OOPP: Tame the Design of Simple Object-Oriented Applications with Graphical Blocks

Alberto Ferrari, Gianfranco Lombardo, Monica Mordonini, Agostino Poggi, and Michele Tomaiuolo(✉)

Dipartimento di Ingegneria e Architettura, Università di Parma, Parma, Italy
{alberto.ferrari,gianfranco.lombardo,monica.mordonini,
agostino.poggi,michele.tomaiuolo}@unipr.it

Abstract. Many and varied experiences are being reported, about the first introduction to programming for young students and neophytes. However, tools and methodologies are needed also for a more comprehensive learning process, which requires to design the architecture of any small but functioning application. We propose a new environment, based on the use of graphical blocks, for designing some object-oriented applications. It merges the positive features of block-programming with the object-oriented paradigm in a graphical educational environment. It is developed as a tool for supporting the objects-early approach. The whole methodology is targeted at high school students, university freshmen and unemployed people who are motivated to learn to code professionally. In these cases, where we have firstly experimented this approach, the concepts of object-oriented programming (OOP) cannot be relegated to a secondary role, but they have to be introduced early and presented in their most intuitive form.

Keywords: Computer programming · Block programming
Object-oriented programming · Education

1 Introduction

Most recent reports, regarding the introduction to coding, target a young or a very young audience [4,17,20]. A feature shared by a number of these projects is the use of puzzle programming for simplifying the very first approach to computer programming, trying to eliminate the syntactic burden.

Our work is oriented in particular to programming courses for high school students and university freshmen. Moreover, there is a growing social pressure to reorient unemployed people towards computer programming, through vocational training courses. In all these cases, the typical syntactic difficulties related with coding are soon accompanied by some challenges of application design. In fact, courses for professional computer programming cannot procrastinate the concepts of object orientation, which become harder to learn at a later stage. Thus, we are working to define an approach which can get the best of both the

B. Guidi et al. (Eds.): GOODTECHS 2017, LNICST 233, pp. 279–288, 2018.
https://doi.org/10.1007/978-3-319-76111-4_28

worlds of object-oriented programming and puzzle programming. In this approach, which we call *Object-Oriented Puzzle Programming (OOPP)*, we propose a tool for the design and development of object-oriented applications, adopting the objects-early methodology [15]. Applications are realized by connecting visual objects representing the fundamental elements of object-oriented programming, in a similar manner to puzzle programming. The user/programmer realizes a puzzle of connected blocks, which represents the object-oriented structure of the application.

The rest of the article is organized in the following way. Section 2 presents the current discussion and the situation about the introduction to coding, especially for high school students and young workers. Section 3 discusses the methodology and tool we propose for Object Oriented Puzzle Programming. Section 4 presents an evaluation of the tool by a class of post-high school students. Finally, some concluding remarks are presented.

2 Related Works

The whole idea of Computational Thinking was popularized by Wing [21], as "*a way of solving problems, designing systems, and understanding human behavior that draws on concepts fundamental to computer science.*" Apart from coding, other aspects and concepts are fundamental to Computational Thinking [6,7], including abstraction and modeling.

The founding idea of block programming, which characterize a number of projects [4,16,22] related to the introduction to Computational Thinking, is to provide a graphical interface with blocks of diverse types. The user/programmer can combine the blocks in various ways through drag and drop operations, in such a way to form a puzzle representing the solution to a given problem. The blocks represent the basic elements of the language and they are differentiated by form and color, to ease their identification and usage. The difficulties, faced by novices which approach programming for the first time [1,3,9], are essentially of two different kinds: on the one hand it is necessary to formalize the ideas about the solution in the form of an algorithm, on the other hand it is necessary to adhere to the rigorous syntax of a programming language. Block programming eliminates the possibility to introduce syntactic errors and allows users to focus entirely on the logics of assigned problems and their solutions. In fact, the composition of blocks is rigidly constrained by existing slots, which represent the syntactic constraints of the language.

Although most educational projects and experiences proposing a block programming approach are oriented to a very young audience, projects directed toward high school students are beginning to appear. In [19] some points of strength and weakness are analyzed, as they are perceived by high school students which are allowed to move from a textual programming environment to a block-based one.

