
Preface

The Abstract Window Toolkit (AWT) provides the user interface for Java programs. Unless you want to construct your own GUI or use a crude text-only interface, the AWT provides the tools you will use to communicate with the user. Although we are beginning to see some other APIs for building user interfaces, like Netscape's IFC (Internet Foundation Classes), those alternative APIs will not be in widespread use for some time, and some will be platform specific. Likewise, we are beginning to see automated tools for building GUIs in Java; Sun's JavaBeans effort promises to make such tools much more widespread. (In fact, the biggest changes in Java 1.1 prepare the way for using the various AWT components as JavaBeans.) However, even with automated tools and JavaBeans in the future, an in-depth knowledge of AWT is essential for the practicing Java programmer.

The major problem facing Java developers these days is that AWT is a moving target. Java 1.0.2 is being replaced by Java 1.1, with many significant new features. Java 1.1 was released on February 18, 1997, but it isn't clear how long it will take for 1.1 to be accepted in the market. The problem facing developers is not just learning about the new features and changes in Java 1.1, but also knowing when they can afford to use these new features in their code. In practice, this boils down to one question: when will Netscape Navigator support Java 1.1? Rumor has it that the answer is "as soon as possible"—and we all hope this rumor is correct. But given the realities of maintaining a very complex piece of software, and the fact that Netscape is currently in the beta process for Navigator 4.0, there's a possibility that "as soon as possible" and "soon" aren't the same thing. In other words, you should expect Java 1.0.2 to stick around for a while, especially since Web users won't all replace their browsers as soon as Navigator has 1.1 support.

This state of affairs raises obvious problems for my book. Nothing would have made me happier than to write a book that covered AWT 1.1 only. It would be significantly shorter, for one thing, and I wouldn't have to spend so much effort pointing out which features are present in which release. But that's not the current reality. For the time being, programmers still need to know about 1.0.2. Therefore, this book covers both releases thoroughly. There are many examples using 1.0.2; many more examples that require 1.1; and more examples showing you how to update 1.0.2 code to use 1.1's features.

Sun has done a good job of maintaining compatibility between versions: 1.0 code runs under Java 1.1, with very few exceptions. All of the 1.0 examples in this book have been tested under Java 1.1. However, Java 1.1—and particularly, AWT 1.1—offer many advantages over older releases. If nothing else, I hope this book convinces you that you should be looking forward to the day when you can forget about writing code for Java 1.0.2.

New Features of AWT in Java 1.1

Having spent all this time talking about 1.0.2 and 1.1 and the transitional state we're currently in and having alluded briefly to the advantages of Java 1.1, you deserve a brief summary of what has changed. Of course, you'll find the details in the book.

Improved event handling

Java 1.1 provides a completely new event model. Instead of propagating events to all objects that might possibly have an interest, objects in Java 1.1 register their interest in particular kinds of events and get only the events they're interested in hearing. The old event model is still supported, but the new model is much more efficient.

The new event model is also important in the context of JavaBeans. The old events were pretty much specific to AWT. The new model has been designed as a general purpose feature for communication between software components. Unfortunately, how to use events in this more general sense is beyond the scope of this book, but you should be aware that it's possible.

New components and containers

Java 1.1 provides one new component, the `PopupMenu`, and one new container, the `ScrollPane`. Pop-up menus are a staple of modern user interfaces; providing them fixes a serious omission. `ScrollPane` makes it trivial to implement scrolling; in Java 1.0, you had to do scrolling “by hand.” In Java 1.1, you also get menu shortcuts (i.e., the ability to select menu items using the keyboard), another standard feature of modern user interfaces.

Java 1.1 also introduces a `LightweightPeer`, which means that it is possible to create “lightweight components.” To do so, you subclass `Component` or `Container` directly; this wasn’t possible in earlier releases. For simple operations, lightweight components are much more efficient than full-fledged components.

Clipboards

Java 1.1 lets you read from and write to the system clipboard and create private clipboards for use by your programs. The clipboard facility is a down payment on a larger data transfer facility, which will support drag and drop. (No promises about when drag and drop will appear.)

Printing

Java 1.1 gives components the ability to print.

The rest

There are many other new features, including more flexible use of cursors; the ability to use system color schemes, and thus make your program look like other software in the run-time environment; more image filters to play with; and the ability to prescale an image.

