An Objects-First Approach to Teaching Object Orientation based on objectKarel

Stelios Xinogalos[1], Maya Satratzemi[1], Vassilios Dagdilelis[2]
[1]Department of Applied Informatics, [2]Department of Educational and Social Policy
University of Macedonia
156 Egnatia Str., Thessaloniki
GREECE

Abstract: In the last decade the departments of Information Technology have adopted the Object Oriented Programming (OOP) paradigm for introductory programming courses. In this paper we present the problems encountered when OOP is taught to novices and how the microworld objectKarel that we developed helps in dealing with these problems. The main part of the paper has been devoted to providing a description of a proposal for teaching the fundamental notions of OOP with the help of objectKarel and some preliminary results from its evaluation.

Key-Words: Object-oriented programming, Microworld, Hands-on activities, Program animation.

1 Introduction

In the last decade the departments of Information Technology have adopted the Object Oriented Programming paradigm for introductory programming courses. Usually in these courses C++ is used and recently Java is being used more and more. Although the tools for software development, the teaching support material as well as the experience of teachers in OOP are not as developed as those for structured programming, it appears that in the end, the opinion prevails that the introduction to programming should be made with the OOP paradigm – since OOP is being used continuously more in the work sector.

The adoption of this approach to the teaching of programming has resulted in adding a number of difficulties and misconceptions inherent in OOP to the already existing difficulties and misconceptions of novice programmers, which were located during the teaching of structured programming. One of the most important instructional problems that are related to the OOP paradigm is the fact that the object-oriented technique for the development of programs is difficult for students [5], since it is more abstract than the technique for structured programming and more exacting in the processes of analysis and design. Furthermore, the existing OOP languages are not suitable for an introduction to this paradigm. As a consequence, the introductory programming courses teach the students merely to be «consumers» rather than «creators» of software that can be reused [10].

These opinions have led researchers to develop programming languages and environments specifically designed for education. Bergin et al [1] proposed the language Karel++, a language closer to C++, to provide a means to novice programmers to learn OOP. Kölling et al. [7] devised the programming environment BlueJ for an introduction to OOP using Java as a language. Prompted by the language Karel++ but also by related research on the difficulties and the misconceptions of beginners we developed a programming microworld, called objectKarel ([11], [13]) that is based on the language Karel++. objectKarel incorporates features not usually available in the existing programming environments and solves many problems that have been recorded in the teaching of programming to novices. The strengths of objectKarel are built around three design goals: simplicity, interactivity and visualization. In this work, we firstly present the environment objectKarel through the rationale that guided its development. The main part of the paper has been devoted to describing a proposal for teaching the fundamental concepts of OOP (objects, classes, inheritance, polymorphism, overriding) with the help of objectKarel. This teaching proposal has already been applied and evaluated positively by students, while some results of this evaluation are presented in the paper.

2 The objectKarel environment

One of the most significant difficulties that novice programmers must deal with when introduced to

programming is the extended instruction set of programming languages. Also, students have great difficulty in comprehending the general characteristics of the mental machine, which they learn to control, and its relation with the natural machine. Milne and Rowe [8], report that beginners who are introduced to OOP are unable to comprehend what is happening to their program in memory, as they are incapable of creating a clear mental model of its execution. In order to deal with these obstacles we chose to use the *mini-language of Karel++* that uses the metaphor of a world of robots. We believe that the use of a microworld like this, which is based on a physical metaphor, draws students' attention, gives the opportunity to solve interesting problems even from the first lessons and contributes greatly to decreasing the "distance" between the mental models or descriptions of algorithms in a natural language and their description in a programming language.

In order to support, even more, beginners in comprehending the general characteristics of the "mental machine", we created an advanced *animation and visualization system*. The process of program execution is not hidden and so students do not develop an input-output oriented understanding, as is the case in commercial programming environments. *Program animation* helps students understand the language's semantics and flow of control, as well as the way in which the commands of their program are connected with the actions of the robots. Besides the ability of tracing and step-by-step execution we have also used the technology of *explanatory visualization*, that is the presentation of explanatory messages in natural language about the semantics of the current command.