In the last decades, the object-oriented approach has become one of the most adopted programming paradigm [12,13]. It is popular both at the educational

and professional level. This fact is largely accepted, but there are still a lot of debates focusing on the time in which OOP is best introduced in CS1 courses. The debate about the "paradigm shift" is still actual, but in the last years in many introductory university courses there has been a move from the procedural paradigm to the object-oriented one. A comparative analysis [18] shows that student who have started their curriculum with an object-first approach obtain better results, when they have to design software for solving complex problems. Other studies [2,5,10] present experiences in which the object-first approach has effectively led to great improvements. They show that students obtain better overall results and the failure rate is drastically reduced. An additional advantage of the object-first approach is to facilitate the comprehension of the object-oriented paradigm, while the shift from a paradigm to another is quite difficult for those who started studying the discipline with an imperative/procedural approach. In [12] it is argued that "*it is the switch that is difficult, not object-orientation.*" If the main difficulty lies in the paradigm-shift, from the procedural paradigm to the object-oriented one, then the objects-early approach has the lowest difficulty level, as can be seen in many CS1 courses.

In introductory courses based on the object-early approach, professional tools can be too complex, especially for newbies. Among tools tailored for educational purposes, instead, BlueJ[1] provides a useful IDE for object-early didactics. However, unlike the tool we are going to present, there is little functionality for designing the internal structure of the classes of an object-oriented application.

3 Object Oriented Puzzle Programming

OOPP is an integrated development environment primarily designed for teaching purposes in order to introduce object-oriented design and programming in a simple and intuitive way. OOPP is a tool that allows a user/programmer to design and create simple object-oriented applications by block composition, and then automatically generate the target code that will be the frame of the application to be realized. It includes a workspace that allows you to define, in a visual way, "blocks" of code, with related methods, constructors and attributes, that will then automatically generate the corresponding target code.

In Blockly[2], we have developed a set of high-level blocks that can be used for object-oriented programming. The visual environment is developed entirely in JavaScript and therefore it can be executed within any web browser. This fact further simplifies the use of the application, that does not require any specific software installation. The environment is characterized by three main elements (Fig. 1):

1. a toolbox which contains the available blocks, organized by their type;
2. a work space where to place and link the blocks to form a puzzle/program;
3. a text box to display the corresponding generated code automatically, by converting each block of the puzzle into a sequence of target instructions.

[1] https://www.bluej.org/.
[2] https://developers.google.com/blockly/.

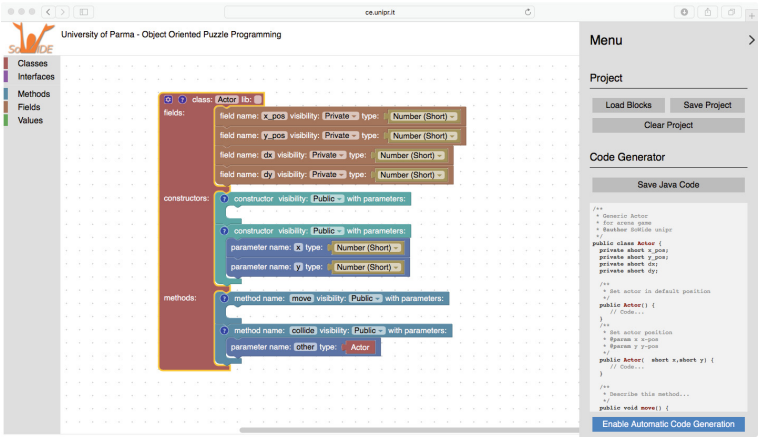

Fig. 1. Object-Oriented Puzzle Programming environment.

