K-002

An Implementation of Hint Function for Code Completion Problem in Java Programming Learning Assistant System

Htoo Htoo Sandi Kyaw‡ Nobuo Funabiki‡ Minoru Kuribayashi‡

pxs93q36@s.okayama-u.ac.jp funabiki@okayama-u.ac.jp

Abstract To assist self-studies of Java programming, we have developed a Web-based *Java Programming Learning Assistant System (JPLAS)* that provides various programming exercise problems such as the *code completion problem (CCP)* to cover different learning stages of Java programming. CCP requests a student to complete a source code that has several missing elements, where the correctness is verified through comparison with the original one. Unfortunately, our preliminary applications of CCP found that some students could not complete some statements. In this paper, we implement a *hint function* for CCP that highlights a missing element location by *blue* and a mistyped element by *purple* in the student answer. We confirm the effectiveness of the proposal through applications to students.

1. Introduction

To assist self-studies of Java programming by students, we have developed a Web-based *Java Programming Learning Assistant System (JPLAS)*. JPLAS provides several types of programming problems such as the *code completion problem (CCP)* [1], to support a variety of students at different learning stages. The server platform of JPLAS adopts the *Linux* for the operating system, *Tomcat* for the application server, and *MySQL* for the database. The application programs in the server are made by *JSP* and *Java,* and those in the browser are by *HTML/CSS* and *JavaScript.*

In CCP, a teacher generates a problem code from a sample *source code* that can be taken from a textbook or a Web site for Java programming by applying the graph-based *blank element selection algorithm* [2]. CCP does not show explicitly where the missing elements exist to a student in the problem code, where an element may be filled in at any place of each statement. The correctness of the whole statement checked through *string matching* between the filled statement and the corresponding statement of the original source code. It is noted that tabs and spaces are removed in this matching to avoid the unnecessary difficulty. In CCP, a student is requested to carefully read the problem code to understand the grammar and code structure to fill in the missing elements with proper words. For studying *readable code* writing [3], the answer code must follow the *coding rules* composed of *naming rule, coding styles, and potential problems* [4].

In our previous study [5], we proposed three improvements to help students in finding correct answers for CCP. However, it was found that some students still could not complete some statements in the given code.

† Graduate School of Natural Science and Technology Okayama University, Japan

In this paper, we implement a *hint function* for CCP to further help students by suggesting the places of the code incompleteness. This function highlights a missing element location by *blue* and a mistyped element by *purple*. We confirm the effectiveness of the proposal through applications to students in Japan and Myanmar.

2. Review of CCP

In this section, we review the instance generation and the answering interface for CCP. A CCP instance can be generated through the following steps:

1. Select a Java source code from a Web site or a textbook that is worth of reading to study the current topic.
2. Apply the *naming rules test* in the *coding rule check function* to the source code and fix the errors if found.
3. Apply the *coding styles test* to the code and fix the errors if found.
4. Apply the *potential problems test* to the code and fix the errors if found.
5. Register every statement in the code as the correct answer unit in the string matching.
6. Apply the *blank element selection algorithm* to select the blank elements from the code.
7. Remove the selected elements from the code to generate the *problem code*.

Figure 1 illustrates the answering interface for CCP on a Web browser. “Problem Code” shows the problem code for the assignment for CCP. “Answer Code” shows the answer forms where each line corresponds to one statement in the problem code that may have missing elements. A student is requested to complete every statement by filling in all the missing elements at the form.

When a student clicks the answer button, the correctness is checked at the server. If at least one element except for a space and a tab in the filled statement is different from the correct one, the corresponding form is highlighted by *pink*. If one tab or space is missing or excess in the answer statement, it is highlighted by *yellow*. Otherwise, the form is not highlighted.

3. Implementation of Hint Function

In this section, we present the implementation of the hint function for CCP.

When a student cannot solve a problem after submitting the answer at a certain number of times, the *hint function* becomes available by requesting the incorrect statement number. Figure 1 also shows this hint function result. Here, a student typed “3” as the incorrect statement number. Then, the corresponding hints appear, where the location for the missing element, *int*, is highlighted by *blue*, and the mistyped element, *tmp*, is highlighted by *purple*. It is noted that *tmp* should be *temp*. The correct answer is “ int num = 121, temp = num, ans = 0;”. By referring to these

highlighted elements, it is expected that the student can solve the problem.

