第 5 章

オブジェクト指向プログラミング教育のための穴埋め問題とその予備的評価

本章の内容は以下の論文を元に加筆・修正したものである．
M. Murata, N. Kato, M. Ohtsuki, T. Kakeshita, Fill-in-the-blank Questions for Object-Oriented Programming Education and Its Preliminary Evaluation, International Journal of Learning Technologies and Learning Environments, 10 pages, （2023 年印刷中）．

要旨：ソフトウェアの品質を向上させるためには，様々な観点からオブジェクト指向技術が重要である．我々は，プログラミング言語 C の穴埋め問題を提供するプログラミング教育ツール「pgtracer」を開発した．実際の授業で pgtracer を使用して収集したデータを分析することで, C 言語プログラミング教育に有用な知見を得ることができた. 本稿では, pgtracer をオブジェクト指向プログラミング用に拡張するために，Java プログラムの穴埋め問題を開発する．穴埋め問題は，プログラムとトレース表から構成される．プログラムとトレース表はそれぞれ Java のクラスとインスタンスに対応する．トレース表は，オブジェクト指向プログラムの動作を理解する上で重要なインスタンス間のメッセージ送信を表現できる．さらに，解答を要しない穴抜きを導入する．これにより，難易度設定の自由度を高めるとともに，穴抜きを埋めるための学生の作業負担を軽減できる．Moodle の Questions 機能の穴埋め問題(Cloze)問題タイプを使って，学生に穴埋め問題を提供した試行実験の結果を報告する．収集した学生データの分析により，オブジェクト指向プログラミング教育に役立つ知見を得ることができると考えており，今後報告する予定である．

キーワード：Learning Analytics(LA)，プログラミング教育，オブジェクト指向プログラミング，Java，穴埋め問題

1. はじめに

近年，情報技術のさらなる発展に伴い，多くのアプリケーション領域で IT を活用したサービスが提供されている．これらのサービスが高度化・複雑化する中で，ソフトウェアの品

質向上や開発効率向上のために，オブジェクト指向プログラミングの重要性は高い．

大学や技術研究所では，ソフトウェア技術者育成のためにオブジェクト指向プログラミング教育が行われているが，時間や教員の不足から，十分なプログラミング演習を行うことが難しい．そこで私たちは，プログラミング演習を支援するために，穴埋め問題を活用した教育支援ツール pgtracer[1,2]を開発し，収集したデータを活用して学習分析（LA）を行っている．穴埋め問題は，穴抜きの位置や大きさによって問題の難易度を簡単に調整でき，個々の穴抜きごとに解答ログを収集できる利点がある．収集したログを LA で活用することで得られた知見は学習効果の向上への活用が期待できる．

本研究の目的は，Java プログラムの穴埋め問題を pgtracer に適用し，学生の学習過程や成果に関する知見を獲得し，Java プログラミング教育を支援することである．本稿では，Java プログラムの穴埋め問題を拡張した問題を提案する．提案した問題を実際の授業で学生に提供し，評価した結果を報告する．

pgtracer の穴埋め問題は，Java プログラムとトレース表から構成される．トレース表はインスタンスに対応し，インスタンスの変数値や出力値を表現できる．我々の研究の革新的な点は，穴抜きがプログラムだけでなくトレース表にも定義できることである．これまでの C 言語プログラミング版の実験を通じて，トレース表の理解度が高い学生は，プログラムの習熟度も高いことが分かっており，トレース表による変数値の変化の可視化は，プログラムの習得に有効である[3，4]．また，Java のプログラムを理解するためには，オブジェクト間のメッセージ送受信をトレースすることが重要である．そこで，本稿では，メッセージ送受信を表現するためにトレース表を拡張する．

プログラムやトレース表の中で穴抜きを定義すると，プログラムやトレース表の他の部分がヒントとなる場合がある．このようなヒントを隠すために，多くの穴抜きが必要となる場合がある．しかし，これでは穴抜きが増え，学生の学習意欲を低下させる可能性がある[4]．そこで，学生が解答する必要のない特殊な穴抜きを導入する．これにより，ヒントを隠しつつ，問題の難易度をより柔軟に制御できる．

問題プログラムには，GoF デザインパターン[5]に掲載されている 12 個の Java プログラムを利用する．これらのプログラムは，Java プログラミング言語とオブジェクト指向設計の機能を十分に活用しているため，オブジェクト指向プログラミングの教育に適している．これらのプログラムを 3 つのレベルに分類する．問題難易度は，穴抜きの場所を適切に設定することで調整する．

提案した穴埋め問題を実際の授業で活用し，収集したデータに対して LA 法を適用することを計画している．授業は，佐賀大学理工学部計算科学科の 3 年次に開講される「プログラミング演習Ⅲ」である．Java プログラム用の pgtracer はまだ開発中であるため，Moodle のアクティビティの一つである Quiz を使って試行実験を行う．収集したログの解析とアンケート結果から，穴抜きの種類による難易度の違いや，トレース表の概念に対する学生の理解度を検証し，提案した問題の妥当性を検証する予定である．

本論文は以下のように構成されている．第 2 節では関連研究について述べる．第 3 節では pgtracer の機能を紹介する．第 4 節では，プログラムの表現方法とプログラム内の穴抜きを定義する．第 5 節では，トレース表の表現とトレース表内の穴抜きについて定義する．第 6 節では，穴埋め問題の難易度の調整方法について提案する．第 7 節では実験対象 t の授業と試行について述べ，第 8 節では試行の結果について述べる．第 9 節では，アンケートの結果について述べている．最後に結論を述べる．

