Introducing Java: the Case for Fundamentals-first

Jane Y YAU
Department of Computer Science, University of Warwick
Coventry, CV4 7AL, UK
janeyau@dcs.warwick.ac.uk

Mike JOY
Department of Computer Science, University of Warwick
Coventry, CV4 7AL, UK

ABSTRACT

Java has increasingly become the language of choice for teaching introductory programming. In this paper, we examine the different approaches to teaching Java (*Objects-first, Fundamentals-first* and *GUI-first*) to ascertain whether there exists an agreed ordering of topics and difficulty levels between nine relatively basic Java topics. The results of our literature survey and student questionnaire suggests that the *Fundamentals-first* approach may have benefits from the student's point of view and an agreed ordering of the Java topics accompanying this approach has been established.

Keywords: Teaching Programming, Java, Fundamentals-first and Objects-first.

1. INTRODUCTION

In recent years, Java has become the language of choice for introductory programming courses in many university computing departments. There are three principal approaches to teaching Java. The *Objects-first* approach advocates that objects should be taught right from the beginning and the establishment of object-oriented thinking is the primary focus [1]. The *Fundamentals-first* approach concentrates initially on basic concepts, before any language-specific programming, such as the object-orientation aspects of Java, in order to build students' confidence in learning, by introducing them to easier topics first [2]. Finally, the *GUI-first* approach commences by introducing students to graphical user interfaces (typically using Applets) to introduce object-oriented programming before fundamental procedural concepts. In each case, the ordering of topics presented to the student is important; simple topics should be taught before more complex topics for which they are prerequisites [3].

As part of on-going work in computer-assisted learning technologies [4], we are interested in the order in which topics in the Java programming language should be taught and their relative difficulty levels. The ordering is particularly significant in a computerized adaptive testing environment – adaptive pre-tests are a way of assessing students in minimal time to ascertain their level of understanding and to locate them at the appropriate level of instruction [5].

This paper reviews and evaluates the different approaches to teaching Java, in order to ascertain whether there exists an agreed ordering of topics and difficulty levels. To assist with this process, an investigation was carried out by conducting a literature survey and a student questionnaire. Nine relatively basic Java topics were studied: *Comments, Assignment* (including Variables and Primitive Data Types), *Expressions* (including Arithmetic Operators), *If-Statements* (including If-Else-Statements), *For-Loops*, *Arrays, Classes* (and Objects), *Methods* and *I/O* (Input & Output). *Comments* are an important part of any well-documented program; whilst *Assignment*, *Expressions, If-Statements, For-Loops* and *Arrays* are essential elements of both structured and object-oriented programming. *Classes* are the essence of object-oriented programming; *Methods* are the underlying building blocks of programs and *I/O* permits reading in and printing or outputting values.

2. APPROACHES TO TEACHING JAVA

There are three widely used approaches for teaching an object-oriented programming language such as Java; *Objects-first, Fundamentals-first* and *GUI-first*.

Objects-first

This approach concentrates on object-oriented programming principles and focuses on objects and inheritance before introducing any of the procedural elements; these procedural elements are in any case always kept in the context of an object-oriented design. Students are required to learn about *Classes* and *Methods* immediately, and then proceed to study basic procedural topics such as *Assignment*, whilst still trying to grasp the complexity of the previous topics being introduced. Objects-first is a challenging approach for learning introductory programming, since the students also have to deal with the technicalities of the syntax of the language. This approach is also contrary to the classic instruction methodology used for introductory programming, which allows a gentle learning curve by starting with a simple program and steadily moving onto more complex programming, thus allowing time for the learner to grasp each concept and incrementally build up their knowledge [1].

Object-oriented thinking, as the primary focus for the Objects-first approach, can also be established by

using the Unified Modeling Language (UML) to develop a visual and intuitive model of objects and their relationships, which will then be translated into code afterwards. Proponents of the Objects-first approach argue that object-oriented programming is a new programming paradigm which requires a new accompanying teaching approach [6].

Object-orientation is arguably not new, as its underlying concepts also existed in the general-purpose programming language *Simula*, developed between 1962 and 1967, which used modern object-oriented concepts such as classes, subclasses, and polymorphic functions. Although a procedural design does not use the same terms, notations and relationships as an object-oriented design, the underlying concepts and goals are nonetheless essentially the same, and object-orientation and procedural concepts are not mutually exclusive [7].

