International Journal of Learning Technologies and Learning Environments
International Institute of Applied Informatics
2023, Vol. 6, No. 1, IJLTLE699

Fill-in-the-blank Questions for Object-Oriented Programming Education and Its Preliminary Evaluation

Miyuki Murata [*], Naoko Kato [†],
Mika Ohtsuki [‡], Tetsuro Kakeshita [‡]

Abstract

Object-oriented technology is important to improve software quality from various perspectives. We have developed pgtracer, a programming education tool that provides fill-in-the-blank questions for the C programming language. By analyzing the data collected by using pgtracer in actual classes, we have obtained useful knowledge for C programming education. In this paper, we develop fill-in-the-blank questions for Java programs to extend pgtracer for object-oriented programming. The fill-in-the-blank question consists of a set of programs and trace tables. A program and a trace table respectively correspond to a Java class and an instance. A trace table contains message sendings between instances, which are important for understanding the behavior of object-oriented programs. Furthermore, we introduce blanks that students do not need to fill. This provides more flexibility in setting the difficulty level while reducing the student workload to fill the blanks. We report the results of a trial experiment in which students were asked to solve some of the fill-in-the-blank questions using the Embedded Answers (Cloze) question type of Moodle's Questions function. Analysis of the collected student data will provide useful knowledge for object-oriented programming education, which will be reported in a future report.

Keywords: Learning Analytics (LA); programming education, object-oriented programming, Java, fill-in-the-blank question

1 Introduction

With the further development of information technology in recent years, IT-based services are being provided in many application domains. As these services become more sophisticated and complex, object-oriented programming is highly important to improve software quality and the development efficiency of these services.

---

[*] National Institute of Technology, Kumamoto College, Yatsushiro, Japan
[†] National Institute of Technology, Ariake College, Omuta, Japan
[‡] Saga University, Saga, Japan

Although object-oriented programming education is provided in universities and institutes of technology to develop software engineers, the lack of time and staff makes it hard to provide sufficient programming exercise. We are thus developing an education support tool pgtracer [1, 2] utilizing fill-in-the-blank questions to support programming exercises and learning analytics (LA) utilizing the collected data. Fill-in-the-blank questions have the advantage that the difficulty level of the question can be easily adjusted according to the location and size of the blanks and that an answer log can be collected for each individual blank. It is expected that the knowledge obtained from LA applying the collected log can be used to improve the effectiveness of the learning process.

The purpose of this paper is to support Java programming education by applying the fill-in-the-blank questions of Java programs to pgtracer to acquire knowledge about students' learning processes and achievements. In this paper, we propose an extension of fill-in-the-blank questions of Java programs. The proposed questions are provided to students in an actual class and are evaluated.

A fill-in-the-blank question for pgtracer is composed of a set of Java programs and trace tables. A trace table corresponds to an instance and represents the values of variables and outputs of the instance. An innovation of our research is that the blanks are defined in a trace table not only in the program. Through our previous experiment of C programming, we found that the students who have a high understanding of the trace table also have a high achievement of the program, and visualizing the change of the variable value by the trace table is useful for the acquisition of the program [3, 4]. It is also important to trace message sending among objects to understand a Java program. Thus, we extend the trace table to represent such message sending in this paper.

When we define a blank within a program or a trace table, other portions of the program or the trace table may serve as hints. To hide such hints, it is sometimes necessary to define many blanks. However, this may increase the number of similar blanks and may reduce students' motivation to learn [4]. By introducing a special type of blanks that students do not need to fill in, we can provide more flexibility in controlling the difficulty level of the questions while hiding hints.

We utilize 12 Java programs to demonstrate GoF design patterns [5]. These programs fully make use of the functions of Java programming language and object-oriented design so that they are suitable for the education of object-oriented programming. We classify these programs into three levels and adjust the difficulty levels of the fill-in-the-blank questions by carefully choosing the place of the blanks.

We are planning to utilize the proposed fill-in-the-blank questions in an actual class and to apply the LA method to the collected data. The class is "Programming Exercise III", which is provided in the third academic year of the Computing Division, Faculty of Science and Engineering, Saga University. Since pgtracer for Java programs is still under development, we conducted a trial experiment using Quiz, one of the Moodle activities. Using the analysis of the logs collected and the results of the questionnaire, we will examine the differences in the difficulty level of the different types of blanks and the student's understanding of the concept of the trace table, and verify the validity of the proposed questions.

This paper is organized as follows. In the next section, we describe some related research. We introduce pgtracer functions in Section 3. In Section 4, we describe how to express programs and define blanks within the program. We describe how to express trace tables and define blanks within the trace table in Section 5. In Section 6, we propose a strategy to adjust the difficulty levels of the fill-in-the-blank questions. Section 7 describes the lecture and the trial, and Section 8 describes the results of the trial. Section 9 discusses the results of the questionnaire. We shall conclude the paper in the last section.

2 Related Works

There are many educational tools for object-oriented programming with different educational objectives.

