Vol.2015-CE-128 No.10
2015/2/14

Javaコードのクラス間連携理解のための コアステートメント抽出アルゴリズム

石原 信也[1,a)]　舩曳 信生[1,b)]

概要：Java プログラミング学習では，既存のソースコードを読み，その処理や構造を理解することが有効である．このコードリーディングをプログラミング教育に取り入れるためには，学習者がソースコードを理解したか否かを確認する方法が必要となる．そのため本グループでは，ソースコード中の肝要なステートメント（**コアステートメント**）を空欄とし，その記述を要求する，**ステートメント空欄補充問題**の提案と授業適用を行っている．本研究では，ソースコード中の空欄化すべき**コアステートメント抽出アルゴリズム**を提案する．本アルゴリズムでは，コアステートメントとして，1）要求仕様を充たすためのアルゴリズム要素，2）オブジェクト指向言語の特徴であるクラス間連携要素，3）データベースなど外部機能との連携要素，の 3 種類を想定し，それぞれの抽出法を定義している．提案アルゴリズムを実装し，5 つの Java ソースコードに適用して得られたコアステートメントを，Java 学習者による抽出結果と比較した．その結果，両者に高い一致がみられ，提案アルゴリズムの有効性が確認された．

キーワード：Java，プログラミング教育，JPLAS，PDG，デザインパターン，コアステートメント

1. まえがき

オブジェクト指向のプログラミング言語 Java は，開発環境，堅牢性，可搬性などに優れ，エンタープライズシステムから組み込みシステムに至るまで，産業界で広く利用されている．そのため，産業界からは，Java 言語技術者の育成が強く求められており，実際，大学，専門学校など多くの教育機関で，Java プログラミング教育が行われている．

Java 言語は，C 言語に似た文法を有することから，一般に，C 言語を学習した人には学習しやすい言語であるとされる．それにも関わらず，Java 言語の学習は，C 言語の学習とは違った難しさがある．その一つは，オブジェクト指向言語である Java 言語では，複数のクラスを用いてコードを構成することからきている．C 言語の学習を終えた人が Java 言語を学習しようとする場合，このような C 言語から拡張された要素が学習上のネックとなり得る．

この C 言語からの拡張要素には，演算子においていくつか存在する．例えば，変数の大きさを調べる sizeof 演算子は C 言語でも定義されているが，Java ではそれに加え，オブジェクト（インスタンス）が指定したクラスまたはその上位のクラスに属しているかどうかを調べる instanceof 演算子が定義されている．

また，C 言語では，その実現したい仕様（アルゴリズム）の実装を，基本的には，構造化プログラミングに沿ったフローチャートと領域図で机上のトレースが可能である．しかし Java 言語では，それに加えて，継承を通じた多態性や多義性を駆使し，クラス間の連携で実装されることが一般的である．

一般に，プログラミング言語が外部機能と連携する要素は，その機能を知っていなければ難しい．例えば，C 言語におけるアセンブラの内包や OS の API の呼出し，Java 言語における正規表現による文字列のハンドリングや SQL によるデータベースへのアクセスは，外部機能への連携と考える．Java 言語では，Web アプリケーションなどにおいて，外部機能との連携が頻繁に使用されるため，その理解が不可欠となる．

一方，C 言語や Java 言語の従来のプログラミング教育では，学生に対して，有効とされている**コードリーディング**に対する学習が明示的に行われていないといった問題点が指摘されている [1].

2002 年には，Kent Beck や Erich Gamma によって，**Junit**[2] が開発され，**テストファースト**の開発技法が広まり始める [3]．テストファーストの開発技法は，**テスト駆動型**

[1] 岡山大学大学院自然科学研究科
Okayama Uniersity
[a)] n.ishihara@okayama-u.ac.jp
[b)] funabiki@okayama-u.ac.jp

開発 TDD （Test-Driven Development）手法とも呼ばれ，文字通り**プロダクトコード**の開発に先立ってそれが実現すべき仕様を，**テストコード**として記載しておく手法であり，レグレションやアジャイル開発に向いているとされる．

一般に，コンピュータ支援教育 CAI（Computer-Assisted Instruction）では，オンライン化されることで，学習結果の即時フィードバック，学習場所・時間の制約の排除，学習環境の構築不要といった点で，学習効率を上げることが可能とされている．反面，プログラミング教育のような実体験が重要となる分野には向かないとされている [1]．