Fundamentals-first

Smolarski [2] maintains that students should grasp all the introductory concepts of programming before moving onto the specific technical features of the language, in this case, the object-oriented aspects of Java. An advantage of this approach is the gain of applicable foundational knowledge which will equip the students with the ability to shift to a new programming language and/or paradigm, if necessary, as they would have been "well-grounded in language-independent fundamentals". This approach is used by professional educators where the fundamental principles are taught first and when mastered students progress to design and problem solving [8].

This approach appears to be consistent with Anderson *et al.*'s *A Revision of Bloom's Taxonomy* [9], its revised framework having two dimensions which are *Cognitive Process* and *Knowledge*. The Cognitive Process dimension consists of six categories in ascending order of cognitive complexity: *Remember, Understand, Apply, Analyze, Evaluate* and *Create*. The Knowledge dimension consists of four categories: *Factual, Conceptual, Procedural* and *Metacognitive*; where the categories lie on a scale from concrete (*Factual*) which is an easier-to-attain low-level skill and abstract (*Metacognitive*) which is a harder-to-master high-level skill [8]. When comparing a basic Java topic such as *Assignment* with a topic related to design and problem solving such as *Classes,* Assignment is cognitively simpler and the knowledge in the topic seems to be more concrete; whereas Classes appear more cognitively complex, requiring more abstract and analytical thinking skills. This suggests that Assignment should be taught before Classes.

Burton *et al*. [6] point out that students who have experience in the procedural paradigm will learn object-oriented programming much more capably and effectively because the procedural paradigm consists of two main components which are algorithmic thinking and structured programming. Algorithmic thinking is also considered as a paradigm in its own right, as an algorithm can possibly consist of elements such as selection and repetition. It is argued that having a firm understanding of algorithms and structured programming before learning the object-oriented paradigm is beneficial for the students as essentially the object-oriented paradigm mainly involves modeling structure and relationships that are present in the procedural paradigm. "An object consists of a collection of variables (attributes) and procedures (behaviors) bundled permanently together (encapsulated) as a unit", hence, procedural programming must precede object-oriented programming.

GUI-first

Authors adopting this approach illustrate the properties common to all Java classes by using Java *Applets* and *Graphical User Interfaces (GUIs)* [10]. Students are taught how to develop GUI programs from the beginning to help them understand the functions of Classes and their components, and thereafter, object-oriented programming and fundamental procedural elements are introduced. This approach may lead students to think that there is more hands-on programming in their course than pure abstract theory, whether this is the case or not, and may be helpful for recruitment [11]. Students may also be more motivated and will gain more satisfaction if they can see that their running program is displaying in the form of a GUI as opposed to a static textual alternative. However, the lack of emphasis on algorithmic thinking, structured programming and object-oriented design, may deter academic staff from adopting this approach, and it has been argued that students must visualize the concepts from an object-oriented point of view from the very beginning before they do any hands-on programming [12].

3. LITERATURE SURVEY

In our first investigation we sought to discover the popularity of the three approaches, and to ascertain whether there is any agreement on the ordering of topics within Java, and 30 recently published academic Java programming books currently in print were selected. An advantage of surveying published books is that the apparent approach and the direction of the learning curve in each book can be determined quite easily because of the linear nature inherent in printed works. This linear structure also gives an indication of the relative difficulty levels of topics within it; hence an ordering of topics can be identified. Published works have all gone through a reviewing process, and many of them are recommended textbooks, and we therefore have confidence in their quality.

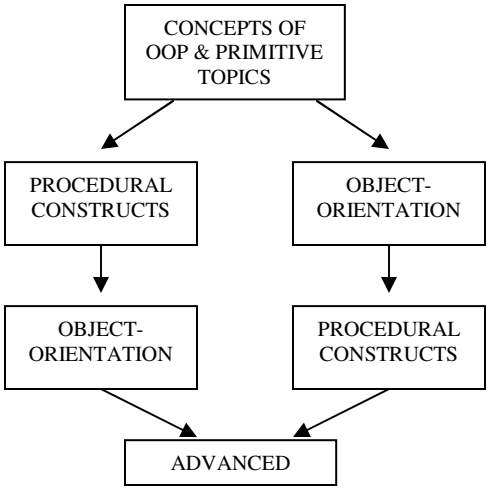
The books exhibit many differences in the ordering of topics, dictated either by prerequisite requirements or

the author's teaching style and preferences. Most of the books surveyed are aimed at students with no programming experience and are an introductory textbook, and the remainder is aimed at students who are familiar with basic programming concepts and programming, with no prior knowledge of Java.

