ENHANCING THE INTRODUCTORY COMPUTER SCIENCE CURRICULUM: C++ OR JAVA?*

STUDENT PAPER

Andrei Irimia
Lipscomb University
Nashville, Tennessee
irimiaan@lipscomb.edu

ABSTRACT
Java is a better choice than C++ to teach introductory programming courses for three primary reasons. First of all, it is easier to learn since many programming concepts can be better taught in it rather than C++. Because the language represents an advanced step of computing technology, programming principles can be conveyed in a much easier fashion in Java than in C++. Secondly, Java's documentation is very useful and well organized. The language offers a good chance for the student to become acquainted with real-life programming situations and its nature is straightforward and friendly. Thirdly, while many features of C++ are rather confusing to the beginner, Java proposes a nice approach to many problems. Debugging in C++ can take a very long time and draw the student away from the purpose of the CS 1 course while Java debugging reinforces the concepts taught in the first course and sets the path for further knowledge acquisition.

An important issue in today's Computer Science education concerns the programming language chosen to teach introductory-level programming courses. Throughout the 1990s, some
colleges and universities in the United States and other countries have selected C++. In recent years, however, several institutions of higher education have decided that Java is a better choice. The University of Hong Kong, for example, has replaced C++ with Java in its programming curriculum in an attempt "to foster leading-edge technology innovation" [8].

The purpose of this paper is to argue in favor of the latter option by providing a comparative analysis of the strengths and weaknesses of the two programming languages from the perspective of the learning student. As a Computer Science student, I have had the opportunity to learn Java at the college level (after learning C++) as part of an honors project in Data Structures programming. As I progressed through the course, I discovered the numerous advantages that would be gained by teaching CS1 courses using Java rather than C++. In this respect, Java constitutes a better option for three primary reasons. First of all, Java is easier to learn while C++ is overly complex. Features such as type matching and fault tolerance also tend to be more efficient in Java, which makes this language easier to use. In addition to these, Java has several "built-in" extensions that would allow students to do more complicated and useful programs (GUI, DB, multithreading, etc.). Secondly, it is easier to introduce advanced programming concepts in Java; studying C++ involves dealing with many theoretical aspects that are specific to this language and therefore not as important to the Java user. An extensive coverage of these features is not needed in a Java course, while in C++ this is absolutely necessary. Thirdly, Java has better documentation—which is in perfect tune with the latest trends in the field of programming—and offers a lot more practical learning opportunities than C++ does.

I concluded that it is much easier to work with Java than with C++ at the very beginning of my course experience.

According to Computer Science professor Allen B. Tucker, the decision concerning the language of an introductory course in programming should "take into account the requirements of industry while maintaining a commitment to sound pedagogical practice" [6]. In addition, it is also relevant to mention the fact that "the College Board (...) sees the programming language as merely a vehicle for presenting concepts, not the end goal of the course (...). Moreover, the Board is anxious to emphasize that its AP course is a rigorous curriculum for computer science majors, going far beyond simple language instruction to focus on such topics as computer-based solutions to problems, algorithms and data structures, design and development" [7]. These two arguments are very important from the point of view of how efficiently CS1 courses are taught. As I look back to the time when I took Introductory Programming as a freshman in college, I can see more clearly how relevant it is to learn a programming language with the specific purpose of understanding data structures and computer-based problem-solving techniques rather than the limitations of interminable debugging and syntactical traps of all kind.

One of the most important challenges of teaching introductory programming courses involves the fact that "it is critical to establish a ... responsive process that would allow the integration of new [Computer Science] principles and artifacts soon after they emerge" [4]. How can this be more easily accomplished using Java rather than C++?

First of all, Java expresses the latest steps in the evolution of programming better than
C++ does. As Professor Lea of the State University of New York at Oswego states, "Java supports a minimal set of constructs that together reflect the bulk of (...) advances in programming methodology that have occurred over the past fifteen years or so.... Pascal, Unix and Java all arose as simpler, more understandable, and more practical alternatives to existing technologies... Each contributed new ideas, and ultimately served as the technical substrate for further theoretical and technical advances" [4]. From my own viewpoint, this perspective is very relevant because the ease with which a college student could learn Java constitutes a potential advantage in the process of concept assimilation that is involved in learning data structures and program design. This issue can have tremendous importance because it suggests that the CS1 student can learn college-level programming better and at a more rapid rate.

