An Informative Test Code Approach in Code Writing Problem for Java Collections Framework in Java Programming Learning Assistant System

Ei Ei Mon[1], Nobuo Funabiki[1*], Minoru Kuribayashi[1], Wen-Chung Kao[2]
[1] Department of Electrical and Communication Engineering, Okayama University, Okayama, Japan.
[2] Department of Electrical Engineering, National Taiwan Normal University, Taipei, Taiwan.

** Corresponding author. Email: funabiki@okayama-u.ac.jp
Manuscript submitted November 6, 2018; accepted April 10, 2019.
doi: 10.17706/jsw.14.5.200-208

Abstract: To enhance Java programming educations, we have developed a *Java Programming Learning Assistant System (JPLAS).* In JPLAS, the *code writing problem* asks a student to implement a source code that passes the given *test code* on *JUnit*, where the details of the implementation are described in the test code. Previously, we confirmed the effectiveness of this *informative test code approach* in studying three *object-oriented programming* concepts for Java. In this paper, we present its application to studying *Java Collections Framework (JCF).* JCF enables us to handle a group of objects by offering appropriate libraries, which is expected to be mastered by the students. For evaluations, we generated five informative test codes for JCF, and asked 19 students from Japan, Myanmar, China, and Indonesia to implement the source codes. Then, all of them completed the source codes passing the test codes, while certain students did not use the expected JCF library functions.

Key words: Java, programming, JPLAS, JUnit, test code, collections framework.

1. Introduction

Nowadays, *Java* has been broadly used in various practical applications as a highly reliable, portable, and scalable objected oriented programming language. Java is a versatile general-purpose programming language that can be used to create cross-platform applications for desktop, mobile, or Web systems [1]. Java was the most popular programming language in 2015 [2] and is in the third in 2018 [3]. Therefore, there have been strong demands of 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 and programming exercises.

To improve programming exercise environments, we have developed *Java Programming Learning Assistant System (JPLAS)* [4]. JPLAS performs excellently not only in reducing teacher loads by marking answers automatically but also in advancing student motivations with immediate responses. JPLAS offers the code writing problem [5], which asks a student to write a source code that passes the given test code on *JUnit* [6]. To help a student write a complex code, the detailed information for the source code implementation is described in the test code. Previously, we confirmed the effectiveness of this *informative test code approach* in studying the three *object-oriented programming* concepts for Java [7]. It is expected that a student will learn how to use the concepts by writing a source code which can pass the test code.

Java collections framework (JCF) [8] provides a strong and useful architecture to store or control a group of objects by offering the appropriate library methods. Unlike arrays, JCF covers *wrapper classes* such as *Integer, Long, or Double.* Moreover, they can be enlarged or shrink in size automatically when the corresponding objects are added or removed. Students are required to master the use of JCF in a source code at Java programming study. *List, Set,* and *Map* are the most useful three interfaces in JCF.

In this paper, we present the application of the informative test code approach for studying the three interfaces in JCF. This test code intends to test whether a method in a source code uses the proper method in the JCF library. For evaluations, we generated five informative test codes for JCF, and asked 19 students from Japan, Myanmar, China, and Indonesia to implement the source codes. Then, all of them successfully completed the source codes whose software metrics confirm the sufficient quality. However, certain students did not properly use methods in the JCF library.

The rest of this paper is organized as follows. Section 2 presents the informative test code approach for JCF. Section 3 shows the evaluation result. Finally, Section 4 concludes this paper with future works.

2. Informative Test Code Approach for Java Collections Framework

In this section, we present the informative test code approach for implementing a source code using JCF.

2.1. Review of Java Collections Framework

Java Collections Framework (JCF) is a hierarchy of interfaces and classes that are used for storing and manipulating a group of objects. The most useful interfaces in JCF are *List, Set, and Map*. An *iterator* method is prepared to go through all the elements in each collection.