Deprecated Methods and JavaBeans

One of the biggest changes in Java 1.1 doesn’t concern the feature set at all. This was the addition of many new methods that differ from a method of Java 1.0 in name only. There are hundreds of these, particularly in AWT. The new method names show an important future direction for the AWT package (in fact, all of Java). The new names obey the naming conventions used by JavaBeans, which means that all AWT classes are potentially Beans. These conventions make it possible for an application builder to analyze what a component does based on its public methods. For example, the method `setFont()` changes the value of the component’s `Font` property. In turn, this means that you will eventually be able to build user interfaces and, in some cases, entire applications, inside some other tool, without writing any Java code at all. An application builder will be able to find out what it needs to know about any component by looking at the component itself, and letting you customize the component and its interactions with others.

Comments in the JDK source code indicate that the older method names have been “deprecated,” which means that you should consider the old names obsolete and avoid using them; they could disappear in a future release.

Reworking AWT to comply with JavaBeans is both necessary and inevitable. Furthermore, it’s a good idea to get into the habit of following the same conventions for your own code; the advantages of JavaBeans are much greater than the inconvenience of changing your coding style.

Other Changes in Java

Other new features are scattered throughout the rest of the Java classes, most notably, improvements in the networking and I/O packages and support for internationalization. Some new features were added to the language itself, of which the most important is “inner classes.” For the most part, I don’t discuss these changes; in fact, I stay away from them and base non-AWT code on the 1.0.2. release. Though these changes are important, covering the new material in AWT is enough for one book. If I used a new feature at this point, I would feel that I owed you an explanation, and this book is already long enough. A future edition will update the code so that it doesn’t rely on any older features.

What This Book Covers

The *Java AWT Reference* is the definitive resource for programmers working with AWT. It covers all aspects of the AWT package, in versions 1.0.2 and 1.1. If there are any changes to AWT after 1.1 (at least two patch releases are expected), we will integrate them as soon as possible. Watch the book’s Web site <http://www.ora.com/catalog/javawt/> for details on changes.

Specifically, this book completely covers the following packages:

- java.awt (1.0 and 1.1)
- java.awt.image (1.0 and 1.1)
- java.awt.event (new to 1.1)
- java.awt.datatransfer (new to 1.1)
- java.awt.peer (1.0 and 1.1)
- java.applet (1.0 and 1.1)

The book also covers some aspects of the sun.awt package (some interesting and useful layout managers) and the sun.audio package (some more flexible ways of working with audio files). It also gives a brief overview of the behind-the-scenes machinery for rendering images, much of which is in the sun.awt.image package.

Organization

The *Java AWT Reference* is divided into two large parts. The first part is a thorough guide to using AWT. Although this guide is organized by class, it was designed to flow logically, rather than alphabetically. I know that few people read a book like this from beginning to end, but if you want to, it’s possible. With a few exceptions, you should be able to read the early chapters without knowing the material that’s covered in the later chapters. You’ll want to read this section to find out how any chunk of the AWT package works in detail.

The second part is a set of documentation pages typical of what you find in most reference sets. It is organized alphabetically by package, and within each package, alphabetically by class. It is designed to answer questions like “What are the arguments to the `FilteredImageSource` constructor?” The reference section provides brief summaries, rather than detailed discussions and examples. When you use a typical reference book, you’re usually trying to look up some detail, rather than learn how something works from scratch.

In other words, this book provides two views of AWT: terse summaries designed to help you when you need to look something up quickly, and much more detailed explanations designed to help you understand how to use AWT to the fullest. In doing so, it goes well beyond the standard reference manual. A reference manual alone gives you a great view of hundreds of individual trees; this book gives you the trees, but also gives you the forest that allows you to put the individual pieces in context. There are dozens of complete examples, together with background information, overview material, and other information that doesn’t fit into the standard reference manual format.

About the Source Code

The source code for the programs presented in this book is available online. See <http://www.ora.com/catalog/javawt/> for downloading instructions.

Obtaining the Example Programs

The example programs in this book are available electronically in a number of ways: by FTP, Ftpmail, BITFTP, and UUCP. The cheapest, fastest, and easiest ways are listed first. If you read from the top down, the first one that works for you is probably the best. Use FTP if you are directly on the Internet. Use Ftpmail if you are not on the Internet but can send and receive electronic mail to Internet sites (this includes CompuServe users). Use BITFTP if you send electronic mail via BITNET. Use UUCP if none of the above works.

