Essential Java resources

A comprehensive listing of must-have books, sites, blogs, events, and more

Skill Level: Introductory

Ted Neward (ted@tedneward.com)
Principal, ThoughtWorks
Neward & Associates

23 Jan 2009

The Java™ platform will be celebrating its 14th birthday soon and one side-effect when a successful and ubiquitous language reaches this kind of milestone is the widespread proliferation of libraries, tools, and ideas — this bonus can leave many newcomers to the Java language adrift in a sea of material. In this article, the author (a solid contributor to that overwhelming sea) tacks through the vast tides and presents a list of the key resources any up-and-coming Java developer should have.

Since its introduction to the programming community as a whole in 1995, the Java platform has evolved far beyond the "applets everywhere" vision that early Java pundits and evangelists imagined a Java world to be like. Instead, the Java world rose up to Swing, coalesced around servlets, rode that into J2EE, stumbled on EJB, sidestepped over to Spring and Hibernate, added generics and became more dynamic, then functionalized, and continues to grow in all sorts of interesting directions even as I write this.

Which leaves the Java programmer who didn't grow up along with the language somewhat overwhelmed.

As a speaker, blogger, consultant, and mentor, I am frequently asked by junior and intermediate programmers working in the Java space for resources to help them master this wide, complex, seemingly endless world. A few years ago, I took a stab at answering the question by posting a Recommended Reading List, but recently the Java editor at developerWorks asked me to put together a broader list of resources,
not focused exclusively on books — a good idea considering that the list had grown

to a point where a prospective consumer could easily spend the next decade
reading them and still not make a dent in the contents.

To avoid the appearance of bias or undue influence, I decided to throw a call to the
blogosphere for resources: Web sites, conferences, books, blogs, tools, anything
that seemed like it would be of interest and use to Java developers as they progress
in the space and might find helpful in navigating the sheer enormity of what's
available.

The response was enormous (for which I thank all those who replied).

A quick note by way of explanation before we begin: These resources are not
necessarily the most popular, most powerful, or most influential resources in the
Java ecosystem — that list might be entirely different (though I suspect many
resources would show up in both places); instead, these are the resources every
up-and-coming Java developer should have in their browser, on their bookshelf, or
on their calendar to attend. As with any kind of list of this sort, there is always room
for reasonable debate as to what should be included that wasn't or what was
included that shouldn't have been.

With no further ceremony, let us proceed.

Web sites and developer Web portals

The Web has clearly changed the nature of shared material and publications (and
it's not lost on me that you are reading this right now on the Web), so it seems
reasonable to begin the survey of key Java resources with key Web sites and
developer portals that every Java developer should keep an eye on.

In many cases, the value of a Web site and/or developer portal is not just in the
resources offered or the articles posted there, but also in the community that
surrounds that portal or Web site. The days of the static Web page and file download
are quickly fading into the background behind the more "active" model of forums,
threaded articles, and interactive discussion between not only the developers who
frequent that site and make up its community but also the community members and
the owners of the portal. Just as technology evolves and developers must evolve to
keep up with it, a portal must evolve to keep up with the changing needs of its
community; having an open relationship with its community is a critical component to
keeping a portal vibrant.

The list of Web sites (with URLs):

- [java.sun.com](http://java.sun.com)
  This is the place from which the JDK originates and is often the first site
that Java developers go to download a copy of the JDK and documentation. But beyond that, it is also a good source for information on the entire Java platform, including a very rich section on the JVM itself (Hotspot), which describes the internals of the JVM to a surprising depth.

- **java.net**
The Java.NET site is a smorgasbord of open-source projects, documentation, blogs, wikis, news, and other community resources. It's a motherlode of information on Java technology, possibly too much. For all that though, cruising through the list of projects can lead to some real gems such as the FEST-Swing project, which aims to create a "fluent interface" for building Swing applications, making them vastly simpler to create unit tests around.

