A Proposal of Grammar-Concept Understanding Problem in Java Programming Learning Assistant System

Soe Thandar Aung, Nobuo Funabiki, Yan Watequlis Syaifudin, and Htoo Htoo Sandi Kyaw
Department of Electrical and Communication Engineering, Okayama University, Okayama, Japan
Email: funabiki@okayama-u.ac.jp

Shune Lae Aung and Nem Khan Dim
Department of Computer Studies, University of Yangon, Yangon, Myanmar
Email: shunelaeaung@gmail.com

Wen-Chung Kao
Department of Electrical Engineering, National Taiwan Normal University, Taipei, Taiwan
Email: jungkao@ntnu.edu.tw

***Abstract*—Nowadays, Java has been extensively adopted in practical IT systems as a reliable and portable object-oriented programming language. To encourage self-studies of Java programming, we have developed a Web-based Java Programming Learning Assistant system (JPLAS). JPLAS provides several types of exercises to cover different levels. However, any type does not question grammar concepts of a source code directly, although it can be the first step for novice students. In this paper, we propose a Grammar-Concept Understanding Problem (GUP) as a new type in JPLAS. A GUP instance consists of a source code and a set of questions on grammar concepts or behaviors of the code. Each answer can be a number, a word, or a short sentence, whose correctness is marked through string matching with the correct one. We present the algorithm to automatically generate a GUP instance from a given source code by: 1) extracting the registered keywords in the code, 2) selecting the registered question corresponding to each keyword, and 3) detecting the data required in the correct answer from the code. As for evaluations, we first generate 20 GUP instances with a total of 99 questions from simple codes on fundamental Java grammar, and assign them to 100 university students in Indonesia. On the other hand, we additionally generate 8 instances with a total of 30 questions, and assign all the instances to 29 undergraduates in Myanmar as the comparative study. The results show that the proposal is effective to improve the performance of the students who are novices in Java programming.**

***Index Terms*—Java, JPLAS, grammar-concept understanding problem, automatic generation algorithm**

I. Introduction

For decades, Java has been frequently used in a variety of applications such as client-server Web applications, Android applications, IoT (Internet of Thing) systems, and cloud service systems, as a highly portable object-oriented programming language. Currently, Java is still one of the most popular programming languages [1]. Thus, Java programming has been educated in numerous universities and professional schools. To enhance the education, we have studied a Web based online *Java Programming Learning Assistant System (JPLAS)* [2], [3].

JPLAS offers various types of programming exercises to cover different levels. In all types, JPLAS will automatically mark an answer from the student at the server. Two methods are adopted for this automatic marking. One is the *string matching* between the answer and the correct one stored in the database. If every character is identical, the student answer will be considered correct. Otherwise, it is not. Another is the *software testing*. By running the *test code* on *JUnit*, the correctness of the answer will be verified.

Currently, in JPLAS, we have defined and implemented six different problem types, called the *Value Trace Problem (VTP)* [4], the *Element Fill-in-Blank Problem (EFP)* [5], the *Code Completion Problem (CCP)* [6], the *Code Correction Problem (CRP)* [7], the *Statement Fill-in-Blank Problem (SFP)* [8], and the *Code Writing Problem (CWP)* [9]. For VTP, EFP, CCP, the *string matching* is adopted in marking, where a set of elements in a source code, such as numbers or words, will be requested in the answer. For CRP, SFP, and CWP, the *software testing* is adopted, where a full or part of a source code is requested in the answer.

To learn programming effectively, it is suggested that students solve simple problems for code reading and grammar concept studies first, and then practice the coding problems using object-oriented programming concepts. Therefore, students are expected to solve the programming exercises along this order of problem types in JPLAS.

Manuscript received November 30, 2020; revised August 25, 2021.

doi: 10.12720/jait.12.4.342-350

Nevertheless, all problem types in JPLAS do not explicitly require the student's knowledge and understanding of basic grammar concepts and keywords of Java programming that are used in the given source codes. In the programming study, novice students first need to realize the basic grammar concepts and keywords to read source codes correctly. Thus, teachers teach the fundamentals in the lectures using textbooks before assigning programming exercises to students. In all programming languages, each concept has been initiated for a specific purpose. Hence, the correct understanding of the concepts is essential to read or write source codes smoothly.

In this paper, we propose a *Grammar-Concept Understanding Problem (GUP)* as a new problem type in JPLAS and a first step problem for the novice students. The main research question of this paper is how to find the student who lacking the basic knowledge in programming. By solving the GUP instances, the teacher can know how much the students need necessary knowledge and how much they understand on programming concepts. Besides, we can encourage the students to study by themselves if they cannot solve. On the other hand, the students have to understand the keyword concerned with the Java programming as the first step. If the student doesn't have proper knowledge, it is impossible to continue study in programming.

