An Error Code Highlighting Function in Java Programming Learning Assistant System Using Test-Driven Development Method

Nobuo Funabiki, Yuuki Fukuyama, Yukiko Matsushima, Toru Nakanishi, and Kan Watanabe

***Abstract*—Recently, the objected-oriented programming language *Java* has been used in many practical systems including enterprise servers, smart phones, and embedded systems due to its high reliability and portability. To enhance effects of Java programming educations, we have developed a Web-based *Java Programming Learning Assistant System (JPLAS)*. JPLAS adopts the *Test-Driven Development (TDD) method* for automatic testing of source codes written by students on the server to assist their self-learning. In JPLAS, a teacher needs to register each Java programming assignment with a statement, a reference source code, and a test code. Then, students should write source codes by referring this statement and the test code so that their source codes can be tested automatically at the server by a testing software tool called *JUnit* with the test code. Unfortunately, the current JPLAS is not friendly to students who have difficulty in reading the log message from *JUnit* that contains the information on the errors in the source code. In this paper, we implement the *error code highlighting function* to help such students to find erroneous lines in the code by highlighting them graphically. We evaluate the effectiveness of this function through an experiment for 42 students who are currently taking the Java programming class.**

***Index Terms*—JPLAS, Java programming education, Web system, test-driven development method, error code highlighting.**

I. Introduction

RECENTLY, with penetrations of the *Information and Communication Technology (ICT)* into our societies, adverse affects of system failures caused by software bugs have become intolerable. They sometimes have stopped functions of railways [1], airlines [2], and large banking systems [3]. Along this trend, the software test has become regarded as the last crucial step to avoid productions of software bugs in systems. In this trend, a *Test-Driven Development (TDD) method* has been focused as an effective software developing method that can avoid bugs by writing and testing source codes at the same time [4]. A *test code* is a program code to test the correctness of the outputs of the methods that are implemented in the *source code* under the test.

Java is a useful and practical objected-oriented programming language that has been used in a lot of important practical systems including Web systems, enterprise servers, smart phones, and embedded systems. Thus, a Java programming education in schools has been important in order to foster professional Java programmers into societies.

To enhance effects of the Java programming education by assisting self-studies of students and reducing teaching loads of teachers, we have proposed and developed a Web-based *Java Programming Learning Assistant System (JPLAS)* [5][6]. JPLAS adopts the TDD method for automatic testing of source codes from students. In JPLAS, a teacher first needs to register each Java programming assignment with a statement, a reference source code, and a test code using a Web browser. Then, students should write source codes by referring the statement and the test code in the assignment so that their source codes can be tested automatically on a testing tool called *JUnit* [7]. *JUnit* tests the correctness of the source code using the test code. Every time a source code is submitted to the Web server, it is tested automatically by using *JUnit* at the server.

Unfortunately, the current JPLAS is not sufficiently friendly to students who cannot read the log message from *JUnit* that contains the information on the errors necessary to fix the problems in the source code. Because JPLAS has been developed to encourage such students to learn the Java programming by themselves, the error message in JPLAS should be improved so that these students can easily find the erroneous lines without helps by a teacher.

In this paper, we newly implement the *error code highlighting function* in JPLAS to help such students to find erroneous lines in the source code easily by highlighting them graphically. We evaluate the effectiveness of JPLAS with this function through an experiment to 42 students who are currently taking the Java programming class in our department.

The rest of this paper is organized as follows: Section II introduces the TDD method. Section III reviews our JPLAS. Section IV presents the error code highlighting function. Section V discusses the evaluation result. Section VI concludes this paper with some future works.

II. Test-driven Development Method

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

A. Outline of TDD Method

In the TDD method, the test code should be written before or when the source code is implemented, so that it can verify whether the 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 every required specification,
(2) to write the source code, and
(3) to repeat modifications of the source code until it passes every test using the test code.

