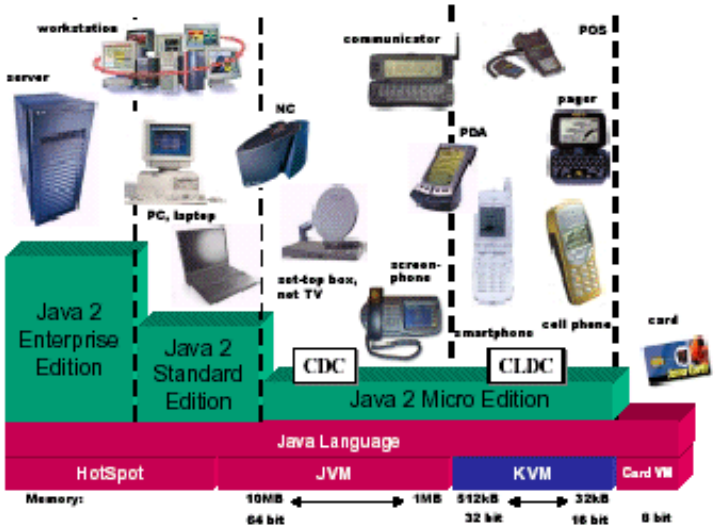
J2ME - Java 2 Micro Edition

Curso en formato PDF Tabla de contenidos

¿Qué es J2ME?
Tal como indica su nombre, J2ME (Java 2 Micro Edition) es la versión del lenguaje java desarrollado por SUN y que está orientada al desarrollo de aplicaciones para dispositivos pequeños, con capacidades restringidas tanto en pantalla gráfica, como de procesamiento y memoria (teléfonos celulares, PDAs, HandHelds, etc). A través de esta serie de articulos vamos a conocer qué es y que se necesita para poder desarrollar nuestras propias aplicaciones.
Primero y para los que ya están familiarizados con las clases de Java pueden ver en la figura-1 donde se ubica el API J2ME dentro de la plataforma Java de SUN.

Figura-1: Arquitectura plataforma Java de SUN



Tal como se observa en la figura anterior, el API J2ME utiliza la KVM (Kilo Virtual Machine) que es el nombre que recibe la maquina virtual java para los dispositivos englobados en CLDC (Connected Limited Device Configuration).

J2ME está constituído por los siguientes componentes:
• Una Máquina Virtual (**KVM** - Kilo Virtual Machine) de reducido tamaño que ejecuta el "bytecode" de las clases java.

• Conjuntos de Clases básicas, llamadas **Configuraciones**, orientadas a conformar el corazón de las implementaciones para dispositivos de características específicas.

• Librerías Java, llamadas **Perfiles**, orientadas a implementar funcionalidades de más alto nivel para familias específicas de dispositivos.

KVM
Corresponde a la Máquina Virtual más pequeña desarrollada por Sun. Para una configuración estandar, requiere entre 50 y 80 Kbytes de memoria, y dado su bajo requerimiento de memoria dinámica para funcionar, su requerimiento final no debiera superar los 180 Kbytes.
Configuraciones
La configuración define las mínimas librerias Java y las capacidades de la maquina virtual.
Actualmente está en proceso de test, aunque ya están disponibles la especificación CLDC 1.0 (Connected Limited Device Configuration), para dispositivos con un total de memoria entre 128 KB y 512 KB, conectividad a la red y alimentación limitada. Y la especificación CDC (Connected Device Configuration).
Perfiles (Profiles)
El primer perfil desarrollado se denomina MIDP (Mobile Information Device Profile), que en conjunto está diseñado para operar con el CDLC y permitir la ejecución de aplicaciones java en dispositivos móviles (MID, Mobile Information Devices). En el desarrollo de este perfil participan empresas tales como Ericcson, Fujitsu, Matsushita, Motorola, Nokia, NTT DoCoMo, Palm Computing, RIM, Sony, Siemens y Sun Microsystems, entre otros e incluye teléfonos celulares y PDA's (Personal Digital Assistants).

A continuación desarrollaremos estos conceptos y veremos un ejemplo de aplicación practica.

Top Next

Tabla de contenidos

Back Tabla de contenidos Next

Nota: he introducido este capitulo en primer lugar dedicado a todos los impacientes que necesitan ya realizar desarrollos.

Instalación y uso del J2ME Wireless Toolkit

El J2ME Wireless Toolkit soporta el desarrollo de aplicaciones java para dispositivos de tipo MIDP. Adicionalmente tiene unas herraminetas de desarrollo de aplicaciones básicas, como un preverificador de código (byte-code preverifier) y un Emulador. Además, también provee de herramientas para dos diferentes entornos:

- Desarrollo de aplicaciones desde ficheros fuente java hacia el entorno MIDlet incluyendo ficheros jar y jad.
- Dos entornos de desarrollo de tipo GUI:
  - **kToolbar Development environment**
    Mínimo entorno de desarrolo que le permite editar, compilar y ejecutar aplicaciones java con un pequeño emulador.
  - **Forte for Java Development environemnt**
    Provee un módulo opcional para desarrollar con el entorno Forte for Java (http://www.sun.com/forte/ffj/ce)

Instalación del J2ME Wireless Toolkit

1. Descargar la herramienta (http://java.sun.com/products/j2mewtoolkit)
2. Ejecutar el **j2me_wirelees_toolkit-1.0.exe**

Usar kToolBar (kToolBar Development environment)

Después de haber instalado el la herramienta, se puede ya ejecutar:

Start -> Programs -> J2ME Wireless Toolkit 1.0 -> kToolBar

Top Back Tabla de contenidos Next

Back Tabla de contenidos Next

Capitulo 2: Especificaciones de la J2ME

La Java 2 Micro Edition (J2ME) fue anunciada en Junio de 1999 por Sun Microsystems con el proposito de habilitar aplicaciones java para pequeños dispositivos. En esta conferencia lo que realmente se enseñó fue una primera versión de una nueva Java virtual machine (JVM) que podia ejecutarse en dispositivos Palm.

Una nueva máquina virtual: KVM

La plataforma Java 2 Standard Edition (J2SE) soporta dos diferentes máquinas virtuales: la clásica JVM (Java Virtual Machine) y HotSpot Virtual Machine, ahora se añade una nueva dirigida a entornos con capacidades limitadas. Esta nueva maquina se ha denominado KVM (Kuaui Virtual Machine).
La KVM acepta el mismo conjunto de bytecodes (con algunas pequeñas excepciones) que la clásica JVM.
En realidad la plataforma J2ME es más que la KVM, de hecho la JVM puede ser usada con J2ME.
J2ME soporta dos diferentes máquinas virtuales:

- La clásica máquina virtual (JVM) para arquitecturas de 32-bit y gran capacidad de memoria.
- La KVM para arquitecturas de 16-bit o 32-bit con capacidades de memoria limitada.

