

An Interview with Ed C. Epp

By Tom Sumner

Ed C. Epp is the author of *Prelude to Patterns in Computer Science Using Java*, published by Franklin, Beedle & Associates. I spoke with Ed recently about how his work developing classes for C++ led to a keen interest in Java, the expressiveness of computer languages, and the importance of patterns in learning to program.



"Java will not be around forever. There will be other languages that come along in five or ten years, so it's important that students take ideas that will last through these generations of languages."

What attracts you to Java?

When I look at Java, it attracts me from a point of view of being able to develop programs, and also it attracts me pedagogically because it is an attractive language to teach students how to program. It's attractive from both aspects. People can rightfully argue that a language like Eiffel or Smalltalk might be better, but from the point of view of where the world is, Java is in a very nice place because it captures a lot of concepts very nicely and it has some popularity. So students feel like they've learned something in school that they can use.

Why did you choose to write a book on Java? What brought you to that project?

It's one of those accidents of history. I never planned to write a book. I didn't want to write a book because it's a long, hard, tedious process. And it's not a process that necessarily gives one a large financial reward. So I didn't view it from that point of view. The thing that attracted me was that the books I had access to were not taking advantage of Java. They were approaching Java as you would Pascal or C++. My feeling is that when you take a look at a language, you know a language kind of cries out for an approach. Even though all languages are equally powerful and you can express ideas in many spoken languages, you know that some languages are more expressive for talking about certain kinds of things or approach-

ing the world in a particular kind of way.

When I talk about Java, I simply wanted to teach it differently. It got its start with the things I was doing in C + + . The first thing I thought was, well, there are some certain concepts that have to be worked with—one is objects. Objects are the heart and soul of Java. Some of the books seemed to be trying to teach Java without them. And that to me was frustrating. In addition, graphics and GUIs are such a core that you really have to incorporate them without ignoring text IO.

“A person really needs to see how to build something useful. You can’t just learn about bricks. You have to take a look at the houses around you and see how you use bricks.”

Another thing that made me want to write a book is that when I look at learning a new language there are two things that help me. One is that I need to see an example. The other is that I have to understand the syntax and semantics. A lot of books were focusing on just the syntax. And I think that stifles learning. A person really needs to see how to build something useful. You can’t just learn about bricks. You have to take a look at the houses around you and see how you use bricks and how you place windows to fashion a living space.

The third thing I pulled out was the idea of patterns. I really see that as the core of the book. Java will not be around forever. There will be other languages that come along in five or ten years, so it’s important that students take ideas that will last through these generations of languages. When you look at architecture, there are reasons that some buildings are more livable than others. There are patterns of placing windows or doors or rooms that really make a building very livable. So the idea of patterns in software is important because the way you design a program determines how easy it is to live with that design. It determines how easy it is for the software architect and the software engineer to enhance and modify the software. I was really able to leverage that at Intel. Using just a few of the simple patterns I made, I was able to create an application that was very easy for them to expand. So it just kind of solidified the idea—this book is called “Prelude to Patterns.” You start with very elementary things that are at the statement level. You put a few statements together to build different kinds of loops, different kinds of selection

statements. And by the end we introduce a few object patterns that should answer how to put objects together to create designs that are flexible, and that becomes another core idea of the book.

From each example, there are important patterns that help pull that code together. The idea of an example and a pattern is something that students can take with them for any language they learn. It makes the language transferable.

Can you give a synopsis of your history with Java?

When I started teaching at the University of Portland, the first step we made was to move from Modula 2 and Pascal to C + + . The reason was to leverage objects. One of the things I'd dreamed about was to have a library of objects that you could put together as you would Lego or Tinker Toys to build interesting things.

So the first several years I spent getting a handle on C + + and how to put things together and how to teach that to students. That involved a substantial amount of effort. When I started teaching it, I wanted to incorporate GUIs and graphics with C + + . I built a very small library, which I called Vista. What it included was some graphics instruments to draw boxes and lines and circles—something that made Windows programming a lot easier.

