Informática Móvil

Diseño de aplicaciones con J2ME

Introducción

José Ramón Arias García
Ignacio Marín Prendes

UNIVERSIDAD DE OVIEDO
Área de Arquitectura y Tecnología de Computadores

Curso 2004/2005

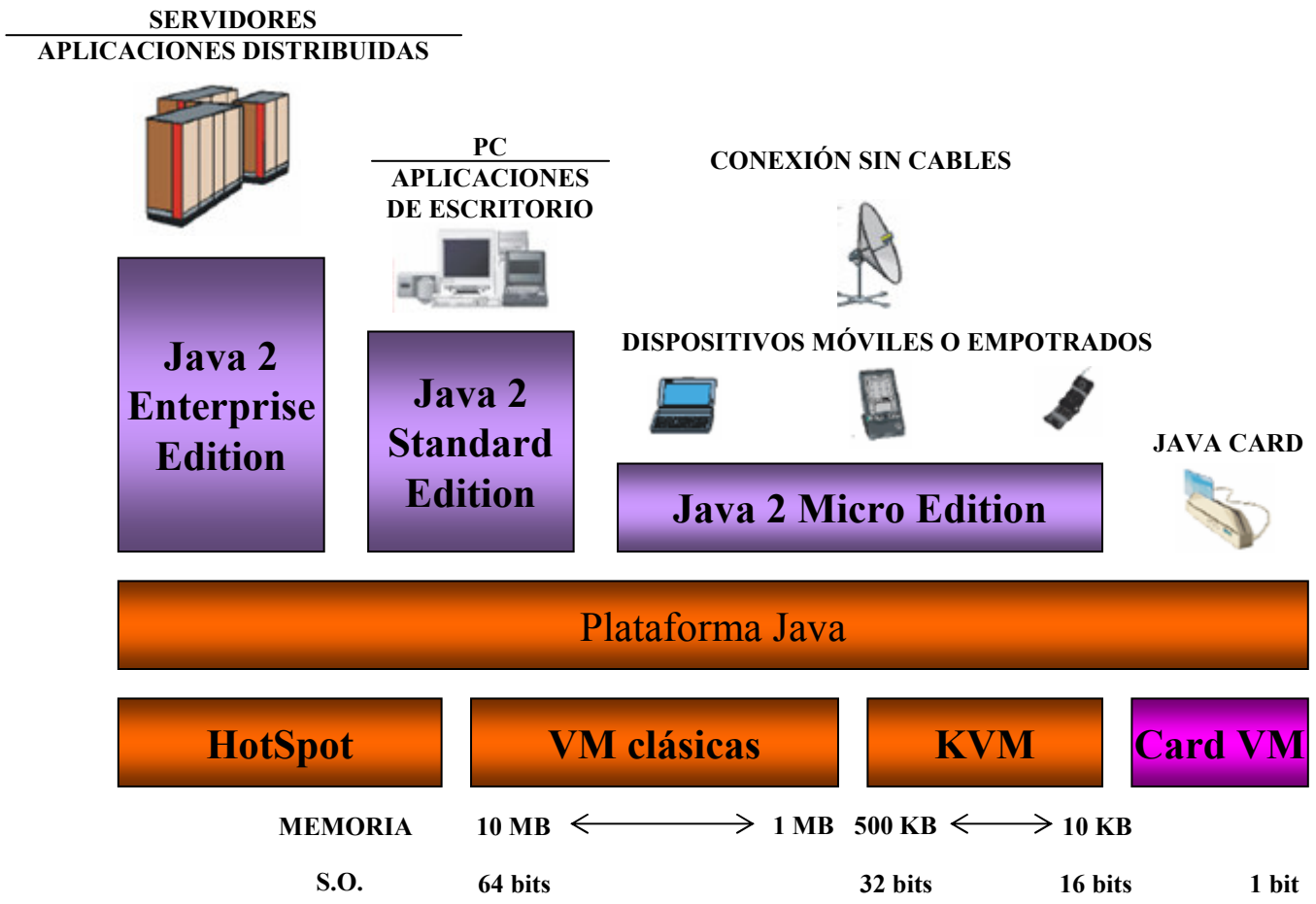
J2ME: Java para dispositivos móviles

- Java 2 Platform Micro Edition
  - Entorno en tiempo de ejecución para aplicaciones basado en una versión "reducida" de la JavaVM
  - Orientado a sistemas empotrados, teléfonos móviles, pagers, PDAs, set-top boxes, sistemas telemáticos para la industria del automóvil y electrodomésticos "inteligentes"
- Otros sabores Java 2 Platform
  - J2EE (Enterprise Edition): entornos empresariales
  - J2SE (Standard Edition): aplicaciones de escritorio
  - JavaCard: entorno smart card

