Journal of Software Engineering and Applications, **2012, 5, 816-822**
http://dx.doi.org/10.4236/jsea.2012.510094 Published Online October 2012 (http://www.SciRP.org/journal/jsea)

The Effects of Objects-First and Objects-Late Methods on Achievements of OOP Learners

Murat Pasa Uysal

Rochester Institute of Technology, Saunders College of Business, Department of MIS and Decision Sciences, Rochester, USA.
Email: muysal@saunders.rit.edu, mpuysal@gmail.com

Received August 15th, 2012; revised September 17th, 2012; accepted September 29th, 2012

ABSTRACT

Our research explored the effects of objects-first and objects-late methods on achievements of object-oriented programming (OOP) learners during a graduate course. The course's scope was virtually identical for two groups, but the structure of the contents differed in sequence. The objects-first method emphasized the design and discussion of the object-oriented concepts from the very beginning while the objects-late deferred these concepts to the late lectures. The objects-first learners used all visual functionalities of BlueJ IDE. However, the objects-late learners started with only the text-based interfaces of BlueJ and they benefited its visual support in the last lectures. At the end of the study, we found that there was a statistically significant difference between OOP learner groups.

Keywords: Object-Oriented Programming; Objects-First; Objects-Late

1. Introduction

It is fair to say that object-oriented programming (OOP) has gained much interest while it is becoming a common programming paradigm. This is partly because it is similar to our view of the real world. However, it has been difficult to introduce OOP to novice programmers [1-4]. The instructional methods, tools and the programming languages constitute the intrinsic or external factors for learning programming and they may greatly determine the complexity of this paradigm [5,6]. Consequently, teaching and learning OOP have been sources of many studies since they are challenging not only for students, but also for instructors. The studies are continually evolving in the hope of finding better tools, methods or innovative strategies. The literature review of teaching OOP suggests that the research studies conducted on teaching OOP can be broadly divided into three categories, as 1) Programming or educational tools for OOP learners; 2) Instructional approaches to teaching OOP; and 3) Learners' characteristics, conceptions and attitudes to OOP.

The instructional methods, objects-first and objects-late, are two of the important topics centering in the research area of teaching OOP. The primary reason would be the debates that are continuing whether introducing the objects and classes to the students at the very beginning is the right decision [7]. Because, there are many pedagogical dimensions involved in the issues, and we need to classify and clarify these effecting parameters carefully [8]. It seems that instructional effectiveness of these methods will continue being a subject to many discussions and more empirical evidence is needed to reach reliable conclusions. In this study, we compared two instructional methods for teaching OOP, which are the objects-first and the objects-late method. The next sections present this study along with the findings and discussion.

2. Tools and Methods for Teaching OOP

The studies on educational tools of OOP include instructional environments, such as programming micro-worlds, and the other tools for enhancement of current programming environments. These instructional tools generally differ in their support for adopted OOP paradigm, user interface, interactivity and visualization. As students' readiness and the knowledge level have important effects on learning, a special notice has to be given to the needs of novice learners when deciding on instructional environment [9-11]. Within environment, the ease of use should be provided by a graphical user interface that enhances editing, compilation and debugging options. The visualization can be supplied to aid thinking in terms of classes and object behaviors while allowing the use of both graphical and textual representations. The fundamental problem of existing programming environments is that they are not completely object-oriented, they are complex and they focus mainly on user interfaces [12]. Therefore, it is expected from an OOP learning environment that it meets the needs such as ease of use, support for code reuse, object-support, learning support, and finally group support [13]. BlueJ, DrJava, JKarelRobot, JCreater LE and Alice are some of the examples of educational environments used in the literature [10,13-15].

