Issue 024: Java

Graham Lee

September 7th, 2020

Welcome to the twenty-fourth issue of *De Programmatica Ipsum*, closing our second year with a celebration of the 25 years of *Java*. In this edition:

- Adrian tells the story[1] of the success, the backslash, and the renaissance of Java (the programming language) and Java (the virtual machine.)
- Graham explains what the real story of Java[2] is: ubiquity, stability, and long term outlook.
- In the Library section[3], Adrian reviews what programming languages[4] authors chose for their classic books. Hint: it was not always Java.

Enjoy this issue! Please subscribe to our free newsletter[5] to stay updated about new releases, or contribute[6] if you would like to support our work.

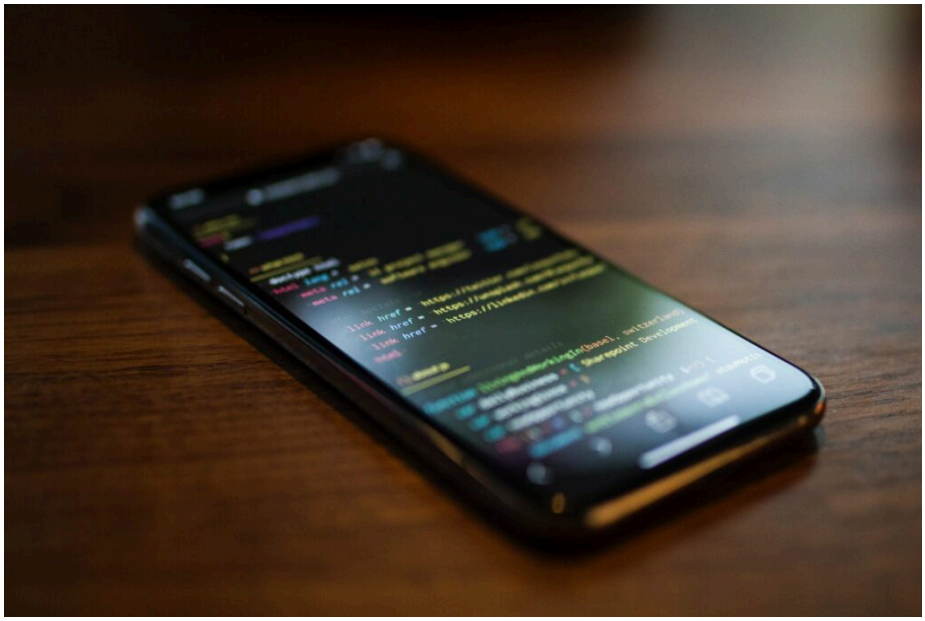
Cover photo by Michael C[7] on Unsplash[8].

[1] https://deprogrammaticaipsum.com/write-anywhere-run-once/
[2] https://deprogrammaticaipsum.com/java-the-programmer-environment-that-has-it-all/
[3] https://deprogrammaticaipsum.com/category/library/
[4] https://deprogrammaticaipsum.com/how-to-choose-a-programming-language-for-your-book/
[5] https://deprogrammaticaipsum.com/newsletter/
[6] https://deprogrammaticaipsum.com/contribute/
[7] https://unsplash.com/@michealcopley03?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[8] https://unsplash.com/s/photos/java?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Write Anywhere, Run Once

Adrian Kosmaczewski

September 7th, 2020



Back in the days when I had a day job in the .NET galaxy, I had a colleague who was a terrific C# developer... born in the island of Java. Given the looks I got from him, I think I was the first to point this irony fact to him. Or maybe I just pissed him off. In any case, here go some personal anecdotes around Java, stretching back 23 years.

First Java Applet

I have a folder in a backup disk somewhere. I find this folder every time I search for something else, which is how searching for stuff works. The files inside are dated "September 1997." They have the `*.java` and `*.class` extensions, and all together can bring a small calculator to life. I usually write a calculator as a first app[1] with the languages I learn, and this one was no exception.

Every time I used to find that folder I would launch the calculator with the `appletviewer` command. My applet, that could barely perform the four basic arithmetic operations, would invariably just run like the first day.

I compiled this applet on my PC with IBM's Applet Development Kit for Windows 3.1[2]. And 20 years later, the same binaries were running unchanged in a MacBook with Java 8. The calculator was based on the Abstract Window Toolkit[3], a pre-Swing UI framework that was simple and effective, and better yet, had a native look & feel in all platforms. The code was a mess of static methods and global variables, but it just worked™®©.

[1] https://akos.ma/blog/being-a-developer-after-40/
[2] https://www.infoworld.com/article/2077259/ibm-brings-java-to-windows-3-1.html
[3] https://en.wikipedia.org/wiki/Abstract_Window_Toolkit

But Java applets were deprecated[4] in Java 9, and completely removed from Java SE 11. And now I cannot run my calculator applet anymore, although I understand that one can still run them in Internet Explorer 11[5]. Which makes me wonder how many intranets out there might still require Internet Explorer 11 with Java applet support.

