Business and Scientific Applications of the Java Programming Language

Angelo Bertolli

April 24, 2005

Abstract

While Java is arguably a good language with that to write both scientific and business applications, the Java programming language was designed with certain features and concepts that often make it better suited to business applications than science applications. This paper attempts to analyze the strengths and weaknesses of Java with particular regard to business applications such as a network database front-end or access interface, and scientific applications that involve matrix manipulation.

1 Introduction

> It is the challenge of the programming language design to achieve the power, expressiveness, and comprehensibility that human readability requires while at the same time retaining the precision and simplicity that is needed for machine translation. [1]

Java is a popular object oriented programming language developed in 1991 by James Gosling at Sun Microsystems. [2] The goals in creating Java were to use the object oriented paradigm, to let the same program run on different platforms, to create an infrastructure for networks, and to allow secure execution of foreign code. "Java technology must enable the development of secure, high performance, and highly robust applications on multiple platforms in heterogeneous, distributed networks." [3]

Human readability is achieved through abstracting the functional details of the computing machine. This is helpful because it reduces the overal complexity presented to the programmer, as "a human being can only retain a certain amount of detail at once." [4] The programmer is able to concentrate on the problem being solved instead of thinking about the system he is programming.

This paper tries to evaluate the design theories behind Java with regards to scientific and business applications, in particular: a business application such as a front-end or access interface to a network database, and a scientific application that uses matrix manipulation such as gaussian elimination, matrix multiplication, kriging, etc. While this paper tries to refrain from making evaluations based on the performance of other programming languages, it is an inevitable and necessary comparison.

[1] Louden (1993)
[2] Wikipedia (2005)
[3] Gosling and McGilton (1997)
[4] Louden (1993)

2 Java Language Design

Java is a purely object oriented programming language. All methods or subprograms are defined in classes; Java does not support procedural programming. It was orginally designed as a solution to programming embedded systems for consumer electronics. In an interview, Gosling described the design goals of Java to be productivity, reliability, security, and portability. In addition, Java had to have built-in network support, and includes the ability to create and control multiple threads. "Java is now widely used in a wide variety of different applications areas." [5]

2.1 Portability

Java's most notable design feature is the Java Virtual Machine. Instead of generating machine executable code, the compiler generates bytecodes: "a high-level, machine-independant code for a hypothetical machine that is implemented by the Java interpreter and run-time system." [6] This means that a Java program compiled to bytecode can be run on any system that has the same version of the Java Virtual Machine.

This has lead to Java being a popular choice for web programming, because programs compiled to bytecode can be transferred over the Internet quickly and allows for client-side web implementation instead of server-side services. Java remains to be one of the best tools for web programming due to its portable design.

However, due to the high expectation of portability Java implies, it can sometimes be disappointing. Sometimes the target machine doesn't have the same version of the bytecode interpreter that causes the program to behave differently. Sometimes the look and feel is not the same in graphics and toolkit libraries such as AWT or Swing on different machines. Java could be considered "write-once, test everywhere," [7] but the modifications to the code to enable cross platform execution are minor at most.

2.2 Memory Management

One feature of the Java programming language is known as garbage collection. Garbage collection removes "the need to deallocate memory explicitly from the programmer," [8] as references are automatically dereferenced when appropriate. Memory leaks can still occur if the programmer does not dereference an object, but this is at a higher conceptual level.

All objects in Java are allocated onto the heap. There is no primitive array construct available to programmers. Instead arrays are instances of a predefined class.

2.3 Security & Reliability

For security reasons Java does not provide pointers to memory locations, but references: "hardware-specific data types and pointers to arbitrary memory were deliberately omitted." [9] This can harm performance as these must be accessed through another interface (such as C libraries).

[5] Sebesta (2003)
[6] Gosling and McGilton (1997)
[7] Garfinkel (2001)
[8] Louden (1993)
[9] Wikipedia (2005)

Java claims to be network secure, and can securely execute code on remote machines. This was important in addition to making Java architecture-neutral for "distributing dynamically extensible software across networks." [10]

Additionally, Java enchances security and reliability by allowing coercion only to wider types, and does index range checking on arrays.