List is a more flexible version of an array to be used to insert and retrieve the objects. It allows the duplicated objects. *List* includes the methods for *get, set, add, indexOf,* and *lastIndexOf*. *List* can be implemented as *ArrayList* or *LinkedList*. *Set* is an unordered collection of objects and does not allow duplicate objects. Any object in *Set* cannot be accessed by using the index. *Set* can be implemented as *TreeSet*, *HashSet*, or *LinkedHashSet*. *Map* is an object that maps keys to values and cannot contain duplicate keys. *Map* includes the methods for *put, get, remove, containsKey,* and *containsValue*. The three basic implementations of *Map* are *HashMap, TreeMap,* and *LinkedHashMap. Iterator* is used in JCF to retrieve elements one by one [9]. There are three *iterator* interfaces*: Enumeration, Iterator,* and *ListIterator.*

2.2. Overview of Informative Test Code for JCF

The *informative test code* describes the necessary information for implementing the source code using the key point under study, such as the grammar, the framework, or the concept. This test code includes the class names, the method names, the access modifiers, and the data types for the important member variables, the argument types, and the returning data types for the methods, and the exception handling in the expected source code. By writing a source code to pass this test code, a student is expected to learn how to use the point under study in the source code. In general, an *informative test code* consists of a test method for testing names and data types of member variables in the source code, named "*variableNameTest*", a test method for testing the number of methods, their names, and returning data types, named "*methodNameTest*", and one or more test methods for testing actions of methods in the code, named "*(action)Test*".

2.3. Informative Test Code for List

test code 1 demonstrates an informative test code for *List*. *variableNameTest* tests that in the source code, "*ArrayListImp* class" is implemented, one variable "*list*" is defined, and its data type is *List* object. *methodNameTest* tests that in the code, three methods exist, their names are "*addImp*", "*swapImp*", and "*removeImp*", they have no arguments, and the returning data type is *List* object. *addTest* tests that *addImp*

returns the *List* object of *String*, and adds *"Red", "Green", "Blue", "Black"* and "*Orange*" to *list* in this order. *swapTest* tests that *swapImp* exchanges the first and third data in *list*. *removeTest* tests that *removeImp* removes *Red* and *Blue* from *list*.

Source code 1 shows an example source code for **Test code 1**.

Test code 1

```
1   public class ArrayListImpTest {
2       @Test
3       public void variableNameTest() throws NoSuchFieldException, SecurityException {
4           ArrayListImp obj = new ArrayListImp();
5           Field f1 = obj.getClass().getDeclaredField("list");
6           Field[] f = obj.getClass().getDeclaredFields();
7           assertEquals(1, f.length);
8           assertEquals(f1.getType(), List.class);
9       }
10      @Test
11      public void methodNameTest() throws NoSuchMethodException, SecurityException f
12          ArrayListImp obj = new ArrayListImp();
13          Method[] m = obj.getClass().getDeclaredMethods();
14          assertEquals(3, m.length);
15          Method m1 = obj.getClass().getDeclaredMethod("addImp", null);
16          Method m2 = obj.getClass().getDeclaredMethod("swapImp", null);
17          Method m3 = obj.getClass().getDeclaredMethod("removeImp", null);
18          assertEquals(List.class, m1.getReturnType());
19          assertEquals(List.class, m2.getReturnType());
20          assertEquals(List.class, m3.getReturnType());
21      }
22      @Test
23      public void addTest() {
24          ArrayListImp obj = new ArrayListImp();
25          List<String> list = obj.addImp();
26          assertEquals(5, list.size());
27          assertEquals("Red", list.get(0));
28          assertEquals("Green", list.get(1));
29          assertEquals("Blue", list.get(2));
30          assertEquals("Black", list.get(3));
31          assertEquals("Orange", list.get(4));
32      }
33      @Test
34      public void swapTest() {
35          ArrayListImp obj = new ArrayListImp();
36          List<String> list = obj.addImp();
37          obj.swapImp();
38          assertEquals(5, list.size());
39          assertEquals("Blue", list.get(0));
40          assertEquals("Red", list.get(2));
41      }
42      @Test
43      public void removeTest() {
44          ArrayListImp obj = new ArrayListImp();
45          List<String> list = obj.addImp();
```

```
46              obj.removeImp();
47              assertEquals(3, list.size());
48              assertEquals("Green", list.get(0));
49              assertEquals("Black", list.get(1));
50              assertEquals("Orange", list.get(2));
51          }
52  }
```