En realidad, la KVM continene un reducido nucleo de clases java, y además pueden ser convertidas a formato ROM.
La KVM no es el corazón de Micro Edition, sino que ha sido la primera versión oficial como referencia de máquina virtual para la configuración J2ME. Los desarrolladores pueden descargarla del site de Sun Microsystems y compilarla para Palm, Windows o Solaris, e incluso migrarla (port) a cualquier otra plataforma de otros fabricantes.

Configuraciones y Perfiles (Configurations and Profiles)

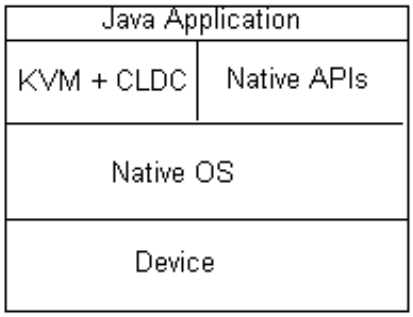
Teniendo en cuenta que la Java Standard Edition no trabaja en pequeños dispositivos, J2ME usa configuraciones y perfiles para personalizar el entorno Java runtime.
Una **configuración** define el J2ME runtime básico como una máquina virtual y un conjunto de clases que se pueden ejecutar en dispositivos con capacidades limitadas. Hasta fecha se han definido dos configuraciones:

- Connected Limited Device Configuration (CLDC)
- Connected Device Configuration (CDC)

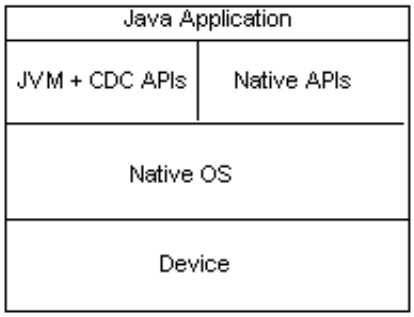
Cada una de ellas provee un mínimo conjunto de facilidades que los dispositivos en la configuración deben soportar. El CDC usa la JVM mientras que la CLDC usa la KVM.

Un **perfil (profile)** añade clases específicas para una particular configuración de J2ME.
Asi, las configuraciones están orientadas a los dispositivos y los perfiles a las aplicaciones.

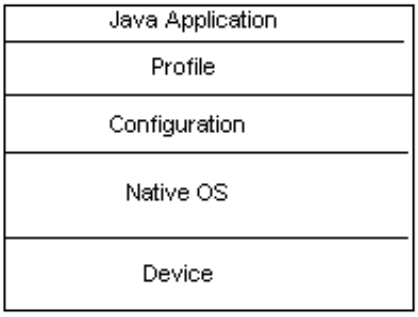
Connected, Limited Device Configuration (CLDC)

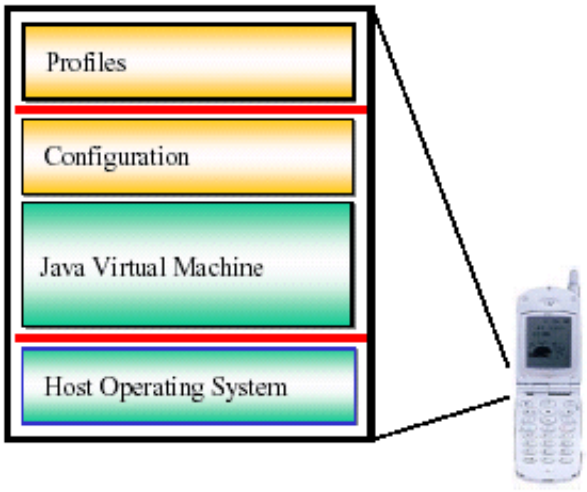


Connected Device Configuration (CDC)



Profiles





Top Back Tabla de contenidos Next

Back Tabla de contenidos Next

Capitulo 3: K Virtual Machine (KVM)

Resumen de las características del diseño de la KVM:

- Llamada así por su tamaño (kilobytes).
- Núcleo de J2ME.
- CLDC se ejecuta en el tope de KVM.
- KVM está basado en el Spotless System originalmente desarrollado por Sun Labs.
- Implementado en C, aproximadamente 24.000 líneas de código.
- Tamaño estático de VM ejecutable: entre 40 y 80 KB dependiendo de la plataforma y las opciones de compilación (w/ o romizing).
- En Palm y Win32 aproximadamente 60 KB.

Preloading/prelinking ("Romizing")

Una JVM soportando CLDC debe pre-cargar/pre-linkar (preload/prelinking) algunas clases. Esta tecnologia se denomina **Romizing.** Normalmente las implementaciones de VM pequeñas eligen preload todas las clases del sistema para una particular configuración o perfil. El mecanismo de preloading son implementaciones dependientes y usualmente están bajo el alcance de una configuración específica.

Aspectos técnicos (KVM Technical Overview)

- Compatible con JVM, con algunas restricciones.
- Implementación modular, con varias opciones en tiempo de compilación para afinar la VM (tamaño vs. velocidad) y opciones de debugging.
- Plataforma independiente, totalmente multithreading usando "green" threads.
- Pequeño ya rapido algoritmo recolector de basura (GC) (garbage collector algorithm).

KVM: Optimizaciones

- System class preloading: precarga de las clases del sistema (lo que se conoce como "**romizing**") usando JavaCodeCompact.
- Runtime simonizing of immutable structures: el termino **simonizing** se refiere a la facilidad por la cual ciertas estructuras de datos inmutables en tiempo de ejecución son movidas desde la memoria dinámica a la memoria estática para poder salvar el Java Heap Space. El termino se denominó después Doug Simon, el nombre del ingeniero que lo implementó. La técnica está implementada en la Palm, la cual es el típico ejemplo de entorno donde hay restricciones de Heap Space por programa.
- Chunky stacks y segmented heap: **Chunky Stacks** es una optimización que muestra como Java Heap Memory puede ser asignada a multiples chunks o segmentos. Esto permite a VM asignar más Heap Space en plataformas con recursos restringidos.

KVM Porting

- Código fuente bajo SCSL (Sun Community Source License).
- Tres KVM ports son validos:
  - Win32
  - PalmOS (3.01 o superior)
  - Solaris TM Operating Environment
- Rapido acceso a los partners y clientes de Sun que quieren trasladar (ported) la KVM a otras plataformas. Ver KVM Porting Guide.

