Generations of Informative Test Codes for Studying Encapsulation, Inheritance, and Polymorphism in Java Programming Learning Assistant System

Ei Ei MON[*1], Nubuo FUNABIKI [*2], Khin Khin ZAW[*3]
Graduate School of Natural Science and Technology
Okayama University, Japan
Email: eieimon@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, such that every student understands and freely uses them. Previously, to help their studies, we proposed the *informative test code* approach for the code writing problem in JPLAS, where we only generated the informative test codes for *Stack* and *Que* using the three concepts for evaluations. In this paper, we generate *informative test codes* for additional eight source codes using them in textbooks or Web sites and investigate their solution performances by five students who are studying Java programming in our group.

Key words：JPLAS, code writing problem, 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 let the students to study writing a source code for a given assignment. The correctness of the source code is verified through running the *test code* on *JUnit* based on the *test-driven development (TDD)* method [2]. In Java programming, *encapsulation*, *inheritance* and *polymorphism* are three fundamental concepts as the *object-oriented programming (OOP)* language that every student should understand and freely uses them.

Previously, to help their studies, we proposed the *informative test code* approach for the code writing problem in JPLAS [3], where we only generated the informative test codes for *Stack* and *Que* using the three concepts. The informative test code describes names of class, methods, and variables, access modifiers, data types, exception handling, code behaviors, and class inheritances that are related to the three concepts.

In this paper, we generate informative test codes for additional eight source codes in textbooks [4] or Web sites [5]-[9] and investigate solution performances of them by five students who are studying Java programming in our group.

2. Proposal of Informative Test Code for Three OOP Concepts

In this section, we present the *informative test code* for three OOP concepts [10]. **source code 1** describes the classes to show the information of the teacher and student such as the name and the salary using the three concepts. In *Teacher* class, there are two variables: *salary* and *name*. *salary* is declared as *private* to be hidden from other class and *name* is declared as *protected*, to be inherited in *Student* class. The five methods, *setSelary*, *getSalary*, *setName*, *getName*, and *status*, are declared using *public* as the setter/getter methods.

source code 1

```
1:  public class Teacher {
2:      private int salary;
3:      protected String name;
4:      public void setSalary(int salary){
5:          this.salary=salary;
6:      }
7:      public int getSalary(){
8:           return salary;
9:      }
10:     public void setName(String name){
11:         this.name=name;
12:     }
13:     public String getName(){
14:        return name;
15:     }
16:     public String status(){
17:        return name+" is teacher";
18:     }
19: }
20: public class Student extends Teacher{
21:     public String status(){
22:         return name+" is student";
23:     }
24: }
```

test code 1 represents the generated informative for **source code 1**. In **test code 1,** *test 1* method tests the names, access modifier, data types of variables and the methods in *Teacher* class in lines 4 to 18. Then, it tests the super class in line 20, and the names of the variables and the methods in the super class *Teacher* in lines 21 to 25. Finally, it tests the overwrite method *status* in the sub class in lines 26 to 28. *test2* method tests the behaviors of the variables and the methods by using the library function *setAccessible* in line 33. *get* is used to test the value of the private variable *salary* in lines 36. It also tests the value of the private variable in line 38.