3.1 The Blockly Framework

Blockly includes some standard categories of blocks: logical blocks, cycles, math operators, lists, variables, and so on. Starting from these categories, we have created additional blocks for object-oriented programming: interfaces, classes, constructors, methods, and parameters. The new blocks are shaped to allow only connections that respect the syntactic constraints. Defining a new class or interface makes a new block in the toolbox instantly available, which is the new type of data that you can then use in developing the application. To keep the environment simple and intuitive, as much as possible, only the main features of object-oriented programming have been introduced. For example, it is not possible to explicitly define inner classes and abstract classes. With the present features, it is possible to put into practice the fundamental concepts of object-oriented programming such as encapsulation, inheritance, and polymorphism. Our didactic experience shows us that more advanced features, which can be found for example in the UML language specifications, are neither necessary nor desirable in such an introductory environment. OOPP is thought to be a tool for the initial stages of a didactic path for object-oriented design. Hence, it is unsuitable for complex applications. As the same authors of Blockly say: "*Blockly is currently designed for creating relatively small scripts... Please do not attempt to maintain the Linux kernel using Blockly.*" – Neil Fraser - Google.

Blockly is defined as "*a library for building visual programming editors*", that is a tool for application developers, and many are in fact the educational applications that use it as a starting point for development. It provides a workspace that allows users (novice programmers) to write programs by linking the various blocks. Each visual object (block) actually represents a code object. Blockly also allows to create new blocks. Then, these new blocks can be used in a variety of applications. In a sense, creating a set of new blocks corresponds to the creation of a new language with its new syntax and semantics. The syntax is defined by

the structure of the blocks, their color, their shape and the possible connections. In practice, Blockly allows to write only syntactically correct programs, with textual code replaced by visual blocks. The *Block Factory* (Fig. 2) is a graphical tool of Blockly, which can be used to start creating new blocks, by defining their main features through a simple and intuitive interface. Block semantics is then defined by providing the corresponding code for each of them, which can be written in any programming language. The double representation of the program (Fig. 1), both as a puzzle and as a textual code, is very interesting and stimulating in an educational setting. In fact, the novice programmer can compare the two versions.

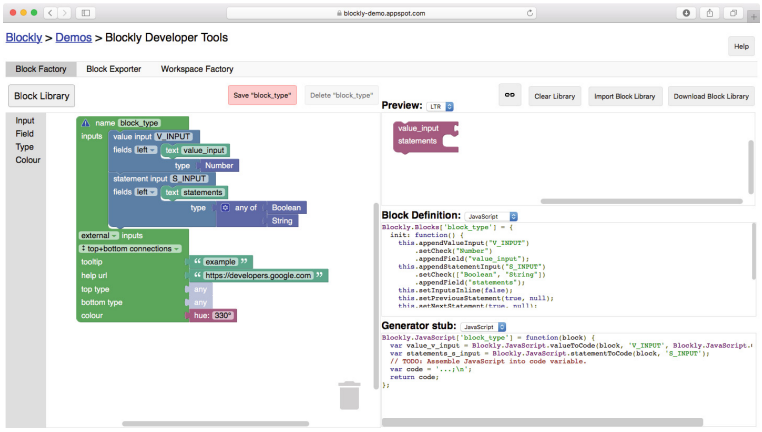

Fig. 2. Blockly Block factory.

3.2 OOPP: The Working Environment

OOPP is a web application developed in JavaScript and entirely executed within a browser. Its graphical interface provides a toolbox that presents the various categories of blocks that can be dragged and linked together to form a puzzle in a work area. Moreover, it provides a context menu that allows the activation of all the features inherent to the current project and necessary for the generation of code. A puzzle representing the set of interfaces and classes of an application is defined by a set of blocks of various types. In particular, we have extended Blockly with object-oriented blocks. Besides the blocks that define interfaces and classes, there are other blocks for attributes, constructors, methods, parameters, data types, etc. An entire OOPP project can be saved into a file, and then loaded again.