Top Tabla de contenidos Next

Back Tabla de contenidos Next

Capitulo 4: Connected Limited Device Configuration (CLDC)

Una Java Community process (JCP) ha estandarizado una KVM basada en plataforma Java para dispositivos con recursos limitados. Los requerimientos están especificados en JSR-000030 (JSR-30).
Las especificaciones CLDC 1.0 se pueden descargar libremente y Sun provee una referencia de implementación.
La configuración contiene la KVM, un interprete del lenguaje Java valido para microprocesadores RISC/CISC de 16 o 32-bit con unos cientos de kilobytes de memoria.

Grupo de Dispositivos CLDC (CLDC Target Devices)
Comprende dispositivos de consumo conteniendo un Operating System (OS), frecuentemente Real Time Operating System (RTOS). Los vendedores de los sistemas operativos son los responsables de la implementación de las especificaciones de CLDC y MIDP para que las aplicaciones puedan ejecutarse en sus plataformas.
Aspectos tipicos:

- 128-512 KB de memoria
- Velocidad de procesador: 16-32 MHZ
- Normalmente alimentados con baterias
- Frecuentemente con limitada conexión de red de banda ancha (9600)
- Gran volumen de manufacturación
- Otros posibles dispositivos:
  - Dispositivos de control como máquinas de punto de venta, sensores y routers.
  - Aplicaciones domesticas.
  - Equipos de audio y video moviles.
  - Dispositivos de control de inventario de escaneado por código de barras.
  - Terminales punto de venta (TPV).

Grupo de miembros expertos en CLDC

America Online	Ericsson	Matsushita	Motorola
NTT DoCoMo	Palm Computing	Sharp	Sony
Bull	Fujitsu	Mitsubishi	Nokia
Oracle	Research In Motion (RIM)	Samsung	Siemens
Sun Microsystems	Symbian		

Alcance de CLDC

CLDC cubre las siguientes áreas:

- JVM y facilidades del lenguaje.
- Modelo de seguridad.
- Input/Output
- Soporte de red.
- Internacionalización.

Áreas intencionadamente dejadas fuera del alcance:

- Instalación de aplicaciones y gestión del ciclo de vida.
- Soporte a la interface de usuario.
- Manejador de eventos.
- Modelo de aplicaciones de alto nivel.
- Soporte para bases de datos.
- Estas facilidades están definidas en los **perfiles** como por ejemplo MIDP.

Compatibilidad entre KVM y JVM

- Total compatibilidad para el lenguaje java y la VM.

- Principales diferencias a nivel de lenguaje:
  - No hay soporte para punto flotante en CLDC 1.0. El hardware para punto flotante no es valido en muchos dispositivos CLDC.
  - Limitaciones en las librerias CLDC.
  - Dificultades para usar el modelo de seguridad J2SE.
- Diferencias en la implementación de la VM:
  - No hay Java Native Interface (JNI).
  - No hay reflection.
  - No hay grupos de thread
  - No hay referencias weak
  - No hay finalization
  - Limitado soporte en el manejo de errores.
  - Nueva implementación del classfile verification.

Seguridad en CLDC

- No se puede soportar totalmente el modelo de seguridad J2SE en los dispositivos CLDC.
  - El modelo de seguridad de la plataforma J2SE es mucho más grande que la implementación completa de CLDC.
- La seguridad CLDC consiste en dos partes:
  - Seguridad de maquina virtual de bajo nivel, una aplicación ejecutandose en la VM no puede dañar el dispositivo de ninguna manera.
  - Seguridad a nivel de aplicación: modelo Sandbox.

Top Tabla de contenidos Next

Back Tabla de contenidos Next

Capitulo 4_1: Seguridad CLDC

Seguridad a nivel de aplicación en CLDC
La seguridad a nivel de aplicación se ejecuta mediante el model Sandbox.
El modelo sandbox CLDC requiere que:

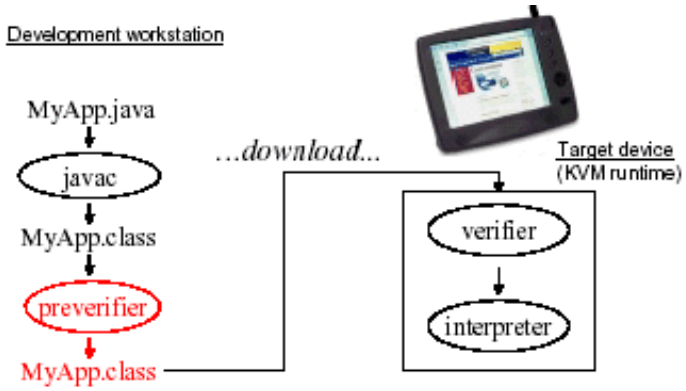
- Las clases java hayan sido adecuadamente verificadas y que garanticen que son aplicaciones java validas.
- Sólo hay una limitación, que el conjunto de APIs Java usado por el programador de aplicaciones sea el predefinido por el CLDC, perfiles y licencia de clases abiertas.
- La descarga y gestión de aplicaciones java se situa a nivel de código nativo, y el programador no puede desbordar el mecanismo de carga de clases estandard o el sistema de clases de la máquina virtual.
- El conjunto de funciones nativas accesible por la máquina virtual es cerrado. Las aplicaciones no pueden descargar nuevas librerias conteniendo funcionalidades nativas.

Top Tabla de contenidos Next

Capitulo 4_2: Classfile Verification CLDC

El verificador de clases estandard de Java es demasiado grande para los típicos dispositivos CLDC, de hecho es más grande que el propio KVM y el consumo de memoria es excesivo (> 100 KB para aplicaciones típicas).
Así que CLDC/KVM introduce un nuevo verificador de clases en dos pasos.

Figura 4_2_0



Veamos la figura anterior con un pequeño ejemplo:

1) Crear el programa fuente Hello.java

```
public class Hello {
   public static void main (String[] args)
   {
      System.out.println ("Hello CLDC/MIDP");
   }
}
```

2) Configurar el entorno CLDC

2.1) setKvm.bat

```
SET J2ME_HOME=C:\j2me_cldc
set allclasspath=".;classes;%J2ME_HOME%\bin\api\classes
set j2meclasspath=%J2ME_HOME\bin\api\classes
```

2.2) setKvm.sh

```
#!/bin/csh
setenv J2ME_HOME="$HOME"/j2me_cldc
setenv J2ME_HOME="$HOME"/j2me
set j2meclasspath="$J2ME_HOME"/bin/api/classes
```

3) Compilar el fuente

3.1) compile.bat

```
call ..\..\Scripts\setKvm.bat
md tmp
javac -g:none -d tmp -classpath %allclasspath% - bootclasspath %j2meclasspath% Hello.java
```

