Introduction

The Java 2 Micro Edition (J2ME) is the version of the Java 2 platform that's designed for use with smaller, less-powerful devices such as mobile phones, Personal Digital Assistants (PDAs), TV set-top boxes (for Web browsing and e-mail without a whole computer), and embedded devices. Since these devices vary quite a bit in their capabilities, the J2ME platform has two different *configurations*, each with its own choice of *profiles*. The Connected Limited Device Configuration (CLDC) is the configuration you'll be working with in this book. It's designed for mobile phones and low-level PDAs. More precisely, CLDC is intended for devices with a 16-bit or 32-bit processor, at least 160 kilobytes (KB) of nonvolatile memory, at least 32KB of volatile memory, and some network connectivity, possibly wireless and intermittent. CLDC's unique profile is the Mobile Information Device Profile (MIDP). This book is specifically concerned with the 2.0 version of MIDP (although I'll explain what changes and restrictions you need to make to write a program that's compatible with the 1.0 version of MIDP). The other configuration associated with J2ME is the Connected Device Configuration (CDC), which isn't covered in this book.

The configuration specifies the type of Java Virtual Machine (JVM) that's used and what will be in the minimal class libraries (the `java.*` packages and the `javax.microedition.io` package in the case of CLDC). CDC specifies a complete JVM, but the JVM of CLDC has some limitations compared to the standard JVM. A profile is added on top of the configuration to define a standard set of libraries (the other `javax.microedition.*` packages in this case). MIDP contains packages for a user interface, media control, input/output, data storage, and security.

How the CLDC Differs from the Rest of the Java Universe

The designers of the CLDC specification have made an effort to make CLDC resemble the standard platform as closely as possible, and they've done a pretty good job of it. Nothing critical to small applications appears to be missing. I'll give a general outline of the changes here, and I'll refer you to later chapters in this book for a more in-depth discussion of the aspects that have changed the most dramatically.

Differences in the JVM

The JVM specified in CLDC is mostly the same as the standard JVM. Unsurprisingly, a few of the costlier noncritical features have been eliminated. One example is the

method `Object.finalize()`. According to the Javadoc, the `Object.finalize()` method is called on an object when the JVM determines that it's time to garbage collect that object. The actions the object can take in its `finalize()` method aren't restricted, so in particular it can make itself available again to currently active threads! The garbage collection algorithm is already expensive, and this method clearly undermines its efficiency by obligating the JVM to recheck objects that had already been marked as garbage. It's no wonder this method was eliminated in CLDC since it's not hard at all to keep track of the objects that you're still using without requiring the JVM to check with you before throwing anything away.

Some of the other areas where the JVM's set of features have been reduced are in security, threads, and exceptions/errors. See the "Understanding the Differences Between MIDP Security and Security in Other Java Versions" section in Chapter 7 for a discussion of the differences in the security model. See the "Differences Between CLDC Threads and Threads in Standard Java" section in Chapter 4 for information about threads. The changes in the error-handling system are that CLDC doesn't allow asynchronous exceptions and that the set of error classes has been greatly reduced. Instead of 22 possible errors, you now have only `OutOfMemoryError`, `VirtualMachineError`, and `Error`. On the other hand, almost all the exceptions in the `java.lang.*` package have been retained.

You may not notice a few changes to the JVM just by looking at the Application Programming Interface (API). In CLDC the JVM is allowed to perform some optimizations (such as prelinking classes) that were disallowed to the standard JVM. Such changes shouldn't concern the application programmer in general. The one exception is that an additional preverification stage has been added after compilation. The preverification process adds extra information to the classfile to make the bytecode verification algorithm easier at run time when the device checks that your classfile is valid before using it. You easily accomplish the preverification step with standard tools (see the "Using KToolbar" section in Chapter 1 and the "Compiling and Running from the Command Line" section in Chapter 1). Preverification isn't technically required 100 percent of the time, but it aids in compatibility, and there's no reason not to do it.

One more general item to be aware of is that although a CLDC-compliant platform is required to support Unicode characters, it's required to support only the International Organization for Standardization (ISO) Latin 1 range of characters from the Unicode standard, version 3.0. For more information about character encoding in Java, see `http://java.sun.com/products/jdk/1.3/docs/guide/intl/encoding.doc.html`.

Differences in the Libraries

As you may guess, the standard libraries have been drastically reduced. It's unfortunate in many cases, but doing without some helpful tools is one of your challenges as a J2ME developer, just as you need to place more of a priority on writing tight, efficient code than a Java 2 Standard Edition (J2SE) or Java 2

Enterprise Edition (J2EE) developer would. The only `java.*` packages that you have available to you are `java.lang.*`, `java.util.*`, and `java.io.*`. That means you have to do without `java.lang.reflect.*`, `java.math.*`, `java.security.*`, and many others. Many of the missing packages have been replaced by MIDP packages that are more appropriate for small devices, as you'll see throughout this book.

Although the three remaining `java.*` packages have been greatly reduced, it's clear that the designers of the CLDC have tried to keep as much as possible and create familiar replacements for classes and methods that had to be removed. The `java.lang.*` package has been pared down to just the classes that correspond to data types (`Integer`, and so on) and a few necessary items: `Math`, `Object`, `Runnable`, `String`, `StringBuffer`, `System`, `Thread`, and `Throwable` (plus the exceptions and errors discussed previously). The `java.util.*` and `java.io.*` packages have been similarly reduced to their essentials. For examples of how to use the MIDP versions of the `java.io.*` classes, see the "Serializing More Complex Data Using Streams" section in Chapter 5. For a discussion of the changes to the `java.util.*` package, see the "Using the java.util Package" section in Chapter 2.

What's New in MIDP 2.0

It's possible to write some fun basic games using the earlier (1.0) version of MIDP. I wrote the example game in Chapter 2 to be compatible with MIDP 1.0 so you can see what's available there. But MIDP 2.0 is loaded with additional features that allow you to create a much richer gaming environment. The most obvious new tool you get in MIDP 2.0 is the package `javax.microedition.lcdui.game`, which is a special package to help you optimize game graphics and controls (see Chapter 3). Also, now you can add music to your game (see the "Adding Music" section in Chapter 4), which was impossible in MIDP 1.0. Plus, a MIDP 2.0 device is more likely than a MIDP 1.0 device to support socket connections in addition to the standard Hypertext Transfer Protocol (HTTP)/secure HTTP (HTTPS) connections (although socket connections are optional in both MIDP 1.0 and MIDP 2.0). Sockets can simplify programming multiplayer network games (see the "Using Plain Sockets" section in Chapter 6). Additionally, a security system has been added to allow the device to regulate which applications can have access to protected resources such as sockets (see Chapter 7).

So many new, powerful features for games exist in the new MIDP 2.0 API that there's a good chance you won't even want to bother making special versions of your games for MIDP 1.0 devices. As I'm writing this (April 2004), almost all the Java-enabled small devices on the market are capable of running only MIDP 1.0. But MIDP 2.0 devices are already appearing on the market, and small devices have fast turnover. The cost of a Java-enabled cell phone is generally reduced when the customer purchases it along with a yearlong phone-service contract. Consequently, a typical user will reason that since the phone wasn't too expensive to begin with, he might as well buy a new one the next year. It probably won't be long before the MIDP 1.0 devices are the ones that are rare.