A GUP instance consists of a Java source code, a set of questions, and the correct answers. Each question describes a basic grammar concept in Java programming that appears in the source code, and requests to point out the corresponding element or keyword in the source code. The answer is marked by the *string matching* like VTP and EFP.

To help teachers design GUP instances, we also propose the algorithm to automatically generate a GUP instance from a source code. To use this algorithm, the teacher needs to select a Java source code that will be studied by students for code reading. The algorithm first extracts the keywords or elements that are related to basic grammar concepts from the code. Then, it singles out the questions corresponding to the keywords, where the keywords in the code become the correct answers to the questions.

The *keyword list* and the *question list* are prepared for the algorithm. By updating them, the algorithm can deal with the extensions of the Java grammar. Besides, by changing them to a different programming language, the algorithm can be used there.

This algorithm involves several limitations. When the same keyword appears in the source code again, the algorithm will generate the same question for the keyword. A large number of Java source codes may have common keywords such as *class*, *access modifier*, *static*, *void*, and *main*. For those common keywords, the corresponding same questions are duplicated even for one code. To avoid it, the teacher needs to remove the duplicate or redundant questions before presenting the GUP instance to students.

In the evaluations, we first generate 20 GUP instances with a total of 99 questions from simple codes on fundamental Java grammar in a textbook [10], and assign them to 100 undergraduates in Indonesia. The results show that 87 students have acquired the necessary knowledge of the fundamental Java grammar to continue studying Java programming while the remaining students do not achieve the required level and need instructions of the teacher. As for the comparative study, we additionally generate 8 instances with a total of 30 questions, and assign all the instances to 29 undergraduates in Myanmar. It is proved that all the students have obtained the necessary knowledge to continue studying Java programming. Thus, the proposal is effective in identifying the students who are lack of the fundamental knowledge of Java programming and need more instructions.

The rest of this paper is organized as follows: Section II reviews our JPLAS preliminary works to this paper. Section III presents the details of the proposal. Section IV demonstrates the GUP instance generation algorithm. Section V shows evaluations of the proposal. Section VI introduces related works in literature. Finally, Section VII concludes this paper with future works.

II. Review of JPLAS

In this section, we review our preliminary works on JPLAS.

A. JPLAS Software Platform

JPLAS is a Web application system which allows a teacher to offer assignments of programming exercises to plenty of students in the class at the same time, and to manage their learning activities on the server. For the server platform in Fig. 1, *Linux* is adopted for the operating system, *Tomcat* is for the Web application server, and *MySQL* is in the database. The applications in JPLAS are implemented based on the *MVC model*, where *Java* is used for the *model (M)* part, *HTML/CSS/ JavaScript* are for the *view (V)* part of the browser, and *JSP* is for the *control (C)* part [3].

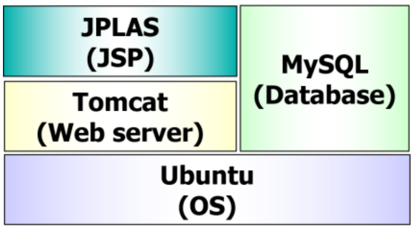

Figure 1. JPLAS server platform.

B. Implemented Problem Types

JPLAS provides the following types of programming exercise problems to cover different learning stages of Java programming:

- Value Trace Problem (VTP): VTP requests to answer the actual values of important variables in the given Java source code. The code often implements a fundamental data structure or algorithm.

- Element Fill-in-Blank Problem (EFP): EFP requests to fill in the blank or missing elements with the proper words in the given source code. The locations of the blank elements are explicitly shown in the source code.
- Code Completion Problem (CCP): CCP requests to fill in the blank or missing elements with the proper words in the given source code, like EFP. However, their locations are not shown in the code. Students need to discover the locations, and complete the whole statements.
- Code Amendment Problem (CAP): CAP requests to amend the incorrect elements in the source code. The incorrect elements are either missing or wrong. Students need to find out the locations of the incorrect elements, and complete the whole statements.
- Code Correction Problem (CRP): CRP requests to correct the incorrect source code so that it can pass the given test code on *JUnit*. The source code has several errors that cannot be passed by the test code.
- Statement Element Fill-in-Blank Problem (SFP): SFP requests to fill in the blank statements in the given source code so that it can pass the given test code on *JUnit*.
- Code Writing Problem (CWP): CWP requests to write a source code that passes the given test code on *JUnit*. To help a student, the detailed information for the source code implementation is usually described in the test code.

In JPLAS, the answer to each problem will be marked automatically on the server using the program. For VTP, EFP, CCP, and CAP, the answer is marked by the *string matching* with the correct one that is stored in the database. For CRP, SFP, and CWP, the answer is marked by the *software testing* using the test code on *JUnit*.

