
# Curso de Java

## Unidad VII

# “Java para Dispositivos Móviles”

Rogelio Ferreira Escutia

# *Contenido*

*1) Historia*
*2) Características de J2ME*
*3) Descargar una aplicación en un dispositivo móvil*
*4) Instalación de Herramientas de Desarrollo*
*5) Desarrollo de Aplicaciones*
*6) MIDlets*
*7) Interfaces de Usuario*

# 1) Historia

# Fusión de Tecnologías

- Java 2 Micro Edition: Java in Small Things, James White y David Hemphill, Manning Publications 2002

# J2ME - Historia

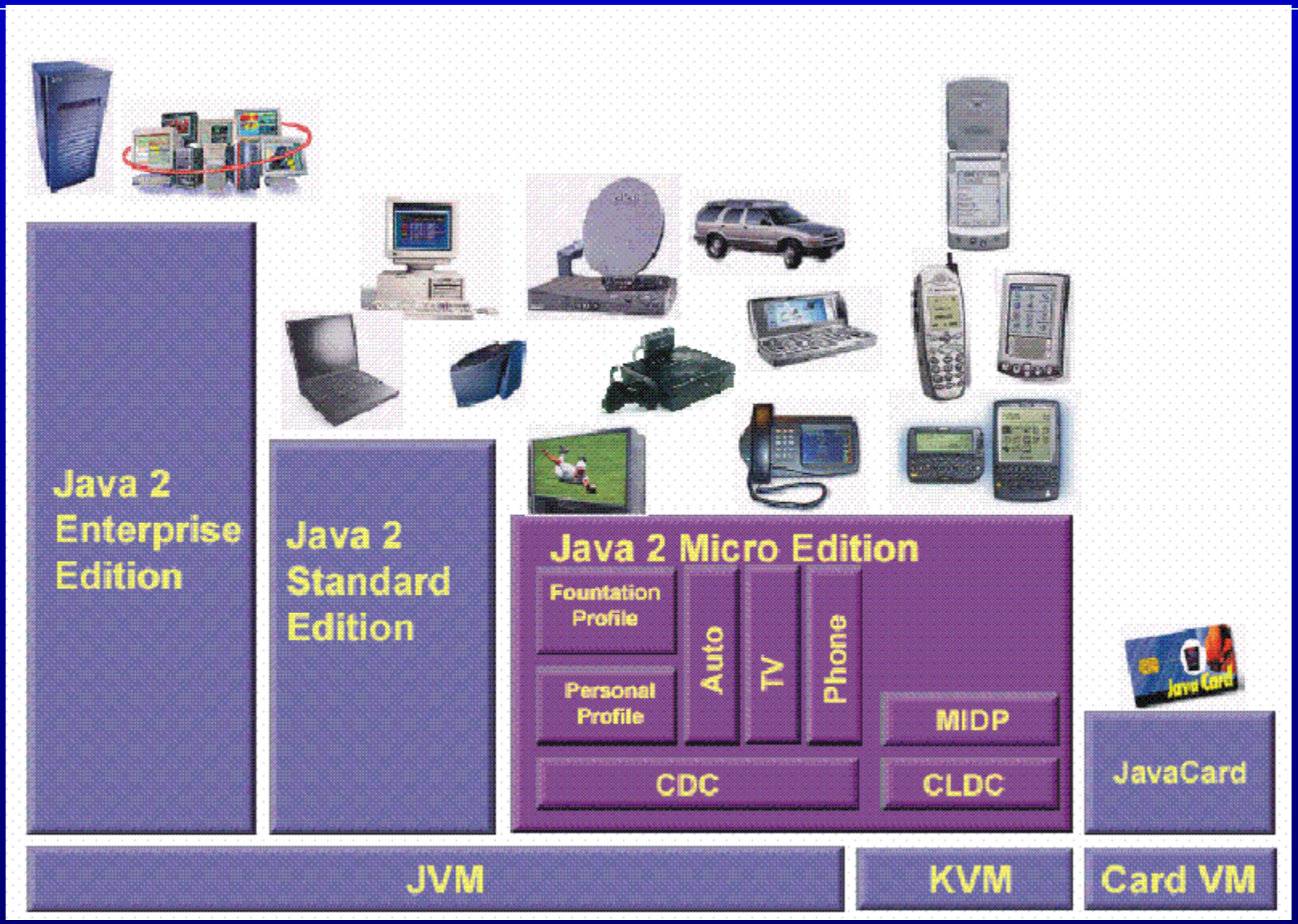
- La edición Java 2 Micro Edition fue presentada en 1999 por Sun Microsystems con el propósito de habilitar aplicaciones Java para pequeños dispositivos.
- En esta presentación, lo que realmente se enseñó fue una primera versión de una nueva Java Virtual Machine (JVM) que podía ejecutarse en dispositivos Palm.
- Java Micro Edition es la versión del lenguaje Java que está orientada al desarrollo de aplicaciones para dispositivos pequeños con capacidades restringidas tanto en pantalla gráfica, como de procesamiento y memoria (teléfonos móviles, PDA`s, Handhelds, Pagers, etc).

# Posibles Plataformas – J2ME


- Java 2 Micro Edition: Java in Small Things, James White y David Hemphill, Manning Publications 2002

# 2) Características de J2ME

# Arquitectura Java



# Entorno de Ejecución

- **Un entorno de ejecución de J2ME se compone de:**

  **a) Máquina virtual.**
  **b) Configuración.**
  **c) Perfil.**
  **d) Paquetes Opcionales.**

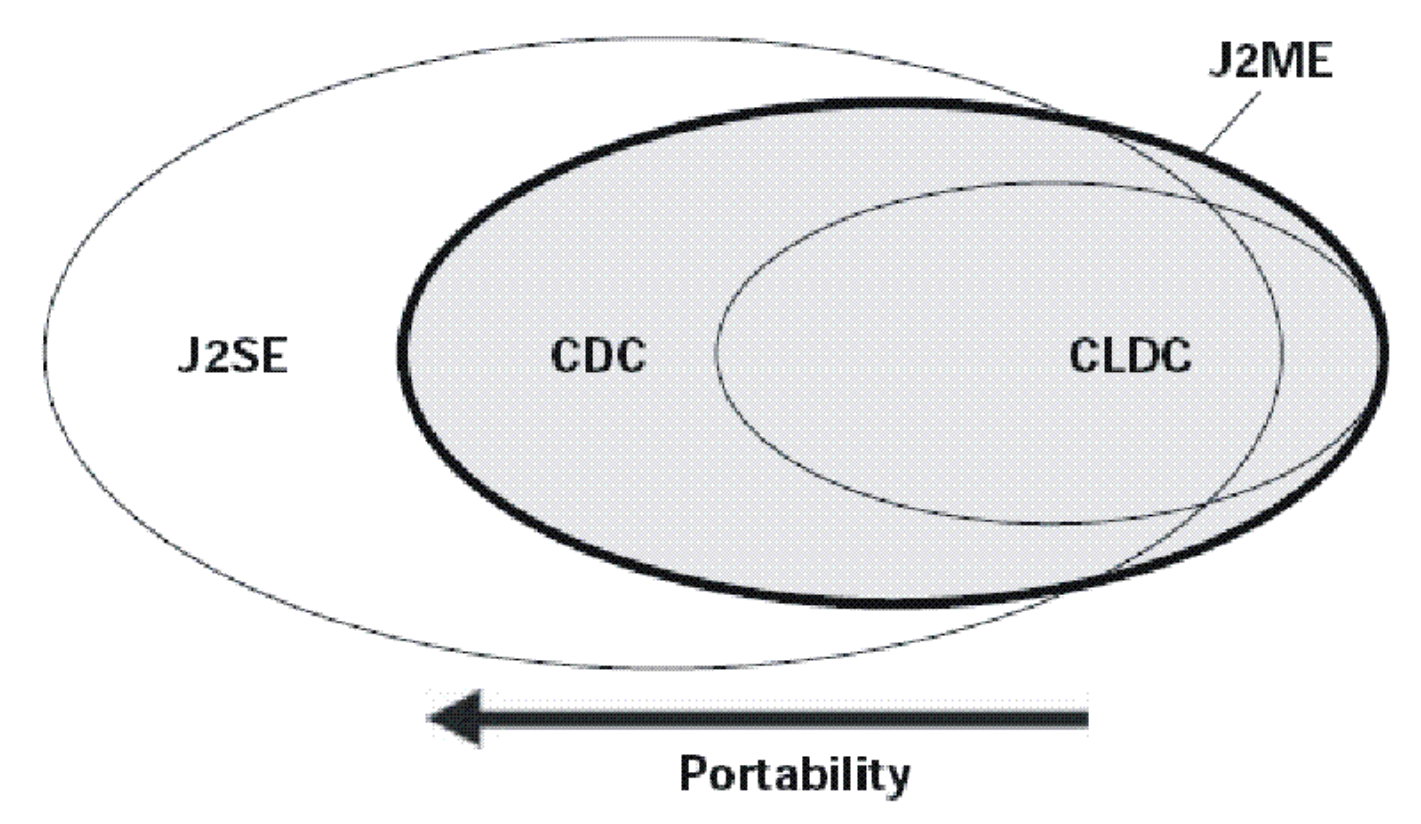
# Clases y Máquina Virtual

- Las diferentes tecnologías Java comparten un conjunto más o menos amplio de APIs básicas del lenguaje, agrupadas principalmente en los paquetes java.lang y java.io.
- J2ME contiene una mínima parte de las APIs de Java.
- J2ME usa 37 clases de la plataforma J2SE provenientes de los paquetes java.lang, java.io, java.util.
- J2ME a diferencia de J2SE utiliza una máquina virtual distinta de la clásica JVM denominada KVM.
- Esta KVM tiene unas restricciones que hacen que no posea todas las capacidades incluidas en la JVM. J2ME es un subconjunto de J2SE (excepto por el paquete javax.microedition).

# Configuraciones

- Las configuraciones, que son un conjunto de clases básicas orientadas a conformar el corazón de las implementaciones para dispositivos de características específicas.
- Existen 2 configuraciones definidas en J2ME:
- Connected Limited Device Configuration (CLDC) enfocada a dispositivos con restricciones de procesamiento y memoria. La VM (Virtual Machine) de la configuración CLDC se denomina KVM.
- Connected Device Configuration (CDC) enfocada a dispositivos con más recursos. La VM (Virtual Machine) de la CDC se denomina CVM.

# Relación entre J2SE y J2ME




- Java 2 Micro Edition: Java in Small Things, James White y David Hemphill, Manning Publications 2002

# KVM

- KVM es la Máquina Virtual más pequeña desarrollada por Sun.
- Su nombre KVM proviene de Kilobyte (haciendo referencia a la baja ocupación de memoria, entre 40Kb y 80Kb).
- Se trata de una implementación de Máquina Virtual reducida y especialmente orientada a dispositivos con bajas capacidades computacionales y de memoria.
- La KVM está escrita en lenguaje C, aproximadamente unas 24000 líneas de código.