Source code 1

```
1   public class ArrayListImp {
2       List<String> list = new ArrayList<String>();
3       public List<String> addImp() {
4           list.add("Red");
5           list.add("Green");
6           list.add("Blue");
7           list.add("Black");
8           list.add("Orange");
9           return list;
10      }
11      public List<String> swapImp() {
12          Collections.swap(list, 0, 2);
13          return list;
14      }
15          public List<String> removeImp() {
16          list.remove(0);
17          list.remove(1);
18          return list;
19      }
20  }
```

2.4. Informative Test Code for Set

Test code 2 reveals an informative test code for Set. *variableTest* tests that in the source code, *TreeSetImp* class is implemented, two variables, "*tset*" and "*iterator*", are defined, and their data types are *TreeSet* of *String* objects and *Iterator* respectively. *methodTest* tests that the number of the methods is three, their names are "*addImp*", "*removeImp*", and "*displayImp"*, they have no argument, and their returning data types are *TreeSet*. *addTest* tests that every element in *tset* is *String* object, the number of elements is six, the first element is *"ABC"*, and the last one is "Test". *removeTest* tests that two elements are removed from *tset*, the first element becomes *"ABC"*, and the last one becomes *"String"*. *displayTest* tests that the output data from *tset* at the console is "*ABC Ink Jack Pen String Test*" after applying *addImp* to *tset*.

Test code 2

```
1   public class TreeSetImpTest {
2       ..............
3       @Test
4       public void addTest() {
5           TreeSetImp obj = new TreeSetImp();
6           TreeSet<String> tset = obj.addImp();
7           for (Object x : tset) {
```

```
8                   assertEquals("String".getClass().getName(), x.getClass().getName());
9               }
10              assertEquals(6, tset.size());
11              assertEquals("ABC", tset.first());
12              assertEquals("Test", tset.last());
13      }
14      @Test
15      public void removeTest() {
16              TreeSetImp obj = new TreeSetImp();
17              TreeSet<String> tset = obj.addImp();
18              obj.removeImp();
19              assertEquals(4, tset.size());
20              assertEquals("ABC", tset.first());
21              assertEquals("String", tset.last());
22      }
23      @Test
24      public void displayTest() {
25              TreeSetImp obj = new TreeSetImp();
26              TreeSet<String> tset = obj.addImp();
27              ByteArrayOutputStream baos=new ByteArrayOutputStream(1024);
28              PrintStream s = System.out;
29              PrintStream st = new PrintStream(baos);
30              System.setOut(st);
31              obj.displayImp();
32              System.setOut(s);
33              assertEquals("ABC Ink Jack Pen String Test", baos.toString().trim());
34      }
35  }
```

Source code 2 shows an example source code for **test code 2**. In *addImp*, the *six* strings are added to *tset* with the consideration of *displayTest*. In *removeImp*, one element among *"Ink", "Jack", "Pen",* and *"Test"* are removed. Here, *"Pen"* and *"Test"* are actually removed. In *displayImp*, each element in *tset* is output at the console sequentially using *Iterator*.

Source code 2

```
1   public class TreeSetImp {
2       TreeSet<String> tset = new TreeSet<String>();
3       Iterator iterator;
4       public TreeSet<String> addImp() {
5               tset.add("ABC");
6               ……..
7               tset.add("Test");
8               return tset;
9       }
10      public TreeSet<String> removeImp() {
11              tset.remove("Pen");
12              tset.remove("Test");
13              return tset;
14      }
15      public void displayImp() {
16              Iterator<String> itr = tset.iterator();
17              while (itr.hasNext()) {
```

```
18                  String item = itr.next();
19                  System.out.print(item + " ");
20          }
21  }
```