Comparativa entre versiones Java

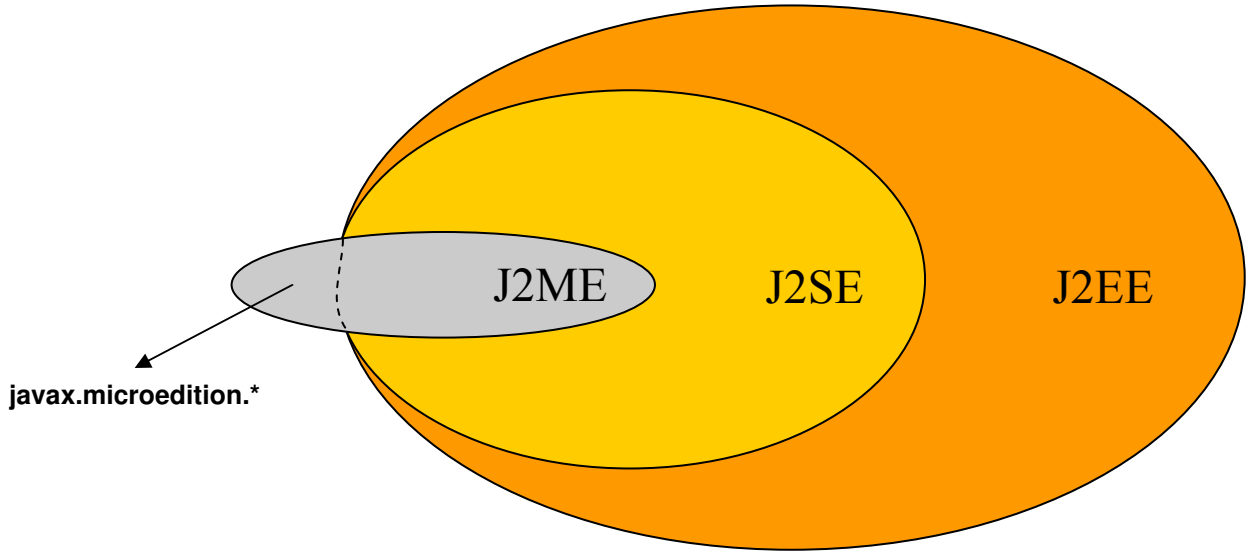
- J2SE
  - Heredera directa del lenguaje Java original
  - Orientada a desarrollo de aplicaciones de escritorio
- J2EE
  - Orientada a entornos empresariales
  - Ejecución de aplicaciones distribuidas (EJBs)
  - Integración de entornos heterogéneos, servicios web, servicios de nombres, persistencia de objetos, objetos distribuidos, XML, autentificación, gestión de transacciones, …
- J2ME
  - Orientada a dispositivos de mano y empotrados
  - Capacidades de cálculo, de almacenamiento y gráficas limitadas
  - Versión reducida de la JVM: KVM (Kilo Virtual Machine)
- JavaCard, ¿Personal Java?, ¿Embedded Java?

Estructura de la arquitectura Java



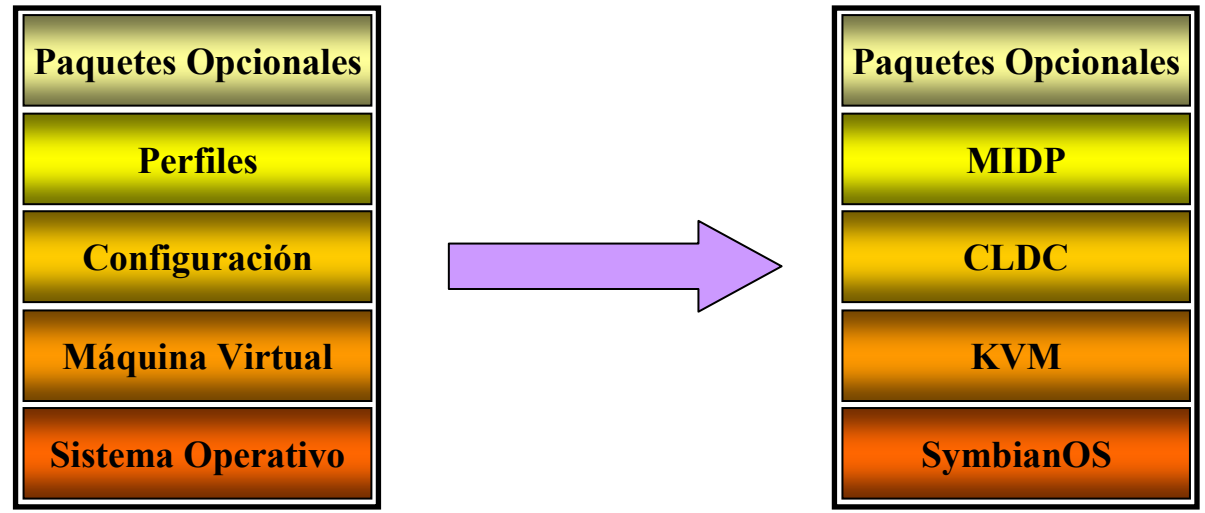
Relación entre las ediciones de Java

- Se suele considerar que:
  - J2EE es un superconjunto de J2SE
  - J2SE es un superconjnto de J2ME
- Esto último no es del todo exacto
  - J2ME dispone de un paquete adicional: **javax.microedition**
- De todas formas, es una visión demasiado simplista
  - Ámbitos de aplicación totalmente dispares



Arquitectura J2ME

- Arquitectura en capas de un entorno de ejecución J2ME



Arquitectura General

Ejemplo

Arquitectura J2ME

- Java 2 Platform Micro Edition comprende:
  - Configuraciones
  - Perfiles
  - Paquetes opcionales
- Los desarrolladores de software eligen cuáles usar y los combinan
- Objetivo: construir un entorno Java en tiempo de ejecución que
  - Cumpla las necesidades de la aplicación
  - Se ajuste a los recursos de un conjunto de dispositivos objetivo (target devices)
- Recursos: memoria, CPU, dispositivos E/S (red, pantalla, teclado, puntero, captura/reproducción de sonido, ...)

La máquina virtual

- Máquina virtual: software encargado de interpretar bytecodes
  - Efectúa las llamadas necesarias a los servicios del S.O. subyacente
  - Gestiona el cumplimiento de las reglas de seguridad y corrección de código del lenguaje Java
  - Independencia del S.O. y hardware subyacentes
- JVM estándar: demasiado pesada
- J2ME define varias VM adecuadas a los recursos disponibles
  - KVM (utilizada por la configuración CLDC)
  - CVM (utilizada por la configuración CDC)

