活動記録を別途記録することにして、図1に示すように、つぎの手順で解析を行う。

- (1) 何らかの事例が発生したときにTAが対応する。
- (2) TAは、対応した事例を記録しておく。これをTA活動記録とする。
- (3) TA活動記録を参照し、似たような事例を他の学習者も起していないかを、全学習者の履歴の中から検索する。
- (4) 多くの学習者が起こしていた事例であれば、検索に用いた検索条件を、システムに登録する。
- (5) 登録された検索条件を分類基準として、事例の分類を行う。すなわち、一つの検索条件に一致する事例は、すべて同じ種類であるとして扱う。

5. 関連研究

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³⁾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）の回答状況や、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⁴⁾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。⁵⁾⁶⁾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

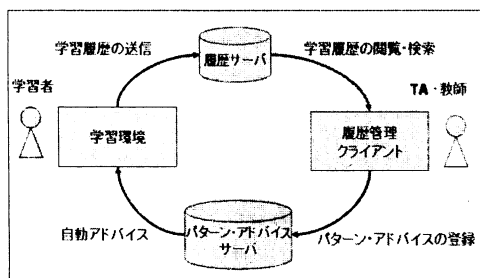


図 2 proGrep の構成

を提出させるに留まっている。

6. 学習履歴管理システム proGrep

本研究で提案する、学習履歴の収集活用を支援するシステムを「proGrep」という。proGrep の構成を図 2 に示す。

次に、proGrep の構成要素と、それらのもつ機能を説明する。

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。

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イベントを検索することができる。

6.3 パターン・アドバイスの登録

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、TA や教員は、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。

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履歴を生み出すようなイベント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。

7. 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。

7.1 学習環境

学習環境として、“JavaEditor” という統合開発環境を作成した。動作イメージを図 3 に示す。JavaEditor 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。

JavaEditor 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、JavaEditor にはアドバイス機能がある。学習環境にて、イベントが発生した瞬間に、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と、イベントに関連する場所を指摘 (図 3 の中央部分) する。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 3 の右側のウィンドウ)

7.2 履歴管理クライアント

履歴管理クライアントは、Web アプリケーションとして実装した。履歴サーバに蓄積された履歴を検索する機能の他に、特定の学習者の履歴を一つ一つ順番に閲覧する機能もある。

7.2.1 履歴閲覧機能

環境 ID を入力すると、その環境から送信された学習履歴 (つまり、その環境で発生したイベントの一覧) を時系列順に表示する。作成イベント、翻訳イベント、実行イベントがあり、特定のイベントのいずれかだけを抽出して表示することも可能である。

7.2.2 履歴検索機能

履歴サーバに蓄積された履歴から、特定のイベントを検索する機能である。動作イメージを図 4 に示す。

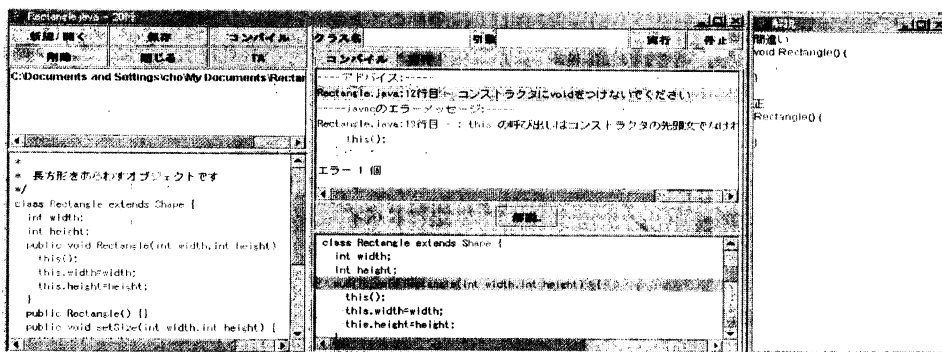


図3 JavaEditor

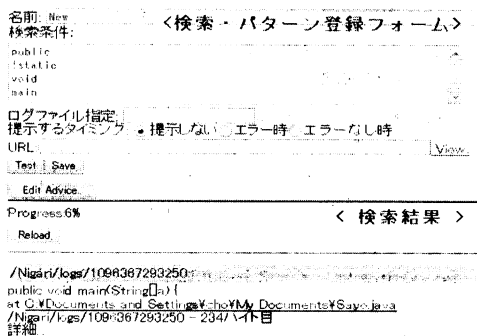


図4 履歴検索機能

```
public
!static
void
main
```

図5 検索条件の例(1)

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

図6 検索条件の例(2)

現段階では、作成イベントのみが検索できる。

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。

- 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6は、「セミコロンで終わっていないくて、かつ { でも終わっていないくて、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。

