A Comparative Analysis Of C# And Java As An Introductory Programming Language For Information Systems Students

Ihssan Alkadi, (Mail: ixa6880@louisiana.edu),University of Louisiana, Lafayette
Ghassan Alkadi, (mail: galkadi@selu.edu), University of Louisiana, Lafayette
Harlan Etheridge, (Email: harlan@louisiana.edu), University of Louisiana, Lafayette

ABSTRACT

Since the introduction of the C# programming language by Microsoft, the decision to teach C# or Java or both as the introductory language for Information Systems (IS) majors has always been, and continues to be an ongoing predicament. The purpose of this paper is to submit analyses based on a comparison on which language is better as the primary language taught as an introductory language for IS majors. In this paper we intend to show an answer to the question: "C# or Java, that is the question!" Each language has pros and cons for use in an introductory programming language course; however we make a suggestion as to which is the most optimal choice.

INTRODUCTION

In January 2000, Microsoft and Sun Microsystems settled a 1997 lawsuit over Microsoft's use of the Java programming language. Sun had alleged that Microsoft had violated the terms of its Java license by modifying the language so as to make it impossible to run programs written in Microsoft's version of Java on any operating system other than Windows. Microsoft agreed to pay Sun $20 million, terminate its Java license, and cease advancing its own version of Java (Visual J++) beyond the pre-existing version 1.1.4.

But soon after the legal battle concluded, Microsoft surprised the technology industry by ending support for Java technology and introducing its own C# programming language as an alternative to the Java language.

Java has been growing in popularity among software developers for six years. It has quickly grown beyond its original scope of a mere web-page enhancement utility to a fully functional software development tool. Analysts predict that Java will soon replace Visual Basic as the most popular programming language in the tech world. Microsoft, on the other hand, is the most powerful software company in the world, and its Windows family of operating systems is deeply rooted as the most popular operating systems among end-users. Consequently, Microsoft and Sun are gearing up for serious competition between their programming languages. This paper addresses the similarities and differences between Java and C#. Both Java and C# are object-oriented programming languages with syntax and functional specifications very similar to those of C++. All classes in Java and C# descend from a root class named "object." Both languages support single-inheritance from classes, but multiple-inheritance from "interfaces," a concept introduced to Object-Oriented Programming (OOP) with the release of Java. Both languages utilize a "garbage collector" to ease the burden of memory management on programmers.

OVERVIEW

Both Java and C# offer much functionality to make life easier for programmers. For example, do you need to know the length of an array or string? To do this in C# or Java, there is no need to write your own functions or search through a library of header files as you would in C++ or C. Many commonly needed functions, especially functions designed to provide information about the current state of objects, are built in to Java and C#. Any C or C++ programmer knows the importance of being aware of garbage in newly allocated arrays, and of the bounds of arrays when traversing them. Java and C# programmers need not concern themselves with such details, since these languages automatically initialize arrays (and all other objects for that matter) upon allocation, and have built-in bounds-checking for arrays and strings that will throw run-time-exceptions if an attempt is made to access an out-of-bounds element in a string or an array.

The source code for both languages is compiled into an intermediate, lower-level byte-code, which runs in a "managed execution environment." In the case of Java, the source code is compiled into Java byte code, which can be executed on any platform via the Java Virtual Machine (JVM) for that platform. In the case of C#, cross-platform portability is not the goal; rather, multi-language integration is the purpose of the Common Language Interface (CLI). The CLI compiles source code from several programming languages, from C# to C++ to Visual Basic and so on, into an assembly-like language that can be executed in the .NET environment. The end result is that different programmers can write source code in the language of their choice, and still merge their code together via the CLI into a single program or software system. While Sun has made it easier for Java programmers to create programs for virtually any platform, Microsoft has made it easier for programmers of virtually any language to create programs for the Windows platform.

Many of the differences between Java and C# are merely cosmetic. The most obvious syntactical differences are that certain keywords in C# are capitalized while the same keywords in Java are not. All method names in C# must be capitalized; the reverse is true in Java.

Other differences are more significant. While C# allows operator overloading, a feature that many C and C++ programmers would be unwilling to live without, Java will not allow this level of functional customization. In Java, traditional primitive data types (such as integer, floating point, and character) remain primitive. That is, they are not actual classes, but atomic data structures. Not so with C#. Everything in C# (yes, even the simple integer) is a class, complete with its very own set of attributes and methods.

A notable difference is that Java, unlike C++, does not have destructors to explicitly deallocate memory resources. Java relies on its garbage collector for this. Instead of destructors, Java has "finalizers," which essentially mark a resource for garbage collection, but exactly when the resource will be deallocated depends on when (during runtime) the JVM calls the garbage collector. C#, like C++, does indeed have destructors, but they really just act like Java's finalizers. They do not deallocate resources when called; they simply mark the resource for garbage collection. When all is said and done, memory management is probably the area in which Java and C# differ most.

Perhaps the most noted difference in the rival languages is the issue of pointers—that most controversial tool of memory management. Pointers have strong supporters and strong detractors. Proponents of pointers will find C# more accommodating than Java, but not much more. Java is famous for its absence of pointers. C#, to a limited extent, allows pointers to be used in methods or code-blocks identified by the "unsafe" modifier. The use of pointers in a system that is already managed by a garbage collector presents some unique situations for programmers. While the garbage collector is generally free to move objects around in memory as necessary during the garbage collection process, C# provides the "fixed" keyword to tell the garbage collector to leave an object alone. This fixes in memory the memory location(s) of the object(s) being pointed to.

There may be no runtime effects at all from using pointers in C# that is as long as the garbage collector does not run at the same time that pointers are being used. If pointers are allocated, used, and deallocated before the garbage collection process begins, then as far as the garbage collector is concerned, the pointers never existed and it will do its job as usual. However, if the garbage collector begins its process while there are active pointers in memory, the garbage collector will recognize the fixed keyword, and like the angel of death to Moses, it will essentially "pass over" the pointers and their values. This occurrence may result in fragmentation of the heap, if the garbage collector moves all non-fixed objects to a new memory area, while leaving behind the fixed objects in the "old" memory area. But even in the event of heap fragmentation, the result would merely be decreased performance of the program—not an out-right crash.

SUMMARY

The syntax and functional specifications of Java and C# are more similar than different. A programmer accustomed to writing in either language could almost effortlessly adapt to the other. Because both languages are descendants of C++, C++ programmers could also move easily between projects utilizing any of these programming languages. In a single weekend, a Java or C++ programmer can learn the unique characteristics of C#, and be prepared to start a job as a C# developer by Monday. In the end, there is little to distinguish C# from Java.

Consequently, due to the similarities of C# and Java, the predominance of the Windows operating system in the world of business, and the ease of creating programs for Windows with C#, we recommend that C# be the programming language to which IS majors are first exposed. Not only will this give IS majors a head start in creating Windows-based applications, but also will give them a strong background in OOP. Also, those students who wish to develop Java programs will easily be able to do so, given their exposure to C#.

NOTES

NOTES