The authors are with the Department of Electrical and Communication Engineering, Okayama University, 3-1-1 Tsushimanaka, Okayama, 700-8530, Japan e-mail: funabiki@cne.okayama-u.ac.jp.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

B. 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 or a *class*. Because *JUnit* has been designed with the Java-user friendly style, its use including the test code programming is basically easy for Java programmers. In *JUnit*, a test is performed by using a given method whose name starts from "assert". This paper adopts the "assertEquals" 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 *MyMath* class source code, we explain how to write a test code. *MyMath* class returns the summation of two integer arguments.

```
1: public class Math{
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 MathTest {
4:     @Test
5:     public void testPlus(){
6:         Math ma = new Math();
7:         int result = ma.plus(1, 4);
8:         asserEquals(5, 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, and
(3) to compare the result with its expected value for the arguments in (2) using the *assertEquals* method.

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 any function to be 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 easy, 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. Beside, students should experience the software test that has become important in software companies.

III. Java Programming Learning Assistant System (JPLAS)

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

A. Server Platform

JPLAS is implemented as a Web application using *JSP/Servelt*. 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.

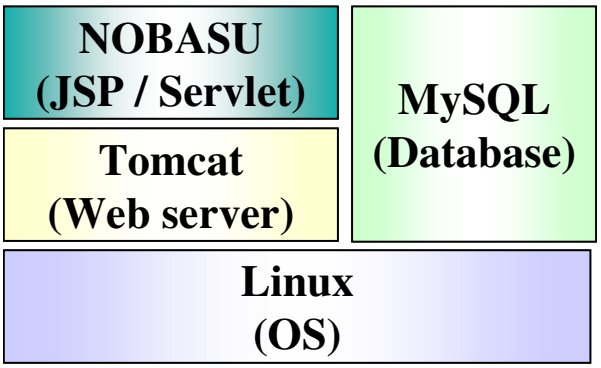

Fig. 1. Server platform.

B. Teacher Service Functions

JPLAS has user functions both for teachers and students. *User functions for teachers* include the registration of new classes, the registration and management of assignments, and the verification of source codes that are submitted from students. To register a new assignment, a teacher needs to input an assignment title, a problem statement, a model (or reference) source code, and a test code. After the registration, they are disclosed to the students except for the model 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, JPLAS can automatically generate a template for the test code from the model source code. Then, a teacher only needs to specify concrete numbers for the arguments in each test method to complete the test code.

To evaluate the difficulty of assignments and the comprehension of students, it allows a teacher to view the number of submissions for code testing by each student. If a teacher finds that many submissions have been tried by many students for an assignment, it can be considered very difficult for them, and should be changed to an easier one. If a teacher finds a student who submitted source codes in many times whereas other students did in fewer times, this student should be cared specifically.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

C. Student Service Functions

User functions to students 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. Actually, the student 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* [8] so that a student can write codes on a Web browser. Any submitted source code is stored in the database at the server so that they can view old ones.

D. Example

Figures 2 and 3 show the test code and a source code from a student for the *ElGamal* encryption programming assignment. Figure 4 shows the test result, where one error is detected because the last argument of the "encrypt" method in the source code is different from the specification given in the test code.

IV. Error Code Highlighting Function

In this section, we present a newly implemented *error code highlighting function* in JPLAS to help students who have difficulty in reading the log message from *JUnit*.

A. Outline

JPLAS provides the output log from *JUnit* to students to help them to fix the problems in their source codes if there exist. However, this log is usually hard to understand for students who even cannot find the lines that they need to modify in their source codes. To help such students, we newly implement a function of highlighting the lines that contain the error codes found by *JUnit*. This function actually highlights both the lines in the test code that return errors at the test and the corresponding lines in the source code.

1) Highlighting in Test Code: This function highlights the lines containing the test methods such as *assertEquals* that return errors in the test code. In our implementation, these lines are extracted from the log of *JUnit* that contains the erroneous line information. Figure 5 shows an example of a *JUnit* output log, where the line framed by a rectangular indicates that the 10th line in the test code returns the error. We note that the 24th line also returns the error, which is omitted in Figure 5 to save space. Figure 6 shows the corresponding interface to students for the test code highlighting.

2) Highlighting in Source Code: Then, the function highlights the lines in the source code that declare the methods containing error lines found by *JUnit*. These erroneous methods in the source code are extracted from the codes for the methods that return errors in the test code. Figure 7 shows an example of the source code highlighting corresponding to Figure 6.

V. Evaluation

In this section, we evaluate the effectiveness of JPLAS with the newly implemented error code highlighting function through its application to students taking a Java programming class in our department.

A. Assignment and Questionnaire for Evaluation

To evaluate the error code highlighting function in JPLAS, we prepared one Java programming assignment in JPLAS to 42 sophomore students in our department who are currently taking a Java programming class. This assignment requests them to write a simple Java program of calculating the area of a circle and a rectangle. Actually, we gave them a source code for this assignment that has one error intentionally so that they can fix it for answers, where we asked them to answer this assignment within $15min.$. Here, we note that these students have used JPLAS to know how to use it. Then, we asked them to reply to the six questions in Table I with five grades.

TABLE I
Questions for questionnaire.

	Question
Q1	Do you think to use the system is easy ?
Q2	Do you think this system is helpful in checking the program functions ?
Q3	Do you think to read the test code is helpful in understanding the assignment specification ?
Q4	Do you feel the response time after the source code submission is long ?
Q5	Do you think the error code highlighting function is helpful in fixing the errors in the code ?
Q6	Do you think this system is helpful in understanding Java programming ?

B. Evaluation Results

Table II shows the replies from the students for the six questions in Table I.

TABLE II
Questionnaire results with five grades.

		1	2	3	4	5	
Q1	hard	1	14	13	8	6	easy
Q2	useless	0	9	17	9	7	useful
Q3	useless	0	9	19	10	4	useful
Q4	long	2	5	9	14	12	short
Q5	useless	0	2	22	9	9	useful
Q6	useless	3	5	22	6	6	useful

For Q1, a similar number of students replied positively (1 or 2) or negatively (4 or 5). It indicates that the usability of the system is basically acceptable for them. Some students felt that the text form in a Web browser is not convenient as a programming editor if it is compared with an editor on a personal computer. Thus, a more advanced editor on a Web browser should be introduced into JPLAS.

For Q2 and Q3, most students replied 3 or more, whereas some students replied 2. It indicates that JPLAS is generally helpful in writing a correct Java code. Because the time for using JPLAS in this experiment was too short for them, some of them might feel negatively. Thus, we should let them use JPLAS for a longer time at next experiments. The result for Q6 also indicates the insufficiency of the use of this system.

For Q5, most students replied 3 or more, which indicates that the newly implemented error code highlighting function is useful in finding errors in source codes to fix them. Thus, we expect that with the help of this function, students can learn the Java programming more willingly by using JPLAS.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

