Introducción a Java Micro Edition (Java ME)

Florina Alménarez Mendoza
Celeste Campo
Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid
{florina, celeste}@it.uc3m.es

Parte de este material se basa en transparencias de Natividad Martínez Madrid (nati@it.uc3m.es)

Contexto

- **Objetivo**
  - Conocer la arquitectura de Java Micro Edition para desarrollar aplicaciones multi-plataforma para dispositivos móviles portables

- **Bibliografía**
  - ***Wireless Java Programming with Java 2 Micro Edition.*** Feng, Yu and Zhu, Jun. SAMS 2001 . L/D 004.438 JAVA FEN. Capítulo 2 y 3.
  - http://www.oracle.com/technetwork/java/javame/index.html
  - **Programming wireless devices with the Java 2 platform, micro edition:** J2ME Connected Limited Device Configuration (CLDC), Mobile Information Device Profile (MIDP). R. Riggs. Addison-Wesley, 2003.

Índice

- Introducción
- Arquitectura
  - Máquinas virtuales
  - Configuraciones
  - Perfiles
  - Paquetes opcionales
- MIDP/CLDC/KVM
  - CLDC/KVM

Introducción

- La plataforma Java ME es una colección de tecnologías y especificaciones para desarrollar aplicaciones
  - dispositivos con memoria, pantalla y capacidad de procesamiento limitadas
- Solución abierta que proporciona portabilidad de aplicaciones entre distintas plataformas móviles
  - Diferentes API's y VMs, pero el mismo lenguaje de programación, Java
- Estandarizado bajo el *Java Community Process* (**JCP**)
  - ***Java Specification Request* (JSR) 68**: J2ME *Platform Specification*
    - Arquitectura de la plataforma y actividades de estandarización
  - **JSR 185**: *Java Technology for Wireless Industry* (JTWI)
    - Específico para teléfonos móviles de siguiente generación
    - Cómo trabajan de forma conjunta varias tecnologías asociadas con MIDP para proporcionar una solución para la industria de servicios inalámbricos

Arquitectura

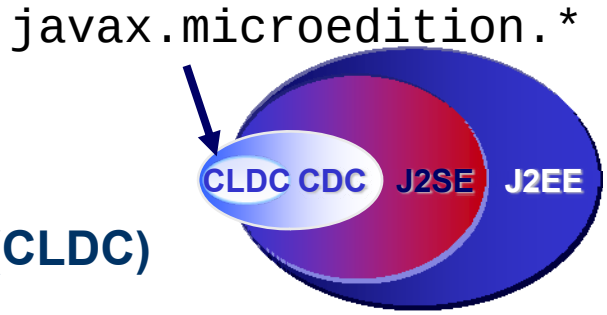
- Para conseguir flexibilidad y adaptación, J2ME se estructura en capas:
  - **Máquina virtual**
  - **Configuración**
    - Mínimo conjunto de clases disponibles
    - Engloba un segmento horizontal de mercado
  - **Perfiles**
    - Clases adicionales para un segmento vertical de mercado
  - **Paquetes opcionales**
    - APIs adicionales

Máquinas virtuales

- Una JVM
  - Interpreta código intermedio (bytecode) de los programas Java precompilados a código máquina ejecutable por la plataforma
  - Efectúa las llamadas pertinentes al sistema operativo
  - Observa las reglas de seguridad
- Dos VM definidas para J2ME:
  - **CVM**: Compact Virtual Machine, C Virtual Machine
    - orientada a dispositivos embebidos y electrónica de consumo (set-top box, TV digital, electrodomésticos,...)
    - misma funcionalidad que JVM con mejor uso de memoria (≈ 2MB)
  - **KVM**: “Kilo” Virtual Machine , K Virtual Machine
    - dispositivos con poca memoria, capacidad de proceso limitada y con conexión a red intermitente:
    - memoria mínima 128 KB, procesadores de 16 ó 32 bits RISC o CISC, ocupa entre 40 y 80 KB