2. 関連研究

オブジェクト指向プログラミングの教育ツールは，様々な目的で開発されている．

Hsiao らは，Java 用の Web ベースのパラメータ化された問題を提案した[6]．パラメータ化された問題は，変数の最終値や表示されるテキストを解答する．pgtracer は，最終値だけでなく，プログラムの各ステップの変数値にも穴抜きを設定できる．最終的な正しさだけでなく，各ステップでの変数値もトレースできることは，プログラムの理解度を推定するのに有効である．

Truong らは，Java プログラムの静的解析フレームワークを紹介した[7]．学生は与えられた問題のプログラム全体を解答する．このシステムでは，模範解答とあらかじめ定義された差分を解析に利用する．pgtracer では，プログラム全体を穴抜きと定義することで，同様の問題を作成できる．さらに，pgtracer では，全学生の解答ログを分析することで，想定以外の誤答の傾向を発見できる．

Hauswirth らは，Java プログラミングを教えるために Informa クリッカーシステムを提案した[8]．Informa は，多肢選択問題などプログラムコードに関連するいくつかのタイプの問題を提供するが，変数のトレースには対処していない．

Funabiki の研究グループは，穴埋め問題を利用した Java のプログラミング教育システムを提案している[9,10,11]．これらの問題では，プログラム文の一部，文全体，出力値のいずれかを穴抜きとして定義している．しかし，プログラム実行の各ステップにおけるメッセージ送受信や変数値を扱うことはできない．本提案問題では，これらの部分についても空白を定義できるため，より柔軟な穴抜きの設定が可能である．

Java プログラムを用いた LA に関する研究の多くは，プログラムエラーやコンパイルエラーを分析している．Edwards らは，学生が書いた Java のプログラムで発生した静的解析エラーを分析し，Checkstyle と PMD を用いて検出した．また，最も一般的なエラーなどの知識を得ている[12]．McCall らは，学生プログラムのエラーを分析し，エラーの分類とその頻度について研究している[13]．Altadmri らは，25 万人以上の学生の 1 年分のコンパイルイベントを用いて，エラーの頻度，修正までの時間，ユーザー間のエラーの分布などを分析している[14]．

pgtracer は，最終的な解答だけでなく，解答の過程で生徒が入力した解答も収集する．こ

れらのデータを利用することで，最終解答に至るまでの生徒の行動を分析できる．

3. プログラミング教育支援ツール pgtracer の穴埋め問題活用

Pgtracer は，穴埋め問題を作成する機能，問題を提供し，採点する機能，ログを収集し分析する機能の大きく 3 つの機能を提供する．

A. 穴埋め問題

穴埋め問題は，プログラム，トレース表，プログラム用マスク，トレース表の 4 つの XML ファイルから構成される．pgtracer は，これらの XML ファイルを自動的に作成する機能を提供する．トークン，連続したトークン，式，文は，プログラム内の穴抜きとして設定できる．トレース表では，変数値，ステップ番号，変数名を穴抜きとして設定できる．我々は，穴抜きの位置によって問題の難易度を制御できることを明らかにした[1,2].

B. pgtracer の機能

pgtracer は様々な機能を提供する．最初の機能は，穴埋め問題を編集する．pgtracer は，プログラム，トレース表，プログラムのマスク，トレース表の XML ファイルを生成する機能を提供する．

2 番目の機能は，穴埋め問題の出題と自動採点である．pgtracer は，生徒の解答により穴埋めされたプログラムを実行し，プログラムの実行過程を正しいトレース表と比較する．

また，pgtracer は，各穴抜きを埋めた直後の生徒の解答，正答，所要時間などを自動的に収集する．pgtracer は，収集したログに対して，学生ごとの分析機能，問題ごとの分析機能，穴抜きごとの分析機能，生徒の詳細な学習過程の分析機能など，様々な視点からの分析機能を提供する[2]．これらの分析機能は，学生の達成度や学習過程の分析に役立つ．

C. pgtracer を用いたプログラミン教育プロセス

図 1 は，pgtracer を活用したプログラミング教育のプロセスを示す．教師は正しいプログラムと入力ファイルを用いて穴埋め問題を作成する．学生は，問題データベースから問題を選び，解答する．pgtracer は，生徒の解答を自動採点する．同時に，pgtracer は解答ログを収集する．学生と教員は，分析機能を使って，解答ログを分析できる．解析した結果を用いて，教員は問題データベースを改良し，学生個人またはクラス全体にフィードバックできる．