The other determining factor for teaching OOP is the instructional approach that has a close relationship with the programming paradigms. The way of structuring or organizing the programming tasks performed on a computer determines these paradigms. For example, imperative and declarative programming has found an important place in a wide range of applications. With the increased popularity of OOP paradigm, objects-first and objects-late instructional approaches have been attracting much interest among OOP researchers. The objects-first method puts emphasis on both design and programming principles of OOP from the very beginning. However, the objects-late method initially starts with non-object-oriented concepts, such as the statements and the control structures, and it defers the discussion of classes and objects to later lectures. Additionally, the conceptual approach, the apprentice approach, the model-first approach and the hybrid methodologies could be given as the other methods to teach OOP [16-18].

3. Method

The goal of this study was to find the effects of objects-first and objects-late methods on achievements of OOP learners. The academic performance constituted the dependent variable of our research design. The hypothesis was:

Hypothesis: The achievements of students will not differ significantly according to the OOP teaching methods.

We carried out the study with two experimental groups. Both the participants in group-1 and group-2 used BlueJ IDE. This was to eliminate possible effects of different instructional tools. The students were pre-tested and post-tested, and group-2 was administered as the treatment. The pre-tests and post-tests were given as open-ended academic achievement tests to determine whether the participants (**Table 1**) acquired the OOP knowledge and skills.

3.1. Subjects

The twenty male graduate OOP learners participated to this study. The students previously received a BS degree in systems engineering. They knew structured programming with Java at an introductory level and they previously used JCreater LE as a software development tool.

Table 1. The representation of the research design.

Group	Pre-test	Method	Tool	Post-test
Group-1	X	Objects-late	BlueJ	X
Group-2	X	Objects-first	BlueJ	X

3.2. OOP Environments

When designing the OOP instructional environments, it is important to give a special notice for choosing a programming environment. In general, professional software development tools are not suited for introducing learners to OOP. They are optimized for rapid application development as well as the needs of software professionals. Therefore, we considered mostly the pedagogical issues when deciding on the software development tool for this study. There are specific requirements for a first year object-oriented teaching. First, the basic concepts of OOP should be presented with simple and well-defined instructional strategies to teach in a consisted and understandable environment. Second, programming tool should display object-orientation by adopting a basic abstraction approach. Third, it should have an understandable execution model while avoiding the concepts that would possibly lead to erroneous programs. Finally, development environment should be easy to use with a debugger so that students can focus on learning OOP concepts rather than the environment itself [3].

BlueJ is the programming tool chosen for this study. It is an IDE especially for introductory teaching of OOP. It is hypothesized that OOP is not intrinsically complex, but it is made more complicated by a lack of appropriate tools [13]. Commercial tools mainly focus on user interfaces and they are too complex for the beginners. However, BlueJ's environment facilitates the discussion of object-oriented design by allowing interaction with objects before implementing them. It helps adopting a true objects-first approach while introducing the classes, objects and methods concepts before talking about Java and its syntax. The direct interactivity with classes and objects, the mechanisms allowing sophisticated and detailed testing of classes, and finally the visualization of object-oriented (OO) concepts could be given as the reason for preferring it as an objects-first teaching tool. A novice programmer using BlueJ is not only able to execute a complete application, but also she can directly interact with the objects of any classes without writing a complete application. Furthermore, the simple debugging mechanism showing the internal states of objects allows a quick and easy understanding of the Java code.

The unique nature of BlueJ is its user interfaces supporting a greater degree of interaction than the other OOP environments. Most importantly for our study, it allowed us to adopt two different OOP instructional approaches by enabling both conventional text-based and UML like programming interfaces.

3.3. Procedure

The students participated in a graduate course aiming to introduce OOP to the students with little prior knowledge.

The fundamentals of OOP concepts were given in this course. It consisted of two-hour lecture and two-hour lab per week, with 13 weeks per semester. The same instructor taught the students in group-1 and group-2 on different days. The expected learning outcomes were:

The participants should

- Be able to define the basic concepts of OOP: classes, objects and methods;
- Develop skills for reading, writing and debugging OO programs;
- Understand statements, iterative and conditional structures in Java language;
- Be able to write event-based interactive programs;
- Be able to apply the abstraction, inheritance, polymorphism and encapsulation concepts to their programs.