Configuraciones

- ¿Qué es una configuración?
  - Mínimo conjunto de clases disponibles para un grupo de dispositivos. Los grupos se establecen según requisitos similares de memoria y procesamiento.
- ¿Qué define?
  - Características soportadas del lenguaje de programación Java.
  - Características soportadas por la Máquina Virtual Java.
  - Bibliotecas básicas de Java y APIs soportadas.
- Las configuraciones se especifican vía la iniciativa JCP que genera los correspondientes JSR
- Existen dos configuraciones actualmente:
  - **Connected Device Configuration (CDC)**
  - **Connected, Limited Device Configuration (CLDC)**



CDC
Connected Device Configuration

- Orientado a dispositivos con:
  - 512 KB de ROM
  - 256 KB de RAM
  - Conexión a red (fija)
  - Soporte completo a la especificación de JVM
  - Interfaz de usuario relativamente limitado
  - Basado en J2SE v1.3
- Especificado en **JSR 36 (CDC 1.0)** y **JSR 218 (CDC 1.1)**
- Ejemplos: Internet *screen phones*, DTV *set-top boxes* y sistemas telemáticos de automóviles
- Iniciativas anteriores: PersonalJava, JavaTV, JavaPhone

CDC
Librerías incluidas

Nombre de Paquete CDC	Descripción
`java.io`	Clases e interfaces estándar de E/S
`java.lang`	Clases básicas del lenguaje
`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
`javax.microedition.io`	Clases e interfaces para conexión genérica CDC

CLDC
Connected Limited Device Configuration

- Orientado a dispositivos con:
  - 192 KB a 512 KB de memoria disponible para Java
  - Procesador de 16 o 32 bits, velocidad 8-32 MHz
  - Limitaciones de consumo (baterías)
  - Conectividad a red (inalámbrica)
  - Restricciones importantes en el interfaz de usuario
- Especificado en **JSR 30 (CLDC 1.0)** y **JSR 139 (CLDC 1.1)**
- Especificación CLDC 1.0/1.1 disponible:
  - Sun proporciona una implementación de referencia de CLDC sobre KVM, para Linux, Windows y Solaris
  - Principales fabricantes de móviles la implementan en la mayoría de sus modelos (Nokia, Siemens, Samsung,...)

CLDC
Librerías incluidas

Nombre de paquete CLDC	Descripción
`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 J2SSE
`javax.microedition.io`	Clases e interfaces de conexión genérica CLDC

Perfiles

- Conjunto de clases Java que complementan una configuración para un conjunto específico de dispositivos (“segmento vertical”)
- ¿Qué definen?
  - APIs que controlan el ciclo de vida de la aplicación,
  - Interfaz de usuario, etc.
- Los perfiles permiten la portabilidad de aplicaciones J2ME entre diferentes dispositivos
- Se especifican vía la iniciativa JCP que genera los correspondientes JSR
- Un dispositivo puede soportar múltiples perfiles

Perfiles sobre CDC

- **Foundation Profile** (JSR 46, JSR 219):
  - Perfil básico para dispositivos sin interfaz gráfico.
- **Personal Basis Specification** (JSR 129):
  - Perfil gráfico para dispositivos con interfaz gráfico básico.
- **Personal Profile** (JSR 62, JSR 216):
  - Perfil gráfico basado en AWT (dispositivos con interfaz gráfico).
  - Evolución de Personal Java.

Perfiles sobre CLDC

- **Mobile Information Device Profile** (JSR 37, JSR 118):
  - Perfil para dispositivos inalámbricos: móviles, PDAs,…

- **Information Module Profile** (JSR 195):
  - Perfil para dispositivos con interfaz gráfica limitada: parquímetros, alarmas,…

