

THE BASICS OF J2ME

**FOR PUBLIC
RELEASE**

Topics in this Chapter

- Java Editions
- Why J2ME?
- Configurations
- Profiles
- Java Virtual Machines
- Big Picture View of the Architecture
- Compatibility between Java Editions
- Putting all the Pieces Together

Chapter

1

It all started with one version of Java—now known as Java 2 Standard Edition (J2SE)—and the tagline “Write Once, Run Anywhere™.” The idea was to develop a language in which you would write your code once, and then it would run on any platform supporting a Java Virtual Machine.

Since its launch in 1995, the landscape has changed significantly. Java has extended its reach far beyond desktop machines. Two years after the introduction of Java, a new edition was released, Java 2 Enterprise Edition, providing support for large-scale, enterprise-wide applications. The most recent addition to the family is the Micro Edition, targeting “information appliances,” ranging from Internet-enabled TV set-top boxes to cellular phones.

Java Editions

Let's begin with a quick summary of the Java platforms currently available:

- **Standard Edition (J2SE):** Designed to run on desktop and workstations computers.

Chapter 1 The Basics of J2ME

- **Enterprise Edition (J2EE):** With built-in support for Servlets, JSP, and XML, this edition is aimed at server-based applications.
- **Micro Edition (J2ME):** Designed for devices with limited memory, display and processing power.

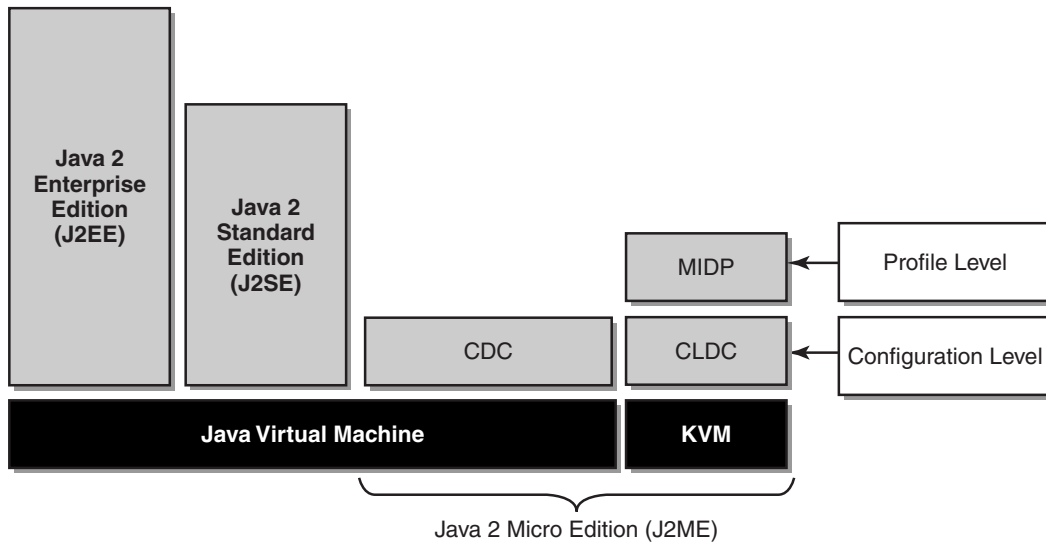


Note

In December of 1998, Sun introduced the name “Java 2” (J2) to coincide with the release of Java 1.2. This new naming convention applies to all editions of Java, Standard Edition (J2SE), Enterprise Edition (J2EE), and Micro Edition (J2ME).

Figure 1–1 shows various Java editions.

Figure 1-1 The various Java editions



Why J2ME?

J2ME is aimed squarely at consumer devices with limited horsepower. Many such devices (e.g., a mobile phone or pager) have no option to download and install software beyond what was configured during the manufacturing process. With the introduction of J2ME, “micro” devices no longer need to be “static” in nature. Not unlike a web browser downloading Java applets, an implementation of J2ME on a device affords the option to browse, download and install Java applications and content.

Small consumer electronics have a way of changing our lives. Mobile phones let us communicate when away from our home or office. Personal digital assistants (PDAs) let us access email, browse the internet and run applications of all shapes and forms. With the introduction of Java for such devices, we now have access to the features inherent to the Java language and platform. That is, a programming language that is easy to master, a runtime environment that provides a secure and portable platform and access to dynamic content, not to mention an estimated developer community of over 2 million people.

Although it would be nice to have the entire J2SE Application Programming Interface (API) available on a micro device, it’s not realistic. For example, a mobile phone with its limited display cannot provide all the functionality available in the Abstract Window Toolkit, the first graphical user interface released with Java. The “Micro Edition” was introduced to address the special needs of consumer devices that are outside the scope of J2SE and J2EE.