Visual J++

There was once a programming environment made by Microsoft called Visual J++[6]. It allowed one to write, debug, and run Java code on Windows. Visual J++ was the first serious, usable, complete IDE Java ever had, at least until IntelliJ IDEA[7] appeared in 2001.

Visual J++ had a faster compiler, and the generated bytecode was much faster than what the official Java SDK for Windows produced. You could also access functionality inside of packages starting with the `microsoft.*` name, but of course that kind of broke the whole point of Java, which was to make cross-platform stuff that you could write once and run anywhere[8]. So Sun got fed up and sued[9] Microsoft, who a few years later gave Sun some cash and threw Visual J++ away.

Microsoft eventually came up with C# and .NET. These were respectively still very much like Java (the language) and Java (the virtual machine.) Both C# and Java had eerily similar "Hello, World!" implementations, even though C# used PascalCase instead of camelCase. Anders Hejlsberg, the creator of C#, had a long tradition with Pascal, having created Turbo Pascal and Delphi. Later on he went to Microsoft to produce Visual J++, C#, and lately TypeScript[10]. The choice of PascalCase in C# is pervasive; not only classes start with uppercase, but also methods and whatnot. Oh, and the position of the opening curly bracket was the second major difference between them.

I actually used Visual J++ in my job once. Back in 1998, we needed to create a component to find out the width and height in pixels of the images that people were uploading to our ASP website running on IIS and Windows NT 4.0. See, our website allowed users to post messages, and those messages could have up to four pictures attached to them, so we wanted to know many things. First, are they actually uploading images, or are those zip archives, or Word files loaded with viruses? And then, what are the proportions of the image, so that the (pre-CSS) layout of our HTML looks good?

We needed to read two specific small integers (width and height) out of every image uploaded to the site. We did not find any COM component in the market to help us, so in August 1998 I tackled what was, until then, the most complex challenge I had ever had in programming. Design and code a component which, once fed a stream of bytes, would tell us whether the file was a GIF or a JPEG and, if that was the case, the screen size of the image contained therein.

For GIF files this is trivial, as the information is always in the same location in the file; for JPEG, it was a bit more complicated. So I asked for help in comp.graphics.algorithms[11] (this was 11 years before Stack Overflow, after all) and with some trial and error, found out how

[4] https://www.oracle.com/technetwork/java/javase/migratingfromapplets-2872444.pdf
[5] https://www.oracle.com/technetwork/java/javase/javaclientroadmapupdatev2020may-6548840.pdf
[6] https://en.wikipedia.org/wiki/Visual_J%2B%2B
[7] https://www.jetbrains.com/idea/
[8] https://en.wikipedia.org/wiki/Write_once%2C_run_anywhere
[9] http://techlawjournal.com/courts/sunwvmsft/Default.htm
[10] https://www.typescriptlang.org/
[11] https://groups.google.com/forum/#!topic/comp.graphics.algorithms/ZqcJb7AOgCo

to traverse a JPEG file to find those two integers. Lo and behold, I found out I had a copy of Visual J++ pre-installed in my work computer, so I used it. We later rewrote that in Visual Basic, I guess because we did not quite figure out how to make a COM component out of that Visual J++ project. Maybe it was not possible, I do not remember why.

After the release of .NET, Microsoft introduced a successor of Visual J++ (they never get tired huh) called J#[12], which compiled Java code into .NET (CLR) bytecode, but then again, it was discontinued a few years later.

Backlash

Fifteen years ago, Bruce Tate published "Beyond Java[13]." This title was the reaction against what seemed as a chokehold of Java and the JVM on the server side of the software development industry. Proposing alternatives such as Python, Smalltalk or Ruby, it remains as the book that spoke the zeitgeist of its era: the mid-2000's were not a good time for Java.

The company I worked for back in 2006 was busy rewriting their flagship app (written in C++) into Java. The project had been going on since 2005, and when I left in 2007, they still had not produced a single working screen to show to their customers or upper management. To consider that this failure was due to the choice of the programming language is not fair, given that such an enterprise is, in general, a strategic mistake[14]; actually, there were many other problems in that organization. But as a younger developer I misplaced my anger into the language, and thus push it aside, vowing never to use it if I could.

I suppose another factor that played against Java in those times was the rise of the Agile Manifesto[15]. Used to very fast write-debug-release cycles with VBScript and ASP in 1997, having to restart Tomcat after every modification of a JSP page made the whole Java developer experience an abysmally horrid one.

I found a large echo in the industry for my distaste of Java. Just to illustrate the backlash against Java, even a popular website such as TheServerSide.com[16] decided to spin TheServerSide.net[17] up, mostly (I guess) as a way to prevent the clash of matter and antimatter and the outburst of vitriol in the comments section of every article.