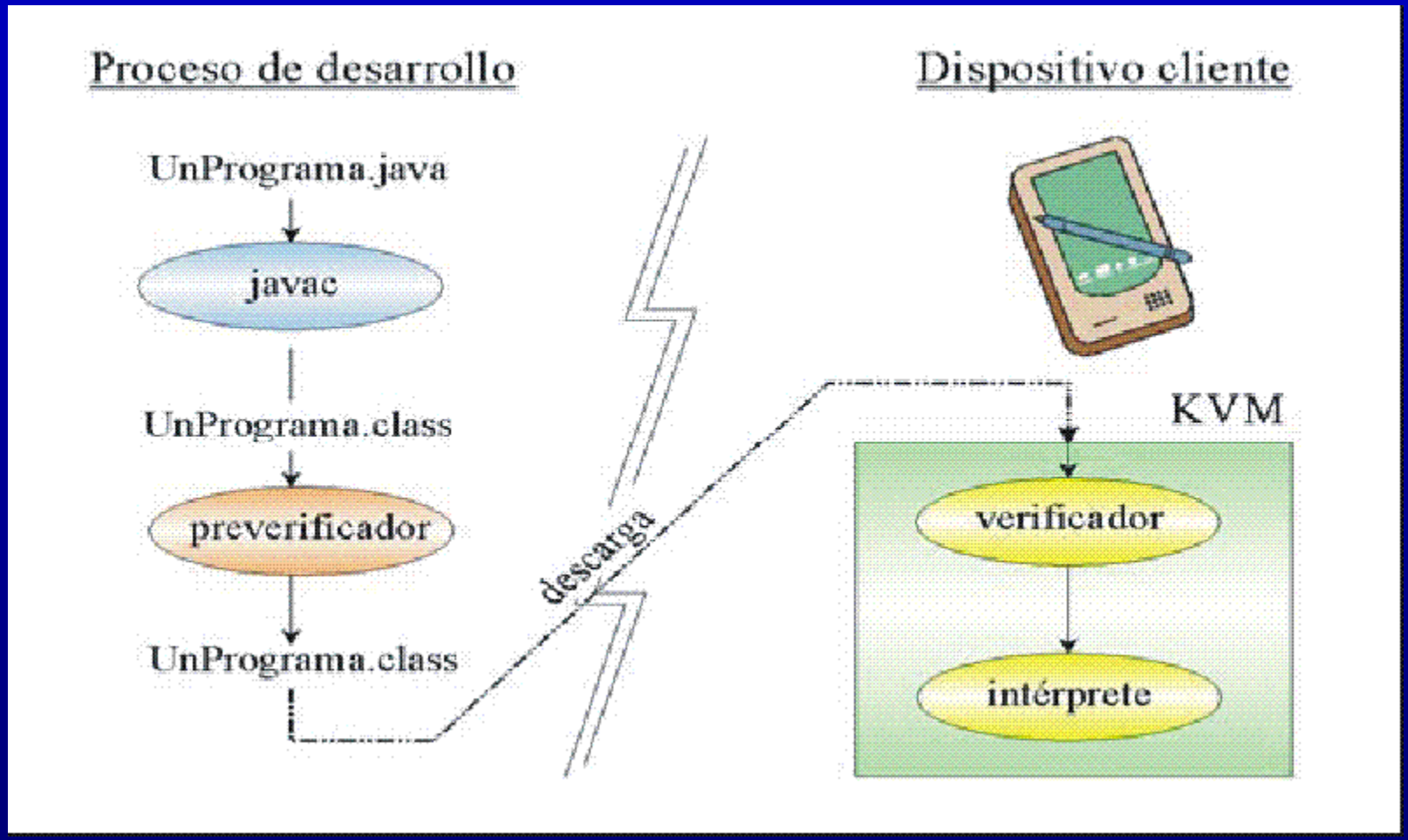
# KVM - Características

- Pequeña, con una carga de memoria entre los 40Kb y los 80 Kb, dependiendo de la plataforma y las opciones de compilación.
- Alta portabilidad.
- Modular.
- Lo más completa y rápida posible y sin sacrificar características para las que fue diseñada.

# KVM - Desventajas

- No hay soporte para tipos en coma flotante, no existen los tipos double ni float.
- No existe soporte para JNI (*Java Native Interface*).
- No existen cargadores de clases (*class loaders*) definidos por el usuario, sólo existen los predefinidos.
- No se permiten los grupos de hilos o hilos *daemon*.
- No existe la finalización de instancias de clases.
- No hay referencias débiles.
- Limitada capacidad para el manejo de excepciones
- Reflexión.

# Ejecución en Java



# CVM

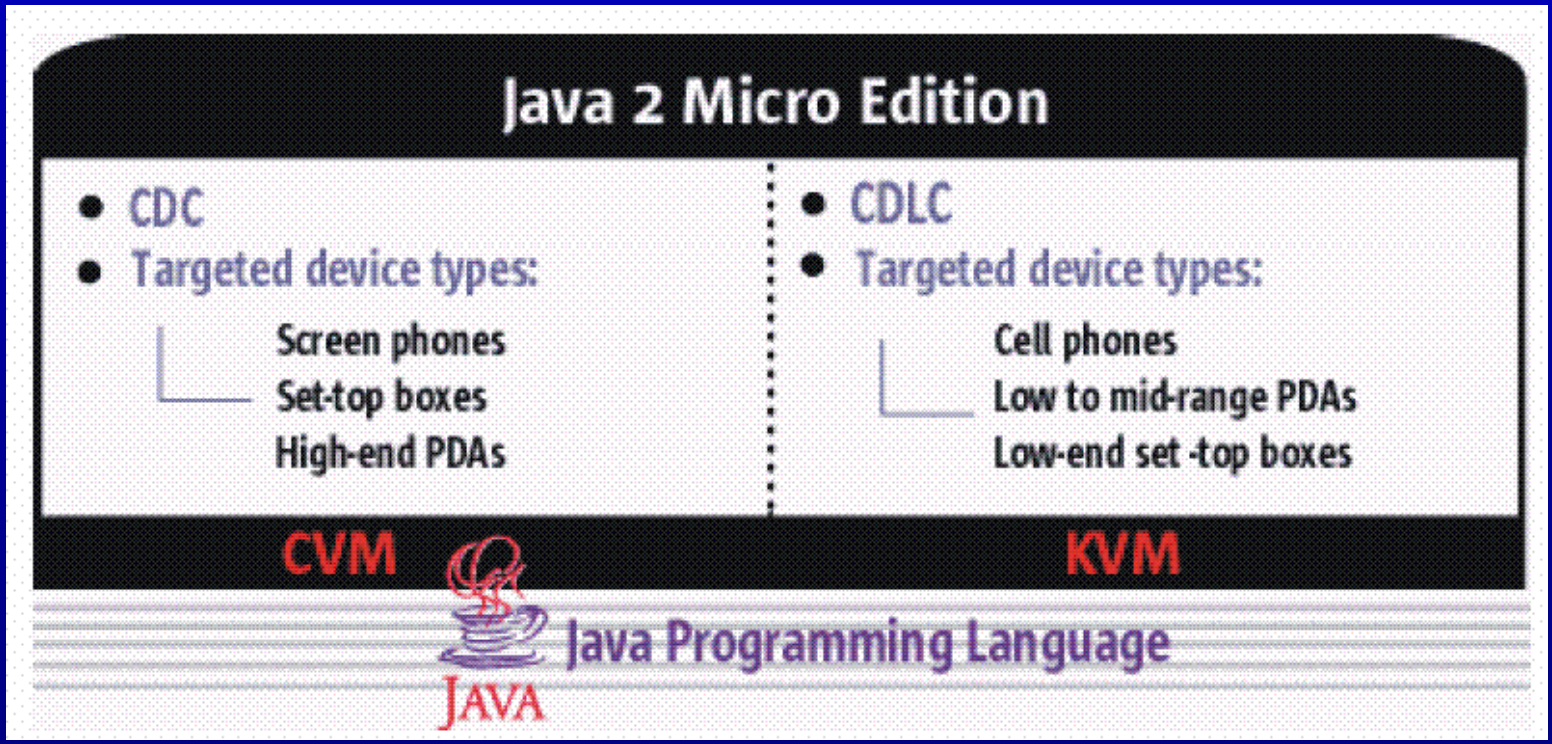
- La CVM (Compact Virtual Machine) ha sido tomada como Máquina Virtual Java de referencia para la configuración CDC y soporta las mismas características que la Máquina Virtual de J2SE.

- Está orientada a dispositivos electrónicos con procesadores de 32 bits de gama alta y en torno a 2Mb o más de memoria RAM.

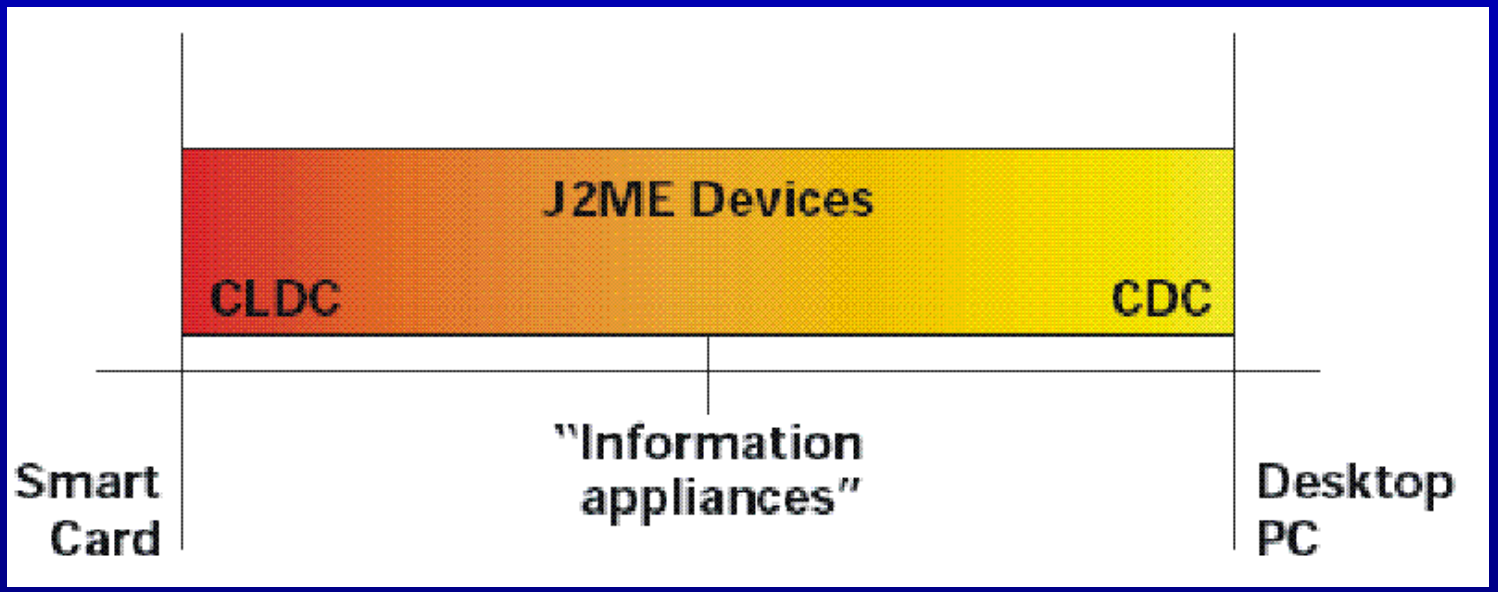
# CVM - Características

- Sistema de memoria avanzado.
- Tiempo de espera bajo para el recolector de basura.
- Separación completa de la VM del sistema de memoria.
- Recolector de basura modularizado.
- Portabilidad.
- Rápida sincronización.
- Ejecución de las clases Java fuera de la memoria de sólo lectura (ROM).
- Soporte nativo de hilos.
- Baja ocupación en memoria de las clases.
- Proporciona soporte e interfaces para servicios en Sistemas Operativos de Tiempo Real.
- Conversión de hilos Java a hilos nativos.
- Soporte para todas las características de Java2 v1.3 y librerías de seguridad, referencias débiles, Interfaz Nativa de Java (JNI), invocación remota de métodos (RMI), Interfaz de depuración de la Máquina Virtual (JVMDI).