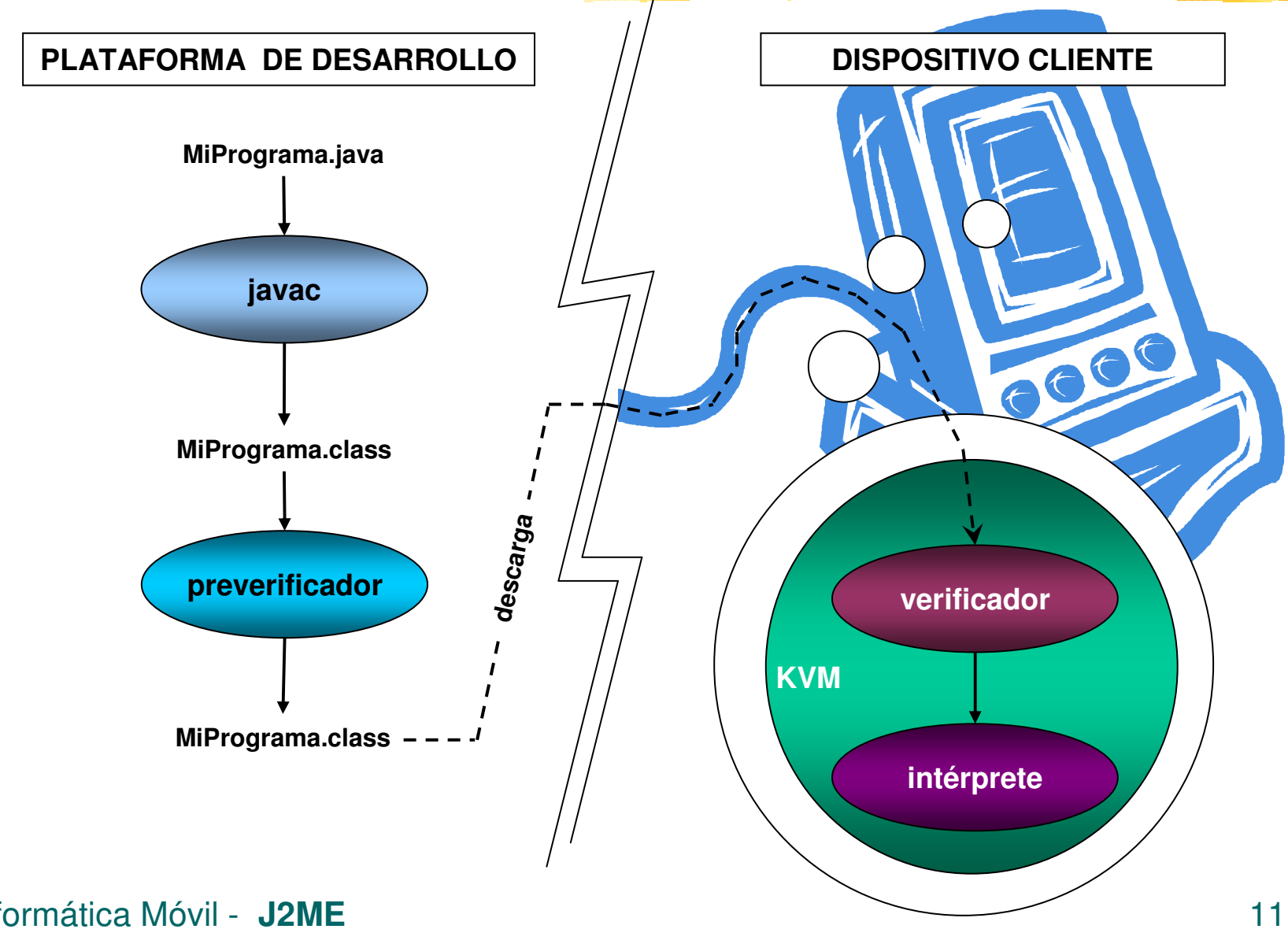
KVM (I)

- Kilo Virtual Machine
  - JVM de referencia para la configuración CLDC
  - Unas 24000 líneas de código fuente en C
  - Pequeña (40/80 KB según plataforma y opciones de compilación)
  - Altamente portable
  - Modulable; completa y rápida, sin sacrificar requisitos necesarios
- Limitaciones respecto a JVM clásica
  - No hay soporte de tipos de coma flotante (float y double)
    - Normalmente, el hardware no soporta operaciones de coma flotante
  - No hay soporte para JNI por la limitación de memoria
  - No existen *class loaders* definidos por el usuario (sólo predefinidos)
  - No se permite grupos de hilos ni hilos *Daemon* → usar *Collection*
  - No existe el método *Object.Finalize()*
  - No hay referencias débiles (candidatos a la recolección de basura)
  - Gestión de excepciones limitada (normalmente, dependiente de dispositivo)
  - No se soporta reflectividad (información de otros objetos en tiempo de ejecución)

KVM (II)

- Verificador de clases
  - Software de la Java 2 Platform dedicado a rechazar clases no válidas en tiempo de ejecución
  - El verificador de la JVM estándar es más grande que la propia KVM
  - Consumo de memoria: más de 100 KB para aplicaciones típicas
  - Verifica los *bytecodes* de las clases Java comprobando que:
    - el código no sobrepase los límites de la pila de la JVM
    - no se utilicen variables locales antes de ser inicializadas
    - se respeta el control de acceso a los elementos de una clase
- KVM+CLDC introducen un algoritmo de verificación de clases en dos pasos
  - Durante el proceso de desarrollo, tiene lugar una etapa de preverificación
  - En el dispositivo cliente, el verificador en tiempo de ejecución hace una comprobación de clases más ligera (respecto de la JVM clásica)

KVM (III)


PLATAFORMA DE DESARROLLO
DISPOSITIVO CLIENTE
MiPrograma.java
javac
MiPrograma.class
preverificador
MiPrograma.class
descarga
verificador
KVM
intérprete

CVM

- Compact Virtual Machine
  - JVM de referencia para la configuración CDC
  - Mismas características que la JVM de J2SE
  - Orientada a arquitecturas de 32 bits y mínimo de 2MB de RAM
- Características
  - Gestión avanzada de memoria
  - Bajo tiempo de espera para el recolector de basura
  - Separación completa de la VM del sistema de memoria
  - Recolector de basura modularizado
  - Ejecución de clases Java fuera de memoria ROM
  - Soporte de hilos nativos; conversión de hilos Java a hilos nativos
  - Soporte para interfaces y servicios orientados a RTOS
  - Soporte para todas las características de Java 2 v1.3: seguridad, referencias débiles, JNI, RMI y JVMDI

