ASTES

Advances in Science, Technology and Engineering Systems Journal
Vol. 3, No. 1, 30-37 (2018)
www.astesj.com
Special issue on Advancement in Engineering Technology

ASTES Journal
ISSN: 2415-6698

A Test Code Generation Method for Coding Standard Input/Output with Exception Handling in Java Programming Learning Assistant System

Ei Ei Mon[1], Nobuo Funabiki[*,1], Ryota Kusaka[1], Khin Khin Zaw[1], Wen-Chung Kao[2]

[1]*Okayama University, Department of Electrical and Communication Engineering, Okayama, Japan*

[2]*National Taiwan Normal University, Department of Electrical Engineering, Taipei, Taiwan*

ARTICLEINFO

Article history:
Received: 30 October, 2017
Accepted: 13 December, 2017
Online: 30 January, 2018

Keywords:
Java programming
JPLAS
Test code
Test case
Automatic generation
JUnit

ABSTRACT

To advance Java programming educations, we have developed the Java Programming Learning Assistant System (JPLAS) that provides the code writing problem. This problem asks a student to write a source code to satisfy the specification of a given assignment, where the correctness is verified by running test code on JUnit. For a novice student, a code of implementing the standard input/output with the exception handling should be mastered at the early stage as the first step programming for human interfaces. However, for a teacher, it is not easy to write the test code for it. In this paper, we propose a test code generation method to generate the test code using the reference source code for the assignment. In the evaluation of this proposal, all the students completed the codes using the generated test codes for exception handling, although the use of exception handling functions was sometimes insufficient or incorrect.

1 Introduction

Recently, the objected oriented programming language *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 [2]. Therefore, there have been strong demands 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 advance Java programming educations, we have developed the Web-based *Java Programming Learning Assistant System (JPLAS)* [3]-[7]. JPLAS inspires students by offering sophisticated learning environments via quick responses to their answers for self-studies. At the same time, it supports teachers by reducing loads of evaluating codes. JPLAS has several types of problems to cover a variety of students at different learning levels. Among them, the *code writing problem* [4] asks a student to write a source code to satisfy the specification of a given assignment.

The *code writing problem* is implemented based on the *test-driven development (TDD) method* [8], using an open source framework *JUnit* [9]. *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. It is noted that the reference source code is essential to verify the correctness of the problem statement and the test code. 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*.

However, teachers at schools 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

[*]Nobuo Funabiki, Department of Electrical and Communication Engineering, Okayama University, Okayama, Japan, funabiki@okayama-u.ac.jp
This paper is an extension of work originally presented in 31st IEEE International Conference on Advanced Information Networking and Applications (AINA-2017) [1]

https://dx.doi.org/10.25046/aj030105

register an incomplete test code that does not verify some 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 the standard input/output with exception handling in a source code. The code of implementing the standard input/output with exception handling should be mastered by novice students at the early stage of Java programming educations as the first step programming for human interfaces.

In this paper, we propose a *test code generation method* for the code writing problem in JPLAS that generates a test code using a reference source code to test the standard input/output with exception handling. This method can generate a test code through the following steps: 1) a *test code template* is provided by our proposal, 2) a set of standard inputs to be tested are made by a teacher, 3) by running the reference code with each input, the corresponding expected standard output is extracted correctly, and 4) this pair of the standard input and the standard output are embedded into the test code template. By repeating steps 3) and 4) for every test standard input, the test code can be completed. To run the source code using the test code on *JUnit*, it introduces the classes to handle the standard input/output functions as the memory access functions in [10].

To evaluate the proposed method, first, we applied it to 97 source codes in Java programming textbooks or Web sites that contain the standard input/output. It has been proved that the generated test codes could correctly verify the source codes except for one code using a random generator. Then, we generated the test codes for three problems and asked five students who are currently studying Java programming to write the source codes using them. It was found that they completed the codes that can pass the test codes, whereas the use of exception handling functions was sometimes insufficient or incorrect.

