Java™ 2 PLATFORM

Jamie Jaworski

Expert Insight

The book presents focused explanations of the core features and complexities of Java 2 Platform, including:

- JFC and the Swing Toolkit
- New Glasgow JavaBeans APIs
- Network programming, RMI, and CORBA

Powerful Software

The companion CD-ROM includes:

- Borland® JBuilder Publisher's Edition
- Kawa
- Complete collection of code and applications from the book

Authoritative Advice

Learn professional Java techniques from Jamie Jaworski, bestselling author and developer of advanced systems for the United States Department of Defense.

SAMS

Unleashed

THE COMPREHENSIVE SOLUTION!

Unleash the power of Java™ 2 Platform

The *Unleashed* series takes you beyond the introductory discussions of the technology, giving you practical advice and in-depth coverage. With these extensive guides, you'll obtain the skills, understanding, and breadth of knowledge to unleash the full potential of Java 2 Platform.

Utilize the tools to finding the solutions you need

- **Borland JBuilder Publisher's Edition and Tutorials**
- **Kawa**
- **Complete collection of code and applications from the book**

Jamie Jaworski is a professional programmer who develops advanced systems for the United States Department of Defense. He has used Java in a wide range of projects, including a terrain analysis program and a genetic algorithm demonstration. Mr. Jaworski is also the author of *Java 1.1 Developer's Guide*.

Detailed information on how to...

- Use Swing's pluggable look and feel to create custom GUIs
- Develop advanced graphics using the new Java 2D and Java 3D APIs
- Integrate drag-and-drop and clipboard capabilities into your applets and applications
- Work with JAR files and digital certificates
- Use the new Glasgow JavaBeans APIs
- Simplify beans communication using InfoBus
- Print from applets and applications
- Develop multi-language programs
- Add audio, video, and animation features
- Replace CGI programs with servlets
- Program TCP/IP sockets
- Work with content and protocol handlers
- Integrate naming and directory services
- Implement remote activation and persistence with Java RMI

Java—Programming
Java 2 Platform
Intermediate–Advanced

SAMS
www.samspublishing.com

Java 2 Platform

Jamie Jaworski

SAMS

Unleashed

Java 2 Platform Unleashed

International Standard Book Number: 0-672-31631-5

Library of Congress Catalog Card Number: 98-83248

Printed in the United States of America

First Printing: April 1999

01 00 99 4 3

Executive Editor
Tim Ryan

Acquisitions Editor
Jeff Taylor

Development Editors
Jon Steever
Tim Ryan

Managing Editor
Patrick Kanouse

Project Editor
Rebecca Mounts

Copy Editors
Sara Bosin
Carol Bowers
Sean Medlock
San Dee Philips
Christina L. Smith
Kate Talbot
Rhonda Tinch-Mize

Indexer
Christine Nelsen

Technical Editor
Jeff Perkins

Software Development Specialist
Dan Scherf

Production
Michael Henry
Linda Knose
Tim Osborn
Staci Somers
Mark Walchle

CHAPTER 23

Integrating Speech and Telephony Capabilities

In This Chapter

In the previous chapters of Part 6, you learned a variety of approaches to making user interfaces more lively and attractive by incorporating multimedia features. Some of the most interesting new technologies in user interface design allow users and computers to talk to each other. Speech recognition enables users to translate speech into commands, data, and text, and simplifies the interface between the user and the computer. Speech synthesis enables the computer to provide output to the user via the spoken word. Although these technologies have been available for a few years, they haven't yet been integrated into mainstream software applications. The Java Speech API, which is being developed by Sun and several other companies, will bridge this gap and make speech capabilities standard features in Java applications.

One of the most common devices that we use to speak and listen is the telephone. Mobile devices, such as the Nokia 9000i, are being developed that integrate computer and telephone capabilities. The Java Telephony API is designed to incorporate telephony features into Java applications. This API will let you place and answer calls from within a Java application, provide touch-tone navigation, and manage multiple telephone connections. A number of advanced telephony capabilities are also being planned.