Configuraciones

- Conjunto mínimo de APIs Java que permiten desarrollar aplicaciones para un grupo de dispositivos.
- Estas APIs describen las características básicas, comunes a todos los dispositivos:
  - Características soportadas del lenguaje de programación Java
  - Características soportadas por la Máquina Virtual Java
  - Bibliotecas básicas de Java y APIs soportadas
- 2 configuraciones en J2ME
  - CLDC, orientada a dispositivos con limitaciones computacionales y de memoria
  - CDC, orientada a dispositivos con menos limitaciones

CDC

- Connected Device Configuration
  - Orientada a dispositivos con cierta capacidad computacional y de memoria
    - Decodificadores de TV digital, set-top boxes, sistemas de navegación en automóviles, algunos electrodomésticos
  - Características
    - Procesador de 32 bits
    - 2 MB o más de memoria total, incluyendo RAM y ROM
    - Funcionalidad completa de la VM de Java2
    - Conectividad a algún tipo de red
  - Utiliza la CVM: similar a la JVM de J2SE
    - Limitaciones gráficas y de memoria
  - Basada en J2SE v1.3
  - Incluye varios paquetes Java de la edición estándar
  - Peculiaridades: contenidas principalmente en javax.microedition.io
    - Incluye soporte para comunicaciones http y basadas en datagramas.

Paquetes CDC

- java.io
  - Clases e interfaces estándar de E/S
- java.lang
  - Clases básicas del lenguaje Java
- java.lang.ref
  - Clases de referencia
- java.lang.reflect
  - Clases e interfaces para reflectividad
- java.math
  - Paquete de matemáticas
- java.net
  - Clases e interfaces de red
- java.security
  - Clases e interfaces de seguridad

Paquetes CDC (II)

- java.security.cert
  - Clases de certificados de seguridad
- java.text
  - Paquete de texto
- java.util
  - Clases de utilidades estándar
- java.util.jar
  - Clases y utilidades para archivos JAR
- java.util.zip
  - Clases y utilidades para archivos ZIP
- javax.microedition.io
  - Clases e interfaces para conexión genérica CDC

CLDC

- Connected Limited Device Configuration
  - Orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, cómputo y memoria
    - teléfonos móviles, buscapersonas (pagers), PDAs, organizadores personales
  - Características
    - Procesador de 16 ó 32 bits (al menos, 25 MHz)
    - Entre 160 Kb y 512 Kb de memoria total disponible
      - mínimo, 128 KB de memoria no volátil para VM y bibliotecas CLDC y 32 KB de memoria volátil para VM en tiempo de ejecución
    - Bajo consumo
    - Conexión a red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600 bps).
  - Utiliza la KVM
  - Funcionalidad
    - Subconjunto del lenguaje Java + restricciones de KVM
    - Subconjunto de bibliotecas del núcleo Java
    - Soporte para E/S básica
    - Soporte para acceso a redes
    - Seguridad

Paquetes CLDC

- java.io
  - Clases y paquetes estándar de E/S (subconjunto de J2SE)
- java.lang
  - Clases e interfaces de la VM (subconjunto de J2SE)
- java.util
  - Clases, interfaces y utilidades estándar (subconjunto de J2SE)
- javax.microedition.io
  - Clases e interfaces de conexión genérica CLDC
- IMPORTANTE
  - Modelo seguridad: sandbox (como applets Java)
  - Ciclo de vida de la aplicación
    - Una configuración no se encarga del mantenimiento del ciclo de vida de la aplicación, interfaces de usuario o manejo de eventos
    - Estas responsabilidades caen en manos de los **perfiles**

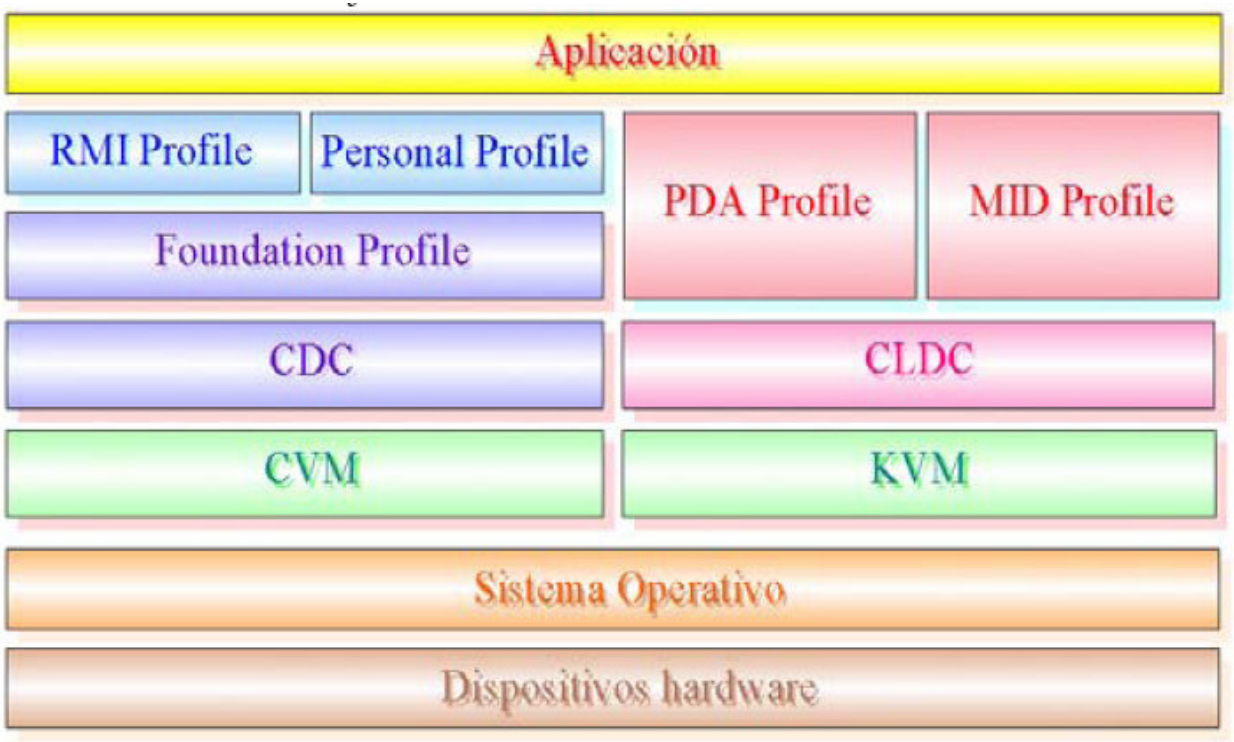
Perfiles

- Encargados de definir APIs que controlan:
  - Ciclo de vida de la aplicación
  - Interfaces de usuario
  - Gestión de eventos
- Es decir, conjunto de APIs orientado a un ámbito de aplicación determinado
- Identifican
  - Grupos de dispositivos (desde el punto de vista de su funcionalidad: electrodomésticos, teléfonos móviles, etc.)
  - Tipo de aplicaciones a ejecutar sobre ellos
- Interfaz gráfica
  - Componente muy importante en la definición de perfiles (desde texto en móviles a PDAs táctiles)