C. *Limitation*

These exercise problems assume that the students have already acquired the basic grammar concepts and keywords of Java programming in the lectures with textbooks. To avoid a huge dropout from the course due to insufficient knowledge, teachers should confirm the understanding levels of students in basic grammar concepts and keywords of Java programming, and help the students out who may not catch up with them.

Therefore, JPLAS should provide a new type of programming exercises that directly ask the grammar concepts or keywords that appear in a source code. In the next section, we will present the *Grammar-Concept Understanding Problem (GUP)* for further studies.

III. Proposal of Grammar-Concept Understanding Problem

In this section, we present the definition of the *Grammar Concept Understanding Problem (GUP)* and the algorithm to automatically generate a GUP instance.

A. *Definition of Grammar-Concept Understanding Problem*

A GUP instance consists of a Java source code, a set of questions, and the correct answers to the questions. Each question describes a basic grammar concept in Java programing that appears in the source code, and requests to pick up the corresponding element or keyword in the source code. The student answer is marked by the *string matching* with the corresponding correct answer.

B. *Example of GUP Instance*

Here, we show an example of the GUP instance. **source code1** shows the source code.

source code1

```
1 import java.util.Scanner;
2 public class UserIntegerInput {
3   public static void main(String[] args) {
4     Scanner scanner = new Scanner(System.in);
5     int num = scanner.nextInt();
6   }
7 }
```

Then, the set of questions and the corresponding correct answers are given as follows:

1) Which keyword is used to refer to the classes and interfaces in other packages? (import)
2) Which library needs to run the Scanner class? (java.util.Scanner)
3) Which keyword allows from other class in Line2? (public)
4) What is class name? (UserIntegerInput)
5) Which keyword allows the method to run without creating an object? (static)
6) Which keyword describes no returning data in Line 3? (void)
7) Which keyword represents the entry point from which the JVM can run this program? (main)
8) Which data type is used in Line 3? (String)
9) Which keyword represents the parameters passed to the main method? (args)
10) What is the object name of Scanner class? (scanner)
11) Which keyword is used to create a new object or instance? (new)
12) Which keyword represents the standard input stream that passes the predefined object for creating an object of Scanner class? (System.in)
13) Which data type is used in Line5? (int)
14) Which method is used to scan the next token of the integer input? (nextInt)

It is noted that the correct answer is indicated inside the blankets. Fig. 2 illustrates the user interface for this GUP instance.

IV. GUP Instance Generation Algorithm

In this section, we introduce the *GUP instance generation algorithm* to assist a teacher to generate a new GUP instance among the selected source code.

