A Proposal of Test Code Generation Tool for Java Programming Learning Assistant System

Nobuo Funabiki, Ryota Kusaka, Nobuya Ishihara
Department of Electrical and Communication Engineering
Okayama University, Okayama, Japan
Email: funabiki@okayama-u.ac.jp

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

***Abstract*—To advance Java programming educations, we have developed the *Java programming learning assistant system (JPLAS)* that can verify the correctness of the code from a student automatically using the *test-driven development (TDD) method*. Then, to register a new assignment in JPLAS, teachers are required to write the *test code* in addition to the *reference source code*. Unfortunately, most teachers at schools are not accustomed to writing test codes. In this paper, we propose a *test code generation tool* that automatically generates the *test cases* from the reference source code by extracting the outputs for given inputs using functions in *JUnit*. As assignments for Java novice students, the code that contains standard inputs/outputs is emphasized. For evaluation, we collected 97 codes containing standard inputs/outputs from Java programming text books or Web sites. The experimental result has shown that the proposed tool correctly generated the test codes for them except for one code using a random generator.**

***Keywords*-Java programming, JPLAS, test code, test case, automatic generation, JUnit**

I. Introduction

Recently, *Java* has been widely used in various practical application systems in societies and industries due to the high reliability, portability, and scalability. Java was selected as the most popular programming language in 2015 [1]. Therefore, there has been a strong demand from industries for Java programming educations. Correspondingly, a plenty of universities and professional schools are currently offering Java programming courses to meet this challenge. A typical Java programming course consists of grammar instructions in the class and programming exercises in computer operations.

To assist teachers and students in advancing in Java programming educations, we have developed the Web-based *Java Programming Learning Assistant System (JPLAS)* [2]-[4]. As a main function, JPLAS provides the *code writing problem* [2] to support students for self-studies of code writing. It can inspire students by offering sophisticated learning environments via quick responses to answers. At the same time, it supports teachers by reducing loads of evaluating codes with prompt and proper feedback or students.

The *code writing problem* is implemented based on the *test-driven development (TDD) method* [5], using an open source framework *JUnit* [6]. *JUnit* automatically tests the codes on the server to verify their correctness using the *test code* when they are submitted by students. Thus, students can repeat the cycle of writing, testing, modifying, and resubmitting codes by themselves, until they can complete the correct codes for the assignments.

To register a new assignment for the code writing problem in JPLAS, a teacher has to prepare a problem statement describing the code specification, a reference source code, and a test code using a Web browser. Then, a student should write a source code for the assignment while referring the statement and the test code, so that the source code can be tested by using the given test code on *JUnit*. The reference source code is essential to verify the correctness of the problem statement and the test code.

However, although teachers at school are able to write source codes without difficulty, most of them are not accustomed to writing a test code that can run on *JUnit*. A teacher may spend much time in struggling to write a test code, and may register an incomplete test code that does not verify specific requirements described in the problem statement correctly. This incomplete test code must be avoided because it may produce inappropriate feedback to a student and undermine confidence to JPLAS. On the other hand, a commercial tool for generating a test code is usually expensive and may not cover a test code that verifies standard input/output functions in a source code. The use of standard input/output functions is expected to be mastered by novice students at the early stage of Java programming educations. Hence, it may not be appropriate for use in JPLAS.

In this paper, we propose a *test code generation tool* for JPLAS that automatically generates a test code from a reference source code prepared by a teacher that contains standard input/output functions. This tool enumerates the *test cases* by observing the output of *JUnit* to each input to the source code. To evaluate the proposed tool, it was applied to 97 source codes in Java programming textbooks or Web sites that contain standard input/output functions. It has been proved that the generated test codes could correctly verify the source codes except for one code.

The rest of this paper is organized as follows: Sections II and III introduce the TDD method and JPLAS. Section IV discusses the background of our proposal. Section V presents the test code generation tool for JPLAS. Section VI shows the evaluation result. Finally, Section VII concludes this paper with some future works.