2.4 Performance

Java has a poor reputation for performance, despite the fact that newer implementations use "just-in-time compilation" and "dynamic recompilation" that have improved the speed of Java execution over its original incarnations. For a specific application, Java's performance is "difficult to predict due to dynamic compilation and garbage collection." [11] Typically programs written in Java use more memory than other languages. This is most likely due to the heavy usage of objects that require more overhead for maintaining, and to the fact that it must keep track of memory allocation (again for garbage collection).

However, the goal of Java was not to out-perform other languages that are compiled to machine executable code. Instead it was meant to be a "middle ground between very high-level and portable but slow scripting languages and very low level and fast but non-portable and unreliable compiled languages." [12] [NOTE: "Very high-level" means interpreted languages like Perl or Python, while "very low-level" means compiled languages such as C or Fortran.]

2.5 Other Notes

Java maintains a C and C++ like syntax, but removes function calls and requires every code block to be contained inside an object. Because Java is almost context-free, and highly structured, it is arguably less writable than languages like C and C++. Java is unforgiving with type casting and doesn't allow operator overloading, both of which hinder the programmer's ability to write code quickly.

> The creators of Java tried to make a better C++. But they ended up with a language that is ugly, hard to read and that requires an inordinate amount of typing because of a variety of pedagogical restrictions imposed by Java's creators. [13]

Java has some other idiosyncrasies worth mentioning that may affect programming business and scientific applications:

- Arithmetic expressions in Java cannot be used for control expressions.
- Java only supports single inheritance, but allows some of the same features as multiple inheritance through the use of interfaces. Interfaces are much like abstract classes that have only methods. They define methods that an object is guaranteed to have if the definition of that object implments that interface.
- Java enables easy creation of threads.

[10] Wikipedia (2005)
[11] Wikipedia (2005)
[12] Gosling and McGilton (1997)
[13] Garfinkel (2001)