Viewing the problem from this perspective, one may argue that choosing Java as the language of an introductory course in programming might be a hasty step taken at the impulse of industry necessities. After all, some suggest, "[d]ifferent institutions have different educational priorities and constituencies and must therefore shape their programs accordingly. Some are driven more by industry's needs, whereas others are driven more by the general goals of liberal arts and science education" [6]. Judging Java from that perspective, it is useful to take into account three major arguments that Prof. Lea proposes. First of all, "nearly all [Java] programs are made from parts, most of which are not written by the program author. Locating, understanding, and using external components are valuable, demanding technical skills in which CS students need training and experience" [4]. In my opinion, this argument has its own relevance because Java can thus offer a good possibility to the student to become better acquainted with real-life programming tasks. These are very necessary for many, not only in order for them to catch a glimpse of how their field of knowledge can operate in practical circumstances, but also to make career decisions. In Dr. Lea's opinion, teaching Java offers the opportunity to lay emphasis on the magnitude of such issues, which have become increasingly important in the latest years. The second argument is the fact that "[a]t all levels of scale, most software that people use is reactive-programs responding to UI events, incoming phone calls, incoming stimuli of all sorts (...). Java provides a simple and fun entry point into reactive programming via Applets" [4]. This important feature of Java's practicality can be seen as a positive feature since "one indication of obsolescence in the curriculum is the fact that many important topics have yet to find a niche in the core curriculum offered at many institutions" [6].

Many keywords that make Java code intelligible do not exist in C++; this is usually what makes Java syntax much less complicated than that of C++. As Prof. L. F. Johnson puts it, "the complexity of C slows the student's study of programming concepts. The neat aspects of C distract [it]" [2]. For example, inheritance is declared in Java using the keyword "extends", while in C++ the same purpose is accomplished by using a colon followed by the class section to be inherited and then by the name of the parent class. The keyword "extends" makes more sense here because a beginner can easily create a subclass by remembering that a child class "extends" the attributes and content of the parent class. What I was able to conclude after my Java programming experience is that, many times, C++ programs require much more research into the structural matters of the language than Java applications ever do. Remembering C++ syntax can take a while due to its abstract nature compared to Java's, which can make more sense to
the CS1 student and therefore constitute one less obstacle to overcome.

As it was suggested at the beginning, teaching programming in C++ involves the necessity of discussing many aspects of this language that either do not exist in other programming environments or that are of secondary importance. This can become more obvious when a comparison is drawn between C++ and Java. In C++, for example, an important area covered in introductory courses is that of variables that store addresses, commonly called pointers. As the student goes further into the study of advanced C++ features, understanding reference variables becomes essential in order to grasp how data structures such as linked lists, trees and graphs are implemented. When advanced pointer notation is introduced, good knowledge of reference variables becomes even more important. Writing about the arguments against introducing C in the CS1 course, L. F. Johnson remarks that C "is a step backward, because it is actually the modern assembler language, and we now use assembler language mainly for reasons of technology limitations" [2]. As a conclusion, the author adds, "the whole thrust of programming languages and of software engineering is to move away from the obsolete assembler approach to software construction" [2].

In Java, on the other hand, due to the manner in which storage locations are created, a variable can be thought of as the equivalent of object pointers in C++. This means that the Java instructor does not have to teach pointer notation: Java variables can be used as pointers. The code of most data structures covered in advanced programming courses is therefore much easier to understand in Java. Most of these structures' implementations, including linked lists and trees, can be more straightforward in Java, which usually means less programming time and increased potential of usage. In C++, pointer errors are very frequent. In Java, "the run-time system will reliably generate a run-time error, instead of producing random results. [One doesn't] worry about memory management" [1]. As a Data Structures student, I learned that many concepts related to pointers could often be more straightforward when they are taught in Java. Due to the nature of Java variables, the Elementary Programming student can understand issues related to pointers more successfully because all these concepts are not obscured by the complexity of C++ syntax and by the memory management philosophy that lies behind it.