1550-445X/17 $31.00 © 2017 IEEE
DOI 10.1109/AINA.2017.60

II. Test-driven Development Method

In this section, we introduce the test-driven development method along with its features.

A. Outline of TDD Method

In the TDD method, the test code should be written before or while the source code is implemented, so that it can verify whether the current source code satisfies the required specifications during its development process. The basic cycle in the TDD method is as follows:

1) to write the test code to test each required specification,
2) to write the source code,
3) to repeat modifications of the source code until it passes each test using the test code.

B. JUnit

In JPLAS, we adopt *JUnit* as an open-source Java framework to support the TDD method. *JUnit* can assist the unit test of a Java code unit or a *class*. Because *JUnit* has been designed with the Java-user friendly style, its use including the test code programming is less challenging for Java programmers. In *JUnit*, a test is performed by using a given method whose name starts from "assert". This paper adopts the "assertThat" method to compare the execution result of the source code with its expected value.

C. Test Code

A test code should be written using libraries in *JUnit*. Here, by using the following **source code 1** for *MyMath* class, we explain how to write a test code. *MyMath* class returns the summation of two integer arguments.

source code 1

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

Then, the following **test code 1** can test the *plus* method in the *MyMath* class.

test code 1

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

The names in the test code should be related to those in the source code so that their correspondence becomes clear:

- The class name is given by the *test class name* + *Test*.
- The method name is given by the *test* + *test method name*.

The test code imports *JUnit* packages containing test methods at lines 1 and 2, and declares *MathTest* at line 3. *@Test* at line 4 indicates that the succeeding method represents the test method. Then, it describes the test method.

The code test is performed as follows:

1) to generate an instance for the *MyMath* class,
2) to call the method in the instance in 1) using the given arguments,
3) to compare the result with its expected value for the arguments in 2) using the *assertThat* method, where the first argument represents the expected value and the second one does the output data from the method in the source code under test.

D. Features in TDD Method

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

1) The test code can represent the specifications of the source code, because it must describe the function tested in the source code.
2) The test process for a source code becomes efficient, because each function can be tested individually.
3) The refactoring process of a source code becomes effective, because the modified code can be tested instantly.

Therefore, to study the TDD method and writing a test code is useful even for students, where the test code is equivalent to the source code specification. Besides, students should experience the software test that has become important in software companies.

III. Java Programming Learning Assistant System

In this section, we review the outline of our Java programming learning system *JPLAS*.

A. Server Platform

JPLAS is implemented as a Web application using *JSP/Java/JavaScript*. For the server platform, it adopts the operating system *Linux*, the Web server *Apache*, the application server *Tomcat*, and the database system *MySQL*, as shown in Figure 1.

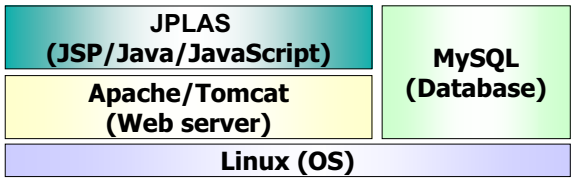

Fig. 1. JPLAS server platform.

B. Teacher Service Functions

JPLAS has user functions both for teachers and students. *Teacher service functions* include the registration of courses, the registration and management of assignments, and the verification of source codes that are submitted by students. To register a new assignment, a teacher needs to input an assignment title, a problem statement, a reference (model) source code, and a test code. After the registration, they are

disclosed to the students except for the source code. Note that the test code must be able to test the model code correctly. Using the correspondence between a source code and a test code in Section II-C, it is possible to automatically generate a template for the test code from the source code. Then, a teacher merely needs to specify concrete values for the arguments in each test method to complete the test code.