More or less at the same time, many Java developers started to use anything but Java (the language) in their Java (the virtual machine) applications: there came Clojure[18] for functional programming purists, Scala[19] for scalable backend services, JRuby[20], JavaScript[21] or Jython[22] for scripting, Groovy[23] for build systems, and more[24]: Smalltalk[25], BASIC[26], Cold-

[12] https://en.wikipedia.org/wiki/J_Sharp
[13] https://www.oreilly.com/library/view/beyond-java/0596100949/
[14] https://www.joelonsoftware.com/2000/04/06/things-you-should-never-do-part-i/
[15] http://agilemanifesto.org/
[16] http://www.theserverside.com/
[17] http://www.theserverside.net/
[18] https://clojure.org/
[19] http://www.scala-lang.org/
[20] http://jruby.org/
[21] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino
[22] https://www.jython.org/
[23] http://groovy-lang.org/
[24] https://en.wikipedia.org/wiki/List_of_JVM_languages
[25] http://www.redline.st/
[26] https://github.com/teverett/jvmBASIC

Fusion[27], Pascal[28], Ballerina[29]... You can even run PHP inside the JVM if you really want, using a plugin called Quercus[30].

Then Android[31] came up, and as a way to bring all those J2ME[32] developers into the party, they chose Java (the programming language.) Clever enough, the original team decided not to use Java (the virtual machine), but to rewrite it, to make those mobile apps faster and snappier in smaller and cheaper devices. By that time Sun was swallowed[33] by Oracle, and Oracle's lawyers had enough spare cash to sue Google[34] about this. Last time I checked they were still yelling at each other.

Looking back in time, the biggest problem of developing early apps for Android was not Java per se; in the opinion of this author, it was Eclipse[35]. Thankfully Google decided to ask somebody else[36] for help and thus Android Studio[37] was born in 2014. Three years later Kotlin came along[38], and these days nobody notices that Java (the virtual machine) is somehow at the heart of those 2.5 billion smartphones out there. What mobile developers did notice was that, thanks to the decline in quality of Apple's Xcode, writing an Android application in 2018 had become a much more enjoyable experience[39] than writing an iOS one. And Kotlin[40], a lightweight and modern language for Java (the virtual machine) coming from the same folks that made Android Studio, is not a small part of that success.

The fact that Java (the virtual machine, in its various forms) ended up being more popular than Java (the language) most probably make Steve Yegge[41], Joel Spolsky[42], and Paul Graham[43] nod in approval (or dismay.)

JavaOps

In spite of all the backlash, Java seems to be back in popularity these days. It's like a small renaissance of the language, somehow, happening right now in front of our eyes.

Java SE 14 was released[44] in March, bringing many features expected by developers.

Thanks to Quarkus[45] (not to mix with Quercus, mentioned previously) developers can deploy lightweight Java services on a Kubernetes clusters, with exactly the "write-debug-release" developer experience I expected back in 2006.

[27] https://www.adobe.com/products/coldfusion-family.html
[28] https://www.elementscompiler.com/elements/oxygene/default.aspx
[29] https://ballerina.io/learn/by-example/
[30] http://caucho.com/
[31] https://www.android.com/
[32] https://en.wikipedia.org/wiki/Java_Platform%2C_Micro_Edition
[33] https://en.wikipedia.org/wiki/Sun_acquisition_by_Oracle
[34] https://en.wikipedia.org/wiki/Google_v._Oracle_America
[35] https://twitter.com/sburlot/status/5199586622
[36] https://www.jetbrains.com/
[37] https://developer.android.com/studio
[38] https://techcrunch.com/2017/05/17/google-makes-kotlin-a-first-class-language-for-writing-android-apps/
[39] https://akos.ma/books/Android_for_iOS_Devs/Android_for_iOS_Devs_Java_Ed_2018.html
[40] https://kotlinlang.org/
[41] http://steve-yegge.blogspot.ch/2006/03/execution-in-kingdom-of-nouns.html
[42] https://www.joelonsoftware.com/2005/12/29/the-perils-of-javaschools-2/
[43] http://www.paulgraham.com/hundred.html
[44] https://www.oracle.com/corporate/pressrelease/oracle-announces-java14-031720.html
[45] https://quarkus.io/

Conferences like QCon[46], Devoxx[47], and GOTO[48] all feature Java-related tracks, not to mention Oracle's Code One[49], JAX[50], or jconf.dev[51], entirely dedicated to the Java ecosystem.

Kevin Henney just compiled "97 Things Every Java Programmer Should Know[52]" in a new book. Joshua Bloch recently published the 3rd edition of his classic[53], and there are many, many more books[54] published every year about the language and its uses.

For those who miss J# there is IKVM[55], an implementation of Java (the virtual machine) that runs in the .NET and Mono implementations of the CLR. Or if you prefer, JLang[56] extends the Polyglot compiler to translate Java into LLVM IR[57].