3.2) compile.sh

```
#!/bin/csh
```

```
source "$J2ME_HOME"/bin/setKvm.sh
test [-d tmp];mkdir tmp
javac -g:none -d tmp -classpath "$allclasspath" -bootclasspath "$j2meclasspath$ Hello.java
```

4) Preverificar las clases

4.1) preverify.bat

```
call ..\..\Scripts\setKvm.bat
%J2ME_HOME%\bin\preverify -d classes -classpath %j2meclasspath% tmp
```

4.2) preverify.sh

```
#!/bin/csh
source "$J2ME:HOME"/bin/setKvm.sh
test [-d tmp];mkdir tmp
"$J2ME_HOME"/bin/preverify -d classes -classpath "$j2meclasspath" tmp
```

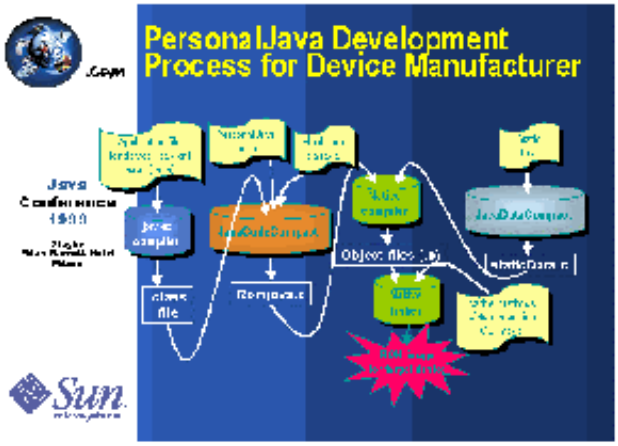
5) Ejecutar la aplicación CLDC

5.1) run.bat

```
call ..\..\Scripts\setKvm.bat
cd classes
%J2ME_HOME%\bin\kvm Hello
```

5.2) run.sh

```
#!/bin/csh
source "$J2ME_HOME"/bin/setKvm.sh
cd classes
"$J2ME_HOME"/bin/kvm Hello
```



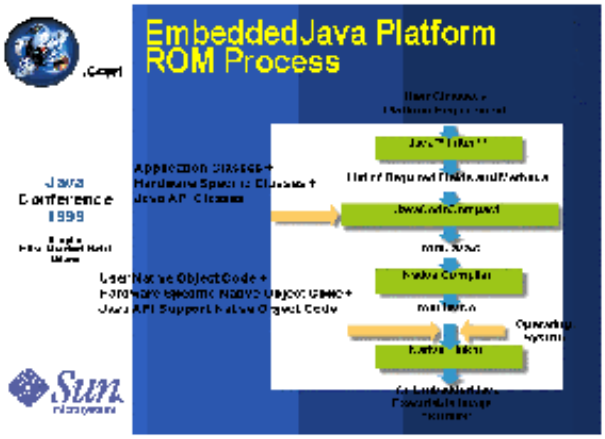
Top Tabla de contenidos Next

Back Tabla de contenidos Next

Capitulo 4_3: JavaCodeCompact

La herramienta JavaCodeCompact es usada para "ROM"ize, o crear una imagen ROM, del bytecode java para introducirlo en la memoria permanente de un dispositivo. Especificamente, la herramienta es usada para combinar la libreria de clases básicas, la máquina virtual y cualquier aplicación estática, dentro de ficheros C representando una capa ROM de la VM con la libreria de clases y la aplicación.
El fichero fuente es convertido con el ensamblador del Real Time Operating System (RTOS) en un fichero objeto.
Finalmente este fichero objeto es linkado, junto con otros ficheros objeto (incluyendo el RTOS y librerias nativas) para crear un fichero ejecutable que será introducido en el dispositivo ROM.

Figura 4_3_1



Top Back Tabla de contenidos Next

Capitulo 4_4: Librerias CLDC

Clases heredadas de la plataforma Java 2:

java.lang.*	java.io.*	java.util.*

Nuevas clases introducidas por CLDC:

javax.microedition.io.*

java.lang.*

Object	Runtime	Thread	String
Throwable	Boolean	Short	Long
Class	System	Runnable	StringBuffer
Math	Byte	Integer	Character

java.io.*

InputStream	OutputStream	DataInput	DataOutput
Reader	Writer	ByteArrayInputStream	ByteArrayOutputStream
DataInputStream	DataOutputStream	InputStreamReader	OutputStreamWriter
PrintStream			

java.util.*

Calendar	Date	TimeZone	Vector
Stack	Hashtable	Enumeration	Random

Exception classes

java.lang.Exception	java.lang.ArrayIndexOutOfBoundsException
java.lang.StringIndexOutOfBoundsException	java.lang.NegativeArraySizeException
java.lang.NullPointerException	java.lang.SecurityException
java.util.EmptyStackException	java.util.NoSuchElementException
java.io.EOFException	java.io.IOException
java.io.InterruptedException	java.io.UnsupportedEncodingException
java.io.UTFDataFormatException	java.lang.ClassNotFoundException
java.lang.IllegalAccessException	java.lang.InstantiationException
java.lang.InterruptedException	java.lang.RuntimeException
java.lang.ArithmeticException	java.lang.ArrayStoreException
java.lang.ClassCastException	java.lang.IllegalArgumentException
java.lang.IllegalThreadStateException	java.lang.NumberFormatException
java.lang.IllegalMonitorStateException	java.lang.IndexOutOfBoundsException

Error Classes

java.lang.Error	java.lang.VirtualMachineError	java.lang.OutOfMemoryError

Internationalization

new InputStreamReader(InputStream is);	new InputStreamReader(InputStream is, String name);
new OutputStreamWriter(OutputStream os);	new OutputStreamWriter(OutputStream os, String name);

Properties

microedition.platform	microedition.encoding	microedition.configuration	microedition.profiles

Top Back Tabla de contenidos Next

Capitulo 4_5: Soporte para red en CLDC

Las librerias para red (networking), I/O y almacenamiento del Standard J2SE son demasiado grandes para los dispositivos CLDC: más de 100 clases y un tamaño estático de los ficheros de clases de más de 200 kilobytes. Así que CLDC introduce un nuevo marco de trabajo que incluye 6 básicos tipos de interfaces:

- Entrada Serie (Basic Serial Input)
- Salida Serie (Basic Serial Output)
- Datagramas orientados a los dispositivos de comunicación
- Circuitos orientados a los dispositivos de comunicación (TCP,etc)
- Conexión servidor web (Basic web server connection)

Ejemplos de conexiones:

Formato general

```
Connector.open("<protocol>:<path>:<parameters>");
```

HTTP

```
Connector.open("http://www.codecamps.com");
```

Sockets

```
Connector.open("socket://129.144.111.222:9000");
```

Serial ports

```
Connector.open("comm:0;baudrate=9600");
```

Files

```
Connector.open("file:codecamps.dat");
```

Jerarquia de la clase Connection

class	methods
Connection	public void close() throws IOException;
InputConnection	public InputStream openInputstream() throws IOException;
	public DataInputStream openDataInputStream() throws IOException;
OutputConnection	public OutputStream openOutputstream() throws IOException;
	public DataOutputStream openDataOutputStream() throws IOException;
StreamConnection	Combinación de Input y OutputConnections
ContentConnection	public String getType();
	public String getEncoding();
	public String getLength();
StreamConnectionNotifier	public StreamConnection acceptAndOpen() throws IOException;
DatagramConnection	public String getAddress();
	public int getMaximumLength();
	public int getNominalLength();
	public void setTimeout(int time);
	public void send(Datagram datagram);

public void receive(Datagram datagram);
public Datagram newDatagram(int size);
public Datagram newDatagram(byte[] buf, int size);
public Datagram newDatagram(buf, int size, String addr);

Ejemplo de conexión

```
import java.io;
import javax.microedition.io.*;
public class GenConnection {
   public static void main(String[] args) {
      InputStream in = null;
      OutputStream out = null;
      int count = 0;
      int total = 0;
      StreamConnection con = null;
      byte[] b = new byte[64];
      try {
         in = Connector.openInputStream("testhttp://sraju:8080/hello.txt");
         while((count = in.read(b,0,64)) > 0)
            total += count;
      } catch (Exception e1) { // Catch FileIOException
         try {
               in.close();
         } catch (Exception e2) { System.out.println("Error"); }
        }
   System.out.println("Numero de caracteres leidos: " + total);
   System.out.println("leido de hello.txt: " + new String(b));
}
```

Top Back Tabla de contenidos Next

Back Tabla de contenidos Next

Capitulo 5: Mobile Information Device Profile (MIDP)

MIDP está diseñado para trabajar sobre CLDC. Las especificaciones se pueden obtener en Java Community Process JSR-37.

Grupo de expertos MIDP (MIDPEG):

Nokia	NTT DoCoMo	Palm	RIM
Samsung	Sharp	Siemens	Sony
Sun	Symbian	Telecordia	AOL
Ericsson	Espial Group	Fujitsu	Hitachi
J-Phone Tokyo Co.	Matsushita	Mtsubishi	Motorola
NEC	DDI		

Alcance de MIDP

- Definir la arquitectura y APIs asociados para habilitar una aplicación abierta para dispositivos moviles.
- Habilitar desarrollos de aplicaciones de terceras partes para estos dispositivos.
- Simplicidad.
- Tamaño pequeño: 128 K ROM
- Eficiencia
- MIDP fue desarrollado asumiendo los siguientes requisitos:
  - Un mínimo nucleo (kernel) para gestionar el hardware ejecutando el JVM.
  - Un mecanismo para leer y escribir en la memoria no volatil.
  - Una mínima capacidad para grabar gráficos bit-mapped en la pantalla.
  - Acceso en lectura y grabación para los dispositivos sin cable de la red (device wireless network)
  - Un mecanismo para proveer almacenamiento persistente durante un tiempo básico.

Librerias de clases para MIDP

- Application Lifecycle Package: **javax.microedition.midlet**
- User Interface Package: **javax.microedition.Icdui**
- Persistence Package: **javax.microedition.rms**
- Networking Package: **javax.microedition.io**
- Language and Utility Package: **java.lang** y **java.util**

Arquitectura MIDP

Top Back Tabla de contenidos Next

Capitulo 6: MIDlet

En el entorno MIDP, la unidad básica de ejecución es el MIDlet, que es una clase que extiende la clase javax.microedition.MIDlet.

- Reside, al menos en parte, en la memoria no volatil como una ROM o una EEPROM.
- Un MIDlet permanente debe ser descargado y escrito en el almacenamiento persistente del MID.
- Un usuario puede ejecutar el MIDlet repetidamente sin necesidad de volverlo a descargar.
- Una aplicación para MID tiene las siguientes propiedades:
  - Está concebida para un dispositivo específico.
  - Se instala, interactua y se borra, por ejemplo decargandolo via redes inalambricas desde un servidor web.
- Un MIDlet sigue un ciclo consistente en cinco fases y es responsabilidad de la aplicación moverse por estas fases:
  - Recuperación
  - Instalación
  - Lanzamiento
  - Gestión de la versión
  - Borrado

Ciclo de vida de un MIDlet

1) Recuperación
Una aplicación recupera el MIDlet desde la propia fuente. Se pueden realizar los siguientes pasos:

- Se debe especificar un medio donde el MID soporte la recuperación via cable serie, puerto infrarojos (Infra Red Data Access (IRDA)) o redes inalambricas (wireless network).
- Se produce una negociación entre el MID y el MIDlet sobre la capacidad de memoria volatil, el tamaño del MIDlet y el coste de descarga.
- Una vez se inicia el proceso el MID lee el MIDlet y lo ubica en la memoria del dispositivo.

2) Instalación
La implementación del MIDP verifica que el MIDlet no viola las reglas de seguridad y si es así transforma el MIDlet en un formato público para un dispositivo específico. Este paso puede ser tan simple como grabarlo en el almacenamiento persistente o puede preparar el MIDlet para ser ejecutado desde la memoria no volatil.

3) Lanzamiento
El usuario selecciona y lanza el MIDlet. En este momento el MIDlet entra en la KVM y los métodos del ciclo de vida del MIDlet son invocados.
El gestor de la aplicación crea una nueva instancia del MIDlet llamando al constructor por defecto (sin argumentos) y el MIDlet es posicionado en estado de pausa.
Cuando el usuario decide ejecutar el MIDlet, el gestor de la aplicación llama al método startApp(), poniendo el MIDlet en estado de activo.
El MIDlet adquiere cualquier recurso que necesite y realiza el servicio.
El gestor de la aplicación puede enviar una señal al MIDlet para detenerlo mediante una llamada al método pauseApp(). Esta acción provoca que el MIDlet se ponga en estado de pausa y libere algunos recursos.
También puede enviar una señal para destruirlo mediante el método destroyApp(). Esta acción pone al MIDlet en estado destruido. El MIDlet puede grabar el estado o las preferencias del usuario y también puede provocar una limpieza general.

