Beginning J2ME: From Novice to Professional, Third Edition

SING LI AND JONATHAN KNUDSEN

Apress®

Beginning J2ME: From Novice to Professional, Third Edition

ISBN (pbk): 1-59059-479-7

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Steve Anglin
Technical Reviewer: Chris Harris
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Assistant Publisher: Grace Wong
Project Manager: Laura Cheu
Copy Manager: Nicole LeClerc
Copy Editor: Ami Knox
Production Manager: Kari Brooks-Copony
Production Editor: Laura Cheu
Compositor: Susan Glinert Stevens
Proofreader: Liz Welch
Indexer: Carol Burbo
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013, and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, e-mail orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section.

To Kimlan

To Andrew and Elena

Contents at a Glance

Contents

About the Authors

SING LI is a systems consultant, avid open source developer, and active freelance writer. With over two decades of industry experience, Sing is a regular contributor to printed magazines and e-zines. His book credits include *Beginning JavaServer Pages*; *Professional Apache Tomcat 5*; *Pro JSP, Third Edition*; *Early Adopter JXTA*; *Professional Jini*; and numerous others. He is an active evangelist of the mobile Java, VON, and P2P evolutions.

JONATHAN KNUDSEN is the author of several other Java books, including *Learning Java*, *Java 2D Graphics*, and *Java Cryptography*. He is also the author of *The Unofficial Guide to LEGO® MINDSTORMS™ Robots*, but, sadly, was unable to parlay that success into a full-time career. Jonathan has written numerous articles about Java and a few about LEGO robots as well. He is the father of four children and enjoys bicycling and playing the piano. For more information, see `http://jonathanknudsen.com/`.

About the Technical Reviewer

CHRIS HARRIS is from Dublin, Ireland, and received his BS in mathematics and computer science from the Dublin Institute of Technology. He has worked in the wireless software industry for over five years, and has been involved in the Java Community Process as both Specification Lead and Expert Group member. He currently works in Bordeaux, France, for a mobile games company called IN-FUSIO.

Acknowledgments

Thanks to everyone at Apress for putting this book together on such a tight schedule. Thanks to Gary Cornell for the initial vision for such a title. To Steve Anglin, for putting the two of us together on this fascinating project. To Laura Cheu, our "sleepless in New York" project manager, without whom this book would have never wrapped in time. To Ami Knox, our tireless copy editor, for transforming the techno-babble we churn out into understandable material. Last but not least, a hearty thanks to Chris Harris, for keeping us honest with his excellent technical review.

Preface

This book describes how to program mobile telephones, pagers, PDAs, and other small devices using Java technology. It is about the Mobile Information Device Profile (MIDP), which is part of the Java 2 Platform, Micro Edition (J2ME). It is concise and complete, describing all of MIDP as well as moving into several exciting advanced concepts such as 3D graphics and cryptography.

This third edition covers MIDP 2.0, and has been updated to track the Java Technology for the Wireless Industry (JTWI 1.0) de facto standard. Every chapter has been revised and meticulously updated, and four completely new chapters have been added.

Who Are You?

You're probably reading this book because you're excited about building wireless applications with Java. This book is aimed at people who already have experience programming in Java. At a minimum, you should understand the Java programming language and the fundamentals of object-oriented programming. Some chapters delve into subjects that in themselves could occupy entire books. These chapters include suggested reading if you want to get up to speed on a particular subject.

If you are unfamiliar with Java, we suggest you read an introductory book or take a course. *Learning Java, Second Edition* (O'Reilly 2002) is a good introduction to Java for programmers who are already experienced in another language such as C or C++.

The Structure of This Book

This book is organized into 18 chapters and one appendix. There are basically three sections. The first two chapters are introductory material. Chapters 3 through 16 provide complete coverage of the MIDP 2.0 and JTWI 1.0 APIs, together with some of the most frequently used optional APIs available. Chapters 17 and 18 cover advanced topics. The complete breakdown of chapters is listed here:

- Chapter 1, "Introduction," provides context and motivation for the rest of the book. J2ME is explained in detail, gradually zooming in to MIDP and JTWI.
- Chapter 2, "Building MIDlets," is intended to be a teaser. It includes an example application that allows you to look up the definitions of words over the Internet using any MIDP device. Along the way you'll learn a lot about developing applications for the MIDP platform.
- Chapter 3, "All About MIDlets," goes into detail about the life cycle and packaging of MIDP applications. It includes coverage of the MIDP 2.0 security architecture.
- Chapter 4, "Almost the Same Old Stuff," describes the pieces of the MIDP API that will be familiar to Java programmers.