Hsiao et al. proposed web-based parameterized questions for Java [6]. The questions are as examining the final value of a variable or predicting the text being printed. Pgtracer allows setting blanks not only in the final value but also in the variable values at each step of the program. The ability to trace not only the final correctness but also the variable values at each step is useful for estimating the degree of understanding of the program.

Truong et al. introduced a static analysis framework for the Java program [7]. The student answers the entire program of an assigned question. The system uses model answers and predefined gaps for analysis. In pgtracer, the same kind of problem can be created by defining the entire program as blank. Furthermore, by analyzing the answer log of all students, pgtracer can discover trends in errors other than the expected ones.

Hauswirth et al. proposed the Informa clicker system to teach Java programming [8]. Informa provides several types of questions related to program code such as a multiple-choice question but does not address the tracing of variables.

Funabiki's research group proposed several programming education systems for Java utilizing fill-in-the-blank problems [9, 10, 11]. The problems in each of these systems define the blanks for the part of a program statement, an entire statement, or an output value. However, they cannot address message sending or variable values for each step of program execution. Our proposed question provides more flexibility because it can define the blanks for these parts as well.

Most of the studies on LA using Java programs have analyzed program and compilation errors. Edwards et al. analyze static analysis errors occurring in student-written Java programs, and detected using Checkstyle and PMD. They obtained knowledge such as the most common errors [12]. McCall et al. analyzed errors in student programs and studied the classification of errors and their frequency [13]. Altadmri et al. analyze the frequency, time-to-fix, and spread of errors among users using a year's worth of compilation events from over 250,000 students [14].

Pgtracer collects not only the final answers but also the answers that the students input during the answering process. By using these data, it is possible to analyze the behavior of the students until they reach the final answer.

3 Programming Education Support Tool Pgtracer Utilizing Fill-in-the-Blank Questions

Pgtracer provides the functions to create fill-in-the-blank questions, functions to provide a question to the students, functions to evaluate student answers, functions to collect student learning logs, and to analyze the log.

3.1 A Fill-in-the-Blank Question

A fill-in-the-blank question is composed of four XML files, representing a program, a trace table, masks for the program, or masks for the trace table. A token, consecutive sequence of tokens, an expression, and a statement are the candidate of blanks within a program. A variable value, a step number, and a variable name are the candidate of blanks within a trace table. We have verified that the difficulty level of a question can be controlled by the place of the blanks [1, 2].

3.2 The Functions provided by pgtracer

Pgtracer provides various functions. The first function is to edit a fill-in-the-blank question. Pgtracer provides the functions to generate XML files, for a program, a trace table, masks of a program, and a trace table.

The Second function provides fill-in-the-blank questions for a student and automatic evaluations of the student’s answer. Pgtracer fills the blanks within a program using student answers. Then pgtracer executes the filled program and compares the execution process of the program with the correct trace table.

Pgtracer also automatically collects student logs, such as the student's answer, the correct answer, and the required time, just after filling in each blank. Pgtracer provides the analysis functions for the collected log from various viewpoints such as the analysis functions of each student, each question, each blank, and the detailed learning process of a student [2]. The analysis functions are useful to analyze students’ achievement and their learning process.

3.3 Programming Education Model using pgtracer

Figure 1 represents the programming education process utilizing pgtracer. A teacher creates a fill-in-the-blank question using the correct program and the input file. A student chooses the question from the question database and answers the question. Pgtracer automatically evaluates the student's answer. At the same time, pgtracer collects the answer log. The students and the teacher can analyze the answer log using the analysis functions. The teacher then can improve the question database to provide a better set of fill-in-the-blank questions and may give feedback to the student or the entire class.

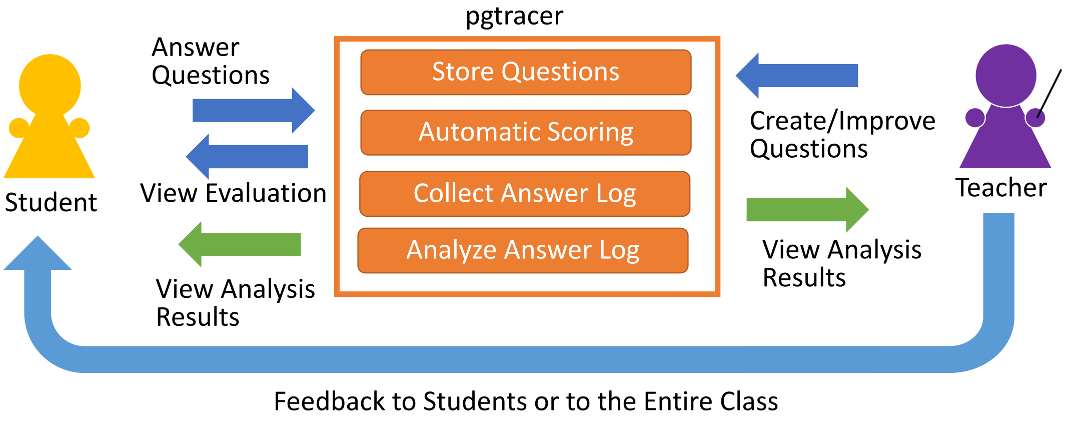
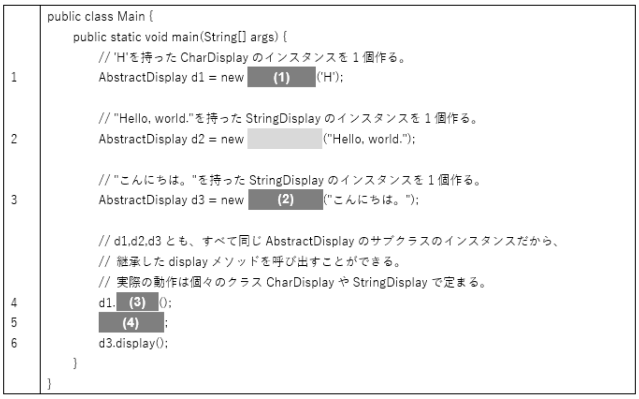