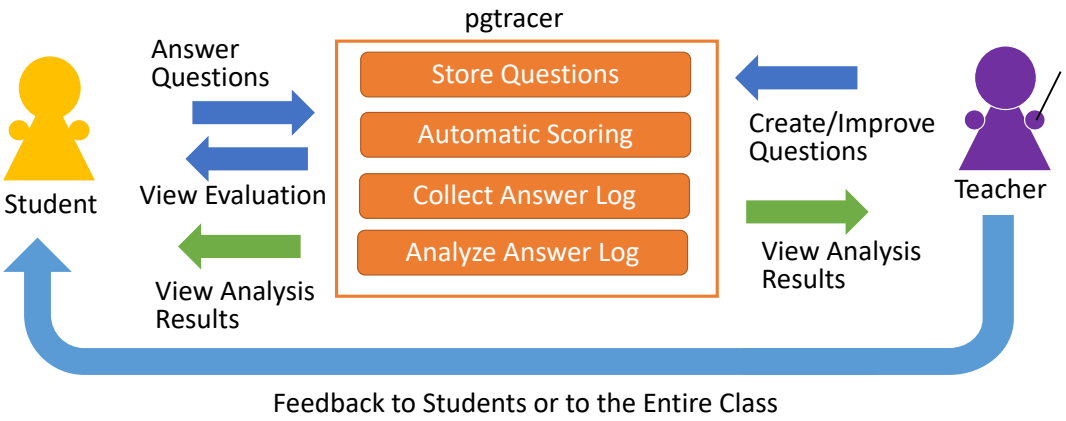

図 1：pgtracer を利用したプログラミング教育プロセス

4. 穴埋め問題のためのプログラム

ここでは，プログラムの表現方法とプログラム内の空白の定義について説明する．図 2 はその例である．

```
    public class Main {
        public static void main(String[] args) {
            // 'H'を持った CharDisplay のインスタンスを 1 個作る。
1           AbstractDisplay d1 = new  (1) ('H');

            // "Hello, world."を持った StringDisplay のインスタンスを 1 個作る。
2           AbstractDisplay d2 = new      ("Hello, world.");

            // "こんにちは。"を持った StringDisplay のインスタンスを 1 個作る。
3           AbstractDisplay d3 = new  (2) ("こんにちは。");

            // d1,d2,d3 とも、すべて同じ AbstractDisplay のサブクラスのインスタンスだから、
            // 継承した display メソッドを呼び出すことができる。
            // 実際の動作は個々のクラス CharDisplay や StringDisplay で定まる。
4           d1.  (3) ();
5           (4) ;
6           d3.display();
        }
    }
```

図 2：穴抜きを含むプログラムの例（テンプレートメソッド，メインクラス）

A. プログラムの表現

Java プログラムの最も上位の構成要素はクラスであるため，1 つのクラスに対して 1 つのソースファイルが存在する．したがって，Java プログラムの穴埋め問題には，一般的に複数のソースファイルが含まれる．また，インスタンス変数やローカル変数の定義など，初期化文が含まれる場合があるため，各文章にステップ番号を付した．ステップ番号は以下のルールに従って付され，第 5 節で説明するトレース表のステップ番号に対応する．

- インスタンス変数定義を含む各文に，連続したステップ番号を割り当てる．
- メソッド定義の文に，1 から始まる連続したステップ番号を割り当てる．
- Java で定義された，代入文，メソッドコール，if, else, switch, case, default, while, return

などの制御文に，それぞれ連番を付与する．

- ネスト構造を持つ複合文には，「x.y」のようなステップ番号を割り当てる．

プログラムは，正常にコンパイルされなければならない．また，Java プログラムのガイドラインを以下のように定義する．

- 入れ子構造を正しく表現するためにインデントを使用する．
- クラス定義の前に，クラスの概念や特徴を説明する独立したコメントを記載する．
- メソッドの定義の前には，そのメソッドが提供する機能を説明する独立したコメントが付けられている．
- 変数定義には，変数に格納された値を説明する同じ行のコメントが付きます．
- 一連の発言の前には，その意図やアルゴリズムを説明する独立したコメントが付きます．
- 独立したコメントの前には空白行があります．
- 可能な限り 1 つの文言のみを 1 行で記述する．
- 変数名と関数名はキャメルケースの慣例に従いる．ただし，クラス名の最初の文字は大文字にする．

B. プログラムの穴抜き

トークンとは，変数名，クラス名，演算子，キーワードなど，それ以上細分化できない文字列のことです．コメントは，トークンとして定義されます．Pgtracer では，任意のトークンの並びで空白を定義することができます．したがって，文の一部，文全体，連続した文の列を空白として定義することができます．

ブランクには 2 種類あります．一つは答えが必要なもの，もう一つはそうでないものです．後者の穴抜きを「無視できる穴抜き」と呼ぶことにした．無視できる空白を導入することで，空白の難易度をより柔軟に制御できる．異なる対象に対して，同じようなメッセージを繰り返し送る場合を考えてみよう．このような文の中に空白を設けると，空白に近い文が空白を埋めるヒントになることがあります．他の文の中に適切な無視できる空白を定義することで，受験生が埋めなければならない空白の数を増やすことなく，ヒントとなるトークンを隠すことができるのです．これにより，問題の難易度をより柔軟にコントロールできる．

無視できる空白を使ってコメントを隠すことも可能なので，問題の難易度をコントロールすることも可能です．空白と無視できる空白を区別するために，これらの空白の背景は異なる色で塗りつぶされている．数字のない空白は，図 2 の無視できる空白を表する．

5. 穴埋め問題のためのトレース表

A. トレース表の表現

Java のプログラムは，オブジェクト間のメッセージ送受信によって実行される．トレース表を実行ステップの順番で表現すると，表が複雑になってしまう．そこで，オブジェクトごとにトレース表を定義し，メッセージの受け渡しをトレース表で表現する．この方法は，シーケンス図のようなオブジェクト指向プログラミングの概念にも合致する．提案するトレース表を以下に定義する．図 3 に例を示す．

Caller of the Method			Step of Called the Method			Argument of the Method	Instance Value	Local Variable of the Method	Output / Return Value of the Method
Caller of			Class	method	step	char	char	int	output
object	method	step				ch	ch	i	
Main	main	1	CharDisplay		1				
Main	main	1	CharDisplay	CharDisplay	1	H	H		
Main	main	(1)	AbstractDisplay	display	1		H		
			(2)	open	1		H		<<
			AbstractDisplay	display	2		H	0	
			AbstractDisplay	display	2.1		H	0	
			(3)	print	1		H	0	H
			AbstractDisplay	display	2		H	1	
			AbstractDisplay	display	2.1		H	1	
			(4)	print	1		H	1	H
(omitted)									
			AbstractDisplay	display	2		H	5	
			AbstractDisplay	display	3		H		
			(5)	close	1		H		>>

図 3：穴抜き含むトレース表の例
（テンプレートメソッド，インスタンス ID＝charDisplay#1）

インスタンス識別子の表現方法

ここでは，インスタンスと Main メソッドごとにトレース表を定義することにする．インスタンスに対応するトレース表はインスタンス名を持ち，Main メソッドを表すトレース表は"Main"という名称を持つ．

