# Teaching Object Oriented Programming to Novices

Thesis for the degree Doctor of Philosophy

Submitted to the Scientific Council of the Weizmann Institute of Science

Rehovot, Israel

By: Noa Ragonis

June 2004

## Abstract

This research study describes various aspects of teaching object oriented programming (OOP) as the primary paradigm for novices. The objective of the study was to lighting up various dimensions and provides a significant basis for receiving answers to the questions: (1) What key concepts of object oriented programming are important and should be included in an introductory course in order to enable construction of a suitable knowledge model of the paradigm, by high school novices? (2) What are the perceptions that novices build during learning basic concepts in object oriented programming? (3) What teaching sequence would maximized the understanding of OOP key concepts?

The research is a constructivist qualitative study that implements the paradigm research method on perception of object oriented programming concepts in computer science. Extensive data collection was carried out through the entire period: observations and field notes, audio and video recordings, and collection of artifacts. The collected artifacts includes: homework assignments, class work, examinations, and final project. The various artifacts proved to be particularly fruitful if that they showed precisely the what concepts were understood and what concepts were problematical. The collection of data was done over two years of teaching. The study presents a unique implementation of the constructivist qualitative research principles in the field of analysis of findings and their presentation.

The implemented approach was “objects first” implemented in the Java programming language and in use of the BlueJ visual development environment. To suit this implementation, a collection of basic concepts was chosen: class, object, attributes (fileds), methods, constructors, instantiation, simple class and composed classes, method invocation, the object’s state and the ways to change it, mutators and accessors. It is also important to teach skills for using ready-made classes according to a given interface.

The guiding decision regarding the sequence of teaching concepts were: to present from the very beginning simple and composed classes in

implementing a containment connection, avoid presenting a main program for considerable time, emphasizing on teaching structure and conception for object oriented programming rather than developing of algorithms. The main research population was high school novices. In addition, a population of pre-service teachers and in-service teachers in refresher courses was examined too.

Following the collection of data and its analysis we found outstanding perceptions and difficulties in four main categories. The findings are presented in two ways: a narrative description that tells the story of the research, and an in-depth analysis of four categories of main concepts. The uniqueness of the study is in the detailed characterization of students' perceptions and the attempt to point to possible reasons for difficulties and erroneous perceptions of the students.

The four categories of concepts were: object vs. class, instantiation and constructors, simple vs. composed classes, and program flow. In each category we identified difficulties and perceptions. Episodes were assigned to (one or more) category as needed. Sub-categories of concepts were created in each category. Altogether we identified 58 different characteristics pertaining to the four categories of concepts.

Each of the four categories of OOP concepts is presented in a separate chapter of the thesis.

Object vs. class – Students perceptions regarding the basic relationship between class and its' driven objects. A class is a pattern from which objects can be created, but conversely, to find the fields and methods of an object you must look in the class. The sub-categories that arose in this category were: the nature of class as a pattern (6 characteristics), connections between object and class (5 characteristics), object creation (1 characteristics), and identification of objects (5 characteristics).

The difficulties and perceptions that were found to be frequent in the first phases of teaching were: "It is possible to define a method that does not relate to any attribute," "It is difficult to students to understand the significance and classification of various methods", "What is a creation of an object?", "Who is the object?". During progression in practice with BlueJ the concepts gained meaningful and most of the difficulties disappeared. Students who deal with classes common in everyday life and their textual representations in diagrams, "forget" that they deal with a presentation of a computerized system. They relate to concepts and considerations of everyday life and "put aside" the formal rules.

The perception of the object identification concept and mainly the difficulty in "un unequivocal identification of the object entity due to multiple

presentations” is the most significant in understanding and inseparable from the problem of “what a creation of an object is.” The causes are the attempt to handle abstract conception on one hand, and the practice in BlueJ in which the object has a more realistic look yet has many presentations. Another perception that came up in this context viewed the class as “a collection of objects.” This perception repeated also in regard to composed classes.

Despite the difficulties, it seems that the theoretical teaching using the diagrams and the support of BlueJ visualization instills in a good enough way the OOP core concepts. Students understand from the first month the basic connections between the class as a pattern for object creation. They understand the generalization – a class represents the common denominator of entities. They understand the meaning and result of operating methods on objects. The students further encounter new difficulties when they have to verify their concepts comprehension, when they are needed to implement them in the programming language.