pgtracer
Store Questions
Automatic Scoring
Collect Answer Log
Analyze Answer Log
Student
Answer Questions
View Evaluation
View Analysis Results
Teacher
Create/Improve Questions
View Analysis Results
Feedback to Students or to the Entire Class

Figure 1: Programing Education Process utilizing pgtracer.

4 A Program for Fill-in-the-Blank Question

In this section, we describe how to express programs and to define blanks within a program. Figure 2 represents an example.



```
    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();
        }
    }
```

Figure 2: Example of a Program with Blanks (Template Method, Main Class)

4.1 Representation of a Program

Since the most high-level component of a Java program is a class, there is one source file for each class. Thus, a fill-in-the-blank question of a Java program generally contains multiple source files. We also assign a step number to each statement including instance variable and local variable definitions since they may contain initialization statements. The step numbers are assigned according to the following rules and correspond to the step numbers in the trace table described in Section 5.

- Assign a sequential step number to each statement including instance variable definition.
- Assign a sequential step number starting from 1 to the statement in the method definition.

- Assign a sequential number to each statement defined in Java, such as assignment statement, method call, or control statement such as if, else, switch, case, default, while, return, etc.
- Assign step numbers such as “x.y” to compound statements having nested structures.

A program must be compiled successfully. We also define the following guidelines for the Java program.

- Indentation is used to correctly represent nested structure.
- Class definitions are preceded by an independent comment explaining the concept or feature of the class.
- Method definitions are preceded by an independent comment explaining the feature provided by the method.
- Variable definitions are accompanied by a comment of the same line explaining the stored values in the variable.
- A sequence of statements is preceded by an independent comment explaining their intention or algorithm.
- An independent comment is preceded by a blank line.
- Only one statement is described in a single line if possible.
- Variable and function names follow the Camelcase convention. However, the first letter of a class name should be capitalized.

4.2 Blanks within a Program

A token is a string that cannot be further subdivided, such as a variable name, a class name, an operator, a keyword. A comment is defined as a token. Pgtracer allows defining a blank at an arbitrary sequence of tokens. Thus, a part of a statement, an entire statement, and a sequence of consecutive statements can be defined as a blank.

There are two types of blanks. One requires an answer, the other does not. We called a blank of the latter type an ignorable blank. By introducing the ignorable blanks, the difficulty level of a blank can be controlled more flexibly. Consider the case to repeatedly send similar messages to different objects. If a blank is defined within one of these statements, the statements near the blank may become a hint to fill the blank. By defining appropriate ignorable blanks within other statements, we can hide the clue tokens without increasing the number of blanks that the student must fill. Thereby the difficulty level of the question can be controlled more flexibly.

Since it is also possible to hide comments using the ignorable blank, we can also control the difficulty level of a question. To distinguish between the blanks and the ignorable blanks, the background of these blanks is filled with different colors. The blanks without a number represent ignorable blanks in Figure 2.