In this chapter, you'll preview the Speech and Telephony APIs and learn about the capabilities they will provide. You'll learn how Java Speech will be used to add speech recognition and synthesis to your programs, and how Java Telephony will be used to develop sophisticated telephony applications. When you finish this chapter, you'll understand what these two important APIs can bring to your Java programs.

The Java Speech API

The Java Speech API provides the capability to incorporate speech technology (both input and output) into Java applets and applications. When it becomes available, it will support speech-based program navigation, speech-to-text translation, and speech synthesis. The Java Speech API is being developed by Sun in collaboration with IBM, AT&T, Texas Instruments, Phillips, Apple, and other companies. At the time of this writing, the following specifications had been developed:

- Java Speech API Specification—Defines the packages used to implement basic Java Speech capabilities, speech recognition, and speech synthesis.
- Java Speech Programmer's Guide—Describes how to use the Java Speech API to develop speech-enabled applications.

- Java Speech Grammar Format (JSGF) Specification—Describes the JSGF and explains how it is used to create platform-independent speech recognition grammars. These grammars identify the words that a user speaks and their meaning in particular program contexts.
- Java Speech Markup Language (JSML) Specification—Describes the role of JSML and shows how it's used to mark up text documents for use with speech synthesizers.

These products are available at the Java Speech Web site, located at `http://java.sun.com:80/products/java-media/speech/index.html`.

The Speech API consists of the following three packages:

- `javax.speech`—Provides classes and interfaces that support audio connectivity and manage the use of speech processing engines.
- `javax.speech.recognition`—Provides classes and interfaces that support speech recognition.
- `javax.speech.synthesis`—Provides classes and interfaces that support speech synthesis.

The `javax.speech` package consists of the following classes and interfaces:

- `Central`—Class that provides central access (via `static` methods) to all capabilities of the Speech API.
- `Engine`—Interface that is implemented by speech recognition and synthesis engines.
- `EngineAttributes`—Defines the attributes that are supported by an `Engine` object.
- `EngineCentral`—Provides the operating modes of a speech `Engine` in terms of `EngineModeDesc` objects.
- `EngineModeDesc`—Defines an `Engine` operating mode.
- `EngineList`—A collection of `EngineModeDesc` objects.
- `AudioManager`—Interface that defines methods for controlling audio input and output and managing audio events.
- `VocabManager`—Interface that defines methods for managing words that are used by a speech engine.
- `Word`—Encapsulates speakable words.

- `SpeechEvent`—The superclass of all speech events.
- `AudioEvent`—Subclass of `SpeechEvent` that is generated by speech `Engine` objects based on audio input and output processing.
- `AudioListener`—Defines methods for handling `AudioEvent` objects.
- `AudioAdapter`—Implementation of the `AudioListener` interface.
- `EngineEvent`—Reports changes in speech `Engine` status.
- `EngineListener`—Defines methods for handling `EngineEvent` objects.
- `EngineAdapter`—Implementation of the `EngineListener` interface.

The following sections cover the `javax.speech.recognition` and `javax.speech.synthesis` packages.

Speech Recognition

Speech recognition allows computers to listen to a user's speech and determine what the user has said. It can range from simple, discrete command recognition to continuous speech translation. Although speech recognition has made much progress over the last few years, most recognition systems still make frequent errors. These errors can be reduced by using better microphones, reducing background noise, and constraining the speech recognition task. Speech recognition constraints are implemented in terms of grammars that limit the variety in user input. The JSGF provides the capability to specify *rule grammars*, which are used for speech recognition systems that are command- and control-oriented. These systems only recognize speech as it pertains to program operation and do not support general dictation capabilities.

Even with the constraints posed by grammars, errors still occur and must be corrected. Almost all applications that employ speech recognition must provide error-correction facilities.