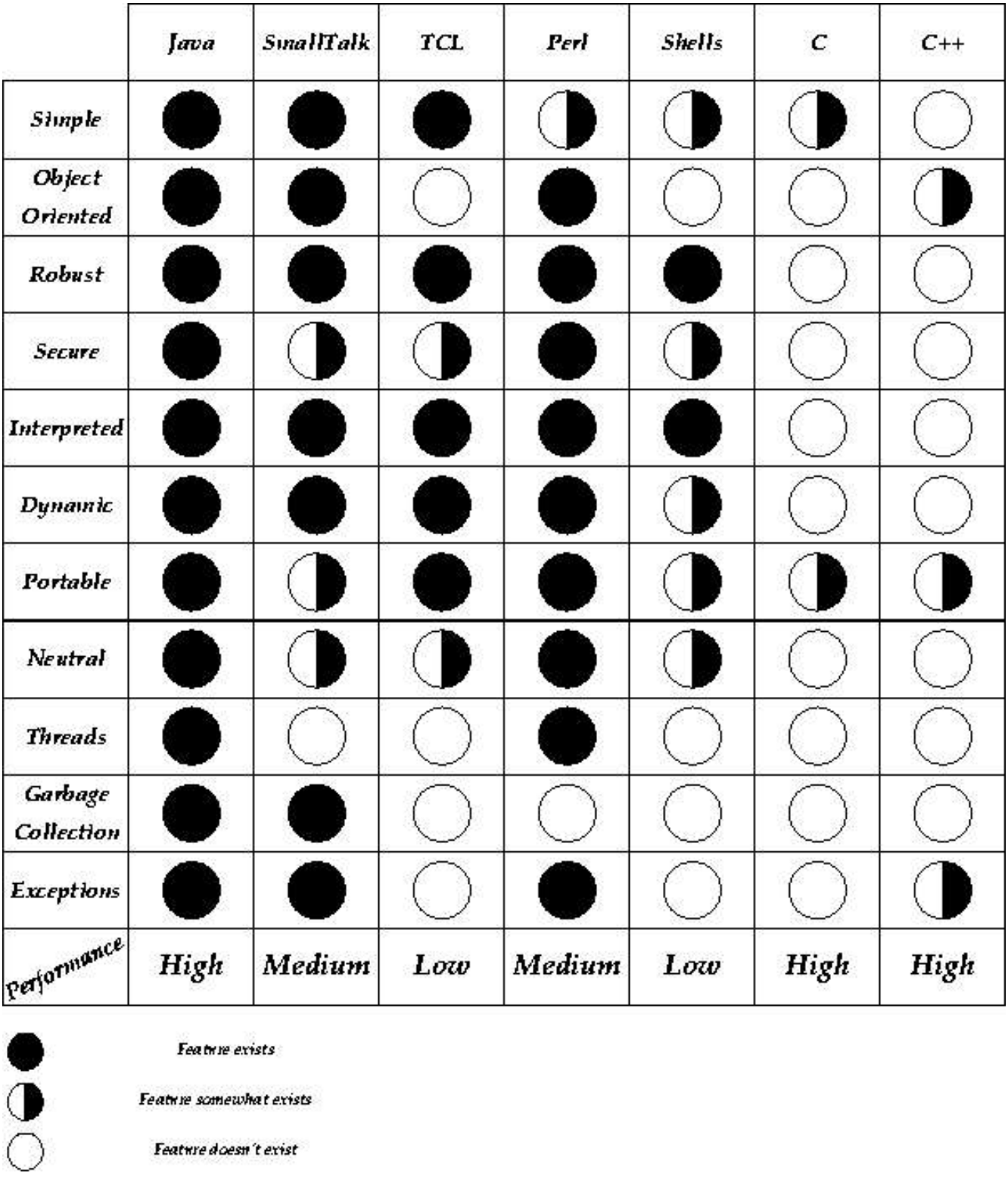

Figure 1: Comparison from the Java Whitepaper
Gosling and McGilton (1997)

3 Java for Business Applications

From the outset, Java may not seem like a good choice for business applications. "The vast majority of high-profile attempts to use Java to create major desktop applications have failed," [14] Two examples of this are Corel's rewrite of WordPerfect (et al.) that was abandoned, and Netscape's attempt to rewrite large parts of Navigator in Java. Netscape discovered that Java was too slow to accomplish the job. And even Sun Microsystem's own Star Office was written in C/C++. However, these are very large desktop applications, and this does not necessarily preclude Java from being used to create lighter business applications.

Business applications often need to be able to create reports of different and sometimes elaborate formats. They need to be able to represent data in a human understandable way, as well as need "precise ways of describing and storing decimal numbers, and character data." [15]

Java's portability is a great boon to building business applications that need to be distributed among many platforms. Implementability ("the efficiency with which a translator can be written" [16] ie. Java Virtual Machine) is taken care of by Sun Microsystems which provides a virtual machine for a variety of platforms, and by other contributors who may also provide their own virtual machines. This is especially nice in developing a front-end or access interface to a database. Each instance of the program will behave consitently both to the database and to the user.

Since Java can operate securely across a network, it is ideal for connecting to remote databases. Java's automatic memory management is also ideal for creating an application that has to allocate and deallocate large amounts of memory. It allows the programmer to concentrate on other aspects of the program. And since all objects in Java are allocated onto the heap, it can make efficient usage of the systems memory to load and display the database information. The extensive compile-time checking will help prevent errors and ensure proper operation of the interface.

The performance of Java for database access will rely heavily on how elaborate the displays it generates must be. Diplaying user interfaces is particularly time consuming in Java. Since in this case Java will only be used to access information, and not necessarily perform many calculations on it, it should perform very well.