5 A Trace Table for Fill-in-the-Blank Question

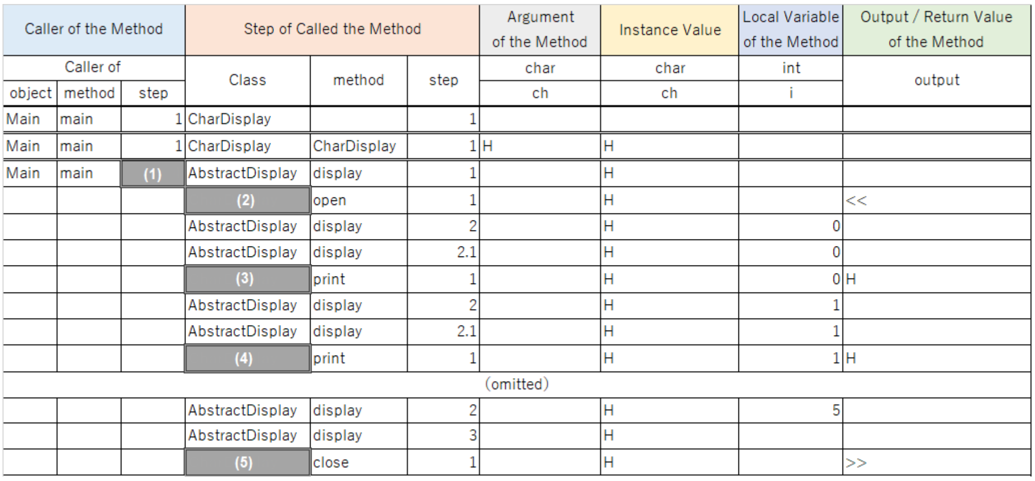
5.1 Representation of a Trace Table

A Java program is executed through message passing between objects. If the trace table is expressed in the order of execution steps, the table will be complicated. Thus, we define a trace table for each object and represent each message passing in the trace table. This approach also matches with the object-oriented programming concept such as sequence diagram. The proposed trace table is defined below. Figure 3 provides an example.

5.1.1 *Representation of Instance Identifier*

We shall define a trace table for each instance and the Main method. A trace table corresponds to an instance that has the name of the instance, while a trace table representing the Main method has the name "Main".

Each instance is expressed as "ClassName#X" to distinguish each instance of a class. Here, "ClassName" is the string of the class name with the first letter is changed to the lowercase letter, and "X" is a sequence number representing the order of generation. For instance, the identifier of the first generated instance of class "CharDisplay" is "charDisplay#1"



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		>>

Figure 3: Example of a Trace Table with Blanks
(Template Method, Instance ID = charDisplay#1)

Table 1: Columns of a Trace Table for each Object

Column	Sub-Item	Value
Caller of the Method	Object	Instance or Class Name
	Method	Method Name
	Step	# of Step
Step of Called the Method	Class	Class Name which the method is defined
	Method	Method Name
	Step	Step Number of the Code
Argument of the Method	• Upper Column: Data Type, Class Name • Lower Column: Argument	Argument value when the corresponding code is executed

Column	Sub-Item	Value
	Name	
Instance Variable	• Upper Column: Data Type, Class Name • Lower Column: Variable Name, Instance Name	Value or Object Identifier
Local Variable of the Method		
External Object	• Upper Column: Class Name • Lower Column: Instance Name	Object Identifier
Output / Return Value of the Method	Return Value of the Method	Return Value or Returned Object Identifier
	Output	Output String

5.1.2 Columns of a Trace Table

The trace table for an object has columns listed in Table 1, while the trace table for the Main method has the same columns except “Caller of the Method”. The “Caller of the Method” column describes the calling object. The object is generated by a call to the constructor. After generating the object, other objects may send messages to the object to execute some operations.

In a trace table, all the objects are listed as sub-items of the “External objects” column, if the current object creates the object or sends a message to the object. The value of the sub-item is the name of a called method. We can then understand the lifetime and the message flow of each object generated within the program.

The trace table of a class including a static method is named to the class name. The trace table has the item “Class Variable” and the sub-item “Data Type” in addition to the items shown in Table 1.

5.1.3 Multiple Method Calls

An object may be called multiple times. To distinguish the series of processes in each method call, a double line is drawn in the trace table. Specifically, a double line is drawn under the row corresponding to the return statement.

5.1.4 Statement Representation Containing Multiple Operations

A single line of code may contain multiple operations. For example, the following code contains two operations.

```
AbstractDisplay d1 = new CharDisplay(‘H’);
```

One is the generation of an instance of the class "CharDisplay" and the other is the assignment of the generated instance to the variable d1.

Understanding each operation is necessary to understand the program. Therefore, in the trace table, these two operations are represented by two rows in the order of execution. Here, the step numbers are all the same. The operation represented by a row in the trace table can be recognized by referring to the column of instance variables and external objects.

5.1.5 Representation of Variable Values

If the data type of the variable is a basic data type such as an integer, the stored value is displayed. If an instance is stored, the cell displays the corresponding instance identifier.

Nothing is displayed if the variable is not allocated a space within the program or the variable is in an indefinite state. Although these two cases are different in a strict sense, the target course does not require students to strictly distinguish them. At an elementary level of object-oriented programming education, we consider that presenting these subtle differences may cause confusion among students and hinder their understanding process. Considering this, we use the same notation for both of two cases.

5.1.6 Abbreviation of Iterative Steps

If the number of iterations is large, the trace table becomes quite large and may become hard to understand. Therefore, we decided to describe the iterative process by omitting the intermediate steps. To clarify the omitted rows, a row describing (omitted) is inserted.

5.2 Blanks within a Trace Table

In a trace table, the value of each cell can be defined as a blank. Specifically, variable values, output values, step numbers, object identifiers, class names, method names, etc. Moreover, method arguments, instance variables, local variables, and external objects, a blank can be defined for their sub-items, data types or class names, variable names, or object names can be defined as a blank.

