Teaching Programming: Going beyond "Objects First"

Jorma Sajaniemi[1] and Chenglie Hu[2]

[1] University of Joensuu, Department of Computer Science, P.O.Box 111, 80101 Joensuu, Finland, saja@cs.joensuu.fi, WWW home page: http://www.cs.joensuu.fi/~saja/

[2] Carroll College, Department of Computer Science, Waukesha, WI 53186, USA, chu@cc.edu

Abstract. The prevailing paradigm in teaching elementary programming uses Java as the first programming language and the "objects first" approach as the conceptual basis. This approach has several shortcomings, e.g., high drop-out rates and poor skills in basic constructs like loops. This paper suggests an alternative approach that combines a strong start in basic constructs with early object-orientation. The key idea of our approach is to start with concepts that are common to both procedural and object-oriented programming, i.e., basic control and data structures and a simple form of the interplay between responsibility and implementation. Only then various abstraction mechanisms—procedural, functional, object-oriented, and data as well as the interplay between responsibility and implementation in these abstractions—will be introduced. The alternative approach is also compared with the ACM Computing Curricula.

1 Introduction

The prevailing paradigm in teaching elementary programming uses Java as the first programming language and the "objects first" approach as the conceptual basis. The use of Java is motivated by its extensive use in industry and students' wish to learn a "real" language that can "guarantee" them a job in the future. The objects first approach is used to avoid possible negative transfer effects from procedural programming that make the transition to object-oriented programming hard. Thus the widespread use of object-oriented programming in software industry has led to the abandonment of the previous teaching paradigm: Pascal as the first programming language and procedural programming as the conceptual basis. The new approach is not free of problems either. For example, panels in recent computer science education conferences [1–4] have criticized the current approach due to the relatively high complexity of Java and the abstract nature of the objects first approach. It is therefore worthwhile to study alternative approaches to basic programming education.

Regardless of their career goals, most students will need to have basic understanding of programming and the programming skills required for problem-solving. After graduation they will use a variety of programming languages and tools in, e.g., different phases of information system development and software engineering work. More important than the mastery of some specific language is then the mastery of programming concepts utilized by those languages and tools. These concepts comprise object-orientation as well as other programming paradigms. This paper suggests a new elementary programming education approach that combines a strong start in basic constructs with early object-orientation. The key idea of our approach is to start with concepts that are common to both procedural and object-oriented programming. This is followed by various abstraction mechanisms—object-oriented, procedural, functional, and data abstraction—representing different paradigms used in current programming practice.

The basic constructs needed to understand the idea of programming include, e.g., variables, assignment and simple imperative control structures such as sequence, iteration and conditionals. An appropriate name for our approach would thus be "imperative-first". However, it has been a common tradition (see, e.g., [5]) to use this term to cover not only imperative control structures but also procedural abstraction. For clarity, we will thus use a fresh name, "variables-first", to emphasize our approach that starts with the essential part of an imperative programming paradigm: variables and, particularly, their roles in imperative control structures.

The rest of this paper is organized as follows. Section 2 makes a literature survey on the problems of the current approaches to programming education. Section 3 then presents the new approach and section 4 compares it with the ACM Computing Curricula. Section 5 contains the conclusion.

2 Current Problems

There are several shortcomings of the current "objects first with Java" paradigm. The objects first approach means that programming courses start with the introduction of objects that model some application domain, the attributes of these objects, the responsibilities of objects and their relationships with other objects, and finally the implementation of the responsibilities by the use of methods. For example, a programming textbook [6] that uses this approach starts with modeling simple graphical user interface elements. The first chapter of fourteen pages long contains no program code but introduction to the following concepts (listed in the summary of the chapter): object, class, method, parameter, signature, type, multiple instances, state, method-calling, source code, result—all to be understood in the context of abstract picture drawing and GUI interfaces without showing a complete working program even at the level of "Hello World". As a result, learners have to work with abstractions for a long time before they can base those abstractions to program code—another abstraction even though at a more concrete level. The high drop-out rates of objects first programming courses may be due to problems in acquiring correct understanding of the abstractions that those courses start with [7].

Modeling an application domain without knowledge of the programming techniques needed in the implementation of the model is like designing knitting models without knowing the knitting techniques that are required to produce basic knitting patterns. Knitting is not taught by starting with knitting models but with practicing the use of knitting needles in order to first obtain skills that are necessary to produce basic knitting patterns. Only by knowing the basic building blocks and by understanding the basic techniques needed in combining the building blocks will it be possible for students to make designs that can actually be implemented.