- Chapter 5, "Creating a User Interface," is the first of a handful of chapters devoted to MIDP's user interface packages. It provides an overview of MIDP's user interface package and goes into detail about the simple visual components.
- Chapter 6, "Lists and Forms," picks up where Chapter 5 left off, describing MIDP's advanced user interface components.
- Chapter 7, "Custom Items," shows how to create your own form items in MIDP.
- Chapter 8, "Persistent Storage I: MIDP Record Store," describes MIDP's mechanism for storing data.
- Chapter 9, "Persistent Storage II: File Connection and PIM API," covers popular optional APIs for accessing a device's file system, memory cards, and PIM features.
- Chapter 10, "Connecting to the World," contains all the juicy details about how MIDP applications can send and receive data over the Internet.
- Chapter 11, "Wireless Messaging API," describes WMA, a standard component of JTWI 1.0 that can be used to access the rich Short Message Service (SMS) and Cell Broadcast Service (CBS) available on modern wireless networks. This chapter also covers the new WMA 2.0 for working with audio and video messages via Multimedia Messaging Service (MMS).
- Chapter 12, "Bluetooth and OBEX," provides coverage of the optional API that enables communications of devices through Bluetooth radio Personal Area Networks (PANs) and infrared links.
- Chapter 13, "Programming a Custom User Interface," describes the low-level API that can be used for specialized application user interfaces.
- Chapter 14, "The Game API," describes MIDP 2.0 features for creating games, including sprites and tiled layers.
- Chapter 15, "3D Graphics," includes a hands-on, easy-to-understand introduction to the Mobile 3D Graphics optional API (M3G), providing you with a springboard into the fascinating world of 3D graphics programming on mobile devices.
- Chapter 16, "Sound, Music, and Video: MMAPI," is about MIDP 2.0 new multimedia capabilities and the Mobile Media API (MMAPI). You'll learn how to produce simple tones, play sampled audio data, play MP3 music, play video clips, and even take snapshots with your camera-phone.
- Chapter 17, "Performance Tuning," describes techniques for coping with the limited resources that are available on small devices.
- Chapter 18, "Protecting Network Data," discusses how to protect valuable data on the insecure Internet. It includes two sample applications that demonstrate cryptographic techniques for protecting data.
- Finally, the appendix, "MIDP API Reference," contains an API reference for the classes and interfaces that make up MIDP. The method signatures for the public API of each class and interface are listed for handy quick reference.

CHAPTER 1

Introduction

Java 2 Platform, Micro Edition (J2ME) is the second revolution in Java's short history. When Java was introduced in 1995, it looked like the future of computing was in *applets*, small programs that could be downloaded and run on demand. A slow Internet and a restrictive all-or-nothing sandbox security model accounted for the initially slow adoption of applets. Java, as a platform, really took off with the advent of *servlets*, Java programs that run on a server (offering a modular and efficient replacement for the vulnerable CGI). Java further expanded into the server side of things, eventually picking up the moniker of Java 2 Platform, Enterprise Edition (J2EE). This was the first revolution, the blitz of server-side Java.

The second revolution is the explosion of small-device Java, and it's happening now. The market for small devices is expanding rapidly, and Java is important for two reasons. First, developers can write code and have it run on dozens of small devices, without change. Second, Java has important safety features for downloadable code.

Understanding J2ME

J2ME isn't a specific piece of software or specification. All it means is Java for small devices. Small devices range in size from pagers, mobile phones, and personal digital assistants (PDAs) all the way up to things like set-top boxes that are just shy of being desktop PCs.

J2ME is divided into *configurations*, *profiles*, and *optional APIs*, which provide specific information about APIs and different families of devices. A configuration is designed for a specific kind of device based on memory constraints and processor power. It specifies a Java Virtual Machine (JVM) that can be easily ported to devices supporting the configuration. It also specifies a strict subset of the Java 2 Platform, Standard Edition (J2SE) APIs that will be used on the platform, as well as additional APIs that may be necessary. Device manufacturers are responsible for porting a specific configuration to their devices.