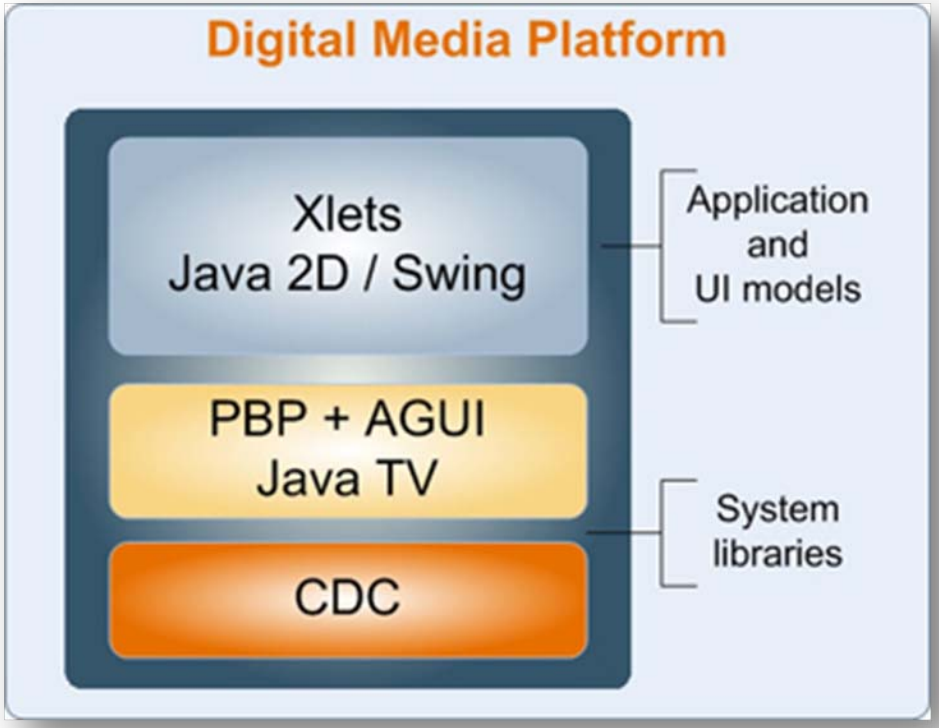
Paquetes opcionales

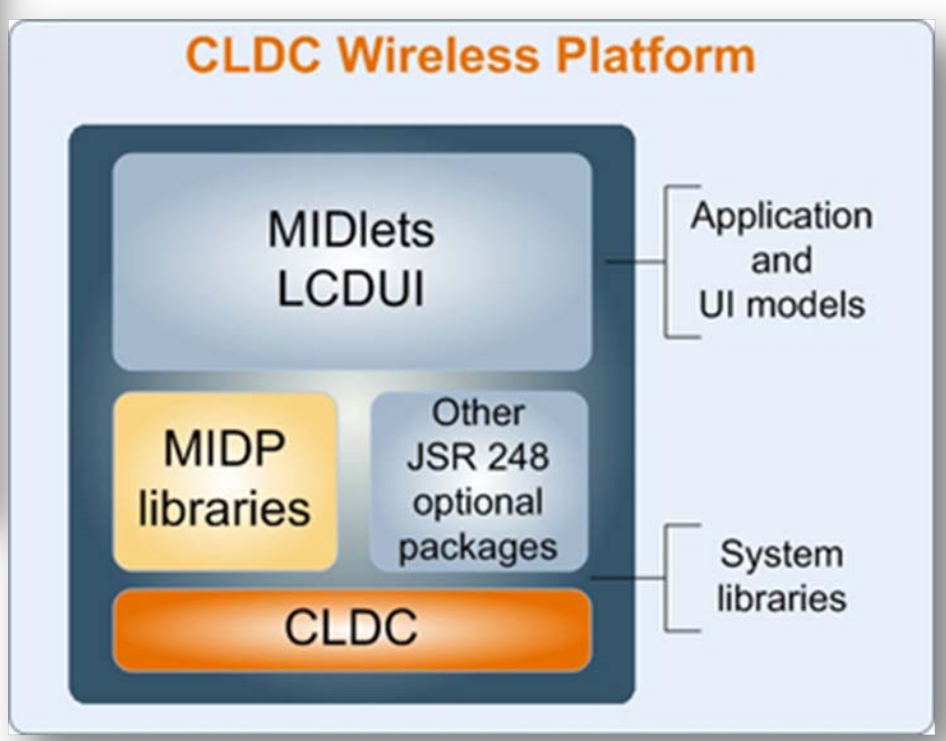
- Conjunto de APIs adicionales que pueden ser añadidos de forma flexible sobre diferentes perfiles
- Utilizados en una multitud de dispositivos y familias de dispositivos, ya que contienen una funcionalidad que es independiente del segmento vertical
  - Bluetooth, gestión de contenido multimedia, localización, …
- Un dispositivo puede soportar múltiples paquetes opcionales
- Ejemplos de paquetes opcionales sobre **CDC**:
  - **JSR 66: RMI Optional Package.** Subconjunto de J2SE RMI.
  - **JSR 169: JDBC Optional Package.** Soporte JDBC en dispositivos CDC.
  - **JSR 209: Advanced Graphics and User Interface Optional Package.** Facilidades de migración para interfaces de usuario y gráficos avanzados de J2SE a J2ME