The question of the first programming language is not free of problems, either. Whereas Pascal was originally designed for educational purposes and was simple and consistent, Java is designed for professional use, uses cryptic notation that is not always consistent, and contains versatile class libraries that are too complex for novice use [2]. It is symptomatic that in a learning object designed to teach the concept of arrays [8] the tasks under the heading "test your understanding" require knowledge of Java syntax details rather than real understanding of the array concept. Similarly, in a Java-based CS1 course for academically diverse students [9], the purpose of many programming assignments is to introduce students to the functionality of components in a specific Java library rather than to promote understanding of object-orientation.

Even simple notational issues may be problematic for novices. For example, in Java, semicolons are normally used to indicate sequence of execution, e.g., "`i=0; i++;`". But in the case of for-loops such as "`for (i=0; i<=10; i++) ...`" semicolons do not imply the execution of all three parts at each round, which has caused problems for novices in understanding how the for-loop works. Educational languages can be designed to avoid such problems, e.g., in Pascal the above construct reads "`for i:=0 to 10 do ...`" which avoids the mental overloading of semicolons. When professional languages are used in educational settings, such issues with certain language constructs require extra care from teachers when the constructs are first introduced.

There is a considerable amount of evidence that novices learning programming have severe problems in understanding the basic concepts of programming (see [10] for a review). For example, the notion of a variable has been proven to be hard to understand; basic control structures like iteration are often misunderstood; and even the use of special notation like semicolons poses problems. A study conducted in four universities [11] concluded that "many students do not know how to program at the conclusion of their introductory courses" and that "many students have not even acquired the technical skills needed for getting a program ready to run". It is no wonder that students have faulty mental models concerning objects, attributes, and methods [12, 13], when their mental models of much simpler structures like variables and basic control structures are often fragile [14] and faulty.

A classic overview of programming pedagogy [15] notes that "One wonders, for example, about teaching sophisticated material to CS1 students when study after study has shown that they do not understand basic loops" and stresses the importance of teaching valid mental models because "if the instructor omits them, the students will make up their own models of dubious quality". The objects first approach tries to provide valid mental models of object-oriented design, class hierarchies, etc to novices, yet often prohibitive to master for those who do not understand the basic building blocks of programs.

3 The Alternative Approach

The objects first approach tries to avoid the negative transfer effects by avoiding teaching procedural programming before object-orientation. At the same time it sacrifices concreteness of programs and uses abstractions that are overly hard to root on novices' existing knowledge [16]. There have been recent suggestions to overcome this problem by, e.g., using various tools to help novices learn to program [17]; starting with the use of ready-made classes [18] (perhaps in a single static method[19]) and postponing class definitions to a later phase; and starting with a traditional procedural approach using static methods only [20]. The developers of these tools and course syllabi usually report improved student satisfaction and in some cases also improved learning outcomes.

In this paper we suggest an alternative solution that starts with conceptually simple—yet essential—concepts: the concept of variables, their responsibilities (like keeping track of the number of input items), and the implementation of responsibilities by assignment statements and basic control structures (e.g., conditional, iteration). Only then would the concepts of objects and attributes, their responsibilities, and the implementation of responsibilities by methods be efficiently introduced. This "variables first, objects then" approach starts with more concrete concepts than the objects first approach does, yet independent of the introduction of procedural programming. Furthermore, the same basic ideas of responsibility and its implementation are applied twice: first to variables (they deliver, keep, or update their contents in designated programming processes) and then to objects (they deliver, keep, or update their attributes through the service methods).

The responsibilities of variables can be treated by the recently introduced notion of "roles of variables" [21] that consists of a small number of roles like "stepper" (a role covering the notion of counting items), "gatherer" (a role covering the notion of accumulation) etc. Roles give a vocabulary for responsibilities of variables and provide a sound basis for the variables first part of our new approach. Roles belong to experts' tacit programming knowledge [22] and their use in teaching elementary procedural programming has been found to enhance learners' programming skills [23, 24]. In object-oriented programming, roles apply to attributes, local variables and method parameters [25]. Thus the transition from variables to objects is conceptually more manageable, requiring only a shift in the abstraction level.

The set of basic control structures introduced in the variables first part should include sequence, selection, and iteration but other control structures common to both object-oriented and procedural programming may be considered, also. For example, exception handling and concurrency (or multithreading) are central concepts in object-orientation [26] and their basic ideas can be introduced apart from objects and classes. Similarly, for the introduction of the common parts of methods and procedures we suggest using routines that cover parameter passing and recursion but that will not be used for structured design as such. Routines are motivated by the need for clarity ("separate different parts of the code") and code re-use ("define once—use several times") that are required both in object-oriented and in procedural programming. In object-oriented programming, routines turn into constructors and methods whereas in procedural programming they turn into hierarchical procedures and functions.