The course's scope was virtually identical for two experimental groups, but the structure of the contents differed in their sequence (**Tables 2** and **3**). The contents were designed and grouped into the units so that they were consisted with either of the OOP teaching methods. The two-hour lecture included necessary programming topics, OOP concepts and illustration of sample programs. The weekly exercises and the lab studies were different in both presentations and example applications. Thus, instructional activities were tailored to each of the group's Java programming interfaces.

Table 2. The sequence and structure of the course topics for objects-late method.

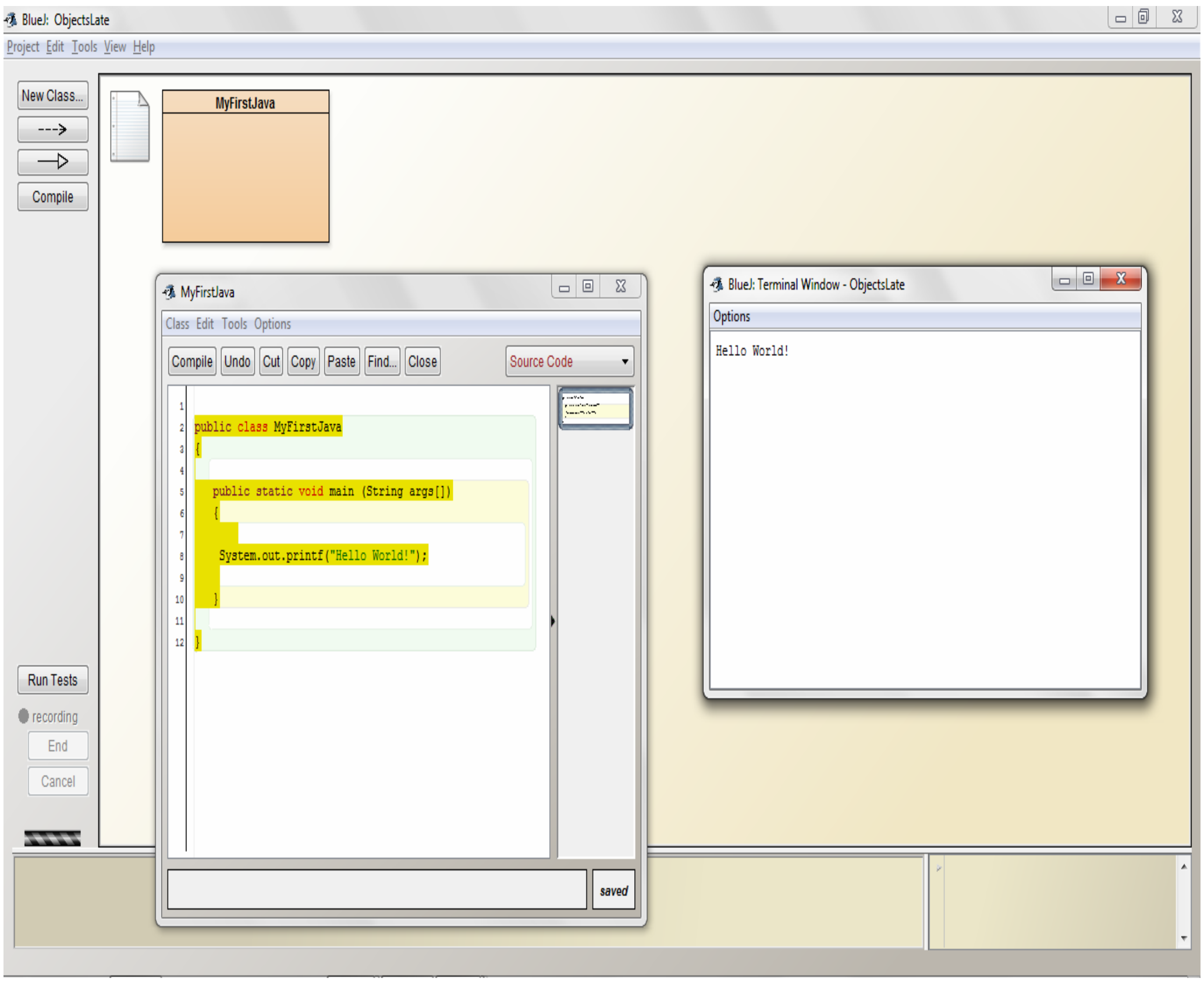
Weeks	The objects-late lecture topics (group-1)
1	Introduction of BlueJ IDE, representation of class, methods and attributes at an introductory level.
2	Data types, variables, constants. Assignment statements. Referencing to object and class concepts.
3	The use of operators and conditional statements with sample applications.
4	Loops and iteration structures with sample applications.
5	Editing and modifying existing classes. Creating new classes with members & methods, observing object behaviors and states with sample applications.
6	Event handling in Java applications.
7	Working with graphics (*i.e.* drawing lines, rectangle, and circle). Introducing the AWT.
8	The "Shapes" project: creating single classes of circles, squares and triangles which can be moved, resized and changed in color.
9	Introducing abstraction, inheritance, polymorphism concepts with basic UML diagrams
10	The design principles of a large project, interaction of multiple objects to perform a complete programming task.
11	The "Picture" project: creating a picture class combining various shapes to draw a picture.
12	Introduction to software engineering, object-oriented design principles, software development processes.
13	The post-test.