Speech recognition is supported by the `javax.speech.recognition` package, which consists of 15 interfaces and 19 classes. These classes and interfaces make up four major groups: `Recognizer`, `Grammar`, `Rule`, and `Result`.

The `Recognizer` interface extends the `Engine` interface to provide access to a speech recognition engine. `RecognizerAttributes` and `RecognizerModeDesc` are used to access the attributes and operational modes of the `Recognizer`. `Recognizer` objects generate `RecognizerEvent` objects as they change state during speech processing. The `RecognizerInterface` defines methods for handling these events. The

`RecognizerAdapter` class provides a default implementation of this interface. The `AudioLevelEvent` is generated as a result of a change in the audio level of a `Recognizer`. The `RecognizerAudioListener` interface defines methods for handling this event, and the `RecognizerAudioAdapter` class provides a default interface implementation.

The `Grammar` interface provides methods for handling the grammars used by a `Recognizer`. It is extended by `RuleGrammar` and `DictationGrammar`, which support rule grammars and dictation grammars. The `GrammarSyntaxDetail` class is used to identify errors in a grammar. The `GrammarEvent` class is used to signify the generation of `Result` object that matches a `Grammar`. The `GrammarListener` interface defines methods for handling this event, and the `GrammarAdapter` class provides a default implementation of this interface.

The `Rule` class encapsulates rules that are used with a `RuleGrammar`. It is extended by `RuleAlternatives`, `RuleCount`, `RuleName`, `RuleParse`, `RuleSequence`, `RuleTag`, and `RuleToken`, which specify different aspects of grammar rules.

The `Result` interface provides access to the recognition results generated by a `Recognizer`. The `FinalResult` interface is used for results that have been finalized (accepted or rejected). It is extended by `FinalRuleResult` and `FinalDictationResult` to provide additional information for `RuleGrammar` and `DictationGrammar` objects. The `ResultToken` interface provides access to a single word of a `Result`. The `ResultEvent` class is used to signal the status of results that are generated by a `Recognizer`. It is handled via the `ResultListener` interface and the default `ResultAdapter` class. The `GrammarResultAdapter` class is used to handle both `ResultEvent` and `GrammarEvent` objects.

The `SpeakerManager` interface is used to manage speaker profiles for a `Recognizer`.

Speech Synthesis

Speech synthesis is the opposite of speech recognition. It allows computers to generate spoken output to users. It can take the form of bulk text-to-speech translation, or of intricate speech-based responses that are integrated into an application's interface.

Speech synthesis systems must satisfy the two main requirements of understandability and naturalness. Understandability is improved by providing adequate pronunciation information to speech generators. This eliminates "guesses" on the part of the speech synthesizer. JSML is used to provide pronunciation information, as required. Naturalness is improved by using a non-mechanical voice and managing emphasis, intonation, phrasing, and pausing. JSML also provides markup capabilities that control these speech attributes.

When you're synthesizing speech, it is often desirable to select attributes of the voice that is generated. For example, you might want to choose between male and female voices or old and young voices. The Speech API provides control over these features. In addition, text that is to be synthesized can be marked up with event markers that cause events to be generated as they are processed. Event handlers can be designed to manipulate graphical interface components in synchronization with speech synthesis. For example, you can design a speaker's face that changes facial expressions as it "talks."

The flexibility of the synthesis component of the Speech API is provided by JSML. JSML, like HTML, is an SGML-based markup language. JSML allows text to be marked up using the following synthesis-related information:

- Paragraph and sentence boundaries
- Pronunciation of words and other text elements
- Pauses
- Emphasis
- Pitch
- Speaking rate
- Loudness

These capabilities may not have your computer reading poetry, but they will allow you to greatly enhance any speech that it generates. Listing 23.1 provides an example of a JSML file.