For what it is worth, Java remains an uncannily popular language[58], holding the second position (at the time of this writing) in all TIOBE[59], RedMonk[60], and PYPL[61] rankings.

Ironically enough, Microsoft publishes one of the most downloaded extension packs for Visual Studio Code, the Java Extension Pack[62]. Maybe it is a redemption, now that .NET has become[63] an actual cross-platform thing.

And in spite of all this buzz, the official Java website[64] remains faithful to its origins in 1995, a Geocities-era looking, corporate-born contraption.

Cover photo by Caspar Camille Rubin[65] on Unsplash[66].

---

[46] https://qconferences.com/
[47] https://www.devoxx.co.uk/
[48] https://gotopia.tech/
[49] https://www.oracle.com/code-one/
[50] https://jaxlondon.com/
[51] https://jconf.dev/
[52] https://www.oreilly.com/library/view/97-things-every/9781491952689/
[53] https://www.pearson.com/us/higher-education/program/Bloch-Effective-Java-3rd-Edition/PGM1763855.html
[54] https://medium.com/javarevisited/10-books-java-developers-should-read-in-2020-e6222f25cc72
[55] https://www.ikvm.net/
[56] https://polyglot-compiler.github.io/JLang/
[57] https://www.llvm.org/
[58] https://inside.java/
[59] https://www.tiobe.com/tiobe-index/
[60] https://redmonk.com/sogrady/2020/02/28/language-rankings-1-20/
[61] http://pypl.github.io/PYPL.html
[62] https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack
[63] https://dotnet.microsoft.com/
[64] https://www.java.com/en/
[65] https://unsplash.com/@casparrubin?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[66] https://unsplash.com/s/photos/java-app?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Java: The Programmer Environment That Has It All

Graham Lee

September 7th, 2020

Let me start with an admission: it took me *weeks* to work out what to talk about for the Java issue of *De Programmatica Ipsum*. There is just so much to it.

I recently took a twelve-year old Swing app[1] and—with no code changes and minimal project changes—compiled it on the latest JDK. It now runs on the latest Java Runtime Environment, and every JRE back to 2014's version 8 (which is still supported for Oracle customers until December 2030, by the way). The previous version: no code changes, just built on an earlier JDK, worked back to at least 2004's J2SE 5.0 release. So I could talk about the phenomenal compatibility and future-proofing of Java programming.

However, that Swing UI does not necessarily look the best, so I thought maybe I could talk about the embarrasment of riches available when it comes to designing user interfaces for Java software. The original, classic Abstract Window Toolkit (AWT) is still there, and still not deprecated, so I *could* make the UI out of native platform widgets. Then my code would be compatible back to the first-ever release of Java, back in 1995. Or I could use something more modern: JavaFX[2] and Apache Pivot[3] are the choices for working with marked-up UI files like Microsoft's XAML, Apple's Interface Builder, or Vue.js. A more Java-ish way to build UIs is Eclipse's SWT[4] with the JFace[5] helpers.

If I am modernising, why would I limit myself to Java? The JVM ecosystem directly supports

[1] https://github.com/iamleeg/grotag
[2] https://openjfx.io
[3] http://pivot.apache.org
[4] https://www.eclipse.org/swt/
[5] https://wiki.eclipse.org/JFace

interesting languages like Clojure[6], Scala[7] and Kotlin[8]. And then there are JVM-native ports of Python[9], Ruby[10], Smalltalk[11], and others. That is before we even start on the possibilities when you switch to a modern runtime environment like GraalVM[12].

I could carry on. This has not been merely a fraction of the Java-based technology available, it has been a fraction of the Java-based technology *I have used* in the last couple of decades. We did not talk about build systems, about server technologies, about education environments, about developer environments, about mobile, about smart cards...

Because that is the *real* story of Java. Once the ownership questions had been sorted out by Apple's Mac Runtime for Java[13] dying out, by *Sun versus Microsoft* killing the (deliberately incompatible) Microsoft JRE, by Sun's open-sourcing their own implementation to create the OpenJDK[14], we were left with the simple observation: the Java Runtime Environment is everywhere computers are (give or take), and the Java Developer Kit lets you do whatever your developers are interested in.

And all of that in a hassle-free, continuous way. No "ground-up rewrite" to support some new language the senior dev on your team insists is the next big thing: JVM classes all interoperate so you let them try their thing on the edges of your existing, well-maintained project. Some people see that as tedium: they observe code that is steeped in the 1990s design patterns phraseology, like AbstractSingletonProxyFactoryBean[15], and see this as a sign of a staid technology, beloved of managers but not deserving serious technical interest.

Meanwhile, others see signs of *success*. That code (actually from 2006) is still there because, well, it is *still there*, because its authors did not get distracted by the shiny shiny and get bogged down in another rewrite, because as manufacturers and markets came and went, the Java runtime was still there, and the bytecode could still execute.