Fundamentals-first* vs. *Objects-first

Of the texts surveyed, 18 adopt the Fundamentals-first approach, 7 are Objects-first and 5 are GUI-first, suggesting that Fundamentals-first is most likely to be adopted. The books have been divided into the three categories and investigated further. Figure 1 illustrates the two most popular approaches – Fundamentals-first and Objects-first – and their accompanying ordering of topics. The topics can be divided into five groups: *Concepts of Object-oriented Programming, Primitive Topics (Comments* and *Output), Advanced (Input), Procedural Constructs (Assignment, Expressions, If-Statements, For-Loops* and *Arrays)* and *Object-Orientation (Classes* and *Methods).*

Figure 1. Fundamentals-first and Objects-first approaches are represented by the left and right flow of events respectively.



Concepts of Object-Oriented Programming

A common feature revealed amongst these different categories of book was that the most of them (86%) introduced *Concepts of Object-Oriented Programming* at the very beginning. This is a good indication that the students should commence with learning the basic concepts before any detailed fundamentals of programming and object-oriented programming [2]. Meisalo *et al*. [13] performed a study (referred to as Study A below) on how difficult students found the various Java topics and none of the students surveyed found the following introductory topics difficult: *Algorithms* and *Basics of Java*. These *Object-Oriented Concepts* include the notion of identifying "things" (*Objects*), creating them as instances of "categories of things" (*Classes*), setting up operations for them (*Methods)* and representing information about them (*Attributes)* [14].

Primitive (Comments* and *Output), Advanced (Input)

Following the Concepts of Object-Oriented Programming, the topics chosen by most of the authors were *Comments* and *Output*, whereas *Input* is introduced last. Comments and Output require easier-to-attain low-level skills since commenting only requires *factual* knowledge, and outputting only requires a single simple statement: `System.out.println()` allowing many different primitive data types in Java to be displayed.

However, input is much more complex, and there is a collection of input methods which read in different data types such as `read()`, `readString()`, `readDouble()`, which read in bytes, strings and doubles, respectively. Standard output is comparatively straightforward whilst several declarations and conversion methods are required to set up to read numeric values for input, hence, novice students often find this complex [7]. Input also requires the student to be aware of more complex topics such as packages, abstract classes, subclasses, constructor parameters, creating and passing objects, and handling exceptions. This is because the `java.io` package must be imported; `System.in` belongs to the abstract class `InputStream`; and in order to input, an object must be created by supplying `System.in` as a constructor parameter to a subclass `InputStreamReader`; and then this object is passed to another class called `BufferedReader` as the keyboard input works best when it is buffered; as well as ensuring all `IOExceptions` are caught [15]. Hong reports encountering problems whilst attempting to teach *Input* in Java as an introductory programming language, mainly due to the syntax and the vast amount of complex details within it [16]. A study performed by Sayers *et al*. [17] (referred to as Study B below) shows that the students rated *Input* to be much more difficult than *Output*. Given these reasons, it has been decided that *I/O* should not be grouped together as one topic since one is much more complex than the other.

Procedural Constructs

After Concepts of OOP and Primitive Topics, 87% of the books introduced the Procedural Constructs followed by Object-Orientation (the Fundamentals-first approach), whilst the remaining 13% progress to *Object-Orientation* first before the Procedural Constructs (using the Objects-first approach). However, there is a common agreed ordering of Procedural Constructs for both approaches, which was revealed in the survey. 73% taught the same following linear order of topics: *Assignment, Expressions, If-Statements, For-Loops* and *Arrays,* as shown in Figure 2.

Figure 2. The Most Common Ordering of Procedural Constructs.

This ordering of topics has been supported by both Studies A and B. Study A shows the 24% of students surveyed found *Variables* difficult, a result mirrored by Study B, which reported that the students surveyed considered *Variables & Data Types* to be the easiest topic. Furthermore, Smolarski [2] suggests that the topic Data Types is simple enough to be covered early on in an introductory programming course.