The relevant research has also shown that the syntactic and semantic rules of programming languages [4] create so many difficulties to beginners that they focus their attention on these rules and not in developing programming skills. These difficulties, in combination with syntactic and semantic errors that are usually presented in coded form and are in no way instructive for the students, often discourage and disappoint them. Therefore, with the aim of minimizing the number of syntax errors, we decided to incorporate a *structure editor* for developing programs: 1) choosing the appropriate action (class/method declaration, construction of object) or choosing a message to send to an object from a single menu (Fig. 1), which is automatically updated whenever the user declares/deletes/edits a class/method; 2) interacting with the system through dialog boxes. We chose to incorporate this editor with the express aim of helping the beginner to focus on the solution of the problem and the acquisition of concepts rather than on the syntactic details of the programming language. Furthermore, our programming environment detects and reports *understandable and highly informative error messages* of all types: the line number reported is the actual line of the error; messages report not only what is wrong but also explain why it is wrong; the error messages use physical language and not codes.

Finally, objectKarel incorporates *e-lessons*, consisting of theory and *hands-on activities*, which aim at supporting the teaching of programming. The beginners familiarize themselves with the taught concepts rather than writing a program from the beginning and they are given the opportunity to experiment via ready examples. Kölling et al. state in [7] that it is wrong to begin the teaching of OOP from scratch. Writing a class involves design. One has to decide what class(es) should exist and what the methods should be. Instead, a student should start by making small changes to existing code [7]. In this way, students can go through a sequence of exercises that they can understand step by step. Moreover, via the ready examples that were incorporated in the theory and in the activities students are given the chance to learn a lot from studying well written programs and copying style idioms.

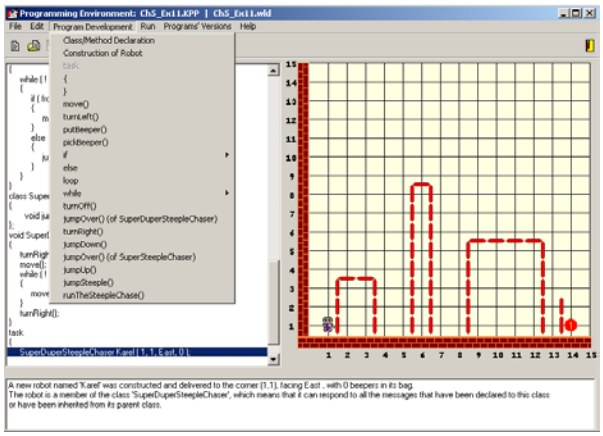

Fig 1. The main window of objectKarel.

Other known tools based on Karel++ and its predecessors are JKarelRobot [2] and Jeroo [9]. The restrictions of these tools, regarding object-orientation, are that students can create just one robot in JKarelRobot, while in Jeroo there is only one class, students can create up to four robots, inheritance is not supported and the Jeroo class can be extended with void methods, but not with predicates. On the other hand, a book and an accompanying simulator for a Java-based

descendant of Karel++, called Karel J Robot [12], have been recently published and use pure Java.

3 Teaching OOP with objectkarel

In this section we present the way the fundamental concepts of OOP can be taught. However, objectKarel can be used for teaching control and repetition structures too. In an introductory programming course that uses the OOP paradigm, we consider that one should start directly with the concept of objects. Of course, teaching objects as the first basic notion of a programming language is not always easy with traditional programming environments. However, this is possible with objectKarel since the objects are robots.

The lessons are structured and are carried out at the laboratory. In each lesson, the students familiarize themselves with certain notions of OOP using the relevant theory and especially, the selected examples that have been incorporated into the programming environment with the form of hands-on activities. In these activities, the students as a rule investigate the proposed code executing it in one go or step-by-step, or by converting the code in order to realize the changes that the transformations involve in the robots' behaviour. The environment, thanks to its special editor, the characteristic messages that are presented in each case, the fact that the robots' behaviour is visible and generally because of its special features, facilitates the student in this investigation. Moreover, the successive activities proposed, progressively familiarize the student with the features of the environment. Lastly, in each lesson, a sequence of exercises is proposed whose aim is to clarify or to emphasize certain aspects of each proposed notion. Following, we present some lessons in order to clearly explain the teaching methodology for the basic concepts of OOP that we propose with the help of objectKarel.

3.1 Objects - Classes

This lesson aims at teaching the following concepts: object, construction & initialization of an object, messages/methods, attributes & behaviour of an object, class, program/task. First, the theory of the unit «Introduction» is used so as present the microworld of objectKarel and introduce students to the main principles of OOP. Next, the theory of the unit "Objects & Messages" is used so as to present the basic class of robots. With the help of the corresponding activity (similar to Fig. 3, but without the "Properties" panel) the 4 messages to which each robot responds to are explained: move(), turnLeft(), putBeeper(), pickBeeper(). This activity aims at familiarizing students with the most basic notion of OOP - which is sending messages to objects – by simply clicking buttons rather than using from the very beginning the programming language. When students click a button labeled with the name of the message, they see: i) the result of executing the message in the world as well as the robot-object which executes this message; ii) the syntax of the command, which was executed by clicking the button, in the programming language that they will later use to develop their programs.