4) Gestión de la versión
Una nueva versión del MIDlet puede volverse valida después de la instalación. El software de gestión de la aplicación debe saltar las pistas de las que el MIDlet ya habia sido instalado (identification) y su número de versión (version management).
Usando esta información, la antigua versión del MIDlet puede ser actualizada a una nueva versión.
Los atributos del MIDlet, incluyendo el número de versión, están contenidos en el descriptor del MIDlet o en el manifest file que es incluido con el MIDlet en un fichero JAR.

5) Borrado
El gestor de la aplicación puede borrar una instalación previa del MIDlet. los siguientes pasos ocurren durante el proceso de borrado: el gestor debe realizar una inspección y borrar la imagen instalada del MIDlet, y los posibles recursos, como registros

que han sido grabados en el almacenamiento persistente.

Estados de los MIDlet

Paused
El MIDlet está inicializado. Este estado se consigue:

- Después de que el MIDlet haya sido creado.
- Cuando el MIDlet retorna un constructor público sin argumentos.
- Si ocurre una excepción, la aplicación entra en el estado Destroyed.

Active
El MIDlet está funcionando normalmente. Este estado se consigue:

- Justo antes de la llamada al método MIDlet.startApp().

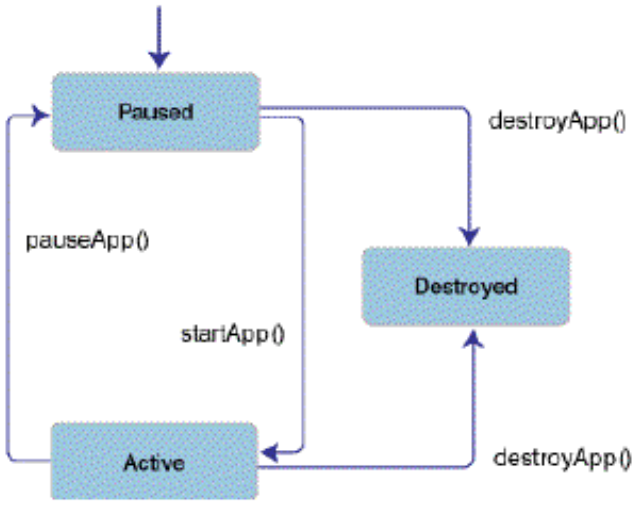
Destroyed
El MIDlet ha liberado todos los recursos y está terminado. Este estado se consigue:

- Cuándo el método MIDlet.destroyApp() retorna satisfactoriamente.
- Cuándo el método MIDlet.notifyDestroyed() retorna satisfactoriamente.

Arquitectura y Ciclo de vida de un MIDlet

El Software de Gestión de la Aplicación encuentra un MIDlet desde una fuente.

1. Retrieval
2. Installation
3. Launching
   3.a Creación de una instancia del MIDlet - Paused
   3.b Inicialización del MIDlet - Active
   3.c Terminación del MIDlet - Destroyed
4. Version Management



Top Back Tabla de contenidos Next

Back Tabla de contenidos Next

Capitulo 7: Ejemplos practicos (KVM, Palm OS y MIDlets)

En este apartado desarrollaremos tres tipos de ejercicios, donde cada uno de ellos utiliza un perfil (profile) y una configuración (configuration) determinada. Antes de comenzar, recordar que un perfíl (profile) en si mismo no es nada, sólo una especificación y se implementa mediante una configuración (configuration). Para comenzar vamos a desarrollar tres ejemplos usando el perfíl MIDP y la configuración CLDC:

7.1 Este primer caso es el más sencillo y el unico objectivo es utilizar las clases mínimas para poder ejecutar una aplicación en la **KVM.**

7.2 Este ejemplo amplia la aplicación anterior para poderla ejecutar desde una **Palm.**

7.3 En este ejemplo usaremos **MIDlets.**

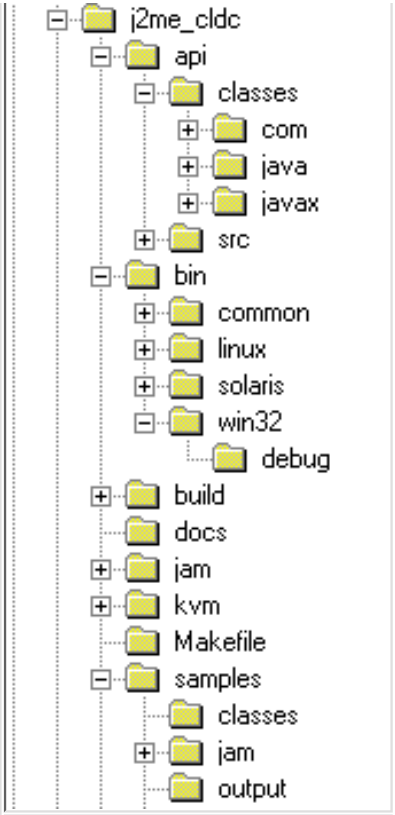
7.1. KVM (HelloWorld)

Hemos instalado el API J2ME en la unidad local C: y ejecutamos el fichero setJ2ME.bat para configurar las variables de entorno:

```
rem J2ME Environment
set J2ME_HOME=c:\j2me_cldc
set path=%path%;c:\j2me_cldc\bin\win32
set classpath=%classpath%;c:\j2me_cldc\api\classes
```

Para este ejemplo hemos instalado la siguiente estructura de directorios:

	Fichero HelloWorld.bat: rem Compilar, Preverificar y Ejecutar HelloWorld.java echo **** compilando javac -d c:\j2me_cldc\samples\classes c:\j2me_cldc\samples\src\HelloWorld.java echo **** preverificando preverify -classpath c:\j2me_cldc\api\classes;c:\j2me_cldc\samples\classes -d c:\j2me_cldc\samples\output HelloWorld echo **** ejecutando kvm -classpath c:\j2me_cldc\samples\output HelloWorld



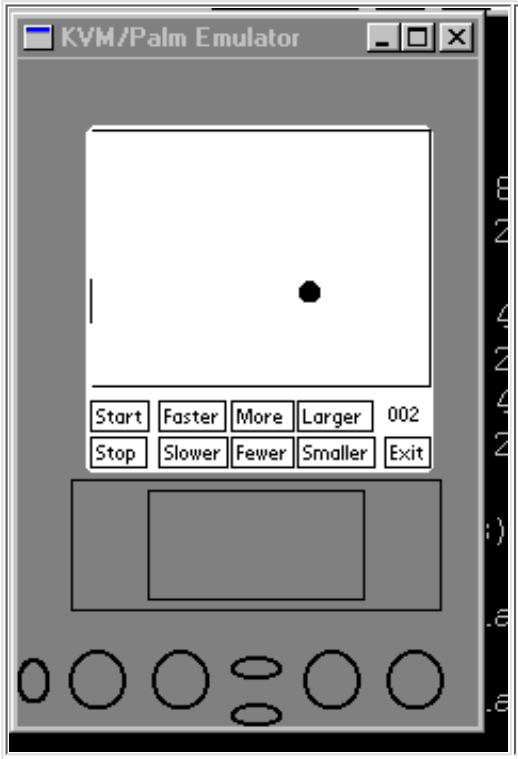
Ejecución:

```
C:\j2me_cldc\samples>kvm -classpath c:\j2me_cldc\samples\output HelloWorld
Hello World!
```