4 Java for Scientific Applications

Scientific applications typically need simple data structures and arrays, and many floating point operations. This would be typical of a program that performs matrix manipulation. When performing matrix manipulation you want both fast floating point calculations, and to be able to load as much data into memory at once as possible since I/O operations are costly.

While Java has high reliability in general ("the assurance that a program will not behave in unexpected or disastrous ways during execution" [17]), many scientific applications are simple from a programming aspect. Matrix manipulation should require minimal user interaction, and should not inherently encounter many reliability problems.

"Java's paradigm of bytecode interpretation is widely seen to provide portability only at the expense of performance degradation relative to native code." [18] From a 1999 study at the school of Informatics

[14] Garfinkel (2001)
[15] Sebesta (2003)
[16] Louden (1993)
[17] Louden (1993)
[18] Wikipedia (2005)

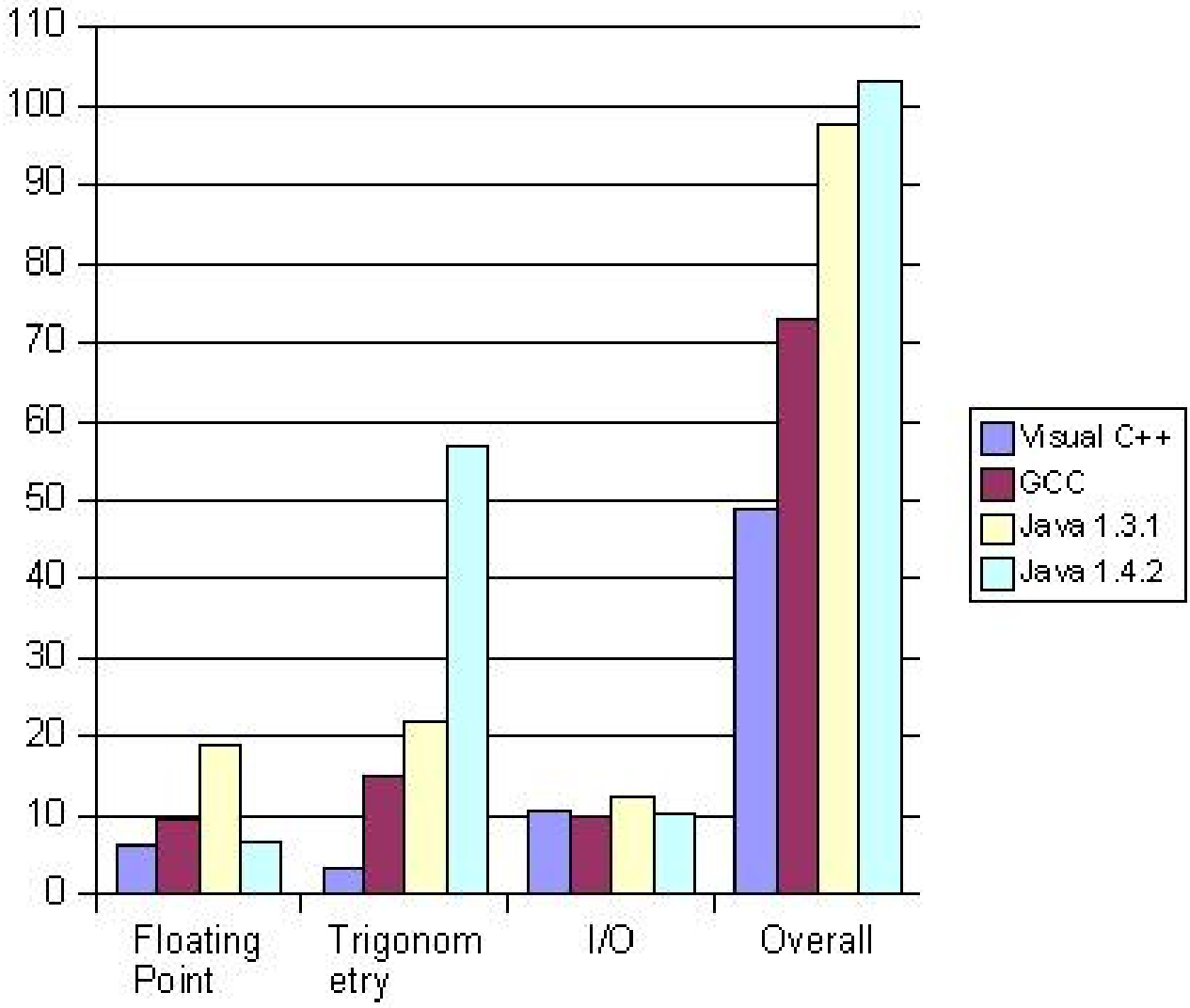

Figure 2: Benchmark Comparison
Cowell-Shah (2004)

at the University of Karlsruhe (Germany), programs written by graduate students show that C or C++ "could complete the given task between one and five minutes" while Java "required between two and 30 minutes." [19] This is most likely due to the fact that Java's bytecode is interpreted by the virtual machine which requires one more step: loading the virtual machine into memory and compiling it on-the-fly. Since most scientific applications are targetted toward a single platform, the portability added by the virtual machine is often not a consideration.

However, Java may fare a bit better in the specific area of matrix manipulation, such as matrix multiplication or Gaussian elimination, or in any cases where we can reduce trigonometric computations. "If we exclude the trigonometry component, Java performed virtually identically to Visual C++, the fastest of Microsoft's languages." [20]

Another concern with Java is memory. The Java Virtual Machine "is significantly larger than com-

[19] Garfinkel (2001)
[20] Cowell-Shah (2004)

parable runtime environments when considering resident set size (memory dedicated to this specific program). It has been seen to grow to as much as 900M." [21]