It is also emphasized that each object is self-sufficient, or in other words it has its own «natural» existence and identity and that is why we always use the name of the object we send a message to: <object-name>.Message().

Lastly, it is clarified that each robot with the above capabilities constitutes an instance of the basic model – a class called Primitive_Robot. In order to: (i) explain to the students that a class can supply us with all the objects we want provided that we give the suitable command for their construction and initialization; (ii) clarify the concept of attributes, the values of which are altered with the execution of methods; and (iii) behaviour, we use the activity of the unit «Classes». The relevant text for this activity is as follows: *«Press the button "Construction and Initialization of an Object" in order to create a robot that is a member of the class Primitive_Robot. In the dialogue box that appears give a name to the robot and initialise its properties. Next, help the robot collect all the beepers in its bag (without executing an error shutoff) by sending it the appropriate messages. Watch how the execution of the messages changes the values of the robot's properties, as well as the syntax of the messages in the programming language.»*

When students click the button «Construction and Initialization of an object» the dialogue frame «Construct an object (robot)» is presented (Fig. 2). This dialogue box also appears during the programs' development for constructing objects. At the top of this frame there is a template of the command for creating and initializing an object and at the bottom there is a short explanation of its meaning. Students select the class of the object from the popup list, (in this particular case it is predetermined), give a name to the object and initialize its attributes.

If the user's commands are correct, the frame closes and the card of the activity (Fig. 3) is informed with: the name of the new robot; the initial values of its attributes; the messages which the robot can respond to, and the form of students' actions in the programming language (code pane).

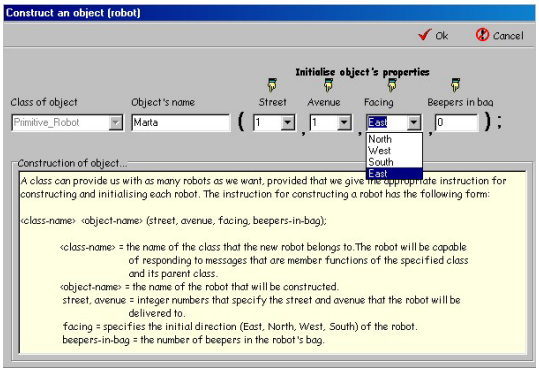

Fig. 2. Constructing & Initializing an object.

Yet again using the available messages (buttons on the card) the student can solve the problem without having to write a program. The program is developed in stages: each time the student chooses to send a message to the robot - by clicking the buttons - the corresponding command is appended to the text area. In this activity, the student is also given the form of the main task block.

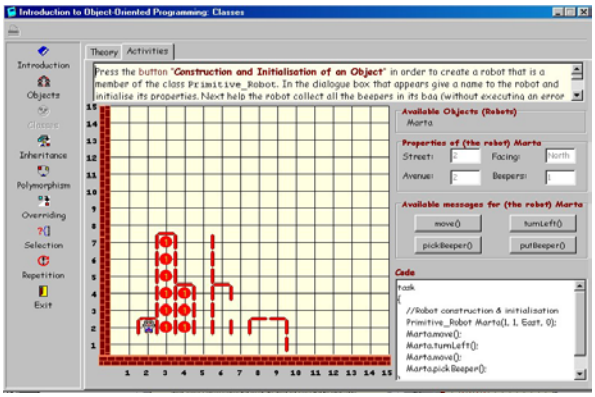

Fig. 3. The activity of the unit Classes.

Next, the structure of a program and the features of the programming environment are presented through an example. Students open an existing program, compile it, create a situation in the robots' world, execute the program in all possible ways, and use the structure editor for making changes that cause syntax, logical and execution errors. Finally, students carry out various assignments.

3.2 Inheritance

The objective of this lesson is: (1) for the students to comprehend the notion of inheritance, the advantages of creating new classes and re-using/modifying existing classes; (2) to familiarize students with the declaration of new classes and the definition of methods; and (3) present a simplified form of UML class diagrams. The basic notions, which this particular lesson refers to, are: base class, parent class/superclass, subclass, multilevel inheritance, inheritance hierarchy, declaration, scope resolution operator, dictionary, a simplified form of UML class diagrams. The notion of inheritance is presented with the use of an example. Specifically, the problem of the robot-traveller included in the unit «Inheritance» is used: *«A robot should be programmed so that when it travels, it covers large distances. We assume that the robot begins at the intersection of the 1st avenue and the 2nd street and it must move in the direction of east along the 2nd street for 10 kilometers (1 km = 8 blocks), pick up a beeper and then move 5 km north. Since the robots of the Primitive_Robot class can only move one block and do not understand the concept of km we need to translate our solution into commands that move the robot one block at a time. This means that our program will consist of 120 move() commands:*

```
task
{
  Primitive_Robot Karel (1, 2, East, 0);
  Karel.move(); … //79 times Karel.move();
  Karel.pickBeeper();
  Karel.turnLeft():
  Karel.move(); … //39 times Karel.move();
}
```