Paquetes opcionales sobre CLDC

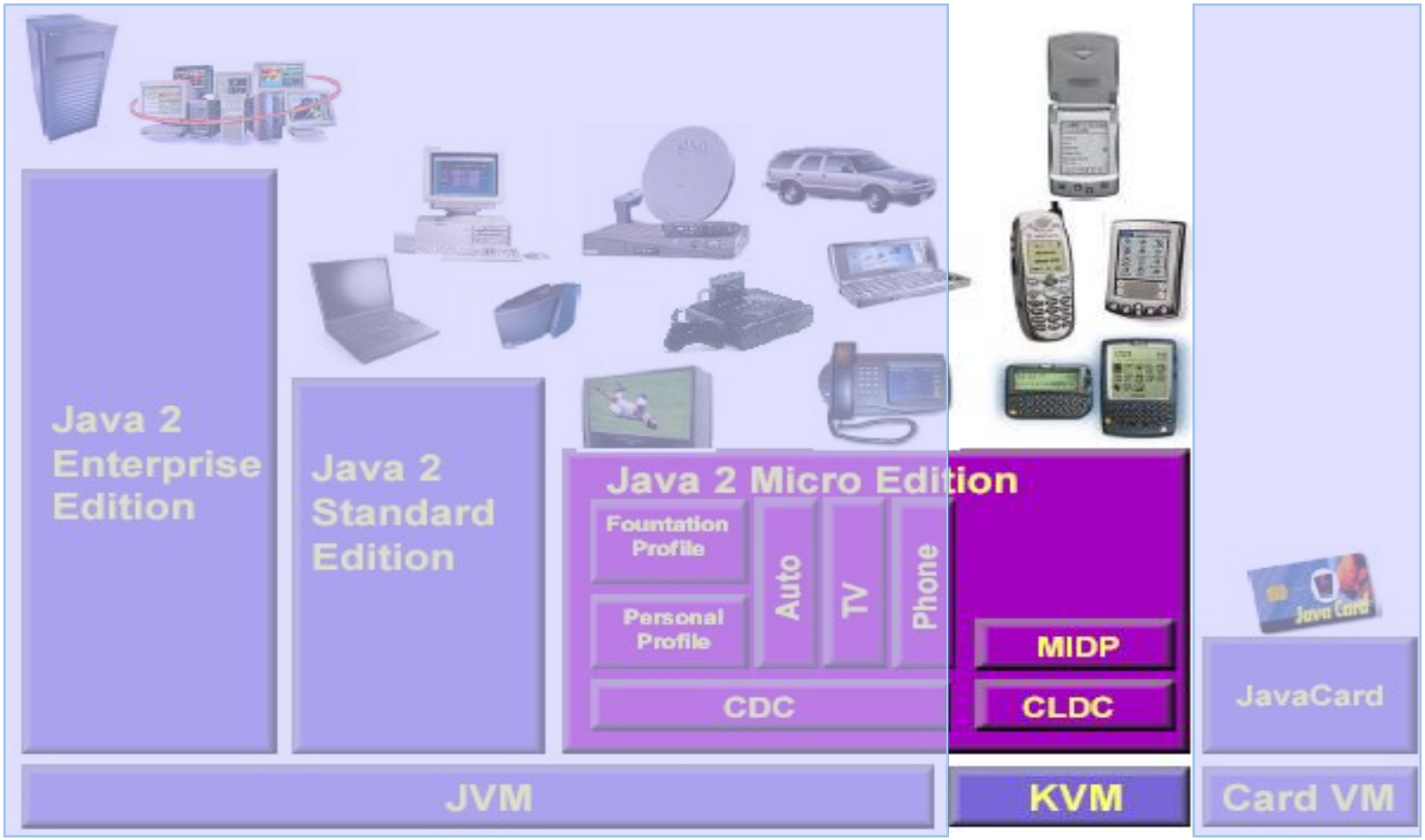
- **JSR 75: PDA Optional Package.** Acceso a ficheros y datos personales
- **JSR 82: Bluetooth API.** Desarrollo de aplicaciones que usan Bluetooth
- **JSR 120, JSR 205 (2.0): Wireless Messaging API**. Acceso a sistemas de envío de mensajes (SMS, Cell Broadcast Service)
- **JSR 135: Mobile Media API (MMAPI).** Acceso y reproducción de recursos multimedia (audio, video)
  - JSR 234: Funcionalidades multimedia avanzadas
- **JSR 172: Web Services APIs.** Desarrollo de clientes Web en dispositivos móviles
- **JSR 177: Security and Trust Services.** Añade almacenamiento seguro, APIs criptográficas, firmas digitales, gestión de credenciales
- **JSR 179, 293: API de Localización (versiones 1.0 y 2.0).** Acceso a la información de localización física