各インスタンスは，クラスの各インスタンスを区別するために，"ClassName#X"と表現される．ここで，"ClassName"はクラス名の先頭文字を小文字に変更した文字列であり，"X"は生成順序を表すシーケンス番号である．例えば，最初に生成されたクラス"CharDisplay"のインスタンスの識別子は，"charDisplay#1"となる．

トレース表の列

オブジェクトのトレース表は表 1 の列を持ち，Main メソッドのトレース表は"Caller　of the　Method"を除いて同じ列を持つ．メソッドの呼び出し元」列は，呼び出し元のオブジェクトを記述する．このオブジェクトは，コンストラクターの呼び出しによって生成される．オブジェクトの生成後，他のオブジェクトはそのオブジェクトにメッセージを送り，何らかの処理を実行できる．

トレース表では，現在のオブジェクトがオブジェクトを作成するか，オブジェクトにメッセージを送信すると，すべてのオブジェクトが「外部オブジェクト」列の小項目としてリストアップされる．小項目の値は，呼び出されたメソッドの名前である．これによりプログラ

ム内で生成された各オブジェクトの生存期間とメッセージの流れを把握できる．

静的メソッドを含むクラスのトレース表は，クラス名に名前が付けられる．トレース表には，表 1 に示す項目の他に，「クラス変数」という項目と「データ型」という小項目がある．

表 1：各オブジェクトのトレース表の列

列	小項目	価値観
メソッドの呼び出し元	オブジェクト	インスタンス名またはクラス名
	方法	メソッド名
	ステップ	#ステップ数
メソッドを呼び出すステップ	クラス	メソッドが定義されているクラス名
	方法	メソッド名
	ステップ	コードのステップ番号
メソッドの主張	上段：データ型，クラス名 下段：引数名	対応するコードが実行されたときの引数値
インスタンス変数	上段：データ型，クラス名 下段：変数名，インスタンス名	値またはオブジェクト識別子
メソッドのローカル変数		
外部オブジェクト	上段：クラス名 下段：インスタンス名	オブジェクト識別子
メソッドの出力／戻り値	メソッドの戻り値	戻り値または戻りオブジェクト識別子
	出力	出力文字列

複数のメソッド呼出し

オブジェクトは複数回呼び出されることがある．各メソッド呼び出しの一連の処理を区別するために，トレース表には二重線を引く．具体的には，return 文に対応する行の下に二重線を引く．

複数の演算を含む文表現

Java では 1 行のコードに複数の操作を含めることができる．例えば，次のコードは 2 つの操作を含んでいる．

```
AbstractDisplayd1=new CharDisplay('H')；
```

一つはクラス"CharDisplay"のインスタンスの生成，もう一つは生成されたインスタンスを変数 d1 に代入である．

各操作を理解することは，プログラムを理解するために必要である．そこで，トレース表では，この 2 つの操作を実行順に 2 行で表現する．ここで，ステップ番号はすべて同じと

する．トレース表のある行で表される操作は，インスタンス変数や外部オブジェクトの列を参照することで識別できる．

変数値の表現法

変数のデータ型が整数などの基本データ型の場合，格納されている値が表示される．インスタンスが格納されている場合，対応するインスタンス識別子が表示される．

プログラム内で変数に領域が割り当てられていない場合や，変数が不定形の場合は何も表示されない．この 2 つのケースは厳密には異なるが，実験対象とする授業ではこれらを厳密に区別することを学生に求めていない．オブジェクト指向プログラミング教育の初級レベルでは，このような微妙な違いを提示することは，学習者の混乱を招き，理解の妨げになると考えられる．そこで，2 つの場合を同じ表記で表現することにした．

変数値の表現法

繰り返し回数が多いとトレース表がかなり大きくなり，理解が困難となる可能性がある．そこで，中間ステップを省略して反復処理を記述することにした．省略された行を明確にするために，（省略）と記述した行を挿入する．

B. トレース表内の穴抜き

トレース表では，各セルの値を空白として定義できる．具体的には，変数値，出力値，ステップ番号，オブジェクト識別子，クラス名，メソッド名などである．また，メソッド引数，インスタンス変数，ローカル変数，外部オブジェクトについては，その小項目，データ型やクラス名，変数名，オブジェクト名について穴抜きを定義できる．

プログラムの場合と同様に，トレース表で解答を要しない無視できる空白を定義できる．トレース表では，変数の状態は新しい値が代入されるまで変化しない．そのため，プログラムから変数値を推測させたい場合でも，前後の行の値から推測できる場合がある．これを防ぐために，穴抜きの前後の変数値も空白として定義する必要があった．しかし，空白が増えることで入力回数が増えるため，問題の難易度と回答の面倒さを混同する学生もおり，問題の難易度を検証する上では適切ではなかった．これにつては，解答を必要としない無視できる空白を導入することで解決できる．

6. 穴埋め問題の作成

本節では，穴埋め問題の開発方針を提案する．作成した問題は，佐賀大学の計算機コースの 3 年次に提供される「プログラミング演習Ⅲ」で提供する予定である．1 年次には Python を，2 年次には C++を用いた構造化プログラミングを学び，基本的なプログラミング技術や基本的なソートや整列のアルゴリズムを習得する．プログラミングⅢ」では，Java を用い

たオブジェクト指向プログラミングの学習を開始する．

A. 対象とする講義

対象講義は，効率と高品質を維持するために，ソフトウェア開発の実践現場で使用される様々なツールの目的と使い方を学ぶことを目的する．具体的には，統合開発環境（IDE）Eclipse を使用して Java 言語でソフトウェアを開発し，JUnit や Jenkins などのツールの使用方法を学ぶ．

