Teaching OOP with BlueJ: A Case Study

Stelios Xinogalos[(*)], Maya Sartatzemi[(*)], Vassilios Dagdilelis[(**)], Georgios Evangelidis[(*)]
[(*)]Department of Applied Informatics, [(**)]Department of Educational and Social Policy
University of Macedonia, Thessaloniki, Greece
{stelios, maya, dagdil, gevan}@uom.gr

Abstract

In this paper we present our findings on teaching OOP with BlueJ in the context of a one-semester programming course. We organize our findings, i.e., the difficulties, the errors, and the misconceptions that students encounter, in two categories: (a) difficulties attributed to the special characteristics of OOP, and, (b) difficulties that may be attributed to the features of the programming environment.

1. Introduction

The last 15 years the object-oriented programming paradigm is taught in many university departments, either in the context of an introductory programming course or as a subsequent programming course. This development resulted as an attempt to make use of the advantages of OOP, which lie in the fact that it supports the programming concepts instructors were struggling to teach for so many years, such as structured programming, modularization and program design, as well as techniques for solving problems, such as program development in groups and code reusability. Even though the university community, as well as the industry adopted with exceptional optimism OOP as a very good tool for teaching novices the programming methodologies, it finally appears that teaching OOP is particularly difficult, as shown in relative studies. The main difficulties result from:

(a) The lack of programming environments appropriately designed for teaching OOP in general, and for novice programmers more specifically.

(b) The transition from the procedural programming paradigm to the OOP paradigm.

(c) The teaching philosophy that is based on the approach followed for teaching procedural programming. This means that concepts of OOP, such as classes, objects, inheritance, and so on, are taught after the basic elements of a programming language, the control structures and functions.

In order to deal with the difficulties mentioned above, various efforts have been made that resulted in a) special programming environments [6], such as BlueJ, DrJava, jGRASP, b) mini programming languages [6], such as Karel++, that were supplemented by the development of programming microworlds (objectKarel, JKarel), and c) books or generally guidelines or series of lessons on how OOP can be taught effectively. A special effort is that of the creators of BlueJ, who, in parallel to the environment they developed, tried to review some basic guidelines for teaching OOP in their paper "Guidelines for Teaching Object Orientation with Java" [3]. In addition, their book "Objects First with Java: A practical introduction using BlueJ" [1], attempts to teach OOP according to the eight guidelines mentioned in [3]. The environment of BlueJ, clearly constitutes an environment that makes it easier for novices to learn OOP.

This paper[1] attempts to study to what degree the teaching proposal of Kölling & Barnes is effective, that is teaching OOP with the help of BlueJ and the lessons proposed in their book.

2. Description of the lessons and the main findings of the study

We present the results we obtained from teaching OOP in the context of a one-semester programming course at the department of Management of Technology at the University of Macedonia. The teaching of the course, as we have already mentioned, was based on the series of lessons suggested by Kölling and Barnes at their book *"Objects First with Java: A practical introduction using BlueJ"* [1] and the programming environment used was BlueJ.

[1]This research is being funded by the Greek Ministry of Education and the European Union as part of the project "*Pythagoras όό-Funding of research groups in the University of Macedonia*"

At this university department OOP is taught in the 3rd semester. Students had been taught the main concepts of programming in the 2nd semester with C as a programming language, and in the final exams a high failure rate was recorded. The course of the 3rd semester consisted of a weekly two-hour theory session, where the instructor presented the corresponding concepts with the use of slides, and a weekly two-hour laboratory session, where students usually solved in cooperation with the instructor or by themselves some assignments. Eleven theory lessons and eleven laboratory lessons were given. Approximately 45 students attended the lessons.

The lessons that we planned with BlueJ, which were based on the didactic path suggested by the book, seem to be effective as a whole. More specifically, the answers given at the questionnaires and the solutions at the suggested problems (solved either in the lab or at home), show that, in general, students comprehend basic concepts of OOP. However, we consider some of the problems students meet in the process of problem solving as important, and so we present them in the following paragraphs.

2.1 Difficulties attributed to the special characteristics of OOP

As most of the studies about the teaching of OOP show, students confuse some of the language's elements. Furthermore, they face difficulties in using other elements of the language for implementing solutions to given problems. Next, we present the difficulties recorded in our study:

- Some students, as Holland et al. [2] state, confuse classes with objects. Furthermore, we observed, in a much smaller scale, confusion between classes and methods. Similarities in the identifiers of entities, that otherwise have concrete roles (such as NumberofNotes, NoteNumbers for example), seem to play an important role in this confusion.
- Although students seem to comprehend the concept of object collections, they find it difficult to use flexible size collections for grouping objects (such as ArrayList), and even fixed size collections (such as arrays). These difficulties refer to using object collections for implementing solutions in general, and not to errors about the boundaries of an array [5] or using for the first element of an array/collection the index 1 instead of 0 [4].
- Although most of the students seem to comprehend a situation where multilevel inheritance is used, a significant number of them find it difficult to comprehend the way multiple inheritance is implemented in Java. For example, they believe that a concrete class can extend more than one concrete classes that are not related to each other.
- Some students believe that abstract classes, and not interfaces, can contain only abstract methods. A greater number of students believe that both abstract classes and interfaces can contain only abstract methods.
- A significant number of students misinterpret the information supplied from class diagrams. For example, students interpret a 'uses relation' depicted in a class diagram as a situation where 'the class that is shown to use another class is the only class that has access to its methods'.
- Weakness in describing the function/role of classes, methods and fields. Greater difficulty was observed in describing the role of a class field that stored a reference to an object of another class, and specifically the class ArrayList, which belongs to the Standard Class Library of Java. Although we expected that the environment of BlueJ would make it easier for students to comprehend what is stored in each field through the visualization provided (inspect feature), it seems that when a field of a class stores a reference to an object of another class and not a value of a primitive type, even the graphical representation does not easily lead to an accurate mental model of the object.
- Wrong use of method calls and dot notation was observed. Specifically, in a class where both internal and external method calls should be used, some students used in both cases an external method call: <object>.<method name>.