- **InfoQ**
Created by Floyd Marinescu, founding editor-in-chief of The ServerSide, InfoQ differs from many of the other resources listed here in that it is explicitly technology-neutral, covering not only Java code but also .NET and Ruby and offering great coverage of agile approaches and insights. Though the commenting community on InfoQ isn't as large or as diverse as that of TheServerSide, the discussion is usually more tame and less hostile.

- **TheServerSide**
Often called "the the Slashdot of Java," TheServerSide was the first place that enterprise Java topics were openly questioned, hotly debated, and aggressively pursued. Some of the earliest thoughts around lightweight containers (as typified by Spring) occurred here and many of the individuals now considered to be "thought leaders" in the Java community emerged here. Although more recently the debates have sometimes taken on the character of shrill shouting matches, TSS will always be the original home of much of what turned into the Java ecosystem we now live within; as such, it will always be a source of information, historical if nothing else.

- **DZone**
DZone began life as a resource for Java developers, but more recently has begun to branch out into the other technology sectors, including .NET and Ruby, as well as several other topical "zones" of coverage. As such, while it's possible for the Java developer to focus entirely on just the Java zone, some cross-pollination does occur and it's the wise developer who will take advantage of that.

- **IBM® developerWorks**
If you're reading this article, then you've already found dW. In the interest of full disclosure, I must note that I have contributed articles other than this one to this site, but I can say with integrity that dW stands as a great resource for articles and material on Java coding and the JVM. In
particular, be sure to check out *Java theory and practice*, a long-running series by Brian Goetz, which includes a three-part description of the JVM garbage collector that introduces the core concepts of GC in a surprisingly approachable way.

- **IBM alphaWorks**
  alphaWorks, sister site to developerWorks, is where IBM researchers and fellows provide the tools and technologies they believe will define the mainstream tomorrow. Organized into just under a dozen different categories, alphaWorks offers Java developers a chance to try tools, languages, libraries, and other projects that in some cases are even ahead of the bleeding edge — they’re in the research stages. Note that many projects serve to prove a particular research point and then stop, leaving the project at whatever state of effectiveness it currently holds. *Caveat emptor*: Some projects are not open source and others don’t include source. But a fair number of alphaWorks projects end up making it into mainstream, either as a supported product from IBM, as an available tool for download from some of IBM’s open-source locations. In any event, keeping an eye on alphaWorks will be an eye toward where and how the Java ecosystem is evolving.

- **OnJava**
  OnJava is an independently run site that tracks events, blogs on conferences, and offers up various opinions on the enterprise Java world. Much of the information here can, in fact, also be found elsewhere, but this is true of almost all developer portals (including the ones mentioned above), and a developer looking to keep up with the various developments in the Java ecosystem could do far worse than to simply turn to the full list of articles on the site and start reading them one by one.

- **Artima**
  Created by Bill Venners, Artima has evolved into a huge collection of articles, blogs, and interviews, not all of it entirely focused on Java development, and has some of the biggest names in the Java space writing there. (Check out Bill Joy’s disbelief over the complexity of the Java language after Java 5 was released and the example he uses to justify his worries — Enum<> — as a classic example.) Well worth visiting.

Note that for many of the above, taking full advantage of the site/portal will require some form of registration. As of this writing, all of them are essentially free and require a username/password login solely so they can uniquely identify visitors and an e-mail address so they can send periodic topical newsletters or e-mails. (To the best of my knowledge, all will allow you to opt out of the e-mail, but I haven’t tried because I find that every so often, the newsletter tells me something I find interesting and didn't already know about, making them genuinely useful things to get in my
Weblogs

Blogs represent an easy way for developers to stay apprised of thoughts, memes, concepts, ideas, debates, and other intellectual discussions around technology and aware of those "thought leaders" who seem to be influential within the specific technology spheres. Even better, blogs can be consumed locally via feed readers that pull the articles/entries down locally so that they can be consumed even while offline. In fact, blogs have become so ubiquitous that specialized devices (like the Amazon Kindle) can make it easy to grab the info developers want.