最初の 1 週間は，IDE についての講義と Java の基本的なプログラミングを練習する．その後，Git と GitHub への登録，ソフトウェアテスト技術とユニットテスト設計，JUnit，Jenkins，UML 図，デザインパターンを学びます．最後の 4 週間では開発演習を行う．この授業計画と並行して，本稿で提案する穴埋め問題を提供し，Java プログラミングの習得を目指す．

B. 穴埋め問題の提供

授業内容に対応した演習問題を提供することで，受講者の理解を深めることを目的とする．問題は，『Java で学ぶデザインパターン入門』から 12 個のトピックを選び，そのサンプルプログラムを用いて作成した[15]．トピックは授業で扱う内容を考慮して選択し，問題レベルはトピックの内容や授業の順番に合わせて初級，中級，上級と設定した（表 2）．

表 2：提供する穴埋め問題のテーマ

順序	難易度	デザインパターン	教科書の章
1	初級	Template Method	3
2		Factory Method	4
3		Iterator	1
4		Composite	11
5	中級	Decorator	12
6		Strategy	10
7		Abstrac tFactory	8
8		Observer	17
9	上級	Adapter	2
10		Builder	7
11		Command	22
12		Visitor	13

C. 穴埋め問題の作成方針

これまでの研究を通じて，C 言語の穴埋め問題に関して以下のような知見を得ている[3,4]．

- トレース表の中だけで定義された空白を含む問題が最も易しくで，次いでプログラムの中だけで定義された空白を含む問題が易しい．トレース表とプログラムの両方に空白が定義されている問題は，学生にとって最も難しい．
- プログラム内の空白は，個々のトークン，トークンの並び，文全体の順で難易度が上がる．
- トレース表内のブランクは，ステップ番号，変数値，変数名の順に難易度が高くなる．
- コメントを付与することで問題の難易度が下がる
- 設問に定義された穴抜きの数が増えると，学生の回答意欲が低下する．

学習者は 2 年次までの授業でプログラミングの基礎は学んでいるため，オブジェクト指向プログラミングの基礎やデザインパターンの学習棚目の問題とした．

プログラム内の空白は，主にクラス定義など Java 特有の文法の理解度確認とメッセージ送受信の実行フローを確認するために定義する．また，トレース表内の空白は，主にクラスやメソッド名，ステップ番号など呼び出し側のオブジェクトに関する事項や，呼び出されたオブジェクトに関する項目，メッセージの送受信に伴って変化する値などを定義する．

これらを考慮し，以下のような方針で，話題のレベルを適切に設定した問題を作成した．

(1) 1 問につき 10 個程度の空白を定義する．ここで，無視できる空白は含めない．
(2) 1 つのテーマに対して，難易度の異なる 2 つの問題を作成する．
(3) 各設問の教育目的を明確にする．
(4) 各穴抜きの意図を明確にする．
(5) プログラム内の空白を定義するために，初級レベルでは個々のトークンを基本として使用する．中級，上級レベルでは，より長いトークン列を使用する．
(6) トレース表内の空白を定義するために，初級レベルでは主に変数値を使用する．中級レベルでは，インスタンス識別子やメソッドの空白を多く使用する．

これまでの経験から，問題の穴抜きが多すぎると学生のモチベーションが低下し，pgtracer の継続利用ができなくなることが分かっている．前回の実験では，対象となる学生が C 言語の初心者であること，プログラムの総行数が 20 行程度であることから，1 問につき 5 つの空白を定義していた．しかし，本稿では，受講生のプログラミング習熟度が高いと予想されるため，1 問あたり約 10 個の穴抜きを定義した．また，同じ理由でプログラムの総行数も大きくなっている．

方針(2)は，問題の種類による難易度の違いを調査するために設定する．さらに，方針(3)，(4)で明確にした設問の教育目的，各穴抜きの意図を学習ログの分析に活用する．方針(5)，(6)は，初級，中級，上級の違いを明確にするために設定する．

C. 穴埋め問題の例

表 3 は，「テンプレートの方法」で説明したプログラムをもとに作成した問題の教育目的

と，問題の一部の穴抜きの意図を示している．作成した問題を図 2 に示す．

穴抜き(1)と(2)について考えてみると，変数 d1，d2 は抽象クラス「AbstractDisplay」の変数なので，ユーザーは「AbstractDisplay」のサブクラスである「CharDisplay」と「StringDisplay」のいずれかのコンストラクターを呼び出すことを理解し，引数から適切なコンストラクターを導出する必要がある．コメントから適切なコンストラクターを導出することは比較的容易であると思われる．なお，ステップ 2 のコードは，穴抜き(2)と同じ処理であるため，示していない．

穴抜き(3)と(4)は，メソッド呼び出しの文中に定義されている．穴抜き(4)については，出力値からインスタンスを導出することが必要である．さらに，トークンの列ではあるが，コメントとステップ 6 の文から導出できる．

このように，トークン，トークンの並び，文全体といった穴抜きの長さと，コメントの表示・非表示を利用して，穴抜きの難易度をコントロールできる．この問題は，穴抜きをそのままにしてコメントを非表示にすると難易度が上がる．

表 3：プログラム内の一部のブランクの教育目的と意図
（テンプレート方式）

教育目標	抽象クラスと抽象メソッドの使い方を認識する．

ブランク #	正解	ブランクのある方の意図
(1)	CharDisplay	コンストラクターの呼び出しを理解する．
(2)	StringDisplay	コンストラクターの呼び出しを理解する．
(3)	display	メソッドコールを記述できる．
(4)	d2.display()	出力からインスタンスとメソッドを導出できる．

7. トライアル実験

試行が行われた講義は「プログラミング演習Ⅲ」である．この講義の対象者は，佐賀大学の 3 年生である．受講者数は 76 名である．この試みは，2021 年度の前期に実施した．

本講義では Java 言語を学習するが，この講義内では言語の詳細な説明は行わず，基本を説明した後の補助教材として，外部のオンライン学習サイトでの自習を推奨している．

