Information Engineering Express
International Institute of Applied Informatics
2015, Vol.1, No.3, 19 – 28

A Proposal of Statement Fill-in-blank Problem Using Program Dependence Graph in Java Programming Learning Assistant System

Nobuya Ishihara* , Nobuo Funabiki * , Wen-Chung Kao †

Abstract

To assist Java educations, we have developed a Web-based *Java Programming Learning Assistant System (JPLAS)*, that provides the *element fill-in-blank problem* and the *code writing problem*. The former problem is designed for novice students to study the grammar and coding style of Java by filling in correct words to the blanks in a given Java code which are marked automatically through checking for coincidences of correct ones. The latter problem is for students to study writing Java codes for given specifications described in natural language that are automatically verified using the *test-driven development (TDD) method*. Unfortunately, rough transition exists due to different difficulties found between the two problems. In this paper, we propose the *statement fill-in-blank problem* in JPLAS by asking students to fill in the gap and write one whole statement in a code where the correctness is verified by the TDD method. The blank statement is selected by generating the *Program Dependence Graph (PDG)* of the code and finding the statement that has the largest dependence in PDG. We verify the effectiveness through applications of the Java programming course in our department.

Keywords: JPLAS, Java programming education, statement fill-in-blank problem, test-driven development method, program dependence graph.

1 Introduction

Java has been used as a reliable, portable, and practical programming language among many important practical ICT systems, including Web systems, enterprise servers, smart phones, and embedded systems [1]. As a result, Java has been educated in many universities and professional schools to foster professional Java programmers into societies. The effective education of Java programming has been essential in meeting strong demands for high-quality Java programmers and engineers from societies.

To enhance educational effects of Java programming courses, we have developed a Web-based *Java Programming Learning Assistant System (JPLAS)* that can assist self-study

* Department of Electrical and Communication Engineering, Okayama University, Okayama, Japan
† Department of Electrical Engineering, National Taiwan Normal University, Taipei, Taiwan

students and reduce workloads of teachers. JPLAS provides the *element fill-in-blank problem* [2] and the *code writing problem* [3]. The former problem is designed for novice students to study the grammar and writing style of Java by filling in correct words to the blanks in a given Java code, which are marked automatically through checking for coincidences of correct ones. The latter problem is for students to study writing Java codes for given specifications described in natural language, which are automatically verified using the *test-driven development (TDD) method* [4]. A software tool called *JUnit* [5] is adopted here to test whether submitted Java codes from students satisfy the test cases in the test code that is prepared by the teacher.

Unfortunately, the transition from the first problem to the second is not smooth due to difference of the difficulties. For the element fill-in-blank problem, a student can mechanically solve it by selecting a possible element for each blank without thinking of completing the statements. As a result, a student may not reach the level of writing a code from scratch, even though he or she has solved many element fill-in-blank problems. However, it is necessary for such a student to write a whole statement by himself or herself.

In this paper, we propose the *statement fill-in-blank problem* in JPLAS to fill in the gap between the two problems. To address this problem, a student needs to write one whole statement that is blanked in a given Java code. Because there can be multiple correct answers even to one statement, the correctness of the answer is verified using the TDD method. To generate a proper problem, the blank statement is selected by using the *Program Dependence Graph (PDG)* of the code [6][7] and in finding the statement that has the largest dependence with other ones.

The statement fill-in-blank problem can help a student to study the *code reading* that is an essential way of mastering proper writing styles of Java codes by following them in high quality codes. It is also indispensable for a student to understand and modify existing codes that can happen in real worlds.

To evaluate the proposed statement fill-in-blank problem in JPLAS, we generated 39 problems which were assigned to 45 students who are currently taking the fundamental Java programming course in our department. Through observing reduced time in solving problems by students and the correlation between the number of solutions as well as the average of final grades, these results confirm the effectiveness of our proposal in this paper.