The key is to determine which of the literally millions of blogs to read. Every developer will have their own preferences over which blogs to read and which not to, but some blogs to consider starting with include:

• The Java Posse
• The Java Specialists' Newsletter
• Martin Fowler's Bliki
• Any RSS feeds offered by the Web sites listed in the "Web sites" section

Remember that (most of) these are personal opinions; sometimes the opinions shared there will turn out later to be factually incorrect — never base a production decision solely on the information in a blog entry. But if used to kick the thought process out of a rut, blogs can be an incredibly powerful skills-enhancing resource.

Packages and/or libraries

Keeping track of all the libraries and packages in the Java ecosystem is a chore, one that can become all too easy for the Java developer to ignore. As a result, developers will sometimes miss the fact that something they're writing by hand already exists inside the platform (if only they'd stop to discover it!).

What follows is my list of the "Top Ten Java Packages Every Java Developer Should At Least Know Exists (and Ideally Know How To Use)"

• java.lang.reflect
  So many of Java's libraries and tools depend on the Reflection package that learning it has almost become a requirement (if for no other reason than to understand when using it will fail due to security policy or other kinds of infrastructural mismatches). Hand-in-hand with Reflection is also
Java 5 Annotations and \texttt{java.lang.annotation}.

- \texttt{javax.script}
  With the emphasis on dynamic languages sweeping through the Java ecosystem, knowing how to use scripting languages from within Java code can allow developers to extend their systems without having to go through the expensive compile-link-run-test cycle.

- \texttt{javax.management} and \texttt{java.lang.management}
  The management package, also known as Java Management Extensions (JMX), is a powerful tool for monitoring Java applications as they run; as such, developers should be familiar and comfortable with JMX in order to build software that can actually be used by system administrators. Don’t forget to look at the \texttt{jconsole} tool that ships with the JDK as an example of an application health dashboard, as both an example of what can be built and a useful tool in its own right. (Pay close attention to the script demo that builds a \texttt{jconsole} plug-in to enable scripting support inside \texttt{jconsole}, a powerful tool in the hands of any competent system administrator.)

- \texttt{java.util.concurrent}.*
  Concurrency is the biggest thing developers will face in the coming decade, and the principal weapon Java developers will wield against concurrency bugs will be the classes in this package and its subpackages (\texttt{.atomic} and \texttt{.locks}). Learning to use it will be just as critical, if not more so, than learning how to write a class. Learn it, live it, love it.

- \texttt{java.util}.
  The same is true of the classes in \texttt{java.util}. These are so frequently brought in, particularly the Collections classes, that it's arguable \texttt{java.util.*} should be automatically imported like \texttt{java.lang} is.

- \texttt{java.beans}
  JavaBeans are, without a doubt, one of the core elements used throughout the Java ecosystem, going under the less marketing-y name of ”POJO"s (plain old Java objects). They're still based on the JavaBeans code, API, and specification, so spending a little time with the classes in \texttt{java.beans} can only help write better POJOS.

- \texttt{java.util.logging}
  Diagnostic and audit logs very quickly become powerful tools for tracing the execution of running programs whether for debugging purposes or administration and monitoring purposes. My advice: Learn to use this package and its open-source cousin \texttt{log4j}).

- \texttt{java.security}.*
  The Java security APIs, the policy mechanism, and the permissions classes are a core part of what makes the Java environment a safe computing platform, but only if developers know how to take advantage of
Steer clear of the crypto APIs — don't use them until you've exhausted every other option available to you. They're not hard to use but getting crypto right (meaning without huge flaws) is something that takes years and a lot of peer reviews from other crypto experts. Use other mechanisms first, like SSL/TLS and the GuardedObject/SignedObject/SecureObject classes.