このような背景の中で，本グループでは，Java プログラミング教育を支援するための Web によるオンラインサービスシステムとして，**Java プログラミング学習支援システム JPLAS（Java Programming Learning Assistant System）**を提案・実装している [4]-[5]．JPLAS では，学生の解答のオンラインでの自動採点を行うことで，教員負担の軽減，学生の自習環境の支援を実現する．また，JPLAS では，Java プログラミング学習の進捗状況や学生の理解度に応じて，適切な学習環境を提供するために，以下の 3 種類の問題を用意している．

最初の問題は，教員が模範となる良質の Java コードを選択し，その中から学ぶべき語を空欄として学生に与え，そこに適切な語のスペルの入力を要求する**エレメント空欄補充問題**である [6][5]．本問題は，Java の文法などを学ぶ初学者を対象に，基本的な文法の学習に加え，コードリーディングの習慣の定着を狙いとしている．

JPLAS の 2 つ目の問題は，Java コード中の肝要なステートメント（**コアステートメント**）を空欄とし，その補充を要求する**ステートメント空欄補充問題**である．一通り文法学習を終え，コード作成を学ぶ学生にとっては，既存のソースコードを読み，その処理や構造を理解することが有効である．本問題では，コアステートメントの補充により，学生が与えられたソースコードを理解したか否かを確認する手段を提供している．

3 つ目の問題は，**テスト駆動型開発手法**により，学生の解答コードの自動検証を行う**コード作成問題**である [4]．ここでは，学生は，JPLAS で提示される**テストコード**（テスト用の Java コード）の実行でエラーを検出しない Java コードの作成が求められる．本問題は，2 種類の問題を通じての文法や良質な Java コードの学習を終え，クラス全体のコード作成を学ぶ学生を対象としている．

本研究では，この中でステートメント空欄補充問題を対象としている．

ステートメント補充問題では，コードリーディングの対象となるソースコード（問題コード）を選定し，その中からコアステートメントを空欄として，学生には，その空欄に適切なステートメントを入力させる形式の演習問題である．コアステートメントは，ソースコード内で最も重要なものである．プログラムスライシングといった，重要ステートメントを抽出するための解析技法は，Java 言語に適用するには，計算量が膨大となったり，実装が困難であるため，新しいアルゴリズムが必要である．

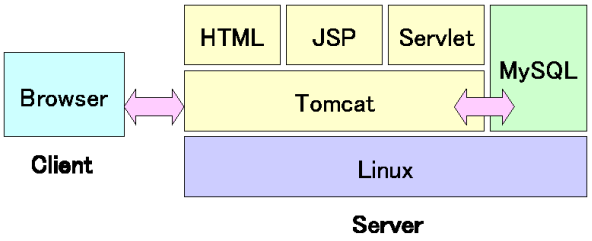

図 1 JPLAS の実装環境
Fig. 1 Platform for JPLAS.

そこで本研究では，Java 言語では，C 言語に似た文法を有することに加え，C 言語から拡張された要素を有することに着目して，以下の 3 種類の学習すべき要素を想定し，それぞれでのコアステートメントの抽出アルゴリズムを与えることとした．

1) 要求仕様を充たすためのアルゴリズム要素
2) オブジェクト指向言語の特徴であるクラス間連携要素
3) データベースなどの外部機能連携要素

1) では，先行研究 [7] において実装したように，文献 [8] に従い，プログラム依存グラフ PDG（Program Dependence Graph）を利用することで抽出する．2) では，ドット演算子と引数に注目してステートメントをランキングすることで抽出する．3) では，文字列定数の中身に注目することで抽出する．

提案アルゴリズムの評価として，5 つの Java コードに適用して抽出されたコアステートメントを，Java の学習者による主観的な抽出結果と比較した．その結果，両者に高い一致がみられ，提案アルゴリズムの有効性が確認された．

以下に本論文の章構成を示す．まず，2 章で JPLAS の先行研究について述べる．次に，3 章および 4 章で提案アルゴリズムとその検証について述べる．最後に，5 で本論文のまとめと今後の課題を述べる．

2. JPLAS の先行研究

本章では，JPLAS の先行研究について述べる．

2.1 JPLAS の実装環境

JPLAS の実装環境では，図 1 に示すように，サーバ OS に Linux，Web サーバ兼アプリケーションサーバに Tomcat を利用し，サーバプログラムは JSP と Servlet を用いて記述している．また，データベースには MySQL を用いている．これらは，すべてオープンソースソフトウェアである．ユーザ（学生， 教員）は，Web ブラウザを用