The capabilities of devices within the “Micro Edition” may vary greatly. An Internet Screenphone (a hardware device designed to provide access to email, news, online banking, etc.) may have a much larger display than a pager. However, even devices that seem similar in size may vary greatly in their capabilities. A cell phone and PDA are both limited in physical size, yet a typical cell phone may have a display with a total resolution of 12,288 pixels (96 × 128), whereas a PDA resolution may start at 20,000 pixels and go up from there.

One Java platform will most definitely not fit all. To better understand how J2ME will accommodate a broad range of consumer electronics and embedded devices, we need to introduce two new concepts, configurations and profiles.

Configurations

To support the broad range of products that fit within the scope of J2ME, Sun introduced the Configuration.

A **Configuration** defines a Java platform for a broad range of devices. A Configuration is closely tied to a Java Virtual Machine (JVM). In fact, a Configuration defines the Java language features and the core Java libraries of the JVM for that particular Configuration.

The dividing line as to what a Configuration applies is for the most part based on the memory, display, network connectivity (or limitations of) and processing power available on a device.

The Sun J2ME FAQ states the following: “The J2ME technology has two design centers—things that you hold in your hand and things you plug into a wall.” This may be a good general definition, but that’s exactly what it is, general. Don’t let this be your sole guide in deciding which Configuration applies.

Following are typical characteristics of devices within the two currently defined Configurations:

Connected Device Configuration (CDC)

- 512 kilobytes (minimum) memory for running Java
- 256 kilobytes (minimum) for runtime memory allocation
- Network connectivity, possibly persistent and high bandwidth

Connected, Limited Device Configuration (CLDC)

- 128 kilobytes memory for running Java
- 32 kilobytes memory for runtime memory allocation
- Restricted user interface
- Low power, typically battery powered
- Network connectivity, typically wireless, with low bandwidth and intermittent access

Although this division seems pretty clear, this won’t always be the case. Technology is continually advancing. Remember your first computer? What was “state-of-the-art” in 1985 (when I purchased my first personal computer) pales in comparison to what is available today.

The point is, as technology offers us more processing power, with increased memory and screen capabilities, the overlap between these categories will become larger. This is a nice segue to our next discussion, Profiles.

Profiles

It's all well and good that devices will fall within one Configuration or the other. For example, a typical cellular phone, PDA and pager will all fit the guidelines of the CLDC. However, what seems limiting to one device in a Configuration may be an abundance to another. Recall the analogy of the cellular phone screen size versus that of a PDA.

To address this broad range of capabilities, and to provide for more flexibility as technology changes, Sun introduced the concept of a Profile to the J2ME platform.

A **Profile** is an extension, if you will, to a Configuration. It provides the libraries for a developer to write applications for a particular type of device. For example, the Mobile Information Device Profile (MIDP) defines APIs for user interface components, input and event handling, persistent storage, networking and timers, taking into consideration the screen and memory limitations of mobile devices.

Beginning in Chapter 3, the remainder of this book will focus on MIDP specifically. This will include everything from the hardware and software requirements to complete coverage of all the APIs.

How are Configurations and Profiles Developed?

Excerpt from J2ME FAQ (<http://java.sun.com/j2me/faq.html>): Configurations and Profiles are defined by open industry working groups utilizing Sun's Java Community Process Program. In this way industries can decide for themselves what elements are necessary to provide a complete solution targeted at their industry. For more information on the Sun Community Process Program, see: <http://jcp.org>

Java Virtual Machines

As you well know, the engine behind any Java application (or applet, servlet, etc.) is the JVM.

Once you've compiled your Java source code into a class file(s), and optionally included them in a Java Archive (JAR) file, the JVM translates the class files (more accurately, the byte code in the class files) into machine code for the platform running the JVM. The JVM is also responsible for providing security, allocating and freeing memory and managing threads of execution. It's what makes your Java programs go, so to speak.

For CDC, the virtual machine has the same specification as J2SE. For CLDC, Sun has developed what is referred to as a reference implementation of a virtual machine, known as the K Virtual Machine, or simply KVM. This virtual machine was designed to handle the special considerations of resource-constrained devices. It's clear the KVM is not the "traditional" Java virtual machine:

- The virtual machine itself requires only 40 and 80 kilobytes of memory
- Only 20–40 kilobytes of dynamic memory (heap) are required
- Can run on 16-bit processors clocked at only 25 MHz

The KVM is Sun's implementation of a JVM that fits the guidelines of the CLDC. It is not necessarily the only JVM that is or will be available.

How are the KVM and CLDC Related?

From Sun's documentation: "CLDC is the *specification* for a 'class' of Java virtual machines that can run on the categories of devices targeted by CLDC and support the profiles." Essentially, the CLDC outlines requirements that must be met by the virtual machine. The KVM is what is known as a reference implementation—it is a virtual machine that meets the CLDC requirements.