- **javax.xml.parsers.***, **javax.xml.bind**, and **javax.xml.transform.***
  XML looms largely in the Java developer's mindset and being able to parse it (both directly as via an object-XML library) and transform it (using XSLT) is a powerful skill to have, if not outright necessary.

- **javax.sound.***
  OK, it's not a required skill, but knowing how to pop sound effects every so often can help liven up otherwise mundane code. Besides, sometimes sound can be used to offer signals and feedback on a system when no display is immediately handy. And beyond that, there's just something morbidly fun about a machine throwing a stack trace and playing a high-pitched scream from your favorite horror film to let the user know that something bad just happened.

Of course, there are numerous other packages, both inside the JDK and outside the JDK that are worth learning, but mastering these will put you in good standing for most projects, not to mention most job interviews.

**Conferences**

Conferences continue to play a major role in the education of Java developers, particularly for those topics that haven't yet gathered enough critical mass to have filtered into large training companies' collections of topics yet. They also represent a golden opportunity for developers to meet with other developers of all skills and responsibilities, giving the attendee the chance to do something quite powerful (and entirely necessary): vent about how bad it is where you work.

Attending a conference also has a useful side effect (and one which can be parlayed into bigger and better things if actively massaged): *That of building up the attendee's developer peer group.* Thanks to the widespread ubiquity of e-mail, mailing lists (such as GoogleGroups or Yahoo Groups), and other lightweight groupware communication tools, it becomes easier than ever to keep track of the people met at a conference. Those friendships struck up over lunch or at the evening parties can last a lifetime and pay off over time, particularly when evaluating new technologies.

Let the words of a 30-year veteran conference attendee, speaker, organizer, and crasher (not me, but a friend of mine) be your guide: "Newbies to conferences go for the sessions; veterans go for the parties and the people. The parties help you meet people and the people help you advance your career."
Conferences include:

- **No Fluff, Just Stuff**
  This one I particularly heartily endorse and not just because I'm a regular speaker there, but because seven of the Top Ten JavaOne 2007 speakers are NoFluffJustStuff speakers (current or previous). Nowhere else will Java developers find the same intense focus on technical topics at a better price point or held in a more convenient location: Your hometown. (At more than 30 shows per year all over the North American continent — and with expansion into Europe and Asia on the horizon — chances are good you can drive to an NFJS show rather than fork out the thousands of dollars required to travel to some far-off city and stay in hotels.) It's also no surprise that a number of the books recommended in this article are written by authors who appear at the NoFluff shows, either: The organizers of NFJS go to great lengths to find the "primary source" on various Java and Agile Development topics.

- **JavaOne**
  It's large, it's impersonal, it's always held in San Francisco (which is either a pro or a con, depending on what your company's travel budget looks like), but no other gathering on Earth brings quite as many key Java movers-and-shakers together into one place like JavaOne does.

- **JAOO**
  Not to leave our European Java brethren out in the cold, JAOO is held once a year in Aarhus, Denmark, and has been running solid for close to two decades with no signs of slowing down. Not only does JAOO attract some top-notch speakers and authors, but it also has the relatively rare distinction of being a cross-technology conference, meaning that an attendee can sit in on a Java session in the morning, chat with some C++ developers over lunch, catch the latest developments on Microsoft's® Silverlight (or whatever else is new) in the afternoon, and share some beer with Smalltalkers in the evening. If you're looking to stretch your viewpoint, JAOO or its sister conference, QCon (held in conjunction with the InfoQ folks) is a great place to start. And for those of the Java brethren living "Down Under," JAOO came to Brisbane and Sydney for the first time this year (2008); all indications are that they will be back again in future years.