いてサーバにアクセスし，そのサービスを受けることとしている．そのため，学生は，インターネット接続の端末を使用することで，次節以降に示す，3 種類の問題を解くことで，いつでもどこでも，Java プログラミング学習が可能となる．

2.2 JPLAS の 3 種類の問題

JPLAS には，3 種類の問題が実装されている．

2.2.1 エレメント空欄補充問題

エレメント空欄補充問題では，問題コードの中の学習すべきエレメント空欄とし，学生には，そこに当てはまるエレメントの入力が解答を求められる．各解答をサーバで，空欄化前の語との比較により正誤判定を行う．空欄化されたエレメントが別解のない一意の正解となるように，適切に選択する必要があるため，グラフ理論を応用した空欄語選択アルゴリズムを提案している．

2.2.2 ステートメント補充問題

ステートメント補充問題では，問題コードの中のコアステートメントを空欄とし，学生には，与えられた要件を充たすステートメントの入力が求められる．**図 2** に本問題の例を示す．エレメント空欄補充問題と異なり，正解となるステートメントは通常，一意に記述できないため，テストコードを用いたテストにより，正誤判定を行う．そのため，ブラウザ上で，解答ステートメントを問題コードの空欄部に埋め込み，コードを完成させてから，サーバに送信している．

2.2.3 コード作成問題

コード作成問題では，学生は，課題文とテストコードを与えられ，それらの要件を充たすコードの作成が求められる．要件を満たす，様々な解答コードに対処するために，JUnit [2] により，テストコードを用いた自動検証を行っている．

2.3 ステートメント補充問題の先行研究

本研究では，ステートメント補充問題のためのコアステートメント抽出アルゴリズムを提案する．そのため，本問題の先行研究を紹介する．

2.4 プログラム依存グラフ

先行研究では，コアステートメントの抽出に，**プログラム依存グラフ** *PDG*（*Program Dependency Graph*）を利用している．PDG は，プログラム内のステートメントの間に存在する依存関係を表す有向グラフである．PDG の点はステートメントであり，辺で結ばれた点同士にデータ依存あるいは制御依存の関係があることを表す．データ依存関係は，各変数に対して，その定義ステートメントから参照ステートメントへ辺を作成することで表される．

先行研究では，データ依存関係のみを考慮している（**図 4**）．また，次のような「見なし」を行っている．

(1) 「変数」には，プリミティブなものに加え，「オブジェクト」も含むこととした．逆に，オブジェクト内の変数（メンバ変数）にその値の変更などの何らかの操作があった場合は，オブジェクトそのものが再定義・参照されたものと見なした．これは同じ操作を，オブジェクトごととメンバ変数ごとに重複してカウントを防ぐためである．
(2) オブジェクトへの代入は，代入文の左辺にそのオブジェクトが現れるものと，オブジェクトのメソッドを呼び出すものとした．
(3) オブジェクトの参照は，代入文の右辺にオブジェクトが現れるものと，メソッドの引数として使用されているものとした．いわゆる変数の「使い回し」はないものとした．

次にコードの例を示す．

```
b.setTime(a.end.selTime()+i*7*1000*60*60*24);
c.next.start = d.parent.start;
```

1 行目の例では，変数 i に加えて a，b も変数と見なし，i は参照，b は再定義， a は参照であり再定義であり，end は a のメンバ変数だが，含まない．2 行目の例では，c，d を変数とし，c が再定義であり，d がは参照である．

2.5 プログラム依存グラフの問題点

PDG を用いてコアステートメントを抽出する際に，1 つの PDG の適用範囲と，文字列定数の制約に問題がある．まず，1 つの PDG は，メソッド（関数）単位で適用しなければならない．変数の使いまわし，引数，広域変数を考慮に入れても，パッケージ外のクラスやメソッドなどバイナリファイルで提供されている領域に掛かると，ソースコードからの解析は難しくなるため，1 つの PDG は狭い範囲に限られる．

また，PDG では，文字列定数が抽出されないため，出題者の意図と異なるコアステートメントが抽出される場合がある．正規表現や SQL 文などの文字列定数は，学習価値のあるものであるが，変数に着目して抽出を行うため，それらを含むステートメントが，必ずしも抽出されるわけではない．これらの問題点の対策が必要である．