The rest of this paper is organized as follows: Sections 1 and 2 review JPLAS and the related works respectively. Section 3 shows the test-driven development. Section 4 proposes the statement fill-in-blank problem. Section 5 discusses evaluation results. Section 6 concludes this paper with some future works.

2 Related Works

In this section, we survey some studies of computer-aided instruction (CAI) systems that offer automatic marking functions of answers. No CAI system provides the similar problem like the statement fill-in-blank problem from our survey.

In [8], Delev et al. introduced *E-Lab* for solving and auto-grading programming problems for introductory programming courses to C, C++, and Java. *E-Lab* only provides the code writing problem. Students submit codes in a Web browser using a Web-based code editor. The codes are stored, compiled, and executed on the server, and are assessed by a simple black-box testing method using input and output text files. This system keeps records of all the attempts from students using the version control system (Git).

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

In [9], Kitaya et al. proposed a Web-based system that automatically scores Java programming assignments. This system receives a code submitted by a student and returns the test result immediately. The test consists of the compiler check, the JUnit test for a code with multiple classes/methods, and the result test for a code with only a main method which reads/writes data from/to the standard input or output devices.

In [10], Verdú et al. proposed *EduJudge* that integrates the existing on-line programming trainer *UVA On-line Judge* into the e-learning platform *Moodle* along with the competitive learning tool *QUESTOURnament*. The *EduJudge* system allows teachers to apply different pedagogical approaches, makes problems become easy to search through an effective search engine, and provides an automated evaluation of the solutions.

In [11], Teramoto et al. developed a Problem Solving Environment (PSE) for education and learning support called *TSUNA-TASTE* that collects the system-usage information of the students and stores them in the database. It can collect process names, active window titles, typing information of the key-board, the mouse information, and the starting and error information of the C compiler.

In [12], Klašnja-Milićević et al. presented an idea of integrating a recommender system into an existing Web-based Java tutoring system that introduces intelligence and is made adaptive to meet every learner's need and interactions. The architecture contains elements of two different categories for e-Learning systems.

In [13], Joy et al. described a Web-based submission and assessment system called *BOSS* that supports coursework assessments through collecting submissions, performing automatic tests for correctness and quality, checking for plagiarism, and providing an interface for marking and delivering feedback. They described how automated assessments are incorporated into *BOSS*. They also defined measures of a good program: comments in code, code style, correctness of code, code structure, code testing, use of external libraries, documentation, choice and efficiency of algorithm and code as well as attempts to incorporate tools to support automatic measurements.

In [14], Caiza et al. reviewed latest automatic tools for programming assignment assessments. They depicted requirements and key features of these tools. These assessments also carried out comparisons and analysis where the results indicate that the major issue is the lack of a common grading model.

In [15], Shamsi et al. presented a grading system for Java introductory programming courses. The system can both dynamically and statically grade submitted codes based on JUnit framework and their graph representations. The target of their system is similar to our system, but it is rather complex.

3 Test-driven Development Method

In this section, we review the TDD method and its features because the statement fill-in-blank problem uses it to verify answers from students.

3.1 JUnit

In JPLAS, we adopt *JUnit* as an open-source Java framework to support the TDD method. *JUnit* can assist an automatic unit test of a Java code unit *class*. Since *JUnit* has been designed with the Java-user friendly style, including the use of test code programming is rather simple for Java programmers. In *JUnit*, a test can be performed by using a method in

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

the library whose name starts with ”assert”. This paper adopts the ”assertEquals” method to compare the execution result of the source code with its expected value.