The use of both an If-Statement and a For-Loop involves conditions, which are formed by *Arithmetic Operators*. These conditions are pre-determined by assigned variable values with a specific type declared. Assignments are also used by Arrays to store values. This suggests that Assignment and Expressions are prerequisite topics for If-Statements, For-Loops and Arrays. In addition, an If-Statement usually has one or more conditions which need to be satisfied; whereas a For-Loop is more complex, having a start and a stop condition and an update-part, which suggests that a For-Loop is more difficult than an If-Statement. Study B also indicates that the students rated If-Statements to be easier than For-Loop*s*. However, in Study A, 48% of the students considered *If-Statements and Logical Operations* to be a difficult topic whereas only 24% had the same opinion regarding Loops. A possible reason that these students considered If-Statements to be a more complex topic than Loops may be due to the fact that their If-Statements topic included relatively more complex Logical Operations concepts such as *AND, OR, XOR* and *NOT*.

Both Studies A and B highlighted that the majority of the students surveyed found Arrays the most difficult topic out of the Procedural Constructs; in Study A, the highest percentage of students (72%) found Arrays the most difficult. Jenkins [18] emphasized the fact that students always seem to experience difficulties with Arrays; it was suggested that one of the possible reasons may be because they were described as "*Variables that can hold multiple values*", which is not very self-explanatory.

Object-Orientation (Classes* and *Methods)

In the Objects-first approach, Object-Orientation precedes Procedural Constructs; whereas it is the reverse order in the Fundamentals-first approach. Classes also precedes Methods in the majority of the books which adopt the Objects-First approach; whereas this order was taken in only 60% of the books which adopt the Fundamentals-first and the GUI-first approaches and it was the reverse order in the remaining 40%. In Study A, 60% rated Methods to be difficult, the second highest percentage after Arrays (72%); whereas out of all the topics surveyed in Study B, students found Methods to be the most difficult, and more difficult than Arrays. These contradictory results found in the two studies may mean that there is no definite difficulty ordering between Methods and Arrays; however because Classes contains a special kind of Method called a *Constructor*, arguments can be made for teaching Methods either before or after Classes.

4. STUDENT QUESTIONNAIRE

The primary objective of the questionnaire was to compare the professionals' apparent ordering of topics in the published books with students' perceptions of the difficulty levels of these topics. Undergraduate students of various computing courses and years of study were given questionnaires to complete. The students were asked to provide details of their course, gender, age, what computing background they had, their understanding of basic concepts and programming prior to entering university, what programming and scripting languages they have studied and an indication of their perceived difficulty levels for each of the nine topics surveyed (on a scale of 1 to 10, where 1 is the least difficult and 10 is the most difficult).

Difficulty of Learning Java as the First Programming Language

Preliminary results from the survey indicate that some students who have studied an advanced computer-related subject at school, and have learnt a visual or procedural programming language before they entered university, rated the Java topics to be easier than those students who do not have previous understanding of basic concepts of programming and/or programming experience.

Hadjerrouit [12] remarks that Java is difficult for students to learn as their first programming language because the object-oriented paradigm is presented much more abstractly than the procedural one. This is also supported by Weisert [19] and Burton *et al.* [6] who argue that procedural programming must be undertaken first for students with no programming experience because many concepts which exist in both procedural and object-oriented paradigms such as algorithms, interconnected components and clean interfaces are much easier to understand if written procedurally. In contrast, critics argue that novices are able to learn the object-oriented approach much more easily than procedurally-experienced programmers, as the transition between the two paradigms is a barrier [12]. However, there is no evidence from the preliminary results of the survey to indicate that students who have studied procedural languages previously to rate the Java topics to be more difficult than the students who have not studied them.

Students' Perceived Difficulty of the Java Topics

Figure 3 shows the students' perceived difficulty levels of the various Java topics, and figure 4 shows the *arithmetic mean* of each Java topic, calculated from each of the students' ratings of each of the topic on the scale of 1 to 10 (where 1 is easy and 10 is hard). This mean is sorted to show a linear order of the students' perceived difficulty levels.

Figure 3. Students' Ratings on the Java Topics.