Memory and performance concerns are particularly important to matrix manipulation. The fact that Java creates arrays using objects means that doing anything with arrays (such as matrix manipulation) will require both more time and memory than could be done with traditional arrays.

5 Conclusion

For business applications, Java should fare well due to some key strengths. In particular Java would make a good choice for writing a front-end or access interface to a remote database.

- Java is highly portable and can be used to distribute software easily across platforms.
- Java operates securely over a network, and has built-in networking.
- Java abstracts memory management away from the programmer.

Business applications tend to be end-user applications, and the ability to maintain a common user interface is important.

For scientific applications, Java is less impressive due to weaknesses in performance and memory usage. It is a particularly poor choice for manipulation of large matrices.

- Java uses extra memory (overhead) for the objects it creates, memory management, and for the virtual machine itself.
- Java's floating point calculations are slow.
- Java doesn't allow fine-tuned memory control since it abstracts memory management from the programmer.
- Java's recompilation is slow due to extensive compile-time checking.

But the real reason scientific applications do not benefit much from Java is because they do not need the portability that is one of Java's trade-offs.

[21] Taylor (200X)

References

Cowell-Shah, C. W. (2004). Nine language performance round-up: Benchmarking math & file i/o. Available on the Internet: http://osnews.com/story.php?news_ id=5602.

Garfinkel, S. (2001). Java: Slow, ugly and irrelevant. Available on the Internet: http://dir.salon.com/tech/col/garf/2001/01/08 /bad_ java/index.html.

Gosling, J. and McGilton, H. (1997). The java language environment. Technical report, Sun Microsystems, Inc. Available on the Internet: http://java.sun.com/docs/white/langenv/.

Louden, K. C. (1993). *Programming Languages: Principles and Practices*. PWS Publishing Company, Boston, MA.

Sebesta, R. W. (2003). *Concepts of Programming Languages*. Addison Wesley, Boston, MA, sixth edition.

Taylor, J. S. (200X). The java problem. Available on the Internet: http://www.archub.org/javamemo.txt.

Wikipedia (2005). Java programming language. Available on the Internet: http://en.wikipedia.org/wiki/Java_ programming_ language.

Zawinski, J. (2000). java sucks. Available on the Internet: http://www.jwz.org /doc/java.html.