The rest of this paper is organized as follows: Sections 2 and 3 introduce the TDD method and JPLAS respectively. Section 4 presents the test code generation method. Section 5 shows the evaluation result. Sections 6 shows related works. Finally, Section 7 concludes this paper with some future works.

2 Test-driven Development Method

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

2.1 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, and
3) to repeat modifications of the source code until it passes each test using the test code.

2.2 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.

2.3 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      assertThat(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 test code performs the following functions:

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.

2.4 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.

3 Java Programming Learning Assistant System

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

3.1 Server Platform

JPLAS is implemented as a Web application using *JSP/-Java*. 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. For the browser, it assumes the use of *Firefox* with *HTML*, *CSS*, and *JavaScript*.

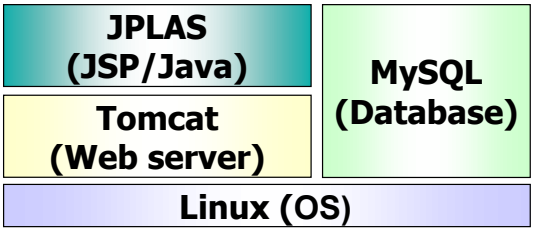

Figure 1: JPLAS server platform.

3.2 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 2.3, 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.

3.3 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* [11] 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.

4 Proposal of Test Code Generation Method

In this section, we propose the test code generation method for coding the standard input/output with exception handling.

4.1 Scope of Source Code under Test

At the early stage of the Java programming education, the responsibility of a student is to master how to write a source code that contains the standard input/output with exception handling. Thus, a teacher in a Java programming course should prepare a considerable number of assignments for writing source codes containing them, where many Java programming textbooks offer such assignments for novice students.

The source code in this paper must contain the functions for the standard input/output and the exception handling. Then, if the proper data is given to the code from the standard input, it must handle it correctly and outputs the message specified in the assignment to the standard output. On the other hand, if the improper data is given, it must handle it using the exception handling command without abortion and outputs the corresponding message.

4.2 Requirements in Test Code

Subsequently, the test code must satisfy the following requirements:

1. The input data from the standard input (keyboard) must be described in the test code to test the standard input in the source code.
2. The output data to the standard output (console) must be received by the test code to test the standard output in the source code.

3. The input data must be elaborated in the test code for the standard input.

4. The input data in the test code should cover any possible one for the standard input, including the proper and improper ones.

5. The expected output data for each input data must be narrated in the test code correctly.

4.3 Solutions for Requirements

Test code generation method adopts following functions and commands to solve the above mentioned requirements by referring the test code implementation in [10]:

- To describe the standard input data to the source code, the *Inputln* method in *StandardInputSnatcher* class is adopted in the test code. It is noted that *StandardInputSnatcher* class is extended from *InputStream* class.

- To receive the standard output data from the source code, the *readLine* method in *StandardOutputSnatcher* class is adopted in the test code. It is noted that *StandardOutputSnatcher* class is extended from *PrintStream* class.

- Any possible standard input data is prepared by a teacher beforehand. It is used in the argument of *Inputln*.

- To obtain the expected standard output data from the code for each input data, the reference source code is executed with this input data.

- Each pair of the standard input and output data is embedded into the test code.

4.4 Conditions of Source Code

Currently, to avoid the complexity, the proposed method confines the applicable source code that satisfies the following conditions:

1. it has the *main* method only.

2. it contains the standard input function.

3. it contains the standard output function for handling the proper input.

4. it contains the standard output function for handling the exception.

It is noted that a source code containing multiple standard input/output functions can be handled by increasing the number of *Inputln* or *assertThat* in the test code accordingly. Besides, if a code does not have the *main* method, it can be handled by describing the proper statements to execute the method for the standard input/output in the test code.