A. Moodle を使った試行

実験では，pgtracer が開発中であったため，Moodle の Questions 機能である穴埋め問題(Cloze)を使用した．そのため，いくつかの制約があった．

まず，C 版の pgtracer と異なり，問題の自動生成機能がないため，手動で穴埋め問題を

作成し，画像として提示した．次に，C 版の pgtracer は，コンパイルして実行できるため，穴抜き定義の自由度が高かったが，埋め込み回答（Cloze）問題タイプは，短い文章しか設定できず，正規表現も使えないため，設定できる穴抜きの自由度に制限があった．

受験回数の制限も可能であるが，今回の試行では受験回数を 2 回に設定した．しかし，複数回の受験を許可した場合，答えを覚えてしまい，次の試験でその答えをコピーする学生もいた．そこで，2 回受験した場合は平均点で評価することで，1 回目の受験でもまじめに取り組む仕組みとした．

C. 試行に利用した問題

各週に，表 4 に示す問題を出題した．問題 ID の PR や TR などの接尾辞は，それぞれ問題の種類がプログラムであること，トレース表であることを示す．右の列は，初回試行の受験者数である．

Java 言語に慣れていない学生もいたため，最初の週はごく基本的なプログラムとトレース表の問題を提示した．その後，最も単純な構造のデザインパターンを提示した．同じ問題に対してプログラムとトレース表を同時に提示するのは，トレース表がプログラム全体を提示するため，問題があった．また，他の演習がある場合，学生の負担が大きくなってしまう．そこで，トレース表とプログラムを別の週に分け，プログラムの提示方法をトレース表の前に提示するように調整した．

トレース表については，Iterator パターンのトレース表が提示された際に，簡単なプログラムでの回答方法を説明した資料を用意した．Iterator パターンについては，講義中にプログラムの説明を行ったので，プログラムに関する問題は省略した．Factory メソッドパターンのプログラムの穴埋め問題は，プログラムに問題が見つかったため，講義終了後に再度掲載した．Composite トレース表の問題の記述に誤りがあったため，その後，プログラムを先に掲載した．

表4：問題と受験者数

週	デザインパターン	難易度	問題の種類	問題 ID	穴抜き数	受験者数
1	for 文		プログラム	Ex01_PR	6	67
			トレース表	Ex01_TR	4	
6	Template Method	初級	プログラム	Ex06_PR	12	71
		初級	トレース表	Ex06_TR	12	
7	Factory Method	初級	プログラム	Ex07_PR	9	58
8	簡単な例		トレース表	Ex08_SP	2	67
	Iterator	初級	トレース表	Ex08_TR	10	
10	Factory Method	初級	トレース表	Ex10_TR	9	61
11	Composite	初級	プログラム	Ex11_PR	11	69

週	デザインパターン	難易度	問題の種類	問題 ID	穴抜き数	受験者数
13	Strategy	中級	プログラム	Ex13_PR	10	66
14	Observer	中級	プログラム	Ex14_PR	10	65

8. 解答結果

ほとんどの受講生が繰り返し挑戦した．2 回目の受験ではすでに正解が公開されているため，以下の分析では 1 回目の受験の結果のみを掲載する．Ex01_PR，Ex01_TR，Ex08_SP は，トレース表の解答方法や考え方を説明する問題である．したがって，これらの問題を除いて議論する．

表 5 は，問題タイプが「プログラム」である穴埋め問題において，穴抜きを 1 つのトークンと複数のトークンを含むトークン列の 2 種類に分類し，それぞれの穴抜き数と平均正答率を示している．初級，中級ともに，複数のトークンを含む穴抜きの平均正答率は，トークンを含む穴抜きの平均正答率より低い．これは，複数のトークンを含む穴抜きは，学生にとって回答が難しいことを示している．1 つのトークンからなる穴抜き，複数のトークンを含む穴抜きのいずれにおいても，中級レベルのほうが初級レベルよりも平均正答率が低い．中級レベルでは，プログラムの処理フローを理解していないと正解を導き出せないような穴抜きを設定した．そのため，1 つのトークンからなる穴抜きであっても，出題者の意図する難易度を反映させることができた．

表5：種類が"プログラム"の問題の穴抜き数と正答率（%）

難易度	問題数	ブランクのすべて		1 つのトークンからなるブランク		複数のトークンを含むブランク	
		穴抜き数	正答率の平均値 (%)	穴抜き数	正答率の平均値 (%)	穴抜き数	正答率の平均値 (%)
初級	3	32	70.6	22	76.5	10	57.7
中級	2	20	55.8	11	67.2	9	41.9

表 6 に問題タイプが「トレース表」である穴埋め問題の穴抜き数と平均正答率を，穴抜きの種類別に分類して示す．表 6 から，メソッド呼び出し元のステップ番号とインスタンスを解答する変数値の平均正答率が低いことがわかる．このことから，オブジェクト指向特有のメソッド呼び出しやインスタンスに関する問題は，学生にとって難しいものであると考えられる．また，第 9 章で述べるように，複数のトレース表やプログラムが存在することによるトレースの難しさにより，正解に至らなかった学生もいたと考えられる．

表 6：タイプが"トレース表"の問題の穴抜き数と正答率（%）.