# CDC y CLDC


Java 2 Micro Edition
CDC
Targeted device types:
Screen phones
Set-top boxes
High-end PDAs
CDLC
Targeted device types:
Cell phones
Low to mid-range PDAs
Low-end set -top boxes
CVM
KVM
Java Programming Language
JAVA

# Dispositivos Soportados - J2ME




- Java 2 Micro Edition: Java in Small Things, James White y David Hemphill, Manning Publications 2002

# CDC

- **CDC (*Connected Limited Configuration*) está orientada a dispositivos con cierta capacidad computacional y de memoria, por ejemplo, decodificadores de televisión digital, televisores con internet, algunos electrodomésticos y sistemas de navegación en automóviles.**

- **CDC usa una Máquina Virtual Java similar en sus características a una de J2SE, pero con limitaciones en el apartado gráfico y de memoria del dispositivo. Ésta Máquina Virtual se denomina como CVM (Compact Virtual Machine).**

# CDC - Características

- CDC está enfocada a dispositivos con las siguientes capacidades:

  Procesador de 32 bits.

  Disponer de 2 Mb o más de memoria total, incluyendo memoria RAM y ROM.

  Poseer la funcionalidad completa de la Máquina Virtual Java2.

  Conectividad a algún tipo de red.

# CDC - Paquetes

| NOMBRE DE PAQUETE CDC | DESCRIPCION |
|---|---|
| java.io | Clases de interfaces estándar de E/S. |
| java.lang | Clases básicas del lenguaje. |
| java.lang.ref | Clases de referencia. |
| java.lang.reflect | Clases e interfaces de reflection. |
| java.math | Paquete de matemáticas. |
| java.net | Clases e interfaces de red. |
| java.security | Clases e interfaces de seguridad. |
| 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 y comprimidos. |
| javax.microedition.io | Clases e interfaces para conexión genérica CDC. |

# CLDC

- CLDC (*Connected Limited Device Configuration*) está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, cómputo y memoria.
- Algunos de estos dispositivos son: teléfonos móviles, buscapersonas (pagers), PDAs, organizadores personales, etc.
- CLDC está orientado a dispositivos con ciertas restricciones, algunas de éstas restricciones vienen dadas por el uso de la KVM, necesaria al trabajar con la CLDC debido a su pequeño tamaño.

# CLDC - Requisitos

- Disponer entre 160 Kb y 512 Kb de memoria total disponible.
- Como mínimo se debe disponer de 128 Kb de memoria no volátil para la Máquina Virtual Java y las bibliotecas CLDC, y 32 Kb de memoria volátil para la Máquina Virtual en tiempo de ejecución.
- Procesador de 16 o 32 bits con al menos 25 Mhz de velocidad.
- Ofrecer bajo consumo, debido a que éstos dispositivos trabajan con suministro de energía limitado, normalmente baterías.
- Tener conexión a algún tipo de red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600 bps).

# CLDC - Características

- Un subconjunto del lenguaje Java y todas las restricciones de su Máquina Virtual (KVM).
- Un subconjunto de las bibliotecas Java del núcleo.
- Soporte para E/S básica.
- Soporte para acceso a redes.
- Seguridad.

# CLDC - Paquetes

| NOMBRE DE PAQUETE CLDC | DESCRIPCION |
|---|---|
| java.io | Clases y paquetes estándar de E/S. Subconjunto de J2SE. |
| java.lang | Clases e interfaces de la Máquina Virtual. Subconjunto de J2SE. |
| java.util | Clases, interfaces y utilidades estándar. Subconjunto de J2SE. |
| javax.microedition.io | Clases e interfaces de conexión genérica CLDC. |

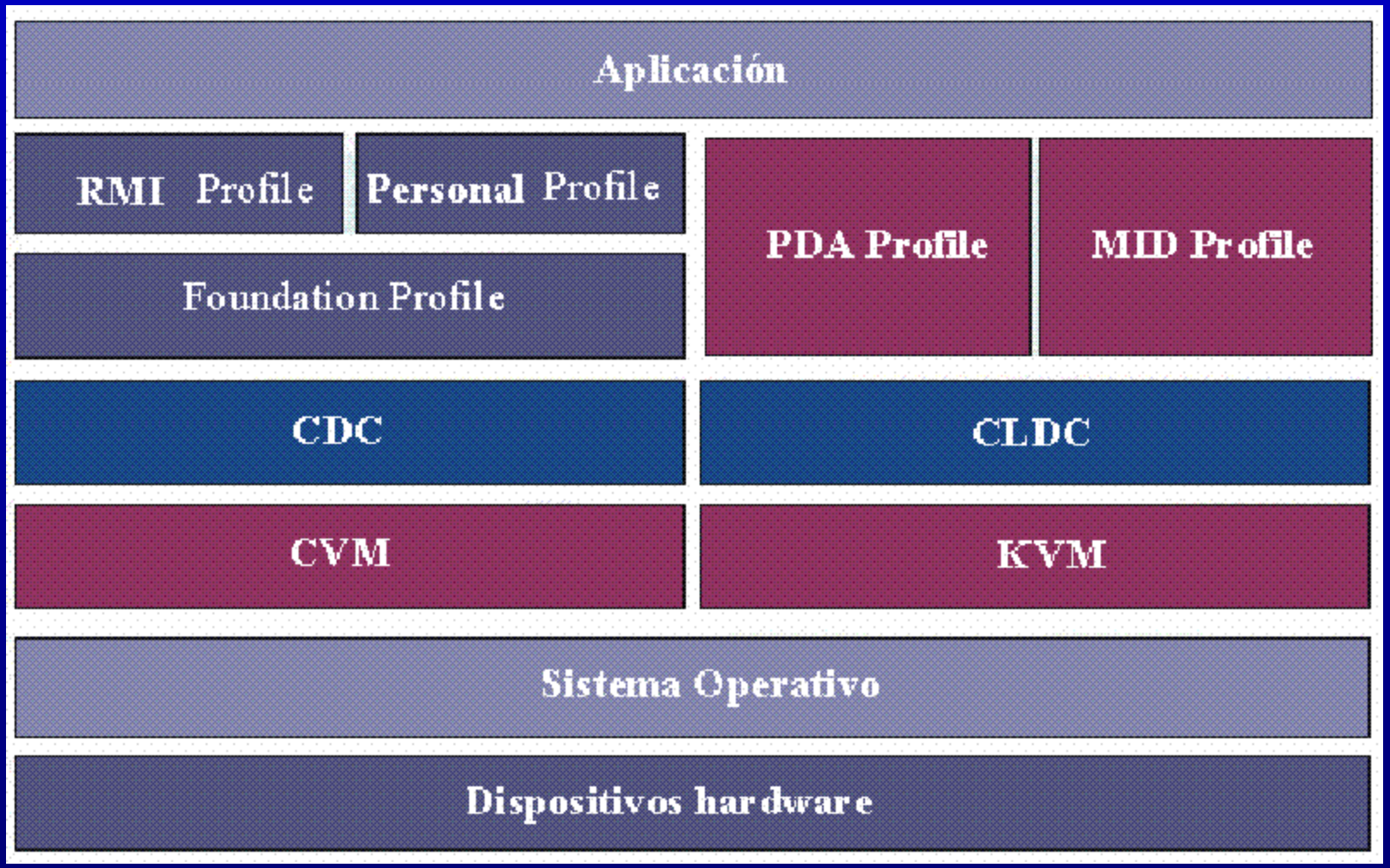
# Perfiles

- Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionan (electrodomésticos, teléfonos móviles, etc.) y el tipo de aplicaciones que se ejecutarán en ellos.
- Las librerías de la interfaz gráfica son un componente muy importante en la definición de un perfil. Aquí nos podemos encontrar grandes diferencias entre interfaces, desde el menú textual de los teléfonos móviles hasta los táctiles de los PDAs.
- El perfil establece unas APIs que definen las características de un dispositivo, mientras que la configuración hace lo propio con una familia de ellos. Esto hace que a la hora de construir una aplicación se cuente tanto con las APIs del perfil como de la configuración.

# Tipos de Perfiles

- **Para la configuración CDC tenemos los siguientes perfiles:**
  - ***Foundation Profile*.**
  - ***Personal Profile*.**
  - ***RMI Profile*.**
- **Para la configuración CLDC tenemos los siguientes:**
  - ***PDA Profile*.**
  - ***Mobile Information Device Profile* (MIDP).**

# Perfiles


Aplicación
RMI Profile
Personal Profile
PDA Profile
MID Profile
Foundation Profile
CDC
CLDC
CVM
KVM
Sistema Operativo
Dispositivos hardware

# Perfil CDC: Foundation Profile

- Este perfil define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica como, por ejemplo, decodificadores de televisión digital.
- Este perfil incluye gran parte de los paquetes de la J2SE, pero excluye totalmente los paquetes "java.awt" *Abstract Windows Toolkit* (AWT) y "java.swing" que conforman la interfaz gráfica de usuario (GUI) de J2SE.
- Si una aplicación requiriera una GUI, entonces sería necesario un perfil adicional.

# Perfil CDC: Foundation Profile

| PAQUETES DEL FOUNDATION PROFILE | DESCRIPCION |
|---|---|
| java. Lang | Soporte del lenguaje. |
| java.util | Añade 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 | Incluye códigos y certificados. |

# Perfil CDC: Personal Profile

- El *Personal Profile* es un subconjunto de la plataforma J2SE v1.3, y proporciona un entorno con un completo soporte gráfico AWT.
- El objetivo es el de dotar a la configuración CDC de una interfaz gráfica completa, con capacidades web y soporte de *applets* Java.
- Este perfil requiere una implementación del *Foundation Profile*.

