This article was published in 2000, in *Gamelan*

Java as a Scientific Programming Language (Part 1)

By Ken Ritley

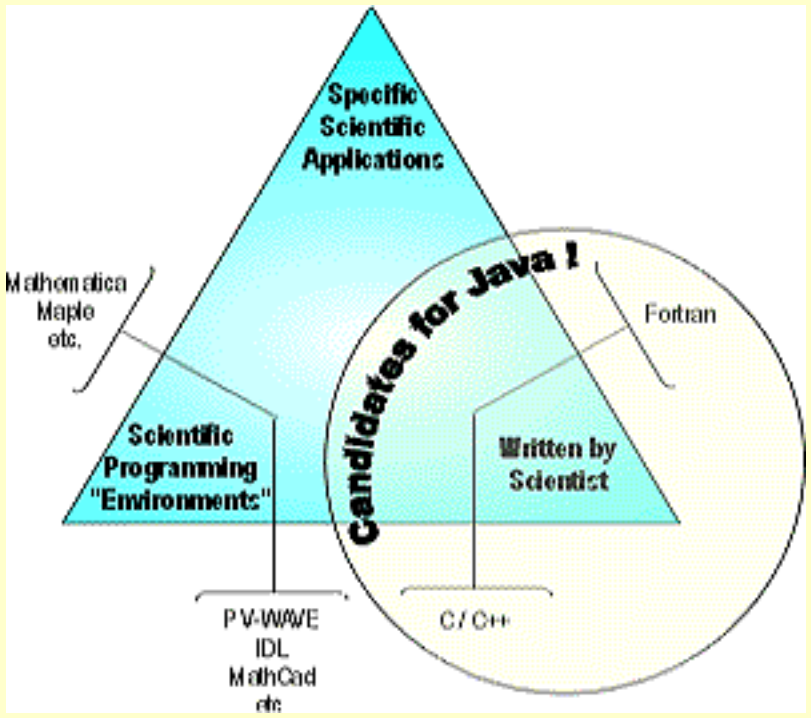
We live in a technological world, at the heart of which are scientists and engineers. They need programming tools to help make important discoveries and bring the next generation of technology to market.

In this article, we'll discuss what scientific programs are and whether Java is suited for the high-performance, numerically intensive applications which technical applications demand — in short, whether Java has a future for scientific computing. We'll also provide a list of resources for scientists new to the Java language. In Part 2, we'll examine the structure of a typical scientific program more closely, and we'll give you a short "style guide" that can help scientists write good Java programs.

The Programs Scientists Use

When the author developed scientific programs as a physics undergraduate in the early 1980s, there was only one type of scientific program: the type you wrote yourself, in Fortran, for large multi-user mainframe systems.

Types of Scientific Programs



Commercial scientific applications accomplish some specific task, such as image processing, the analysis of electronic circuits, the simulation of load-bearing structures for mechanical engineering, and so forth. These are usually complete with fancy graphics and GUIs — and cost hundreds to thousands of dollars.

Scientific programming environments, such as Mathematica, PV-Wave, MathCad, Origin and

Igor Pro, are perfect examples. Similar to the traditional programming IDEs that programmers know and love, these are what scientists and engineers use for their daily work. Some type of procedural programming language (similar to Fortran) is bundled with a user-friendly GUI and packaged with copious scientific tools for solving equations, plotting data, dealing with huge arrays of numbers, performing simulations, and so on. It's here that scientists can quickly test a new idea, develop a scientific model and see if it agrees with experimental data, or devise a new type of data analysis, for example.

Traditional programming languages such as Fortran and C still play a vital role in scientific research, partly because they are optimized for high-performance (many of the most important calculations take hours or days to run!) but mostly because it's too expensive to abandon the extant "legacy" code, literally thousands of Fortran and C programs written, debugged, tested, and re-tested over the decades. It's hard to believe, but many of the cutting-edge programs from the 1970s are still the scientific work-horses of today.

— *K. R.*

Today, the mainframe has largely been replaced by high-performance workstations and PCs, and there are roughly three categories of scientific programs: high-cost commercial scientific applications, which accomplish a specific scientific task; scientific programming environments, similar to programming IDEs but with data analysis features scientists need; and traditional programming languages, such as Fortran or C, with which scientists can write their own programs (see the sidebar "Types of Scientific Programs"). It's mainly programs in this latter category that are good candidates for Java (see the sidebar "To GUI or Not to GUI").

What Can Java Do for Scientists?

Java is the ideal language for developing business applications because it includes all the tools programmers need, from low-level tools such as hashtables and linked lists, to high-level ones for network security and CORBA. For scientific computing, the case for Java is not so clear.

Java offers both advantages and disadvantages for scientific programming. For the scientist thinking about making the move to Java, it's essential to evaluate these areas carefully. (For the non-scientific programmer, it should be instructive, as well.)