Table 3. The sequence and structure of the course topics for objects-first method.

Weeks	The objects-first lecture topics (group-2)
1	Introduction to BlueJ environment, OOP paradigm, object, class and method concepts.
2	Objects, classes, methods, parameters, fields, object states, presentation and application of these concepts in BlueJ environment.
3	Assignment statements, constructors, data types, presentation and application in BlueJ environment.
4	Programming, debugging and testing principles in BlueJ environment.
5	Abstraction, inheritance, polymorphism concepts, applications in BlueJ environment.
6	Basic object-oriented design principles with UML diagrams.
7	The presentation of operators, conditional statements, loops and iterations in BlueJ with simple examples.
8	Editing and modifying existing classes. Creating new classes with members & methods, and observing object behaviors and states in sample applications.
9	The "Shapes" project: creating single classes of circles, squares and triangles, which can be moved, resized and changed in color.
10	The design principles of a large project, interaction of multiple objects to perform a complete programming task.
11	The "Picture" project: creating a picture class combining various shapes to draw simple pictures.
12	Introduction to software engineering, software development processes and object-oriented design principles.
13	The post-test.

The objects-late method initially followed the classical instructional pattern, which was similar to that in most of the structured Java programming courses (**Table 2**). The participants started using BlueJ's text-based interface. Their first program was about a class with the "main" method saying "Hello World" (**Figure 1**). The instructor specially noticed that the group-1 learners gave their focus to the text-based editor rather than graphical representations. Therefore, the students directly started developing stand-alone applications in BlueJ instead of using the UML like representations of BlueJ.

During the first 7 weeks of lecture and lab hours, group-1 participants followed the same sequence when developing applications. This was: (1st) they opened BlueJ IDE; (2nd) from the "Edit" menu, they used the "New Class" or "Add Class From File" menu options for a start of application; (3rd) they opened the editor window to edit their code; (4th) they cleaned the existing code added their "main method" or they exercised the given application code added from a file. The instructor initially described the Java programming by its syntax rules and semantics. The arrangements of the words and punctuations were presented in a framework syntax while the meaning of these rules given in a form of cause-effect



```
public class MyFirstJava
{

  public static void main (String args[])
  {

   System.out.printf("Hello World!");

  }

}
```

```
Hello World!
```

Figure 1. BlueJ interfaces used by objects-late learners.