In the context of this example students comprehend that often, even for simple problems, very large programs can develop which are difficult to understand, to debug or to modify in order to solve similar problems. This gives us the chance to explain that in order to solve this problem, the programming language of robots gives us the possibility to create new classes of robots, which contain new methods. In other words we can define classes that provide us with robots that have increased capabilities. For example, in the robot-traveller problem we can define a subclass with a method, such as moveKlm(), which will call the method move() eight times. Then we can use an object/robot of this class and call the moveKlm() method. In this case, our program will consist of just 23 commands!

Following, the activity of the unit «Inheritance» is proposed, where the problem of «sweeping a 4 one-block steps stair» is presented and a discussion with the students on its solution takes place. The students study and execute two programs for the above problem – in the 1st a robot of the Primitive_Robot class is used, while in the 2nd a robot of a new class is used. In order for the students to better comprehend the notion of inheritance, the advantages of creating new classes, and re-using existing ones, they are asked to select one of the two programs stated above and to

determine the changes that must be made when the stair has 10 steps. Whatever the answer is, the students are required to justify it. The relevant discussion that follows makes it obvious that new classes offer advantages.

The 2nd lesson is completed with assignments, specially designed to detect whether students have comprehended the taught concepts or not, as well as their difficulties and misconceptions. For example, in the context of the 1st assignment students are given a problem specification and a textual description of the methods of the four classes (3-level inheritance) required for solving it and are asked to design a UML class diagram, implement the classes and write the main task block. The 2nd assignment requires the use of 4 objects of a single class defined by students, and its goal is to ascertain whether Holland's et al. [6] conclusion that some students tend to become confused between classes and their instances (objects) or whether Carter's and Fowler's [3] conclusion that the distinction between objects and classes does not cause problems for the students is verified.

3.3 Polymorphism

The objective of this lesson is for the students to comprehend the concepts of polymorphism, good class design and refactoring, which is explained indirectly through the activity of the corresponding unit (Fig. 4): *«In the robots' world there are two stairs. The steps of one stair are a different distance apart to those of the other stair. The two robots must climb the stairs sweeping the beepers that lie on each step. Click the button "Program Execution" to study and execute the program. Observe that both robots are sent the same message (climbStair()), however, the robots, depending on their class, respond to it differently.»*

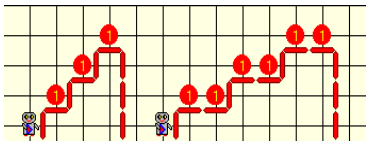

Fig. 4

The program contains the superclass AugmentedRobot that implements a "turnRight" method and the subclasses SmallStair_Sweeper and BigStair_Sweeper, which both implement a method with the common name climbStair() that instructs the robot how it can go up to the next step, gathering the beepers as it goes along. For novices, this solution seems to be fine. However, it is obvious that this solution is an example of "bad" class design. This "bad" class design is discussed with students and a better solution is suggested: a Stair_Sweeper class should be defined, while the classes SmallStair_Sweeper and BigStair_Sweeper should derive from it. This is when students are presented the concept of refactoring, which is applied in the context of the assignments carried out by students.

4 Evaluation

The lessons described briefly in the previous section have already been applied to students of a department of Applied Informatics. Specifically, five two-hour lessons were carried out: (1) Objects-Classes; (2) Inheritance; (3) Selection & Repetition Structures; (4) Polymorphism; (5) Overriding. The lessons were followed by an evaluation of students' knowledge as well as an evaluation of the environment and the lessons by the 24 students that participated. All the students had attended compulsory programming courses and faced difficulties in applying the principles of programming (either imperative or OOP). Due to space limitations, we briefly present students' answers in some questions of the final questionnaire related to the evaluation of the proposed teaching approach. Before that we would like to stress out that: (i) the described lessons are a result of application and evaluation of a prior series of lessons based on objectKarel [11]; (ii) the 3rd lesson can be omitted or divided in 2 lessons based on students' prior knowledge.

Question 1: Evaluate in the range of 1 to 5 (excellent) the structure and quality of the educational material.