Some of the people who do not use Java have good reasons. Many of the "write once, run anywhere" people are getting the same value out of Google's Chromium runtime environment. Others have hitched their carts to a particular vendor's horse and get more support (from the vendor and the associated community) by using that vendor's tools. But for many people, the Java Runtime Environment represents the most widely-adopted computer ever designed, and the wealth of technologies available in the ecosystem represents a broad guarantee that when they reach for "the best tool for the job", it is available for their platform.

Photo by Anni Denkova[16] on Unsplash[17]

---

[6] https://clojure.org
[7] https://scala-lang.org
[8] https://kotlinlang.org
[9] https://www.jython.org
[10] https://www.jruby.org
[11] http://www.redline.st
[12] https://www.graalvm.org
[13] https://www.infoworld.com/article/2076580/apple-releases-macos-runtime-for-java--mrj--version-2-0.html
[14] https://openjdk.java.net
[15] https://docs.spring.io/spring-framework/docs/2.5.x/api/org/springframework/aop/framework/AbstractSingletonProxyFactoryBean.html
[16] https://unsplash.com/@annidenkova?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[17] https://unsplash.com/s/photos/java?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

How To Choose A Programming Language For Your Book

Adrian Kosmaczewski

September 7th, 2020

If you wanted to write a book about any subject related to computers, but not specifically about a particular programming language, which language would you choose? For example: if you wanted to teach programming concepts (algorithms, patterns) to an absolute beginner, which language would convey your thoughts better? Say, if you had to explain algorithms that could be implemented in any Turing-complete language, which one would you pick, and why?

This problem is not new. As this edition of the Library section[1] coincides with the 25th anniversary of Java, I decided to look back in time to see which languages were chosen in famous (and not so famous) computer books. Of course, we are leaving aside all books called "The X Programming Language," since the answer for the question is, invariable, the X programming language, whatever X might be.

Early Examples

Let us begin with "The Art of Computer Programming[2]" (1968), a book about which Graham wrote an article[3] in a previous edition. To illustrate the various themes of the book, Donald Knuth invented his own programming language. Actually not one but two; MIX[4], used in the first editions, and a RISC version called MMIX[5] for the latest ones. Designing

[1] https://deprogrammaticaipsum.com/category/library/
[2] https://www-cs-faculty.stanford.edu/~knuth/taocp.html
[3] https://deprogrammaticaipsum.com/the-art-of-the-art-of-computer-programming/
[4] https://en.wikipedia.org/wiki/MIX
[5] https://en.wikipedia.org/wiki/MMIX

a programming language for your own book is akin to J. R. R. Tolkien creating his own languages[6] to tell a story; it is certainly fair to say that Knuth's work can be easily compared to Tolkien's, if not in popularity, definitely in magnitude and reputation.

Somewhat similar, Niklaus Wirth used his own Pascal language to illustrate the classic "Algorithms + Data Structures = Programs[7]" (1976); the book describes a Tiny Pascal compiler, which is said to have inspired Anders Hejlsberg to create Turbo Pascal[8]. We can add Brad Cox to the same club, whose "Object Oriented Programming: An Evolutionary Approach"[9] (1986) title, also the subject of a popular article[10] by Graham, uses Objective-C as a tool to explain OO concepts which were quite novel at the time.

Later authors have understandably preferred to use existing languages; for example "The Elements of Programming Style[11]" (1974) by Brian W. Kernighan and P. J. Plauger. In this case the authors chose Fortran and PL/I:

> ...since these languages are widely used and are sufficiently similar that a reading knowledge of one means that the other can also be read well enough.

The Reign of C/C++

The rise of Object Oriented Programming (OOP) had it hallmark moment in the 1980s and 1990s. And at the beginning of this craze, the most widely used language to explain it was C++.

I will start with the book that taught me OOP: my beloved copy of "Object-Oriented Analysis and Design with Applications (2nd edition)[12]" (1994) by Grady Booch, which represented also my first exposure to C++ code examples. Grady Booch is one of the "three amigos" who came up with UML[13] and in 2002 IBM bought[14] their company. Interestingly, the 3rd edition[15] of this opus (2007) has examples in C# and Java.

OOP begat the design patterns movement, incarnated by two major books: "Design Patterns for Object-Oriented Software Development[16]" (1994) by Wolfgang Pree, a title that got sadly, unjustly, and immediately shadowed by the success of the eponymous "Design Patterns: Elements of Reusable Object-Oriented Software[17]" (1994) by the "Gang of Four": Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Both books, released one year before the smashing release of Java, show examples in C++.

