Iyer, S. et al. (Eds.) (2022). Proceedings of the 30th International Conference on Computers in Education.
Asia-Pacific Society for Computers in Education

Learning Support System Visualizing Relationships Among Classes and Objects Based on Teacher's Intent of Instruction

Koichi YAMASHITA[a*], Yusuke SUZUKI[b], Satoru KOGURE[b], Yasuhiro NOGUCHI[b], Raiya YAMAMOTO[c], Tatsuhiro KONISHI[b] & Yukihiro ITOH[d]
[a]*Faculty of Business Administration, Tokoha University, Japan*
[b]*Faculty of Informatics, Shizuoka University, Japan*
[c]*Faculty of Engineering, Sanyo-Onoda City University, Japan*
[d]*Shizuoka University, Japan*
*yamasita@hm.tokoha-u.ac.jp

Abstract: While several learning support systems that visualize program behaviors for various object-oriented languages have been developed to date, many of them are not sufficient for learners to understand the concepts specific to object-oriented languages. We have developed a program visualization system that supports learners' understanding by visualizing relationships among objects and classes. We evaluated the effectiveness of our system by introducing it into an actual class and measuring the degree of improvement in learners' understanding based on tests. The evaluation results suggest that our system would have a certain degree of effectiveness for learning programming using object-oriented languages.

Keywords: Programming education, program visualization system, object-oriented conceptual model, educational authoring tool

1. Introduction

To date, several program visualization (PV) systems have been developed to support learners' understanding of programs (Sorva, Karavirta, & Malmi, 2013). We have developed a PV system for C programs called Teacher's Explaining Design Visualization Tool (TEDViT), and have conducted several classroom practices using this system (Kogure et al., 2014). On the other hand, Java, rather than C, has been increasingly adopted as a language used in introductory programming education for novice students in recent years. Therefore, this study attempts to extend TEDViT to support Java and to develop a PV system that can be adapted to a variety of programming learning environments for novice learners.

As an approach to make TEDViT capable of object-oriented languages, we have developed a visualization model called Object-Oriented Conceptual Model (OOCM) and have evaluated effectiveness of PVs based on the model for understanding class and object concepts based on questionnaire surveys (Kogure et al., 2019). However, the PV system based on OOCM had implementation problems and required a non-trivial amount of manual coding works to generate PVs. The purpose of this study is to develop a system visualizing PVs based on OOCM with no manual interventions and to measure their effectiveness by introducing the system into an actual class.

2. PV System Visualizing OOCM Based on Teacher's Intent of Instruction

Object-oriented languages such as Java are increasingly being used in programming education for novice learners. Based on this background, we adapted Object-Oriented Conceptual Model (OOCM) (Kogure et al., 2019) as our approach to make TEDViT support Java. OOCM is a visualization model based on Unified Modeling Language (UML) and can visualize the relationships among objects and classes, including the concepts of inheritance and polymorphism. In this paper, we refer to the PV system developed by Kogure et al. as Kogure system.

Kogure system is a PV system based on TEDViT which visualizes program behavior based on the execution history. Execution history is a log file generated by executing the target program in which the system embedded statements for recording execution statuses of the program. However, due to implementation problems, Kogure system requires manual modification of execution history to visualize OOCM-based PV of Java programs. Therefore, in this study, we developed a new PV system by radically modifying Kogure system to eliminate the need for manual intervention. In our system, the execution history is generated from Java Debug Interface (JDI) framework, so there is no need to add statements to observe the execution status.

Figure 1 shows a screenshot of our system. Our system generates a learning environment consisting of three main visualization fields. Field (A) visualizes the target program code and highlights the statement being executed. Learners can perform stepwise execution of the target program by clicking “Next” or “Prev” button. The source code of each class can also be visualized by changing the target program file with pull-down menu. The data structure to be processed by the method being executed is visualized in field (B). Field (C) visualizes the classes and objects that are interrelated in the target program based on OOCM. In fields (B) and (C), teachers can define the visualization by providing drawing rules to specify when, what, and how variables, classes, and objects are visualized.