LISTING 23.1. AN EXAMPLE JSML FILE.

```
<?XML version="1.0" encoding="UCS-2"?>
<JSML>
<PARA><SENT>This is the <EMP>first</EMP> sentence of
the first paragraph.</SENT> <SENT>This is the second
sentence.</SENT><BREAK SIZE = "large"/><SENT>This is the
<EMP>last</EMP> sentence of this paragraph.</SENT></PARA>

<PARA><PROS RATE="+10%" VOL=".9"><SENT>This is the second
paragraph.</SENT></PARA>
</JSML>
```

The first line identifies the file as being XML version 1.0. The <JSML> and </JSML> tags surround the JSML markup. Within these tags are two paragraphs marked by the <PARA> and </PARA> tags. The first paragraph consists of three sentences marked by <SENT> and </SENT>. The second paragraph has a single sentence.

The word `first` is surrounded by `<EMP>` and `</EMP>`. This signifies that the word "first" should be emphasized. The `<BREAK SIZE="large"/>` tag specifies that a long pause should occur between the second and third sentences.

In the second paragraph, the `<PROS RATE="+10%" VOL=".9">` tag is an example of a *prosody tag*. Prosody tags control the timing, intonation, and phrasing of speech. The `RATE` attribute specifies that the speech rate should be increased by 10%. The `VOL` attribute specifies that the volume of speech should be set at 90% of its maximum.

The example JSML file illustrates the use of JSML tags. However, the markup language is much richer than indicated by the example. For more information on JSML, download the JSML specification from the Java Speech Web site.

JSML is a subset of the eXtensible Markup Language (XML), which is a subset of the Standard Generalized Markup Language (SGML). JSML looks like HTML with different tags. (HTML is also a subset of SGML.) Figure 23.1 shows the relationship between JSML, XML, HTML, and SGML.

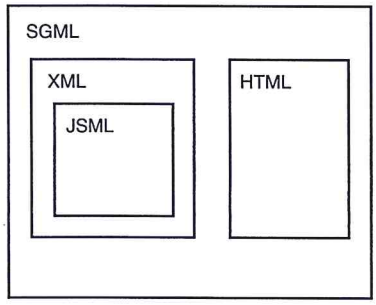

FIGURE 23.1.
The relationship between JSML and other markup languages.

The Java Speech API supports speech generation via the `javax.speech.synthesis` package. This package provides the following five interfaces and six classes:

- `Synthesizer`—Extends `Engine` to implement a synthesizer engine.
- `SynthesizerAttributes`—Provides access to the control attributes of a `Synthesizer`.
- `SynthesizerModeDesc`—Used to specify an operational mode for a `Synthesizer`.
- `SynthesizerEvent`—Event generated by a `Synthesizer` as it changes state.
- `SynthesizerListener`—Defines methods for handling `SynthesizerEvent` objects.
- `SynthesizerAdapter`—An implementation of the `SynthesizerListener` interface.
- `Speakable`—An interface for providing JSML text to a `Synthesizer`.

- `SpeakableEvent`—Event generated by a `Speakable` object that identifies the state of JSML processing.
- `SpeakableListener`—Defines methods for handling `SpeakableEvent` objects.
- `SpeakableAdapter`—An implementation of the `SpeakableListener` interface.
- `Voice`—Allows the age and gender of a voice to be specified.

To use the synthesizer package, invoke the `createSynthesizer()` method of the `Central` class to create a `Synthesizer` object. Pass an argument of the `SynthesizerModeDesc` class to `createSynthesizer()` to specify the `Synthesizer` mode. Invoke the `allocate()` method of the `Synthesizer` (inherited from `Engine`) to start up the `Synthesizer`'s engine. After that, use the `speak()` and `speakPlainText()` methods to put text in the `Synthesizer`'s input queue.

The Java Telephony API

The Java Telephony API (JTAPI) is a set of APIs that provide telephony capabilities for Java applications. It supports basic telephony capabilities, such as call placement and call answering, and advanced capabilities, such as call centers and media streams. JTAPI provides both direct control over telephony resources and indirect access through networked resources. This means that you can create server applications that provide telephony resources over a network, and client applications that use these resources.