An example source code in this scope is as follows:

source code 2

```
1  import java.util.Scanner;
2  public class Sample {
3    public static void main(String args[]){
4      int number;
5      Scanner scan = new Scanner(System.in);
6      try{
7        System.out.print("Enter an integer");
8        String actual = scan.nextLine();
9        number = Integer.parseInt(actual);
10       System.out.println(number +": is input
         number");
11     } catch(NumberFormatException e) {
12       System.out.print("
         NumberFormatException occurs!");
13     }
14   }
15 }
```

source 2 accepts an integer data from a console and outputs a message with this data on a display. In this source code, 1) it has only the *main* method at line 3, 2) *scan* object of *Scanner* class is defined at line 5 as the standard input function, 3) *System.out.println* is called at line 10 as the standard output function for handling the proper input, and 4) *System.out.println* is called at line 12 as the standard output function for handling the exception.

4.5 Test Code Template

Then, the proposed method provides the *test code template* containing the required functions for the above mentioned source code. 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. Besides, the definitions of these classes are also required to complete the test code template.

In this template, *in.Inputln* at line 29 gives the standard input data to the source code, where *in* is an instance of *StandardInputSnatcher* class. The statements at lines 30-37 run the source code and read the standard output data for this input data, where *out* is an instance of *StandardOutputSnatcher* class. *expected* at line 38 represents the expected output data of the source code. The blanks ″ ″ at lines 29 and 38 should be filled by the standard input and output data. *assertThat* at line 39 compares the expected data with the output data of the code. The whole statements at lines 25-40 should be prepared for each input data.

test code template

```
1  import static org.hamcrest.CoreMatchers.is;
2  import static org.junit.Assert.assertThat;
3  import static org.junit.Assert.*;
4  import java.io.InputStream;
5  import org.junit.Before;
6  import org.junit.Test;
7  import Snatcher.StandardOutputSnatcher;
8  import java.io.BufferedReader;
9  import java.io.ByteArrayOutputStream;
10 import java.io.IOException;
11 import java.io.InputStream;
12 import java.io.PrintStream;
13 import java.io.StringReader;
14
15 public class TemplateTest {
16   private StandardInputSnatcher in = new
       StandardInputSnatcher();
```

```
17      private StandardOutputSnatcher out = new
            StandardOutputSnatcher();
18
19      @Before
20      public void setUp() {
21        System.setIn(in);
22        System.setOut(out);
23      }
24
25      @Test
26      public void test1() throws Exception {
27        StringBuffer bf = new StringBuffer();
28        String actual,line,expected;
29        in.Inputln(""); // standard input
30        Sample.main(new String[0]);
31        System.out.flush();
32        while((line = out.readLine()) != null) {
33          if (bf.length() > 0)
34          bf.append("\n");
35          bf.append(line);
36        }
37        actual = bf.toString();
38        expected = ""; // expected standard output
39        assertThat(actual,is(expected));
40      }
41    }
```

4.6 Test Code Generation Procedure

The test code generation procedure using the *test code template* in the proposed method is as follows:

1) A teacher prepares the reference source code for the assignment.

2) He/she prepares a set of possible standard input data to the source code.

3) He/she runs the source code by using each standard input data and observes the corresponding standard output data.

4) He/she embeds the standard input data into ″ ″ at line 29 and the observed standard output data into ″ ″ at line 38 in the test code template.

As the possible standard input data in step 2), the following five data types should be considered. Then, the teacher needs to select one value for each data type, which is used in step 3).

- positive integer: 5
- negative integer: -14
- zero integer: 0
- floating-point number: 0.5
- one-byte character: abc
- two-byte character: A B C

4.7 Generated Test Code Example

This subsection introduces an example of the test code generated by applying the proposed method to **source code 2**. The file name for the generated test code is given as *SampleTest.java*. The following **test code 2** shows a part of the test code.