Ejemplos de arquitecturas





MIDP/CLDC/KVM



Memory: 10MB ↔ 1MB 512KB ↔ 32KB

64 bit 32 bit 16 bit 8 bit

CLDC/KVM

- CLDC/KVM cubre:
  - **máquina virtual** y soporte al lenguaje Java, modelo de **seguridad**, **entrada/salida**, **soporte** a conexiones de **red**, **internacionalización**
- **CVM** ⇨ diferencias con JVM
  - No soporta finalización de instancias de clase
  - Limitaciones en el manejo de errores
  - No soporta *Java Native Interface* (JNI)
  - No soporta reflexión (*reflection*)
  - No soporta cargadores de clase definidos por el usuario
  - No soporta grupos de hilos ni de demonios
  - Verificación de código en dos fases: pre-verificación y comprobación de clases más ligera
  - Soporta referencia débil
  - `Calendar`, `Date`, `TimeZone` rediseñadas

CLDC/KVM - Seguridad

- No soporta el modelo completo de J2SE
  - seguridad a nivel máquina virtual: verificador de clases
  - seguridad a nivel de aplicación: modelo “sandbox”
- Verificador de clases en dos pasos:
  - pre-verificador externo
  - verificador en el dispositivo
- Modelo “sandbox”:
  - No se pueden sobreescribir clases del sistema
  - No se pueden acceder a clases nativas (JNI)
  - Restringido al API proporcionada por el CLDC y el perfil sobre el que desarrolla

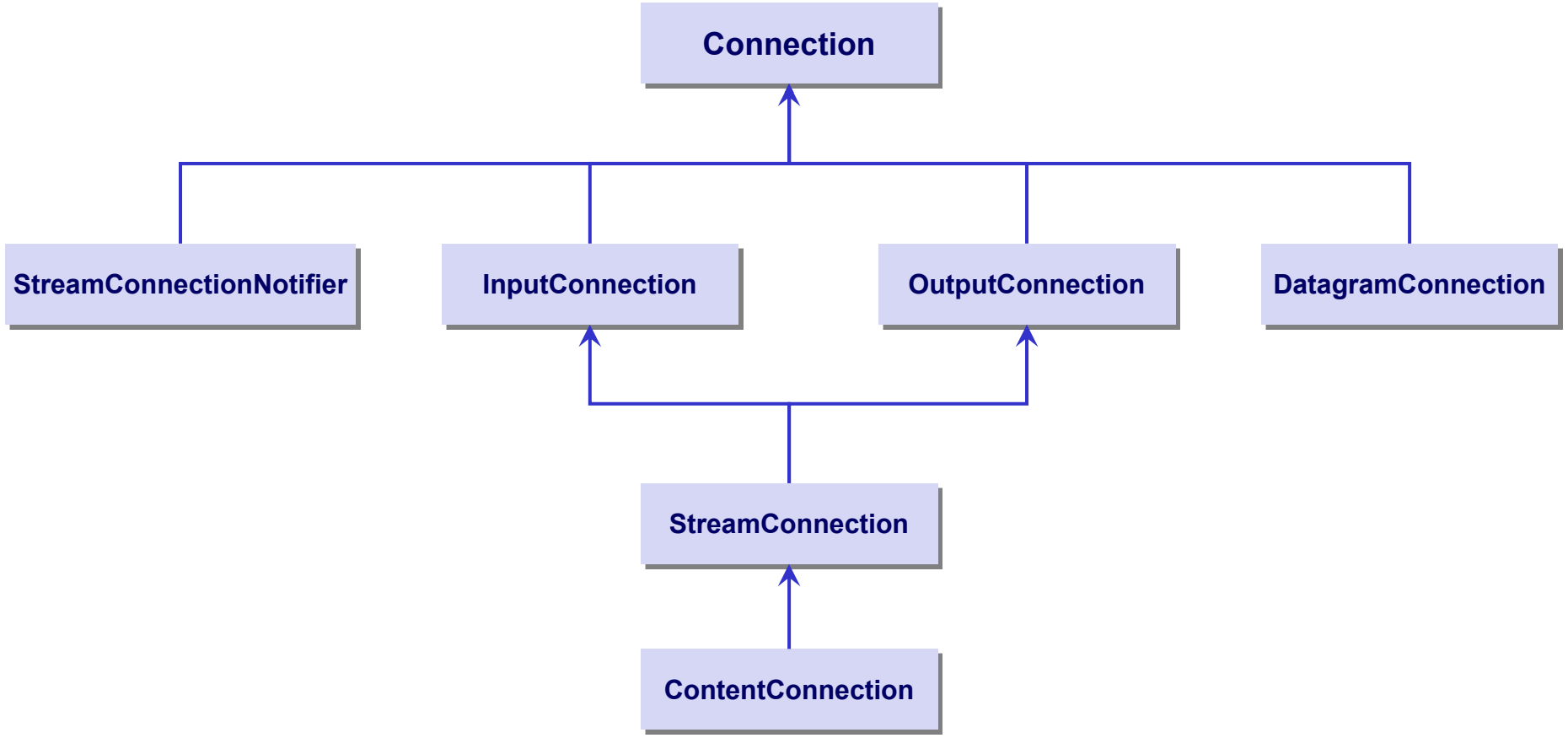
CLDC/KVM – E/S
sistemas de almacenamiento y red

