A first glance, asking whether Cobol exists might seem like a ploy, a setup for describing how the billions of lines of Cobol in the world are responsible for almost every facet of daily life—from paychecks and utility bills to the entire computer-based corporate infrastructure. Yet, although thousands of Cobol programmers are alive and well, keeping the wheels of commerce turning, there is a large contingent in the software world for whom Cobol does not exist. Academia barely teaches it, Java and C programmers ignore it, and many look with disdain both at Cobol and the lowly Cobol programmer toiling away at mundane data-processing tasks.

The irony is that more than the Java, C, or language-of-the-month programmers, Cobol programmers understand data, and in case you haven’t noticed, data is back.

Not that data ever went away. It’s still the driving force behind most of the applications we write. It’s just that the Internet has forced the data issue—how to integrate and move data across platforms, build data warehouses and repositories for data mining and e-commerce, and share data and maintain semantics. These are hard problems—ones that Cobol and Cobol programmers have been wrestling with since the 1960s. Now that we perceive data as being “hot,” Cobol is taking on a new luster and even getting a second look, especially given the notoriously poor data-handling capabilities of “cooler” languages such as C, C++, and Java.

**Ahead of Its Time**

Looking back after more than 40 years, it’s amazing to find that much of what Cobol offers is also associated with more contemporary languages and technologies. Its structural framework, centered on the well-defined environment, data, and procedure divisions, provides textual-conceptual markers for program readability and comprehension—issues we now consider essential for software maintenance and program understanding. Perhaps Cobol’s greatest contribution is its hierarchical, recursive data descriptors, which not only map naturally to hierarchical databases such as IMS but more excitingly to XML (eXtensible Markup Language)
Does Cobol exist?

Continued from page 22

ble Markup Language), seen by many as the next revolutionary step in computing. Cobol’s move-corresponding verb, which allows subtrees of hierarchical data structures to be moved from one data tree to another, provides a powerful way to rearrange data hierarchies—a problem the XML world is currently addressing with XSLT, XML’s stylesheet transform language. For Cobol, it’s “been there, done that.”

Cobol’s data focus is intimately tied to transaction processing, which, until recently, has been seen by the coding elite as a detail of antiquated mainframe technology. However, the realization that transactions are the key to controlling the distributed middle tier has vendors and programmers jumping aboard the transaction bandwagon. Battles rage between Corba’s Transaction Service, Sun’s Java Transaction Service, and Microsoft’s Transaction Server, as programmers scramble to absorb the terminology and nuances of rollback and the ACID properties of transactions (atomicity, consistency, isolation, and durability). Yet for Cobol programmers, transactions are old hat, part of the business of moving data and maintaining consistency across data stores. IBM’s CICS (Customer Information Control Systems), the granddaddy of all transaction monitors, has had close links to Cobol for over 30 years.

In terms of portability, Cobol reigns as the most portable language ever, attributable to Cobol’s machine-independent data descriptors. The ability to segregate platform-specific code in a separate environment division lets programmers write machine-independent data and procedure divisions. This stands in stark contrast to the problems C or Fortran programmers face when trying to port programs to machines with 32-, 64- or 128-bit word sizes. The dependence between data type and word size in both Fortran and C is significant and is known to force complete program rewrites despite language standards. Java’s mandate of fixed sizes for integers and floating-point numbers addresses a problem that is nonexistent for Cobol.

Interestingly, on a theoretical level, one of Cobol’s unsung contributions to programming language theory is the metalanguage notation used to specify the Cobol language itself. Although many don’t think of Cobol as having serious theoretical underpinnings, its metalanguage notation serves as an alternative to the BNF (Backus Normal Form) for defining grammars. However, as Jean Sammet (a member of the original Cobol task force) laments, “Probably the biggest mistake we made out of sheer naiveté was to call it a notation and not a metalanguage. I think had we called it a metalanguage, certain aspects of the history of computer science might be rather different!” (R.L. Wexelbat, History of Programming Languages, Academic Press, New York, 1981, p. 255).

Still Charging Ahead

Despite its stogy, wordy image, Cobol is quite contemporary, maturing in a transactional world that non-Cobol programmers are only beginning to comprehend. Many of Cobol’s underpinnings are current state-of-the-art concepts that the Cobol programmer works with and implicitly understands—self-documenting code, recursive hierarchical data structures, language-based subtree transforms, portability, and metalanguage descriptors. The ultimate irony for Cobol’s detractors is that yes, Cobol exists, and it might be the lowly Cobol programmer that leads us into the brave new data-centric computing millennium.

Frank P. Coyle is the director of the Software Engineering Program at Southern Methodist University. His research interests include programming languages and using XML to tailor instruction for distance-learning environments. He received a BS from Fordham College, an MS from Georgia Institute of Technology, and a Ph.D. from SMU. Contact him at the Computer Science Department, Southern Methodist University, Dallas, TX 75275; coyle@seas.smu.edu.

Cay responds

Frank Coyle first argues that Cobol makes it easy to access the enormous amount of available data. It does make it easy to obtain data stored in the hierarchical format for which Cobol is optimized, but there is no compelling benefit to using Cobol for data stored in a relational or object database. Coyle’s second argument is that there is a close match between Cobol data descriptors and XML. However, Java is light-years ahead when it comes to XML processing. XML parsers and data conversion toolkits exist in great profusion in Java, whereas it’s a challenge to implement an XML parser in Cobol.