To OOP, or not to OOP

It's been argued that objected-oriented programming has been so successful because the world is made up of objects, and the purpose of programs is to describe them. It's an interesting argument, which doesn't always

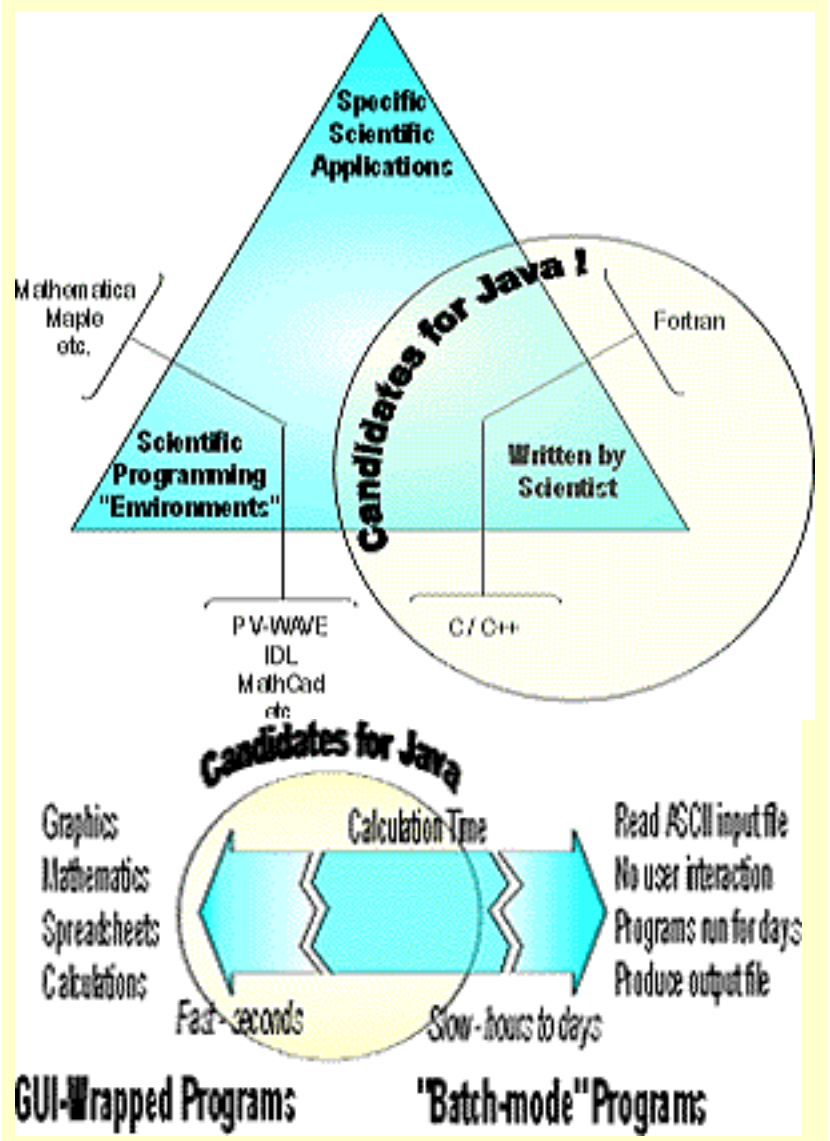
apply to scientific programming. An object-oriented program might be the natural way to deal with airplanes, for example: airplanes have *interfaces* to connect them with other aircraft (refueling airplanes), they have *properties* (number of engines), and they have *methods* (how to take off and land).

In contrast, a scientist is chiefly concerned with pure numbers, such as energy or mass or the velocity of light. The idea is to start with a small collection of numbers, to perform a sequence of distinct, functionally independent mathematical-operations (say, $E=mc^2$), and then to end up with a new set of numbers. The heart and soul of the scientific program lie in mathematical operations.

Java imposes a great deal of object-oriented infrastructure on the programmer, even when it's not needed and not natural for solving a programming problem. Fortunately, there's no need for the scientist to become an OOP specialist — there are good ways to cheat and avoid the mess! Java provides all the tools necessary for writing procedural (not object-oriented) programs. In Part 2 of this article, we'll offer some tips for creating creating global variables, managing libraries of pre-written procedures, and show how everything labeled "static" (methods and variables) can be a scientist's best friend [see Global Variables in Java with the Singleton Pattern

].

To GUI or Not to GUI



The world of scientific programming has a Twilight Zone between two extremes: On the one hand, there are programs that take days to execute; clearly, for such programs a GUI is meaningless. On the other hand, there are

programs that run quickly; clearly, for these a good GUI is essential.