C++, largely thanks to the legendary speed attributed to its name, found a niche in numerical and financial processing back in the 1990s, starting a slow industrial demise of Fortran. In this sense it is worthwhile to mention "Numerical Recipes: The Art of Scientific Computing

[6] https://en.wikipedia.org/wiki/Languages_constructed_by_J._R._R._Tolkien
[7] https://en.wikipedia.org/wiki/Algorithms_%2B_Data_Structures_%3D_Programs
[8] https://en.wikipedia.org/wiki/Turbo_Pascal
[9] https://www.amazon.com/Object-Oriented-Programming-Evolutionary-Brad-Cox/dp/0201548348
[10] https://deprogrammaticaipsum.com/brad-cox/
[11] https://en.wikipedia.org/wiki/The_Elements_of_Programming_Style
[12] https://www.pearson.com/us/higher-education/product/Booch-Object-Oriented-Analysis-and-Design-with-Applications-2nd-Edition/9780805353402.html
[13] https://en.wikipedia.org/wiki/Unified_Modeling_Language
[14] https://www.eweek.com/pc-hardware/ibm-acquires-rational
[15] https://www.pearson.com/store/p/object-oriented-analysis-and-design-with-applications/P100001406134/9780201895513
[16] https://www.amazon.com/Design-Patterns-Object-Oriented-Software-Development/dp/0201422948
[17] https://en.wikipedia.org/wiki/Design_Patterns

(3rd edition)[18]" (2007) whose first edition (published in 1992) featured two distinct versions: one in C and another in Fortran. As for financial modeling, we can mention "Modeling Derivatives in C++[19]" (2004) by Justin London, and the excellent "Financial Instrument Pricing Using C++[20]" (2004) by Daniel J. Duffy, recently updated to C++11[21] in a second edition (2018).

Regardless of when they are published, some low-level topics are better explained using C. Examples are the both extraordinary and fundamental "Linux Programming Interface[22]" (2011) by Michael Kerrisk, and "Beej's Guide to Network Programming: Using Internet Sockets[23]" (2019) by Brian "Beej Jorgensen" Hall.

Java Takes Over

But in 1995, just as the World Wide Web was making headlines, Sun released Java and became, in just a few years, one of the most popular programming languages ever created. Book authors took notice, and promptly started using it in their work. Since Java was being actively marketed as a "better" or "simplified" C++, it was quickly considered that it would make programming books immediately more approachable. The rest is history.

Martin Fowler's "Refactoring: Improving the Design of Existing Code[24]" (2000) and "Program Development in Java: Abstraction, Specification, and Object-Oriented Design[25]" (2000) by John Guttag and Barbara Liskov (Graham wrote about her[26] and this book in this magazine) are two major examples of early books centered around Java.

Kathy Sierra's Head First series in O'Reilly (read the article dedicated to her[27] in this magazine) relied heavily in Java for two major titles: "Head First Design Patterns[28]" (2004) and "Head First Object-Oriented Analysis and Design[29]" (2006).

In the area of university textbooks, "Computer Networking: A Top-Down Approach Featuring the Internet[30]" (2004) by James F. Kurose and Keith W. Ross features a "top down approach" starting at the OSI Application layer, with many code examples in Java. Similarly, "Schaum's Outline of Principles of Computer Science[31]" (2008) by Paul Tymann and Carl Reynolds followed the trend, firmly set in the early 2000s, of university Computer Science programs based in Java.

Grady Booch was not the only one of the "three amigos" to use Java in their books: "The Unified Modeling Language User Guide (2nd Edition)[32]" (2005) by Grady Booch, James Rumbaugh and Ivar Jacobson features examples in JavaBeans (and COM[33]), while "Aspect

[18] http://www.numerical.recipes/
[19] https://www.amazon.com/Modeling-Derivatives-C-Justin-London/dp/0471654647
[20] https://www.wiley.com/en-gb/Financial+Instrument+Pricing+Using+C%2B%2B-p-9780470855096
[21] https://www.wiley.com/en-us/Financial+Instrument+Pricing+Using+C%2B%2B%2C+2nd+Edition-p-9781119170488
[22] https://en.wikipedia.org/wiki/The_Linux_Programming_Interface
[23] https://beej.us/guide/bgnet/
[24] https://refactoring.com/
[25] https://www.amazon.com/Program-Development-Java-Specification-Object-Oriented/dp/0201657686
[26] https://deprogrammaticaipsum.com/barbara-liskov/
[27] https://deprogrammaticaipsum.com/kathy-sierra/
[28] https://www.amazon.com/Head-First-Design-Patterns-Brain-Friendly/dp/0596007124
[29] https://www.amazon.com/Head-First-Object-Oriented-Analysis-Design/dp/0596008678
[30] http://gaia.cs.umass.edu/kurose_ross/
[31] https://www.amazon.com/Schaums-Outline-Principles-Computer-Science/dp/0071460519
[32] https://www.amazon.com/Unified-Modeling-Language-User-Guide/dp/0321267974
[33] https://en.wikipedia.org/wiki/Component_Object_Model

Oriented Software Development with Use Cases[34]" (2004) by Ivar Jacobson and Pan-Wei Ng uses AspectJ[35] (an extension to the Java language.)