Instantiation and constructors – Understanding the instantiation that realizes the connection between the class and the object was vital and affected also from the way it was defined in the programming language. The sub-categories that arose in this category were: general understanding of the instantiation (4 characteristics), understanding the instantiation in a composed class (3 characterizations), and understanding the instantiation when it was affected by the version of its definition in the programming language (5 characterizations).

It seems that it was not clear to the students what was taking place in the computerized process of instantiation of simple and composed objects. The difficulties rose despite the gradual teaching and the use of a supporting visualized environment. I think that these stemmed from three reasons: (1) the duality in relating to the static class and the dynamic run; (2) difficulties in understanding the representation of class in a computerized system by allocation fields of memory to suit its definition; (3) problems in understanding the execution of methods and program flow.

The study describes in detail the effect of various versions of constructors definitions in the programming language on students’ perception. According to the results of the study in the first year, we used in the second year only a version that used parameters to provide values object attributes, a decision that saved a considerable amount of difficulties. In the last assignments of the year, students showed a well formed knowledge regarding the process of creation that was made of memory allocation and execution of the constructor in both a simple and a composed class.

Simple class vs. composed class – Students showed difficulties in understanding the core of object oriented programming – implementing of the connection between different classes. In this study the concept “simple class” relates to a class whose attributes are of built-in types in the language, and the concept “composed class” relates to a class that has attributes of a different user class type. The sub-categories that arrose up in this category were: understanding encapsulation (7 characteristics), understanding the modularity (3 characteristics), class is a collection of objects (3 characteristics), understanding the “black box” (one characteristic), personification (one characterization) and understanding a self method (one characteristic).

The main difficulty that was found was in understanding the encapsulation principle. Yet, the seven perceptions in this category occurred only up to the middle of the school year and disappeared with the advancement of studies. The perceptions indicated a difficulty in basic understanding that an object constituted one entity that included all its attributes and since it was attached to a particular class one could activate on it the methods defined in its’ class. The significant perception that did not disappear was the perception that “there is no need for mutators and accessor when using values of attributes of a simple class type.” This evidence raised the question whether to instruct students to include in each case mutators and accessors to each attribute, or only where needed, and when to integrate these methods in the process of learning.

The erroneous perceptions in the subject of modularity appear especially in regard activation of methods in the context of developing a particular algorithm, and do not point to a meaningful difficulty in the division of the problem domain into entities. Erroneous attachment of methods between the simple class and the composed class appeared only in the beginning of learning. Later the distinction between them was very clear. The problem in the core of modularity is “no use of methods that were defined in the simple class to attainment of a goal in the composed class”, however, this difficulty was not very common. Another perception that disturbed some of the students was in the context of realization of the methods: “How the distinction between methods with an identical name does in executed?” I assume that a teaching approach that would combine teaching of the main method up front, would enable students to view the activation of a method on a simple class object and on a composed class object, and could provide them an answer that the executed method is the one that is defined in the appropriate class.

The perceptions that indicated viewing of the composed class as a collection of simple class objects appeared only in the first half of the school year and completely disappeared afterwards. The use of composed classes in which a

detail from the attributes of the simple class appear side by side with attributes from built-in types can prevent these erroneous perceptions.

Difficulties in understanding “self method” ( a method invocation that appears in the body of another method) appeared only after a composed class was defined. Students were exposed to the way of activating a method on an object, from which they had made an erroneous generalization that method invocation had to always be related to on an explicit object. Here, too, earlier exposure to a project that includes a main class would have shown how the composed object was created and would enable easier understanding of the correct place and structure of activating a method on the composed class.

Program flow – Students find it hard to create a general picture of the execution of a program that solves a certain problem. We included this category because we found that the students asking numerous questions of the form: What actions are carried out? When are they carried out? What triggers the action? What is the order of execution of actions? The sub-categories that arose were in the topics of: Understanding executions of methods (5 characteristics), understanding of data flow (2 characteristics), students who thought that some things just happened with no cause (3 characteristics), students wondered: “how does the computer know?” (2 characteristics) and in general did not understand the overall control over execution – “what happens and when?”