Like the case of the program, it is possible to define ignorable blanks that do not require an answer in a trace table. In a trace table, the state of a variable does not change until a new value is assigned to it. Therefore, even when we want the students to guess the value from the program, they can guess it from the values at the previous and next rows. To prevent this, the values before and after the target value were also defined as blanks. However, this was not appropriate for verifying the difficulty of the question, because some students confused the difficulty of the question with the tediousness of answering since the additional blanks increased the number of inputs. This issue can be solved by introducing ignorable blanks that do not require an answer.

6 Creation of Fill-in-the-Blank Questions

In this section, we propose the development strategy of fill-in-the-blank questions. We plan to provide the created problems in “Exercise in Programming III”, which is provided in the third academic year of the computing courses at Saga University. The students learn Python in the first year, and in the second year, they learn structured programming using C++, so that students are familiar with fundamental programming techniques as well as basic sorting and alignment algorithms. They start learning object-oriented programming using Java at “Programming III”.

6.1 Blanks within a Trace Table

The target class intends to learn the purpose and usage of various tools used in the practical field of software development to maintain efficiency and high quality. Specifically, students develop software in the Java language using the integrated development environment (IDE) Eclipse and learn how to use tools such as JUnit and Jenkins.

In the first week, the students get a lecture about the IDE and practice basic Java programming. Then the students learn Git and registration to GitHub, software testing techniques and unit test design, JUnit, Jenkins, UML diagrams, and design patterns. Finally, development exercises are given for four weeks. In parallel with this lesson plan, the students are provided with the fill-in-the-blank questions proposed in this paper, to help them master Java programming.

6.2 Providing Fill-in-the-Blank Questions

The purpose of the exercises is to deepen the understanding of the students by providing them with exercises that correspond to the course content. Twelve topics are selected from "Introduction to Design Patterns in Java" and their sample programs are used to create the problems [15]. The topics are selected considering the contents to be covered in the class, and the question levels were set as beginner, intermediate, and advanced according to the topic contents and the order of teaching (Table 2).

Table 2: The Topic of the Providing Fill-in-the-Blank Questions

Order	Level	Design Pattern	The section in the Textbook
1	Beginner	TemplateMethod	3
2		FactoryMethod	4
3		Iterator	1
4		Composite	11
5	Intermediate	Decorator	12
6		Strategy	10
7		AbstractFactory	8
8		Observer	17
9	Advanced	Adapter	2
10		Builder	7
11		Command	22
12		Visitor	13

6.3 Development Policy of the Fill-in-the-Blank Questions

Through our previous research, we have obtained the following knowledge about fill-in-the-blank questions for C language [3, 4].

- Questions with blanks defined only within the trace table are the easiest, followed by questions with blanks defined only within the program. The questions with blanks defined within both the trace table and the program are most difficult for the students.
- The difficulty level of blanks within a program increases in the order of individual tokens, sequences of tokens, and entire sentences.
- The difficulty level of blanks within the trace table increases in the order of step number, variable value, and variable name.
- Comments reduce the difficulty level of the problem.

- As the number of blanks increases defined in a question, the motivation of the students decreases to answer the question.

Since the students have already learned the basics of programming through the previous classes, we designed the questions for the students to learn the basics of object-oriented programming and design patterns.

The blanks within the program are defined mainly to check the understanding of Java-specific grammar such as class definitions and to check the execution flow of message sending. Furthermore, the blanks within the trace table are mainly defined for topics about the caller objects, such as classes, method names, and step numbers, items about the called object, and values that change as messages are sent and received.

Considering these factors, we developed the following policies to create questions with appropriate levels of the topic.

(1) Define approximately ten blanks per question. Here, ignorable blanks are not counted.

(2) Create two problems for one topic with different difficulty levels.

(3) Clarify the educational objective of each question.

(4) Clarify the intent for each blank.

(5) To define blanks within a program, we use individual tokens as the basis at the beginner level. We use a longer sequence of tokens for intermediate and advanced levels.

(6) To define blanks within a trace table, we mainly use variable values at the beginner level. At the intermediate level, we use more blanks of instance identifiers and methods.

According to our previous experience, we found that if the number of blanks in the question was too many, students' motivation will decrease, and this would prevent them from continuous use of pgtracer. In our previous experiment, five blanks were defined per question because the target students were beginners of the C language and the total number of lines of the program was approximately 20. In this paper, however, approximately ten blanks were defined for each question because the level of programming proficiency of the students is expected to be higher than that of the previous students. The total number of lines in the program is larger for the same reason.

The second policy is defined to investigate the difference in difficulty between types of questions. Furthermore, the educational objective of the questions and the intention of each blank, clarified by policies 3 and 4, will be used to analyze the learning logs. Policies 5 and 6 are set to clarify the differences among the beginner, intermediate, and advanced levels.

6.4 An Example of a Fill-in-the-Blank Question

Table 3 shows the educational objective of the question created based on the program described in the "Template Method" section and the intent of some blanks of the question. The resulting question can be found in Figure 2.

