N-017

Implementation of Naming Rules Checking Function in Code Validation Program for Code Writing Problem in Java Programming Learning Assistant System

Khaing Hsu Wai† Nobuo Funabiki† Mustika Mentari† Soe Thandar Aung†
Wen-Chung Kao††

khainghsuwai@s.okayama-u.ac.jp funabiki@okayama-u.ac.jp

Abstract To assist self-studies of *Java programming* for novice students, we have developed a web-based *Java programming learning assistant system (JPLAS)*. In JPLAS, the *code writing problem (CWP)* asks a student to write a *source code* that will pass the given *test code* in the assignment, where the correctness is validated by *unit test* using *JUnit*. For novice students, to master writing *readable codes* using proper names for variables, classes, and methods is crucial in *Java programming*, to improve *understandability* and *maintainability*. In this paper, we implement the *naming rules checking function* in the *code validation program* for *CWP* in *JPLAS*, and apply it to 547 source codes from 50 students in a Java programming course in Okayama University, Japan. The results confirm the effectiveness.

1. Introduction

Nowadays, *Java* has been widely adopted in industries as a reliable and portable *object-oriented* programming language. There is a strong demand for Java programming educations. A great number of universities and professional schools are offering Java programming courses to meet these needs. To assist self-study of *Java programming* for novice students, we have developed the *Java programming learning assistant system (JPLAS)*. *JPLAS* offers a personal answer platform implementing *automatic marking functions* that is built on *Node.js* and is distributed using *Docker* [1]. *JPLAS* provides several types of exercise problems, allowing students to progress *Java programming* learning through various stages and levels step-by-step by their own.

Among them, the *code writing problem (CWP)* asks to write a source code that will pass the tests in the given *test code*, which will run and test the source code on *JUnit*. In *CWP*, a student can repeat the cycle of writing, testing, and modifying the source code for the assignment by him/herself, until completing the correct one. To help a teacher in validating the source codes from a lot of students, we have implemented the *code validation program* [2]. It automatically tests all the *source codes* from students using the test code for each assignment, and generates a CSV file that provides the number of passed tests in each source code. Thus, the teacher can quickly assess the correctness of all the student answers and assign their grades.

For novice students, to master writing *readable codes* using proper names for variables, classes, and methods *Java programming* is crucial at the early stage of programming studies, to develop good coding habits and foundational skills. A *readable code* can be made by following coding rules that are composed of *naming rules*, *coding styles*, and *potential problems*. *Coding rules* [3] represent a set of rules or conventions for producing high quality source codes. By following them, the uniformity of a code will be maintained, which enhances the *readability*, *maintainability*, and *scalability*. Among them, we focus on *naming rules* as the first step for novice students to follow.

In this paper, we implement the *naming rules checking function* in the *code validation program* for *CWP* in *JPLAS*. It finds the naming errors in the source code. We applied it to 547 source codes from 50 students in a Java programming course in Okayama University, Japan. The results confirmed the effectiveness of the function.

2. Review of Code Writing Problem

In this section, we review the *code writing problem (CWP)* in *JPLAS*.

An *CWP* instance requests a student to write the *source code* that will pass every *test case* described in the *test code*. The correctness is validated by running the *test code* with the *source code* on *JUnit*. It is necessary to write the source code following the specifications described in the *test code*.

The following **myAddTest** class describes the *test code* for the **myAdd** class in the *source code* that returns the summation of two integer arguments. The **assertEquals** method describes the *test case* of comparing the output of the **plus** method in the *source code* with its correct result.

myAddTest class for *test code* sample.

```
import static org.junit.Assert.*;
import org.junit.Test;
public class myAddTest {
@Test
   public void testPlus(){
      myAdd ma = new myAdd();
      int result = ma.plus(1,4);
      assertEquals(5, result);
   }
}
```

myAdd class for *source code* sample.

```
public class myAdd {
   public int plus(int a, int b) {
      return (a+b);
   }
}
```

† Department of Information and Communication Systems Okayama University, Japan
†† Department of Electrical Engineering, National Taiwan Normal University, Taiwan

Table 1 Summary of application results.

problem name	# of students	proper name	length	Camel case	dictionary	abstract word
HelloWorld.java	49	61	0	2	2	0
MessageDisplay.java	50	65	0	0	0	0
CodeCorrection1.java	50	183	12	2	1	0
CodeCorrection2.java	50	183	12	2	1	0
IfAndSwitch.java	50	246	4	3	0	1
EscapeUsage.java &	50	57	0	0	0	0
ReturnAndBreak.java	50	167	40	5	4	0
OctalNumber.java	50	137	3	6	24	8
Hexadecimal.java	50	129	3	9	15	1
MaxItem.java	50	234	9	3	3	1
MinItem.java	48	225	9	3	4	1
Total	547	1687	92	35	54	12

3. Implementation of Naming Rules Checking Function

In this section, we present the implementation of the *naming rules checking function* in the *code validation program*. The function is modified from the previous one in [3].

The following four naming rules are checked in the function:

1) Camel case: any identifier name must follow *Camel case*, which is checked using the *regular expressions*.
2) English dictionary: any identifier name must be a correct English that is included in an English dictionary. A name composed of multiple words using the *Camel case* is separated into individual words. This test also helps avoid spelling errors and ensure the use of understandable words.
3) Abstract word: any abstract word such as *data*, *size*, and *make* should be avoided. For this test, the list of abstract words is prepared.
4) Word length: the length of any identifier name except for commonly used words in the list such as **i** and **j** for a loop pointer should be between 3 and 17.

The following procedure is implemented in the *code validation program* that will check all the source codes in one folder that were submitted for one assignment from students:

1) Read a source code from each student.
2) Check the *four naming rules* of the code.
3) Save the result in the text file under the *output* folder.
4) Repeat the procedure for every source code in the folder.
5) Generate the CSV file in the *csv* folder and save the summary result for all the codes there.

4. Evaluation

In this section, we applied the proposal to 547 source codes from 50 students to 11 basic CWP assignments in 2023 Java programming course in Okayama University, Japan. Table 1 shows the assignment class name, the number of submitted students, the number of proper names, and the number of errors in identifier names found by the function for each rule. It is noted that a *proper name* indicates the one following the rules, which can enhance *understandability* and *maintainability* of the source code. Table 1 suggests that most students used proper identifier names in their source codes whereas a small number of students made errors in the naming rules.

5. Conclusion

This paper presented the implementation of the *naming rules checking function* in the *code validation program* for the *code writing problem* in *Java programming learning assistant system (PLAS)*. It was evaluated through applications to 547 source codes from 50 students in a Java programming course in Okayama University, Japan. In future works, we will implement the proposal in the answer platform so that a student will write a code while checking the rules.

References

[1] S. T. Aung, N. Funabiki, L. H. Aung, H. Htet, H. H. S. Kyaw, and S. Sugawara, “An implementation of Java programming learning assistant system platform using Node.js,” in Proc. Int. Conf. Inf. Educ. Tech., pp. 47-52, 2022.
[2] K. H. Wai, N. Funabiki, S. T. Aung, K. T. Mon, H. H. S. Kyaw and W.-C. Kao, “An implementation of answer code validation program for code writing problem in Java programming learning assistant system,” in Proc. ICIET, pp. 193-198, 2023.
[3] N. Funabiki, T. Ogawa, N. Ishihara, M. Kuribayashi, and W.-C. Kao, “A proposal of coding rule learning function in Java programming learning assistant system,” in Proc. CISIS, pp. 561-566, Sep. 2016.