Perfiles (II)

- Configuración: APIs para una familia de dispositivos
- Perfil: APIs para un tipo de dispositivo concreto
  - Perfil = Conjunto de APIs que dotan a una configuración de una funcionalidad específica
- Perfiles para CDC
  - Foundation Profile
  - Personal Profile
  - RMI Profile
- Perfiles para CLDC
  - PDA Profile
  - Mobile Information Device Profile (MIDP)

Arquitectura del entorno de ejecución J2ME



- IMPORTANTE
  - Se puede construir un perfil sobre otro
  - Sólo puede haber una configuración

Foundation Profile

- Define APIs sobre CDC orientadas a dispositivos sin interfaz gráfica
  - Incluye gran parte de paquetes de J2SE
  - Excluye totalmente java.awt y java.swing (que conforman la interfaz gráfica de usuario de J2SE)
- Si una aplicación necesita una GUI, entonces es necesario un perfil adicional

Paquetes de Foundation Profile

- java.lang
  - Soporte del lenguaje Java
- java.util
  - Soporte completo para zip y otras funcionalidades (java.util.Timer)
- java.net
  - Incluye sockets TCP/IP y conexiones HTTP
- java.io
  - Clases Reader y Writer de J2SE
- java.text
  - Incluye soporte para internacionalización
- java.security
  - Soporte de certificados

Personal Profile

- Subconjunto de J2SE v1.3
  - Proporciona entorno con soporte completo de AWT
  - Añade a CDC una GUI completa con capacidades web y soporte de applets Java
  - Necesita una implementación de Foundation Profile

Paquetes de Personal Profile

- **java.applet**
  - Clases para crear applets o utilizadas por éstos
- **java.awt**
  - Clases para crear GUIs con AWT
- **java.awt.datatransfer**
  - Clases e interfaces para transmitir datos entre aplicaciones
- **java.awt.event**
  - Clases e interfaces para manejar eventos AWT
- **java.awt.font**
  - Clases e interfaces para manipulación de fuentes

Paquetes de Personal Profile (II)

- **java.awt.im**
  - Clases e interfaces para crear métodos editores de entrada
- **java.awt.im.spi**
  - Interfaces que añaden el desarrollo de métodos editores de entrada para cualquier entorno de ejecución Java
- **java.awt.image**
  - Clases para crear y modificar imágenes
- **java.beans**
  - Clases que soportan Java Beans
- **javax.microedition.xlet**
  - Interfaces usados por Personal Profile para comunicación

RMI Profile

- Perfil construido sobre Foundation Profile
  - Soporta un subconjunto de las APIs RMI de J2SE v1.3
  - Se han eliminado algunas características debido a las limitaciones de cómputo y memoria de los dispositivos
  - Principalmente, las propiedades:
    - java.rmi.server.disableHTTP
    - java.rmi.activation.port
    - java.rmi.loader.packagePrefix
    - java.rmi.registry.packagePrefix
    - java.rmi.server.packagePrefix

RMI Profile (II)

- Toda implementación del perfil debe soportar:
  - La semántica completa de llamadas RMI
  - Soporte de "marshalled objects"
  - Protocolo "RMI wire"
  - Exportación de objetos remotos a través de la APIUnicastRemoteObject
  - Recolección de basura distribuida e interfaces recolector de basura para los lados cliente y servidor
  - La interfaz "activator" y el protocolo de activación en el lado del cliente
  - Interfaces de registro RMI y exportación de un objeto registro remoto

Paquetes de RMI Profile

- **java.rmi**
  - El paquete RMI
- **java.rmi.activation**
  - Subconjunto de clases para RMI Object Activation
- **java.rmi.dgc**
  - Clases e interfaces para recolección de basura distribuida (DGC: Distributed Garbage-Collection)
- **java.rmi.registry**
  - Clase e interfaces para el RMI registry
- **java.rmi.server**
  - Clases e interfaces para soportar el lado servidor de RMI en modo unicast

PDA Profile

- Perfil construido sobre CLDC
  - Objetivo: PDAs de gama baja, con algún tipo de puntero y pantalla con resolución de al menos 20000 pixels y relación de aspecto 2:1 (200x100)
- Consta de dos paquetes opcionales separados que dan soporte a características normalmente presentes en PDAs
  - Uno para acceder a datos PIM (Personal Information Management): **PIM Optional Package**
  - Otro para acceder a sistemas de ficheros: **FileConnection Optional Package**

PIM Optional Package

- Objetivo principal: proporcionar acceso a datos PIM en dispositivos J2ME
- Datos PIM: información incluida en
  - Libretas de direcciones
  - Aplicaciones calendario
  - Aplicaciones de listas de tareas ("to-do lists")
- Puede proporcionar acceso a bases de datos PIM no residentes en el dispositivo
  - En ubicaciones "bien conocidas"
  - Ejemplos: tarjetas SIM; bases de datos creadas y gestionadas por la VM
  - Permite proporcionar
    - BDs PIM en dispositivos J2ME que carecían de ellas
    - Acceso a BDs PIM remotas desde dispositivos J2ME