FTP

To use FTP, you need a machine with direct access to the Internet. A sample session is shown, with what you should type in **boldface**.

```
% ftp ftp.ora.com
Connected to ftp.ora.com.
220 FTP server (Version 6.21 Tue Mar 10 22:09:55 EST 1992) ready.
Name (ftp.ora.com:yourname): anonymous
331 Guest login ok, send domain style e-mail address as password.
Password: yourname@yourhost.com (use your user name and host here)
230 Guest login ok, access restrictions apply.
ftp> cd /published/oreilly/java/awt
```

```

250 CWD command successful.
ftp> binary (Very important! You must specify binary transfer for compressed files.)
200 Type set to I.
ftp> get examples.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for examples.tar.gz.
226 Transfer complete.
ftp> quit
221 Goodbye.
%
```

The file is a compressed *tar* archive; extract the files from the archive by typing:

```
% zcat examples.tar.gz | tar xvf -
```

System V systems require the following *tar* command instead:

```
% zcat examples.tar.gz | tar xof -
```

If *zcat* is not available on your system, use separate *gunzip* and *tar* commands.

```
% gunzip examples.tar.gz
% tar xvf examples.tar
```

Ftpmail

Ftpmail is a mail server available to anyone who can send electronic mail to, and receive it from, Internet sites. This includes any company or service provider that allows email connections to the Internet. Here's how you do it.

You send mail to *ftpmail@online.ora.com*. (Be sure to address the message to *ftpmail* and not to *ftp*.) In the message body, give the FTP commands you want to run. The server will run anonymous FTP for you and mail the files back to you. To get a complete help file, send a message with no subject and the single word "help" in the body. The following is a sample mail session that should get you the examples. This command sends you a listing of the files in the selected directory and the requested example files. The listing is useful if there's a later version of the examples you're interested in.

```

% mail ftpmail@online.ora.com
Subject:
reply-to yourname@yourhost.com      Where you want files mailed
open
cd /published/oreilly/java/awt
dir
mode binary
uuencode
get examples.tar.gz
quit
.
```

A signature at the end of the message is acceptable as long as it appears after “quit.”

BITFTP

BITFTP is a mail server for BITNET users. You send it electronic mail messages requesting files, and it sends you back the files by electronic mail. BITFTP currently serves only users who send it mail from nodes that are directly on BITNET, EARN, or NetNorth. BITFTP is a public service of Princeton University. Here’s how it works.

To use BITFTP, send mail containing your FTP commands to *BITFTP@PUCC*. For a complete help file, send HELP as the message body.

The following is the message body you send to BITFTP:

```
FTP ftp.uu.net NETDATA
USER anonymous
PASS yourname@yourhost.edu Put your Internet email address here (not your BITNET address)
CD /published/oreilly/java/awt
DIR
BINARY
GET examples.tar.gz
QUIT
```

Once you’ve got the desired file, follow the directions under FTP to extract the files from the archive. Since you are probably not on a UNIX system, you may need to get versions of *uudecode*, *uncompress*, *atob*, and *tar* for your system. VMS, DOS, and Mac versions are available. The VMS versions are on *gatekeeper.dec.com* in */pub/VMS*.

UUCP

UUCP is standard on virtually all UNIX systems and is available for IBM-compatible PCs and Apple Macintoshes. The examples are available by UUCP via modem from UUNET; UUNET’s connect-time charges apply.

If you or your company has an account with UUNET, you have a system somewhere with a direct UUCP connection to UUNET. Find that system, and type:

```
uucp uUNET\!~/published/oreilly/java/awt/examples.tar.gz yourhost\!~/yourname/
```

The backslashes can be omitted if you use the Bourne shell (*sh*) instead of *csh*. The file should appear some time later (up to a day or more) in the directory */usr/spool/uucppublic/yourname*. If you don’t have an account, but would like one so that you can get electronic mail, contact UUNET at 703-204-8000.

Once you’ve got the desired file, follow the directions under FTP to extract the files from the archive.