3. コアステートメント抽出アルゴリズムの提案

本章では，コアステートメント抽出アルゴリズムの提案を行う．

3.1 Java プログラミング学習の 3 要素

本研究では，前節で示した先行研究の問題点を解決するために，Java プログラミングにおいて学ぶべき要素を **図 3**

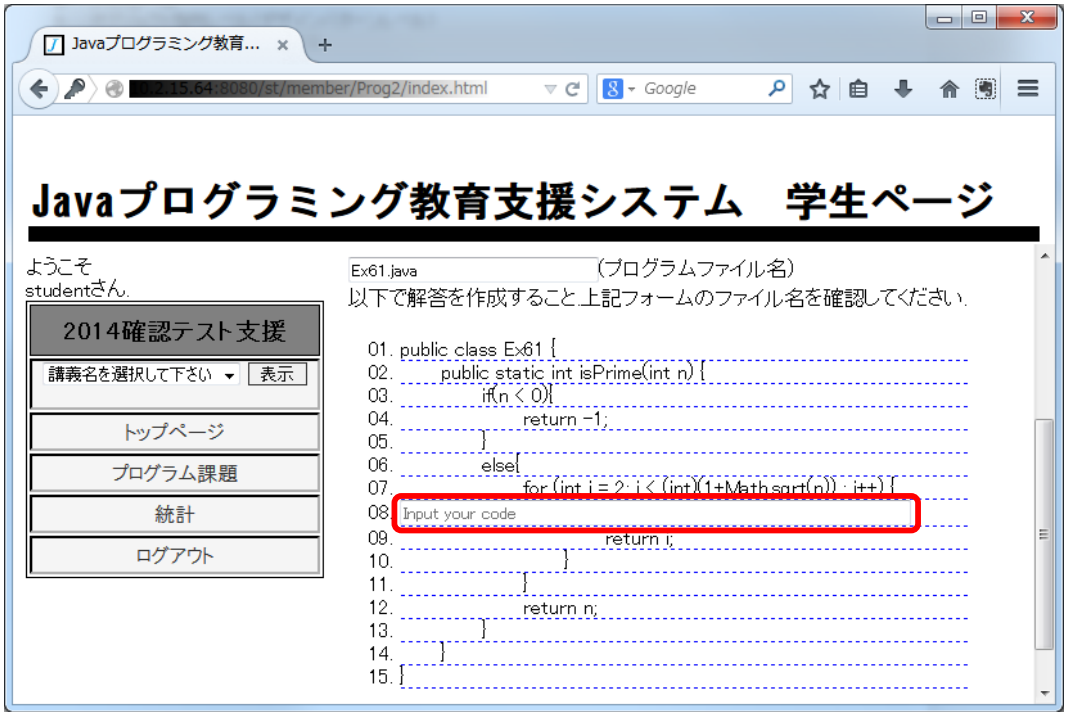

図 2　JPLAS とステートメント補充問題
Fig. 2　Statement Fill-in-blank Problem in JPLAS.

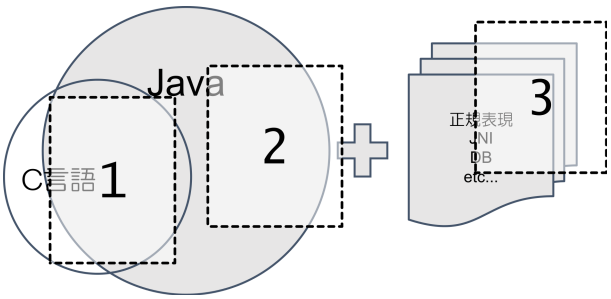

図 3　Java プログラミング学習の 3 要素
Fig. 3　Three items in Java programming learning.