OOPP has been designed with the main goal of easing its use to people with limited or no knowledge on object-oriented programming. Therefore, the information to manage during the definition of a puzzle has been minimized. In each particular phase of the definition, it shows only the components that

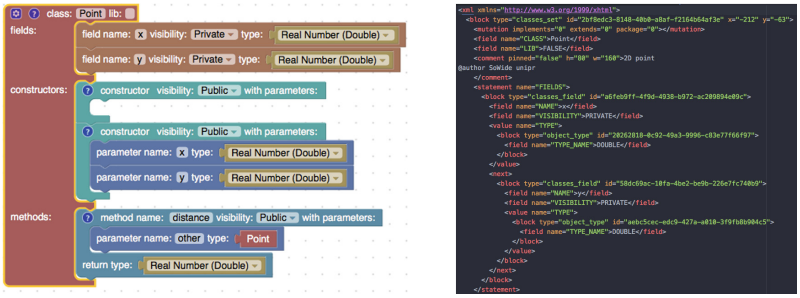

Fig. 3. A simple class and part of its internal representation.

are considered necessary. For example, inheritance is not proposed upfront when starting to define a new class (Fig. 3).

When an application involves a large number of blocks, usually it is necessary to switch from a global view, which allows the analysis of the structure of the application, to a specific view, which allows the analysis of the relationships among a subset of the blocks of the application, or the analysis, modification and extension of a single block. To do it, OOPP allows the visualization at different zoom levels, providing a "collapse" and an "expand" command to automatically minimize and recover to the original size one or all the blocks of the application. For classes and interfaces, it is possible to mark them as provided by an external library; this way, only an *import* instruction is generated.

3.3 Java-Based Projects

The first target language supported by OOPP is Java [8]. Each time that the puzzle is modified in any way, the corresponding Java code is automatically generated anew and shown. This real-time update of the code has the aim to highlight the syntactic features of the target language. Students generally appreciate it a lot, since it allows them to compare the generated code with the block structure, which appears clearer. However, the automatic generation of code can be temporarily disabled. This option is suggested to avoid slowing down the application, in case of particularly complex projects, or while loading large libraries.

The translation of blocks into code is specified in JavaScript files, created for the peculiar blocks of this project, about interfaces, classes, and operational Java code. The constraints, which limit the possible connections among blocks, eliminate most syntactic errors. However, more checks have been implemented, to avoid other common syntactic and semantic errors. This aspect is very important, since the application has an educational scope; as it is documented also in other studies [11], syntactic errors and other issues, related to imprecisely typed code, are one of the most important cause of demoralization and discouragement for neophytes of coding. As an example, we cite here a quite classical error, which

is not merely syntactic, in this kind of projects: a method is not implemented in a class, even if some of its interfaces specify it.

As we have seen, using *mutators* it is possible to organize classes and interfaces in a complex hierarchy. For example, in an application it is possible to define a class and associate it with a certain interface. Figure 4 shows a puzzle representing this example: class X implements interface Y, without providing an implementation of its method Z. From the point of view of blocks, no constraints are violated. However, while generating the corresponding Java code, the process cannot complete and an explicative message for the error is shown. The analysis is performed by collecting all signatures of methods defined in implemented interfaces, and confronting them with methods defined in the class itself; only in the case they correspond, the Java code is generated.

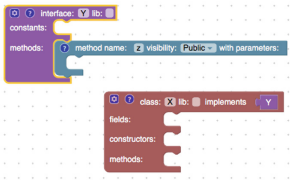

Fig. 4. A simple class and relative interface.

In association with the development environment, an extension has been developed, which allows users to import class and interfaces with all their features. Analyzing a Jar file, the tool can generate an OOPP project file, containing blocks corresponding to all defined classes and interfaces. Various options are available, for example to let blocks appear in "collapsed" mode when loaded, or to set classes and interfaces as "lib", and thus not associated with operational code, but only generating *import* statements. The application uses Java reflection and it can generate a new OOPP project, even when Java source files are not available. Since it is mainly intended as a didactic tool, the import extension does not handle the most complicated settings, e.g. references to external classes, etc.

4 Evaluation

The use of OOPP has been experimented in an introductory CS course, in a school for helping young unemployed students to qualify for developer jobs (an Italian "Corso di Alta Formazione"). The course is organized along the principles of the objects-early methodology, following few introductory lessons about imperative programming. The main objective of the course is to introduce the basic concepts of object-oriented programming and object-oriented software application design, through the Java language. More than 60% of students (Fig. 5), before the course, had no skills or competences about computer

programming, at all. An additional 16,7% had not developed any object-oriented application before, though knowing the paradigm from the theoretical point of view.