```
1  import junit.framework.*;
2
3  public class EncryptionTest extends TestCase{
4          public void testEncrypt(){
5                  Encryption tmp = new Encryption();
6
7                  String[] ciphertext = new String[2];
8                  ciphertext[0] = "8336436347439611471150025544156611861856194156065
9                  ciphertext[1] = "4294111618369885643725059759348432594977520614475
10                 String[] answer = new String[2];
11                 answer = tmp.encrypt("10753419069985177098668631553298165241184585
12 Five string arguments "2866444882128996767048467429129036294605435677825
13                                 "8172276160992595031217378031951847886365097356235
14 for "encrypt" method "7523026006038088529786911260744316104463319722165
15                                 "20091225");
16
17                 assertEquals( ciphertext[0], answer[0]);
18                 assertEquals( ciphertext[1], answer[1]);
19         }
20
21 }
```

Fig. 2. Test code from teacher.

```
1  import java.math.BigInteger;
2  public class Encryption{
3          String[] encrypt(String p, String g, String r, String Y, String M,
4  String[] ciphertext){
5                  BigInteger P = new BigInteger(p);
6                  BigInteger G = new BigInteger(g);
7                  BigInteger R = new BigInteger(r);
8                  BigInteger y = new BigInteger(Y);
9                  BigInteger m = new BigInteger(M);
10                 BigInteger C1 = new BigInteger("0");
11                 BigInteger C2 = new BigInteger("0");
12                 // エルガマルの計算                    ElGamal encryption
13                 // C1 = MY^r mod p, C2 = g^r mod p
14                 C1 = m.multiply(y.modPow(R, P));
15                 C2 = G.modPow(R, P);
16                 ciphertext[0] = C1.toString();
17                 ciphertext[1] = C2.toString();
18                 return ciphertext;
19         }
20 }
```

Fig. 3. Source code from student.

Encryption.javaコンパイル結果 **Compile result**