穴抜きの種類	穴抜き数	正答率の平均値(%)
コンストラクタやメソッドの呼び出し	16	72.5
呼び出しメソッドのステップ番号	3	46.0
コールドスペース	2	61.7
戻り値（インスタンス）	5	64.0
変数値（インスタンス）	1	38.8
変数値（基本型）	4	71.1

9. アンケート結果

講義の最終日に，学生に対してアンケートを実施した．回答者を特定できるように，Moodle のアンケートモジュールを使用した．学生には，回答内容によって不利益を被ることなく，率直な意見を書き込むことができることを告知した．回答数は 69 件であった．
アンケート項目とそれぞれの ID を表 7 に示す．ENQ01 から ENQ08 までの 8 問は，表 8 に示すように 5 段階の回答選択肢を持つ．ENQ09 のみ自由記述式である．

表 7：アンケートの項目

QID	質問
ENQ01	正解のプログラム自体の動作は理解できたか.
ENQ02	トレース表の概念は理解できたか
ENQ03	インスタンス変数とローカル変数のトレースができたか
ENQ04	メッセージの送受信の流れをトレースできたか
ENQ05	穴埋め問題の難易度はどうですか
ENQ06	1 問あたりの穴抜きの数はどうですか
ENQ07	プログラミングへの興味や学ぶ意欲はありますか
ENQ08	プログラミングの学習において，穴埋め問題は有効だと思うか.
ENQ09	改善点や穴埋め問題へのご意見があれば，ご自由に記入してください

表 8：アンケート項目に対する回答選択肢の説明

QID	選択肢				
	1	2	3	4	5
ENQ01	ほとんど理解できていない	あまり理解できていない	理解できた	ほぼ理解できた	しっかりと理解できた
ENQ02					
ENQ03	トレースできていない	あまりトレースできていない	トレースできた	ほぼトレースできた	正しくトレースできた
ENQ04					
ENQ05	とても難しい	やや難しい	ちょうどよい	やや易しい	とても易しい
ENQ06	とても多い	やや多い	ちょうどよい	やや少ない	とても少ない
ENQ07	ほとんどない	ややない	どちらでもない	ややある	とてもある
ENQ08	ほとんど思わない	あまり思わない	どちらでもない	ややそう思う	とてもそう思う

図 4 は，ENQ01 から ENQ08 までの 5 段階の評価結果をしめす．グラフ要素のラベルはパーセンテージを示す．ENQ01 の「プログラムの内容を理解している」，ENQ02 の「トレース表の概念を理解している」については，それぞれ 69.5%，68.1%が「理解している」と回答している．一方，トレース表（ENQ03，ENQ04）において，変数やメッセージの送受信がトレースできていないと答えた人がそれぞれ 39.1%，42.0%であった．

これは，プログラムそのものやトレース表の概念は理解できたものの，変数やメッセージの送受信をトレースすることは難しかったと考えられる．この難しさは，問題の表示形式に原因があると考えられるので，表示形式を改善する方法を検討する必要がある．

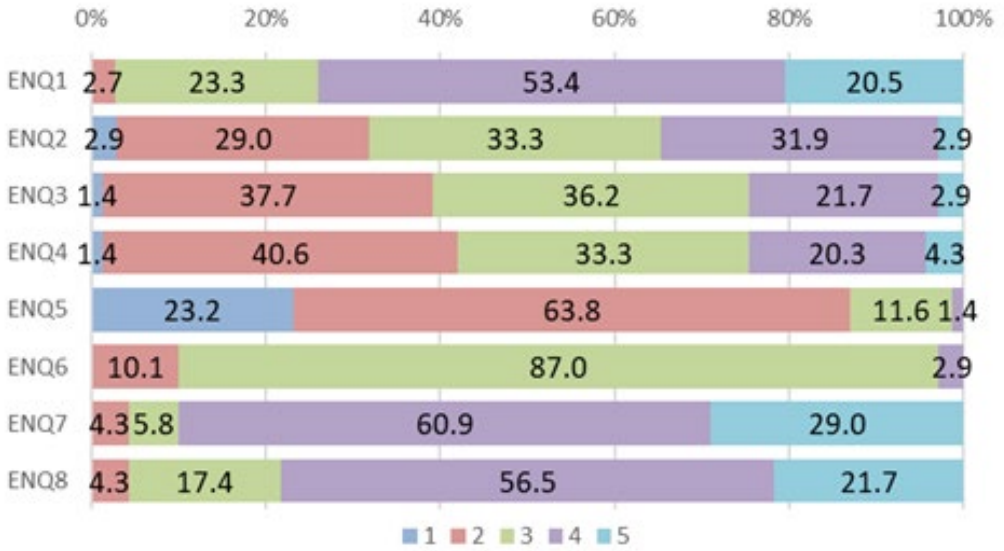

図 4：ENQ01～ENQ08 のアンケート結果

穴埋め問題そのものについては，63.8%の人が「やや難しい」と回答している（ENQ05）．穴埋め問題の量自体は，87.0%の人が適切だと回答している（ENQ06）．

ENQ05 を考察すると，63.8%の学生が「問題がやや難しい」と回答している．このことから，問題の難易度は適切であると考えられる．ENQ06 を考えてみると，87.0%の学生が「穴抜きの数がちょうどよい」と回答しており，問題の難易度は適切であると考えられる．本稿では，解答を求めない穴抜きを導入した．これにより，ヒントとなるコードや変数の値を隠しつつ，適切な解答数の穴抜きを設定することができたと考えられる．したがって，穴抜きを埋めるという学生の作業負担を軽減し，モチベーションの低下を防ぐことができた．

さらに，89.9%が「プログラムに興味があり，学習意欲がある」（ENQ07），78.2%が「穴埋め問題が学習に役立った」（ENQ08）と回答している．これらの結果は，学生が今回の試行実験に真剣に取り組んだことを示しており，ENQ01 から ENQ06 までの回答は信頼できるものと考えられる．

最後に，自由記述項目 ENQ09 から得られた学生の 14 の意見について考察する．プログラムやトレース表の表示方法に関する回答は 5 件あった．今回の試行実験では，プログラムとトレース表は複数のタブに画像として表示したが，プログラムや送受信したメッセージのトレースを理解する際にタブを切り替える必要があり，理解の妨げになった可能性がある．また，解答入力に関する回答が 3 件あった．入力時の大文字と小文字の違いによる不正解を訴えたものである．しかし，実際のプログラミングでは，小文字と大文字の違いが致命的なバグになることもあるため，教育的な配慮をして正しい入力を求めるべきであると考える．

