Introduction to Java ME

Introduction

The basic purpose of this article is to help the beginner to understand the basic concept of Java 2 Micro Edition (Java ME). According to my opinion before understanding different API we must have the basic concept of it, and how Java ME is different than J2SE and J2EE.

Content

Java is known primarily as a server-side programming environment, centered around the technologies that make up the Java 2 Enterprise Edition (J2EE), such as Enterprise JavaBeans (EJBs), servlets, and JavaServer pages (JSPs). Early adopters of Java, however, will recall that it was originally promoted as a client-side application environment. In fact, Java was originally designed as a programming language for consumer appliances. Now Java is returning to its roots with Java 2 Micro Edition. This article, the first in a series on Java ME programming, explains what Java ME is. The Java 2 Platform

What we commonly refer to as "Java" is more formally known as the Java 2 Platform. The Java 2 Platform is split into three editions: Java 2 Standard Edition (J2SE), Java 2 Enterprise Edition (J2EE), and Java 2 Micro Edition (formerly J2ME, now Java ME). Each edition of the platform provides a complete environment for running Java-based applications, including the Java virtual machine (VM) and runtime classes.

The three editions all target different kinds of applications running on different kinds of devices. Desktop-based applications are developed using J2SE, which provides the necessary user interface classes. Server-based applications are developed using J2EE, which emphasizes component-based programming and deployment. Handheld and embedded devices are targeted by Java ME.

What separates one edition from another, then, is primarily the set of class libraries that each edition defines. Loosely speaking, you can think of Java ME as a subset of J2SE and J2SE as a subset of J2EE. It is possible to run the same Java bytecode in each edition, providing the classes referred to by the bytecode are available in all three editions. The catch, of course, is that Java ME-based devices have fewer classes than what J2SE and J2EE provide, especially the smaller devices. After all, there are several thousand core J2SE runtime classes, taking up ten to twenty megabytes of space, which is simply too big for the majority of devices out there today and in the near future.

The various specifications that comprise Java ME are all defined through the Java Community Process (JCP), as is done with J2SE and J2EE. Today, there are close to forty separate Java Specification Requests (JSRs) dealing with Java ME: Small device programming is definitely a hot topic within the Java community. (For more information, see the main JCP Web site at www.jcp.org.) Java 2 Micro Edition

In Java ME, the Java runtime environment is adapted for constrained devices - devices that have limitations on what they can do when compared to standard desktop or server computers. For low-end devices, the constraints are fairly obvious: extremely limited memory, small screen sizes, alternative input methods, and slow processors. High-end devices have few, if any, of these constraints, but they can still benefit from the optimized environments and new programming interfaces that Java ME defines.

Learning about Java ME is not hard: Once you understand the new terminology, it's mostly about learning new APIs (application programming interfaces) and learning how to work in constrained environments. (If you think writing an applet is challenging, wait until you try to fit an application into the 30K of memory some cellphones provide!) You can use most of the same tools you already use in your code development, and with careful coding you can develop libraries of classes that are portable to any device or computer with a Java virtual machine. PersonalJava and EmbeddedJava

Java ME is not the first attempt at adapting Java for constrained environments. Two other technologies, PersonalJava and EmbeddedJava, made it possible to run Java 1.1.x applications on high-end devices.

PersonalJava uses the basic Java 1.1 runtime classes and throws in a few features from Java 2. Support for some of the runtime classes is optional, but a PersonalJava implementation still requires a couple of of megabytes of memory and a fast processor to run, so it's not a practical solution for truly constrained devices like cellphones and many personal digital assistants.

EmbeddedJava makes every behavior of both the Java VM and the runtime classes optional - the implementor can choose exactly which classes and methods are required. There is one limitation, however: The Java runtime environment can only be used by the implementor and cannot be exposed to third parties. In other words, you can use it to write Java code that runs inside a device, usually as part of the software to control the device, but no one else can write applications for the device. This is done to preserve the "write once, run anywhere" nature of Java, since an EmbeddedJava environment can do away with fundamental things like runtime class verification and change the public interfaces of core classes. EmbeddedJava is really a way to build a

"private" Java runtime environment.

Both PersonalJava and EmbeddedJava are being phased out. There is a migration path from PersonalJava to Java ME, as we'll see later in this series, though the current version of PersonalJava continues to be supported. EmbeddedJava is no longer supported because Java ME defines suitable small-footprint runtime environments.