The JTAPI consists of the following 18 packages:

- `javax.telephony`—Provides the core classes and interfaces used by all telephony applications.
- `javax.telephony.capabilities`—Provides support for basic call and connection capabilities.
- `javax.telephony.events`—Defines the basic events used in all telephony applications.
- `javax.telephony.callcenter`—Provides support for developing call center applications.
- `javax.telephony.callcenter.capabilities`—Provides capabilities such as routing and automated call distribution used in call center applications.
- `javax.telephony.callcenter.events`—Defines the events used in call center applications.
- `javax.telephony.callcontrol`—Provides call control features, such as call hold, call transferring, and conferencing.

CHAPTER 51

Java Platforms and Extensions

In This Chapter

Java's immense popularity has resulted in its implementation on a variety of operating system platforms, including Solaris, Windows NT, Windows 98, Windows 95, Windows CE 2.0, Linux, Macintosh, and others. Java's omnipresence has made it a platform within a platform. In this chapter, you'll find out about the Java Platform and examine each of its parts. You'll also look at Personal Java and Embedded Java and learn about the Java classes developed by Netscape, Microsoft, and other software vendors. When you finish this chapter, you'll be familiar with the Java Platform and the capabilities provided by its popular extensions.

Operating System Platforms Supporting Java

Java's service mark is "Write Once, Run Anywhere." This is not an overstatement—Java runs on every major operating system platform. Sun's platform of choice is their own Solaris, and Java runs on both the SPARC and Intel x86 versions of it. Next in line is Microsoft Windows. Java runs on Windows NT, Windows 95, Windows 98, and Windows CE 2.0. And with the help of IBM's Applet Development Kit (ADK), it also runs on Windows 3.1. Of course, IBM didn't stop with Windows 3.1. It has ported Java to its AIX, OS/2 Warp, OS/390, and OS/400 operating systems.

Java runs on the Macintosh and on every major brand of UNIX, including the enormously popular Linux. Java also runs on UnixWare, VxWorks, OS9, Inferno, Chorus, BeOS, and RiscOS. JavaPC is currently being developed to allow Java applications to run on DOS.

As you would expect, Java runs on JavaOS. This means that Java will soon be running on hand-held computers and consumer electronic devices. Who knows? It might be in your next TV or toaster!

By definition, Java runs on all network computers.

The Java Platform

Because Java is ubiquitous, it is an operating platform in its own right. But what exactly is the Java Platform? JavaSoft has taken special care to define it.

The Java Platform is the Java Virtual Machine and a portion of the Java API, referred to as the *Core API*. The Core API is the minimum subset of the Java API that must be supported by the Java Platform. All other Java API classes and interfaces that are not in the Core API are in the *Standard Extension API*. Three special APIs, PersonalJava,

EmbeddedJava, and JavaCard, are subsets of the Core API. Figure 51.1 shows the relationship between the Java Platform, Personal Java, Embedded Java, and Java Card.

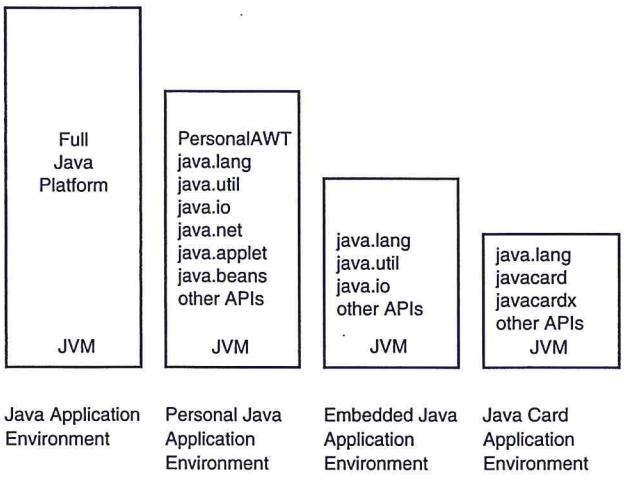

FIGURE 51.1.
The Java application environments.