Consider the blanks (1) and (2), since the variables d1 and d2 are variables of the abstract class "AbstractDisplay", the user is required to understand that the constructors of either "CharDisplay" or "StringDisplay", which are subclasses of "AbstractDisplay", are to be called, and to derive the appropriate constructor from the arguments. It should be relatively easy to derive the appropriate constructor based on the comments. The statement of Step 2 is not shown because it is the same process as the blank (2).

The blank (3) and (4) are in the statement of the method call. For the blank (4), it is necessary to derive the instance from the output value. Furthermore, although it is a sequence of tokens, it can be derived from the comments and the statement in step 6.

Thus, the difficulty of the blanks can be controlled utilizing the length of the blank such as tokens, sequence of tokens, and entire statement, and the show/hide of comments. This question becomes more difficult if the comments are hidden while leaving the blank unchanged.

Table 3: Educational Objective and Intent of Some Blanks within the Program (Template Method)

Educational Objective	Recognize usages of abstract classes and abstract methods.

Blank#	Right answer	The intent of the Blank
(1)	CharDisplay	Understand a constructor call.
(2)	StringDisplay	Understand a constructor call.
(3)	display	Can describe method calls.
(4)	d2.display()	Can derive an instance and a method from the output.

7 Primary Trials in a Lecture

The name of the lecture in which the trial was conducted is "Programming Exercise III". The target students of this lecture are third-year students at Saga University. The number of students who took the course was 76. This trial was conducted in the first semester of the 2021 academic year.

The Java language is taught in this lecture in the third year. However, detailed explanations of the language are not given in the lecture, and self-study at external online learning sites is recommended as supplementary material after the basics are explained.

7.1 Trial using Moodle

This trial used the Embedded Answers (Cloze) question type of the Questions feature, rather than the original Moodle module. This led to several limitations.

First of all, unlike the C version of pgtracer, there is no automatic question generation function, so we had to manually create and present the fill-in-the-blank questions as images. Secondly, the C version of pgtracer had more freedom in accepting descriptions as long as they could be compiled and executed, but the Embedded Answers (Cloze) question type had less freedom in the answers that could be set because only short sentences could be filled in and regular expressions could not be used.

Although it is possible to limit the number of times a student could take the test, the number of attempts was set to two in this trial. However, when students were allowed to take the test more than once, some students would memorize the answers and copy them at the succeeding test. Therefore, when students took the test twice, the average score was used for evaluation so that it would be difficult for them to cut corners on the first attempt.

7.2 Problems used in the trial

In each week, we conducted trials with the following problems as indicated in Table 4. Suffixes such as PR and TR in the problem ID indicate that the problem type is a program or a trace table, respectively. The column on the right is the number of examinees for the first attempt.

Since some of the students were not familiar with the Java language, we presented a very basic program and trace table problem in the first week. After that, we presented the design patterns with the simplest structure. Presenting the program and the trace table for the same problem at the same time was problematic because the trace table presented the entire program. In addition, the student workload was too heavy when there were other exercises to be done. For these reasons, we divided the trace table and the program into separate weeks and adjusted the way the program was presented so that it was presented before the trace table.

For the trace table, a document explaining how to answer the questions in a simple program was prepared when the trace table of the Iterator pattern was presented. For the Iterator pattern, the programs were explained during the lecture, so the questions on the programs were omitted. The fill-in-the-blank questions for the Factory Method pattern program were posted again after the lecture because problems were found in the program. Since there was an error in the description of the problem for the Composite trace table, the program was posted first after that.

Table 4: Problems and Number of Examinees

Week	Design Pattern	level	Problem Type	Problem ID	# of Blanks	# of Examinees
1	“for” statement		Program	Ex01_PR	6	67
			Trace table	Ex01_TR	4	
6	Template Method	Beginner	Program	Ex06_PR	12	71
		Beginner	Trace table	Ex06_TR	12	
7	Factory Method	Beginner	Program	Ex07_PR	9	58
8	Simple Example		Trace table	Ex08_SP	2	67
	Iterator pattern	Beginner	Trace table	Ex08_TR	10	
10	Factory Method	Beginner	Trace table	Ex10_TR	9	61
11	Composite	Beginner	Program	Ex11_PR	11	69
13	Strategy	Interme-diate	Program	Ex13_PR	10	66
14	Observer	Interme-diate	Program	Ex14_PR	10	65

8 Answer Result

Most of the students repeated the attempt. Since the correct answers have already been disclosed for the second attempt, only the results of the first attempt will be included in the following analysis. Ex01_PR, Ex01_TR, and Ex08_SP are the problems to explain how to answer or concept of a trace table. Therefore, we except for these problems for discussion.

Table 5 shows the number of blanks and the average right answer ratio for each of the fill-in-the-blank questions which problem type is “program”, categorizing the blanks into two types which are composed of a simple token and multiple tokens. In both the beginner and intermediate levels, the average right answer ratio for the blanks containing multiple tokens was lower than for the blanks containing a token. This indicates that the blank containing multiple tokens is more difficult for students to answer. For both blanks containing a token or multiple tokens, the average right answer ratio is lower at the intermediate level than at the beginner level. At the intermediate level, we set blanks that require an understanding of the processing flow of the program to derive the correct answer. Therefore, we were able to reflect the difficulty level intended by the contestants, even if it was a blank containing a token.