It is suggested that an educational situation might be better served by a language specially designed using pedagogic principles [27]. The variables first part of the suggested new approach may be benefited by using a mini language in order to avoid the side-effect of the syntax complexity often seen in industrial-standard languages. Once the programming constructs are mastered, transition to a real-world language should be no different than transition to driving a different kind of car with more bells and whistles. That said, an appropriate subset of an industrial-standard language can usually be chosen to minimize the syntax complexity should such a language happen to be adopted.

The suggested approach provides students valid models of basic programming knowledge that can be applied both in object-oriented and procedural programming. It does not stress programming language features or any specific design technique to model a programming problem. Thus problems of any specific perspective to programming [28] can be avoided. The variables first part of the approach is intended to give a good understanding of programming constructs whereas the design and composition of larger programs—including notions such as encapsulation, inheritance, and polymorphism—is postponed later in the curriculum.

4 Comparison with CC2001

ACM Computing Curricula CC2005 [29] provides an overview of the different kinds of undergraduate degree programs in computing that are currently available and for which curriculum standards are now, or will soon be, available. The oldest one is CC2001 that was published by ACM and IEEE-CS and intended to provide curriculum guidelines for degree programs for the various computing disciplines. Curriculum guidelines for more specific domain areas are IS2002 published by the information systems community, SE2004 published by the software engineering community, and CE2004 published by the computer engineering community. Because CC2001 is the most general one, we compare our approach with it.

ACM Computing Curricula CC2001 [5] offers several approaches to introductory courses: imperative-first (which could be better called "procedures-first"), objects-first, functional-first, breadth-first, algorithms-first, and hardware-first. Each of these approaches consist of two courses (with an alternative three-course implementation in some approaches). In the same vein, our approach can be termed variables-first.

The course contents suggested in the previous section is not the same as the imperative-first approach of CC2001 which introduces the whole traditional procedural model; our suggestion avoids intentionally the procedural approach to program decomposition. In fact, our approach is closer to the algorithms-first approach where the basic concepts of computer science are introduced using pseudocode instead of an executable languages and which permits students to work with a range of data and control structures. However, in our new approach we assume that the programs are executable—even if written in some educational programming language that resembles pseudocode and has simple syntax.

Our "variables-first" approach does not try to give exact implementations of two or three introductory courses. However, a possible implementation—presented in the style of CC2001—consists of , e.g., two courses: Fundamentals of programming, and Abstraction mechanisms with the syllabi of Fig. 1. The first course introduces the basic "knitting techniques", i.e., control structures like iteration, conditionals, recursion, exceptions, and concurrency; basic data structures like variables, arrays, records, and strings; and the interplay between responsibility and implementation in the form of roles of variables. The second course introduces "knitting models", i.e., various abstraction mechanisms: procedural, functional, object-oriented, and data abstraction; and the interplay between responsibility and implementation in the form of these abstractions. Other topics like software engineering and algorithm analysis are dispersed around the two courses.

CC2001 defines the core contents for computer science body of knowledge. The core consists of material that essentially everyone teaching computer science agrees is essential to anyone obtaining an undergraduate degree in this field. Core hours correspond to the in-class time required to present the material in a traditional lecture-oriented format. This time does not include the instructor's preparation time or the time students spend outside of class. Table 1 presents a comparison of the core hours of three CC2001 approaches (imperative-first, objects-first and algorithms-first) and the suggested "variables-first" approach.

5 Conclusion

ACM Computing Curricula 2001 states [5, Chapter 7] that "throughout the history of computer science education, the structure of the introductory computer science course has been the subject of intense debate [... and] recommending a strategy for the introductory year of a computer science curriculum all too often takes on the character of a religious war that generates far more heat than light." The report continues by noting that "no ideal strategy has yet been found, and

Fundamentals of Programming:

- Background: History of computing, overview of programming languages and the compilation process
- Simple data: Variables, types, and expressions; assignment
- Simple control structures: Iteration; conditionals
- Algorithms: Problem-solving strategies; implementation strategies; roles of variables
- Simple data structures: Arrays; records; strings
- Machine level representation of data: Bits, bytes, and words; binary representation of integers; representation of character data; representation of records and arrays
- Functional decomposition: Routines without parameters
- Code re-use: Parameter passing
- Recursion: The concept of recursion; divide-and-conquer strategies
- Advanced control structures: Exceptions, concurrency
- Software engineering issues: Tools; processes; requirements; design and testing; risks and liabilities of computer-based systems
- Introduction to basic algorithmic analysis