に示すように，以下の 3 種類とし，それぞれでのコアステートメント抽出アルゴリズムを提案する．

(1) 要求仕様を充たすための**アルゴリズム要素**
(2) オブジェクト指向言語の特徴である**クラス間連携要素**
(3) データベースなどの**外部機能連携要素**

アルゴリズム要素は，図 3 の 1 である，Java 言語のメソッド内部に相当するそのコアステートメントは，先行研究の PDG を用いて抽出する．

クラス間連携要素は，オブジェクト指向の特徴であるクラス間連携に関連する部分であり，PDG のように，エレメント毎の解析を「積み上げて」，コアステートメントを抽出することが出来ない．そのため，本研究では，クラス間連携を利用しているステートメントをコアステートメントとして抽出する．

外部機能連携要素は，その性質上，独特の記述を有している．コード上では，文字列定数として出現することが多いため，コードの構造や変数を解析して取り上げることができない．反面，特定の文字列に注目して検出すると，容易にコアステートメントを抽出することができる．例えば，SQL 文であれば，文字列定数内に”SELECT”などの SQL の予約語が出現するため，それで抽出する．また，文字列クラスの split メソッドなどで使用される正規表現は，文字列に含まれる記号の羅列により抽出する．

3.2　アルゴリズム要素のコアステートメント抽出

アルゴリズム要素のコアステートメントは，各メソッドを対象に PDG を利用して抽出する．PDG では 図 4 のようにデータ依存のみに注目する．以下に本手続きを示す．

(1) 読み込んだソースコードをセミコロン（”;”）と左波括弧（”{ ”）を手がかりにステートメント単位に分割する．
(2) 各ステートメントをエレメント単位に分割する．
(3) 以下の条件を満たすエレメントを変数として抽出する．
  (a) 先頭の文字がアルファベットである．
  (b) 予約語，クラスでない．
  (c) メソッドではない．これは，その後に右括弧（”(“）が続かないことで判断する．
(4) 変数の中で，ドット演算子が前置されないものを処理対象として選出する．ここで，ドット演算子が前置された変数は，インスタンス化されたオブジェクト内部の変数のため，除外する．
(5) 各変数を以下のルールで**定義変数**または**参照変数**に分

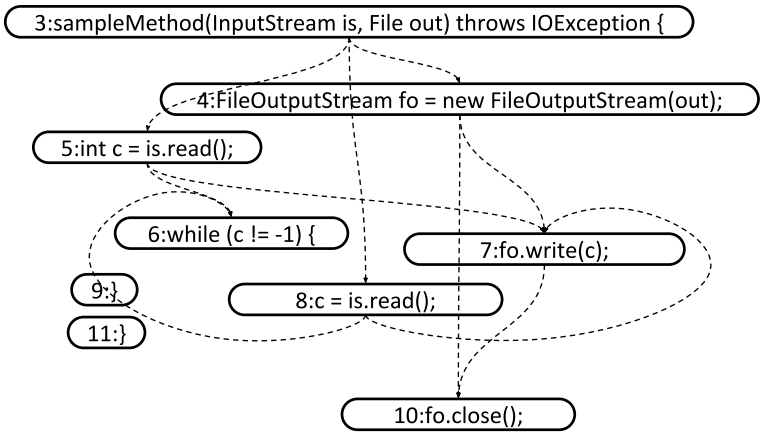

図 4　プログラム依存グラフ
Fig. 4　Program Dependence Graph.

類する．
(a) 定義句（直前の語が型指定）があれば定義変数
(b) 等号の左辺であれば定義変数
(c) 等号の右辺であれば参照変数
(d) メソッドが呼ばれていれば定義変数
(e) 引数は参照変数
(6) 定義変数から参照変数への辺を生成する．
(7) 辺の数をステートメント毎に集計する．

3.3　クラス間連携要素のコアステートメント抽出

クラス間連携要素のコアステートメントの抽出では，ドット演算子と引数に注目する．以下にその手続きを示す．
(1) 指定されたソースコードと同じフォルダーにあるソースファイルを読み込む．
(2) 読み込んだソースコードをセミコロン（”;”）と左波括弧（”{”）を手がかりにステートメントに分割する．
(3) ステートメントを単語に分割する．なお，ここではコメントや文字列定数は 1 語として取り扱う．
(4) 各ステートメントで以下の 2 つの値を算出する．
(a) ドット演算子によって連結されたエレメントの個数
その時，以下のドット演算子はクラス間連携と無関係なため対象から除外する．
(i) package 文と import 文に含まれるドット演算子
これらはパスなどをを示す役割であるため．
(ii) System.out と例外処理の e.printStackTrace．
これらは定型句であるため．
(b) 入れ子を含む引数の数．
(5) 同じステートメントの繰り返しは，繰り返されるたびに回答が容易になると考えられるため，ステートメントが連続して抽出された場合は最初のステートメントのみを選択する．