7.2.3 パターン登録機能

履歴検索機能で使った検索条件は、パターンとして登録することができる。

パターン登録の際は、次の情報を入力する必要がある。

```

FILENAME
Rectangle
NOTEXIST
class Rectangle

```

図 7 検索条件の例 (3)

```

if 文の等式は = ではなく == です
セミコロンがありません
{ と } の数が合いません
( と ) の数が合いません
全角スペースが入っています

```

図 8 定義済みパターン

ある。

- **名前**

このパターンをあらわす適切な名前を指定する。
この名前が学習環境においてアドバイス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。

- ー **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。
- ー **エラー時** 翻訳上のエラーの発生時に、エラーを修正させる目的で提示する。
- ー **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、好ましくない書き方をした場合などに用いる。

- **アドバイス**

このパターンのアドバイスを記述する。現在の実装ではプレーンテキストのみが記述できる。

8. 実 験

8.1 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 言語とプログラミングの基礎の習得
- **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)」⁷⁾
- **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。

8.2 定義済みパターン

授業開始前に予想できたパターンを定義済みパターンとして予め JavaEditor に組み込んでおいた。定義

済みパターンの一覧を図 8 に示す。なお、定義済みパターンのうち、括弧の数が一致しないというパターンは履歴管理クライアントの検索機能では記述できない。

8.3 作成されたパターン一覧

授業中に学習者がおこした事例をもとに作ら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

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

図 9 「コンストラクタに void をつけないでください」に該当するプログラム

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

図 10 「i の範囲に注意してください」に該当するプログラム

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 9 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 10 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、配列の使い方を忘れていたり、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 1 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、これらの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、そのパターンが登録される前であれば、アドバイスは提示されない。このため実際にアドバイスが提示された回数は表 1 の値より少なくなる。そこで、表 2 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もので、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、事例の収集、検索、パターンの登録を速やかに行ったため、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。

8.4 活 用 例

登録された学習履歴やパターンは、次のような用途に活用した。

- 補習
多く検出されたパターンに関して、次回の授業の冒頭にて補習を行った。
- 間違い直し問題
パターンに該当する作成イベントを適当に取りだし、そのプログラムの間違いを修正させる練習問題を作成した。

9. 考 察

9.1 効 果

proGrep を用いることで、次の効果が得られた。

- 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、実際に多くの学習者が直面した事例を抽出することが可能になった。
- 事例に直面した学習者に対して、自動的にアドバイスを提示することが可能になり、実際に一部の学習者には、アドバイスが提示された。
- 登録されたパターンを利用して、教材や指導方法に修正を施すことが可能になった。
- システムを Web アプリケーションとして実装することで、授業中に事例の検索やパターンの登録

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

表 1 パターンの出現回数

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

表 2 アドバイス表示回数

ができ、迅速な対応ができた。

9.2 問題点

一方、次のような問題点が明らかになった。

9.2.1 検索条件の記述能力の限界

履歴管理クライアントの検索機能では、適切な検索条件が記述できず、検索ができない事例が多く見られた。

そこで、TA に学習者の履歴を順番に調べ、事例と思われるイベントのうち、パターンに登録されていないものを抽出してもらった。このような事例が 66 個見つかった。一部を次に示す。

図 11 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 12 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 13 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 14 の例では、メソッドの中にメソッドの定義を書いている。

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 15 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 16 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

```
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 11 検索できない事例

```
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 12 検索できない事例

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。

さらに、今回登録されたパターンは、「displayInfo の戻り値は void です」などに代表されるように、特定の課題の演習を行っている間に発生した事例を検出することはできても、それと類似する他の事例を検出することはできないものが多かった。

```

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 13 検索できない事例

```

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 14 検索できない事例

```

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

```

図 15 検索できない事例

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。

9.3 学習履歴から能力の推定

学習履歴を何らかの意味で分類することはできても、そこから学習者の能力がどの程度なのかを判断できるとは限らない。例えば、ある事例に直面した学生と、そうでない学生に明らかな能力差がある、というような規則を効率よく見つけ出すための、履歴の解析手法について今後検討する必要がある。

参 考 文 献

- 1) 長慎也, 甲斐宗徳, 川合晶, 日野孝昭, 前島真一, 寛捷彦: 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, pp. 25-46

```

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 16 検索できない事例

(2004).

- 2) 長慎也, 日野孝昭, 前島真一, 小田嶋祐介, 佐々木康太郎, 寛捷彦: 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, Vol. CE-76, pp. 9-16 (2004).
- 3) 大友賢二: 項目応答理論入門, 大修館書店 (1996).
- 4) 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).
- 5) Wyk, C. J. V.: Programming as writing: using portfolios, *SIGCSE Bull.*, Vol. 27, No. 4, pp. 39-42 (1995).
- 6) Estell, J. K.: IPP: a web-based interactive programming portfolio, *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, ACM Press, pp. 149-153 (2001).
- 7) 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).