Other Java Books and Resources

This book is part of a series of Java books from O'Reilly & Associates that covers everything you wanted to know, and then some. The *Java AWT Reference* is paired with the *Java Fundamental Class Reference* to document the entire Core Java API. Other books in the series provide an introduction (*Exploring Java*) and document the virtual machine (*Java Virtual Machine*), the language (*Java Language Reference*), multithreaded programming (*Java Threads*), and network programming (*Java Network Programming*), with more to come. *Java in a Nutshell* is another popular Java book in the Nutshell series from O'Reilly. For a complete up-to-date list of the available Java resources, refer to <http://www.ora.com/info/java/>.

In addition to the resources from O'Reilly, Sun's online documentation on Java is maintained at <http://www.javasoft.com/nav/download/index.html>. Information on specific Java-capable browsers can be found at their respective Web sites, which are listed in Table 1. More are sure to be on the way. (Some browsers are platform specific, while others are multi-platform.)

Table 1: Popular Web Browsers that Support Java

Browser	Location
Netscape Navigator	http://home.netscape.com/comprod/products/navigator/
Microsoft's Internet Explorer	http://www.microsoft.com/ie
Sun's HotJava	http://www.javasoft.com/HotJava/
Oracle's PowerBrowser	http://www.oracle.com/products/websystem/powerbrowser
Apple's Cyberdog	http://cyberdog.apple.com/

Newsgroups also serve as a discussion area for Java-related topics. The *comp.lang.java* group has formally split into several others. The new groups are:

<i>comp.lang.java.advocacy</i>	<i>comp.lang.java.machine</i>
<i>comp.lang.java.announce</i>	<i>comp.lang.java.programmer</i>
<i>comp.lang.java.beans</i>	<i>comp.lang.java.security</i>
<i>comp.lang.java.databases</i>	<i>comp.lang.java.setup</i>
<i>comp.lang.java.gui</i>	<i>comp.lang.java.softwaretools</i>
<i>comp.lang.java.help</i>	<i>comp.lang.java.tech</i>

For folks without time to dig through all the noise, *Digital Espresso* provides a periodic digest of the newfeed at <http://www.io.org/~mentor/DigitalEspresso.html>. A list of

Java FAQs is at <http://www-net.com/java/faq/>; one of the most interesting is *Cafe Au Lait*, at <http://sunsite.unc.edu/javafaq/>. (*Cafe Au Lait* is written by Elliotte Rusty Harold, author of *Java Network Programming*.)

Local Java user groups are another good resource. (Having founded one myself, I'm biased.) What they offer varies greatly, but unless you look at one, you are potentially leaving out a vast resource for knowledge and experience. Lists of area user groups are available from JavaSoft at <http://www.javasoft.com/Mail/usr-grp.html>; also check out the Sun User Group's Special Interest Group for Users of Java at <http://www.sug.org/Java/groups.html>. In addition to the usual monthly meetings and forums, some maintain a mailing list for technical exchanges.

Security is a major issue with Java. If you are interested in reading more about Java security issues, Princeton University's Safe Internet Programming Web site at <http://www.cs.princeton.edu/sip/News.html> is an excellent resource.

About Java

Java is one of 13,000 islands that makes up Indonesia, whose capital is Jakarta (see Figure 1). It is home to about 120 million people with an area about 50,000 square miles. While on the island, you can hear traditional music such as gamelan or angklung. The island also has a dangerous volcano named Merapi, which makes up part of the Pacific "Ring of Fire." In 1891, fossils from *Pithecanthropus erectus*, better known as "Java man" (*homo javanensis*) were discovered on the island by Eugene Dubois.

Java's main export is a coffee that is considered spicy and full bodied, with a strong, slightly acidic flavor. O'Reilly has shown good taste in staying away from the pervasive coffee theme in its book titles and cover designs. (However, if you're ever in Sebastopol, check out the coffee at AromaRoasters in Santa Rosa.)

Conventions Used in This Book

Italic is used for:

- Pathnames, filenames, and program names
- Internet addresses, such as domain names and URLs

Typewriter Font is used for:

- Anything that might appear in a Java program, including keywords, method names, variables names, class names, and interface names