The problem with Windows is that in order to do anything of interest you first of all had to know a lot of C + + and write a 60-line program just to pop up a window. There are hundreds of places to make mistakes and you'd get error messages that were simply beyond a novice to handle. So what I did was to build a simple class library that would encapsulate those 60 lines of code into one method call so that a student could pop up a window with one method call and use another method call to draw a rectangle or create a button or do a callback.

Students were becoming accustomed to working in an event-based or graphic environment. So I felt strongly that you needed to start with that kind of environment with students right away.

Five years ago I picked up one of the only books I could

"Students were becoming accustomed to working in an event-based or graphic environment. So I felt strongly that you needed to start with that kind of environment with students right away."

find on Java, Hooked on Java. After spending five or ten minutes with the book I got excited. What they'd done in Java was they'd created this library of graphics and GUI riches and stuff that was easy to work with. So it took all the burden off me of having to create and support this library. I could concentrate on the applications and student instructions.

In a sense, you were deriving your own Java from C++ at the time Java was being developed by Sun?

I was developing some of the features. I wasn't supporting the cross-platform stuff, but I envisioned it running on both Windows and Macintosh. When I was building my libraries I wanted them to be applicable to both. In other places it was not like Java—it wasn't Web-based, and so it was really scaled down. Java has well over 500 classes in just the AWP set of libraries. I was working with maybe five to ten classes and would have maybe a dozen methods with each of those. The scale difference was substantial—I was working on a corner of it.

There were a lot of things that were easier to do in Java than the other language. That made it possible to teach things earlier than I ever had before. Not only could I teach graphics and GUIs, but I could teach threads. In all these instances, Java was a better example of an object-oriented language than C++ . It also made sense to teach inheritance.

The class library for Java was also designed at the time when patterns were coming up. That was another thing that made Java attractive. Not only was it possible to create patterns, but they used patterns in some of the libraries. So Java encouraged the idea of patterns, and patterns is something that I think is terribly important.

Java is famous for its Web-based applications. What are some other applications?

When I think of Java, I view it as I would C++ . It looks a lot like C++ . However, it's clean and simplified. And so from a language perspective, I view it in the same class of languages as I do C++ . For me, Java can be used for anything. When I was at Intel for the summer, I used Java for nonWeb-based

"All languages from lowly machine language all the way up to very high-level languages are equally powerful in that they can all solve the same set of problems. It's a matter of how easy it is to solve a problem."

applications. The reason we used it there was because we were interested in something that was cross-platform. That was very easy to do in Java. C++ is cross-platform, but the GUI stuff isn't. So when I think about Java, I don't think primarily about the Web, I think about a general programming language that is designed to solve any problem.

Would you call it more powerful or easier than C++?

The idea of power . . . There's a definition of power in computer science that deals with all kinds of problems that you can solve. So all languages from lowly machine language all the way up to very high-level languages are equally powerful in that they can all solve the same set of problems. It's a matter of how easy it is to solve a problem.

It's not intrinsic in the program. It's in the writer of the program.

It's in the writer of the program—the programmer. It's easier to be expressive. I find Java more expressive than C++. Not as popular as C, but more expressive than C++.

When did you get started in computing?

I developed this interest in computers in 1966, when, working with relays, I built a small machine as a high-school science-fair project. When it came time for college, though, the school I attended did not offer a course of study in computer science. At that point, I became a physics major with a side interest in computers.

Some time after I finished school, the microprocessor started to become a hot hobby item. My interest piqued, I started to play around with a home-built computer, realized the potential, and decided to make physics my hobby, and computers my main interest.

Two post-graduate degrees in computer science and a couple of teaching jobs later, I became an associate professor of computer science at the University of Portland in 1989. I recently went to work for Intel, where I work at the Intel Architectural Labs. I found my interest in object design patterns, software architecture, and software engineering a tremendous asset as I work on concept prototypes.

“When I think about Java . . . I think about a general programming language that is designed to solve any problem”