- **Devoxx**
  Formerly JavaPolis, this conference in Antwerp, Belgium, which is busting at the seams with phenomenal growth, is a great way to meet your European counterparts (and taste some excellent Belgian beer while you're at it). The speakers are top-notch and focus on cutting-edge topics. There's also a good-sized trade-show floor and many opportunities to interact with your fellow developers.
• **Jazoon**
   A relatively new conference to the scene, the Jazoon conference opened in Zurich (Switzerland) in 2007. With names like Martin Odersky (creator of Scala), Neal Gafter, and Simon Phipps (Sun's Chief Open Source Officer) on the rolls, Jazoon looks to be gathering an attractive crop of speakers. Focused entirely on Java topics and sitting right in the smack center of Europe, Jazoon looks to be a good conference for the European Java brethren to mark on their calendars.

Other conferences with a variety of different target audiences are out there, but the above target the Java developer directly. Developers looking to stretch their view of the world should consider attending an "alternative" technology conference such as the various Perl, Python, Ruby, or even .NET gatherings that happen all across the world; keeping your mind open to new ideas sometimes requires looking at an entirely new set of problems.

**Books**

No greater resource available to the developer is as easy to come by as the lowly and traditional book. Only as far away as your local bookstore (or if you live in the hinterlands, your Web browser and Amazon) lies a treasure trove of technical knowledge and experience ... if only you can sort through the myriad "me-too" "Java for Dummies" clones.

These tomes assume the reader already has a working knowledge of the Java language and tool set; beginning Java books are staggeringly common and pretty much all of them cover the same basic material. Selecting one becomes a choice of approach, author's writing style, and even sometimes whether or not the solutions to the exercises (if exercises are included) are included with the book's CD.

Once you're past the basics, consider these:

- **Effective Java (2nd Ed)** by Joshua Bloch
  The first book that should be on every Java developer's shelf, period, hands-down, bar none.

- **Java Concurrency in Practice** by Brian Goetz
  The second book that should be on every Java developer's shelf because every Java developer, like it or not, operates inside of a multithreaded environment.

- **Better, Faster, Lighter Java** by Justin Gehtland and Bruce Tate
  This slim read served as the "gateway drug" for many Java developers to take up the "lightweight container" environments, pioneered by Spring and filling a gap left behind by most big-iron EJB servers like WebSphere.
or other transaction-processing middleware. The authors do a fantastic job walking through a POJO-based development cycle, pointing out the strengths and weaknesses of the idea and how taking a light(er)weight approach to building enterprise applications with tools like Spring and Hibernate doesn’t mean reducing the functionality or the capability of the resulting system.

- **Effective Enterprise Java** by Ted Neward (me)
  Seventy-five ways to make sure your enterprise Java applications (whether they are written with Spring, EJB, Web services, or none of the above) don’t suck. Just as *Effective Java* is not intended for the beginning Java programmer, *Effective Enterprise Java* is not intended for the developer who is just learning the basics of the "enterprise Java" space.

- **Release It!** by Michael Nygard
  Anyone looking to write software intended for production servers needs to read this book — it is a great insightful look into the things that can go wrong in production and how to deal with potential failures, either by recovering from them or avoiding them entirely.

- Either **Inside the Java 2 Virtual Machine** by Bill Venners or **Component Development for the Java Platform** by Stuart Halloway
  Once the Java developer has learned the language, there comes a time when the underlying platform — the Java Virtual Machine itself, meaning subjects like garbage collection and Reference objects, ClassLoaders, threads and monitors, and so on — becomes the next great frontier to explore. Either of these books (or both) serve as a useful guide to doing just that. The biggest drawback to each is their relative age: *Inside* was published back in the Java 1.2 time frame and *Component-Based* shortly thereafter. Fortunately, most of the Java Virtual Machine remains unchanged since then, so by the time the reader discovers a detail that’s different from JDK 1.2, they will be ready to leave the book behind and move on to the next topic.

- **Patterns of Enterprise Application Architecture** by Martin Fowler and **Enterprise Integration Patterns** by Gregor Hohpe and Bobby Woolf
  Both of these are patterns books, speaking to design approaches and ideas, and as such assume most readers have read the original "Gang of Four" book, *Design Patterns* by Gamma, Helm, Johnson, and Vlissides. Patterns books generally help developers create stronger designs by giving them some basic conceptual building blocks to twist and shift into place but can easily lose their effectiveness when seen as a goal in and of themselves.