3.4　外部機能連携要素のコアステートメント抽出

外部機能連携要素のコアステートメントの抽出では，文字列定数の中身を調査する．以下にその手続きを示す．
(1) 読み込んだソースコードをセミコロンと左波括弧を手がかりにステートメントに分割する．
(2) ステートメントをエレメントに分割する．ここでは，コメントや文字列定数は 1 語として取り扱う．
(3) 各ステートメントの文字列定数（二重引用符号で始まるエレメント）で 40 文字以上のものに着目し，以下のルールでコアステートメントを判定する．
(a) 記号の比率が半分以上のものは正規表現のコアステートメントと判定
(b) select で始まるものは SQL のコアステートメントと判定

4.　提案アルゴリズムの検証

本章では，前章で示した 3 種類の要素含む 5 つの Java コードに対して，提案アルゴリズムによるコアステートメント抽出結果を，50 名の Java 言語の学習者による抽出結果と比較することで，その検証を行う．

4.1　検証方法

4.1.1　検証に用いた Java コード

本検証では，以下の 5 つの Java ソースコードを使用した．それぞれ，複数のクラスを持つ，24，45，54，32，40 ステートメントのプログラムで，package 毎に，1 つのソースファイルで作成したもの，複数のソースファイルに分けたものを取り混ぜた．これらのコードに，提案アルゴリズムを適用し，コアステートメントを抽出した．
(1) Java アプリケーションのフレームの実装コード（Window）
(2) バイナリサーチアルゴリズムの実装コード（BinarySearch）

(3) デザインパターンにおけるプロトタイプの実装コード（Prototype）
(4) デザインパターンにおけるシングルトンの実装コード（Singleton）
(5) デザインパターンにおけるビルダーの実装コード（Builder）

4.1.2 比較対象

次に，Java 言語の学習者に各ソースコードを提示し，各自でステートメント空欄補充問題を作成する場合，どの行を選択するかを選んで貰った．その際，複数ステートメントの選択も可とし，その場合には，順位を振って貰うこととした．

この学習者として，同じ研究室内の学生の 7，および，本学科の一般学生 43 名に依頼した．前者は，Java を用いた研究に従事しており，その基本には熟知しているが，熟練度にはばらつきがある．後者は，主に本学科の 2 年生で，C 言語と C++の履修後，現在，Java プログラミングの授業を履修中である．

4.1.3 Java プログラミング授業の進め方

ここで，比較対象の学生（学習者）が本調査を行うまでの Java プログラミングの授業の進め方を紹介する．まず，最初の 4 回の授業では，C 言語の文法事項の復習から Java への導入に行った．ここでの演習課題には，JPLAS のエレメント空欄補充問題を与えた．

次の 4 回では，String クラスのメソッドに出現する正規表現やデータベース連携のための SQL を中心に，ストリームや Window アプリケーションの作成を学習した．その際，Java 言語の豊富なライブラリを使用するためのマニュアルである，Javadoc の読み方を紹介した．ここでの演習課題には，JPLAS の 3 つの問題，すなわち，エレメント空欄補充問題，ステートメント空欄補充問題，コード作成問題のすべてを与えた．

最後の 4 回では，洗練されたオブジェクト指向プログラムの例として，Gof のデザインパターンを取り上げた．ここでの演習課題には，学生個々のアプリケーション作成を，自由なテーマで作成させた．なお，新規の JPLAS の演習課題は課さず，解答の遅れた学生の挽回期とした．

4.2 検証結果

4.2.1 アルゴリズムによる抽出結果

各 Java コードの 3 つの要素における，提案アルゴリズムによるコアステートメント抽出数を，表 1 に示す．ここで，アルゴリズム要素のコアステートメント抽出には PDG を用いるが，提案アルゴリズムでは，PDG はクラス内のメソッド単位で適用される．また，1 つの PDG で 1 コアステートメントを選択することから，コアステートメント抽出数はメソッド数に等しくなる．

表 1 コアステートメント抽出数

コード	アルゴリズム	クラス間連携	外部連携	学習者
Window	2	3	0	2
BinarySearch	3	3	0	3
Prototype	6	5	0	2
Singleton	2	3	0	3
Builder	6	6	0	5