Beyond Java

By 2005 everybody was raving about Agile and Open Source and Web 2.0[36]. In the middle of that frenzy, a young man called David Heinemeier Hansson presented a new web framework called Ruby on Rails[37]. The recording[38] of a demo explaining how to create a blog engine in 15 minutes, was then uploaded to a new platform called YouTube you might have heard about since. Besides proving that Mac OS X was a suitable programming platform, or that TextMate[39] was a very nice editor, this video galvanized a whole generation to the notion that there were simpler languages than Java. Those thoughts were condensed in Bruce Tate's "Beyond Java[40]" book. These were the times where "think different[41]" meant something.

The reaction was immediate, and programming books started exploring other languages. The essential "Python Programming: An Introduction to Computer Science[42]" (2004) by John M. Zelle (recently updated to its third edition) uses Python. "Programming Collective Intelligence[43]" (2007) by Toby Segaran used Python as well. "RESTful Web Services[44]" (2007) by by Leonard Richardson and Sam Ruby uses... Ruby, as the authors say:

> We chose Ruby because it's concise and easy to read, even for programmers who don't know the language. (And because it's nice and confusing in conjunction with Sam's last name.)

This trend continues to today. Two interesting examples are "The Imposter's Handbook[45]" (2016) by Rob Conery, which uses JavaScript, while "Domain Modeling Made Functional[46]" (2018) by Scott Wlaschin is based in F#.

Outsiders

Of course not everybody falls in the buckets defined above. Some absolute classics have taken orthogonal routes, in some cases with great success.

We must mention them: "Structure and Interpretation of Computer Programs[47]" (1984), also called the "Wizard Book", by Hal Abelson, Jerry Sussman and Julie Sussman, which uses Lisp. "Specifying Systems[48]" (2003) by Leslie Lamport uses the TLA+[49] formal specification language. "Object-Oriented Software Construction[50]" (1988) by Bertrand Meyer

[34] https://www.amazon.com/Aspect-Oriented-Software-Development-Use-Cases/dp/0321268881
[35] https://en.wikipedia.org/wiki/AspectJ
[36] https://en.wikipedia.org/wiki/Web_2.0
[37] https://rubyonrails.org/
[38] https://www.youtube.com/watch?v=Gzj723LkRJY
[39] https://macromates.com/
[40] https://www.oreilly.com/library/view/beyond-java/0596100949/
[41] https://en.wikipedia.org/wiki/Think_different
[42] https://fbeedle.com/our-books/23-python-programming-an-introduction-to-computer-science-3rd-ed-9781590282755.html
[43] https://www.oreilly.com/library/view/programming-collective-intelligence/9780596529321/
[44] https://www.oreilly.com/library/view/restful-web-services/9780596529260/
[45] https://bigmachine.io/products/the-imposters-handbook/
[46] https://pragprog.com/titles/swdddf/domain-modeling-made-functional/
[47] https://mitpress.mit.edu/9780262510875/structure-and-interpretation-of-computer-programs/
[48] http://lamport.azurewebsites.net/tla/book.html
[49] https://en.wikipedia.org/wiki/TLA%2B
[50] https://en.wikipedia.org/wiki/Object-Oriented_Software_Construction

(which we have covered[51] previously in this section) uses Eiffel. "Introduction to Functional Programming[52]" (1988) by Richard Bird & Philip Wadler uses Miranda. And more recently, "AP® Computer Science Principles with Swift[53]" (2019) by Apple Education showcases its newest programming language, Swift.

MIT's "Introduction to Algorithms, Third Edition[54]" (2009) by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein uses a special pseudocode that, for the third edition, took a shape closer to Java, C, C++ and Python.

Finally, in a category of its own, "An Introduction to Functional Programming Through Lambda Calculus[55]" (2011) by Greg Michaelson deserves a special mention; the book illustrates all concepts through lambda calculus[56]. But we know since Alonzo Church that lambda calculus is akin to a model of computation and can simulate a Turing machine... well, it is probably a programming language in its own right.

Criteria

So, how does an author choose a programming language? There are a few factors that come into play. First, there is definitely hype[57] and vendor lobbying; or, as others would call it, "market demand" which would explain choices such as Java, Swift, and various .NET languages to appear in teaching curricula.

Let us consider the target audience; are we talking to math majors, computer science PhDs, or practical programmers? Not to mention the themes of interest of such audiences; which explains why most Artificial Intelligence Machine Learning[58] titles overwhelmingly feature examples in Python. Similarly, mathematically-oriented books will lately prefer Python today to Fortran; in both cases, the rise in popularity of NumPy[59] was enough to drive the choice of the programming language.

Teachers prefer some programming paradigms[60] to others. For books explaining concepts around Object Oriented Programming, C++ and Java are overwhelminly used. For functional programming, though, the choice is a bit larger.