- **Java Power Tools** by John Smart
  Several people suggested this book, a gentle introduction to a variety of tools available in the Java ecosystem. Smart also offers a training course (Java Power Tools Bootcamp) around the book, but the easiest way most
developers will get the info will be through the dead-trees version.

- **The Pragmatic Programmer** by Andy Hunt and Dave Thomas
  This book should be on every programmer’s desk, regardless of the actual language or platform being used. This is a book for the ages and one programmers will be citing for decades to come. Think of it in the same class as Fred Brooks’ classic *The Mythical Man-Month.*

- **The Productive Programmer** by Neal Ford
  This recent release looks to be every bit as classic as *Pragmatic Programmer,* focusing on the various tips, tricks, and techniques that programmers can use to improve their productivity well beyond what the IDE gives them.

- **The Pragmatic Starter Kit**
  This consists of *Pragmatic Unit Testing (with JUnit)*, *Pragmatic Project Automation,* and *Pragmatic Source Control (with Subversion).* Each one covers the core of each topic in a light, focused discussion that cuts directly to the core of the subject without a lot of distractions. Project Automation, in particular, is a great appetizer to the *Ant in Action* book, but also covers the whys and whats of a good build process.

- **Ant in Action (2nd Ed)** by Erik Hatcher and Steve Loughran
  Ant is the premiere build utility for the Java platform (see the next section on tools for more details), and this book is by far the best introduction to the vast power that Ant represents. Along with learning how to use Ant, *Ant in Action* also demonstrates unit-testing, Continuous Integration, and production deployment practices, all from the perspective of the practical Java developer's day-to-day practices.

Tools

Just as there’s more to building a house than “grab a hammer and start nailing boards,” there’s more to Java development than “grab your IDE and start slinging code.” While certainly a good IDE is an enormous help to writing Java code, other tools can offer equal — if not more — value than the IDE.

In no particular order....

- **The IDE**
  Of course, the tools discussion begins with the IDE. In an effort to steer clear of commercial endorsements, both Eclipse and NetBeans are open source and free and thus offer an attractive "first steps" option.

- **Unit testing**
  The perennial favorite here is the original that started the unit testing
revolution, JUnit, but a few others have crept into popular usage, including TestNG, and not surprisingly JUnitPerf, among others.

- **Continuous Integration**
  A Continuous Integration server constantly checks the code out of source control, builds it, runs the unit tests against it, and reports any build failures to the developers, usually via e-mail. Several open-source versions are available, but one that has gathered a lot of attention is Hudson, a Continuous Integration server with a lot of plug-ins and high extensibility. The original is CruiseControl and for that reason alone commands respect.

- **Static analysis**
  FindBugs is an open-source static analysis tool that runs over Java code, performs deep n-way analysis to figure out all the possible code paths, and reports all sorts of errors and warnings it finds, based on a set of extensible rules. Developed by William Pugh (the same William Pugh who discovered a bug in the Java memory model), FindBugs' biggest claim to fame is its ability to analyze Java code for concurrency bugs, something that every programmer can appreciate.

- **Network packet tracing**
  WireShark (which formerly used to be called Ethereal) gives programmers a view of what's traveling across the network, giving them an opportunity to verify that what's moving across the wire is what's supposed to be moving across the wire, and that what's *not* supposed to be there (such as sensitive information or passwords in clear text), isn't.

- **Virtualization**
  A virtualization tool (or perhaps it is more accurate to call it a platform) like VMWare, Xen, VirtualBox, or VirtualPC offers programmers the ability to create an environment identical to the one the production machine will be running without losing the productivity of the environment of the host PC. It also provides a convenient way to have a home for trying "experimental" software without risking the machine that has to stay productive. Most of these have some kind of free option if they're not outright open-source projects.