Core API

Because the Core API must be supported by all Java Platforms, it is the API to which most Java applications will be written. The Core API is quite extensive. It includes all of the packages of the JDK 1.2, plus other packages that will be added to the next JDK 1.3 release. These packages include the following:

- Java 3D API—Provides a 3D library that supports VRML.
- Java Animation API—Supports 2D animation.
- Java Media Framework (JMF)—Provides a common API for multimedia players, multimedia capture, and multimedia conferencing.
- Java Commerce API—Part of this will be Core, and the rest will be Standard Extension. This provides the capability to perform electronic financial transactions. The part of the Java API to be included in the Core is referred to as *Java Wallet* and supports the client-side functions of electronic commerce.

Other APIs that are part of the Standard Extension will be added to the Core after they are widely supported.

Standard Extensions

The Standard Extension API includes all packages that are not part of the Core API. It's not supported on all Java platforms, but it contains the most exciting new API developments. The following Standard Extension API packages are being developed:

- Java Naming and Directory Interface (JNDI)—Provides access to common naming and directory services, such as the Lightweight Directory Access Protocol or LDAP.
- Java Server and Servlet APIs—The Server API facilitates the development of Internet servers, such as Web servers. The Servlet API supports the development of server-side applications.
- Java Commerce API—The Standard Extension part of the Commerce API includes classes and interfaces for implementing the server-side components of electronic financial transactions.
- Java Collaboration API—Supports the development and sharing of multiuser applications over a network.
- Java Telephony API—Provides access to telephonic communication and supports call control.
- Java Speech API—Supports speech recognition and synthesis.
- Java Management API—Supports network and system management functions.

JavaSoft is continually adding to its list of new APIs that are being developed. If the preceding list does not quench your thirst for Java, check the Java API Overview and Schedule Web page (`http://www.javasoft.com/products/api-overview/index.html`) for new developments.

Personal Java

Personal Java has been defined as a subset of the Java Platform for use in consumer electronic devices and mobile computing devices. Examples of Personal Java applications include handheld PCs, set-top Internet boxes, Web phones, and game controllers. Personal Java is designed to be upward-compatible with the Java Platform but still capable of executing in memory-constrained devices. Personal Java includes the JVM, PersonalAWT (a fine-tuned implementation of the Abstract Window Toolkit), and most of the applet, JavaBeans, I/O, networking, language, and utility packages. Optional APIs may be added to tailor Personal Java to specific application environments. Personal Java is capable of running on systems with 2MB of ROM and 1MB of RAM.

Embedded Java

Embedded Java is a subset of the Java Platform that is intended for high-volume embedded devices, such as mobile phones, pagers, printers, copiers, fax machines, medical instruments, and factory automation systems. Embedded Java is upward-compatible with Personal Java. It supports a text-only user interface and a selectable subset of the Java Platform. The Embedded Java API is still under development. Embedded Java is intended to be capable of running on systems with .5MB of ROM and .5MB of RAM.

Java Card

Java Card is a Java API for embedding Java in devices such as smart cards. It allows applications to be written once in Java and run on all smart cards for which the Java Card API has been ported. It also allows multiple applications to run on a single card. The Java Card API is currently in version 2.0. The Java Card API consists of parts of the `java.lang` package, plus the `javacard.framework`, `javacardx.crypto`, `javacardx.cryptEnc`, and `javacardx.framework` packages. These packages provide low-level control of smart card devices and provide security encryption and authentication services. Java Card is designed for hardware environments that have at least 16KB of ROM, 8KB of EEPROM, and 256 bytes of RAM.