Profiles are more specific than configurations. A profile is based on a configuration and provides additional APIs, such as user interface, persistent storage, and whatever else is necessary to develop running applications for the device.

Optional APIs define specific additional functionality that may be included in a particular configuration (or profile). The whole caboodle—configuration, profile, and optional APIs—that is implemented on a device is called a *stack*. For example, a possible future device stack might be CLDC/MIDP + Mobile Media API. See the section "Platform Standardization" later in this chapter for information on JSR 185, which defines a standard J2ME stack.

Currently, there are a handful of configurations and profiles; the most relevant ones for J2ME developers are illustrated in Figure 1-1.

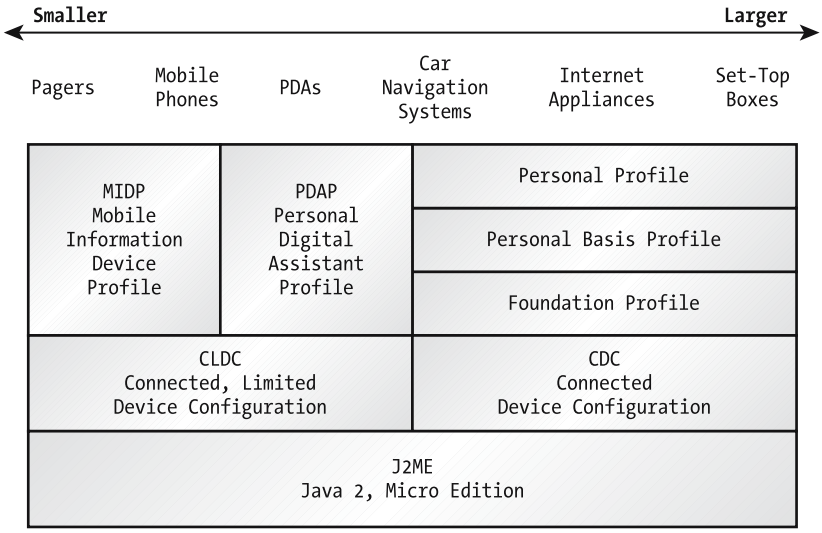

Figure 1-1. *Common J2ME profiles and configurations*

THE JAVA COMMUNITY PROCESS

The Java Community Process (JCP) is designed to ensure that Java technology is developed according to community consensus, and to avoid industry fragmentation. The process is described here:

`http://jcp.org/jsr/all/`

Configurations and profiles first appear in the world as Java Specification Requests (JSRs). You can see a list of current JSRs here:

`http://jcp.org/jsr/all/`

To give you a flavor of what's happening in the J2ME world, Table 1-1 shows some of the configurations, profiles, and optional APIs that are available and under development. This is not a comprehensive list; for more information, check out the JCP web site at `http://jcp.org/`.

Table 1-1. *J2ME Configurations, Profiles, and Optional APIs*

Configurations		
JSR	**Name**	**URL**
30	Connected, Limited Device Configuration (CLDC) 1.0	`http://jcp.org/jsr/detail/30.jsp`
139	Connected, Limited Device Configuration (CLDC) 1.1	`http://jcp.org/jsr/detail/139.jsp`
36	Connected Device Configuration 1.0.1	`http://jcp.org/jsr/detail/36.jsp`
218	Connected Device Configuration 1.1	`http://jcp.org/jsr/detail/218.jsp`