test code2

```
1   .....
2
3     @Test
4     public void test1() throws Exception {
5       StringBuffer bf = new StringBuffer();
6       String actual,line,expected;
7       in.Inputln("5"); // proper standard input data
8       Sample.main(new String[0]);
9       System.out.flush();
10      while((line = out.readLine()) != null) {
11        if (bf.length() > 0)
12        bf.append("\n");
13        bf.append(line);
14      }
15      actual = bf.toString();
16      expected = "Enter an integer" +
17       "5: is input number";
18      assertThat(actual,is(expected));
19    }
20
21    @Test
22    public void test2() throws Exception {
23      StringBuffer bf = new StringBuffer();
24      String actual,line,expected;
25      in.Inputln("abc"); // improper standard input
          data
26      Sample.main(new String[0]);
27      System.out.flush();
28      while((line = out.readLine()) != null) {
29        if (bf.length() > 0)
30        bf.append("\n");
31        bf.append(line);
32      }
33      actual = bf.toString();
34      expected = "Enter an integer" + "
          NumberFormatException occurs!";
35      assertThat(actual,is(expected));
36    }
37  .....
```

5 Evaluation

In this section, we evaluate the effectiveness of the proposed *test code generation method* in terms of generating test codes from existing source codes and writing source codes using the test codes by students.

5.1 Test Code Generation Results

First, we evaluate the method in generating test codes from source codes. For this purpose, 97 source codes were collected from Java programming textbooks or Web sites [12]-[16], and the test codes were generated by applying the proposed method. It is noted that some codes in [15] were modified to using the standard input/output through the console instead of using the dialog box. Then, the correctness of each test code was examined by testing the original source code. It was found that our method generated the test codes that can pass original codes correctly except for one source code, which outputs a random number generated in the code. Thus, the effectiveness of the proposed method was confirmed.

The following **source code 3** shows an example source code in [12] where the method successfully generates the test code shown in **test code 3**. It is noted that *try - catch* is used here instead of *throws* in the original source code.

source code 3

```
1  import java.io.*;
2  class Sample3 {
3    public static void main(String[] args) throws
       IOException {
4      try {
5        System.out.println("Enter two integers
         ");
6        BufferedReader br =
7          new BufferedReader(new
         InputStreamReader(System.in));
8        String str1 = br.readLine();
9        String str2 = br.readLine();
10       int num1 = Integer.parseInt(str1);
11       int num2 = Integer.parseInt(str2);
12       System.out.println("The sum is " + (
         num1+num2) + ".");
13     } catch(NumberFormatException e) {
14       System.out.print("
         NumberFormatException occurs!");
15     }
16   }
17 }
```

test code3

```
1  .....
2
3    @Test
4    public void test1() throws Exception {
5      StringBuffer bf = new StringBuffer();
6      String actual,line,expected;
7      in.Inputln("2"); in.Inputln("7");// proper
         standard input data
8      Sample.main(new String[0]);
9      System.out.flush();
10     while((line = out.readLine()) != null) {
11       if (bf.length() > 0)
12       bf.append("\n");
13       bf.append(line);
14     }
15     actual = bf.toString();
16     expected = "Enter two integers" +"The
         sum is 9.";
17     assertThat(actual,is(expected));
18   }
19
20   @Test
21   public void test2() throws Exception {
22     StringBuffer bf = new StringBuffer();
23     String actual,line,expected;
24     in.Inputln("0.5"); in.Inputln("-3");//
         improper standard input data
25     Sample.main(new String[0]);
26     System.out.flush();
27     while((line = out.readLine()) != null) {
28       if (bf.length() > 0)
29       bf.append("\n");
30       bf.append(line);
31     }
32     actual = bf.toString();
33     expected = "Enter two integers" + "
         NumberFormatException occurs!";
34     assertThat(actual,is(expected));
35   }
36 .....
```

5.2 Source Code Writing Results

Next, we evaluate the proposed method in writing source codes with generated test codes by five students who are currently studying Java programming and have same technical levels. For this purpose, we prepared the following three problems, where all the students completed the source codes that pass the test codes for any problem.