Java Runtime Environment

The Java Runtime Environment, or JRE, is the implementation of the Java Platform and consists of the JVM, Core APIs, and supporting files. It is the JDK without the JDK tools, associated documentation, and source code. Although the JRE has been ported to many operating system platforms, Sun only distributes the JRE for Win32 (Windows NT, Windows 95, and Windows 98) and Solaris (Sparc and Intel versions). The JRE has been tailored to support languages other than English. It is also distributed with Java-enabled products, such as Netscape Communicator, Microsoft Internet Explorer, and Marimba Castanet Tuner.

Java Performance Enhancements

The Win32 Performance Pack is also distributed by Sun as an extension of the JRE. It includes a *Just-In-Time compiler (JIT)* that increases the speed of Java applications by a factor of 10. The JIT compiler compiles Java bytecodes into native x86 machine code so that they can execute directly on Intel x86 and Pentium-class processors.

Navigator Classes

Netscape has added its own extensions to the Java Platform for use within the Netscape Navigator. These extensions are referred to as *Internet Foundation Classes (IFC)*. The IFC is an API for developing Web applications that execute within the context of the Netscape browser. The API includes additional graphical user interface (GUI) controls besides those included with the AWT, a multifont text object for developing word processor-like applications, drag-and-drop support, animation support, and other capabilities. The IFC can be downloaded from Netscape at `http://developer.netscape.com/library/ifc/index.html`.

Netscape intended that the *Java Foundation Classes (JFC)* would eventually include and replace the IFC. However, this is not the case as of JDK 1.2. Eventually, the JFC will incorporate the IFC into the Java API and develop a common set of GUI interface controls implemented as JavaBeans.

Internet Explorer Classes

Not to be left out, Microsoft has developed its own Java API extensions, referred to as the *Application Foundation Classes (AFC)*. The AFC consists of a set of class libraries that implement GUI controls, multimedia capabilities, and support for Microsoft extract cabinet (CAB) files. CAB files are compressed archive files similar to ZIP files. The AFC consists of the following packages:

- `com.ms.ui`—Classes that provide user interface controls.
- `com.ms.fx`—Classes for multimedia effects.
- `com.ms.util.cab`—Classes for working with CAB files.

The AFC is written entirely in Java and is built on top of the AWT. It is intended to run on all Java platforms. The AFCs are available at `http://www.microsoft.com/java/afc/`.

Other Class Libraries

A number of vendors offer their class libraries as freeware, shareware, and commercial products. The Gamelan Web site has an excellent directory of these libraries at `http://www.developer.com/directories/pages/dir.java.html`.

ObjectSpace, Inc. has made their Generic Collection Library for Java (JGL) free for commercial use by Java Developers from its Web Site (`http://www.objectspace.com`).

The JGL contains 11 collection objects (such as lists) used to organize other objects, and 40 algorithms (such as sorting) used to manipulate groups of objects. The JGL is licensed and provided by most major Java tool vendors.

Native Methods

All of the Java API extensions that we've mentioned so far have been pure Java extensions. Although it is technically possible to extend Java using native methods, this approach is not recommended because you lose the platform-independent capability of Java.

There are some special circumstances in which native methods may be an appropriate solution for Java software development. For example, when interfacing Java to legacy systems, it may be necessary to create a Java wrapper around the API of the legacy system that you're attempting to salvage. In this case, you would use native methods to provide the interface with your legacy code. Other reasons to use native methods are based on the absence of appropriate features within the Java API. For example, you may need to access the native NetBEUI or IPX protocols of your operating system. Native methods can be used to wrap a Java interface on these protocols. Chapter 53, "Native Methods," shows how to extend the Java Platform using native methods.

Summary

In this chapter, you learned about the Java Platform and examined each of its parts, as well as the Personal Java, Embedded Java, and Java Card application environments. You looked at the most popular extensions to the Java Platform, including Netscape's Internet Foundation Classes, Microsoft's Application Foundation Classes, and ObjectSpace's JGL. In the next chapter, you'll learn about one of the most promising platforms for Java deployment—JavaOS.