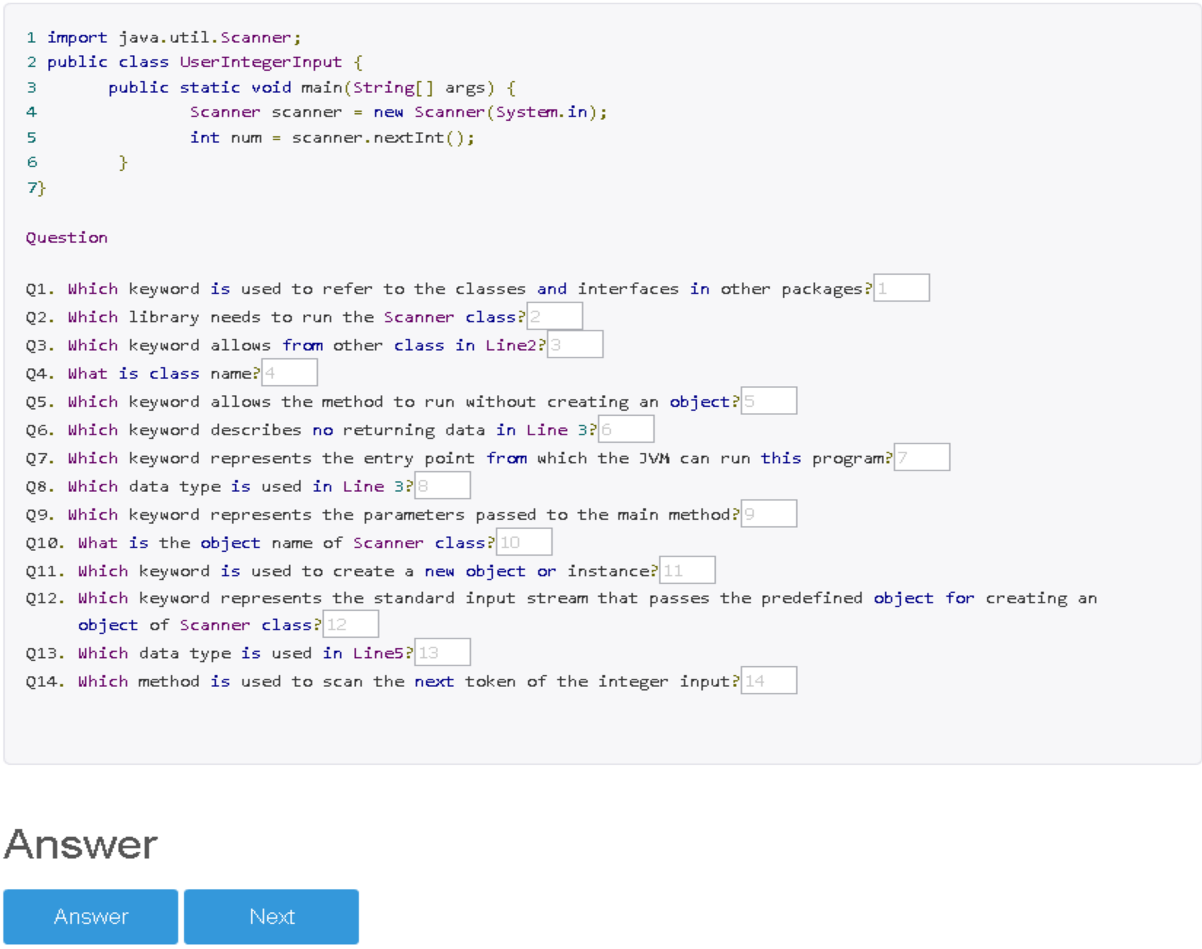

Figure 2. GUP user interface in JPLAS.

A. Input Files

To use the algorithm, a teacher needs to prepare the file of the source code that covers the grammar concepts to be studied by students through solving the GUP instance. Then, the algorithm will read the source code file and generate the GUP instance file through the procedure in Section IV-D. The files for the *keyword list* and the *question list* must be prepared beforehand for this algorithm.

B. Keyword List

TABLE I. KEYWORD LIST

Type	Question	Answer	Keywords
1	Unique	Unique	for, while, do, try, catch, ArithmeticException, NullPointerException, finally, throw, throws, read, IOException, close, void, static, main, args, java.util.Scanner, new, System.in, nextInt, nextLine, extends, this, implements, return, abstract, instanceof, valueOf
2	Unique	in code	class, interface, package, Scanner
3	in code	Unique	int, long, short, byte, double, float, String
4	Multiple	Unique	public, private, protected

The proposed algorithm uses the *keyword list* in Table I to list every possible keyword to represent the basic grammar concepts to be studied through solving GUP instances. The keywords are categorized into the four types, depending on the uniqueness of the selected question and correct answer.

1) For the *type-1* keyword, both the question and the correct answer are unique for any source code. The keyword itself becomes the correct answer.

2) For the *type-2* keyword, the question is unique for any source code. But, the correct answer must be found from the source code using the keyword.

3) For the *type-3* keyword, the question contains the line number information in the source code to specify the keyword, since otherwise, the question can be related to other keywords in the code. The line number must be found from the source code to complete the question. The correct answer is unique for any source code, where the keyword itself is the correct one.

4) For the *type-4* keyword, the question has multiple choices, depending on the concept that the teacher wants to ask to students. One question contains the line number information in the source code to specify the keyword, which must be identified from the source code to complete the question. The correct answer is unique for any question, where the keyword itself is the correct one.

TABLE II. QUESTION LIST

Type	Keywords	Questions
1	for	Which keyword represents Looping?
1	while	Which keyword represents Looping?
1	do	Which keyword always have to execute the loop at least once?
1	try	Which keyword indicates the following lines may cause errors?
1	catch	Which keyword checks the error message when the exceptions occurred in the try block?
1	ArithmeticException	Which exception is thrown when an exceptional condition has occurred in an arithmetic operation?
1	NullPointerException	Which exception is thrown when referring to the members of an object is nothing?
1	finally	Which keyword represents the block that is always executed whether exception is occurred in the try block or not?
1	throw	Which keyword is used in method body to declare the exceptions that can occur in the statements present of the method?
1	throws	Which keyword is used in method signature to declare the exceptions that can occur in the try block?
1	read	Which method reads a byte of data from this input stream?
1	IOException	Which exception is thrown when an input-output operation failed or interrupted?
1	close	Which method is used to terminate this file input stream and releases any system resources associated with the stream?
1	static	Which keyword allows the method to run without creating an object?
1	main	Which keyword represents the entry point from which JVM can run this program?
1	args	Which keyword represents the parameters passed to the main method?
1	extends	Which keyword is necessary to inherit from the super class in the sub class?
1	nextInt	Which method is used to scan the next token of the integer input?
2	class	What is class name?
2	package	What is the package name?
2	scanner	What is the object name of Scanner class?
3	void	Which keyword describes no returning data at Line#?
3	int, long, short, byte, double, float, String	Which data type is used in Line#?
4	public	What is the access modifier at Line #? Which keyword allows from any other class in Line#?
4	private	What is the access modifier at Line #? Which keyword prohibits the access to this code from any other class?
4	Protected	What is the access modifier at Line#? Which keyword allows the access to this code from other class only in the same package?