FileConnection Optional Package

- Objetivo principal: proporcionar acceso a sistemas de ficheros en dispositivos y/o tarjetas de memoria montadas en dispositivos J2ME
- Utiliza un framework llamado GCF
  - Generic Connection Framework
  - Precisa de soporte de sistemas de ficheros en hardware y S.O.
- Los sistemas de ficheros pueden residir en:
  - Memoria del dispositivo
  - Tarjetas de memoria
    - SD/MMC, CF, SmartMedia, Memory Stick

Paquetes de PDA Profile

- **javax.microedition.pim**
  - Acceso a datos PIM en el PIM Optional Package
- **javax.microedition.io.file**
  - Acceso a sistema de ficheros basado en GCF en el FileConnection Optional Package
- Ambos paquetes pueden ser instalados por separado según se desee soporte de una de las especificaciones o de las dos a la vez

MIDP (Mobile Information Device Profile)

- Perfil construido sobre CLDC
  - Objetivo: dispositivos con
    - Reducida capacidad computacional y de memoria
    - Conectividad limitada (en torno a 9600 bps)
    - Capacidad gráfica muy reducida (display mínimo de 96x54 pixels monocromo)
    - Entrada de datos alfanuméricos reducida
    - 128 KB de memoria no volátil para componentes MIDP
    - 8 KB de memoria no volátil para datos persistentes de aplicaciones
    - 32 KB de memoria volátil en tiempo de ejecución para la pila Java
  - Ejemplos: teléfonos móviles, buscapersonas (pagers) o PDAs de gama baja con conectividad
- Primer perfil creado en J2ME

Paquetes MIDP

- **javax.microedition.lcdui**
  - Clases e interfaces para GUIs
- **javax.microedition.rms**
  - Record Management Store. Soporte para el almacenamiento persistente del dispositivo.
- **javax.microedition.midlet**
  - Clases de definición de la aplicación
- **javax.microedition.io**
  - Clases e interfaces de conexión genérica
- **java.io**
  - Clases e interfaces de E/S básica
- **java.lang**
  - Clases e interfaces de la VM
- **java.util**
  - Clases e interfaces de utilidades estándar

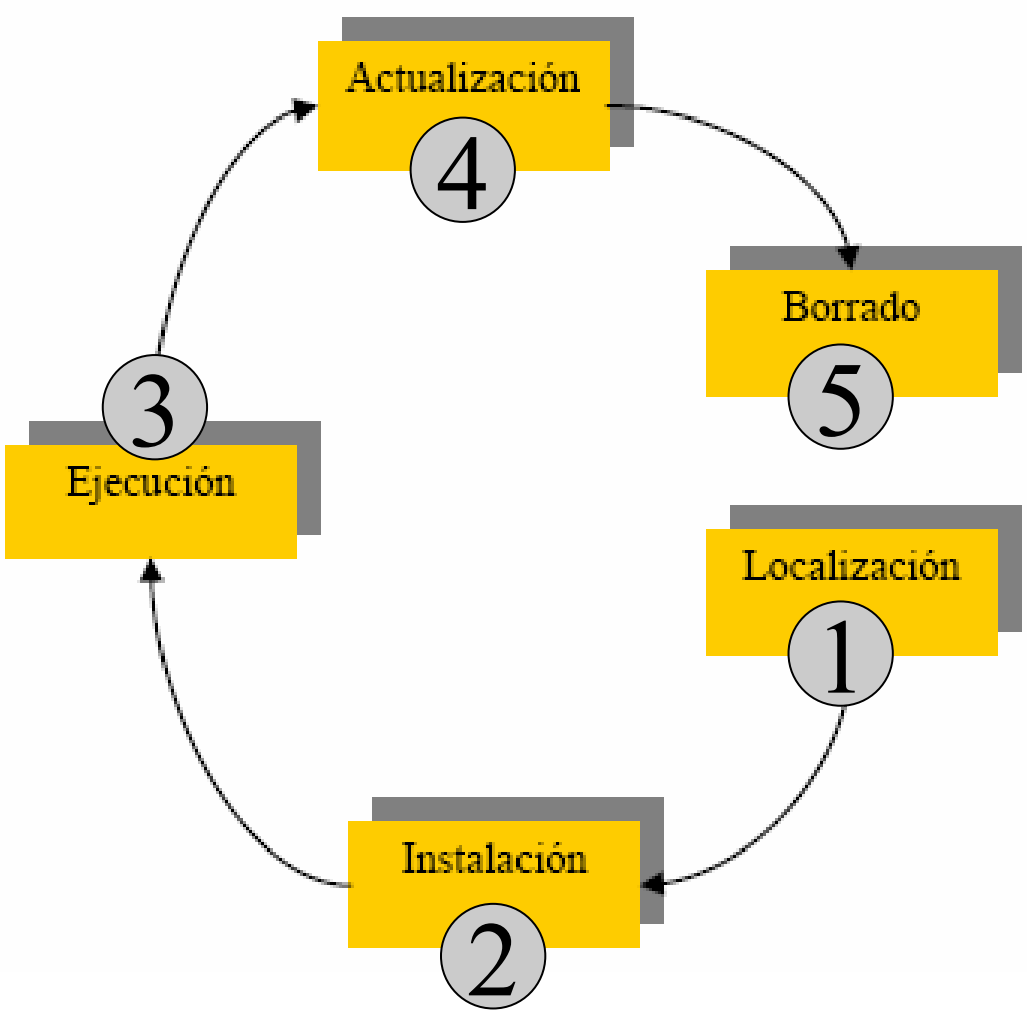
MIDLets

- Un MIDlet es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC
- Están pensados para ser descargados a través de una conexión a internet.
- El medio empleado para garantizar esta descarga recibe el nombre de OTA (Over the Air)
  - «*Over The Air User Initiated Provisioning Recommended Practice for the Mobile Information Device Profile*», Sun Microsystems, 2001 (para MIDP 1.0)
  - «*Over The Air User Initiated Provisioning Specification for the Mobile Information Device Profile*», Sun Microsystems, 2002 (para MIDP 2.0)
- Una aplicación J2ME está formada por:
  - Archivo JAR (Java Archive): contiene a la aplicación en sí
  - Archivo JAD (*Java Archive Descriptor*): contiene información sobre la aplicación.