The C++ programming language was developed on the skeleton provided by C, which was a procedural-oriented language. This means that almost all the features of C had to be accommodated in the new environment, which still leads to confusion and ambiguity. An int in C and C++, for instance, can mean a 16-, 32-bit or even other size-variable-depending on the operating system-while a Java int always has 32 bits. Because of its structure-oriented background, C++ also allows conversions between some data types such as Boolean values and integers, and numbers can still be used in test conditions. This can sometimes result in hard-to-find bugs. As I studied Java, I found its data-type nature much more structuralized that that of C++. This feature could be of great help to a CS1 student because it would facilitate her/his understanding of how data types fit into the general structure of the language. In addition to this aspect, good data-type checking would eliminate the confusion that beginners can face when working on their programs.

Another significant improvement of Java is that its designers have eliminated C++'s
multiple inheritance, which should not even be used according to some programmers. What is offered instead is a new notion called interface that was derived from Objective C and that has turned out to be far more efficient. Interfaces remove the complexity that results from the management of multiple inheritance hierarchies and provide a friendly option for the programmer [1]. As a result of how C++ was created, its code is many times obscure and class implementation can turn out to be much more challenging than in Java. "The designers of Java", Computer Scientist Cay Horstmann explains, "thought hard about what makes C++ code so buggy. They added features to [the language] that eliminate the possibility of creating code with the most common kinds of bugs" [1]. In a CS1 course, "knowledge of the language is a useful by-product and for some the real purpose of the course. What [instructors] really want is students to be prepared to learn new languages as needed. The reality", it is added, "is somewhat less. Unfortunately, the first course sometimes (quite often) degenerates into a desperate struggle by students with the language just to get programs to run" [2].

I agree with this statement for two basic reasons. First of all, I have noticed that many CS 1 students tend to associate the code needed for a program with the programming concepts taught in a manner that disregards the true importance of the concepts. This can be a pitfall because such a situation can pull the student away from the purpose of the course towards the conception that getting a program to work with the right code is the real goal in sight. For the reasons above, this can be accomplished in an easier way with Java and the process of learning can be significantly eased. Secondly, students who tend to disregard the importance of programming concepts per se also tend to have greater and greater problems as they continue their study of programming and computers in general. Getting "the right code" into a complex program is not an easy task for a student who looks at programming from the wrong perspective, especially when that perspective does not involve a good sense of how to apply programming concepts that should have been acquired earlier. This danger can also be more successfully avoided by teaching Java. I have seen quite a few cases of students in my programming courses that could not understand why it is not right to just sit down and write a program in any way that would eventually work and why good memory management and a solid programming framework are so important. In Java, the structure of the language does not only show the student why to view programming from the right angle, but also make her/him to write programs with the right concepts in mind. The educational advantages are thus manifold.

Another useful detail of Java is that there is no need of destructor, release or delete functions: a Java object called the garbage collector takes care of any un-referenced variables by releasing their memory space to the operating system. No function is thus needed to accomplish the purpose of the C++ destructor and the Computer Science instructor need not spend any time discussing it either. Besides destructors, no header files, structures, overloaded operators, or virtual parent classes are needed in Java.

A great advantage of this programming language is that debugging and testing Java programs usually take less time than in C++. As Cay Horstmann puts it, "...Java really does make it easier to produce bug-free code. It does this by eliminating many of C++ more bug-prone features such as pointer arithmetic, manual memory allocation, and null-terminated arrays of chars" [1]. One example is the use of the >> operator, which can be interpreted by
a C++ compiler either as a logical or as an arithmetic shift operator; in Java, however, this ambiguity has been removed through the use of the >> >> operator, which fills the top bits of a byte or word with 0-s, while >> extends the sign bit into the top bits.