Table 5: Number of blanks and right answer ratio (%) for problems which type is “program”

Level of problem	# of problem	All of the Blanks		Blanks containing a Token		Blanks containing Multiple Tokens	
		# of Blanks	Average of Right answer ratio (%)	# of Blanks	Average of Right answer ratio (%)	# of Blanks	Average of Right answer ratio (%)
Beginner	3	32	70.6	22	76.5	10	57.7
Intermediate	2	20	55.8	11	67.2	9	41.9

Table 6 shows the number of blanks and the average right answer ratio for the fill-in-the-blank questions which problem type is “Trace table”, categorized by the type of blanks. Table 6 indicates that the average right answer ratio for the step number of the method caller and the variable value that answers the instance is low. This suggests that problems related to object-oriented specific method invocation and instances are difficult for students. In addition, as will be discussed in Section 9, it is thought that some students did not reach the correct answer due to the difficulty of tracing caused by the existence of multiple trace tables and programs.

Table 6: Number of blanks and right answer ratio (%) for problems which type is “Trace table”

Type of Blanks	# of Blanks	Average of Right answer ratio (%)
Call for constructor or method	16	72.5
Step number of the calling method	3	46.0
Called Class	2	61.7
Return value (Instance)	5	64.0
Variable value (Instance)	1	38.8
Variable value (Basic type)	4	71.1

9 Questionnaire Result

On the last day of the lecture, we took a questionnaire about this trial. We used the Moodle survey module so that we can identify the respondents. The students were notified that they could write their frank opinions without any disadvantage depending on their answers. The number of responses was 69.

The questionnaire items and their respective IDs are listed in Table 7. The eight questions from ENQ01 to ENQ08 have five levels of answer options as shown in Table 8. Only ENQ09 is a free-text survey.

Table 7: Questionnaire Items

Q ID	Question
ENQ01	Did you understand the behavior of the correct program itself?
ENQ02	Did you understand the concept of a trace table?
ENQ03	Did you trace instance variables and local variables?
ENQ04	Did you trace the flow of sending and receiving messages?
ENQ05	How difficult are the fill-in-the-blank questions?
ENQ06	How about the number of blanks per question?
ENQ07	Do you have an interest in programming and a desire to learn it?
ENQ08	Do you think that fill-in-the-blank questions are useful for learning programming?
ENQ09	If you have any suggestions for improvement or comments on the fill-in-the-blank questions, please feel free to write them.

Table 8: Description of the Answer Options for Questionnaire Items

Q ID	Answer Options				
	1	2	3	4	5
ENQ01	Barely understood	Didn't understand much	Understood	Understood generally	Understood very well
ENQ02					
ENQ03	Not traced	Not much traced	Traced	Almost traced	Traced correctly
ENQ04					
ENQ05	Very difficult	Rather difficult	Neither	Rather easy	Very easy
ENQ06	Very many	Rather many	Neither	Rather few	Very few
ENQ07	Unmotivated	Somewhat unmotivated	Neither	A little motivated	Highly motivated
ENQ08	Disagree	Disagree a little	Neither agree nor disagree	Agree with a little	Strongly agree

Figure 4 shows the results of the five-step evaluation from ENQ01 to ENQ08. The labels of a graph element represent percentages. For ENQ01, understanding the contents of the program, and ENQ02, understanding the concept of the trace table, 69.5% and 68.1%, respectively, answered that they understood the program. On the other hand, 39.1% and 42.0% respectively answered that they could not trace the variables and message passing in the trace table (ENQ03 and ENQ04).

This indicates that although the students were able to understand the program itself and the concept of the trace table, they had difficulty in tracing variables and messages sent and received. This difficulty may be caused by the format of the problem display, so we should consider how to improve the display format.