Third, Coyle argues that Cobol programmers can solve difficult data-integrity and transactions problems. Cobol programmers have a strong transaction-processing background—while many non-Cobol programmers are just catching up—but this is a cultural issue that does not stem from an intrinsic advantage of Cobol. Also, although Cobol has an advantage over Java in how it handles financial numbers, this does not mean Cobol is portable—Coyle’s fourth argument. A Cobol program can’t easily run in an embedded device or interact with a Web server. Finally, regarding Coyle’s fifth and final argument, the existence of the Cobol metalanguage might be of interest to theoreticians, but it does not drive any deployment decisions.

If these are the most compelling arguments for Cobol, I recommend using it strictly as an interface to legacy data. Convert the data to XML at your earliest convenience, and use Java to move the data across the Internet.
Cobol has a long and venerable history for building robust and scalable enterprise solutions. I do not share the widespread disdain against Cobol. It is a language well suited for its intended purpose, and it has survived precisely because it was such a good match for its problem domain. However, Cobol is becoming increasingly irrelevant because it is no longer a good match for the kind of enterprise systems in demand in the new millennium.

Java is More than a Language

Java is a programming language, and as programming languages go, it is quite nice. It is object oriented but without the awkwardness that often arises when object-oriented features are grafted onto an old language. It is also simple. Unlike some popular languages, which have become monsters that even their creators can no longer understand, every Java programmer can master the rules of the language (J. Gosling, B. Joy, and G. Steele, The Java Language Specification, Addison-Wesley, Reading, Mass., 1996). Finally, Java is safe. Because it was originally designed to deliver untrusted code over the Internet, many safeguards are built into the language and execution environment. These safeguards do more than just shield against security attacks; they also protect against programmer error.

However, had Java been nothing but a simple, safe, object-oriented language, it would have shared the same eventual obscurity that befalls so many programming languages. Java is different because it is not just a language—it is an entire programming platform. The Java platform contains libraries for networking, database access, Web programming, graphical user interfaces, multithreading, distributed computing, XML, and many other domains. These libraries make Java immediately useful for business applications. Furthermore, because Java code is inherently portable, the exact same code will run on a personal computer, workstation, or mainframe.

Solutions for a Range of Needs

Sun Microsystems originally developed
Cobol versus Java

Today, the compelling market niche for Java is enterprise applications. The fact that portable user interfaces are slower than native user interfaces is of no consequence in server-based applications. Portability, multithreading, network awareness, and database connectivity are key features in this arena, and Java excels in all of them. There are well-established solutions in this space, but corporations are willing to revisit them because of the Internet.

The Internet has changed how customers interact with businesses and how businesses interact with each other. Customers demand instant access to information, and costly leased lines and inflexible Electronic Data Interchange protocols no longer satisfy business-to-business data communication needs. Java technology has an edge in this arena because it was designed to be Internet-ready from the ground up. The Java security mechanisms can execute code over the Internet with clearly specified restrictions. Distributed processes can communicate through remote object protocols such as Corba and the Java-specific Remote Method Invocation protocol. Alternatively, they can exchange data in the XML format.

Adoption Risks
Java and the Java 2 Enterprise Edition (j2ee.sun.com/features/1999/09/J2EE.html) are far more open than traditional single-vendor solutions, but Sun Microsystems exerts a controlling influence that it is reluctant to relinquish. Currently, all major enterprise software vendors enthusiastically support Java standards, but their support could wane. Also, due to its novelty, there is a shortage of trained Java programmers. However, many colleges are switching to Java as the primary instructional language, because it is easier to learn and more useful than C++. Finally, the fact that Microsoft opposes Java’s growth might slow its adoption.

Despite these risks, Java offers a compelling set of technologies for developing business applications. In particular, the Java enterprise platform’s scalability, portability, Web integration, and wide vendor support greatly exceed the scope of Cobol-based solutions. Thus, I predict Cobol will steadily diminish in importance and Java will become the platform of choice for modern business applications.

Cay S. Horstmann is the chief technology officer at Preview Systems and a professor of computer science at San Jose State University. He received his MS in computer science from Syracuse University and his PhD in mathematics from the University of Michigan. He has authored many books and magazine articles about object-oriented programming, Java, and C++, including the Sun Microsystems Press book Core Java. Contact him at cay@horstmann.com.

Frank responds
While Java is certainly in a position to dominate the programming language landscape, recent developments indicate that the field is still open. Sun’s about-face on its commitment to establish an independent Java standardization process has angered many of its industry partners and cast a shadow over Java’s future as the language of e-commerce.

As companies move quickly to position themselves for e-commerce and business-to-business data interchange, those with a large Cobol investment (and there are many) will look more toward integration than replacement. A case in point is the recent alliance between Sun, Informix, and Merant (a leading Cobol vendor) to provide e-business solutions for Home Depot. With Cobol managing the data backbone and Java on the server, Home Depot was able to jumpstart an e-business solution by leveraging Cobol’s strength on the data side with Java’s network capabilities. Such synergies are now imperative in an era when component integration is the key to rapid development.

Where will Java and Cobol be in five years? If Sun can keep the Java juggernaut going, and if Cobol can leverage its native hierarchical data division to include easy access to XML (itself inherently hierarchical), expect to see both Java and Cobol prosper in the expanding world of e-business applications.