Table 1-1. *J2ME Configurations, Profiles, and Optional APIs (Continued)*

Configurations		
JSR	**Name**	**URL**
Profiles		
JSR	**Name**	**URL**
37	Mobile Information Device Profile 1.0	http://jcp.org/jsr/detail/37.jsp
118	Mobile Information Device Profile 2.0	http://jcp.org/jsr/detail/118.jsp
75	PDA Profile 1.0	http://jcp.org/jsr/detail/75.jsp
46	Foundation Profile 1.0	http://jcp.org/jsr/detail/46.jsp
129	Personal Basis Profile 1.0	http://jcp.org/jsr/detail/129.jsp
62	Personal Profile 1.0	http://jcp.org/jsr/detail/62.jsp
219	Foundation Profile 1.1	http://jcp.org/jsr/detail/219.jsp
217	Personal Basis Profile 1.1	http://jcp.org/jsr/detail/217.jsp
Optional APIs		
JSR	**Name**	**URL**
75	PDA Optional Packages for J2ME	http://jcp.org/jsr/detail/75.jsp
82	Java APIs for Bluetooth	http://jcp.org/jsr/detail/82.jsp
135	Mobile Media API 1.1	http://jcp.org/jsr/detail/135.jsp
184	Mobile 3D Graphics	http://jsp.org/jsr/detail/184.jsp
179	Location API for J2ME	http://jcp.org/jsr/detail/179.jsp
120	Wireless Messaging API 1.0	http://jcp.org/jsr/detail/120.jsp
205	Wireless Messaging API 2.0	http://jcp.org/jsr/detail/205.jsp
172	J2ME Web Services APIs	http://jsp.org/jsr/detail/172.jsp
66	RMI Optional Package	http://jcp.org/jsr/detail/66.jsp

Configurations

A configuration specifies a JVM and some set of core APIs for a specific family of devices. Currently there are two: the Connected Device Configuration (CDC) and the Connected, Limited Device Configuration (CLDC).

The configurations and profiles of J2ME are generally described in terms of their memory capacity. Usually a minimum amount of ROM and RAM is specified. For small devices, it makes sense to think in terms of volatile and nonvolatile memory. The nonvolatile memory is capable of keeping its contents intact as the device is turned on and off. ROM is one type, but nonvolatile memory could also be flash memory or battery-backed RAM. Volatile memory is essentially workspace and does not maintain its contents when the device is turned off.

Connected Device Configuration

A connected device has, at a minimum, 512KB of read-only memory (ROM), 256KB of random access memory (RAM), and some kind of network connection. The CDC is designed for devices like television set-top boxes, car navigation systems, and high-end PDAs. The CDC specifies that a full JVM (as defined in the Java Virtual Machine Specification, 2nd edition) must be supported.

CDC is developed under the Java Community Process. For more information on the CDC, see `http://java.sun.com/products/cdc/`. A Linux reference of CDC 1.0.1 implementation is available now.

CDC 1.0.1 is the basis of the Personal Profile 1.0 stack. The Personal Profile 1.0 increases the minimum memory requirement to 2.5MB of ROM and 1MB of RAM, and requires a robust network plus a GUI display on a device that can support applet display.

CDC 1.1 is currently a work in progress. It will support Personal Profile 1.1 and will introduce APIs to match the level of JDK 1.4.

Connected, Limited Device Configuration

CLDC is the configuration that interests us, because it encompasses mobile phones, pagers, PDAs, and other devices of similar size. CLDC is aimed at smaller devices than those targeted by the CDC. The name CLDC appropriately describes these devices, having limited display, limited memory, limited CPU power, limited display size, limited input, limited battery life, and limited network connection.

The CLDC is designed for devices with 160KB to 512KB of total memory, including a minimum of 160KB of ROM and 32KB of RAM available for the Java platform. If you've ever watched J2SE gobble up tens of megabytes of memory on your desktop computer, you'll appreciate the challenge of J2ME. The "Connected" simply refers to a network connection that tends to be intermittent and probably not very fast. (Most mobile telephones, for example, typically achieve data rates of 9.6Kbps.) These connections also tend to be costly, typically billed by the data packets exchanged. Between the high cost and intermittent slow network connection, applications designed in the CLDC space should be very sparing with the use of the network connection.

The reference implementation of the CLDC is based around a small JVM called the KVM (J2ME licensees may use this KVM or implement their own as long as it conforms to the CLDC). Its name comes from the fact that it is a JVM whose size is measured in kilobytes rather than megabytes. While the CLDC is a specifications document, the KVM refers to a specific piece of software.[1] Because of its small size, the KVM can't do everything a JVM does in the J2SE world.

- Native methods cannot be added at runtime. All native functionality is built into the KVM.
- The KVM only includes a subset of the standard bytecode verifier. This means that the task of verifying classes is split between the CLDC device and some external mechanism. This has serious security implications, as we'll discuss later.