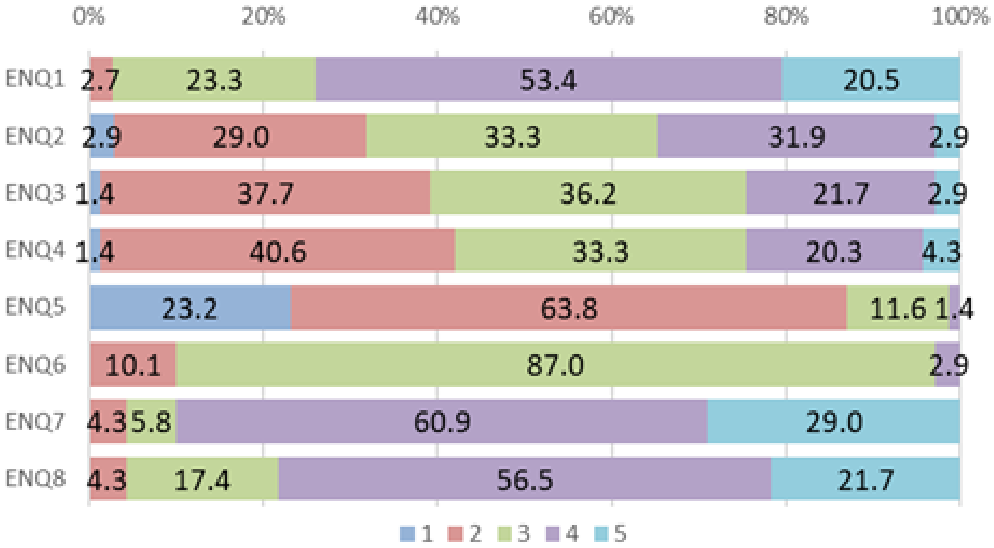


Figure 4: Questionnaire Results from ENQ01 to ENQ08

As for the fill-in-the-blank questions themselves, 63.8% answered that they were rather difficult (ENQ05). The amount of fill-in-the-blank questions itself was appropriate for 87.0% of the respondents (ENQ06).

Consider ENQ05, 63.8% of the students answered that the problems were rather difficult. This suggests that the level of difficulty of the problems is appropriate. According to ENQ06, 87.0% of the students answered that the number of blanks was just right, which is considered appropriate. In this paper, we introduced a blank that does not require an answer. We believe that this allowed us to set an appropriate number of blanks to be answered while hiding codes and variable values that could be used as hints. Therefore, we can reduce the student workload of filling in blanks and prevent them from losing motivation.

Moreover, 89.9% answered that they were interested in the program and willing to learn (ENQ07), and 78.2% answered that the fill-in-the-blank questions were useful for learning (ENQ08). These results indicate that the students took this trial experiment seriously, and the responses from ENQ01 to ENQ06 can be considered reliable.

Finally, consider the 14 opinions of students obtained from the free text item ENQ09. There were five responses regarding the method of displaying programs and trace tables. In this trial experiment, the program and the tracing table were presented as images in multiple tabs, but it was necessary to switch tabs when understanding the program and tracing messages sent and received, and this may have hindered understanding. In addition, there were three responses related to answer writing. They were complained about incorrect answers due to differences in capitalization and lowercase letters when typing. However, in actual programming, the difference between lowercase and uppercase letters can be a fatal bug, therefore we should consider educationally and require correct input.

10 Observation

When creating the blanks within the programs or the trace tables, we clarified the intent of each blank. We found that the student should obtain knowledge about Java syntax, design patterns, processing flow, and combinations of these to get the correct answer. By further categorizing this knowledge and analyzing it together with the students' answer logs, it may be possible to estimate the difficulty level of each blank.

Creating problems is complex because it requires cross-checking of multiple program files and trace tables. Therefore, we expect to have a navigation function from the statement of a method call to the program defining that method or from the instance identifier to the trace table of that instance.

There may be hints before and after the blanks, such as the same value in the trace table or a similar description of the program. We can hide these areas by introducing ignorable blanks that do not require an answer. This allows us to control the number of blanks requiring an answer to satisfy the educational objective of the question.

In this paper, consecutive tokens were defined as one blank, but each token can also be defined as one blank. This would reduce the difficulty of the question since students would have the number of tokens as a hint. For students who cannot solve blanks containing multiple tokens, we can provide the number of tokens as a hint.

11 Conclusion and Future Work

In this paper, we proposed a fill-in-the-blank problem for the Java language. The difficulty level of the question is set by whether the blanks are in the program or the trace table, and by the length of the blanks. Also, the difficulty of the question can be controlled more flexibly by introducing ignorable blanks that do not require an answer.

In this trial, a part of the fill-in-the-blank questions in Java programming, which was designed in a previous study, was solved by students in a lecture where they practiced programming. Because we used the Moodle quiz module rather than extending pgtracer itself, there were some problems in the way the problems were presented and judged. However, these problems should be improved by the implementation of the Java version in the future.

Our research group has been developing a programming education support tool, pgtracer, which provides fill-in-the-blank questions. We have a plan to apply the question proposed in this research to pgtracer and perform an experiment in an actual class. We will analyze the data collected from the experiment by applying the LA method to analyze the mistakes and answering behaviors that students tend to make. The findings obtained from these analyses are expected to be used to support Java programming education.

Acknowledgment

This research is supported by JSPS KAKENHI under grant numbers 20K03232 and 20K03265.

References

[1] T. Kakeshita, R. Yanagita, K. Ohta, “Development and evaluation of programming education support tool pgtracer utilizing fill-in-the-blank question”, Journal of Information Processing: Computer and Education, Vol. 2, No. 2, pp. 20-36, Oct. 2016. (in Japanese)

[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 Technologies 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 Conference on Learning Technologies and Learning Environment (LTLE 2016), Kumamoto, 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] I-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] H. Yuki, An Introduction to Design Patterns using Java Programming Language, revised edition, Softbank Creative, 2004. (in Japanese)