# Perfil CDC: Personal Profile

| PAQUETES DEL PERSONAL PROFILE | DESCRIPCION |
|---|---|
| java. applet | Clases necesitadas para crear applets o que son usadas por ello. |
| 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. |
| java.awt.im | Clases e interfaces para definir métodos editores de entrada. |
| java.awt.im.spi | Interfaces que añaden el desarrollo de métodos editores de entrada. |
| java.awt.image | Clases para crear y modificar imágenes. |
| java.beans | Clases que soportan *JavaBeans*. |
| javax.midlet.xlet | Interfaces que usan el Personal Profile para la comunicación. |

# Perfil CDC: RMI Profile

- Este perfil requiere una implementación del *Foundation Profile* se construye encima de él.
- El perfil RMI soporta un subconjunto de las APIs J2SE v1.3 RMI. Algunas características de estas APIs se han eliminado del perfil RMI debido a las limitaciones de cómputo y memoria de los dispositivos.
- Las siguientes propiedades se han eliminado del J2SE RMI v1.3:
  - Java.rmi.server.disableHTTP.
  - Java.rmi.activation.port.
  - Java.rmi.loader.packagePrefix.
  - Java.rmi.registry.packagePrefix.
  - Java.rmi.server.packagePrefix.

# Perfil CLDC: PDA Profile

- Pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero (ratón o lápiz) y una resolución de al menos 20000 pixels (al menos 200x100 pixels) con un factor 2:1.

- Este perfil se encuentra en fase de definición.

# Perfil CLDC: MID Profile (MIDP)

- Al igual que CLDC fue la primera configuración definida para J2ME, MIDP fue el primer perfil definido para esta plataforma.

- Este perfil está orientado para dispositivos con las siguientes características:

  - Reducida capacidad computacional y de memoria.
  - Conectividad limitada (en torno a 9600 bps).
  - Capacidad gráfica muy reducida (mínimo un display de 96x54 pixels monocromo).
  - Entrada de datos alfanumérica 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.

# MIDP - Características

- A los tipos de dispositivos que se adaptan a estas características son:
  - Teléfonos móviles, buscapersonas (pagers) o PDA´s de gama baja con conectividad.
- El perfil MIDP establece las capacidades del dispositivo, por lo tanto, especifica las API´s relacionadas con:
  - La aplicación (semántica y control de la aplicación MIDP).
  - Interfaz de usuario.
  - Almacenamiento persistente.
  - Trabajo en red.
  - Temporizadores.

# MIDP - Paquetes

| PAQUETES DEL MIDP | DESCRIPCION |
|---|---|
| javax.microedition.lcdui | Clases e interfaces para GUIs. |
| javax.microedition.rms | 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 Máquina Virtual. |
| java.util | Clases e intefaces de utilidades estándar. |

# *MIDlets*

- **Las aplicaciones que realizamos utilizando MIDP reciben el nombre de *MIDlets* (por simpatía con *APPlets*).**
- **Por lo tanto, un MIDlet es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC.**
- **Actualmente, y con un punto de vista práctico, MIDP es el único perfil actualmente disponible.**

# 3) Descargar una aplicación en un dispositivo móvil

# Descarga de MIDlets

- Los dispositivos deben proporcionar mecanismos mediante los cuales podamos encontrar los *MIDlets* que deseemos descargar.
- En algunos casos, descargamos los MI*Dlets* a través de un navegador WAP o a través de una aplicación residente escrita específicamente para identificar *MIDlets*.
- Otros mecanismos como Bluetooth, cable serie, etc, pueden ser soportados por el dispositivo.
- El programa encargado de manejar la descarga y ciclo de vida de los *MIDlets* en el dispositivo se llama Gestor de Aplicaciones o AMS (*Application Management Software*).

# *MIDP y MIDlets*

- **Un dispositivo que posea la especificación MIDP debe ser capaz de:**
  - **Localizar archivos JAD vinculados a un *MIDlet* en la red.**
  - **Descargar el MIDlet y el archivo JAD al dispositivo desde un servidor usando el protocolo HTTP 1.1 u otro que posea su funcionalidad.**
  - **Enviar el nombre de usuario y contraseña cuando se produzca una respuesta HTTP por parte del servidor 401 (*Unauthorized*) o 407 (*Proxy Authentication Required*).**
  - **Instalar el *MIDlet* en el dispositivo.**
  - **Ejecutar *MIDlets*.**
  - **Permitir al usuario borrar *MIDlets* instalados.**

# Localización de la Aplicación

- El descubrimiento de una aplicación es el proceso por el cual un usuario a través de su dispositivo localiza un *MIDlet*.
- El usuario debe ser capaz de ver la descripción del *MIDlet* a través de un enlace que, una vez seleccionado, inicializa la instalación del *MIDlet*.
- Si éste enlace se refiere a un archivo JAR, el archivo y su URL son enviados al AMS del dispositivo para empezar el proceso de instalación.

# *Instalación de MIDlets*

- La instalación de la aplicación es el proceso por el cual el *MIDlet* es descargado al dispositivo y puede ser utilizado por el usuario.
- Cuando existan múltiples *MIDlets* en la aplicación que deseamos descargar, el usuario debe ser avisado de que existen más de uno.
- Durante la instalación, el usuario debe ser informado del progreso de ésta y se le debe de dar la oportunidad de cancelarla.
- La interrupción de la instalación debe dejar al dispositivo con el mismo estado que cuando se inició ésta.

# Actualización de MIDlets

- La actualización se realiza cuando instalamos un *MIDlet* sobre un dispositivo que ya contenía una versión anterior de éste.
- El dispositivo debe ser capaz de informar al usuario cual es la versión de la aplicación que tiene instalada.
- Cuando comienza la actualización, el dispositivo debe informar si la versión que va a instalar es más nueva, más vieja o la misma de la ya instalada y debe obtener verificación por parte del usuario antes de continuar con el proceso.
- En cualquier caso, un *MIDlet* que no posea firma no debe de reemplazar de ninguna manera a otro que sí la tenga.

# Ejecución de MIDlets

- Cuando un usuario comienza a ejecutar un *MIDlet*, el dispositivo debe invocar a las clases CLDC y MIDP requeridas por la especificación MIDP.
- Si existen varios *MIDlets* presentes, la interfaz de usuario debe permitir al usuario seleccionar el *MIDlet* que desea ejecutar.
- Los dispositivos deben permitir al usuario eliminar *MIDlets*. Antes de eliminar una aplicación el usuario debe dar su confirmación. El dispositivo debería avisar al usuario si ocurriese alguna 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.

# 4) Instalación de Herramientas de Desarrollo

# Instalación de Java

- Es necesario instalar primero el J2SE (Java 2 Standard Edition) versión para Windows, la cual se descarga de:

  *http://java.sun.com/j2se/1.5.0/download.jsp*

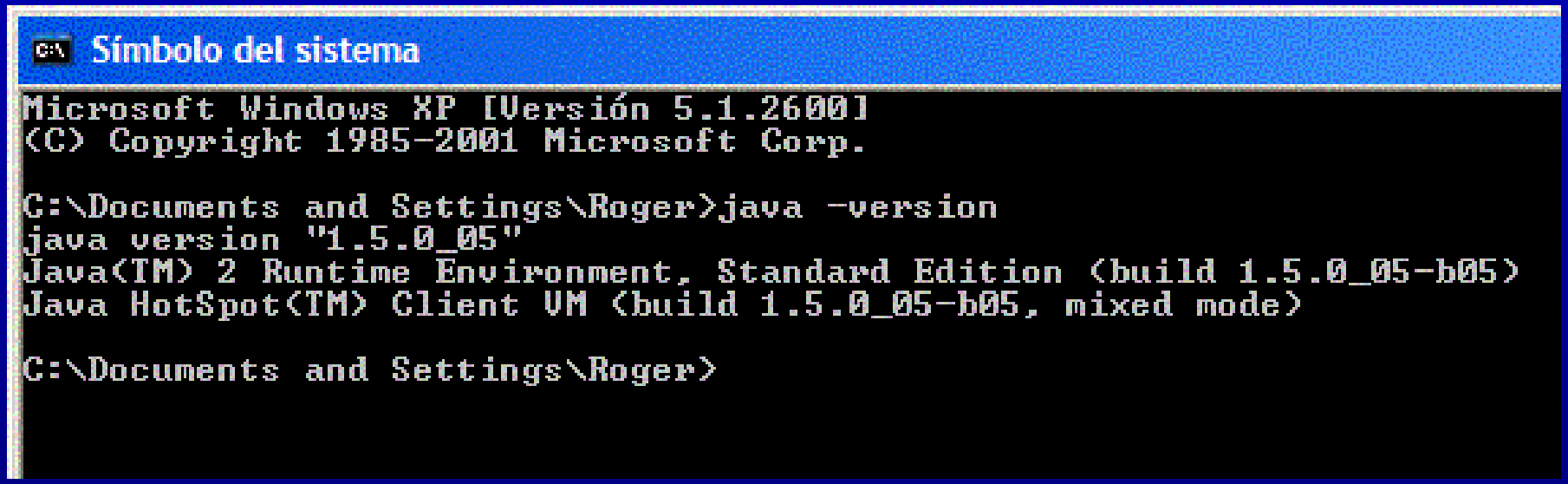
- En esta dirección se descarga el archivo denominado:

  *jdk-1_5_0-windows-i586.exe*

- Este archivo tiene un tamaño de 43.93 MB y se ejecuta para que instale la máquina virtual de Java.

# Instalación de Java

- Para probar si quedo instalado se teclea “java –version” en una terminal de Windows.