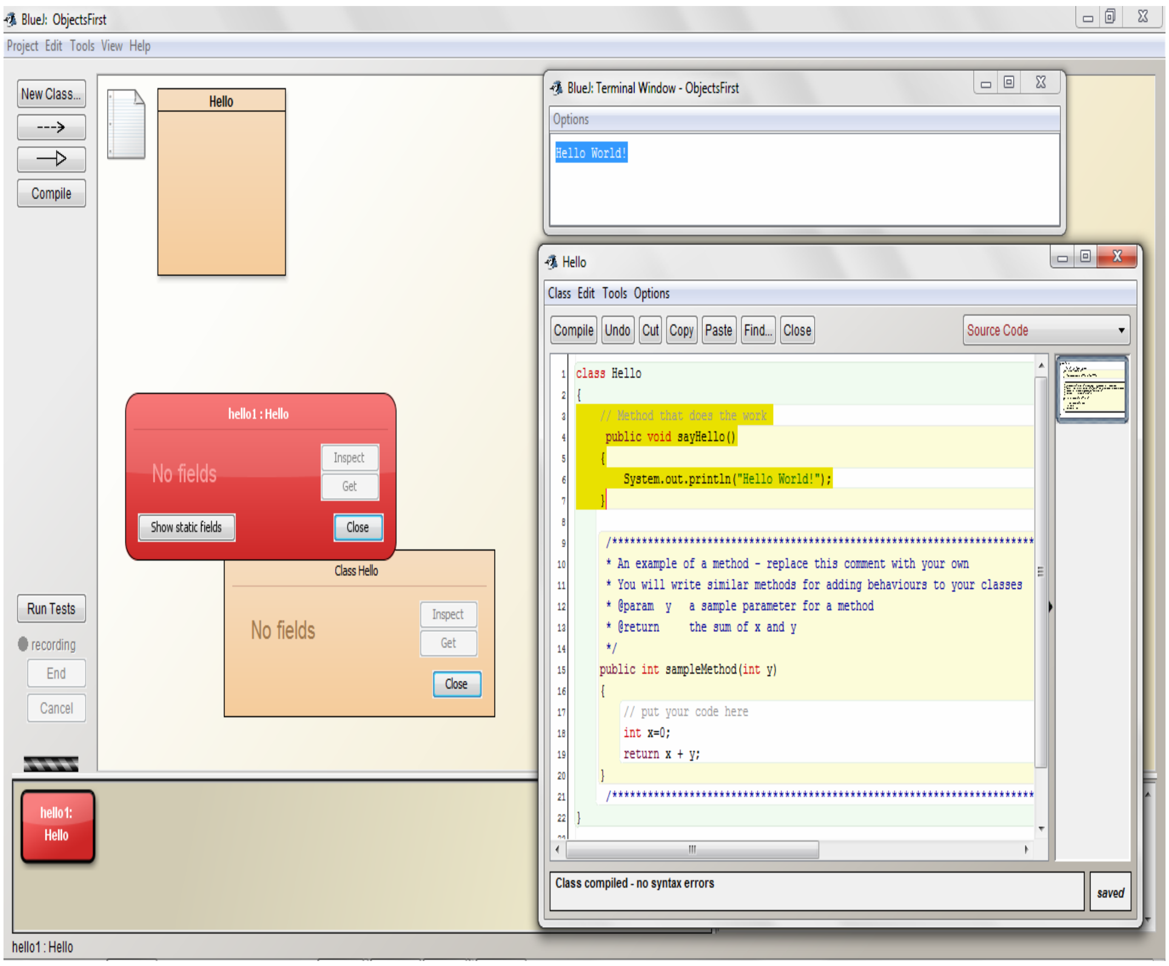
relationships. The participants formed the basic programming constructs such as variables, data types and statements. Later, the selection and iterative control structures were given in the next contents. The lectures also covered the issues of writing clear and understandable codes as well as testing and debugging techniques. The students developed their simple programs during the lab hours. Since the classes are central for data and behaviors in Java language, these constructs were illustrated briefly with in the framework of class definitions. The core OOP concepts such as inheritance and polymorphism were deferred to the late in weekly lectures. The group-1 students benefited BlueJ's visual support for OO design and the support for interaction with objects in the last weeks (8 - 13 weeks).