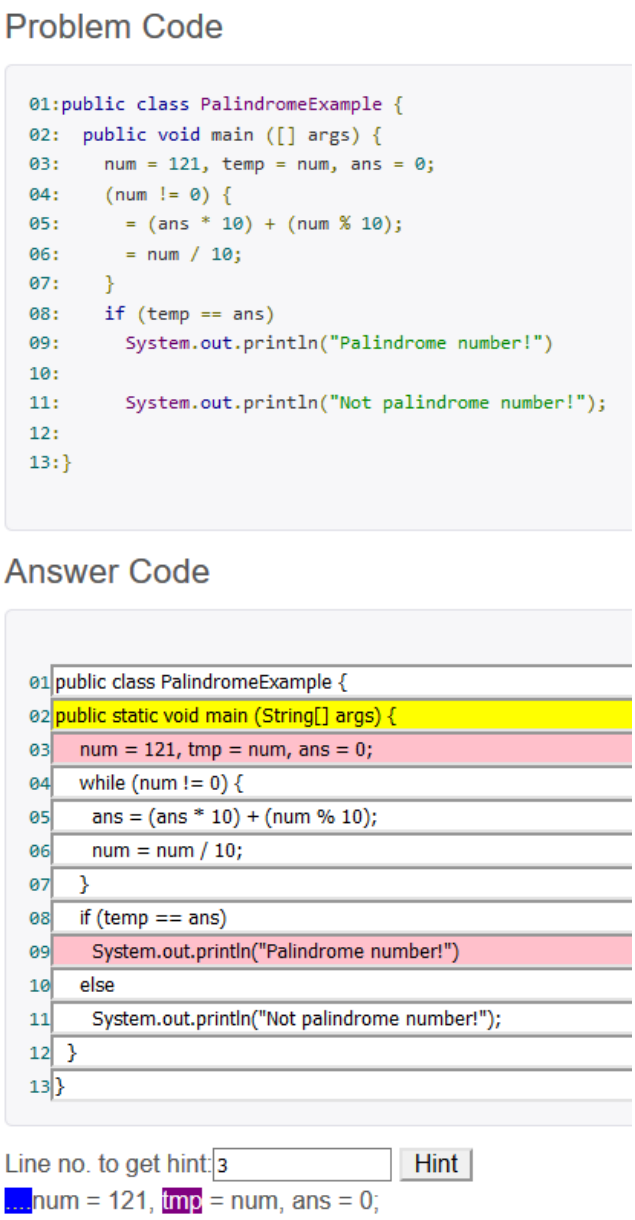

Fig 2　CCP answering interface.

4. Evaluation

In this section, we evaluate the *hint function* for CCP through applications to 12 students in Japan and Myanmar.

We generated six code completion problem using the same source codes in [5], to compare the solution performance between by using hint function and without using hint function. These source codes cover the topics on *variable*, *array*, *collection*, *recursive*, and *polymorphism.*

Table 1 compares the minimum, the maximum, the average, and the standard deviation (SD) of the correct solution rates (%) among the 12 students. It indicates that by using the hint function, the minimum rate was improved from 93.33% to 99.20% and the SD was reduced from 1.94 to 0.25, while the maximum and average rate was not changed. From the results, the hint function helps low-level students improving their solution performances.

However, the correct solution rate does not reach 100% for every student. Table 2 shows the total number of incorrect answers by the students and the number of incorrect students, in addition to the number of lines in the source code (LOC) and the number of missing elements, for each problem. It suggests that students could not solve *Problem 5*.

Here, we analyze the reason. This problem uses the source code that generates *Fibonacci series*, 0, 1, 1, 2, 3, 5, 8, 13, 21, …, recursively. In *Fibonacci series*, each subsequent number becomes the sum of the previous two numbers. Besides, this source code uses *BigInteger* class that can be used for mathematical operations that involve very big integers such that they are outside the limitation of primitive data types. Due to the use of the recursive function and *BigInteger* class, this problem becomes difficult for students. To let them study these concepts, we will generate more CCP using source codes that contain them.

Table 1 Correct solution rates (%)

	without using hint function	by using hint function
min.	93.33	99.20
max.	100	100
ave.	98.85	99.89
SD	1.94	0.25

Table 2 Assigned problems and results.

Problem ID (PID)	LOC	#of missing elements	Total # of incorrect answers	#of incorrect students
1	52	15	0	0
2	44	19	0	0
3	42	24	0	0
4	43	6	0	0
5	31	16	3	2
6	28	18	0	0

5. Conclusion

This paper presented the implementation of the *hint function* for the code completion problem in the Java programming learning assistant system. The effectiveness is confirmed through applications to students.

References

[1] 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. VENOA2018, pp. 855-864, July 2018.
[2] N. Funabiki, Tana, 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, May 2017.
[3] D. Boswell and T. Foucher, The art of readable code, O’Reilly, 2011.
[4] N. Funabiki, T. Ogawa, N. Ishihara, M. Kuribayashi, and W. –C. Kao, “A Proposal of coding rule learning function in Java programming learning assistant system,” in Proc. VENOA, pp. 561-566, July 2016.
[5] H. H. S. Kyaw, N. Funabiki, M. Kuribayashi, and K. K. Zaw, “Three Improvements for code completion problem in Java programming learning assistant system,” in Proc. IPSJ, 2018-4-(13), January 2019.