Many other examples abound. In Java, there is no possibility of error resulting from the use of the = sign when one means = =. Java variables cannot be redefined inside a nested block, which eliminates another source of errors. Finally, the [ ] operator in Java is defined to perform bounds checking, and no pointer arithmetic is allowed as in C++. In his article on the harmfulness of C in the introductory course, L. F. Johnson draws the attention to the fact that "introducing C into the first course conflicts with [the purpose of teaching problem solving] because students end up solving the problems of C instead. The result is the teaching of debugging before they have anything useful to debug" [2]. From my perspective as a student, C++ debugging is quite different from Java debugging because it is much harder to apply a concept learned in order to find the cause of a debugging problem in C++ than it is to do the same thing in Java. C++ bugs can often involve much more than is apparent to the inexperienced programmer; in addition, Java error messages tell more to the student, which makes it easier to remember the programming concept that should be applied in order to eliminate the error.

An argument against Java that is quoted rather often is the fact that C++ will most likely remain the more popular language for quite a while. C++ has indeed been very useful to network developers and Windows programmers; Java, however, has some strong features of its own, such as true portability and the ability to create applications using drag-and-drop technology, which implies that a Java application can take considerably less programming time. As Don MacVittie states, "Java ... lets developers quickly write top-level code that is not time-sensitive, and still drop to C or C++ through the JNI. [...] C and C++ code can be used where it makes the most sense: in highly technical, difficult-to-reproduce pieces, or time-sensitive sections" [5]. It seems then that Java would be a better choice as the language of CS1 since the circumstances above are quite unlikely to occur in an introductory programming course.

The advantages offered by the object-oriented approach of Java are also obvious from other perspectives. One of the useful features of this programming language is that its API documentation is easily accessible on the Internet, which means that students can make extended use of Java's built-in functions and classes. This is not as easy in C++; many inexperienced students can lose themselves among the multitude of documents available in the MSDN library; what this confusion usually results in is frustration and loss of time. With Java, accessing the API documentation is easy and instructive; the student is not only likely to find the information sought, but also to learn much more about the built-in capabilities of the JDK. As a Data Structures student learning Java, I found the API documentation a very useful and well-organized tool, which would have been even better had there been a similar one in C++.

The Java student has the practical advantage of being able to design numerous applications for the Internet, which can turn out as a very useful background for an Internet design class. Experience with Java can also be helpful for a database course, given that there are already quite a few application servers-such as Weblogic's Tengah- that are built entirely
in Java. For some colleges, these can be key arguments since more and more students taking introductory programming courses come from a variety of backgrounds with respect to their interests, nature of their future employment, etc.: "[t]he interests of non-majors in the field of CS&E are increasingly strong and sophisticated. These interests ought to be recognized by the development of new course materials and effective teaching strategies" (Kelemen in [6]).

At many universities throughout the world, teaching CS1 using Java has also turned out to be a good alternative. At the College of New Jersey, for instance, "in a single semester students have created graphics drawings of their own design, written advice-giving expert systems, generated calendars, and have written interactive games such as concentration, score four, and scrabble as final projects" [3]. In addition, Java's modular approach seems to be appreciated by students and writing GUI-based code is also enjoyed. Besides exhibiting basic understanding of events and multithreading, many students seem to embrace Java components, especially AWT. At SUNY-Oswego, students find Java to be "enjoyable, sometimes even fun," and definitely "a valuable real-world skill in itself" [4]. It is even surprising, as the course instructor states, how "students so readily adapt to the notion of building reactive, distributed programs on the internet, and how this, as much or more so than the object-oriented aspects of Java programming, so fundamentally governs their attitudes on what programming is all about."[4]

All these factors call for the adoption of Java as the programming language of the college curriculum. As it has been shown, its advantages do not only emphasize the significant amount of time that an instructor can save by teaching Java, but also the considerable number of practical skills that a good knowledge of this environment offers to the student. Java is platform-independent and contains classes capable of generating GUIs. It supports multi-threading and, quite important, it can make CGI programming easier and server-side processing "extremely efficient" [1]. Its security mechanisms are very robust and some specialists even believe that it will revolutionize client-server computing [1]. The friendliness of Java and its visual capabilities constitute a powerful reason to integrate it into the college curriculum.

REFERENCES


[8] Scott, Megan. "Java in, C++ out at University. (University of Hong Kong)" Computerworld, June 8, 1998 v32 n23 p57 (1).