The fact that some of these are commercial tools shouldn't scare you; if a tool saves you even just an hour a week, how long before that tool pays for itself? And if management doesn't buy that argument, a careful accounting of the time that could be saved by a particular tool (be that developer time or production diagnosis or execution time) generally goes a long way toward convincing them of the folly of a penny-pinching penny-wise/pound-foolish perspective.

**Thanking the contributors**

Thanks to all the readers of my blog (blogs.tedneward.com) who took time from their own schedules to suggest resources that should
Future trends to watch

Of course, it does Java developers no good to spend all this time studying the technologies of today if, when finished, the technologies of tomorrow have moved in and made everything learned suddenly obsolete. Keeping an eye on what's being tossed around as "the wave of the future" can be just as important, if only so that when the CTO of the company happens to be standing next to you in the elevator and asks, "What's all this I hear about functional languages, anyway? Should we be looking into that?", you can have a coherent answer.

In no particular order, some of the "futures" around the Java platform include:

- **Dynamic languages**
  Languages like Groovy and JRuby have spawned other projects worth watching like Clojure and Fan that experiment with the notion that a less-strongly-typed environment can provide better productivity and offer more options. This isn't a call to abandon the Java language, however: The best effects come in the combination of the two.

- **Functional languages**
  A trail being blazed by Scala, functional languages have the advantageous property of being implicitly concurrency-friendly due to their general tendency to encourage composable functions and lack of shared state (although they can definitely sometimes have a syntax that only a mother could love). Jaskell is a Haskell-on-JVM implementation and Clojure (again) fits in here, offering a vision of what software transactional memory looks like in production form.

- **Closures in Java**
  Adding closure support (also called lambdas or anonymous methods) will simplify certain parts of the Java coding experience and are being hotly debated for inclusion in the Java 7 release. Given how deeply this discussion is dividing the community, it behooves Java programmers to do a little research and see what all the fuss is about.

- **The DaVinci Machine**
  A new version of the JVM designed to be friendlier to other languages (like the dynamic and functional languages mentioned above), which may or may not end up being folded back into the mainstream JVM release.
Keeping an eye on this will help understand the pros and cons of running dynamic/scripting languages on the current JVM.

- **User interface developments**
  Between Flash/Flex, Silverlight, and JavaFX, the Web browser is about to get a makeover of stunning (and probably appalling) proportions. This doesn't even consider new server-side Web frameworks in the Java space or the odd-man-out like Google Web Toolkit (GWT). User interface has always been and continues to be (and will for the foreseeable future remain) downright sexy. Knowing one or two, not to mention a "rich client" library like Swing or SWT, will be of huge benefit when trying to build something more interactive than HTML pages can offer.

As with anything "futures"-related, keep in mind that what's being discussed today may turn out to be a wrong direction, a flash-in-the-pan, a momentary distraction, an influence on the industry, or the Next Big Thing; keep an open mind and a healthy sense of skepticism in equal doses.

**Summary**

Please feel free to contribute to this list — it will only yield a better list of resources. Even the passage of time will change its contents because the ecosystem continues to evolve and change and grow. This is a good thing; if we'd had to use the same Java environment we'd been given back in 1995, we'd have already moved on to something else.

Be active in the ecosystem: contribute patches, suggest enhancements, and if all else fails, go off and build your own better mousetrap. Who knows? Your project might make it onto this list five years from now.
Resources

- The developerWorks Java technology zone: Hundreds of articles about every aspect of Java programming.
- developerWorks blogs: Get involved in the developerWorks community.

About the author

Ted Neward
Ted Neward is a consultant with ThoughtWorks, a worldwide consultancy, and the principal of Neward & Associates, where he consults, mentors, teaches, and presents on the Java, .NET, XML Services, and other platforms. He resides near Seattle, Washington.

Trademarks

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.