1. The KVM was originally part of the Spotless system, a Sun research project. See `http://www.sun.com/research/spotless/`.

You can find more information at the CLDC home page, `http://java.sun.com/products/cldc/`. Most deployed devices implement CLDC 1.0, but CLDC 1.1 devices are making their way onto the market as this is written. CLDC 1.1 includes enhancements to CLDC 1.0, including support for floating-point data types.

Profiles

A profile is layered on top of a configuration, adding the APIs and specifications necessary to develop applications for a specific family of devices.

Current Profiles

Several different profiles are being developed under the Java Community Process. Table 1-1 (shown earlier) provides a bird's-eye view.

The Foundation Profile is a specification for devices that can support a rich networked J2ME environment. It does not support a user interface; other profiles can be layered on top of the Foundation Profile to add user interface support and other functionality.

Layered on top of the Foundation Profile are the Personal Basis Profile and the Personal Profile. The combination of CDC + Foundation Profile + Personal Basis Profile + Personal Profile is designed as the next generation of the PersonalJava application runtime environment (see `http://java.sun.com/products/personaljava/`). As such, the Personal Profile has the specific goal of backward compatibility with previous versions of PersonalJava.

The PDA Profile (PDAP), which is built on CLDC, is designed for palmtop devices with a minimum of 512KB combined ROM and RAM (and a maximum of 16MB). It sits midway between the Mobile Information Device Profile (MIDP) and the Personal Profile. It includes an application model based on MIDlets but uses a subset of the J2SE Abstract Windowing Toolkit (AWT) for graphic user interface. Although the PDAP specification is nearly finished, to our knowledge no hardware manufacturer has announced that it will be implementing PDAP. The J2ME world currently is covered by MIDP on the small end and Personal Profile on the higher end.

Mobile Information Device Profile

The focus of this book is the Mobile Information Device Profile (MIDP). According to the MIDP 2.0 specification (JSR-118), a Mobile Information Device has the following characteristics:

- A minimum of 256KB of ROM for the MIDP implementation (this is in addition to the requirements of the CLDC)
- A minimum of 128KB of RAM for the Java runtime heap
- A minimum of 8KB of nonvolatile writable memory for persistent data
- A screen of at least 96×54 pixels
- Some capacity for input, either by keypad, keyboard, or touch screen
- Two-way network connection, possibly intermittent

Try to imagine a device that might be a MIDP device: mobile telephones and advanced pagers are right in the groove, but entry-level PDAs could also fit this description.

More information about MIDP, including a link to the official specification document, is at `http://java.sun.com/products/midp/`. There are two versions of MIDP: MIDP 1.0 (JSR 37), and MIDP 2.0 (JSR 118). Many of the currently available devices do and all new devices will support MIDP 2.0. Compared to MIDP 1.0, MIDP 2.0 features a number of enhancements, including support for multimedia, a new game user interface API, support for HTTPS connection, and other features. Most importantly, MIDP 2.0 is fully backward compatible with MIDP 1.0.

JTWI standard compliance requires devices to implement MIDP 2.0 (see the next section on platform standardization). This book's focus will be on MIDP 2.0. We will mention MIDP 1.0 differences only in this introductory chapter as background information.

Platform Standardization

Given the profusion of configurations, profiles, and especially optional APIs, how do you know what APIs are likely to be available on typical devices? Sun's answer to this question is JSR 185 (`http://jcp.org/jsr/detail/185.jsp`), impressively titled *Java Technology for the Wireless Industry (JTWI)*. This specification attempts to standardize software stacks to bring coherence to the J2ME world. A reference implementation and a TCK (kit for compatibility testing) of the unified software stack is made available with JSR 185. As currently specified, a JTWI-compliant device must have MIDP 2.0 with CLDC 1.0 (or CLDC 1.1), and must support WMA (Wireless Messaging API 1.0—JSR 120). If a JTWI device exposes video or audio API to applications, they must also support Mobile Media API (MMAPI).

In the next generation of J2ME, a concept called Building Blocks is supposed to replace configurations and profiles. A *Building Block* is just some subset of a J2SE API. For example, one Building Block might be created from a subset of the J2SE `java.io` package. Conceptually, a Building Block represents a smaller chunk of information than a configuration. Profiles, then, will be built on top of a set of Building Blocks rather than a configuration.