When teaching programming with the objects-first method, the classes and objects were introduced without going into the further details of Java syntax rules. The participants directly started by creating objects and executing methods of the previously developed sample projects to conceptualize classes, objects and methods (**Figure 2**).

The participants were also expected to learn simple forms of statements, control structures, method implementations and creation of source codes to define object behaviors and to observe their states. Later, they experienced editing, compiling and executing processes while gaining an insight into Java programming. The presentation of the core OOP concepts, such as inheritance, polymorphism, coupling and cohesion concepts were introduced to the group-2 students. Finally, the interaction of multiple objects was presented to these students when invoking each other's methods (**Table 3**).

3.4. Data Analysis

The pre-tests and the post-tests were given to determine whether the OOP knowledge and skills were acquired. Before starting to this study, the experienced OOP instructors evaluated these tests for the content and face validity. The students were required to give the correct descriptions of OOP concepts and to write a complete Java application during the achievement tests. This was



```
Hello World!
```

```
class Hello
{
    // Method that does the work
     public void sayHello()
    {
       System.out.println("Hello World!");
    }

     /*********************************************************************
     * An example of a method - replace this comment with your own
     * You will write similar methods for adding behaviours to your classes
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
       // put your code here
       int x=0;
       return x + y;
    }
     /*********************************************************************
}
```

```
Class compiled - no syntax errors
```

Figure 2. BlueJ interfaces used by objects-first learners.

for the intention of measuring their understanding and OOP skills of students whether they met the learning outcomes. The participants were graded based on the concepts they correctly defined and the Java applications that they developed. We carried out the statistical analysis with the SPSS v.16 software.

3.5. Findings

Table 4 presents the descriptive statistical data of the study.

The null hypothesis stated that no significant statistical difference would exist between the test scores of students in two different groups. However, as it is seen in **Table 5**, there is a statistically significant difference between these experimental groups and we reject the null hypothesis ($z = -2.290$, $p < 0.05$). It is possible to say that the learners instructed with objects-first method achieved higher learning outcomes.

4. Discussion and Conclusions

The objects-first instructional design emphasized the design and discussion of the OO concepts from the very beginning while the objects-late deferred these concepts to the late lectures. The complex notions of OOP, such as the inheritance, class & object, data & behavior and the relationships among the concepts were presented with the visual interfaces of BlueJ. As having different forms of visualization, the representations of OOP concepts possibly helped participants learn and retain the knowledge in the long-term memory. The learners were easily able to derive more semantic knowledge from visual displays than text-based representations. We observed that comprehensible visual representations in the lectures directed the learners' attention to the relevant information rather than confusing programming details. Thus, this method guided the learners of group-2 while they were identifying the perceptual cues associated with the OOP concepts. The objects-first method had a dual nature that depicted OOP concepts while at the same time relying on the fundamental instructional abstractions. The concrete elements of Java programming were expressed with these general and abstract terms, which enabled learners to move away from a conventional environment to a pictorial and functional

Table 4. The descriptive data of the participants.

Learner No.	Group No.	Pre-test			Post-test		
		Concepts 60%	Program 40%	Total 100%	Concepts 60%	Program 40%	Total 100%
1	2	23	8	31	53	27	80
2	2	53	26	69	55	39	94
3	2	29	20	49	41	30	71
4	1	43	19	62	43	33	76
5	1	41	11	52	53	38	91
6	2	38	16	54	49	34	83
7	1	31	13	44	47	30	77
8	2	45	15	60	50	40	90
9	2	49	25	74	52	40	92
10	1	43	19	62	50	40	90
11	1	41	4	45	45	33	78
12	1	53	30	83	56	31	87
13	2	58	25	73	50	30	80
14	1	50	15	65	60	30	90
15	1	44	26	70	55	40	95
16	2	55	20	75	54	39	93
17	2	45	10	55	53	40	93
18	1	32	28	60	40	37	87
19	2	49	11	60	60	30	90
20	1	32	20	52	49	25	74

Table 5. The Mann-Whitney test results of the academic achievements.