The boundary between the two worlds is continually moving as the speed of computers increases. Scientific calculations for which a GUI seems unthinkable today will undoubtedly require a GUI tomorrow.

This is where Java can make an important contribution. Java has built-in cross-platform capabilities for GUIs. By implementing new calculations in Java, scientists can ensure that the cutting-edge programs of today will become the daily work-horses of tomorrow.

— *K. R.*

"Legacy" Code

It's a derogatory term. It conjures notions of dusty boxes filled with punch cards of Fortran GOTOs and Hollerith constants, and mile-high stacks of fan-fold compiler listings!

As we've said, the heart and soul of the scientific program lie in mathematical operations. A scientist may spend months or years devising and testing purely mathematical numerical methods (which scientists still call subroutines). And when it's been proven that a subroutine works, it can be published in a "library", facilitating its shared use between scientists. These libraries are fundamental to the progress of science: like libraries of books, they allow scientists to use the hard work and good ideas of generations of scientists before them. In fact, many scientific programs are nothing more than collections of legacy subroutines patched together in new ways.

Regarding legacy code and Java, the news is all bad. Java was born in Silicon Valley in 1995. It is a new programming language, and therefore there are few software libraries, at least few in comparison with the massive libraries for Fortran and C. It is not just frustrating — it can stop a scientific project dead in its tracks (see the sidebar "Scientific Resources in Java").

And the bad news gets even worse! From the scientific standpoint, Java is so different from Fortran and C, conversion of legacy code in Java by hand may be impossible. Java is quite accommodating of Fortran's array syntax and adjustable-dimension arrays, but Java doesn't support a complex number type, as will be discussed in Part 2. And although Java syntax is similar to C, the elimination of pointer arithmetic means any subroutine involving arrays and matrices — this has been called the "bread-and-butter" of scientific programming! — may not be portable to Java. To make matters worse, Java does not support unconditional branching (GOTO), a feature very heavily implemented in legacy Fortran code.

Scientific Resources in Java

Scientific Resources in Java

A large fraction of scientific programs are patchwork creations. They may tie together dozens of library subroutines, pre-written methods that have been

tested, tested, and re-tested.

So the ability to put together a scientific program depends crucially on the available libraries.

Scientific Libraries

The list of scientific libraries in Java is small but growing. Instead of trying to list them all, here is a meta-list of other lists which scientific programmers may find useful.

NETLIB: A central resource for mathematical computing
http://www.netlib.org/

Java Resources for Science and Engineering
http://www.npac.syr.edu/projects/tutorials/JavaCSE/

Colt: Open Source Libraries for High Performance Scientific and Technical Computing
http://tilde-hoschek.home.cern.ch/~hoschek/colt/index.htm

Java Numerics
http://math.nist.gov/javanumerics/

Java Classes for Operations Research
http://OpsResearch.com/OR-Objects/index.html

Java Numerical Toolkit
http://math.nist.gov/jnt/

Parallel Compiler Run-Time Consortium
http://www.npac.syr.edu/projects/pcrc/

Java Grande Forum
http://www.javagrande.org/

Titanium: A Dialect of Java for Large-Scale Scientific Computing
http://HTTP.CS.Berkeley.EDU/projects/titanium/

Java Access to Numerical Libraries
http://www.cs.utk.edu/f2j/hpjhtml/

JIGL: Java Image and Graphics Library
http://rivit.cs.byu.edu/jigl/

Image/J - Platform-independent Scientific Image Processing Package

http://rsb.info.nih.gov/ij/

Image Processing Package in Java
http://www.ctr.columbia.edu/~dzhong/JIM/JIM.html

Educational Applets

The scientific educational community has jumped on Java applets — they're a terrific way to implement "virtual" scientific demonstrations. As a consequence, the next generation of scientists may be more versed with Java than with traditional scientific langauges such as Fortran or C!

For scientific programmers, such Java applets have something to offer, too. Many of them have well-designed user-interfaces, fancy graphics and scientific plots — all of which can be "borrowed" for use in your own programs.

This is by no means a complete list, but it's definitely worth a look.

Physics Education R&D at North Carolina State University at Raleigh
http://sb-dell.physics.ncsu.edu/Java/
http://www.physics.ncsu.edu:8380/physics_ed/

Physlets at Davidson College
http://webphysics.davidson.edu/Applets/Applets.html

Physics Illuminations at the University of New Orleans
http://www.uno.edu/~rgreene/illum.html

The Optics Project at Mississippi State University
http://www.uno.edu/~rgreene/illum.html

Physics 2000 at the University of Colorado at Boulder
http://www.Colorado.EDU/physics/2000/index.pl

Project Links at Rensselaer Polytechnic Institute
http://links.math.rpi.edu/index.html

Soda Constructor
http://sodaplay.com/constructor/index.htm

— *K. R.*

For translating between other languages, translation programs (such as f2c) have often helped. These differences between Java and Fortran/C are so significant, however, a translation program may be effectively impossible to write. Interestingly enough, there are projects underway (f2j) to convert Fortran source code directly to Java bytecode, as well as to make other numerical libraries Java-accessible (NetSolve). For scientific programmers, these are vital steps in the right direction!