5.2.1 Problem #1

In problem #1, the code accepts an integer data from a console, and outputs a message with this data to a console, where **source 2** is the reference source code and **test 2** is the test code. The source code from a student is expected to use *NumberFormatException* to check the input data format. Then, three students use this class for the exception handling, and one uses *Exception*. However, one student does not use it where he implements the data format checking function.

5.2.2 Problem #2

In problem #2, the code accepts an integer index from a console, and outputs the indexed data from the data array. The student code is expected to use *ArrayIndexOutofBoundException* to check the range of the index. Then, only one student uses this class. The other students implement the index checking function in the codes. Two students use *IOException*, and two students do not use any class for the exception handling. No student use *NumberFormatException* to check the input data format, although the class was requested in problem #1. Unfortunately, many students cannot integrate the knowledge that has been studied sequentially.

5.2.3 Problem #3

In problem #3, the code accepts a file path from a console, and outputs the string at the first line in the file. The student code is expected to use *FileNotFoundException* or *IOException* to check the file path. Then, three students use *FileNotFoundException*, one uses *Exception*, and one uses *IOException*.

5.2.4 Summary of Student Applications

This simple experiment of our proposal shows that the students can generally complete source codes using standard input/output with exception handling that can pass the generated test codes. However, their use of the class for the exception handling is sometimes insufficient or incorrect. It has been observed that these students are not experts, which causes the difference in their source codes, although they have enough programming skills. To let them understand the correct use, it is necessary to improve the proposed method.

6 Related Works

In this section, we introduce some related works to this paper.

In [17], Fu presented a static exception-flow analysis that computes chains of semantically-related exception-flow links and reports entire exception propagation paths. These chains can be used, 1) to show the error handling architecture of a system, 2) to assess the vulnerability of a single component and the whole system, 3) to support the better testing of an error recovery code, and 4) to facilitate the tracing of the root cause of a logged problem.

In [18], Rashkovits showed that most of college students understand the concept of Java exception handling at the basic level, and the majority of them have difficulty in understanding advanced properties such as use of multiple exceptions, flow of control in the context of exceptions, handling exceptions further up the calling chain, catching and handling hierarchically related exceptions, and overriding methods that throw exceptions. They also provided a tutorial of exception handling, and quoted that exception handling is perceived as a relatively difficult task by novice programmers. In future works, we will consider to adopt their contributions.

In [19], Júnior presented a practical approach to preserve the *exception policy* in a system by automatically checking *exception handling design rules*. They are checked through executions of *JUnit* test cases with *dynamic mock objects* that are generated by the supporting tool. Four versions of *Mobile Media in SPL* were used to evaluate whether the policy was preserved or not. The results show that the approach can effectively detect violations on the policy of software product lines.

In [20], Nakshatri presented an empirical study of exception handling patterns in Java projects. It forces developers to think in sophisticated ways to handle the exceptions. In this study, empirical data was extracted from projects by analyzing data in *GitHub* and *SourceForge* repositories. The results were compared with recommendations for best practices in exception handling presented by Bloch [21]. It has been observed that most programmers ignore checking exceptions, and higher classes in the exception class hierarchy are more frequently used.

In [22], Brunet presented the concept of *design test* that automatically checks whether the code conforms to the specific *design rule* by using a test-like program. To support it, *DesignWizard (DW)* had been developed with a fully-fledged API that allows writing design tests for Java codes using *JUnit*. The proposal was applied to three software products in their group and student projects in the undergraduate course. The results showed that this approach was suitable to check conformance between the design rules and the code implementation. Moreover, it has been observed that both designers and programmers appreciated the design tests as executable documents that can be easily kept up to date.

In [23], Akahane presented a Web-based *automatic scoring system* for Java programming assignments to reduce loads of teachers in verifying a huge number of codes and in giving feedbacks to students. The system receives Java application programs submitted by students, and immediately returns the results of *JUnit* tests where the *Java Reflect API* is adopted for testing private classes and methods that have been commonly found in introductory courses. The regular expression is used to compare the output texts of each student program and those of the reference program. Through use in an actual course in their university, it was confirmed that this system was very helpful for students to improve programming skills by correcting mistakes in their programs and repeating their submissions.