What about cross-platform[61] issues? Should all students use the same operating system, or could they learn on whichever one they prefer? In a related fashion, is open source[62] a criteria to consider for the choice of a learning language?

Finally, there is the approachability of the language; how easy is it to read or write it? Is there a Read-eval-print loop (REPL)[63] available for the language? Experience shows that being able to write lines of Ruby, JavaScript, or Python code in a REPL accelerates the adoption of a language, instead of having to setup a complex toolchain to be able to compile and test a simple program.

[51] https://deprogrammaticaipsum.com/bertrand-meyer/
[52] https://www.amazon.com/Introduction-functional-programming-Prentice-international/dp/0134841891
[53] https://books.apple.com/us/book/ap-computer-science-principles-with-swift/id1456795905
[54] https://mitpress.mit.edu/books/introduction-algorithms-third-edition
[55] https://www.amazon.de/dp/0486478831/
[56] https://en.wikipedia.org/wiki/Lambda_calculus
[57] https://deprogrammaticaipsum.com/issue/issue-001-hype/
[58] https://deprogrammaticaipsum.com/issue/issue-011-artificial-intelligence/
[59] https://numpy.org/
[60] https://deprogrammaticaipsum.com/issue/issue-010-programming-paradigms/
[61] https://deprogrammaticaipsum.com/issue/issue-019-cross-platform/
[62] https://deprogrammaticaipsum.com/issue/issue-021-open-source/
[63] https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop

The Polyglot Approach

In the humble opinion of the author of these words, there is no better choice of programming language than many of them at the same time. Quite a few hallmark books have adopted this strategy, which yields rich, diverse books, using the full strength of each chosen language and, at the same time, automatically expanding the horizons of readers.

As Steve Yegge once said[64],

> But you should take anything a "Java programmer" tells you with a hefty grain of salt, because an "X programmer", for any value of X, is a weak player. You have to cross-train to be a decent athlete these days. Programmers need to be fluent in multiple languages with fundamentally different "character" before they can make truly informed design decisions.

From the 1980s, come to mind "The Peter Norton Programmer's Guide to the IBM PC[65]" (1985), whose last chapter explains Assembler, Basic, Pascal and C. Also from that era, "Compilers: Principles, Techniques, and Tools[66]" (1986) also known as the "Dragon Book," by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, whose chapters show a mix of FORTRAN, C, EQN, Modula-2, and shows a compiler for a subset of Pascal as a project at the end of the book.

More recently, "Writing Secure Code, Second Edition[67]" (2002) by Michael Howard and David LeBlanc, illustrates common security vulnerabilities in code with examples written in C, C++, Perl, VBScript, JScript, SQL, Visual Basic.NET, and C#. The "Dinosaur Book" "Operating System Concepts, Seventh Edition[68]" (2005) by Abraham Silberschatz, Peter B. Galvin and Greg Gagne, uses C code snippets illustrating the Win32 API[69], as well as Java for some higher-level concepts such as threads, file locking, and others. The widely celebrated "Working Effectively with Legacy Code"[70] (2004) by Michael Feathers uses Java, C++ and C – the latter two, as expected, to show "legacy" code. Finally, Deitel & Deitel's series "How to Program[71]" feature books in C++, Java, C#, Visual Basic, and other languages.

I would like to highlight as well, although it is not a book *per se*, the excellent Stanford Course CS107 "Programming Paradigms[72]" (2008) where professor Jerry Cain explains various subjects using C, Assembly, C++, Scheme and Python.

I will finish with a quote I could not agree more with, borrowed from "Code Complete (2nd edition)[73]" (2004) by Steve McConnell, and which summarizes my thoughts in this matter:

> The examples are in multiple languages because mastering more than one language is often a watershed in the career of a profesional programmer.

Cover photo by Iñaki del Olmo[74] on Unsplash[75].

---

[64] http://steve-yegge.blogspot.com/2007/12/codes-worst-enemy.html
[65] https://www.amazon.com/Peter-Norton-Programmers-Guide-IBM/dp/0914845462
[66] https://en.wikipedia.org/wiki/Dragon_Book_%28computer_science%29
[67] https://www.amazon.com/Writing-Secure-Second-Developer-Practices/dp/0735617228
[68] https://www.amazon.in/gp/product/0471694665/
[69] https://en.wikipedia.org/wiki/Windows_API
[70] https://www.amazon.com/Working-Effectively-Legacy-Michael-Feathers/dp/0131177052
[71] https://deitel.com/books/
[72] https://www.youtube.com/view_play_list?p=9D558D49CA734A02
[73] https://en.wikipedia.org/wiki/Code_Complete
[74] https://unsplash.com/@inakihxz?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[75] https://unsplash.com/s/photos/choice?utm_source=unsplash&utm_medium=referral&utm_content=cre

ditCopyText