3.2 Test Code

A test code should be written by using libraries in *JUnit*. Here, we use the following *MyMath* class source code and introduce how to write a test code. *MyMath* class returns the summation of two integer arguments.

```
1: public class MyMath{
2:     public int plus(int a, int b){
3:         return( a + b );
4:     }
5: }
```

Then, the following test code can test the *plus* method in the *MyMath* class.

```
1: import static org.junit.Assert.*;
2: import org.junit.Test;
3: public class MyMathTest {
4:    @Test
5:     public void testPlus(){
6:         MyMath ma = new MyMath();
7:         int result = ma.plus(1, 4);
8:         asserEquals(5, result);
9:     }
10:}
```

The test code imports *JUnit* packages containing test methods at lines 1 and 2, and declares *MyMathTest* at line 3. *@Test* at line 4 indicates that the succeeding method represents the test method. Then, it describes the procedure for testing the output of the *plus* method.

This test is performed as follows:

1. An instance *ma* for *MyMath* class is generated.
2. The method of the instance *ma.plus* is called with given arguments.
3. The result *result* is compared with its expected value using the *assertEquals* method.

3.3 Features in TDD Method

In the TDD method, the following features can be observed:

1. The test code represents the specifications of the source code, because it must describe every function which will be tested in the source code.
2. The testing process of a source code becomes efficient, because each function can be tested individually.
3. The refactoring process of a source code becomes easy, because the modified code can be tested instantly.

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

4 Proposal of Statement Fill-in-blank Problem

In this section, we propose the *statement fill-in-blank problem* in JPLAS.

4.1 Flow of Statement Fill-in-blank Problem

In a statement fill-in-blank problem, a Java code with one blanked statement is presented to each student. Then, the student is asked to fill the statement that satisfies the specifications for the code that are described by the teacher. The teacher must write the test code representing the specifications, so that the blanked code filled with the answer statement can be tested by using *JUnit*. We adopt the TDD method, because many variations can exist for the answer statement that satisfies the specifications.

The following steps describe the flow of the statement fill-in-blank problem in JPLAS:

1. **Assignment registration**: a teacher registers the title, statement description, Java code, and test code for the assignment.
2. **Blank statement selection**: a teacher selects the blank statement in the Java code using the PDG-based algorithm in Section 4.2.
3. **Assignment answer**: a student fills in one or multiple statements for the blank statement.
4. **Answer verification**: *JUnit* installed at the Web server tests the Java code combined with the answer using *jQuery* [17], and returns the test result.
5. **Solving status confirmation**: both the teacher and the student can confirm the solving status of any student for the assignments. This interface intends for students to compete with each other by knowing the solving situations of other students.

4.2 Blank Statement Selection Algorithm Using PDG

A statement that plays the critical process in the Java code should be selected for the blank one. For this goal, we adopt the PDG to represent the relationships between the statements in the control flows of the code. In the PDG, a vertex represents a statement and an edge serves as the dependency between the corresponding statements in the code. Then, a vertex with the maximum degree is selected as the blank statement.

4.2.1 Basic Rule for PDG Generation

Due to the *data flow dependence*, an edge is generated between the two vertices for statement $s1$ and statement $s2$ when the following conditions are satisfied:

1. A variable $v1$ is defined at $s1$.
2. $v1$ is referred at $s2$.

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

4.2.2 Modification of PDG Generation for Java

As an object-oriented language, an object or instance in Java can be used in addition to a variable. Thus, we modify the rules for generating the PDG:

1. An object is considered as a variable in the *data flow dependence*. When a variable inside an object (member variable) is accessed, it is regarded that the object is accessed there.

2. The following two cases are regarded as the insertion to an object: 1) the object appears at the left side of an assign statement, and 2) the object is called.

3. The following two cases are regarded as the access to an object: 1) the object appears at the right side of an assign statement, and 2) the object is used as an argument of a method.

4.3 Example of PDG

Figure 1 illustrates the PDG for a simple Java code that transfers the data from the input stream "is" to the file "out". Each node represents a statement or vertex, and each directed edge represents the data dependency between statements. Line 2 is selected as the blank statement because it has the maximum degree of seven.