Top Back Tabla de contenidos Next

Capitulo 7.2: Ejemplos practicos (Palm OS)

7.2. **Palm OS**



Fichero runPong.bat

```
rem run KVMKJAVA for pong.Pong.java
c:\j2me_palm\bin\kjava\win32\kvmkjava -classpath
c:\j2me_palm\bin\common\api\classes;c:\j2me_palm\bin\kjava\api\classes;
c:\j2me_palm\samples\classes pong.Pong
```

Este mismo ejemplo lo vamos a convertir en una aplicación PRC para Palm OS, para lo cual y una vez convertido podemos cargarlo en nuestra propia Palm o ejecutarlo en un emulador.


```
echo **** Building a Palm executable
java -classpath c:\j2me_palm\bin\kjava\tools\palm\classes palm.database.MakePalmApp -v -version "1.1" -icon
c:\j2me_cldc\samples\palm\icons\default.bmp -bootclasspath c:\j2me_cldc\bin\common\api\classes -classpath
c:\j2me_cldc\samples\palm\classes;Pong.jar;c:\j2me_palm\bin\kjava\api\classes pong.Pong
```

Top Back Tabla de contenidos Next

Capitulo 7.3: Ejemplos practicos (MIDlets)

Todos los ficheros necesarios para un MIDlet es lo que se denomina MIDlet Suite y en concreto se compone de:

- Ficheros java encapsulados en un fichero JAR
- Fichero manifiesto describiendo el contenido del fichero jar (manifest.mf)
- Recursos como imagenes también incluidas en el fichero jar
- Fichero descriptor de la aplicación java (Java Application Descriptor File - JAD)

Además, hay un software dependiente del dispositivo y suministrado por el fabricante que es el encargado de instalar, ejecutar y borrar el MIDlet del dispositivo. Este software se denomina Application Manager.

Para poder ejecutar este primer ejemplo hemos creado el fichero **setShowProperties.bat** en el directorio **c:\midp-fcs\classes** donde se puede ver la estructura de directorios y los pasos a seguir:

```
rem set showProperties environment
set PATH=%PATH%;c:\midp-fcs\bin
set CLASSPATH=%CLASSPATH%;c:\midp-fcs\classes
echo **** compile ****
javac -bootclasspath c:\midp-fcs\classes showProperties.java
echo **** preverify ****
preverify -classpath c:\midp-fcs\classes;. -d . showProperties
echo **** JAR File ****
jar cvfm showProperties.jar manifest.mf showProperties.class showProperties
```

Ejemplo-1: showProperties

1) Fuente java: **showProperties.java**

```
import javax.microedition.midlet.*;
public class showProperties extends MIDlet {
   public void startApp() throws MIDletStateChangeException {
      System.out.println("Vendor: " + getAppProperty("MIDlet-Vendor"));
      System.out.println("Description: " + getAppProperty("MIDlet-Description"));
      System.out.println("JadFile Version: " + getAppProperty("JadFile Version"));
      System.out.println("MIDlet-Data-Size: " + getAppProperty("MIDlet-Data-Size"));
   }
   public void pauseApp() {}
   public void destroyApp(boolean unconditional) {}
}
```

2) Fichero manifiesto: **manifest.mf**

```
MIDlet-Name: Show Properties MIDlet
MIDlet-Version: 1.0.1
MIDlet-Vendor: Corporation.
MIDlet-1: ShowProps, showProperties
MicroEdition-Profile: MIDP-1.0
MicroEdition-Configuration: CLDC-1.0
MIDlet-Description: Un simple ejemplo de listado de propiedades.
MIDlet-Data-Size: 1500
```

3) Fichero JAD: **showProperties.jad**

```
MIDlet-Name: Show Properties MIDlet
MIDlet-Version: 1.0.1
MIDlet-Vendor: Corporation.
MIDlet-Jar-URL: file://showProperties.jar
MIDlet-Jar-Size: 1132
MIDlet-1: ShowProps,  , showProperties
JadFile-Version: 1.5
MIDlet-Data-Size: 500
```

4) Ejecutar el MIDlet:

midp -descriptor showProperties.jad


```
C:\midp-fcs\classes>
C:\midp-fcs\classes>midp -descriptor showProperties.jad
Vendor: Corporation.
Description: null
JadFile Version: null
MIDlet-Data-Size: 500
Execution completed successfully
65944 bytecodes executed
16 thread switches
205 classes loaded (195 bytes)
375 objects allocated (18488 bytes)
0 garbage collections
0 bytes collected
0 objects deferred in GC
0 (maximum) objects deferred at any one time
0 rescans of heap because of deferral overflow
0 pointer validations requiring heap scans
Current memory usage 18488 bytes
Heap size 300000 bytes
```

Top Back Tabla de contenidos Next

Capitulo 8.1: Empaquetando multiples MIDlets en el MIDlet Suite

1) Fichero fuente para **MIDlet1.java**

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class MIDlet1 extends MIDlet implements CommandListener {
   private Display display;                    // Referencia al objeto Display
   private TextBox tbxMain;                    // TextBox para visualizar un mensaje
   private Command cmdExit;                    // Comando para salir del MIDlet
   // Constructor
   public MIDlet1() {
      display = Display.getDisplay(this);
      cmdExit = new Command("Exit", Command.SCREEN, 1);
      tbxMain = new TEXTBox("MIDlet 1", "Welcome", 50, 0);
      tbxMain.addCommand(cmdExit);
      tbxMain.setCommandListener(this);
   }
   // Llamado por el Application Manager para iniciar el MIDlet
   public void startApp() { display.setCurrent (tbxMain); }
   // Metodos requeridos
   public void pauseApp() {}
   public void destroyApp(boolean unconditional)  {}
   // Testea si el comando Exit fue seleccionado
   public void commandAction (Command c, Displayable s) {
      if (c == cmdExit) {
         destroyApp(false);
         notifyDestroyed();
      }
   }
}
```

2) Fichero fuente java para **MIDlet2.java**

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class MIDlet2 extends MIDlet implements CommandListener {
   private Display display;                    // Referencia al objeto Display
   private List lstMain;                       // Lista de items
   private Command cmdExit;                    // Comando para salir del MIDlet
   // Constructor
   public MIDlet2() {
      display = Display.getDisplay(this);
      cmdExit = new Command("Exit", Command.SCREEN, 1);
      lstMain = new List("MIDlet 2", Choice.IMPLICIT);
      lstMain.append("Welcome Back", null);
      lstMain.addCommand(cmdExit);
      lstMain.setCommandListener(this);
   }
   // Llamado por el Application Manager para iniciar el MIDlet
   public void startApp() { display.setCurrent (lstMain); }
   // Metodos requeridos
   public void pauseApp() {}
   public void destroyApp(boolean unconditional) {}
   // Testea si el comando Exit fue seleccionado
   public void commandAction (Command c, Displayable s) {
      if (c == cmdExit) {
```

```
        destroyApp(false);
        notifyDestroyed();
    }
  }
}
```