To evaluate the difficulty of assignments and the comprehension of students, a teacher can refer to the number of submissions for code testing from each student. If a teacher finds an assignment with plenty of submissions, it can be considered as quite difficult for the students, and should be changed to an easier one. If a teacher finds a student who submitted codes in many times whereas other students did in a few times, this student may require additional assistance from the teacher.

C. Student Service Functions

Student service functions include the view of the assignments and the submission of source codes for the assignments. A student should write a source code for an assignment by referring the problem statement and the test code. It is requested to use the class/method names, the types, and the argument setting specified in the test code. JPLAS implements a Web-based source code editor called *CodePress* [7] so that a student can write codes on a Web browser. All submitted source codes will be stored in the database on the server as a reference for students.

IV. Background of Proposal

In this section, we discuss the background of the proposed test code generation method from a given source code for JPLAS.

A. Target Source Code for Novice Student

At the early stage of the Java programming education, the responsibility of students is to master how to write a source code that contains standard input/output functions. Therefore, teachers in the Java programming course in our department have prepared a considerable sum of assignments in JPLAS which ask students to write source codes with standard input/output functions. Besides, a variety of Java programming textbooks as well offer such assignments in the sections for novice students. Hence, it is assumed that the source code for the proposed test code generation tool contains standard input/output functions since JPLAS has been designed for novice students. In this paper, we limit the number of standard input and output functions into one, because our proposal targets source codes in programming assignments for novice students. It is noted that this tool can be generalized to handle multiple standard input and output functions

B. Requirements in Test Code

Subsequently, it is essential to solve the following problems in generating the corresponding test code:

1) The input data from the standard input (keyboard) must be described in the test code to test the standard input function in the source code.
2) The input data must be elaborated in the test code for the standard input function.
3) The input data in the test code must cover any possible one for the standard input function.
4) The expected output data for each input data must be narrated in the test code correctly.
5) The output data to the standard output (console) must be received by the test code to test the standard output function in the source code.

C. Adopted Functions

The proposed test code generation tool adopts the following functions and commands to solve the mentioned problems by referring the test code implementation in [8]:

- To describe the input data from the standard input in the test code, the `Inputln` method in `StandardInputSnatcher` class is adopted. It is noted that `StandardInputSnatcher` class is extended from `InputStream` class.
- Any possible input data from a console is prepared by a teacher beforehand. It is used in the argument of `Inputln`.
- To obtain the expected output data from the source code for each input data, the test code runs on *JUnit* without giving the expected value to the `assertThat` method. Then, *JUnit* returns the output data from the source code in the error message as the test fails, which is the specification of *JUnit*.
- To read the output data from the answer source code by a student to the standard output, the `readLine` method in `StandardOutputSnatcher` class is adopted. It is noted that `StandardOutputSnatcher` class is extended from `PrintStream` class.

V. Proposal of Test Code Generation Tool

In this section, we propose the test code generation tool for JPLAS using the functions in Section IV-C.

A. Scope of Source Code for Tool

The proposed tool can handle the source code that satisfies the following four conditions:

1) it has only the `main` method.
2) it contains one standard input function.
3) it contains one standard output function for the typical case.
4) it contains one standard output function for handling the exception.

It is discovered that a code containing multiple standard input/output functions can be handled by increasing the number of `Inputln` or `assertThat` in the test code by their numbers.

An example source code in this scope is as follows:

source code 2

```
1: import java.io.*;
2: public class Sample {
3:   public static void main(String[] args) {
4:     BufferedReader reader =
     new BufferedReader
     (new InputStreamReader(System.in));
5:     try {
6:     System.out.println
         ("Please input one integer:");
7:       String input = reader.readLine();
8:       int numberLines =
          Integer.parseInt(input);
9:       System.out.println
          (numberOfLines+ " is input.");
10:     } catch(Exception e) {
11:     System.out.println(e);
12:    }
13:  }
14: }
```

source 2 has 1) only the `main` method at line 3, 2) one `readLine()` at line 7, 3) one `System.out.println` at line 9, and 4) one `System.out.println` at line 11 to handle `Exception` at line 10.