Group	*n*	Mean rank	Sum of ranks	*z*	*p*
Group-1 (objects-late)	10	6.89	62.00	–2.290	0.022
Group-2 (object-first)	10	12.80	128.00		

environment. This eased the application of basic programming skills. Therefore, high achievements of the learners in group-2 could be attributed to instructional appropriateness of the objects-first method.

Initially allowing the group-2 students to interact with objects and methods possibility formed the concrete understanding of the inner workings of OOP paradigm. Therefore, objects-first strategy, which is supplemented with visual interfaces, made the information apprehensible, apparent and visible to the learners [19,20]. We observed that the learners in group-2 could easily find, design or interpret the relevant information pertaining to programming tasks. Any visual software development environment should provide necessary abstractions. Therefore, the objects-first strategy is thought have provided the learners with appropriate conceptual models, and it enabled them to incorporate abstraction details with concrete programming facts and rules. This strategy also reflected the state, behavior and existing attributes of an entity that an object encapsulated. We observed that the students in group-2 were easily able to explore the OOP concepts and to carry out programming tasks in visual and text-based environments, which also directly supported the objects-first strategy.

During the study, the nature of the objects-first method supported visual design of programs while maintaining access to the text editor. This gave different perspectives to the learners of group-2. Software development requires fundamental skills *i.e.* analysis, design and coding. In terms visualization, different tasks would require different types of representations suited for the learners' preferences. While visual environments could be suitable for the analysis and design tasks of software development phases, text-based environments would be more appealing to traditional programmers. The learners in group-2 felt comfortable in this two-dimensional visual environment through out programming. Therefore, the two-mode of objects-first method (visual and text) enabled the functionalities that the learners needed. As a result, learners were able to switch among representations while course topics were presented with the objects-first strategies.

The learners in group-2 found objects-first method enjoyable since it allowed more freedom and layout opportunities in terms of instructional activities. From experienced to novice ones, the majority of the programmers find a number of advantages of graphical representations over the text-based ones. While the overview of programming context is provided, software structures are made more visible and clearer in objects-first environments. We know that these environments have a higher level of abstractions, they are easier and faster to understand, and they provide closer mapping of programming domain. In our study, the objects-first strategy helped the learners understand what the class or object mean and grasp the related OOP concepts. It is possible to say that the objects-first instructional method, which was rich in visualization and instructional activities, played an important role in high-level achievements of learners in group-2.

Learning OOP is a difficult task, especially for the beginners. It requires internalization of OOP paradigm and adapting the basic skills to the programming tasks. We believe that, interacting with objects from the beginning helped the learners build their concrete understanding and it provided appropriate conceptual models. The objects-first learners could easily put a mental emphasis on the concepts of class and object relationships instead of concentrating on structural programming facts. We are of the opinion that the debates on the OOP teaching methods will continue in the future. Therefore, our paper concludes with an invitation of more studies on instructional methods for OOP.

REFERENCES

[1] M. Ben-Ari, "Constructivism in Computer Science Education," *Proceedings of the* 29*th Technical Symposium on Computer Science Education*, New York, 26 February-1 March 1998, pp. 257-261.

[2] A. E. Fleury, "Student Conceptions of Object-Oriented Programming in Java," *The Journal of Computing in Small Colleges*, Vol. 15, No. 1, 1999, pp. 69-78.

[3] M. Kolling, "The Problem of Teaching Object-Oriented Programming, Part 1: Languages," *Journal of Object-Oriented Programming*, Vol. 11, No. 8, 1999, pp. 8-15.

[4] A. Eckerdal, "Novice Students' Learning of Object-Oriented Programming," Unpublished Doctoral Dissertation, The Uppsala University, Uppsala, 2006.