Table 1. Replies in Question 1.

Educational material	Score (1-5)
Lessons	4,8
Text	4,2
Activities	4,5
Assignments	4,5

Question 2: Did the series of lessons help you in comprehending programming concepts?

Question 3.1: If you had been introduced to programming with the specific programming language and environment do you believe that you would have faced fewer difficulties?

Table 2. Replies in Questions 2 and 3.1

Reply	Question 2	Question 3.1
Yes	87,5 %	87,5 %
No		4,2 %
Don't know	12,5 %	8,3 %

Question 3.2: If yes, which problems do you believe that you would not face? (open-type question)

Table 3. Replies in Question 3.2.

Problem	%
Comprehending OOP concepts/principles	61,9
Focusing on learning the language's syntax	23,8
"Being afraid of programming"	14,3
Comprehending selection/repetition structures	4,7

Question 4: For which circumstances would you suggest the use of objectKarel for teaching programming?

Table 4. Replies in Question 4.

Problem	%
Introducing university students to OOP	42,9
Teaching programming in Secondary Education	28,6
Introduction to programming in Universities	14,3
As an aid to teaching a conventional language	14,3
Teaching programming in small ages and aged ones	9,5

Question 5: *Did the lessons help you comprehend and/or clarify any OOP concepts that you faced difficulties with?*
Question 6: *Do you believe that after attending the specific series of lessons* you would face difficulties with the corresponding concepts in a conventional programming language, such as C++ or Java?

Table 5. Replies in Questions 5 and 6.

Reply	Question 5 (%)	Question 6 (%)
Yes	95	5
No	5	95

5 Conclusions

Programming microworlds are usually used for introducing novices to programming. In our case, objectKarel was used for helping students tackle their difficulties after an unsuccessful introduction to programming. The analysis of the questionnaire shows that objectKarel and the proposed series of lessons helped them overcome various difficulties/misconceptions. We believe that the results would be even better if students had been introduced to OOP with objectKarel from the start. We further believe that such an approach does not exclude then the use of a professional programming language, or even better the metaphor of "Karel J. Robot" that uses Java. On the contrary, it will help beginners to grasp the importance of fundamental notions of OOP and thus be able to deal with more complex problems based on the OOP paradigm.

Acknowledgements
This research is being funded by the Greek Ministry of Education and the European Union as part of the project "Pythagoras II- Funding of research groups in the University of Macedonia".

References:

[1] Bergin, J., Stehlik, M., Roberts, J. and Pattis, R., Karel++ - A Gentle Introduction to the Art of Object-Oriented Programming. 2nd edn., Wiley, New York, 1997.
[2] Buck, D. & Stucki, D.J., JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum, *ACM SIGCSE Bulletin, 33(1)*, 2000, 16-20.
[3] Carter, J. & Fowler A., "Object Oriented Students?", *Proc. of the 3rd ITiCSE*, 1998, 271.
[4] Du Boulay, "Some Difficulties Of Learning To Program". In *Studying The Novice Programmer*, Soloway, E., Sprohrer, J. (Eds.) Lawrence Erlbaum Ass., 283-300, 1989.
[5] Hadjerrouit, S., "A constructivist approach to object-oriented design and programming", *Proc. of the 4th ITiCSE Conference*, 1999, 171-174.
[6] Holland, S. Griffiths, R. & Woodman, M., "Avoiding object misconceptions", *Proceedings of the 28th SIGCSE*, 131-134, 1997.
[7] Kölling M., Rosemberg J., "Guidelines for Teaching Object Orientation with Java", *ACM SIGCSE Bulletin, 33, 3*, 2001.
[8] Milne, I. & Rowe, G., "Difficulties in Learning and Teaching Programming - Views of Students and Tutors", *Education and Information Technologies*, 7:1, 55-66, 2002.
[9] Sanders, D. & Dorn, D., Jeroo: A Tool for Introducing Object-Oriented Programming, *ACM SIGCSE Bulletin, 35(1)*, 2003, 201-204.
[10] Wick, M., "On Using C++ and Object-Orientation in CS1: the Message is Further more important than the Medium", *ACM SIGCSE Bulletin, Vol. 27, Issue 1*, 322-326, 1995.
[11] Xinogalos, S., Satratzemi, M. & Dagdilelis, V. An Introduction to object-oriented programming with a didactic microworld: objectKarel, Computers & Education, Vol. 47, Issue 2, September 2006, 148-171.
[12] Karel J Robot: http://www.cafepress.com/kareljrobot
[13] objectKarel is available at: http://www.csis.pace.edu/~bergin/temp/findkarel.html