C. Question List

The *question list* in Table II is used to list the questions for each keyword. It is noted that Table II shows the part of the questions due to the limited space. For the *type-1* or *type-2* keyword, the corresponding question is unique for any source code. For the *type-3* or *type-4* keyword, the question can be completed after locating the line number of the source code where the keyword appears. In the *question list*, the line number is described by # that must be replaced by the line number.

D. GUP Generation Procedure

A GUP instance file is generated through the following procedure:

1) Read a Java source code file.
2) Extract he keywords in the *keyword list* from the source code.
3) Select the question in the *question list* that corresponds to each extracted keyword.
   3-1) If multiple questions are registered in the *question list* for the keyword, one of them is randomly selected.
   3-2) If the question needs to find the line number of the source code for the keyword, it is found and included in the question.
4) Find the element as the correct answer from the source code.
5) If the same pair of the question and the correct answer is selected, discard them as the duplicated question.
6) Output the GUP instance file of the source code, the questions, and the correct answers.

For example, for **source code1** in Section III-B, the following keywords are extracted:

- Import, java.util.Scanner, public, class,
- UserIntegerInput, public, static, void, main, String,
- args, Scanner, scanner, new, Scanner, System.in, int,
- num, scanner, nextInt.

Then, the 14 questions and the correct answers in Section III-B are selected from these keywords.

V. EVALUATION

In this section, we evaluate our proposal through applications to undergraduate students in two universities in Indonesia and Myanmar respectively.

A. Application to Students in Indonesia University

First, we apply the proposal to students in a university in Indonesia.

1) Application Overview: We generated 20 GUP instances with 99 questions from different source codes that cover the topics of the basic Java grammar. Here, the 13 keywords, *for, while, do, try, catch,* *ArithmeticException, NullPointerException, finally, throw, throws, read, IOException, close*, are not included in these source codes. It is confirmed that the generated questions are suitable for the level of novice students. Then, we required 100 undergraduate Indonesia students to solve the problems using the offline answering function [11].

The results show that 87 students among them have learned the fundamentals of Java grammar and should pursue an advanced level. However, the remaining 13 students fail to achieve the required level. Thus, the teacher needs to take care of these students and provide additional instructions and assignments.

2) Correct Answer Results: Based on the result, first, we analyzed the performance by the number of correctly solved questions, then divided the 100 students into five groups. Table III shows the range of the number of correctly solved questions, the number of students, the range of the number of instances attempted to be solved, and the average number of answer submission times per student with its standard deviation for each group.

TABLE III. CORRECT ANSWER RESULTS

Group	# of solved questions range	# of students	# of attempted instances range	ave. # of submissions (SD)
A	99	51	20	86.0 (58.6)
B	98	10	20	99.9 (43.8)
C	90-97	22	19	90.1 (83.1)
D	89-51	9	19-14	63.7 (49.2)
E	28-0	8	4-0	17.7 (14.0)

The table indicates that in group A, 51 students among 100 solved all the questions correctly. In group B, 10 students did not solve only one question where they attempted to solve all the 20 GUP instances. In group C, 22 students solved 90 or more questions correctly where they did not attempt to solve one GUP instance. In group D, 9 students solved less than 90 questions correctly where they did not try to solve several GUP instances. In group E, 8 students only solved less than 28 questions correctly where they attempted to solve a few GUP instances. Hence, the teacher may spend more time on taking care of the students in group E.

B. Submission Times Results

Next, we analyzed the performance according to the number of times of answer submission. JPLAS allows the students to submit their answers to the server at any time, because it is the tool for self-studies. Table IV shows the range of the answer submission and the corresponding number of students.

The table suggests that in group I, 29 students among 100 submitted their answers 50 or less times to solve 20 GUP instances, which indicates less than 2.5 submissions for each instance on average. These students have thoroughly understood the questions, and carefully prepared the answers before submissions. In groups V, VI and VII, six students submitted answers 200 or more times, which indicates more than 10 submissions for each instance on average. It seems that these students did not well understand the questions and submitted their answers randomly. Furthermore, in group VIII, 7 students did not reach even 20 submissions. The teacher needs to care these 13 students.

