# An Improvement of Informative Test Code Approach for Code Writing Problem in Java Programming Learning Assistant System

Khin Khin ZAW[*1], Nobuo FUNABIKI[*2]
Graduate School of Natural Science and Technology
Okayama University, Japan
Email: p8lj1oji@s.okayama-u.ac.jp, funabiki@okayama-u.ac.jp

**Abstract**: To assist Java programming educations, we have developed a Web-based *Java Programming Learning Assistant System (JPLAS)*. JPLAS provides the *code writing problem* to let students study writing a source code for a given assignment. In Java programming, *encapsulation*, *inheritance*, and *polymorphism* are the three fundamental concepts as the object-oriented programming language that every student should understand and freely use them. Previously, we proposed an *informative test code* approach for the code writing problem to help students studying them. Unfortunately, the current test code allows students to write source codes that pass the tests but do not properly use the three concepts. In this paper, we improve the informative test code by adopting the library functions to test the private variables and the variables/methods in the super/sub classes, so that the source code can be passed only when it uses the three concepts. For evaluations, we generated the improved informative test codes for *Que* and *Stack*, and asked two students to write the source codes, where they completed them using the three concepts.

**Key words:** JPLAS, informative, test code, encapsulation, inheritance, polymorphism

## 1. Introduction

To assist Java programming educations, we have developed a Web-based *Java Programming Learning Assistant System (JPLAS)* [1] that provides the *code writing problem* to write a source code for a given assignment. The correctness is verified through running the *test code* on *JUnit* as the *test-driven development (TDD) method* [2]. In Java programming, the *encapsulation*, *inheritance* and *polymorphism* are fundamental concepts as the *object-oriented programming (OOP)* language that every student should understand and freely uses them.

Previously, we proposed the i*nformative test code* approach for the code writing problem to help students studying these concepts through the code writing problem [3]. The informative test code describes names of class, methods and variables, access modifier, data types, exception handling, behaviors and class inheritance. By writing a source code to pass the *informative test code*, a student is expected to write a source code that adopts the same concepts of the model source code.

Unfortunately, the previous test code does not test private variables for *encapsulation*, and variables/methods in super/sub classes for *inheritance* and *polymorphism.* As a result, it allows students to write the source codes that pass the tests but do not properly use the three concepts.

In this paper, we improve the informative test code by adopting the library functions in Java to test private variables and variables/methods in super/sub classes. For evaluations, we generated the improved informative test codes for *Que* and *Stack* which follow the First-in-First-Out and Last-in Fast-Out. Then, we asked eight students to write the source codes, where they completed them using the three concepts.

## 2. Proposal of Improved Informative Test Code

In this section, we improve the *informative test code* for three concepts [4]-[6].

### 2.1 Informative Test Code for Que Using Encapsulation

The following **test code 1** shows a part of the test code that is generated from the source code for *Que* using *encapsulation*. In this source code, the three variables *content*, *tail*, and *head* are declared as *private* to be hidden from other classes. *content* stores the string or integer values, and *tail* and *head* store the first and last index number of the stored values in *content*. Then, three methods, *push*, *empty*, and *pop* are declared as *public. push* inserts a new value to *content*, *pop* retrieves the bottom value of the *content*, and *empty* returns true if *content* is empty. *content* can be accessed through *pop* and *push* methods.

**test code 1**

```
    ………………………

public void test() throws Exception{
    Que q = new Que();
    Field f1=q.getClass().getDeclaredField("content");
    Field f2=q.getClass().getDeclaredField("tail");
    Field f2=q.getClass().getDeclaredField("head");
    f2.setAccessible(true);
    f3.setAccessible(true);
    //check the behaviors of push method
    q.push(10);
    q.push("a");
    //check the value of private variables
    int tail=(int) f2.get(q);
    assertEquals(2, tail);
    //check the behaviors of pop and empty
    if(!q.empty()) {
        assertEquals(10,q.pop());
        assertEquals("a",q.pop());
    }
    int head=(int)f3.get(q);
    assertEquals(2, head);
}
    ………………………
```

In **test code 1**, library functions, *setAccessible* and *get*, test the values of the private variables, *tail*, *head*, in lines 8, 9, 12, 13, 18 and 19. Then, *pop t*ests the values of the private variable, *content*, in lines 15 and 16.