2.5. Informative Test Code for Map

Test code 3 exhibits an informative test code for *Map*. *variableTest* tests that in the source code, "*HashMapImp* class" is implemented, one variable, "*hmap*", is defined, and the data type is *"HashMap"*. *methodTest* tests that the number of the methods is three, their names are "*putImp*", "*removeImp*", and "*displayImp*", they have no argument, and their returning data types for the first two methods is *"HashMap"* and no returning data for the last one. *putTest* tests that *hmap* accepts the returning data with a pair of *Integer* and *String*, *putImp* adds "*Coconut*", "*strawberry*", "*Apple*", "*Mango*", and "*Orange*" to *hmap* using 1 to 5 for the key. *displayTest* tests that one element is removed from *hmap*, and the output data from *hmap* at the console is "*1=Coconut, 2=Strawberry, 4=Mango, 5=Orange*" after applying *putImp* and *removeImp* to *hmap*.

source code 3 shows an example source code for **test code 3**. In *removeImp*, *"Apple"* is removed. In *displayImp*, each key and value in *hmap* are output at the console sequentially using *Iterator*.

Test code 3

```
1   public class HashMapImpTest {
2       ..............
3       @Test
4       public void putTest() {
5           HashMapImp obj = new HashMapImp();
6           HashMap<Integer,String> hmap = obj.putImp();
7           assertEquals(5, hmap.size());
8           assertEquals("Coconut", hmap.get(1));
9           ..............
10          assertEquals("Orange", hmap.get(5));
11      }
13      @Test
14      public void displayTest() {
15          HashMapImp obj = new HashMapImp();
16          HashMap<Integer,String> hmap = obj.putImp();
17          obj.removeImp();
18          ..............
19          assertEquals("1=Coconut 2=Strawberry 4=Mango 5=Orange", baos.toString().trim());
20      }
21  }
```

Source code 3

```
1   public class HashMapImp {
2       HashMap<Integer, String> hmap = new HashMap<Integer, String>();
3       public HashMap<Integer, String> putImp() {
4           hmap.put(1, "Coconut");
5           ……..
6           return hmap;
7       }
8       HashMap<Integer, String> removeImp() {
```

```
9               hmap.remove(3);
10              return hmap;
11      }
12      public void displayImp() {
13              Iterator<Entry<Integer, String>> itr = hmap.entrySet().iterator();
14              String str = "";
15              while (itr.hasNext()) {
16                      Map.Entry<Integer, String> pair = (Map.Entry<Integer, String>) itr.next();
17                      str += pair.getKey() + "=" + pair.getValue() + " ";
18              }
19              System.out.println(str);
20      }
21  }
```

3. Evaluation

In this section, we evaluate the informative test code approach for JCF. We generated five informative test codes for JCF, three for *List*, one for *Set*, and one for *Map*. Then, 19 students with different programming skills from Japan, Myanmar, China, and Indonesia were asked to complete the source codes passing the test codes. In the end, we measured the software metrics using *metrics plugin for Eclipse* [10].

Table I shows the summary of the measured metrics of the source codes from the students. It is noted that all the students completed the source codes. This table indicates that generally, they are high quality since the metrics values exist in superior ranges. First, *NOM* (*number of methods*) shows that one student always uses one more method than the others, because he creates the constructor to initialize the variables. The test code on JUnit can test the number of constructors and the number of methods separately. However, *metrics plugin for Eclipse* counts them together in total. Thus, it appears to be difficult to control the implementation of the constructor by the test code.

Next, for *swapImp* in *ArrayListImp*, nine students used *Collections.swap* as the proper JCF library function to exchange two objects in the list as in **source code 1**, which is expected to study JCF. By contrast, four students did not adopt the library function, and two students directly set the data in the list, instead of swapping them, which are not expected. For *displayImp* in *TreeSetImp*, 12 students used *iterator* to display the data in *tset* as expected. However, one student used the *enhanced for-loop statement* instead of *iterator*, and two students directly displayed the expected result in the test code by copying it instead of displaying the data in *tset*. The strategy to avoid them will be explored in future works.