# Instalación de J2ME

- Una vez instalada la máquina virtual, se procede a descargar el J2ME, el cual se descarga de:

  *http://www.sun.com*

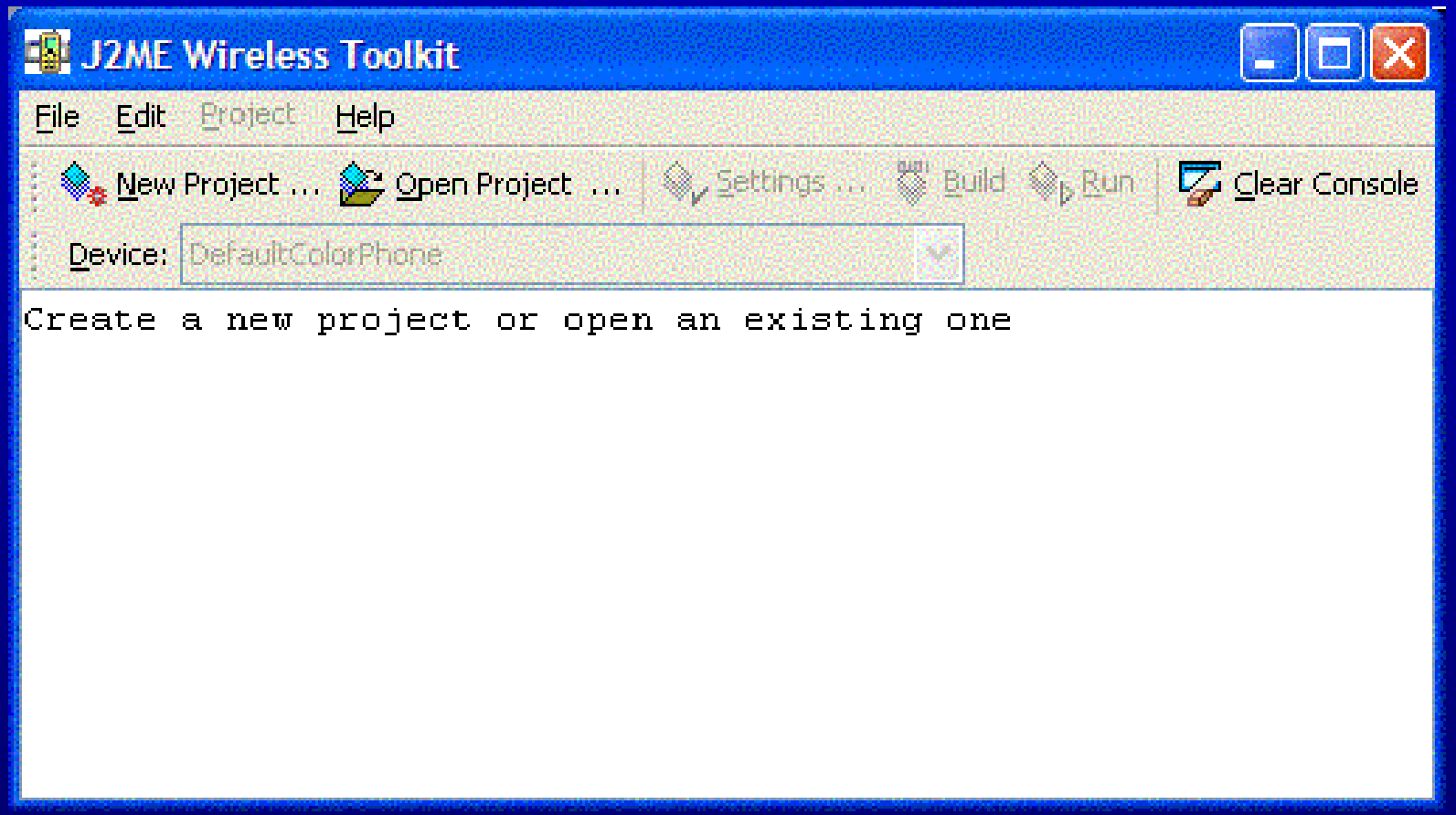
- En esta dirección se descarga el archivo denominado:

  *j2me_wireless_toolkit-2_2-windows.exe*

- Este archivo es la versión para Windows, se ejecuta y se instala el J2ME.

# Instalación de J2ME

- Para arrancar el J2ME se ejecuta “ktoolbar” del menu inicio de J2ME.



# *Entornos de Desarrollo J2ME*

- Existen otras herramientas que implementan algunas características, módulos, librerías, emuladores, etc. de *J2ME*.
- Algunas de estas herramientas son:
  - Sun One Studio Mobile Edition
  - Forte for Java
  - Jbuilder
  - VisualAge Microedition
  - CodeWarrior for Java
  - Nokia Developer's Suite for J2ME

# 5) Desarrollo de Aplicaciones

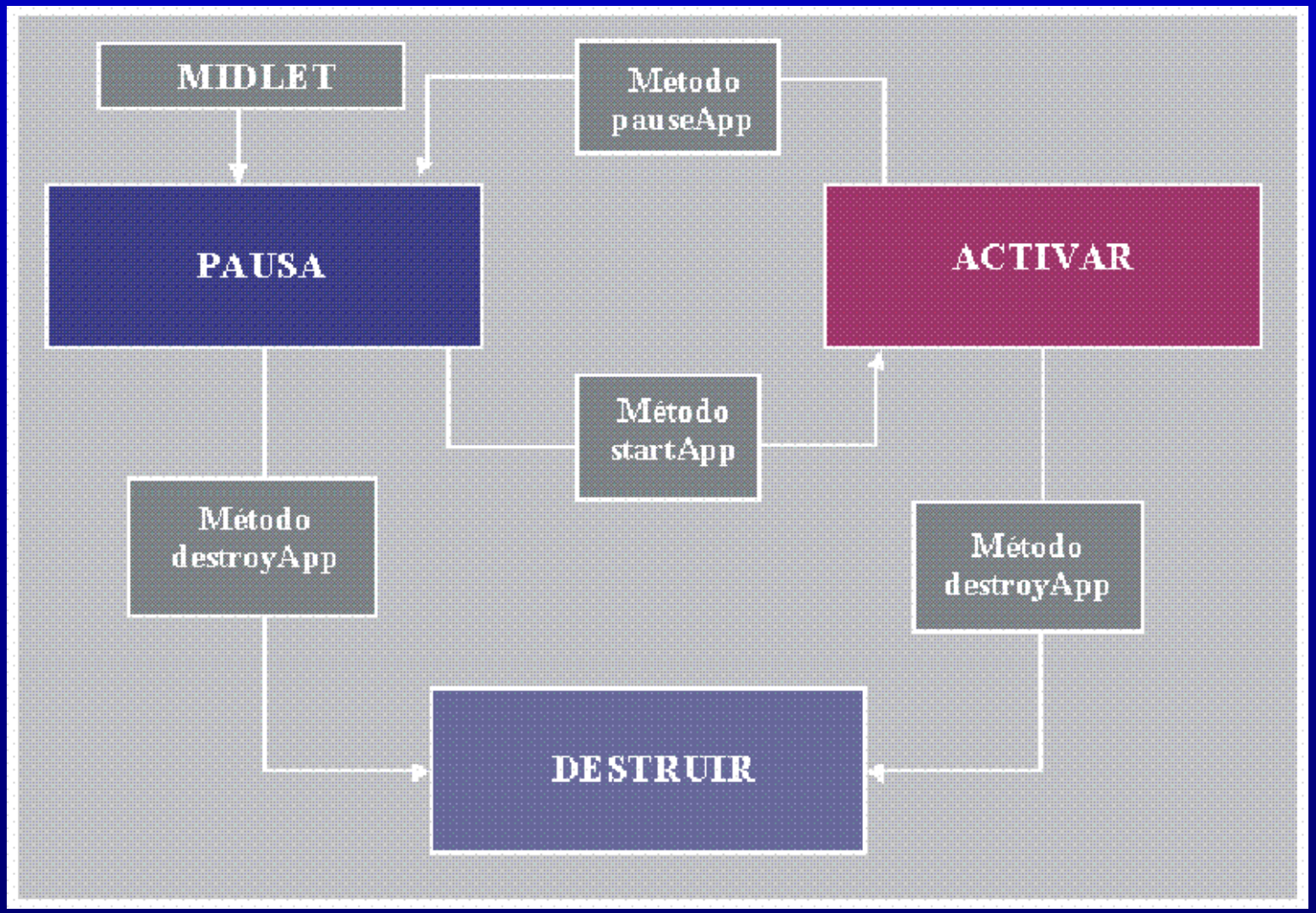
# *Creación de un MIDlet*

- **El proceso para llevar a cabo la creación de un *MIDlet* es el siguiente:**
  - **Crear el *MIDlet***
  - **Escribir el código y compilar.**
  - **Preverificar el código.**
  - **Empaquetar en *JAR* y crear el archivo *JAD*.**
  - **Ejecutar el emulador.**

# *Métodos Básicos*

- Las aplicaciones *MIDP* deben implementar los siguientes tres métodos:
  - startApp (): Este método es utilizado para la ejecución del *MIDlet*. Su objetivo es ejecutar la aplicación y solicitar recursos.
  - pauseApp (): Este método es invocado por el sistema para solicitarle al *MIDlet* que haga una pausa. Libera los recursos adquiridos por el método *starApp ()*.
  - destroyApp (*boolean unconditional*): Este método es llamado por el sistema antes de que sea destruido el *MIDlet*. Además libera todos los recursos adquiridos

# Métodos Básicos



# *Archivos .JAR*

- Los *MIDlets* son empaquetados en carpetas *.JAR* junto a imágenes, logos, para formar una *MIDlet suite*.
- Cada archivo *.JAR* lleva asociado un archivo *.JAD* (*Java Application Descriptor*, Descriptor de Aplicación Java) que es utilizado para gestionar la instalación.
- El archivo *.JAM* (*Java Application Manager*, Manejador de Aplicaciones Java) es el encargado de gestionar la descarga y la instalación de los *MIDlets*.

# Atributos de un MIDlet

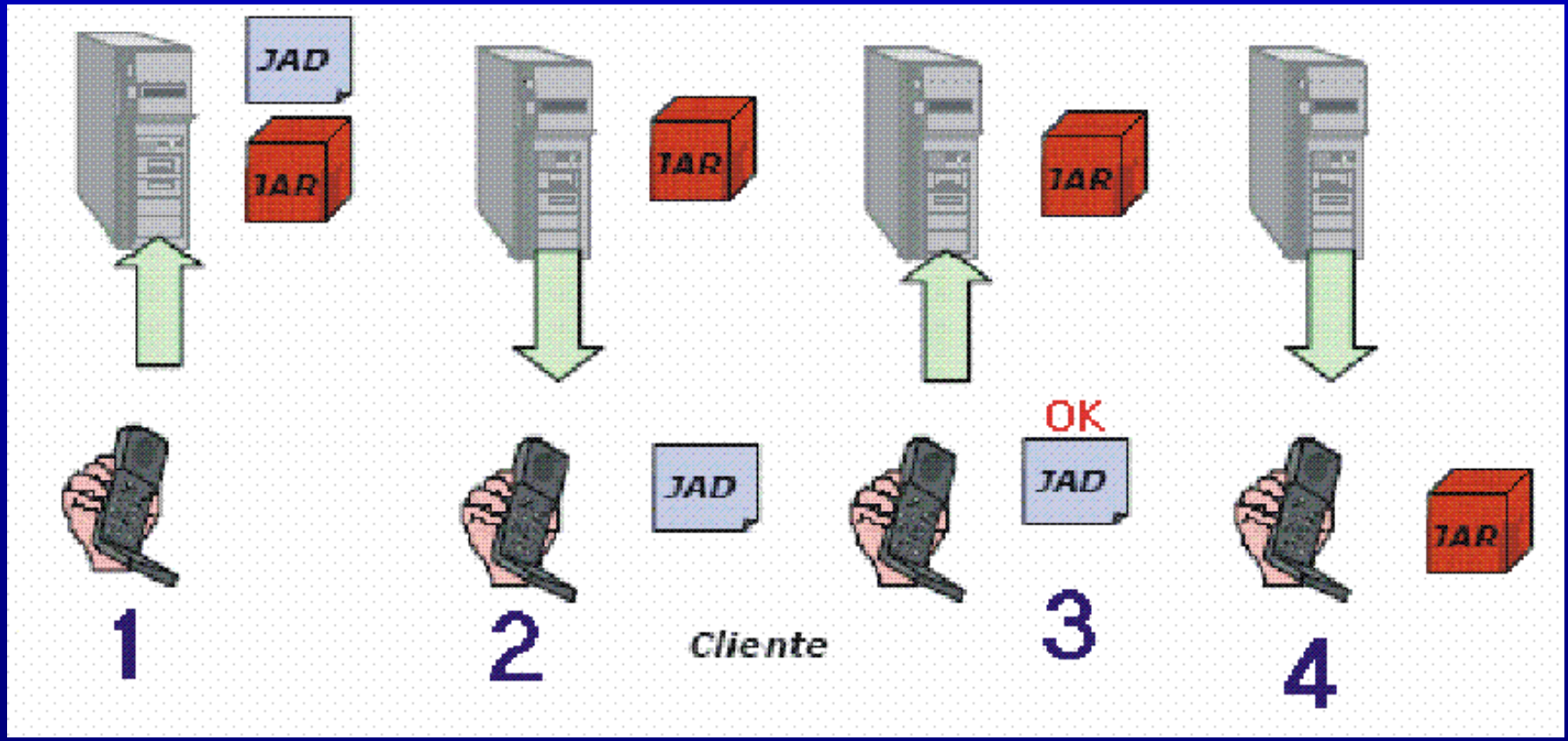
- Los *MIDlets* tienen atributos obligatorios y opcionales