- Nuevo soporte porque el de J2SE presenta los siguientes problemas:
  - Gran tamaño: más 100 clases (200 kB).
  - No estaba pensado para pequeños dispositivos:
    - Se suponía TCP/IP siempre disponible
    - No es fácil de extender a nuevos protocolos no TCP/IP tipo Bluetooth o IrDA
- CLDC introduce **`Generic Connection Framework`**:
  - Soporte a diferentes tipos de protocolos de red
  - Permite definir y usar nuevos protocolos de forma sencilla
  - Compatibilidad con Java estándar, mediante mapeo

CLDC/KVM - GCF

- CLDC especifica un mecanismo general de conexión:
  - **`Connector.open("<protocol>:<address>;<parameters>");`**
  - Por ejemplo:
    - Ficheros:
      - **`Connector.open("file://midp.txt");`**
    - HTTP:
      - **`Connector.open("http://www.sun.com");`**
    - Sockets:
      - **`Connector.open("socket://129.144.111.222:9000");`**
    - Puerto serie:
      - **`Connector.open("comm:0;baudrate=9600");`**
- No implementa ningún protocolo, son los perfiles los que deben definir qué conector(es) debe(n) implementarse

CLDC/KVM – Interfaces GCF



CLDC/KVM
Internacionalización

- Todos los dispositivos CLDC soportan por defecto **`ISO-LATIN1`** (**`microedition.encoding`** con valor "**`ISO8859_1`**")
- Los fabricantes pueden proporcionar códigos adicionales:
  - Por ejemplo, NTT DoCoMo requiere que los teléfonos iMode soporten la codificación japonesa, ShiftJIS
- No se soportan soluciones relacionadas con el formato de fechas, tiempo, o moneda
- Las propiedades del sistema se obtienen vía **`java.lang.System`**
  - La llamada a **`System.getProperty(String key)`** devuelve el valor de la propiedad como un **`String`**
- CLDC deben proporcionar al menos las siguientes propiedades:
  - **`microedition.platform`** - **`microedition.encoding`**
  - **`microedition.configuration`** - **`microedition.profile`**

Programación en Mobile Information Device Profile (MIDP)

Florina Almenárez Mendoza
Celeste Campo
Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid
{florina, celeste}@it.uc3m.es

Parte de este material se basa en transparencias de Natividad Martínez Madrid (nati@it.uc3m.es)