Abstraction Mechanisms:

- Principles of encapsulation: Encapsulation and information-hiding; separation of behavior and implementation
- Abstraction in procedural programming: Procedures and functions; structured decomposition
- Abstraction in functional programming: Functions without variables; recursion over lists, recursive backtracking
- Abstraction in object-oriented programming: Classes and objects; methods; message passing; subclassing and inheritance; polymorphism
- Data abstraction: Classic data structures (list, stack, and queue); procedural implementation; object-oriented implementation
- Object-oriented design: Fundamental design concepts and principles; introduction to design patterns; object-oriented analysis and design
- Using APIs: Class libraries; event-driven programming; packages for graphics and GUI applications
- Software engineering: Building a medium sized system, in teams, with algorithmic efficiency in mind

Fig. 1. Implementation of the "variables-first" approach in two courses.

Table 1. Comparison of the amount of core hours in different approaches. Imperative-first (IF), objects-first (OF), and algorithms-first (AF) are suggested by CC2001; variables-first (VF) is our new suggestion. CC2001 total core hours in parentheses.

Topic	IF	OF	AF	VF
DS5 Graphs and trees (4)	2	-	-	-
PF1 Fundamental programming constructs (9)	9	9	9	9
PF2 Algorithms and problem-solving (6)	3	4	3	4
PF3 Fundamental data structures (14)	12	11	11	11
PF4 Recursion (5)	5	5	5	5
PF5 Event-driven programming (4)	-	2	3	3
AL1 Basic algorithmic analysis (4)	2	2	2	2
AL2 Algorithmic strategies (6)	-	2	4	2
AL3 Fundamental computing algorithms (12)	6	6	6	6
AL4. Distributed algorithms (3)	-	-	-	1
AL5 Basic computability (6)	1	1	1	1
PL1 Overview of programming languages (2)	2	2	2	2
PL2 Virtual machines (1)	1	1	1	1
PL3 Introduction to language translation (2)	-	-	2	-
PL4 Declarations and types (3)	3	3	3	3
PL5 Abstraction mechanisms (3)	3	3	3	3
PL6 Object-oriented programming (10)	10	12	8	11
AR2 Machine level representation of data (3)	1	-	-	1
AR3 Assembly level machine organization (9)	2	-	-	-
HC1 Foundations of HCI (6)	-	1	-	-
GV1 Fundamental techniques in graphics (2)	2	2	2	2
SP1 History of computing (1)	1	1	1	1
SP5 Risks and liabilities (2)	-	1	-	-
SE1 Software design (8)	4	4	4	4
SE2 Using APIs (5)	2	2	2	2
SE3 Software tools and environments (3)	2	2	2	2
SE5 Software requirements and specifications (4)	1	-	1	-
SE6 Software validation (3)	1	1	1	1
SE7 Software evolution (3)	-	-	1	-
Total core hours	75	77	77	77

that every approach has strengths and weaknesses." Finally, it encourages "institutions and individual faculty members to continue experimentation in this area."

In this paper, we have studied the problems of the current main-stream approach to teaching elementary programming, objects first; and surveyed recent suggestions to overcome the problems of this approach. We have then suggested a new approach, "variables-first" that combines a strong start in basic constructs with early object-orientation. The key idea of our approach is to start with concepts that are common to both procedural and object-oriented programming, i.e., basic control and data structures and a simple form of the interplay between responsibility and implementation. Only then will various abstraction mechanisms—procedural, functional, object-oriented, and data as well as the interplay between responsibility and implementation in these abstractions—be introduced. We have also sketched a two-course implementation of this approach and compared it with the Computing Curricula 2001. In the future we plan to try this approach in real classroom settings.

Acknowledgments

This work was supported by the Academy of Finland under grant number 206574.

References

1. Astrachan, O., Bruce, K., Koffman, E., Kölling, M., Reges, S.: Resolved: Objects early has failed (Panel). In: Proceedings of the 36th SIGCSE Technical Symposium on CS Education. (2005) 451–452
2. Bailie, F., Courtney, M., Murray, K., Schiaffino, R., Tuohy, S.: Objects first - does it work? (Panel). Journal of Computing Sciences in Colleges **19** (2003) 303–305
3. Weir, G.R.S., Vilner, T., Mendes, A.J., Nordström, M.: Difficulties teaching Java in CS1 and how we aim to solve them (Panel). In: Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education ITiCSE'05, ACM (2005) 344–345
4. Ranum, D., Miller, B., Zelle, J., Guzdial, M.: Successful approaches to teaching introductory computer science courses with Python. (Special session). In: Proc. of the 37th SIGCSE Technical Symposium on Computer Science Education. (2006) 396–397
5. Joint Task Force on Computing Curricula: Computing curricula 2001. http://www.sigcse.org/cc2001 (2001) (Accessed Nov. 24th, 2005).
6. Barnes, D.J., Kölling, M.: Objects First with Java. 2nd edn. Pearson Education Limited (2005)
7. Milne, I., Rowe, G.: Difficulties in learning and teaching programming—views of students and tutors. Education and Information Technologies **7** (2002) 55–66
8. London Metropolitan University: Arrays learning object. http://www.codewitz.org/demo/index.html (2003) (Accessed Nov. 24th, 2005).