| ATRIBUTOS OBLIGATORIOS | ATRIBUTOS OPCIONALES |
|---|---|
| MIDlet-Name | MIDlet-Description |
| MIDlet-Versión | MIDlet-Icon |
| MIDlet-Vendor | MIDlet-Info-URL |
| MIDlet-<n> (name, icon, class) | MIDlet-Data-Size |
| MicroEdition-Profile | |
| MicroEdition-Configuration | |

# *Instalar un MIDlet en un celular*

- **Para poder instalar un *MIDlet* en un teléfono celular del cliente se siguen los siguientes pasos:**
  - ***Paso 1:*** **El Cliente pide una el fichero *JAD*.**
  - ***Paso 2:*** **El fichero *JAD* es descargado al Cliente.**
  - ***Paso 3:*** **El *JAM* verifica el fichero *JAD*.**
  - ***Paso 4:*** **Se descarga la *MIDlet* suite al Cliente.**

# Instalar un MIDlet en un celular



# “hola Mundo” en J2ME

- Se arranca J2ME (ejecutando “KToolBar”).
- Se crea un proyecto nuevo (“New Project”).
- Se teclea el código fuente (“hola.java”) en un editor.
- Se guarda el código fuente en “C:\WTK22\apps\hola\src”.
- Se compila el proyecto (“Build”).
- Por último se ejecuta (“Run”).
- Se observará en el simulador del J2ME la aplicación. Para iniciar la aplicación se hace click en “Launch”.

# “hola Mundo”

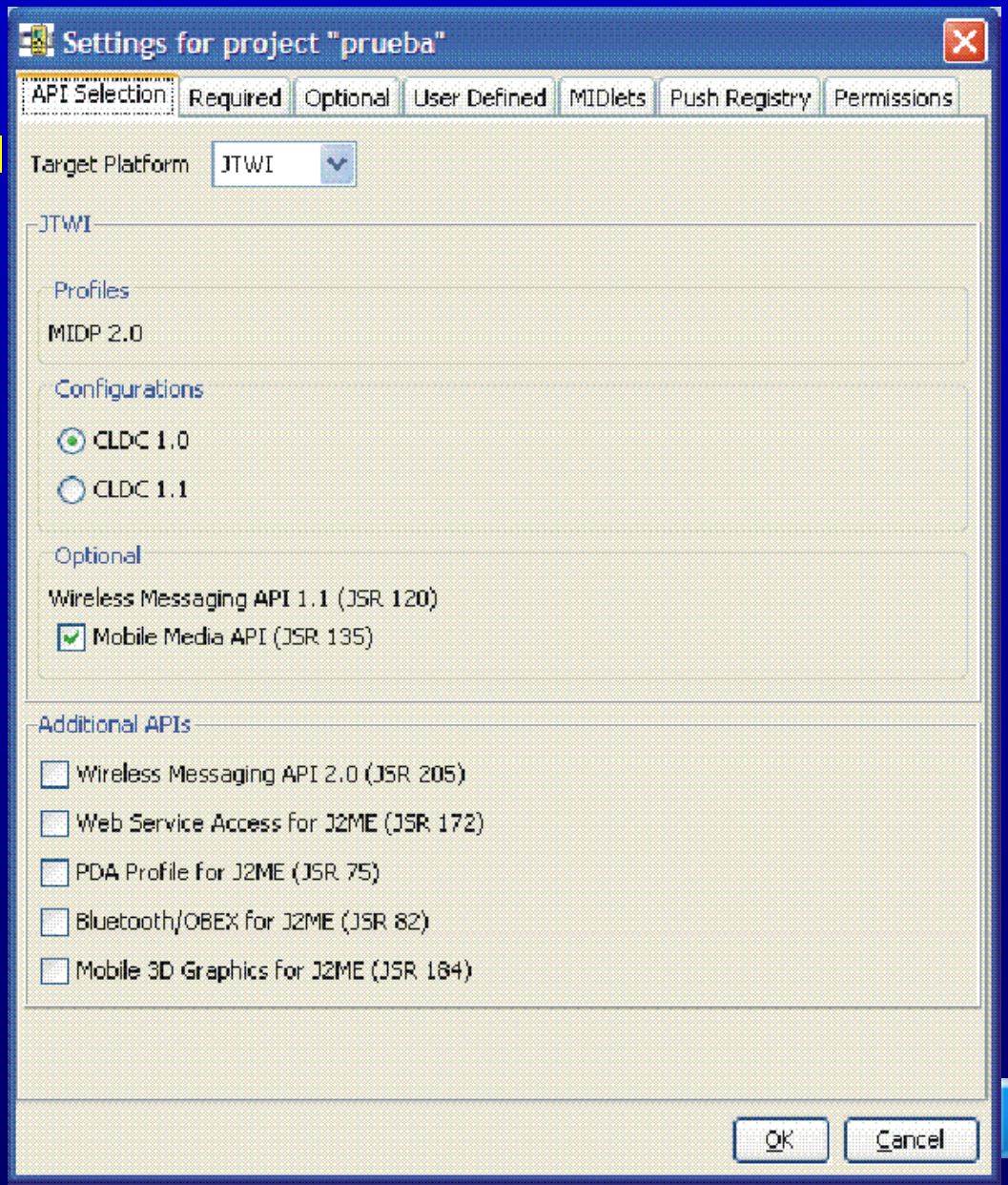
- Programa “hola Mundo” corriendo en el simulador del J2ME con el “DefaultColorPhone”.

# “hola Mundo” en un Celular

- Dentro del J2ME se seleccionan las características con que cuenta el celular donde se descargará el programa (“Settings”).
- Se genera el archivo “hola.JAR” por medio del menú “SettingsProject / Package / Create Package”.
- Este archivo se genera en el directorio “C:\WTK22\apps\hola\bin”.
- Se envía el archivo “hola.JAR” al celular por medio de algún medio (infrarrojo, USB, Bluetooth ó WiFi).
- Se ejecuta la aplicación en el celular.

# “hola Mundo” en un Celular

- Pantalla “settings” del J2ME



# “hola Mundo” en un Celular

- **Existen varias API´s, las cuales son soportadas por algunos celulares, esto depende del modelo y del fabricante. Las mas conocidas son las siguientes:**

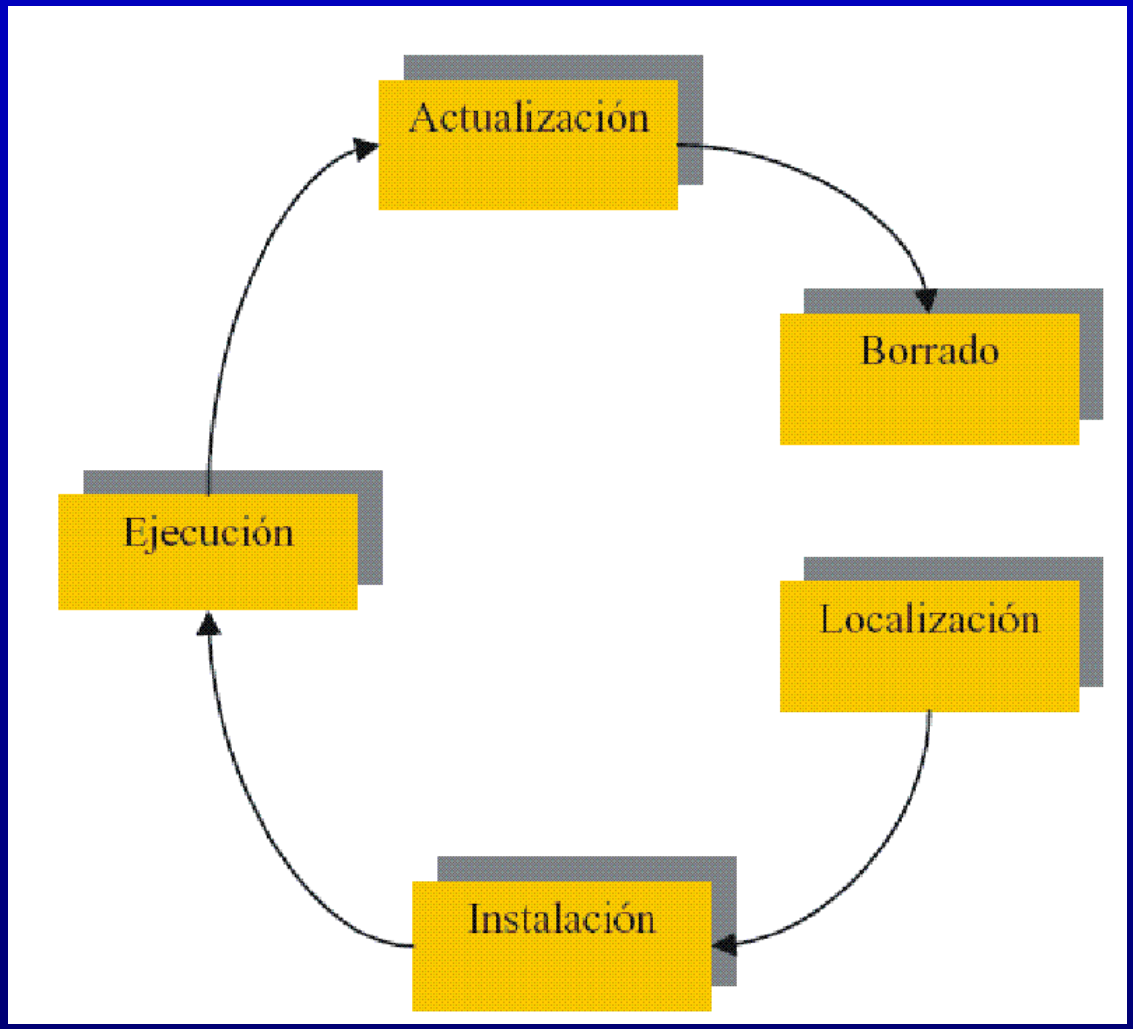
  **Connected Limited Device Configuration (CLDC) 1.1 (JSR 139)**
  **Mobile Information Device Profile (MIDP) 2.0 (JSR 118)**
  **Java Technology for the Wireless Industry (JTWI) 1.0 (JSR 185)**
  **Wireless Messaging API (WMA) 2.0 (JSR 205)**
  **Mobile Media API (MMAPI) 1.1 (JSR 135)**
  **PDA Optional Packages for the J2ME Platform (JSR 75)**
  **Java APIs for Bluetooth (JSR 82)**
  **J2ME Web Services Specification (JSR 172)**
  **Mobile 3D Graphics API for J2ME (JSR 184)**