Memory

Like legacy code, memory and storage space is a particularly dangerous area, which requires careful deliberation before deciding to implement a scientific program in Java.

In C and Fortran 90, new memory for variables can be allocated, and memory for variables no longer in use can be deallocated. Memory allocation in Java is generally no problem; you can
`catch`
allocation errors and have your program respond appropriately if there's not enough memory when you need it.

Deallocation is trickier. Java memory deallocation is handled by the so-called *garbage collector*, a process that runs in parallel to your program that releases memory for objects which are no longer referenced [please see Understanding Automatic Garbage Collection

]. You can start a garbage collection thread, but you can't control it, and you can't easily know when the memory you've released is ready for use again.

There are tricks to beat the problem, once you're aware it exists. But you still need to be cautious — just because you think your variables are dereferenced, they may not be, creating a nasty "memory leak" that garbage collection can't fix.

Speed

Memory allocation and legacy code issues are good reasons for a scientist to think twice about using Java, but execution speed isn't one of them.

Speed is a black-and-white issue. There are applications in which speed is so important that the scientist works exclusively with optimized compilers on supercomputers — and must also spend time on program optimization, usually with the help of special "optimizers" that rewrite the code to eliminate every wasted nanosecond. These are the calculations that take days to run — and sometimes take days before confirming the "garbage in, garbage out" principle because of a typing mistake! Such programs are very bad candidates for Java.

Any program not in this category is a good candidate for Java. With earlier versions of Java, speed was a signficant issue. But as recent scientific benchmarks, including SciMark, and other benchmarks demonstrate, execution speed on a modern PC should be suitable for all non-supercomputer-based applications. And because the speed of computers seems to be increasing on a monthly basis, even without optimization, a marginal program today will execute just fine on a new computer next year!

Additional Factors to Consider

For scientists, learning Java and learning to think *scientifically* in Java may not be an easy step. Here is some additional commentary on the following issues, all of them important for scientists [Java as a Scientific Programming Language (Part 1): More Issues for Scientific Programming in Java

].

- Complex Variables
- Numerical Precision
- Hardware Independence
- IDEs and GUIs

And in Part 2 of this article, we'll offer concrete examples about how to write good scientific programs in Java.

The Future of Java for Scientific Programming

Despite some disadvantages, the future of Java as a scientific programming language looks bright. For those thinking about migrating to Java, there's some very good news on the horizon.

Scientific Libraries

The benefits of Java for scientific programming have impressed many people, and a few high-quality scientific libraries are available. Still others are under construction. (See the sidebar "Scientific Resoureces in Java".) Interestingly, using such libraries is far easier than in Fortran or C. Java provides good mechanisms for importing and managing external libraries (or packages). And there are standardized programs (javadoc) that create HTML documents which describe the library contents and their use.

Distributed Scientific Computing

The built-in threads, networking and client-server features of Java appeal to many scientists at the cutting edge of scientific computing. Such features are lacking in Fortran and C, but these are exactly the tools scientists need to write distributed applications, which subdivide a problem and harness the power of many independent computers to solve it. Known as *Grande Applications*, these are defined as applications "of large-scale nature, potentially requiring any combination of computers, networks, I/O, and memory", and the Java Grande group is a consortium of scientists and programmers working to explore how Java can best be used for such tasks.

Scientific Applets, for both Learning and Research

Applets are ultra-portable Java programs that can be sent over a network and run on a local host, usually by means of a Web browser. Scientific applets for teaching have exploded in popularity, and there are dedicated research groups that specialize in developing Java applets for scientific education. A byproduct of this effort means the next generation of scientists may be more fluent with Java than Fortran or C!

But applets aren't just for teaching purposes. They can be useful for scientific research, particularly when a large number of scientists need to share the same analysis tools. NASA, some physics researchers, and even vendors of commercial scientific software are getting into the act.

So ... if you are a scientist thinking about using the language that's revolutionized modern programming, now is the perfect time to start! And in Part 2, we'll give you some tips and tricks for helping you to write high-quality scientific programs.

About the Author

Dr. Kenneth A. Ritley is a consultant with HP Consulting in Sindelfingen, Germany. Until recently, he was a

physicist in Department Dosch at the Max-Planck-Institut für Metallforschung (Metals Research) in Stuttgart, Germany. He's made scientific computing contributions in the wide-ranging fields of astronomy, antimatter, high-temperature superconductivity, and magnetism.