[5] M. Kolling and J. Rosenberg, "Guidelines for Teaching Object Orientation with Java," *Proceedings of the* 6*th Annual Conference on Innovation and Technology in Computer Science Education*, Canterbury, 25-27 June 2001, pp. 33-36.

[6] H. Zhu and M. Zhou, "Methodology First and Language Second: A Way to Teach Object-Oriented Programming," *OOPSLA* 2003: *Companion of the* 18*th Annual ACM Conference on Object-Oriented Programming*, *Systems*, *Languages*, *and Applications*, New York, 26-30 October 2003, pp. 140-147.

[7] K. B. Bruce, "Controversy on How to Teach CS 1: A Discussion on the SIGCSE-Members Mailing List," *Inroads—The SIGCSE Bulletin*, Vol. 37, No. 2, 2005, pp. 111-117.

[8] A. Ehlert and C. Schulte, "Empirical Comparison of Objects-First and Objects-Later," *Proceedings of the* 5*th International Workshop on Computing Education Research Workshop*, Berkeley, 10-11 August 2009, pp. 15-26. doi:10.1145/1584322.1584326

[9] R. B. Findler, C. Flanagan, M. Flatt, S. Krishnamurthi and M. Felleisen, "Drscheme: A Pedagogic Programming Environment for Scheme," *Programming Languages*: *Implementations*, *Logics*, *and Programs*, Vol. 12, No. 2, 2002, pp. 159-182. doi:10.1007/BFb0033856

[10] E. Allen, R. Cartwright and B. Stoler, "DrJava: A Light-Weight Pedagogic Environment for Java," *Proceedings of the* 33*rd Technical Symposium on Computer Science Education*, New York, 26 February-2 March 2002, pp. 137-141.

[11] S. Georgantaki and S. Retalis, "Using Educational Tools for Teaching Object Oriented Design and Programming," *Journal of Information Technology Impact*, Vol. 7, No. 2, 2007, pp. 111-130.

[12] A. Patterson, M. Kölling, B. Quig and J. Rosenberg, "The BlueJ System and Its Pedagogy," *Journal of Computer Science Education*, Vol. 13, No. 4, 2003, pp. 13-24.

[13] M. Kolling, "The Problem of Teaching Object-Oriented Programming, Part 2: Environments," *Journal of Object-Oriented Programming*, Vol. 11, No. 9, 1999, pp. 6-12.

[14] D. Buck and D. J. Stucki, "JKarel Robot: A Case Study in Supporting Levels of Cognitive Development in Computer Science Curriculum," *ACM SICGSE Bulletin*, Vol. 33, No. 1, 2000, pp. 16-20.

[15] S. Cooper, W. Dann and R. Paush, "Alice: A 3-D Tool for Introductory Programming Concepts," *Journal of Computing in Small Colleges*, Vol. 15, No. 5, 2000, pp. 108-117.

[16] D. J. Barnes and M. Kölling, "Objects First with Java: A Practical Introduction Using BlueJ," 2nd Edition, Prentice Hall, New Jersey, 2005.

[17] M. Torgersen, H. R. Kresten and K. Thorup, "A Conceptual Approach to Teaching Object-Orientation to C Programmers," *The Proceedings of Educators*' *Symposium*: *Conference on Object-Oriented Programming Systems*, *Languages and Applications*, Vancouver, 18-22 October 1998, pp. 3-10.

[18] J. Bennedsen and M. Caspersen, "Programming in Context: A Model-First Approach to CS1," *Proceedings of the* 35*th SIGCSE Technical Symposium on Computer Science Education*, Norfolk, 3-7 March 2004, pp. 477-481.

[19] S. Xinogalos, "Guidelines for Designing and Teaching an Effective Object-Oriented Design and Programming Course," *Advance Learning*, 2010, pp. 397-422.

[20] M. Petre, "Mental Imagery and Software Visualization in High-Performance Software Development Teams," *Journal of Visual Languages and Computing*, Vol. 21, No. 3, 2010, pp. 171-183. doi:10.1016/j.jvlc.2009.11.001