```
コンパイル成功
                Compile success
```

Encryptionクラス実行結果 **Execution result**

```
mainメソッドはありません
                No main method for Encryption class
```

テスト結果 **Test result**

```
.E                              One error detected
Time: 0.004
There was 1 error:
1) testEncrypt(EncryptionTest)java.lang.NoSuchMethodError:
Encryption.encrypt(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava
/lang/String;Ljava/lang/String;)[Ljava/lang/String;
        at EncryptionTest.testEncrypt(EncryptionTest.java:11)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
```

Fig. 4. Test result for source code.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

```
JUnit version 4.8.2
.E..E
Time: 0.006
There were 2 failures:
1) testCalCircle(TestMyArea)
java.lang.AssertionError: expected:<113.04> but was:<24.84>
        at org.junit.Assert.fail(Assert.java:91)
        at org.junit.Assert.failNotEquals(Assert.java:645)
        at org.junit.Assert.assertEquals(Assert.java:441)
        at org.junit.Assert.assertEquals(Assert.java:510)
        at TestMyArea.testCalCircle(TestMyArea.java:10)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
            .
            .
            .
```

Fig. 5. JUnit log example.

```
1  import static org.junit.Assert.*;
2  import org.junit.Test;
3
4  public class TestMyArea {
5
6          @Test
7          public void testCalCircle(){
8
9                  MyArea tmp = new MyArea();
10                 assertEquals( 36*Math.PI,tmp.calCircle(6), 0.000000001 );
11         }
12
13         @Test
14         public void testCalSquare(){
15
16                 MyArea tmp = new MyArea();
17                 assertEquals( 25,tmp.calSquare(5) );
18         }
19
20         @Test
21         public void testCalArea(){
22
23                 MyArea tmp = new MyArea();
24                 assertEquals( 36+(36*Math.PI),tmp.calArea(6),0.00000001 );
25         }
26 }
```

Fig. 6. Test code highlighting example.

```
1  public class MyArea {
2          public double calCircle(int r){
3                  return r + r * 3.14;
4          }
5
6          public int calSquare(int length){
7                  return length * length;
8          }
9
10         public double calArea(int fig){
11                 return this.calCircle(fig) + this.calSquare(fig);
12         }
13
14 }
```

Fig. 7. Source code highlighting example.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

VI. Conclusion

This paper presented the implementation of the *error code highlighting function* in the Web-based Java programming learning assistance system *JPLAS* to help students to find erroneous lines in their source codes by highlighting them graphically. The evaluation in our experiment supports the effectiveness of this function. In future studies, we will improve the user interface by adopting an advanced programming editor for a Web browser, and keep evaluations of the effectiveness of JPLAS by applying it for a longer time with various programming assignments to students.

References

[1] The Free Library, "JR East's automated gate system fails to read data from Suica cards," http://www.thefreelibrary.com/LEAD%3A+JR+East%27s+automated+gate+system+fails+to+read+data+from+Suica...-a0155493841.

[2] VentureData.org, "Japan's All Nippon Airways computer system failure was canceled hundreds of flights," http://www.venturedata.org/?i434_Japans-All-Nippon-Airways-computer-system-failure-was-canceled-hundreds-of-flights.

[3] Reuters, "Mizuho says nearly $1 bln settlements may be disrupted by glitch," http://www.reuters.com/article/2011/03/18/mizuho-idUSL3E7EI02L20110318.

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

[5] N. Funabiki, T. Nakanishi, N. Amano, H. Kawano, Y. Fukuyama, and M. Isogai, "A software architecture and characteristic functions in learning management system "NOBASU"," Proc. Annual IEEE/IPSJ Symposium on Applications and Internet (SAINT 2010), pp. 109-112, July 2010.

[6] N. Funabiki, Y. Fukuyama, Y. Matsushima, T. Nakanishi, and K. Watanabe, " An improved Java programming learning system using test-driven development method," Proc. Int. MultiConference of Engineers and Computer Scientists (IMECS 2012), pp. 587-602, March 2012.

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

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

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)