TABLE IV. SUBMISSION TIMES RESULTS

Group	Submission times range	# of students
I	20-50	29
II	50-100	37
III	100-150	15
IV	150-200	6
V	200-250	3
VI	250-300	2
VII	300- 350	1
VIII	0- 19	7

C. Application to Students in Myanmar University

Next, we apply the proposal to students in one university in Myanmar.

1) Application Overview: In this application, we additionally generated 8 GUP instances with 30 questions from source codes that cover the 13 keywords that were not included in the previous 20 source codes. Then, we asked 29 undergraduate students to solve the instances using the offline answering function, where among them, only 15 students solved both the previous 20 instances and the additional 8 instances.

The results confirm that all the students have acquired the fundamentals of Java grammar and may continue studying Java programming. This difference from the Indonesia students may come from the difference in the motivations of the participated students between the two universities. In the Indonesia university, the teacher requested all the students in the class to answer the GUP instances. On the other hand, in the Myanmar university, the teacher allowed the students to do so voluntarily. Thus, only the self-motivated students might answer the instances.

2) Correct Answer Results: Tables V and VI show the range of the number of correctly solved questions, the number of students, the range of the number of instances attempted to be solved, and the average number of answer submission times per student with its standard deviation for each group, for the previous instances and the additional instances, respectively. Table V indicates that all the students solved 90 or more questions among the 99 correctly where they did not attempt to solve one or two instances. Also, Table VI signifies that all the students solved 28 or more questions among the 30 correctly where they tried to solve all the instances.

3) Submission Times Results: Tables VII and VIII demonstrate the range of the times of answer submission and the corresponding number of students, for the previous instances and the additional instances, respectively. Table VII suggests that every student submitted answers less than 8 times on average for each of the previous 20 instances, and Table VIII does that every student submitted answers less than 7 times on average for each of the additional 8 instances.

TABLE V. Correct Answer Results for Previous Instances

Group	# of solved questions range	# of students	# of attempted instances range	ave. # of submissions (SD)
A	99	18	20	71.4 (40.3)
B	98	3	20	66.7 (17.4)
C	97- 90	8	19 - 18	48.9 (20.2)

TABLE VI. Correct Answer Results for Additional Instances

Group	# of solved questions range	# of students	# of attempted instances range	ave. # of submissions (SD)
A	30	12	8	19.4 (12.7)
B	29	0	8	0.0 (0.0)
C	28	3	8	11.7 (2.9)

TABLE VII. Submission Times Results for Previous Instances

Group	submission times range	# of students
I	20 – 50	12
II	50 – 100	12
III	100 – 150	5

TABLE VIII. Submission Times Results for Additional Instances

Group	submission times range	# of students
I	8 – 30	12
II	30 – 40	2
III	40 – 50	1

VI. Related Works

In this section, we discuss related work in literature. In [12], McIver *et al.* discussed seven undesirable features in programming languages used to teach first-time programmers: (1) less is more, (2) more is more, (3) grammatical traps, (4) hardware dependence, (5) backwards compatibility, (6) excessive cleverness, and (7) violation of expectations. They proposed seven language design principles: (1) start where the novice is, (2) differentiate semantics with syntax, (3) make the syntax readable and consistent, (4) provide a small and orthogonal set of features, (5) be especially careful with I/O, (6) provide better error diagnosis, and (7) choose a suitable level of abstraction.

In [13], Galvez *et al.* presented a blended e-learning experience using an Object Oriented Programming learning tool called *OOPS (Object Oriented Programming System)* and a web-based testing system called *SIETTE*. OOPS can diagnose knowledge levels of students, and generate feedback and hints to help them understand and clear up misconceptions. It is found that most of students have improved scores after solving problems in OOPS.

In [14], Rex *et al.* analyzed the types of errors committed by novice Java programmers and found that there were five categories. Four of them were symbol related or keyword-related errors (invalid symbols or keywords, mismatched symbols, missing symbols, and excessive symbols) and the last was naming-related error (inappropriate naming error).

In [15], Okimoto *et al.* developed a learning support system for C programming that will automatically generate a source code to facilitate the programming instruction through code reading, which is effective for improving basic skills by tracing and debugging, supporting novice learners who feel difficult in programming concept. The system proposes a question that requires learners to answer the proper value of a variable after the execution of the code. The authors utilized the system in a programming course with 108 first year students majoring in informatics, and clarified that the program reading comprehension is challenging for novices.

In [16], Jegede *et al.* analyzed error types and patterns by undergraduate students in Java programming based on fundamental concepts of methods and classes, decision making, object concepts, and looping. The results revealed that similar error types were found across ability levels where students should be instructed based on achievement levels, and learning Java programming should be accomplished with an unintelligent editor.