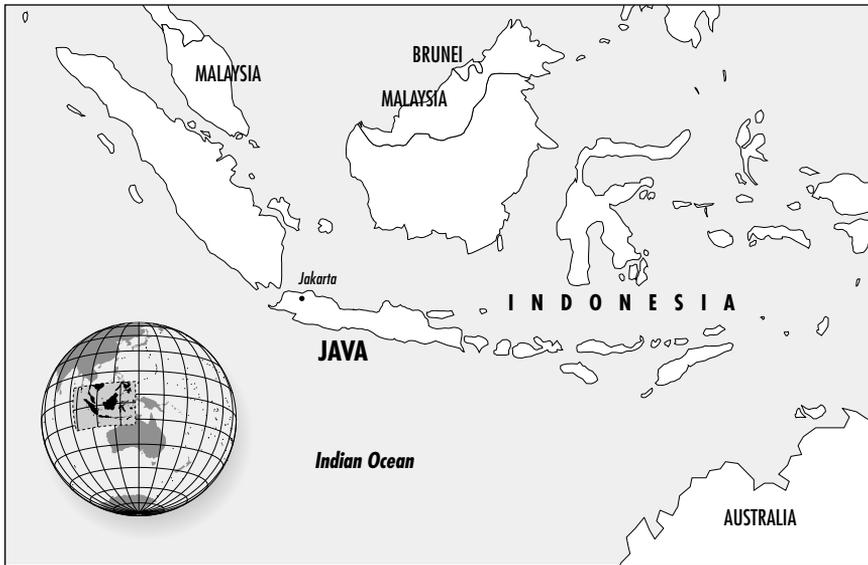


Figure 1: Map of Java, Indonesia

- Command lines and options that should be typed verbatim on the screen
- Tags that might appear in an HTML document

To sort out the potential for confusion between different versions, I use the following dingbats throughout the book:

- ★ Identifies a method, variable, or constant that is new in Java 1.1.
- ☆ Identifies a method from Java 1.0 that has been deprecated. Deprecated methods are available for compatibility but may disappear in a future release. These methods are tagged with the `@deprecated` flag, which causes the Java 1.1 compiler to display a warning message if you use them.

Request for Comments

We invite you to help us improve the book. If you have an idea that could make this a more useful resource, or if you find a bug in an example program or an error in the text, please let us know by sending email to bookquestions@ora.com.

As Java continues to evolve, we may find it necessary to issue errata for this book or to release updated examples or reference information. This information will be found at the book's Web site <http://www.ora.com/catalog/javawt/>.

Acknowledgments

I am grateful to many people who helped me along while working on this book, especially my wife, Lisa, and her patience during this whole process. A special thanks goes to our Old English sheep dog, Sir Dudley Fuzzybuns McDuff for gladly sharing the house with me during the entire process. I am grateful to the people at Sun who helped me become involved with Java so early on: Pete Seymour, Anne Pettitt, Tom McGinn, and Jen Sullivan-Volpe. I am also grateful to my employers, Rapid Systems Solutions (when I started) and the MageLang Institute (when I finished), who let me work on the book. Another thanks goes out to Dale Carnegie Training and John Captain, whose human relations class helped me feel comfortable with public speaking, without which I would not have become immersed in Java so quickly.

Particular thanks are owed to the technical reviewers: Yadu Zambre, Andy Cohen, David Flanagan, Jen Sullivan-Volpe, and Dan Jacobs. All of them performed an invaluable service with their thorough reviews and helped spot my errors and omissions. It seemed everyone contributed many bits of text that eventually found their way into the final product.

Random thanks go out to the many people on the Internet who I never met but provided valuable information, from the newsgroups and mailing lists: Simon “FISH” Morris, Mike Gallant, Eric Link, and many others whose names I did not write down.

Bits and pieces of various figures were borrowed from David Flanagan’s book, *Java in a Nutshell*, and Patrick Niemeyer’s and Joshua Peck’s book, *Exploring Java*. The class hierarchy diagrams come from David’s book. These diagrams were based on similar diagrams by Charles L. Perkins. His original efforts are available at <http://rendezvous.com/java/>.

For the gang at O’Reilly who gave me the opportunity to write this work, I thank everyone who helped along the way. For series editor, Mike Loukides, thanks for all your time and effort, especially with the early drafts. Best of luck to Mike and Judy with their new bundle of joy, Alexandra. Special thanks to Jonathan Knudsen who updated the reference section for the new release. Thanks to Nancy Crumpton and John Files for book production and project management, and to Trina Jackson, Paula Ferguson, and Andy Oram who helped during the review stages. Thanks also to the O’Reilly Tools group, Ellen Siever, Erik Ray, and Lenny Mueller; to Seth Maislin, the indexer; and David Futato and Danny Marcus who handled the proofreading and QCs.

The final product is much better because of their help.