2.2 Difficulties that may be attributed to the features of the environment

BlueJ is an environment that is developed exclusively for helping students that are taught OOP. However, the use of BlueJ, even in combination with the activities suggested by its creators, is not fully freed of problems that arise, maybe in an indirect way, from the use of the environment itself.

A category of difficulties seems to have its roots in the importance given by the authors to the graphical representations of the structure of a project and its elements (classes, methods, objects), in relation to the code, and also to the extended use of visualization for editing various projects – which students were asked to extend, refactor or even explain. The fact that these features exist and constitute a main element of the didactic rationale of the creators of BlueJ, benefits greatly the development of lessons that use systematically these features. However, in many cases

we observed that students faced difficulties in extracting information from the code of a program – in contrast they easily extracted information from the diagrams created by BlueJ. Also, in assignments where students had to develop new code, we observed higher rates of wrong answers (or no answers at all), which may be partly attributed to the lack of balance between the diagrammatic representations and the textual code.

The creators of BlueJ pay much attention to the comprehension of the object-oriented technique to designing applications, and so the environment allows even the compilation and execution of pieces of a project, without the need of a main method. Also, through its interface, it allows the construction of objects by just clicking – the corresponding declarations of objects are created automatically and "silently" from the system itself. Even though this technique saves plenty of time, it benefits the development of misconceptions about the way the language functions.

The creators of BlueJ pay much attention in taking advantage of existing information for solving specific problems. For example, for many of the instructor's questions the full answers exist in parts of the existing code. This didactic technique saves time too, but at the same time it creates a tendency to click-oriented answers, which means that it reinforces the tendency of many students to select a piece of code without thinking if this is the most appropriate. Many times they include together with the correct answer irrelevant pieces of code – which are just wrongly selected with the mouse. So, the answer to a question is more a product of automation that is based on a quick search of the appropriate part from the available sources (code), rather than a product of thinking on the content of the question.

In some cases we also observed that students copied pieces of code from other projects, just because there were some similarities in the corresponding methods. Of course, this tendency does not result from the features of the BlueJ environment itself. It seems to be a consequence of a teaching style that is based heavily on a very helpful interface and the obscure, for the user, development of code by the system itself.

3. Conclusions

The BlueJ environment has some features that respond to the basic principles for an introductory teaching of OOP – as its creators describe them. Also, the lessons of the book are organized in a way that responds to these principles. What we must point out is that student performance for the problems they were assigned was satisfactory, but specific kinds of activities are benefited from these kinds of problems, in contrast with others. We believe that the problems we presented in Section 2 and which we attribute to the BlueJ environment and the series of proposed lessons can be decreased, if not extinguished, by taking into account the following:

- Use of the test class and consequently of the main method much earlier, so as to give the opportunity to students to declare and then construct objects by writing code and not only through the indirect use of icons.
- Improvement of the editor of BlueJ by adding features, such as auto-completion and bracket matching in order to reduce syntax errors.
- Development of small scale projects, not necessarily related to the projects of the book, so as to balance the tendency to click-oriented answers, which reinforces the tendency of many students to select a piece of code, without thinking if this is the most appropriate one.
- More time and emphasis must be given on practicing with internal and external method calls from a class. In order to do this, two supplementary activities can be used. First, the BlueJ environment can help since, through the direct method calls, it can display the statement that contains a call to a method. Furthermore, this activity (direct manipulation of objects for method calling) must be combined with the opposite activity, which means that students must explicitly write the relevant code without making use of the visualization features of BlueJ, so as to give them the opportunity to think and act consciously, beyond their mechanical click-oriented type of behavior.

4. References

[1] Barnes, D. & Kölling, M., Objects First with Java: A practical introduction using BlueJ, Prentice Hall, 2004.

[2] Holland, S. Griffiths, R. & Woodman, M., "Avoiding object misconceptions", ACM SIGCSE Bulletin, Vol. 29, No. 1, 1997, pp. 131-134.

[3] Kölling, M. & Rosenberg, J., "Guidelines for Teaching Object Orientation with Java", ACM SIGCSE Bulletin, Vol. 33 Issue 3, 2001, pp.33-36.

[4] Taylor K., Common Java Coding Errors, http://java.about.com/cs/beginningjava/a/comm_errs.htm, 2005.

[5] Topor, R. W., CIT1104 Programming II: Common (Java) programming errors, http://www.cit.gu.edu.au/~rwt/p2.02.1/errors.html, 2002.

[6] Xinogalos S., Satratzemi M., "An Integrated Programming Environment for Teaching the Object-Oriented Programming Paradigm", LNCS, 2510, 2002, 544-551.