MIDLets: Requisitos funcionales

- Dispositivo MID (*Mobile Information Device*)
  - Dispositivo que sigue la especificación MIDP
- Todo dispositivo MID dispone de un software residente llamado AMS (*Application Management Software* o Gestor de Aplicaciones)
- El AMS gestiona el ciclo de vida de los MIDLets
- Etapas del ciclo de vida de un MIDLet
  - Localización
  - Instalación
  - Ejecución
  - Actualización
  - Borrado

MIDLet: Ciclo de vida



MIDLet: Ciclo de Vida

1. Localización o Descubrimiento

- Selección a través del AMS de la aplicación a descargar
- Según las capacidades del dispositivo, descarga mediante:
  - Cable conectado al ordenador (serie, USB,…)
  - Wi-Fi, BlueTooth, GPRS, UMTS, …
- Protocolo descarga
  - HTTP 1.1 u otro protocolo con funcionalidad similar
- El usuario localiza un MIDLet con su dispositivo
  - Típicamente, a través de hiperenlaces
- Si el destino del hiperenlace es un archivo JAR:
  - El archivo JAR y su URL son enviados al AMS del dispositivo

MIDLet: Ciclo de Vida

1. Localización o Descubrimiento

- Si el destino del hiperenlace es un archivo JAD:
  - Transferencia del JAD y de su JAR asociado
  - Archivo JAD convertido a *Unicode* antes de ser usado
  - El JAD debe contener los atributos obligatorios de la especificación MIDP
  - Atributos JAD comprensibles según la sintaxis de la especificación MIDP
  - El usuario debe poder confirmar la instalación del *MIDlet* y debería haber un control de versiones.

MIDLet: Ciclo de Vida

2. Instalación

- Una vez descargada la aplicación
- El AMS controla el proceso, informando al usuario de la evolución del mismo y de posibles problemas
- Una vez instalado, todas sus clases, archivos y almacenamiento persistente están preparados para su uso
- Después de la fase de instalación, el MIDlet queda almacenado en una zona de memoria persistente del dispositivo MID (hasta que el usuario utilice el AMS para borrarlo)

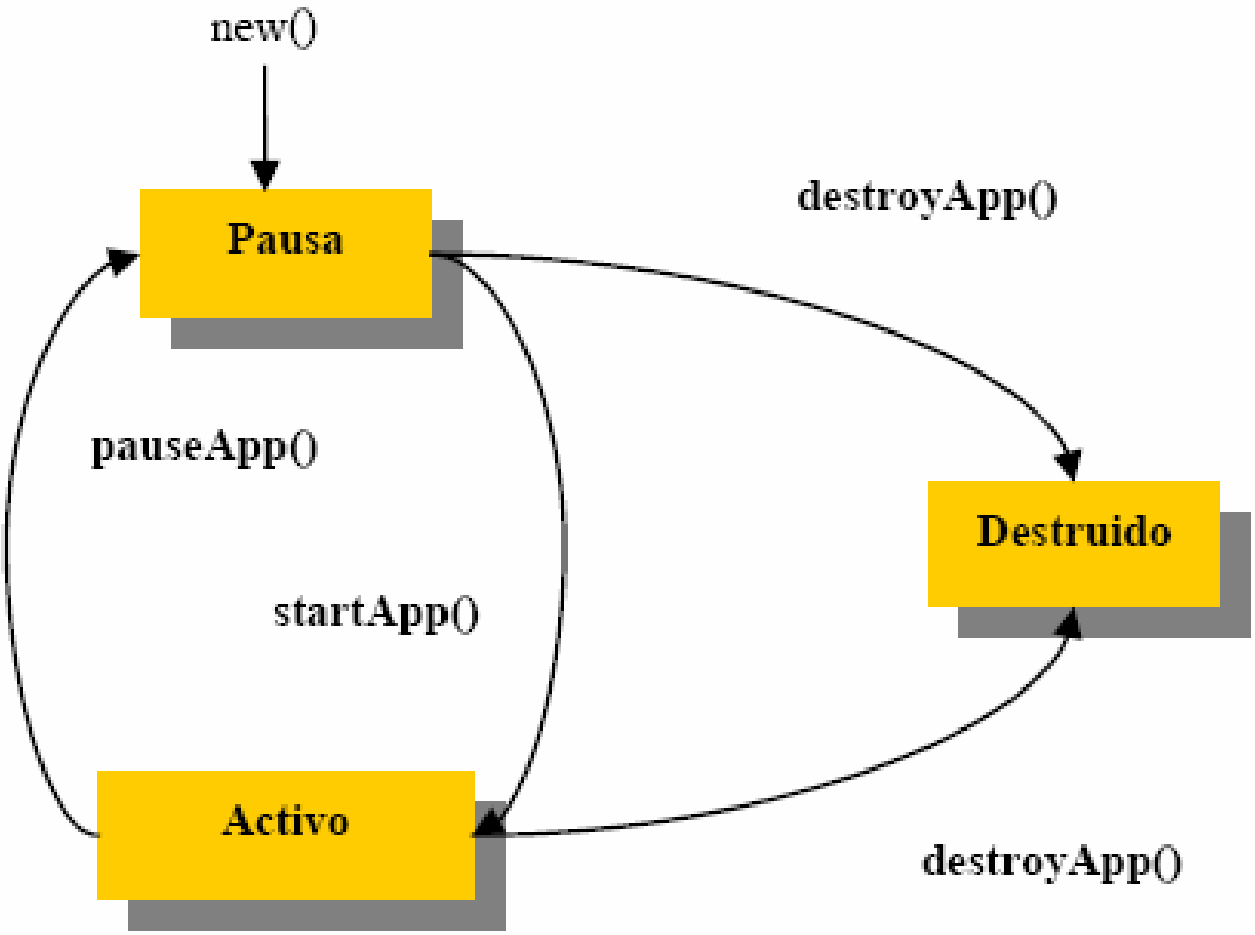
MIDLet: Ciclo de Vida

3. Ejecución

- Mediante el AMS, se inicia la ejecución de los MIDlets.
  - En esta fase, el AMS gestiona los 3 posibles estados de ejecución del MIDlet, según los eventos que se produzcan durante su ejecución
  - Estados: pausa, activo y destruído
  - Transición de estados: mediante la invocación de métodos de la clase MIDLet (new, startApp, pauseApp, destroyApp)
- El dispositivo debe invocar a las clases CLDC y MIDP requeridas por la especificación MIDP
- Si existen varios *MIDlets* presentes, posibilidad de seleccionar el *MIDlet* que se desea ejecutar