B. Tool Usage Procedure

The usage procedure of the proposed tool is as follows:

1) A teacher prepares the list of possible input data to the source code, which is called the *input list* for this tool.
2) A *test code template* is provided in the tool, which can be used for any source code in the scope of this paper.
3) The tool reads one input data from the *input list*, embeds it for the argument of `Inputln` in the template, and runs *JUnit* with this template.
4) The tool reads the output (error message) of *JUnit*, and embeds it as the expected value in `assertThat` in the template.
5) If the *input list* has unused input data, the tool copies the corresponding lines in the template to the data input by `Inputln`, the execution of the source code, the store of the output data in the buffer, and the comparison by `assertThat`, and go to 3).

C. Inputs to Tool

To use the tool, a teacher needs to prepare a *reference source code* for an assignment in the scope, the *input list*, and the *test code template*. In this paper, we adopt the following five data types and concrete values for the standard input, and generate the test code that can verify the source code when each data in the input list is given in the code.

- one-byte positive integer: 15
- one-byte negative integer: -54
- one-byte positive zero: 0
- one-byte character: “abc”
- two-byte character: “ＡＢＣ”

D. Test Code Template

This tool uses the given test code template. The following code describes the core part of the test code template starting from `@Test`. In advance, several `import` statements to use related libraries, and the instance generations for the `StandardInputSnatcher` and `StandardOutputSnatcher` classes are necessary as in [8]. Besides, the definitions of these classes are also required to complete the test code template.

In this template, `in.Inputln` at line 5 gives the standard input to the source code, where `in` is an instance for `StandardInputSnatcher`. `expected` at line 13 represents the expected output data of the source code. In the template, they are blank, so that they should be filled at 2) and 5) in the usage flow in Section V-B. The standard output data from the source code is read at lines 8-12 for the given input data at line 5. `assertThat` at line 14 compares the expected data with the output data of the code. The whole lines are repeated for each input data in the *input list*.

test code template

```
1: @Test
2: public void test() throws Exception {
3:   StringBuffer bf = new StringBuffer();
4:   String actual, line, expected;
5:   in.Inputln("");
6:   ClassName.main(new String[0]);
7:   System.out.flush("");
8:   while((line = out.readLine())!= null) {
9:      if (bf.length() > 0) bf.append("\n");
10:    bf.append(line);
11:  }
12:  actual = bf.toString();
13:  expected = "";
14:  assertThat(actual, is(expected));
15: }
```

E. Test Code Generation Example

This subsection introduces an example of the test code generation by applying the proposed tool to **source code 2**. The file name for the generated test code is given as *SampleTest.java*.

1) The *input list* {15, -54, 0, ”abc”, ”ＡＢＣ”} is used.
2) The first input data 15 is embedded as the input data to the source code in the template at line 5 to extract the expected output of the source code for 15.
3) `ClassName` at line 6 is changed into `Sample` that is the class name in **source code 2**.
4) By running *JUnit* with this template, the following error message is obtained:
   Expected = ””;
   but was ”15 is input.”
5) The message “15 is input.” is extracted as the expected output data from the input 15.
6) This expected output data are embedded in the corresponding part at line 13.
7) By selecting the next input data in the *input list* in 1) sequentially, the same procedure in 2)-7) is repeated until the end of the list.

Next, **test code 2** shows a part of the generated test code.

test code 2

```
1: @Test
2: public void test() throws Exception {
3:   StringBuffer bf = new StringBuffer();
4:   String actual, line, expected;
5:   in.Inputln("15");
6:   Sample.main(new String[0]);
7:   System.out.flush("");
8:   while((line = out.readLine())!= null) {
9:      if (bf.length() > 0) bf.append("\n");
10:    bf.append(line);
11:  }
12:  actual = bf.toString();
13:  expected = "15 is input.";
14:  assertThat(actual, is(expected));
15: }
```

VI. Evaluation

In this section, we evaluate the proposed tool by applying it to 97 source codes that are collected from Java programming textbooks or Web sites [9]-[13] and examining the correctness of the generated test codes in testing the original source codes. Here, we changed some problems in [12] to using standard inputs/outputs through the console instead of using dialog boxes.