10. 考察

プログラム内の穴抜きやトレース表を作成する際に，それぞれの穴抜きの意図を明確にした．その結果，Java の文法，デザインパターン，処理の流れ，それらの組み合わせなどの知識を得ることで正解にたどり着けることが分かった．この知識をさらに分類し，受講者の解答ログと合わせて分析することで，各穴抜きの難易度を推定できると考えられる．

問題作成は，複数のプログラムファイルやトレース表のクロスチェックが必要であり，複雑である．そこで，メソッド呼び出しの文からそのメソッドを定義するプログラムへ，あるいはインスタンス識別子からそのインスタンスのトレース表へのナビゲーション機能が必要と考える．

穴抜きの前後には，トレース表の値が同一である，プログラムの説明が似ているなどのヒントがある場合がある．このような部分を，解答を要としない無視できる穴抜きを導入することで隠すことができる．これにより，解答を必要とする穴抜きの数を制御できる．

本稿では，連続したトークンを 1 つの空白と定義したが，各トークンを 1 つの空白と定義することも可能である．そうすれば，学生はトークンの数をヒントとして得られるので，

問題の難易度が低下する．複数のトークンを含む穴抜きを解けない学生には，トークンの数をヒントとして提供することができる．

11. 結論と今後の課題

本論文では，Java 言語における穴埋め問題を提案した．問題の難易度は，穴抜きがプログラム内かトレース表内か，また穴抜きの長さによって制御できる．また，解答を要としない無視できる穴抜きを導入することで，問題の難易度をより柔軟に制御できる．

今回の試みでは，先の研究で考案した Java プログラミングの穴埋め問題の一部を，学生に提供した. pgtracer そのものを拡張するのではなく, Moodle の質問機能を利用したため，問題の提示や判定方法に問題があった．しかし，これらの問題は，今後 Java 版を実装することで改善される．

私たちの研究グループでは，穴埋め問題を提供するプログラミング教育支援ツール「pgtracer」を開発している．今回提案する問題を pgtracer に適用し，実際の授業で実験を行う計画を立てている．実験から収集したデータに LA 法を適用して分析し，学生が陥りがちなミスや解答行動を解析する予定である．これらの分析から得られた知見は，Java プログラミング教育の支援に活用されることが期待できる．

謝辞

本研究は，日本学術振興会科研費補助金（助成番号 20K03232，20K03265）の助成を受けている．

参考文献

[1] 掛下,柳田,太田,"穴埋め問題を活用したプログラミング教育支援ツール pgtracer の開発と評価",情報処理学会論文誌：教育とコンピュータ,Vol.2,No.2,pp.20-36,2016.

[2] T. Kakeshita, K. Ohta, “Student log analysis functions for web-based programming education support tool pgtracer”, IPSJ Trans. on Education and Computer, Vol. 5, No. 2, pp. 456-468, 2019.

[3] T. Kakeshita, M. Murata, “Application of Programming Education Support Tool pgtracer for Homework Assignment”, International Journal of Learning Technolo-gies and Learning Environments, Vol. 1, No. 1, pp. 40-61, 2018.

[4] M. Murata, T. Kakeshita, “Analysis method of student achievement level utilizing web-based programming education support tool pgtracer”, 5th International Con-ference on Learning Technologies and Learning Environment (LTLE 2016), Ku-mamoto,

Japan, pp. 316-321, July 2016.
[5] J. Gamma, E. Helm, R. Johnson, R. Vlissides, Design Patterns Elements of Reusable Object Oriented Software, Addison-Wesley Professional, 1994.
[6] Han Hsiao, P. Brusilovsky, S. Sosnovsky, “Web-based parameterized questions for object-oriented programming”, E-Learn'2008: World Conference on E-Learning, 2008.
[7] N. Truong, P. Roe, P. Bancroft, "Static analysis of students' Java programs", Sixth Australasian Computing Education Conference (ACE 2004), 2004.
[8] M. Hauswirth, A. Adamoli, "Teaching Java programming with the Informa clicker system", Science of Computer Programming, Vol. 78, Issue 5, pp. 499-520, 2013.
[9] N. Funabiki, Y. Matsushima, T. Nakanishi, et al., "A Java programming learning assistant system using test-driven development method," IAENG International Journal of Computer Science, vol. 40, no.1, pp. 38-46, 2013.
[10] K. K. Zaw, N. Funabiki, W.-C. Kao, "A proposal of value trace problem for algorithm code reading in Java programming learning assistant system," Information Engineering Express, Vol. 1, No. 3, pp. 9-18, 2015.
[11] H. H. S. Kyaw, N. Funabiki, W.-C. Kao, "A proposal of code amendment problem in Java programming learning assistant system," International Journal of Information and Education Technology, Vol. 10, No. 10, pp. 751-756, 2020.
[12] S. H. Edwards, N. Kandru, M. B. M. Rajagopal, “Investigating static analysis errors in student Java programs”, International Computing Education Research (ICER) conference, pp. 65-73, 2017.
[13] D. McCall, M. Kolling, “Meaningful categorization of novice programmer errors”, In Frontiers in Education Conference, pages 2589-2596, 2014.
[14] A. Altadmri, N. C. C. Brown, “37 million compilations: Investigating novice programming mistakes in large-scale student data” SIGCSE '15 Proceedings of the 46th ACM Technical Symposium on Computer Science Education, pp. 522-527, 2015.
[15] 結城浩, 増補改訂版 Java 言語で学ぶデザインパターン入門, Softbank Creative, 2004.