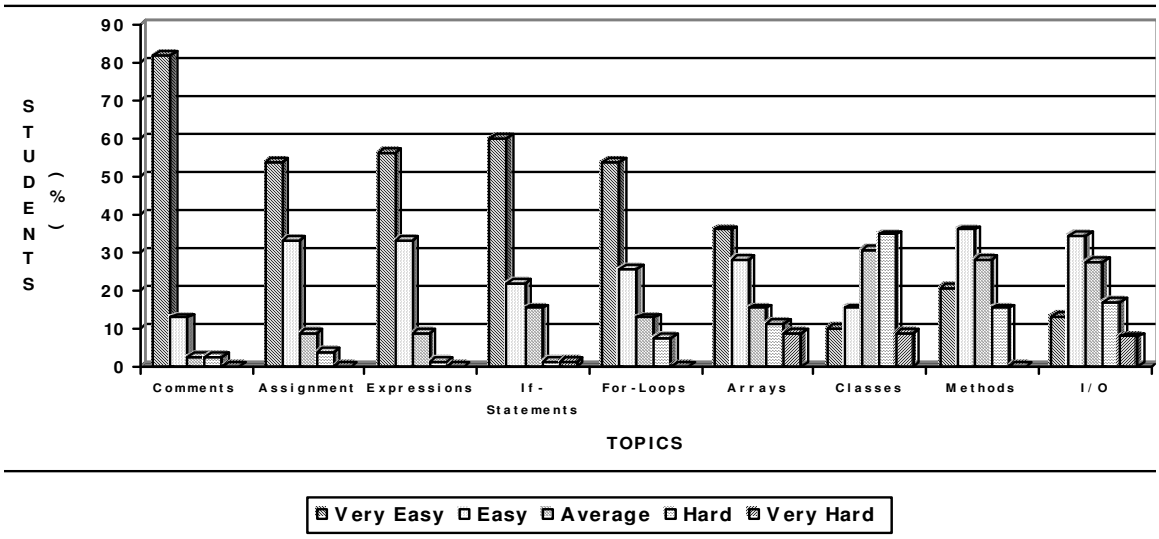
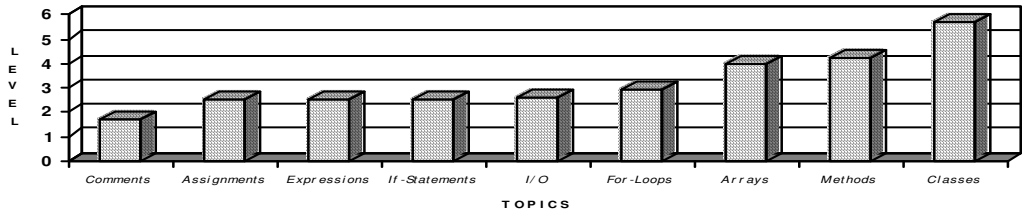


Figure 4. The Mean of Students' Perceived Difficulty Levels.



As shown in Figures 3 and 4, Classes were perceived to be the most difficult topic. It seems that the Fundamentals-first approach is a more encouraging teaching approach for the students as it starts off with the basics first and increase difficulty gradually towards Classes. Study A also revealed that 13% of the students enrolled on a first year computer science distance learning course had discontinued their studies, because they found the exercises or the theory too difficult or had failed their retakes. The Fundamentals-first approach will motivate students to pursue their course further if they are faced with easier topics to begin with and their confidence can be built up in this way.

The students' perception of the order of difficulty of the Java topics is consistent with the topic ordering in the Fundamentals-first Java textbooks surveyed. Comments are perceived to be the least difficult which suggests that these should be taught first. The perceived difficulty levels of Assignment, Expressions and If-Statements are the similar, suggesting there may not be a definite difficulty ordering between them. However, Expressions should be preceded by If-Statements because of the prerequisite requirement. Arrays are perceived to be more difficult than For-Loops which are perceived to be more difficult than If-Statements, hence achieving the consistency with the results gained from the literature survey.

Methods and Classes are perceived to be the two most difficult topics with Classes to be the most difficult of all, suggesting that Classes should be taught last. The perceived difficulty of I/O is not consistent with the results obtained from the literature survey possibly because Input and Output were grouped together therefore students were rating these two topics as a whole. The reason that the students might not perceive Input to be very difficult might be due to the way that they were taught. One of the teaching approaches for Input is to provide the students with a class or template with all the necessary declarations to set up the *InputStreamReader* and the *BufferedReader* so that the students are only expected to 'fill in the gaps' [7].