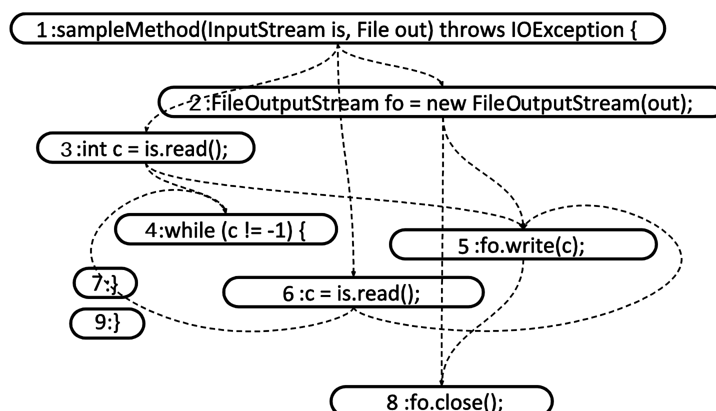

Figure 1: Sample PDG.

5 Evaluation

In this section, we evaluate the statement fill-in-blank problem in JPLAS through applications of 45 students taking the Java programming course in our department. Most of them are sophomores who have studied C and C++ programming for half year respectively.

5.1 Statement Fill-in-blank Problems for Evaluations

For this course, we generated 39 statement fill-in-blank problems using the proposed blank statement selection algorithm.

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

Table 1: Statement fill-in-blank problems for evaluations.

code topic	# of problems	ave. LOC	ave. # of classes	ave. # of methods
Java basic syntax	5	11.8	1.0	1.3
numeric operation	4	31.3	1.0	1.3
Java OOP syntax	4	28.8	1.0	1.3
strings operation	5	24.8	1.0	1.0
file operation	5	50.0	1.4	3.6
parameter check	5	10.4	1.0	1.0
GUI	4	28.8	1.5	5.0
design pattern	7	58.6	3.6	6.7

5.2 Solving Results by Students

Figure 2 shows the distribution of the spending time to solve each problem by students, depicted by "dots", and the number of students who solved each problem correctly, depicted by "bars". The dotted line represents the average spending time where it decreases as students solve more problems. Note that the spending time is limited within 10*min*, to exclude the resting time by students.

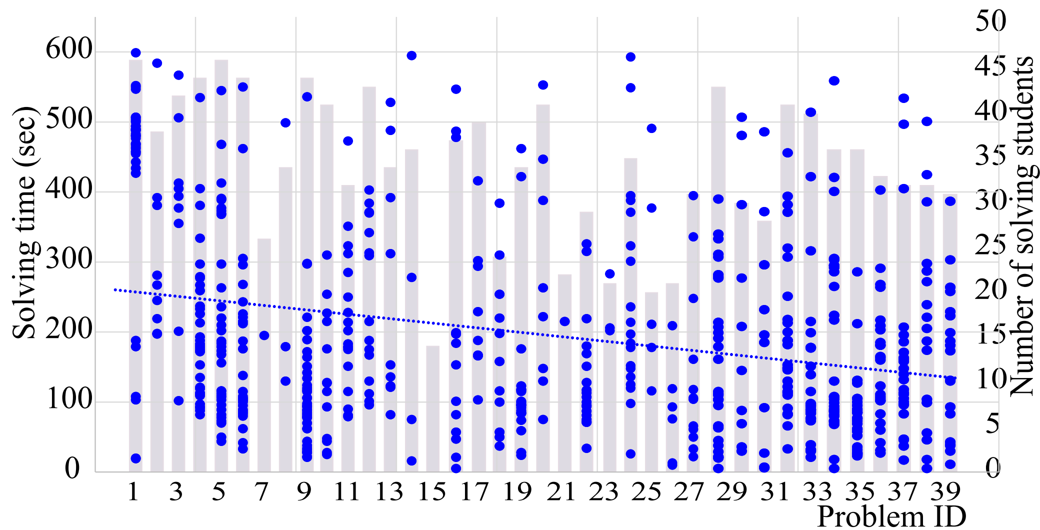

Figure 2: Solving results by students.