9. Comer, J., Roggio, R.: Teaching a Java-based CS1 course in an academically-diverse environment. In: Proceedings of the 33th SIGCSE Technical Symposium on CS Education. Volume 34(1) of ACM SIGCSE Bulletin. (2002) 142–146
10. Robins, A., Rountree, J., Rountree, N.: Learning and teaching programming: A review and discussion. Computer Science Education **13** (2003) 137–172
11. McCracken, M., Wilusz, T., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Ben-David Kolikant, Y., Laxer, C., Thomas, L., Utting, I.: A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In: Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education ITiCSE'01, ACM (2001) 125–140
12. Eckerdal, A., Thuné, M.: Novice Java programmers' conceptions of "object" and "class", and variation theory. In: Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education ITiCSE'05, ACM (2005) 89–93
13. Holland, S., Griffiths, R., Woodman, M.: Avoiding object misconceptions. SIGCSE Bulletin **29** (1997) 131–134
14. Perkins, D.N., Martin, F.: Fragile knowledge and neglected strategies in novice programmers. In Soloway, E., Iyengar, S., eds.: Empirical Studies of Programmers, NJ: Norwood, Ablex Publishing Company (1986) 213–229
15. Winslow, L.E.: Programming pedagogy — a psychological overview. SIGCSE Bulletin **28** (1996) 17–22
16. Hu, C.: Rethinking of teaching objects-first. Education and Information Technologies **9** (2004) 209–218
17. Powers, K., Gross, P., Cooper, S., McNally, M., Goldman, K.J., Proulx, V., Carlisle, M.: Tools for teaching introductory programming: what works? (Panel). In: Proc. of the 37th SIGCSE Technical Symposium on Computer Science Education. (2006) 560–561
18. Pedroni, M., Meyer, B.: The inverted curriculum in practice. In: Proc. of the 37th SIGCSE Technical Symposium on Computer Science Education. (2006) 481–485
19. Roumani, H.: Pactice what you preach: full separation of concerns in CS1/CS2. In: Proc. of the 37th SIGCSE Technical Symposium on Computer Science Education. (2006) 491–494
20. Reges, S.: Back to basics in CS1 and CS2. In: Proc. of the 37th SIGCSE Technical Symposium on Computer Science Education. (2006) 293–297
21. Sajaniemi, J.: An empirical analysis of roles of variables in novice-level procedural programs. In: Proceedings of IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02), IEEE Computer Society (2002) 37–39
22. Sajaniemi, J., Navarro Prieto, R.: Roles of variables in experts' programming knowledge. In Romero, P., Good, J., Bryant, S., Chaparro, E.A., eds.: Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2005), University of Sussex, U.K. (2005) 145–159
23. Sajaniemi, J., Kuittinen, M.: An experiment on using roles of variables in teaching introductory programming. Computer Science Education **15** (2005) 59–82
24. Byckling, P., Sajaniemi, J.: Roles of variables and programming skills improvement. In: Proc. of the 37th SIGCSE Technical Symposium on Computer Science Education. (2006) 413–417
25. Sajaniemi, J., Ben-Ari, M., Byckling, P., Gerdt, P., Kulikova, Y.: Roles of variables in three programming paradigms. Computer Science Education (in press)
26. Culwin, F.: Object imperatives! In: Proceedings of the 30th SIGCSE Technical Symposium on CS Education. Volume 31(1) of ACM SIGCSE Bulletin. (1999) 31–36

27. Du Boulay, B., O'Shea, T., Monk, J.: The black box inside the glass box: Presenting computing concepts to novices. International Journal of Human-Computer Studies **51** (1999) 265–277
28. Christensen, H.B.: Implications of perspective in teaching objects first and object design. In: Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education ITiCSE'05, ACM (2005) 94–98
29. The Joint Task Force for Computing Curricula 2005: Computing curricula 2005. tp://www.acm.org/education/curric_vols/CC2005-March06Final.pdf (2005) (Accessed Apr. 26th, 2006).