The difficulties and the perceptions in the subject of methods invocation repeatedly raise the difficulty that some of the students have in understanding the difference between defining a class and executing methods from within its’ definition – the static/dynamic dimension. This problem becomes even more acute in understanding of the program flow as was demonstrated in the perceptions that: “Methods are executed by the order of their appearance in the class,” or “It is possible to activate one method only once.” A better understanding of the class as a pattern for creating objects and the methods that are defined in it as a collection that can be use according to need, would have prevented this un clear encounter with concepts in the context of program flow.

*****

The research conclusions chapter in the thesis integrates the results of all chapters. The conclusions of the research are presented in the following categories: (i) a summary of the conceptions held by the students and their difficulties, (ii) a summary of the recommendations for teaching, (iii) a suggestion for a syllabus for teaching OOP to novices that takes into account the these recommendations, (iv) an explanation of the unique contribution of the research and recommendations for further research.

From examination of students understanding at the end of the teaching process we could see that the basic concepts were understood as well as the principles of object oriented programming we emphasized like encapsulation, modularity and data hiding. In the summarizing questionnaire all the students explained properly the objective of instantiation, explained the process involved in its activation and also implemented the creation of a new composed object in Java. They also demonstrated almost perfectly classification of new methods to the appropriate class. Their explanations used the appropriate OOP terms. Students also demonstrated understanding of program flow through description, analysis and expansion of the main method defined in the project, including a detailed description of the process scenario as a result of carrying out the “main” method. In the standpoint questionnaire that they filled before carrying out the final project, we could see that most of the concepts were no longer difficult for them. Three concepts were shown to be more difficult in a significant way: a composed class, main program and the mutators and accessors methods.

The success in the development of the final personal projects, measured by criteria of OOP principles, was very high.

This study shows that it is possible to teach object oriented programming to high school novices in Java. The success of the students in planning and implementing a final project, as did other findings, confirms it. From the study we learn that there is great importance to the order in which concepts are presented, for building a proper model of knowledge about the basic concepts in the field. The great number of perceptions and difficulties described does not mean that teaching object oriented programming is unsuitable to novices. Most of the perceptions came up in low frequency, including perceptions that appeared only once. Also many of the perceptions characterized a particular period of learning and disappeared with the advancement of learning. Awareness of curricula developers and teachers’, regarding the multitude of perceptions and students’ difficulties found here could be used in building curricula that enable fruitfully process of teaching and learning.

Teaching Object Oriented Programming (OOP) to novices is widely known to be quite problematic. Students might be able to write a piece of code in an OOP language, usually Java, but their conceptual grasp of object-oriented concepts seems to be limited. This leads to poor implementation of object-oriented concepts and inability to take advantage of the strengths of OOP. Object-Oriented programming (OOP) refers to a type of programming in which programmers define the data type of a data structure and the type of operations that can be applied to the data structure. As Java being the most sought-after skill, we will talk about object-oriented programming concepts in Java. An object-based application in Java is based on declaring classes, creating objects from them and interacting between these objects. I have discussed Java Classes and Objects which is also a part of object-oriented programming concepts, in my previous blog. Edureka 2019 Tech Career Guide is out! Hottest job roles, precise learning paths, industry outlook & more in the guide. Download now. Object Oriented Programming. object-orientation is a collection of techniques that enable developers to construct secure, accessible, sustainable, well-documented, recyclable software systems that meet their users â€˜ needs. it is stated that object-orientation offers new mind resources for software developers to use in solving a wide range of issues. a new view of computation is given by object-orientation.Â  You use a remote control if you want to shift the tv channel from your place. that remote control is an entity concealed inside of it with a variety of attributes and behaviors. you also recognize and believe that pressing a button would execute that specific function without thoughtful of these secret attributes the microchips, wiring, etc. in the abstract, youâ€™ve communicated. Object-oriented programming is a method of programming based on a hierarchy of classes, and well-defined and cooperating objects. Classes. A class is a structure that defines the data and the methods to work on that data. When you write programs in the Java language, all program data is wrapped in a class, whether it is a class you write or a class you use from the Java platform API libraries. The ExampleProgram class from the simple program in the first lesson of Part 1 is a programmer-written class that uses the java.lang.System class from the Java platform API libraries to print a character string to the command line. Copy. Copied to Clipboard.