Unfortunately, no student could solve *Problem 34* that uses a code to implement *RSS reader*, where the following statement was blanked:

```
119: temp.appendChild(document.createTextNode(dfmt(fd.mp3s.get(i).pubDate)));
```

For this blank statement, fourfold arguments and fourfold dot operations are requested, which can be very hard for students.

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

5.3 Variations in Answer Codes

In these statement fill-in-blank problems, only one statement was blanked and to be filled in by students. Nevertheless, we observed a variety of answer codes in some problems. Actually, among 38 correctly solved problems, 22 problems had two or more correct answer codes to them. We note that if two codes have the simple difference in use of spaces, we regard them as the same one.

5.3.1 Equation

First, there are several variations in describing an equation. Here, the appearing order of variables in the equation is changed, or the different calculation method for the average value is used, or the cast statement is inserted. We note that the number in a bracket represents the number of students who gave the corresponding answer code.

```
original code (31): int middle = (low + high) / 2;
answer A (1): int middle = (high + low) / 2;
answer B (1): int middle = low + (high - low) / 2;
answer C (1): int middle = (int)((high + low) / 2);
```

5.3.2 Constant Definition

Then, there are several variations in defining a constant. Here, the *white* is specified by the color code, or *white* is given by Capitals.

```
original code (19): Color c = Color.white;
answer B (9): Color c = new Color(255,255,255);
answer C (2): Color c = Color.WHITE;
answer D (1): Color c = (Color.white);
```

5.4 Correlation Analysis with Final Assignment

As the final assignment, every student was assigned to write a complete Java code for some service function. A student may freely select to write a game, a sound scale player or even a drawing tool, for example. Then, the code was evaluated by the teacher and the students in terms of the code's uniqueness, its complexity, and completeness. To evaluate the effectiveness of the statement fill-in-blank problem in JPLAS, we analyzed the correlations between the grades of the final assignments and the solving results of statement fill-in-blank problems among students.

To simplify this correlation analysis, we calculated the average grade among the students who solved the same number of problems in JPLAS correctly, and plotted them in a graph. Then, we calculated the correlation coefficient between the two factors.

Figure 3 shows the relationship between the number of correctly solved problems and the average grade of the final assignment. The correlation coefficient is 0.737, which indicates the strong correlation between them. By solving the larger number of statement fill-in-blank problems correctly, students can improve the grade of the final assignment. This result supports the effectiveness of the statement fill-in-blank problem in JPLAS in the Java programming study.

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

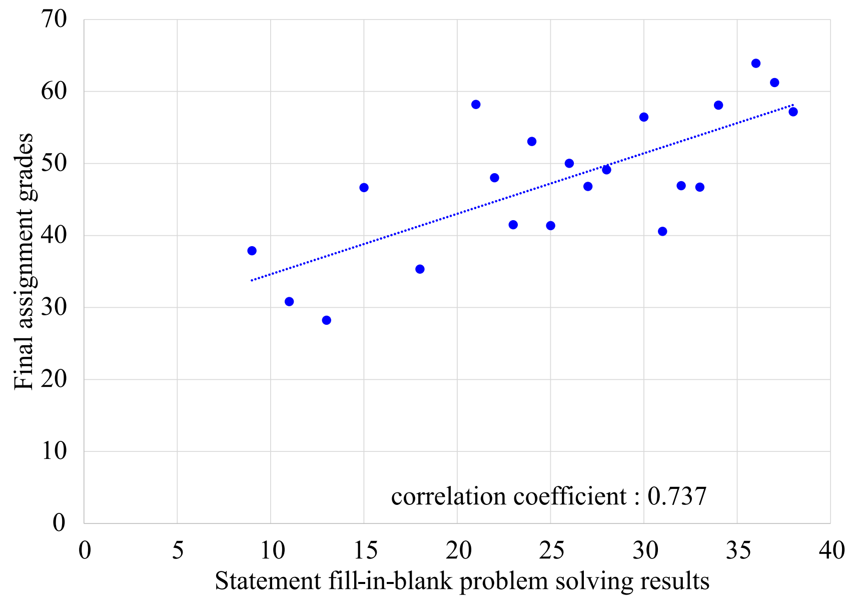

Figure 3: Relationship between statement fill-in-blank problem solving results and final assignment grades.