VII. Conclusion

This paper proposed the *Grammar-Concept Understanding Problem (GUP)* as a new type exercise problem in JPLAS. A GUP instance gives questions on grammar concepts or behaviors in the code. Each answer may be a number, a word, or a short sentence, whose correctness is marked through string matching with the correct answer.

For evaluations, 28 GUP instances with a total of 129 questions from simple source codes on fundamental Java grammar were generated and assigned to 100 students in one university in Indonesia and to 29 students in one university in Myanmar respectively. The results show that the proposal is effective in identifying the students who do not understand Java programming well and need more instruction from the teacher.

For the limitations, this algorithm involves several limitations. When the same keyword appears in the source code again, the algorithm will generate the same question for the keyword. A large number of Java source codes may have common keywords. For those common keywords, the corresponding same questions are duplicated even for one problem. To avoid it, the teacher needs to remove the duplicate or redundant questions before presenting the GUP instance to students.

In future works, we will generate a variety of questions for advanced Java programming topics using various codes and apply them to students in Java programming courses.

Conflict of Interest

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

All the authors conducted the research together. Particularly, Soe Thandar Aung, Nobuo Funabiki, Htoo Htoo Sandi Kyaw, and Wen-Chung Kao generated the problems, analyzed the data, and wrote the paper. Yan Watequlis Syaifudin, Shune Lae Aung, and Nem Khan Dim assigned the problems to their students and collected the data. All the authors had approved the final version.

ACKNOWLEDGMENT

Foremost, I would like to express my sincere gratitude to my supervisor Prof. Nobuo Funabiki for the continuous support of my research, for his patience, motivation, enthusiasm, and immerse knowledge. His guidance helped me in all the time of research and writing of this paper. Besides my supervisor, I would like to thank the rest of my research committee Prof. Wen-Chung Kao and Htoo Htoo Sandi Kyaw for their encouragement and insightful comments and enlightening me the first glance of research. My sincere thanks also goes to Yan Watequlis Syaifudin, Shune Lae Aung and Nem Khan Dim for offering me to assign the problems to the students in their universities and collect the data.

REFERENCES

[1] The Top Programming Languages.

[2] S. I. Ao, *et al.* (2018). *IAENG Transactions on Engineering Sciences: Special Issue for the International Association of Engineers Conferences 2016 (Volume II)*, World Sci. Pub. [Online]. 517-530. Available: http://www.worldscientific.com/worldscibooks/10.1142/10727

[3] N. Ishihara, N. Funabiki, M. Kuribayashi, and W. C. Kao, "A software architecture for Java programming learning assistant system," *Int. J. Comput. Soft. Eng.*, vol. 2, no. 1, 2017.

[4] K. K. Zaw, N. Funabiki, and W. C. Kao, "A proposal of value trace problem for algorithm code reading in Java programming learning assistant system," *Inform. Eng. Exp.*, vol. 1, no. 3, pp. 9-18, 2015.

[5] N. Funabiki, K. K. Zaw, N. Ishihara, and W. C. Kao, "A graph based blank element selection algorithm for fill-in-blank problems in Java programming learning assistant system," *IAENG Int. J. Comput. Sci.*, vol. 44, no. 2, pp. 247-260, 2017.

[6] H. H. S. Kyaw, S. T. Aung, H. A. Thant, and N. Funabiki, "A proposal of code completion problem for Java programming learning assistant system," in *Proc. VENOA-2018*, July 2018, pp. 855-864.

[7] N. Funabiki, S. He, H. H. S. Kyaw, and W. C. Kao, "A proposal of code correction problem for Java programming learning assistant system," in *Proc. VENOA-2019*, 2019, pp. 671-680.

[8] N. Ishihara, N. Funabiki, and W. C. Kao, "A proposal of statement fill-in-blank problem using program dependence graph in Java programming learning assistant system," *Inform. Eng. Exp.*, vol. 1, no. 3, pp. 19-28, 2015.

[9] N. Funabiki, Y. Matsushima, T. Nakanishi, K. Watanabe, and N. Amano, "A Java programming learning assistant system using test-driven development method," *IAENG Int. J. Comput. Sci.*, vol. 40, no. 1, pp. 38-46, 2013.

[10] P. J. Deitel and H. M. Deitel, *Java: How to Program*, 9th ed., Prentice Hall, 2011.

[11] N. Funabiki, H. Masaoka, N. Ishihara, I. W. Lai, and W. C. Kao, "Offline answering function for fill-in blank problems in Java programming learning assistant system," in *Proc. IEEE ICCE-TW 2016*, May 2016, pp. 324-325.

[12] L. McIver and D. Conway, "Seven deadly sins of introductory programming language design," in *Proc. Int. Conf. Soft. Eng.: Edu Pract.*, Jan. 1996, pp. 309-316.