# Celular Sendo X

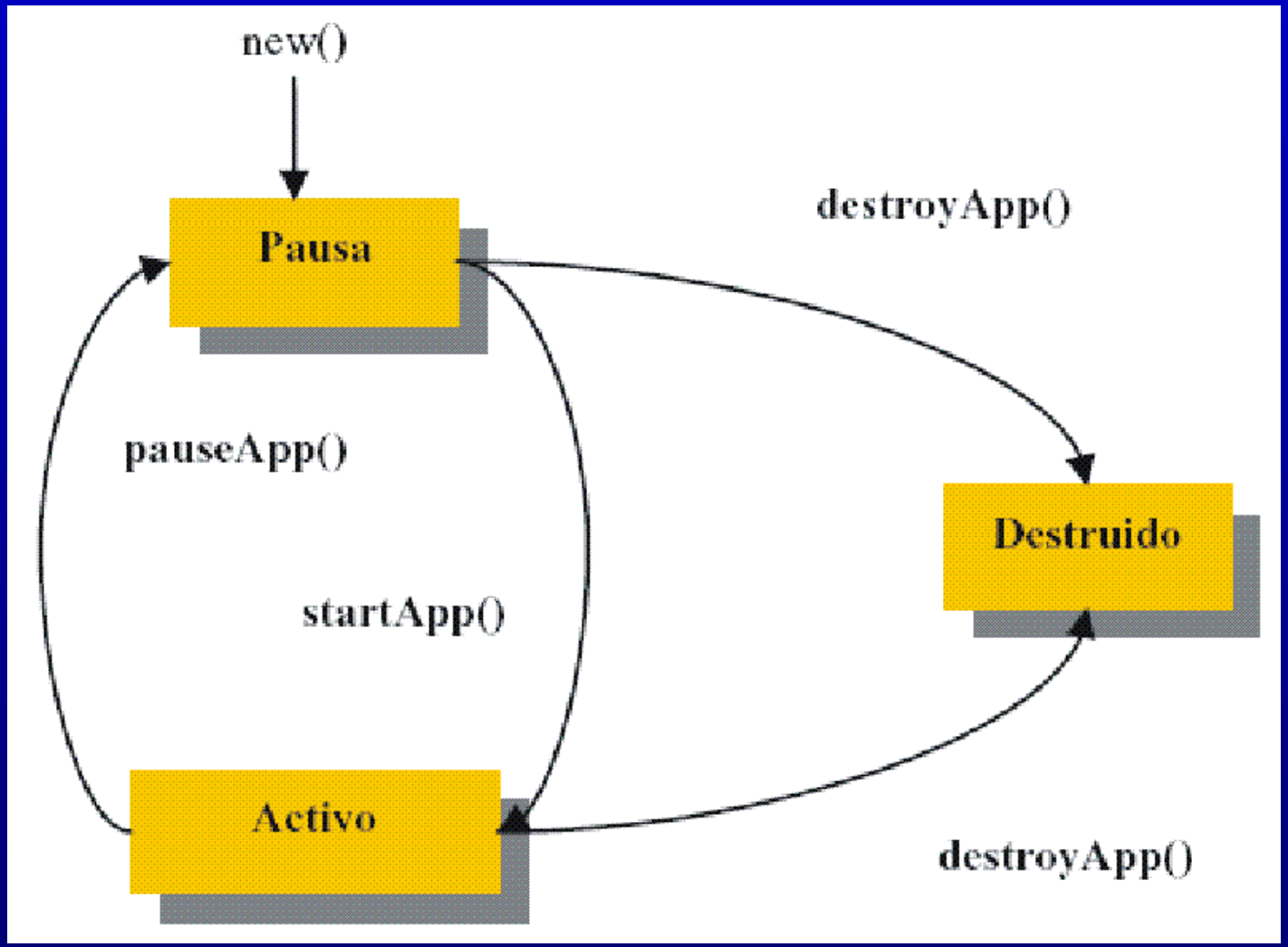
# Celular Sendo X

# 6) MIDlets

# Ciclo de vida de un MIDlet



# Estados de un MIDlet



# Estructura de un MIDlet

- Los *MIDlets*, al igual que los *applets* carecen de la función main(). Aunque existiese, el gestor de aplicaciones la ignoraría por completo.
- Un *MIDlet* tampoco puede realizar una llamada a System.exit(). Una llamada a este método lanzaría la excepción SecurityException.

# Estructura de un MIDlet

```
import javax.microedition.midlet.*;

public class prueba extends MIDlet {
        public prueba() {
                // Éste es el constructor de clase
                // Aquí debemos inicializar nuestras variables
        }
        public void startApp(){
                // Aquí incluiremos el código que queremos que el
                // MIDlet ejecute cuándo se active.
        }
        public void pauseApp(){
                // Aquí incluiremos el código que queremos que el
                // MIDlet ejecute cuándo entre en el estado de pausa (Opcional)
        }
        public void destroyApp(boolean unconditional){
                // 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)
        }
}
```