MIDLet: Ciclo de Vida

3. Ejecución (diagrama de estados)



MIDLet: Ciclo de vida

3. Ejecución (aspectos a tener en cuenta)

- **Pausa:**
  - Al llamar al constructor del MIDlet, éste permanece en "Pausa" por un corto periodo de tiempo
  - "Activo" → "Pausa", bien por una acción del usuario o bien a causa del propio AMS (p.e., llamada entrante)
  - Uso mínimo de recursos y nunca mantener un recurso compartido
- **Destruido**:
  - El *MIDlet* ha finalizado su ejecución o una aplicación prioritaria necesita ser ejecutada en memoria No puede transitar a otro estado
  - Se liberan todos los recursos utilizados en ejecución
  - Eliminado de la memoria volátil del dispositivo.
  - El MIDLet continúa en la zona de memoria persistente, dispuesto a ser ejecutado otra vez

MIDLet: Ciclo de Vida

4. Actualización

- El AMS detecta, tras una descarga, si el MIDlet descargado es una actualización de un MIDlet ya presente en el dispositivo
- Si es así, debe de informar acerca de esto, además de ofrecer la oportunidad de decidir si interesa o no dicha actualización

MIDLet: Ciclo de Vida

5. Borrado

- El AMS debe permitir al usuario eliminar MIDlets
- Antes de eliminar una aplicación, el usuario debe dar su confirmación
- El dispositivo debería advertir al usuario de cualquier circunstancia especial durante la eliminación del MIDlet
  - Por ejemplo, el MIDlet a borrar podría contener a otros MIDlets, y el usuario debería de ser alertado ya que todos ellos quedarían eliminados

MIDLets: Proceso de desarrollo

- La plataforma de desarrollo (PC, por ejemplo) y la de ejecución (dispositivos MID, *Mobile Information Device*) son distintas
- Uso de algún emulador para realizar las pruebas de la aplicación en la plataforma de desarrollo
  - Este emulador puede representar a un dispositivo genérico o puede emular algún modelo de MID específico
- Etapas básicas en la construcción de un MIDLet:
  - 6 etapas iterativas: Desarrollo, compilación, preverificación, empaquetamiento, ejecución y depuración

MIDLets: Etapas del proceso de desarrollo

- Etapas
  - **Desarrollo**: Escritura del código que conforma el *MIDlet*
  - **Compilación**: Compilación de la aplicación mediante un compilador J2SE
  - **Preverificación:** Comprobar que el código del *MIDlet* no viola ninguna restricción de seguridad de la plataforma J2ME
  - **Empaquetamiento**: Creación del archivo JAR que contiene los recursos de la aplicación, así como de un archivo descriptor JAD
  - **Ejecución**: Mediante un emulador que permita ejecutar el *MIDlet*
  - **Depuración**: Detección de fallos detectados en el *MIDlet* durante la anterior fase de ejecución.
- Cualquier aplicación en Java sigue este proceso, excepto las etapas de empaquetamiento y preverificación (exclusivas de la plataforma J2ME)

Paquete *javax.microedition.midlet*

- Define las aplicaciones MIDP y su comportamiento ante el entorno de ejecución
- Clases incluidas:
  - MIDlet
    - Esqueleto de aplicación MIDP
  - MIDletstateChangeException
    - Indica que el cambio de estado ha fallado

Clase *javax.microedition.midlet.MIDlet*

- public abstract class MIDlet
  - Las aplicaciones deben extender esta clase
  - Puede cambiar de estado invocando a sus métodos

Resumen	
Constructores	
protected	MIDlet()
Métodos	
int	checkPermission(String permiso)
protected abstract void	destroyApp(boolean unconditional)
String	getAppProperty(String key)
void	notifyDestroyed()
void	notifyPaused()
protected abstract void	pauseApp()
boolean	platformRequest()
void	resumeRequest()
protected abstract void	startApp()

Clase *javax.microedition.midlet.MIDlet*

```
import javax.microedition.midlet.*
public class MiMidlet extends MIDlet {
      public MiMidlet() {
            /* Éste es el constructor de clase. Aquí debemos
            inicializar nuestras variables.
            */
      }
      public startApp(){
            /* Aquí incluiremos el código que queremos que el
            MIDlet ejecute cuándo se active.
            */
      }
      public pauseApp(){
            /* Aquí incluiremos el código que queremos que el
            MIDlet ejecute cuándo entre en el estado de pausa
            (Opcional)
            */
      }
      public destroyApp(){
            /* Aquí incluiremos el código que queremos que el
            MIDlet ejecute cuándo sea destruido. Normalmente
            aquí se liberaran los recursos ocupados por el
            MIDlet como memoria, etc. (Opcional)
            */
      }
}
```

ESQUELETO TÍPICO DE UN MIDlet
(métodos obligatorios)

Clase *javax.microedition.midlet.MIDlet*

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class HolaMundo extends MIDlet{
private Display pantalla;
private Form formulario = null;
      public HolaMundo(){
            pantalla = Display.getDisplay(this);
            formulario = new Form("Hola Mundo");
      }
      public void startApp(){
            pantalla.setCurrent(formulario);
      }
      public void pauseApp(){
      }
      public void destroyApp(boolean unconditional){
            pantalla = null;
            formulario = null;
            notifyDestroyed();
      }
}
```

Hola Mundo

Informática Móvil

Diseño de aplicaciones con J2ME

Introducción

José Ramón Arias García
Ignacio Marín Prendes

UNIVERSIDAD DE OVIEDO
Área de Arquitectura y Tecnología de Computadores

Curso 2004/2005