The definition of Building Blocks is a JSR, which is briefly described here: `http://jcp.org/jsr/detail/68.jsp`. Progress on JSR 68 has been extremely slow since its creation in June 2000.

In the meantime, JSR 185 will better serve as a standardization platform. Recently, leveraging the success of the JTWI work, Nokia and Vodafone have submitted a new JSR, JSR-248: Mobile Service Architecture for CDC (`http://jcp.org/jsr/detail/248.jsp`), to define a new standard software stack for the next generation of mobile devices.

Anatomy of MIDP Applications

The APIs available to a MIDP application come from packages in both CLDC and MIDP, as shown in Figure 1-2. Packages marked with a + are new in CLDC 1.1 and MIDP 2.0.

CLDC defines a core of APIs, mostly taken from the J2SE world. These include fundamental language classes in `java.lang`, stream classes from `java.io`, and simple collections from `java.util`. CLDC also specifies a generalized network API in `javax.microedition.io`.

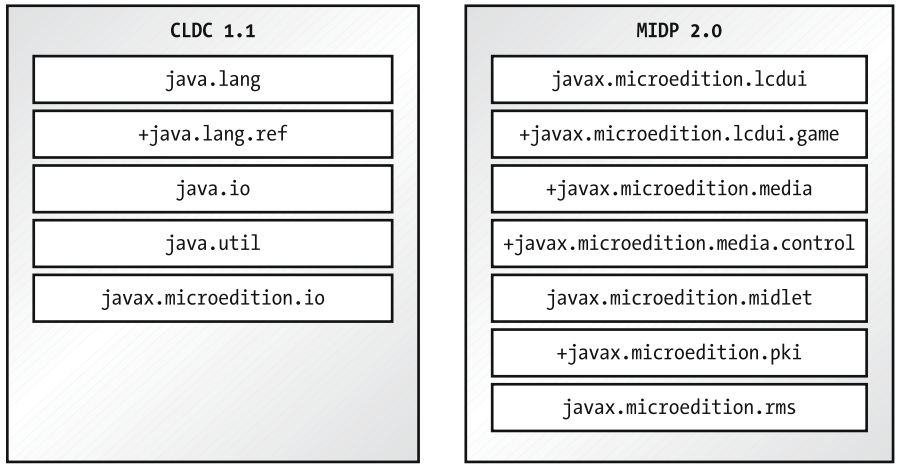

Figure 1-2. *MIDP packages*

Note While the MIDP 2.0 specification suggests that MIDP 2.0 will most likely be paired with CLDC 1.1, the JTWI compatibility platform only requires implementation atop CLDC 1.0. As a result, most current implementations of MIDP 2.0 are paired with CLDC 1.0. Historically, the MIDP 2.0 specification was moving faster through the Java Community Process than the CLDC 1.1 specification.

Optionally, device vendors may also supply Java APIs to access device-specific features. MIDP devices, then, will typically be able to run several different flavors of applications. Figure 1-3 shows a map of the possibilities.

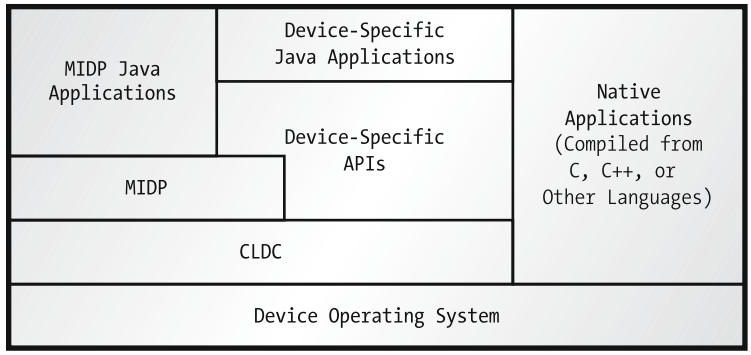

Figure 1-3. *MIDP software components*

Each device implements some kind of operating system (OS). Native applications run directly on this layer and represent the world before MIDP—many different kinds of devices, each with its own OS and native applications.