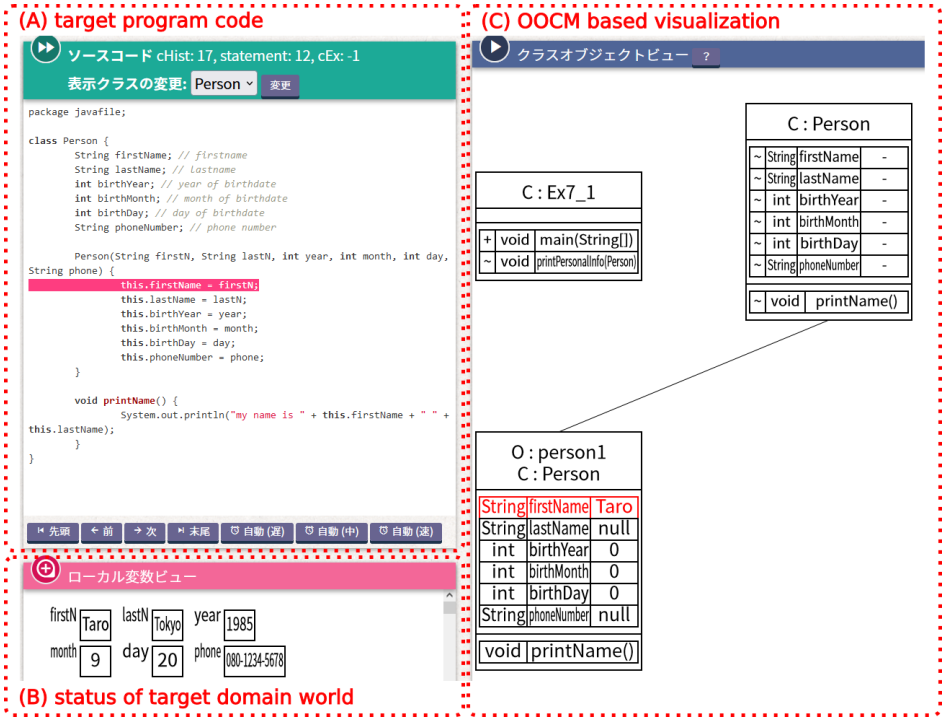

Figure 1. Learning environment generated by our system.

3. Evaluation

In order to evaluate the effectiveness of our system, we conducted a classroom practice using our system, incorporating it into an actual introductory programming course. The hypothesis to be verified is that, compared to the usual classroom exercise consisting of code reading and program execution of sample program, observing the behavior of sample program using our system would cultivate better understanding of the concepts specific to object-oriented programming. The actual class in which our system was introduced was an on-demand class offered to first-year university students majoring in computer science. 68 students with less than one year of programming experience participated in this practical class.

The practiced class includes two exercises: Exercise A, which was usual-style exercise as in regular classes, and Exercise B, which used our system. Both exercises were based on the same sample program, but Exercise A required only code reading and program execution, while Exercise B required only learning in the learning environment of our system. We divided the participants into two groups,

Group 1 and Group 2 and reversed the order in which Exercise A and Exercise B were performed. We tested the participants' understanding three times: before the class (pre-test), after the first exercise (middle-test), and after the second exercise (post-test). Because the classes were on-demand, we did not perform grade equalization based on the pre-test score and grouping into two groups was done randomly. Each test consisted of 20 questions: 3 questions on the properties of concepts specific to object-oriented languages, 6 questions on the behavior of class and object variables, 5 questions on the behavior of programs with inheritance, and 6 questions on the behavior of programs with overriding. The score was calculated with 100 as the maximum score.

Table 1 shows the results of the tests for each of the two groups based on the average of the scores. The underlined scores in Table 1 show the scores immediately after Exercise B where participants use our system. We can see that both groups improved their test scores by learning with our system, suggesting that our system has a certain effect on learning concepts specific to object-oriented languages. On the other hand, especially in Group 1, the improvement in test scores can be seen even when learning without using our system. Hence, the results do not positively support the hypothesis that our system is more effective than the usual-style exercises. However, there are some positive indications for the use of our system, such as a higher improvement in scores in Group 2 compared to the usual-style exercises and a significant improvement in the lowest test scores after the exercises using our system. These results suggest that our system is effective for learning programming in object-oriented languages such as Java.

Table 1. *Average Scores of Pre-, Middle- and Post-test*

Group	N.	1st exercise	Pre-test	Middle-test	Post-test
Group 1	29	No system	75.3	80.0	<u>83.2</u>
Group 2	39	With system	80.7	<u>83.6</u>	84.1

4. Conclusion

In this paper, we describe a PV system which visualizes program behaviors based on teachers' intents of instructions, employing OOCM as an approach to visualize concepts specific to object-oriented languages. We evaluated the effectiveness of our PV system by introducing it in an actual class. The evaluation results did not positively support our hypothesis that observing the behavior of the sample programs using our system would cultivate better understandings of object-oriented concepts, compared to the usual classroom exercises consisting of code readings and program executions. However, some positive indications were obtained from the distribution of scores, suggesting that our system would have a certain degree of effectiveness on learning programming with object-oriented languages such as Java.

Acknowledgements

This study was supported by JSPS KAKENHI Grant Numbers JP18K11567, JP19K12259, and JP22K12290.

References

Kogure, S., Fujioka, R., Noguchi, Y., Yamashita, K., Konishi, T., & Itoh, Y. (2014). Code reading environment according to visualizing both variable's memory image and target world's status. *Proceeding of the 22nd International Conference on Computers in Education (ICCE2014)*, 343-348.

Kogure, S., Ogasawara, K., Yamashita, K., Noguchi, Y., Konishi, T., & Itoh, Y. (2019). Application of Programming Learning Support System to Object-Oriented Language. *Proceedings of the 26th International Conference on Computers in Education*, 348-350.

Sorva, J., Karavirta, V., & Malmi, L. (2013). A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Transactions on Computing Education (TOCE), 13*(4), 15.