4.2.2 学習者による抽出結果

学習者によるコアステートメント抽出では，全学生から総数で 351 のステートメントが回答され，その中から重複を排除すると，ステートメント数は 93 であった．その中には，まず，アノテーション（@Override など）のみのステートメントなど，今回，提案アルゴリズムで対象としていない要素が含まれており，学習者の抽出結果からは，除外することとした．また，少数の学習者のみの回答を例外として除くために，ステートメント毎に平均回答数を上回るもののみに限定した．これらにより，21 ステートメントと残った．更に，これも今回対象外である，Java の文法やライブラリが問題となっている 6 つのステートメントを除外した．その結果，学習者によるコアステートメント抽出結果として，15 ステートメントとなった．

ここで，最後に除外した 6 ステートメントは，以下のコード例に示すように，いずれも，new を含んでおり，new 以外には配列，Map，HashMap，ArrayList などを含んでいる．なお，new を含むステートメントの数は，全コードで 15 である．

```
WindowTest windowText = new WindowTest();
frame = new Frame("Window Test");
    // Set the initial value for the Frame
Entry[] table = new Entry[MAX];
    // Array to store the data
private Map<String, Prototype> hashmap
     = new HashMap<String, Prototype>();
List list =  new  ArrayList();
Director director =  new  Director(new Con...);
```

以上により得られた，学習者のコアステートメント抽出数を，表 1 の「学習者」の欄に示す．ここでは，学習者は，3 種類の要素の区別を意識せずに抽出しているため，その総数としている．

4.2.3 アルゴリズムと学習者抽出の比較

5 つの Java コードに対する，提案アルゴリズム，学習者，それぞれによるコアステートメント抽出結果を比較する．ここでは，3 つの要素の中で，最もオブジェクト指向言語としての重要な要素である，クラス間連携要素のみとして比較した場合と，3 つの要素すべてで比較した場合について議論する．

1. クラス間連携要素のみでの比較

クラス間連携要素のみを考慮した場合，提案アルゴリズムによるコアステートメントの抽出数は，5 つの Java コード全体で 20 である．これに対し，学習者による抽出は 15 である．これらの中で，同じステートメントを選択しているものは 12 である．このことから，クラス間連携要素のみでの比較では，学習者の抽出したステートメントの中で，80% を提案アルゴリズムでも抽出可能であると言える．

クラス間連携要素のみに限定した場合の提案アルゴリズム，学習者によるコアステートメント抽出結果の比較を，表 2 に示す．ここで，「重複」は両手法で共通のステートメント数，「提案」は提案アルゴリズムのステートメント数，「学習者」は学習者の抽出したステートメント数，「行数」は各 Java コードの総ステートメント数，「重複率」は重複ステートメント数を学習者の抽出したステートメント数で除した値である．

表 2 クラス間連携要素のみでのコアステートメント抽出結果の比較

コード	重複	提案	学習者	行数	重複率
Window	2	3	2	24	100%
BinarySearch	1	3	3	45	33%
Prototype	2	5	2	54	100%
Singleton	2	3	3	32	67%
Builder	5	6	5	40	100%

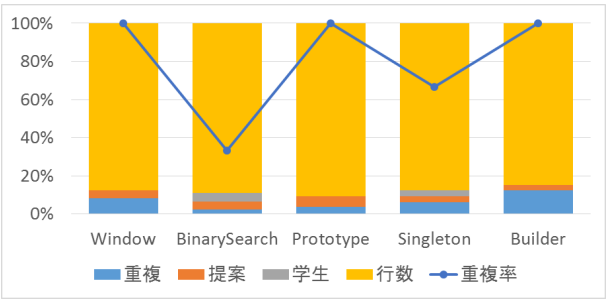

図 5 5 つの Java コードに対するコアステートメント抽出数の比較

表 2 の結果を，そのステートメント数に関するスケールがわかるようにグラフ化したものを，**図 5** に示す．本グラフの横軸は，5 つの Java コードを示す．棒グラフの各部分は，それぞれの Java コードにおいて，重複，提案のみ，学習者のみ，それ以外のステートメント数の割合を示す．また，折れ線グラフは，学習者の抽出したステートメントを提案アルゴリズムが抽出できた割合を示す．この結果より，5 つの Java コードに対して，提案アルゴリズムでは，その 10.8% のステートメントを選択し，学習者による主観的抽出結果の 80% をカバーすることができたと言える．

2. 3 要素での比較

次に，提案アルゴリズムによるコアステートメントの抽出数は，その 3 要素全体を考慮した場合，24 である．これは，全ステートメントの 13.9% となる．しかし，学習者が抽出したステートメントに対するカバー率は 80% にとどまった．