Layered on top of the device OS is the CLDC (including the JVM) and the MIDP APIs. MIDP applications use only the CLDC and MIDP APIs. Device-specific Java applications may also use Java APIs supplied by the device vendor.

Advantages of MIDP

Given the spectrum of configurations and profiles, why is this book about MIDP? First, MIDP comes at a critical time, a time when MIDP devices, like mobile phones, are an exploding market. Simultaneously, MIDP devices are achieving the kind of processing power, memory availability, and Internet connectivity that makes them an attractive platform for mobile networked applications. MIDP is already deployed on millions of handsets all over the world.

Second, of course, MIDP is the first J2ME profile that is ready for prime time. You will start writing applications as soon as you head into the next chapter!

Portability

The advantage of using Java over using other tools for small device application development is portability. You could write device applications with C or C++, but the result would be specific to a single platform. An application written using the MIDP APIs will be directly portable to any MIDP device.

If you've been following Java's development for any time, this should sound familiar. It's the same "Write Once, Run Anywhere" (WORA) mantra that Sun's been repeating since 1995. Unfortunately, WORA is a bit of a four-letter word for developers who struggled with cross-platform issues in JDK 1.0 and JDK 1.1 (particularly the browser implementations). While Java's cross-platform capabilities in Java 2 are generally successful, WORA still has the taint of an unfulfilled promise.

Does MIDP deliver painless cross-platform functionality? Yes. There will always be platform-specific bugs in MIDP implementations, but we believe MIDP works as advertised because it is so much smaller than desktop Java. Less code means fewer bugs when porting to multiple platforms. Most of the cross-platform incompatibilities of JDK 1.0 and JDK 1.1 were caused by the nightmare of trying to fit disparate windowing systems into the AWT's peer-based component architecture. MIDP has nothing approaching the complexity of AWT, which means there's an excellent possibility that MIDP applications will seamlessly run on multiple platforms right out of the starting gate. Furthermore, while the JDK 1.0 test suite only included a few dozen tests, the MIDP compatibility test suite includes several thousand tests.

Security

A second compelling reason for using Java for small device development is security. Java is well known for its ability to safely run downloaded code like applets. This is a perfect fit—it's easy to imagine nifty applications dynamically downloading to your mobile phone.

But it's not quite such a rosy picture. For one thing, the JVM used in the CLDC only implements a partial bytecode verifier, which means that part of the important task of bytecode verification must be performed off the MIDP device.

Second, the CLDC does not allow for application-defined classloaders. This means that most dynamic application delivery is dependent on device-specific mechanisms.

MIDP applications do offer one important security promise: they can never escape from the confines of the JVM. This means that, barring bugs, a MIDP application will never be able to write to device memory that doesn't belong to the JVM. A MIDP application will never mess up another application on the same device or the device OS itself.[2] This is the killer feature of MIDP. It allows manufacturers and carriers to open up application development to the world, more or less free from certification and verification programs, without the fear that rogue coders will write applications that crash phones.

In MIDP 2.0, MIDlet suites can be cryptographically signed, and then verified on the device, which gives users some security about executing downloaded code. A new permissions architecture also allows the user to deny untrusted code access to certain API features. For example, if you install a suspicious-looking MIDlet suite on your phone, it will only be able to make network connections if you explicitly allow it to do so.

MIDP Vendors

Several large players have thrown their weight behind MIDP. A quick browse of the JSR page for MIDP exposes the most important companies.

Two Asian companies led the charge to provide network services for Java-enabled mobile phones. In Korea, LG TeleCom deployed a service called ez-i in mid-2000. Later that same year, NTT DoCoMo deployed their wildly popular i-mode. The APIs developed for LG TeleCom (KittyHawk) and NTT DoCoMo (i-Appli) are similar to MIDP but were completed before the MIDP 1.0 specification.

In the United States, Motorola was the first manufacturer to produce a MIDP telephone. The i50sx and i85s were released on April 2, 2001, with service provided by Nextel. Motorola has since expanded its offerings with a handful of new devices.

Nokia has also made serious commitments to MIDP, and the expert group that created the MIDP specification includes an impressive list of manufacturers—Ericsson, Hitachi, Nokia, Sony, Symbian, and many more. You can go read the industry predictions if you wish—a gazillion MIDP phones sold in the next three years, and so on. It's a safe bet that your MIDP application will have a large market. For a comprehensive listing of MIDP devices, visit `http://wireless.java.sun.com/device/`.