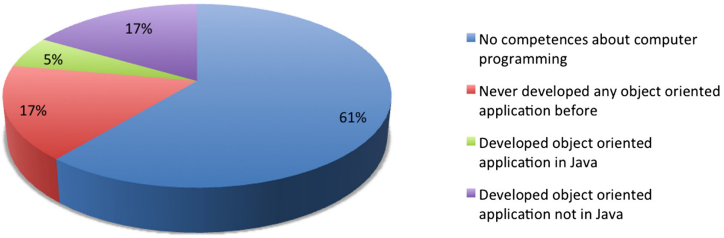

Fig. 5. Competences about computer programming.

In the very first part of the course, lab activities have been characterised by the use of objects of predefined classes, and their reciprocal interaction. This educational approach is inspired by the ideas presented by Kölling [14] and by the BlueJ development environment.

At the moment of designing and realizing classes, and when introducing the concepts of inheritance and polymorphism, students have had the choice to use OOPP, or a more traditional environment. In the first phase, all students have experimented the use of OOPP to design their application and generate the structure of classes for their applications.

Almost all students have appreciated the usability of the OOPP environment; also among neophytes of object-oriented programming, more than 90% has evaluated OOPP as simple to use (Fig. 6). It has been found useful, at least for designing classes for various proposed problems, by 67% of students; 28% of them designed classes for all problems using OOPP environment. In particular, students have reported that, operating through the composition of blocks, it is easier both (*i*) to define the various needed elements (fields, methods, constructors) of a class, and (*ii*) to understand the syntactic structures of the generated code, which is available in real-time during the composition of the puzzle.

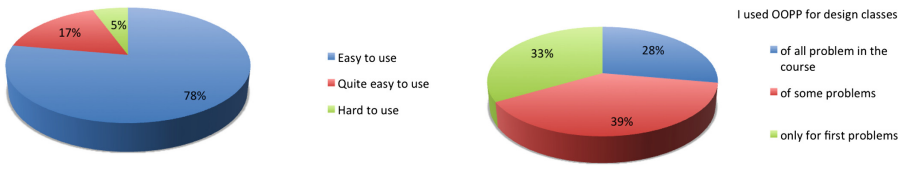

Fig. 6. Usability and use of OOPP.

4.1 Future Developments

The system functionalities have been experimented successfully, for the part related to design, i.e., it allows users to define the high level class model of a Java application in a simple and intuitive way.

A possible development regards the possibility to generate code in more programming languages, implementing the object-oriented paradigm. In particular, one underway extension regards the Python language, in the light of its growing usage in educational contexts. Apart from minor adjustments to blocks, different management of interfaces and support for multiple inheritance, it is necessary to redefine block-code conversions. Anyway, the application structure remains almost the same.

Though not being the initial scope of this project, an extension can also be thought, for allowing a user to develop a complete application, all inside a unified OOPP project. For constructors and methods, it is necessary to provide an implementation, using the operational blocks already available with other systems based on Blockly, together with specific blocks for instantiation and usage of objects. In this case, the comprehensive puzzle obviously becomes larger, or much larger. Thus, the use of such operational blocks inside a unified OOPP project is suggested only for very simple applications. However, the `zoom` and `collapse` functionalities can help to manage a workbench with plenty of blocks.

5 Conclusions

In this paper, we have presented OOPP, a project that merges puzzle programming and object-oriented paradigm in the form of a didactic tool to support the objects-early methodology for introducing the object-oriented paradigm, for high school students and university freshmen. OOPP is a web application which provides blocks for the basic elements of object-oriented programming, i.e., classes, attributes and methods. A user/programmer can create a puzzle as a collection of connected graphical programming blocks. The puzzle could then be automatically translated to Java or any other supported language. In fact, OOPP is not tied up to a specific target language. We think that our project will be useful in CS1 courses, but also in vocational courses, as a starting environment for modeling and designing classes. Automatic generation of code from a puzzle of blocks to diverse object-oriented languages is a valid help to compare syntactical differences from a unique starting project.