この場合の提案アルゴリズム，学習者それぞれによるコアステートメント抽出結果の比較を，表 3 に示す．ここで，「提案」における括弧内の数は，他の 2 要素を考慮したことで増加したコアステートメント数である．

表 3 コアステートメント抽出数比較 (3 要素)

コード	重複	提案	学習者	行数	重複率
Window	2	4(+1)	2	24	100%
BinarySearch	1	4(+1)	3	45	33%
Prototype	2	6(+1)	2	54	100%
Singleton	2	4(+1)	3	32	67%
Builder	5	6(+0)	5	40	100%

この場合，5 つの Java コードの中で，BinarySearch と Singleton において，重複が少なく，個々の抽出数が多いことから，両者の差が大きいと言える．この理由を分析するため，以下の BinarySearch のコードを調査した．このソースコードで学習者が抽出したステートメントは，2 行目と 3 行目である．これに対して提案アルゴリズムの PDG による抽出では，1 行目のステートメントが選ばれた．実際，バイナリサーチアルゴリズムで最も重要なステートメントは 1 行目と考えられることから，提案アルゴリズムが正解と言える．すなわち，学習者の認識不足が違いの原因と考えられる．

```
int middle = (low + high) / 2;
if (key == table[middle].key) {
  return table[middle].data; // found
```

5. まとめ

本研究では，Java プログラミング学習のためのステートメント空欄補充問題において，空欄とするコアステートメントの抽出アルゴリズムを提案した．Java 学習で学ぶべき 3 つの要素を定義し，それぞれでのコアステートメント抽出手順を明らかにした．提案アルゴリズムの検証として，Java プログラミングの学習者による抽出結果と比較し，80% 以上の一致を見た．

今回，学習者の抽出したコアステートメントには，アノテーションや Java 特有の文法，ライブラリなどが含まれていた．これらは，提案アルゴリズムで除外しているが，今後は，それらの要素も，コアステートメントとして抽出する必要がある．また，複数クラスを有するコードは長い上に，ファイルも一つとは限らないため，JPLAS の Web ブラウザで表示をするには，UML を活用するなどの工夫が必要と言える．

参考文献

[1] 吉田英輔, 角川裕次： テスト駆動開発に基づくプログラミング学習支援システム：初心者開発者のためのセルフトレーニングアーキテクチャ, 信学技報, SS2005-44, pp. 27-32 (2005).
[2] JUnit, 入手先 ⟨http://www.junit.org/⟩.
[3] 渡辺修司： JUnit 実践入門体系的に学ぶユニットテストの技法, 技術評論社, p.308 (2012).
[4] Funabiki, N., Matsushima, Y., Nakanishi, T., Watanabe K. , and Amano, N. : A Java programming learning assistant system using test-driven development method, *IAENG Int. J. Computer Science*, vol. 40, no.1, pp. 38-46 (2013).
[5] Funabiki, N., Korenaga, Y., Nakanishi, T.,and Watanabe, K. : An extension of fill-in-the-blank problem function in Java programming learning assistant system, *Human. Tech. Conf.*(*R10-HTC2013*), pp. 95-100 (2013).
[6] Funabiki, N., Korenaga,Y., Matsushima, Y., Nakanishi, T. and Watanabe, K. : An online fill-in-the-blank problem function for learning reserved words in Java programming education, *Int. Symp. Front. Inform. Sys. Net. Appl.*(*FINA 2012*), pp. 375-380 (2012).
[7] 石原信也, 舩曳信生, 中西透： Java プログラミング学習支援システムにおけるステートメント補充問題機能の実装," 信学技報, ET2013-98, pp. 35-40 (2014).
[8] 柏原昭博, 寺井淳裕, 豊田順一： いかにプログラム空欄補充問題を作るか?, 信学技報, vol. 99, no. 81, pp.9-16 (1999).
[9] 松島由紀子, 笠井康裕, 舩曳信生, 中西透, 天野憲樹： テスト駆動型開発手法による Java プログラミング教育支援システムの提案, 信学技報, ET2009-8, pp. 7-12 (2009).
[10] 塔娜, 舩曳信生, 中西透, 渡邊寛： グラフ理論を用いた Java エレメント補充問題生成アルゴリズムの提案, 電気・情報関連学会中国支部連合大会講演論文集, pp. 303-304 (2013).