Fragmentation

Platform fragmentation is a serious concern in the MIDP community. Many devices that implement MIDP 1.0 also include device-specific APIs. These APIs access device-specific features or provide functionality that wasn't addressed in MIDP 1.0's least-common-denominator specification. Current software vendors, particularly game developers, sometimes create and distribute multiple versions of an application, each tailored to a specific platform. Obviously this is a concern: part of the point of using MIDP in the first place is the ability to write one set of code and deploy it on multiple platforms.

2. A MIDP application could conceivably launch a denial-of-service attack (that is, sucking up all the processor's time or bringing the device OS to a standstill). It's widely acknowledged that there's not much defense against denial-of-service attacks. Applications and applets in J2SE suffer from the same vulnerability.

MIDP 2.0 addresses a long list of the shortcomings inherent with MIDP 1.0. Its timing is good, so the current adoption and deployment of MIDP 2.0 devices should provide a standard, unified platform for wireless development.

Another fragmentation issue is the confusion surrounding the assembly of configurations, profiles, and optional APIs into a software stack. As a developer, you want to understand exactly what set of APIs will be available or are likely to be available, but there seem to be so many choices and so many possibilities. The standardization on a software stack, via JTWI (JSR 185—`http://jcp.org/jsr/detail/185.jsp`), should bring clarity to this issue.

Summary

J2ME is the Java platform for small devices, a broad field that covers pretty much everything smaller than a breadbox. Because J2ME spans such a diverse selection of hardware, it is divided into configurations, profiles, and optional APIs. A configuration specifies a subset of J2SE functionality and the behavior of the JVM, while profiles are generally more specific to a family of devices with similar characteristics. Optional APIs offer added functionality in a flexible package. The Mobile Information Device Profile, which is the focus of this book, includes APIs for devices like mobile phones and two-way pagers.

CHAPTER 2

Building MIDlets

MIDP applications are piquantly called MIDlets, a continuation of the naming theme begun by applets and servlets. Writing MIDlets is relatively easy for a moderately experienced Java programmer. After all, the programming language is still Java. Furthermore, many of the fundamental APIs from `java.lang` and `java.io` are basically the same in the MIDP as they are in J2SE. Learning the new APIs (in the `javax.microedition` hierarchy) is not terribly difficult, as you'll see in the remainder of this book.

The actual development process, however, is a little more complicated for MIDlets than it is for J2SE applications. Beyond a basic compile-and-run cycle, MIDlets require some additional tweaking and packaging. The complete build cycle looks like this: Edit Source Code ➤ Compile ➤ Preverify ➤ Package ➤ Test or Deploy.

To show how things work, and to give you a taste of MIDlet development, this chapter is dedicated to building and running a simple MIDlet. In later chapters, we'll delve into the details of the MIDP APIs. In this chapter, you'll get a feel for the big picture of MIDlet development.

Tooling Up

MIDlets are developed on regular desktop computers, although the MIDlet itself is designed to run on a small device. To develop MIDlets, you'll need some kind of development kit, either from Sun or another vendor. Remember, MIDP is only a specification; vendors are free to develop their own implementations.

The world is full of MIDlet development tools if you know where to look. Furthermore, many of these tools are freely available.

The bare bones set of tools is Sun's MIDP reference implementation. This includes the `preverify` tool (more on this later), a MIDP device emulator, source code, and documentation. You can download the MIDP reference implementation by following the links from `http://java.sun.com/products/midp/`. However, we don't recommend using the reference implementation unless you really enjoy being in the middle of the gritty details of building and packaging MIDlets. (You should also investigate the reference implementation if you are interested in porting the MIDP runtime to a new device or platform.)

A much better tool for beginners is Sun's J2ME Wireless Toolkit, available from `http://java.sun.com/products/j2mewtoolkit/`. The J2ME Wireless Toolkit (or J2MEWTK, as it's affectionately known) includes a GUI tool that automates some of the tedious details of building and packaging MIDlets, providing a simple path from source code to running MIDlets. At the same time, the J2ME Wireless Toolkit is a relatively lightweight solution, almost a miniature IDE, not something that will choke your machine.