Table 1. Comparison of Metric Values for JCF

Problem Name	NOC	NOM	VG	NBD	LCOM	TLC	MLC
BookData	2	3-4	2	2	0	31-40	11-19
ArrayListImp	1	4-5	1-1.25	1-1.25	0.5-0.75	29-36	12-19
LinkedListImp	1	4-5	1-1.50	1-1.50	0	23-29	9-15
TreeSetImp	1	3-4	1-1.33	1-1.33	0.5-0.75	25-32	13-18
HashMapImp	1	3-4	1-1.33	1-1.33	0	20-28	9-14

4. Conclusion

In this paper, we presented the informative test code approach to the code writing problem for studying the *Java Collections Framework (JCF)* in *Java Programming Learning Assistant System (JPLAS).* The informative test code describes the detailed information for the code implementation including methods and variables. We evaluated the effectiveness of the approach by asking 19 students to write source codes

using JCF libraries for five informative test codes, where all of them completed high quality source codes that pass the test codes. However, some students did not use JCF library functions. In future works, we will improve the informative test code to confirm the proper use of library functions for JCF.

References

[1] Garbade, M. J. (2018). Top 3 most popular programming languages in 2018. Retrieved from: https://hackernoon.com/top-3-most-popular-programming-languages-in-2018-and-their-annual-salaries-51b4a7354e06

[2] Cass. S. (2015). The 2015 top ten programming languages — IEEE spectrum. Retrieved from: https://www.linkedin.com/pulse/2015-top-ten-programming-languages-ieee-spectrum-farzin-pashaee

[3] Cass. S., & Bulusu, P. (2018). Interactive: The top programming languages 2018. Retrieved from: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018

[4] Funabiki, N., Zaw, K. K., Ishihara, N., & Kao, W.-C. (2016). Java programming learning assistant system: Retrieved from: https://www.worldscientific.com/worldscibooks/10.1142/10727

[5] Funabiki, N., Matsushim, Y., Nakanishi, T., Watanabe, K., & Amano, N. (2013). A Java programming learning assistant system using test-driven development method. *IAENG Int. J. Comput. Sci.*, *40(1)*, 38-46.

[6] JUnit. Retrieved from: http://www.junit.org/

[7] Zaw, K. K., Funabiki, N., Mon, E. E., & Kao, W.-C. (2018). An informative test code approach for studying three object-oriented programming concepts by code writing problem in Java programming learning assistant system. *Proceedings of the* G*lobal Conf. Consum. Elect*.

[8] JCF. Retrieved from: https://docs.oracle.com/javase/tutorial/collections/

[9] Iterators in Java. Retrieved from: https://www.geeksforgeeks.org/iterators-in-java/

[10] Metrics 1.3.6. Retrieved from: http://metrics.sourceforge.net.

Ei Ei Mon received B.E. degree from Technological University, Monywa, and the M.E. degree from Mandalay Technological University, Myanmar, in 2006 and 2011. Since 2015, she has been a lecturer in Technological University, Monywa, Myanmar. She is currently a Ph.D. candidate in Okayama University, Japan. Her research interests include computer network and educational technology. She is a student member of IEICE.

Nobuo Funabiki received B.S. and Ph.D. degrees in mathematical engineering and information physics from the University of Tokyo, Japan, in 1984 and 1993, and M.S. in electrical engineering from Case Western Reserve University, USA, in 1991. Since 2001, he has been a professor in the Department of Electrical and Communication Engineering, Okayama University, Japan. His research interests include computer network, optimization algorithm, and educational technology. He is a member of IEEE, IEICE, and IPSJ.

Minoru Kuribayashi received B.E., M.E., and D.E. degrees from Kobe University, Japan, in 1999, 2001, and 2004. Since 2015, he has been an associate professor in Graduate School of Natural Science and Technology, Okayama University, Japan. His research interests include digital watermarking, information security, cryptography, and coding theory. He is a senior member of IEEE and IEICE.

Wen-Chung Kao received M.S. and Ph.D. in electrical engineering from National Taiwan University, Taiwan, in 1992 and 1996. In 2004, he joined National Taiwan Normal University, Taiwan, where he is a professor at Department of Electrical Engineering. His research interests include SoC, flexible electrophoretic display, and machine vision system. He is a fellow of IEEE.