Big Picture View of the Architecture

We've covered an assortment of information about J2ME. Let's put all this together into two separate scenarios. The first is a "generic" software architecture, if you will, of J2ME. The second is the architecture as it will apply to our interests as we progress through the book.

Generic Architecture

It begins with the host Operating System (OS) as the base (see Figure 1-2), followed by the virtual machine (VM). The VM will take one of two forms:

- For systems complying with the CDC, it will be the "traditional" virtual machine; that is, the same feature set as in the Java 2 Standard Edition.
- For systems complying with the CLDC, it will be the KVM or a virtual machine that meets the specifications as required by the CLDC.

CLDC or CDC core libraries are next in the hierarchy. Profiles are the topmost layer, and are designed to provide a toolkit for writing applications for a particular device family.

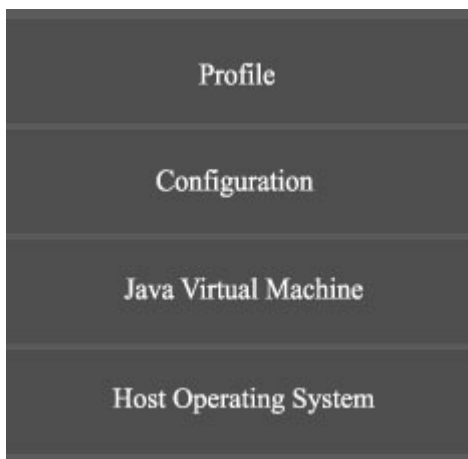


Figure 1-2 "Generic" J2ME architecture

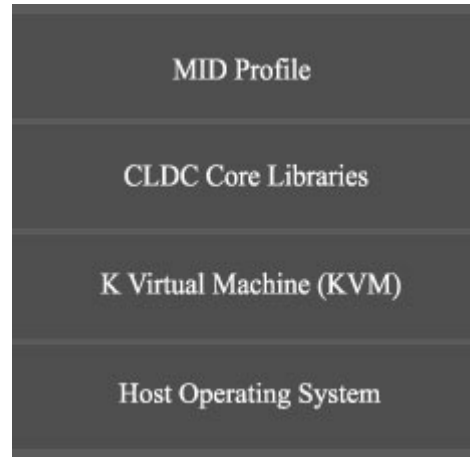


Figure 1-3 MID Profile architecture

MIDP Architecture

As before, the host OS is the base. The virtual machine will be the KVM. Remember, the KVM is Sun's implementation of a JVM meeting the CLDC specification—it may not be the only implementation available of a virtual machine for MIDP. CLDC core libraries are next, followed by MID Profile.

Compatibility between Java Editions

At the beginning of this section, I introduced Sun's Java tagline: "Write Once, Run Anywhere." Now that we've introduced Configurations, Profiles and a KVM, do you think this still applies? Well, the answer is, sort of.

Will J2SE applications run on J2ME?

J2ME is basically a slimmed down version of J2SE. Many components have been removed to keep the platform small and efficient. An obvious example is that of the Abstract Window Toolkit—many mobile devices do not have the screen capabilities to provide advanced user interface components such as overlapping windows and drop-down menus.

On the other hand, if you write J2SE code that adheres only to the classes that are available within the J2ME Configuration you are targeting, then your programs will run on both platforms. Keep in mind, such applications

will most likely be very constrained, with little to no user interface, as J2ME and J2SE offer completely different APIs for handling the display.

Will J2ME applications run on J2SE?

The same rules apply here. If you limit the code to what is common on both platforms, the answer is yes. However, the majority of software you write for a J2ME device will require special interface and event handling code. Thus, you are greatly limited to what types of programs will be appropriate for both platforms.

Putting all the Pieces Together

Sun created the Java 2 Micro Edition to allow development of Java applications for devices that do not have the same processing power and memory found on a typical desktop platform. Products may include cellular phones, PDAs, pagers, entertainment and automotive navigation systems, to name just a few.

J2ME is divided into two broad categories, known as Configurations. CDC is a set of APIs to support “fixed” devices such as a television set-top box. CLDC is a set of APIs targeted at devices that have limited processing power, display and memory. The majority of these devices will also be mobile (e.g., cellular phones and pagers).

A Configuration is closely tied to a Java virtual machine. For CDC, the virtual machine is compatible with the virtual machine of the Java 2 Standard Edition. The KVM, a virtual machine that takes into consideration the limited resources available on devices that fit this configuration, was developed for the CLDC.

On top of Configurations are device Profiles. Here you will find the APIs for user interface design, networking support and persistent storage. The Mobile Device Information Profile and the associated libraries are the main focus of this book.