References

1. AICA: Informatica nei licei nel contesto della riforma della scuola, documenti di Mondo Digitale (2003)
2. Bennedsen, J., Schulte, C.: What does objects-first mean?: an international study of teachers' perceptions of objects-first. In: Proceedings of the Seventh Baltic Sea Conference on Computing Education Research, vol. 88, pp. 21–29. Australian Computer Society, Inc. (2007)

3. CINI, GII, GRIN: Manifesto sull'informatica nella riforma della scuola superiore. http://www.grin-informatica.it/opencms/export/sites/default/grin/files/manifesto.pdf
4. Cooper, S., Dann, W., Pausch, R.: Alice: a 3-D tool for introductory programming concepts. J. Comput. Sci. Coll. **15**(5), 107–116 (2000)
5. Cooper, S., Dann, W., Pausch, R.: Teaching objects-first in introductory computer science. ACM SIGCSE Bull. **35**(1), 191–195 (2003)
6. Denning, P.J.: The profession of it beyond computational thinking. Commun. ACM **52**(6), 28–30 (2009)
7. Denning, P.J., Comer, D.E., Gries, D., Mulder, M.C., Tucker, A., Turner, A.J., Young, P.R.: Computing as a discipline. Computer **22**(2), 63–70 (1989)
8. Ferrari, A., Poggi, A., Tomaiuolo, M.: Object oriented puzzle programming. In: Didattica Informatica-Didamatica 2016, pp. 1–10 (2016)
9. Gander, W., Petit, A., Berry, G., Demo, B., Vahrenhold, J., McGettrick, A., Boyle, R., Mendelson, A., Stephenson, C., Ghezzi, C., et al.: Informatics education: Europe cannot afford to miss the boat. ACM (2013). http://europe.acm.org/iereport/ie.html
10. Gries, D.: A principled approach to teaching OO first. SIGCSE Bull. **40**(1), 31–35 (2008). http://doi.acm.org/10.1145/1352322.1352149
11. Kelleher, C., Pausch, R.: Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv. (CSUR) **37**(2), 83–137 (2005)
12. Kölling, M.: The problem of teaching object-oriented programming, part 1: languages. J. Object-Oriented Program. **11**(8), 8–15 (1999)
13. Kölling, M.: The problem of teaching object-oriented programming, part 2: environments. J. Object-Oriented Program. **11**(9), 6–12 (1999)
14. Kölling, M.: Using BlueJ to introduce programming. In: Bennedsen, J., Caspersen, M.E., Kölling, M. (eds.) Reflections on the Teaching of Programming: Methods and Implementations. LNCS, vol. 4821, pp. 98–115. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77934-6_9
15. Lister, R., Berglund, A., Clear, T., Bergin, J., Garvin-Doxas, K., Hanks, B., Hitchner, L., Luxton-Reilly, A., Sanders, K., Schulte, C., Whalley, J.L.: Research perspectives on the objects-early debate. SIGCSE Bull. **38**(4), 146–165 (2006). http://doi.acm.org/10.1145/1189136.1189183
16. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al.: Scratch: programming for all. Commun. ACM **52**(11), 60–67 (2009)
17. Upton, E., Halfacree, G.: Raspberry Pi User Guide. Wiley, New York (2014)
18. Vilner, T., Zur, E., Gal-Ezer, J.: Fundamental concepts of CS1: procedural vs. object oriented paradigm-a case study. ACM SIGCSE Bull. **39**(3), 171–175 (2007)
19. Weintrop, D., Wilensky, U.: To block or not to block, that is the question: students' perceptions of blocks-based programming. In: Proceedings of the 14th International Conference on Interaction Design and Children, pp. 199–208. ACM (2015)
20. Wilson, C.: Hour of code: we can solve the diversity problem in computer science. ACM Inroads **5**(4), 22–22 (2014)
21. Wing, J.M.: Computational thinking. Commun. ACM **49**(3), 33–35 (2006)
22. Wolber, D., Abelson, H., Spertus, E., Looney, L.: App Inventor. O'Reilly Media Inc., Sebastopol (2011)