[13] J. Galvez, E. Guzmn, and R. Conejo, "A blended E-learning experience in a course of object oriented programming fundamentals," *Know.-Base. Syst.*, vol. 22, no. 4, pp. 279-286, May 2009.

[14] P. Rex, Bringula, G. M. A. Manabat, M. A. A. Tolentino, and E. L. Torres, "Predictors of errors of novice Java programmers," *World J. Edu.*, vol. 2, no. 1, Feb. 2012.

[15] K. Okimoto, S. Matsumoto, S. Yamagishi, and T. Kashima, "Developing a source code reading tutorial system and analyzing its learning log data with multiple classification analysis," *Art. Life Robot.*, vol. 22, pp. 227-237, 2017.

[16] P. O. Jegede, E. A. Olajubu, A. O. Ejidokun, and I. O. Elesemoyo, "Concept based analysis of Java programming errors among low, average and high achieving novice programmers," *J. Inform. Tech. Edu.: Innov. Pract.*, vol. 18, pp. 49-59, June 2019.

S. Thandar Aung received the B.E. degree in Information Technology from Thanlyin Technological University, Yangon, Myanmar, in 2017. She is currently a research student in Department of Electrical and Communication Engineering at Okayama University, Japan. Her research interests include educational technology.

N. Funabiki received the B.S. and Ph.D. degrees in mathematical engineering and information physics from the University of Tokyo, Japan, in 1984 and 1993, respectively. He received the M.S. degree in electrical engineering from Case Western Reserve University, USA, in 1991. From 1984 to 1994, he was with Sumitomo Metal Industries, Ltd., Japan. In 1994, he joined the Department of Information and Computer Sciences at Osaka University, Japan, as an assistant professor, and became an associate professor in 1995. He stayed at University of Illinois, Urbana-Champaign, in 1998, and at University of California, Santa Barbara, in 2000-2001, as a visiting researcher. In 2001, he moved to the Department of Communication Network Engineering (currently, Department of Electrical and Communication Engineering) at Okayama University as a professor. His research interests include computer networks, optimization algorithms, educational technology, and Web technology. He is a member of IEEE, IEICE, and IPSJ.

Y. Watequlis Syaifudin received the B.S. degree in Informatics from Bandung Institute of Technology, Indonesia, in 2003, and the M.S. degree in Information Technology from Sepuluh Nopember Institute of Technology, Surabaya, Indonesia, in 2011, respectively. In 2005, he joined State Polytechnic of Malang, Indonesia, as a lecturer. He is currently a Ph.D. candidate in Graduate School of Natural Science and Technology at Okayama University, Japan. His research interests include educational technology and database systems. He is a student member of IEICE.

H. Htoo Sandi Kyaw received the B. E. and M. E. degrees in information science and technology from University of Technology (Yatanarpon Cyber City), Myamar, in 2015 and 2018, respectively. She is currently a Ph.D. candidate in Graduate School of Natural Science and Technology at Okayama University, Japan. Her research interests include educational technology and Web application systems. She is a student member of IEICE.

S. Lae Aung received the B.S. degree in computer science from University of Yadanabon, Mandalay, Myanmar, in 2012, and the M.S. degree in computer science from University of Yangon, Myanmar, in 2015. In 2017, she joined Department of Computer Studies at University of Yangon, Myanmar, as a lecturer, where currently, she is also a Ph.D. candidate. Her research interests include educational technology, assistive technology, and human computer interaction.

N. Khan Dim received the B.S. and M.S. degrees in computer science from University of Yangon, Myanmar, in 2008 and 2011, and Ph.D. in computer science from Kochi University of Technology, Japan, in 2016, respectively. She is currently a lecturer in Department of Computer Studies at University of Yangon, Myanmar. Her research interests include human-computer interaction and assistive technology.

W. Chung Kao received the M.S. and Ph.D. degrees in electrical engineering from National Taiwan University, Taiwan, in 1992 and 1996, respectively. From 1996 to 2000, he was a Department Manager at SoC Technology Center, ERSO, ITRI, Taiwan. From 2000 to 2004, he was an Assistant Vice President at NuCam Corporation in Foxlink Group, Taiwan, where he was responsible for leading embedded software team to develop digital still/video cameras. In 2002, he was also invited to form SiPix Technology Inc., Taipei, Taiwan, where he was in charge of setting up the research team of the company and studying flexible electrophoretic display. Since 2004, he has been with National Taiwan Normal University (NTNU), Taipei, Taiwan, where he is currently the Research Chair Professor at Department of Electrical Engineering and the Dean of College of Technology and Engineering. His current research interests include system-on-a-chip (SoC) as well as embedded software design, flexible electrophoretic display, machine vision system, digital camera system, and color imaging science.