A. Results

The tool generates the correct test codes for 96 source codes that can pass their original ones correctly. Thus, we confirm the effectiveness of the proposed test code generation tool for code writing problems asking standard input/output functions in JPLAS. The following **source code 3** shows an example source code such that the tool successfully generates the test code.

source code 3

```
1: import java.io.*;
2: public class Sample2 {
3:   public static void main(String[] args) {
4:     System.out.println
      ("Please input word.");
5:     BufferedReader reader =
        new BufferedRreader(
        new InputStreamReader(System.in));
6:     try {
7:      String input = reader.readLine();
8:      System.out.println(input+"was input.");
9:     } catch(Exception e) {
10:       System.out.println(e);
11:    }
12:  }
13: }
```

The core part of the corresponding test code is revealed as follows:

test code 3

```
1: @Test
2: public void test() throws Exception {
3:   StringBuffer bf = new StringBuffer();
4:   String actual, line, expected;
5:   in.Inputln("15");
6:   Sample2.main(new String[0]);
7:   System.out.flush("");
8:   while((line = out.readLine())!= null) {
9:      if (bf.length() > 0) bf.append("\n");
10:    bf.append(line);
11:  }
12:  actual = bf.toString();
13:  expected = "Please input word. \n
       15 was input.";
14:  assertThat(actual, is(expected));.
15: }
```

B. Failed Code

In the failed source code in **source code 4**, a random number is generated at line 3 and is output on the console at line 3. Thus, the expected output data at testing the source code is different from the one at generating the test code. It is necessary to use the same *seed* between them to control the random number generator used in the code. Actually, the use of the certain random number generator should be described in the specification as one of the internal behaviors of the code in the assignment. The control of the seed should be also included there.

source code 4

```
1: import java.io.*;
2: public static void main(String[] args) {
3:   int number = (int) (Math.random());
4:   System.out.println(number);
5: }
```

VII. Conclusion

In this paper, we proposed a *test code generation tool* from a source code containing standard inputs/output functions for code writing problems in JPLAS. In the evaluation, the effectiveness is verified through applying the tool to 97 source codes in Java programming text books or Web sites. In future works, we will extend the proposed tool to consider random generators in a source code, other input/output functions than the standard ones, other methods than the `main` method, and improve the readability of the generated test code to make it easier for novice students.

Acknowledgments

This work is partially supported by JSPS KAKENHI (16K00127).

References

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

[2] N. Funabiki, Y. Matsushima, T. Nakanishi, 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, Feb. 2013.

[3] 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,” Inf. Eng. Express, vol. 1, no. 3, pp. 9-18, Sep. 2015.

[4] 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,” Inf. Eng. Express, vol. 1, no. 3, pp. 19-28, Sep. 2015.

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

[6] JUnit, http://www.junit.org/.

[7] CodePress, http://codepress.sourceforge.net/.

[8] Diary of kencoba, http://d.hatena.ne.jp/kencoba/20120831/1346398388.

[9] M. Takahashi, Easy Java, 5th Ed., Soft Bank Creative, 2013.

[10] H. Yuuki, Java programming lessen, 3rd Ed., Soft Bank Creative, 2012.

[11] Y. D. Liang, Introduction to Java programming, 9th Ed., Pearson Education, 2014.
[12] Java programming seminar, http://java.it-manual.com/start/about.html.
[13] Kita Soft Koubo, http://kitako.tokyo/lib/JavaExercise.aspx.