# “Hola Mundo” en J2ME

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class hola extends MIDlet {
        private Display pantalla;
        private Form formulario = null;
        public hola() {
                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();
        }
}
```

# CLDC - Clases Heredadas

- CLDC proporciona un conjunto de clases heredadas de la plataforma J2SE.
- En total, usa unas 37 clases provenientes de los paquetes java.lang, java.util y java.io.
- Cada una de estas clases debe ser idéntica o ser un subconjunto de la correspondiente clase de J2SE.
- Tanto los métodos como la semántica de cada clase deben permanecer invariables.

# CLDC - Clases Heredadas

- **Clases de datos heredadas de J2SE**

**Clases de Datos**
**(Subconjunto de java.lang)**

java.lang.Boolean
java.lang.Byte
java.lang.Character
java.lang.Integer
java.lang.Long
java.lang.Short
java.lang.Boolean
java.lang.Byte
java.lang.Character
java.lang.Integer
java.lang.Long
java.lang.Short

# CLDC - Clases Heredadas

- **Clases de datos heredadas de J2SE**

**Clases de Utilidades**
**(Subconjunto de java.util)**

java.util.Calendar
java.util.Date
java.util.Enumeration
java.util.Hashtable
java.util.Random
java.util.Stack
java.util.TimeZone
java.util.Vector

# CLDC - Clases Heredadas

- **Clases de datos heredadas de J2SE**

**Clases de E/S**
**(Subconjunto de java.io)**

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

# CLDC - Clases Propias

- La plataforma J2SE contiene a los paquetes java.io y java.net encargados de las operaciones de E/S, debido a las limitaciones de memoria de CLDC no es posible incluir dentro de él a todas las clases de estos paquetes.
- No se han incluido tampoco las clases del paquete java.net, basado en comunicaciones TCP/IP ya que los dispositivos CLDC no tienen por qué basarse en este protocolo de comunicación.
- Para suplir estas "carencias" CLDC posee un conjunto de clases más genérico para la E/S y la conexión en red que recibe el nombre de "*Generic Connection Framework*". Estas clases están incluidas en el paquete javax.microedition.io

# CLDC - Clases Propias

- **Clases e interfaces incluidos en el paquete javax.microedition.io**

| Clase | Descripción |
|---|---|
| Connector | Clase genérica que puede crear cualquier tipo de conexión. |
| Connection | Interfaz que define el tipo de conexión más genérica. |
| InputConnection | Interfaz que define una conexión de *streams* de entrada. |
| OutputConnection | Interfaz que define una conexión de *streams* de salida. |
| StreamConnection | Interfaz que define una conexión basada en *streams*. |
| ContentConnection | Extensión a StreamConnection para trabajar con datos. |
| Datagram | Interfaz genérico de datagramas. |
| DatagramConnection | Interfaz que define una conexión basada en datagramas. |
| StreamConnectionNotifier | Interfaz que notifica una conexión. Permite crear una conexión en el lado del servidor. |

# MIDP - Paquetes

| PAQUETES DEL MIDP | DESCRIPCION |
|---|---|
| javax.microedition.lcdui | Clases e interfaces para GUIs. |
| javax.microedition.rms | 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 Máquina Virtual. |
| java.util | Clases e intefaces de utilidades estándar. |

# 7) Interfaces de Usuario

# Interfaces – Alto Nivel

- En la Interfaz de Alto Nivel se usan componentes tales como botones, cajas de texto, formularios, etc.
- Estos elementos son implementados por cada dispositivo y la finalidad de usar las API´s de alto nivel es su portabilidad.
- Al usar estos elementos, perdemos el control del aspecto de nuestra aplicación ya que la estética de estos componentes depende exclusivamente del dispositivo donde se ejecute.
- En cambio, usando estas API´s de alto nivel ganaremos un alto grado de portabilidad de la misma aplicación entre distintos dispositivos.

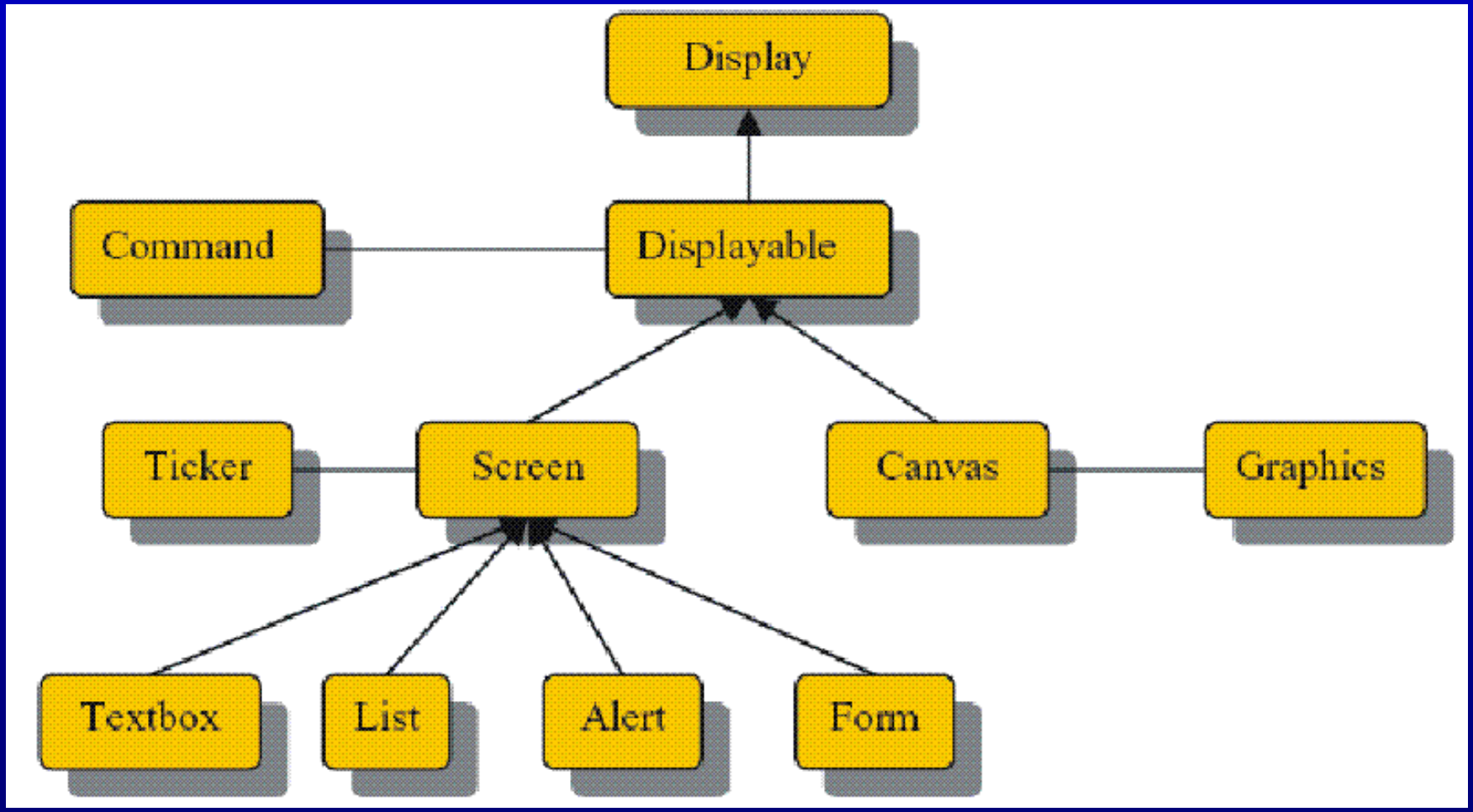
# Interfaces – Bajo Nivel

- Al crear una aplicación usando las API´s de bajo nivel, tendremos un control total de lo que aparecerá por pantalla.
- Estas API´s nos darán un control completo sobre los recursos del dispositivo y podremos controlar eventos de bajo nivel como, por ejemplo, el rastreo de pulsaciones de teclas.
- Generalmente, estas API´s se utilizan para la creación de juegos donde el control sobre lo que aparece por pantalla y las acciones del usuario juegan un papel fundamental.
- El paquete javax.microedition.lcdui incluye las clases necesarias para crear interfaces de usuario, tanto de alto nivel como de bajo nivel.

# clase Display

- La clase Display representa el manejador de la pantalla y los dispositivos de entrada.
- Todo *MIDlet* debe poseer por lo menos un objeto Display.
- En este objeto Display podemos incluir tantos objetos Displayable como queramos.
- La clase Display puede obtener información sobre las características de la pantalla del dispositivo donde se ejecute el *MIDlet*, además de ser capaz de mostrar los objetos que componen nuestras interfaces.

# clase Display



# Métodos de la clase Display

| Métodos | Descripción |
|---|---|
| void callSerially(Runnable r) | Retrasa la ejecución del método run() del objeto *r* para no interferir con los eventos de usuario. |
| boolean flashBacklight(int duracion) | Provoca un efecto de *flash* en la pantalla. |
| int getBestImageHeight(int imagen) | Devuelve el mejor alto de imagen para un tipo dado. |
| int getBestImageWidth(int imagen) | Devuelve el mejor ancho de imagen para un tipo dado. |
| int getBorderStyle(bolean luminosidad) | Devuelve el estilo de borde actual. |
| int getColor(int color) | Devuelve un color basado en el parámetro pasado. |
| Displayable getCurrent() | Devuelve la pantalla actual. |
| static Display getDisplay(MIDlet m) | Devuelve una referencia a la pantalla del MIDlet *m*. |
| boolean isColor() | Devuelve *true* o *false* si la pantalla es de color o b/n. |
| int numAlphaLevels() | Devuelve el número de niveles *alpha* soportados. |
| int numColors() | Devuelve el número de colores aceptados por el MID. |
| void setCurrent(Alert a, Displayable d) | Establece la pantalla *d* despues de la alerta *a* |
| void setCurrent(Displayable d) | Establece la pantalla actual |
| void setCurrent(Item item) | Establece la pantalla en la zona dónde se encuentre el *item* |
| boolean vibrate(int duracion) | Realiza la operación de vibración del dispositivo. |

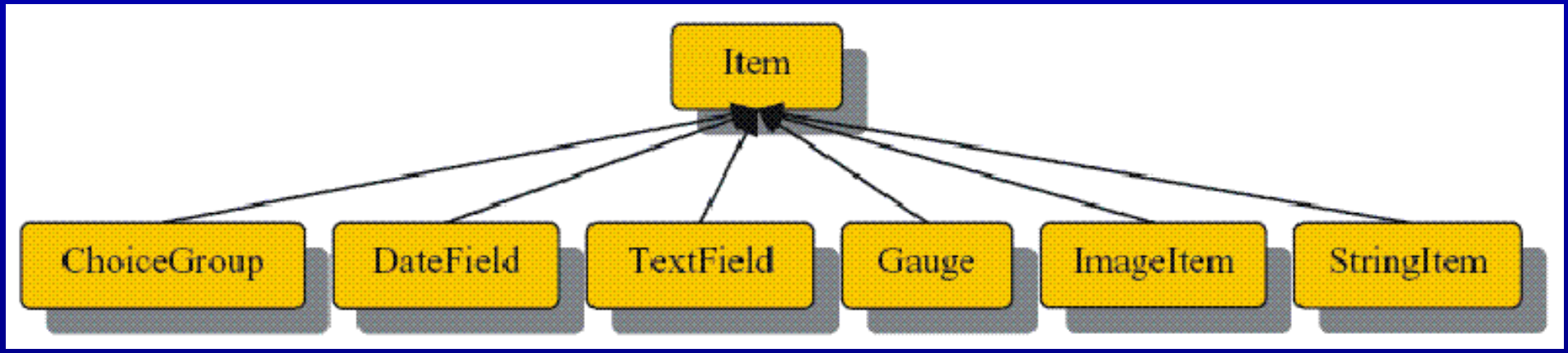
# Creación de una pantalla

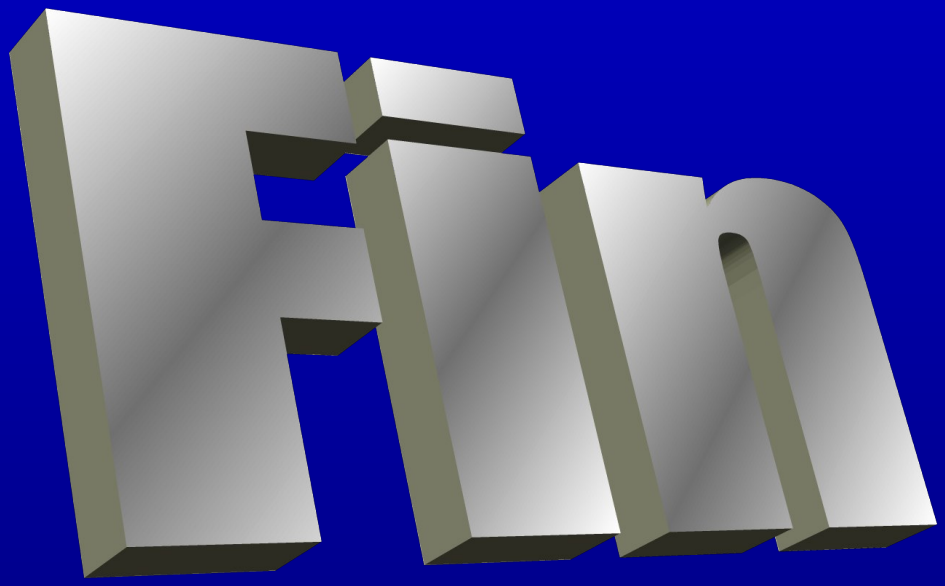
- Todo *MIDlet* debe poseer al menos una instancia del objeto Display. Para obtenerla se utiliza:
- Display pantalla = Display.getDisplay(this)
- La llamada a este método se realiza dentro del constructor del *MIDlet*, de esta forma se asegura que el objeto Display esté disponible durante toda la ejecución del programa. Dentro del método startApp tendremos que hacer referencia a la pantalla que queramos que esté activa haciendo uso del método:
- setCurrent()
- Hay que tener en cuenta que cada vez que salimos del método pauseApp, entramos en el método startApp, por lo que la construcción de las pantallas y demás elementos que formarán parte de nuestro *MIDlet* la tendremos que hacer en el método constructor.

# Métodos de la clase Displayable

| Métodos | Descripción |
|---|---|
| void addComand(Command cmd) | Añade el Command *cmd*. |
| int getHeight() | Devuelve el alto de la pantalla. |
| Ticker getTicker() | Devuelve el T*icker* (cadena de texto que se desplaza) asignado a la pantalla. |
| String getTitle() | Devuelve el título de la pantalla. |
| int getWidth() | Devuelve el ancho de la pantalla. |
| bolean isShown() | Devuelve *true* si la pantalla está activa. |
| void removeCommand(Command cmd) | Elimina el Command *cmd*. |
| void setCommandListener(CommandListener l) | Establece un *listener* para la captura de eventos. |
| void setTicker(Ticker ticker) | Establece un *Ticker* a la pantalla. |
| void setTitle(String s) | Establece un título a la pantalla. |
| protected void sizeChanged(int w, int h) | El AMS llama a este método cuándo el área disponible para el objeto Displayable es modificada. |

# clase Item


Item
ChoiceGroup
DateField
TextField
Gauge
ImageItem
StringItem




*Unidad VII – Java para Dispositivos Móviles*