3) Fichero **manifest12.mf**

```
MIDlet-Name: Ejemplos de MIDlets
MIDlet-Version: 1.0
MIDlet-Vendor: Siemens.
MIDlet-1: MIDlet1, /App.png, MIDlet1
MIDlet-2: MIDlet2, /App.png, MIDlet2
MicroEdition-Profile: MIDP-1.0
MicroEdition-Configuration: CLDC-1.0
```

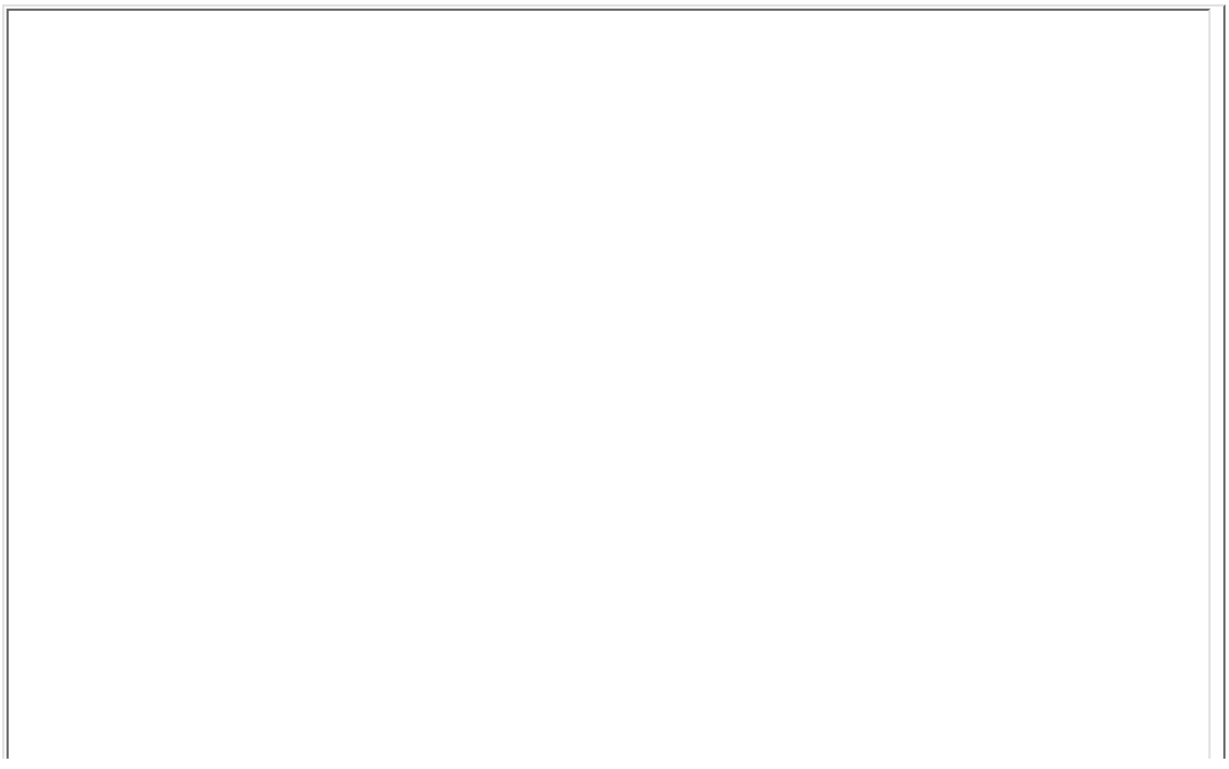
4) Fichero **setMIDlets.bat**: Compilar, preverificar, crear manifest, jar y jad.

```
rem set MIDlets environment
set PATH=%PATH%;c:\midp-fcs\bin
set CLASSPATH=%CLASSPATH%;c:\midp-fcs\classes
echo **** compile ****
javac -bootclasspath c:\midp-fcs\classes MIDlet1.java MIDlet2.java
echo **** preverify ****
preverify -classpath c:\midp-fcs\classes;. -d . MIDlet1 MIDlet2
echo **** JAR File ****
jar cvfm MIDlets.jar manifest12.mf MIDlet1.class MIDlet2.class App.png
```

5) Fichero **MIDlets.jad**

```
MIDlet-Name: Ejemplos de MIDlets
MIDlet-Version: 1.0
MIDlet-Vendor: Siemens.
MIDlet-Description: Dos ejemplos para compilar y ejecutar un MIDlet
MIDlet-Jar-URL: file://MIDlets.jar
MIDlet-Jar-Size: 3072
MIDlet-1: MIDlet1, /App.png, MIDlet1
MIDlet-2: MIDlet2, /App.png, MIDlet2
```

4) Ejecutar MIDlets: **midp -descriptor MIDlets.jad**



Top Back Tabla de contenidos Next

Capitulo 8.2: MySuite

1) Fichero fuente java para MySuite**:** ManyBalls.java
2) Fichero fuente java para MySuite: ManyCanvas.java
3) Fichero fuente java para MySuite: SmallBall.java
4) Fichero fuente java para MySuite: Sampler.java
5) Fichero MySuite.jar
6) Fichero **MySuite.jad**

```
MIDlet-Name: HelloWorld
MIDlet-Version: 1.0.0
MIDlet-Vendor: Sun Microsystems, Inc.
MIDlet-Description: Sample Hello World MIDlet
MIDlet-Info-URL: http://java.sun.com/j2me/
MIDlet-Jar-URL: MySuite.jar
MicroEdition-Profile: MIDP-1.0
MicroEdition-Configuration: CLDC-1.0
MIDlet-1: ManyBalls,, example.manyballs.ManyBalls
MIDlet-2: Sampler,, example.lcdui.Sampler
MIDlet-Jar-Size: 15875
```

4) Ejecutar MySuite: **midp -descriptor MySuite.jad**

Top Back Tabla de contenidos Next