In [24], Kitaya presented a Web-based scoring system of programming assignments to students, which is similar to JPLAS. Their test consists of compiler check, *JUnit* test, and result test. The result test verifies the correctness of a student code composed of only the *main* method that reads/writes data from/to the standard input/output devices, by comparing the results of this code and of the reference code. However, the method has several disadvantages from our proposal: 1) it is only applicable to a code composed of the *main* method with the standard input/output, 2) it uses other programs to use the redirection for handling the standard input/output, and 3) it needs several input files to check the correctness for different input data. On the other hand, our method is applicable to a code containing other than the *main* method, it needs only *JUnit* with a test code, and all the input data can be described in a single test code.

7 Conclusion

In this paper, we proposed the *test code generation method* for the code writing problem in JPLAS that requires implementing a Java source code containing the *standard input/output* with *exception handling*. To access the standard input/output from the test code on *JUnit*, the *test code template* is first prepared with the *input/output snatcher classes*. Then, the test code is completed by embedding the input and output extracted by running the *reference source code* into the template. This proposal is helpful in reducing the teacher load in writing the test code for the programming assignment that requires the standard input/output with exception handling, which is common for novice students. The effectiveness is evaluated through applying the method to 97 source codes in Java programming text books or Web sites, and asking five students to write source codes using the generated test codes for three problems. In future works, we will extend the proposed method to handle other input/output functions, other methods than the *main* method, and improve the readability of the generated test code to make it easier for novice students.

References

[1] N. Funabiki, R. Kusaka, N. Ishihara, and W.-C. Kao, "A proposal of test code generation tool for Java programming learning assistant system," Proc. IEEE Int. Conf. Adv. Inform. Netw. Appl., pp. 51-56, March 2017.

[2] S. Cass, The 2015 top ten programming languages, `http://spectrum.ieee.org/computing/software/the-2015-top-ten-programminglanguages/?utm_so.`

[3] 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. Science, vol. 44, no. 2, pp. 247-260, May 2017.

[4] 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. Science, vol.40, no.1, pp. 38-46, Feb. 2013.

[5] 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.

[6] 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.

[7] 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, Sep. 2017.

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

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

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

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

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

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

[14] Y. D. Liang, Introduction to Java programming, 9th Ed., Pearson Education, 2014.

[15] Java programming seminar, http://java.it-manual.com/start/about.html.

[16] Kita Soft Koubo, http://kitako.tokyo/lib/JavaExercise.aspx.

[17] C. Fu and B. G. Ryder, "Exception-chain analysis: revealing exception handling architecture in Java server applications", Proc. Int. Conf. Soft. Eng., pp. 230-239, May 2007.

[18] R. Rashkovits and I. Lavy, "Students' understanding of advanced properties of Java exceptions", J. Inform. Tech. Edu., vol. 11, pp. 327-352, 2012.

[19] R. J. S. Júnior and R. Coelho, "Preserving the exception handling design rules in software product line context: a practical approach", Proc. Latin-American Symp. Depend. Comp. Work., pp. 9-16, 2011.

[20] S. Nakshatri, M. Hegde, and S. Thandra "Analysis of exception handling patterns in Java projects: an empirical study", Proc. IEEE/ACM Work. Conf. Mining Soft. Rep., pp. 500-503, May 2016.

[21] J. Bloch, Effective Java, 2nd Ed., Prentice Hall PTR, Upper Saddle River, NJ, USA, 2008.

[22] J. Brunet, D. Guerrero, and J. Figueredo, "Design tests: an approach to programmatically check your code against design rules", Proc. Int. Conf. Soft. Eng., pp. 255-258, May 2009.

[23] Y. Akahane, H. Kitaya, and U. Inoue, "Design and evaluation of automated Scoring Java programming assignments" Proc. Int. Conf. Soft. Eng., Art. Intel., Net. Para./Dist. Comp., pp.1-6, 2015.

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