test code 1

```
1:   ………………………
2: public void test1() throws Exception{
3:     Teacher t = new Teacher ();
4:     Field tf1=t.getClass().getDeclaredField("salary");
5:     Field tf2=t.getClass().getDeclaredField("name");
6:     assertEquals(tf1.getModifier(), Modifier.PRIVATE);
```

```
    assertEquals(tf2.getModifier(),
                                     Modifier.PROTECTED);
    assertEquals(tf1.getType(), int.class);
    assertEquals(tf2.getType(),String.class);
    Method tm1=t.getClass().getDeclaredMethod
                                         (“setSalary,int.class”);
    Method tm2=t.getClass().getDeclaredMethod
                                            (“getSalary,null”);
    Method tm3=t.getClass().getDeclaredMethod
                                       (“setName,String.class”);
    Method tm4=t.getClass().getDeclaredMethod
                                              (“getName,null”);
    Method tm5=t.getClass().getDeclaredMethod
                                               (“status,null”);
    assertEquals (tm1.getModifiers(), Modifier.PUBLIC);
    …………
    assertEquals(tm1.getReturnType(),void.class);
    …………
    Student s=new Student();
    Class<?> parentClass=s.getClass().getSuperClass();
    Field parentf1=parentClass.getDeclaredField
                                                   (“salary”);
    Field parentf1=parentClass.getDeclaredField(“name”);
    Method parentm1=parentClass.getDeclaredMethod
                                      (“setSalary”,int.class);
    …………..
    Method parentm1=parentClass.getDeclaredMethod
                                              (“status”,null);
    Method sm1=s.getClass().getDeclaredMethods
                                             (“status”, null);
    assertEquals(sm1.getModifiers(),Modifier.PUBLIC);
    assertEquals(sm1.getReturnType(), String.class);
}
public void test2() throws Exception{
    Teacher t = new Teacher();
    Field f = t.getClass().getDeclaredField(“salary”);
    f.setAccessibe(true);
    t.setSalary(5000);
    t.setName(“Mr.Yamada”);
    int salary = (int)f.get(t);
    assertEquals(5000,salary);
    assertEquals(5000,t.getsalary());
    assertEquals(“Mr.Yamada”,t.getName());
    assertEquals(“Mr.Yamada is teacher”, t.status());
    Student s = new Student();
    s.setName(“Mary”);
    assertEquals(“Mary”, s.getName());
    assertEquals(“Mary is student”, s.status());
}
```

3. Evaluation

We generated the informative test codes for eight source codes P1~P8 using three concepts. They contain the classes for teacher, two animals: *one for inheritance and one for polymorphism*, author, book, car, circle, and average calculation. Then, we asked five students to solve them. Table 1 shows the average metrics values of the complete codes measured by *Metrics plugin for Eclipse*. For each test code, any source code has the same NOC and NOM, except P5 by one student where he made default constructors but did not use them. In P8, NOM is 8 since he made default constructors but did not use them again. VG, NBD, and LCOM have good values in any source code. VG and NBD are 1 for any test code, since any code does not include nested loops and conditional statements. MLC is different depending on the code writing skill.

This evaluation results show that these students can complete the high-quality source codes using three concepts by following the intentions of the test codes. However, the current test code cannot test the unnecessary constructors in the source code.

Table 1: Metric Results

	P1	P2	P3	P4
NOC	2	3	1	1
NOM	2	3	6	8
VG	1	1	1	1
NBD	1	1	1	1
LCOM	0	0	0.53~0.667	0.643~0.75
TLC	16~19	18	26~28	34~35
MLC	2~4	3	8~9	11
	P5	P6	P7	P8
NOC	2	1	1	2
NOM	3~5	0	6	5~7
VG	1	1	1	1
NBD	1	1	1	1
LCOM	0.5	0	0.5~0.667	0.4~0.5
TLC	17~23	12~15	23~26	23~35
MLC	3~4	3~6	6~9	5~8

4. Conclusion

This paper generated informative test codes for eight source codes using *encapsulation*, *inheritance*, and *polymorphism* for the code writing problem in JPLAS. In future works, we will improve the informative test code to test unnecessary constructors in the source code.

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] Java Books, https://beginnersbook.com/2013/03/.
[5] Complete Java, https://www.zealseeds.com/Lang/LangJava/BasicGrammar/InheritanceOfJava/index.html.
[6] Introduction to Java, http://www1.bbiq.jp/takeharu/java100.html.
[7] Java Programming, http://java.sevendays-study.com/ex-day2.html.
[8] Java Class, https://itsakura.com/java-inheritance.
[9] Inheritance and polymorphism, https://gist.github.com/rtoal/1685886e6605fe73b792.
[10] OOP concept, https://stackify.com/oops-concepts-in-java/.