5. CONCLUSION

We have discussed the advantages and disadvantages of the Fundamentals-first and Objects-first approaches for teaching Java, and suggest that Objects-first approach is a more challenging strategy both for the teachers to teach and for the students to learn. The Fundamentals-first approach, which is consistent with the classic instruction methodology for teaching introductory programming and the levels of cognitive ability categorized in Bloom's Taxonomy, is arguably more favorable from the student's point of view, especially for students learning programming for the first time. We have considered the possible order in which topics in introductory Java programming might be presented, assuming a Fundamentals-first approach. A survey of Java textbooks, supported by a survey of students' perceptions of the difficulty of various topics, has yielded an ordering of those topics, from the least difficult to the most difficult, as follows: *Comments, Output, Assignment, Expressions, If-Statements, For-Loops, Arrays, Methods, Classes* and *Input*.

6. REFERENCES

[1] S. Cooper, W. Dann and R. Pausch, "Teaching Objects First in Introductory Computer Science", **SIGCSE Technical Symposium on Computer Science Education**, 2003, pp. 191-195.

[2] D.C. Smolarski, "A First Course in Computer Science: Languages and Goals", **University of Debrecen**, 2003.

[3] D. Callear, "Teaching Programming: Some Lessons from Prolog", **LTSN Centre for Information and Computer Science**, 2000.

[4] M. Joy, B. Muzykantskii, S. Rawles and M. Evans, "An Infrastructure for Web-based Computer-Assisted Learning". **ACM Journal on Educational Resources in Computing**, Vol. 2, Issue 4, 2002, pp.1-19.

[5] I. Arroyo, R. Conejo, E. Guzman and B.P. Woolf, "An Adaptive Web-based Component for Cognitive Ability Estimation", **University of Massachusetts**, 2001.

[6] P.J. Burton and R.E. Bruhn, "Teaching Programming in the OOP Era". **ACM SIGCSE Bulletin**, Vol. 35, 2003, pp.111-115.

[7] J. Lewis, "Myths about Object-Orientation and Its Pedagogy", **SIGCSE Technical Symposium on Computer science education**, 2000, pp. 245-249.

[8] I. Sanders and C. Mueller, "A Fundamentals-based Curriculum for First Year Computer Science", **SIGCSE Technical Symposium on Computer Science Education**, 2000, pp. 227-231.

[9] L.W. Anderson et al, **A Taxonomy for Learning, Teaching, and Assessing - A Revision of Bloom's Taxonomy of Educational Objectives**, Addison Wesley Longman, Inc., 2001.

[10] Rick Decker and Stuart Hirshfield, **Programming.Java - An Introduction to Programming Using Java**, Brooks/Cole, 2000.

[11] T.E. Gibbons, "Using Graphics in the First Year of Programming with C++", **College of St. Scholastica**, 2002.

[12] S. Hadjerrouit, "Java as First Programming Language: A Critical Evaluation". **ACM SIGCSE Bulletin**, Vol. 30,1998, pp.43-47.

[13] V. Meisalo, S. Torvinen, J. Suhonen and E. Sutinen, "Formative Evaluation Scheme for a Web-based Course Design", **Innovation and Technology in Computer Science Education**, 2002, pp. 130-134.

[14] Roger Garside and John Mariani, **Java: First Contact**, Course Technology, 1998.

[15] Judy Bishop, **Java Gently Programming Principles Explained**, Addison Wesley, 1998.

[16] J. Hong, "The Use of Java as an Introductory Programming Language", **ACM Electronic Publication**,http://www.acm.org/crossroads/xrds4-4/introjava.html, 1998.

[17] H.M. Sayers, M.A. Nicell and S.J. Hagan, "Teaching Java Programming: Determining the Needs of First Year Students", **LTSN Centre for Information and Computer Science**, 2003, pp. 106-110.

[18] T. Jenkins, "A Participative Approach to Teaching Programming", **Integrating Technology into Computer Science Education**, 1998, pp. 125-129.

[19] C. Weisert, "Learning to Program: It Starts with Procedural", **Information Disciplines, Inc.**, 1997.