Contexto

- Conocer las APIs de programación de MIDP
- Aprender a desarrollar aplicaciones con MIDP 2.0

Bibliografía:

- ***J2ME : Java 2 micro edition : manual de usuario y tutorial***. Froufe, Agustín y Jorge, Patricia. Ra-Ma. [2004]. L/S 004.438 JAVA FRO, L/D 004.438 JAVA FRO. Capítulos 6, 9 al 12.
- ***Especificación de MIDP 2.0*** (JSR 118). Disponible en `http://www.jcp.org`
- ***Wireless Java Programming with Java 2 Micro Edition***. Feng, Yu and Zhu, Jun. SAMS [2001]. L/D 004.438 JAVA FEN. Capítulos 5 al 9.
- Java Mobile – Start Learning (Tutorial from Oracle)

Índice

- **Introducción**
- MIDlets
  - Conceptos básicos
  - Desarrollo y despliegue
- Librerías de MIDP
- Interfaz de usuario
- Almacenamiento persistente
- Conectividad

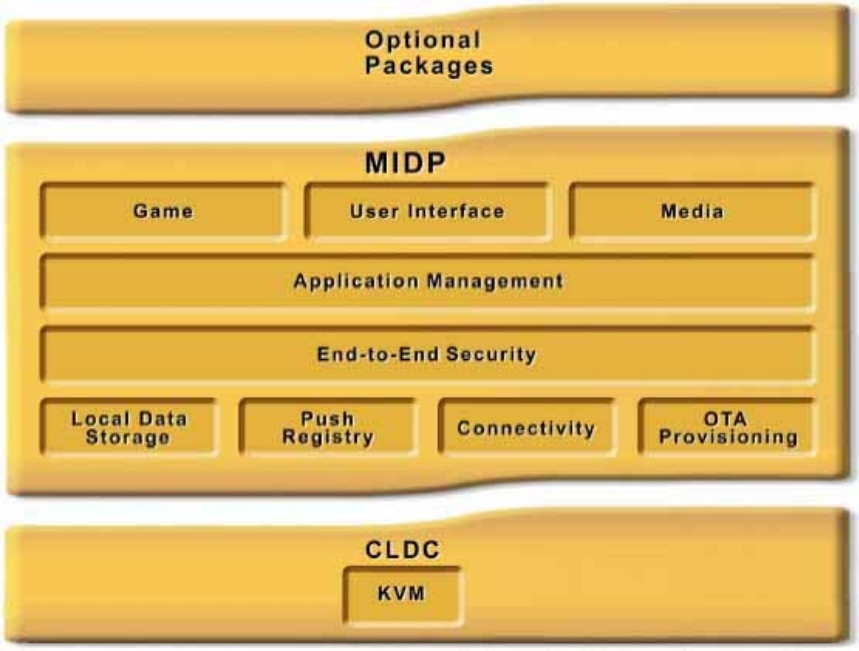
Introducción

- **MIDP 1.0**
  - JSR 30
  - Final Release: Sep, 2000
- **MIDP 2.0**
  - JSR 118
  - Final Release: Nov, 2002
  - Final Release 2: Jun, 2006
- **MIDP 3.0**
  - JSR 271
  - Final Release: Dec, 2009
  - MIDlets en CLDC, CDC, y OSGi

Introducción (II)

- Define el conjunto de APIs disponibles para el desarrollo de aplicaciones portables entre dispositivos móviles.

- MIDP **cubre**:
  - Ciclo de vida de la aplicación
  - Interfaz de usuario
  - Soporte de red
  - Almacenamiento persistente
  - Sonidos
  - Juegos en 2D
  - Seguridad extremo a extremo
  - *Timers*, excepciones, …



Fuente: http://grasia.fdi.ucm.es/j2me/_J2METech/MIDP.html

- Asume la existencia de un *Application Management System* (AMS):
  - Dependiente del dispositivo
  - Instala, interacciona con, actualiza versiones de y borra MIDlets