6 Conclusion

This paper proposes the *statement fill-in-blank problem* in *Java Programming Learning Assistant System (JPLAS)* of filling in the blank statement in a Java code. The blank statement is selected by using the *Program Dependence Graph (PDG)* of the code and in finding the statement that has the largest dependence with other ones. The correctness of the answer is verified at the JPLAS server by the *test-driven development (TDD) method*. The effectiveness is verified through applications of the Java programming course at our department. The future works may include the selection of multiple blank statements, the construction of the problem database, and continuous applications in Java programming courses.

References

[1] The 2015 Top Ten Programming Languages, `http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages/?utm_so`, IEEE Spectrum, July 2015.

[2] Tana, N. Funabiki, and N. Ishihara, " A proposal of graph-based blank element selection algorithm for Java programming learning with fill-in-blank problems," Proc. Int. MultiConf. Eng. Comp. Sci. (IMECS 2015), March 2015, pp.448-483.

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

[4] K. Beck, *Test-driven development: by example*, Addison-Wesley, 2002.

[5] *JUnit*, `http://www.junit.org/`.

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

[6] J. Ferrante, K. J. Ottenstein, and J. D. Warren, “The program dependence graph and its use in optimization,” ACM Trans. Program. Lang. Syst., vol. 9, no. 3, July 1987, pp. 319-349.

[7] A. Kashihara, A. Kashihara, and J. Toyoda, “Making fill-in-blank program problems,” IEICE Tech. Report, vol. 99, no.81, May. 1999, pp. 9-16.

[8] T. Delev and D. Gjorgjevikj, “E-Lab: Web based system for automatic assessment of programming problems,” Proc. ICT Innov. 2012 Web, 2012, pp. 75-83.

[9] H. Kitaya and U. Inoue, “An online automated scoring system for Java programming assignments,” Int. J. Info. Edu. Tech., vol. 6, no. 4, April 2016, pp. 275-279.

[10] E. Verdú et al., “A distributed system for learning programming on-line,” Comp. Edu., vol. 58, 2011, pp. 110.

[11] T. Teramoto, T. Okada, and S. Kawata, “An education-support PSE system: TSUNA-TASTE,” J Conv. Info. Tech., vol. 5, no. 4, June 2010, pp. 216-223.

[12] A. Klašnja-Milićević et al., “Integration of recommendations and adaptive hypermedia into Java tutoring system,” Comp. Sci. Info. Sys., vol. 8, no. 1, Jan. 2011, pp. 211-224.

[13] M. Joy, N. Griffiths, and R. Boyatt, “The BOSS online submission and assessment system,” J. Edu. Res. Comput. (JERIC), vol. 5, no. 3, Sep. 2005, pp. 1-28.

[14] J. C. Caiza and J. M. Del Alamo, “Programming assignments automatic grading: review of tools and implementations,” Proc. Int. Tech., Edu. Develop. Conf. (INTED2013), March 2013, pp. 5691-5700.

[15] F. Shamsi and A. Elnagar, “An intelligent assessment tool for students’ Java submissions in introductory programming courses,” J. Intel. Learn. Syst. App. (JILSA), vol. 4, no. 1, 2012, pp. 59-69.

[16] *CodePress*, `http://sourceforge.net/projects/codepress/`.

[17] *jQuery*, `http://jquery.com/`.

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.