## 2.2 Informative Test Code for Stack Using Inheritance and polymorphism

The following **test code 2** shows a part of the test code that is generated from the source code for *Que* using *inheritance* and *polymorphism*. In the source code, the variables, *content*, *tail*, and *head*, and the methods, *empty* and *push*, are inherited from *Que.* Then, *pop* is overwritten in *Stack* from that in *Que.*

In the **test code 2**, the library function, *getSuperClass*, tests the super class in line 5. Then, the names of the variables, *content*, *tail*, and *head*, and the methods, *push*, *pop*, and *empty*, in the super class *Que* are described and tested in lines 6 to 11. Then, the names, access modifier and data types of *pop* in the sub class *Stack* are also described and tested in lines 12 to 14.

**test code 2**

```
1:
2:    ………………………..
3: public void test() throws Exception {
4:     Stack s = new Stack();
5:     Class<?> parentClass= s.getClass().getSuperClass();
        //check variable name
6:     Field f1=parentClass().getDeclaredField("content");
7:     Field f2=parentClass().getDeclaredField("tail");
8:     Field f2=parentClass().getDeclaredField("head");
9:     Method m1=parentClass().getDeclaredMethod
                                        ("empty",null);
10:    Method m2=parentClass().getDeclaredMethod
                                        ("pop",null);
11:    Method m3=parentClass().getDeclaredMethod
                                ("push",Object.class);
12:    Method mstack=s.getClass().getDeclaredMethod
                                        ("pop",null);
13:    assertEquals(mstack.getModifier(),
                                Modifier.PULIC);
14:    assertEquals(mstack.getReturnType(),Object.class);
15: }
16:    …………………….
```

## 3. Evaluation

In this section, we evaluate the improved *informative test codes* through applications to eight students in our group. First we prepared the informative test codes for *Que* and *Stack*.

Table 1: Metrics results for *Que*

| Metrics | Queue | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 |
| NOC | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| NOM | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| VG | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 1 |
| NBD | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| LCOM | .5 | 5 | .5 | .5 | .5 | .6 | .5 | .3 |
| TLC | 16 | 18 | 10 | 18 | 19 | 15 | 15 | 23 |
| MLC | 3 | 6 | 3 | 6 | 7 | 3 | 3 | 12 |

Then, we asked them to write the source codes for *Que* and *Stack* using the improved informative test codes. Table 1 shows the metric results of the eight source codes for *Que.* In any source code, NOC is 1 and NOM is 3. They have the same number of classes and methods. This reason is that any source is implemented using the class and methods given in the test code. They also have the good metrics for VG, NBD, and LCOM, where VG is 1-2, NBD is 1 and LCOM is 0.3- 0.6 respectively.

Table 2: Metrics results for *Stack*

| Metrics | Stack | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 |
| NOC | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| NOM | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| VG | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 |
| NBD | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| LCOM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TLC | 4 | 9 | 4 | 9 | 6 | 6 | 7 | 5 |
| MLC | 1 | 3 | 1 | 3 | 1 | 1 | 1 | 3 |

Table 2 shows the metric results of the eight source codes for *Stack.* In any code, NOC is 1 and NOM is 1. They have the same number of classes and methods by implemented using class and methods in test code. They also have good metrics for VG, NBD and LCOM where VG is 1-2, NBD is 1 and LCOM is 0 respectively.

This evaluation results show that the students completed high-quality source codes using three concepts for *Que* and *Stack* by following the intentions of the test codes. However, in both code, larger VG and MLC appear since the codes are complex and hard to be modified. The current test code cannot test the code quality directly.

## 4. Conclusion

In this paper, we proposed the improved *informative test code* for the code writing problem in JPLAS to help the students studying the *encapsulation*, *inheritance*, and *polymorphism.* We evaluated the effectiveness through the test codes for *Que* and *Stack using* three concepts. In future works, we will improve the test code to test the quality more and generate test codes using other source codes for the three concepts to assign them to students.

## References

(1) N. Funabiki, Y. Matsushim, T. Nakanishi, K. Watanabe, and N. Amano, "A Java programming learning assistant system using test –driven development method," IAENG Int.J. Computer Science, vol.40, no.1, pp.38-46, Feb.2013.

(2) K. Beck, Test-driven development: by example, Addison-Wesley, 2002.

(3) K. K. Zaw and N. Funabiki, "An informative test code approach for code writing problem in java programming learning assistant system," IEICE Tech. Report, SS-2017-10, pp.31-36, Oct. 2017.

(4) Encapsulation, https://www.tutorialspoint.com/java/java_encapsulation.htm.

(5) Inheritance, https://www.javapoint.com/inheritance-in-java

(6) Polymorphism, ttps://www.javapoint.com/runtime-polymorphism-in-java

(7) Stack and Que, https://en.wikibooks.org/wiki/Data_Structures/Stacks_and_Queue.