Índice

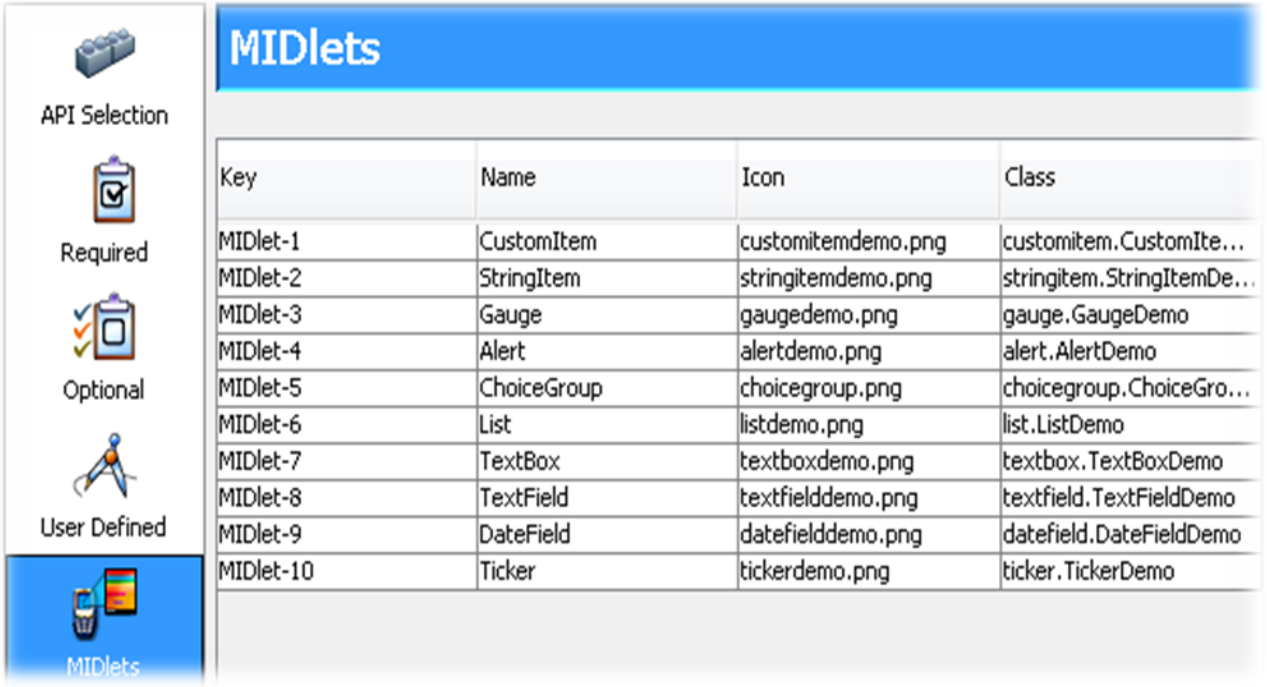
- Introducción
- **MIDlets**
  - **Conceptos básicos**
  - **Desarrollo y despliegue**
- Librerías de MIDP
- Interfaz de usuario
- Almacenamiento persistente
- Conectividad

MIDlets

- Un MIDlet es la unidad básica de ejecución en MIDP
  - Tiene un ciclo de vida bien definido.
  - Da información descriptiva sobre sí mismo.
  - Extiende **`javax.microedition.midlet.MIDlet`**
- Existe el concepto de MIDlet permanente:
  - Reside, al menos en parte, en memoria no volátil (ROM, EEPROM).
  - Puede descargarse de la red y grabarse en memoria persistente.
  - Pueden ser ejecutados repetidas veces por el usuario sin necesidad de volver a descargarlos.

MIDlet Suite

- Conjunto de aplicaciones (MIDlets) que comparten recursos en el contexto de una única máquina virtual.
- Sólo desarrollemos un MIDlet se debe empaquetar en un MIDlet Suite.



Key	Name	Icon	Class
MIDlet-1	CustomItem	customitemdemo.png	customitem.CustomIte...
MIDlet-2	StringItem	stringitemdemo.png	stringitem.StringItemDe...
MIDlet-3	Gauge	gaugedemo.png	gauge.GaugeDemo
MIDlet-4	Alert	alertdemo.png	alert.AlertDemo
MIDlet-5	ChoiceGroup	choicegroup.png	choicegroup.ChoiceGro...
MIDlet-6	List	listdemo.png	list.ListDemo
MIDlet-7	TextBox	textboxdemo.png	textbox.TextBoxDemo
MIDlet-8	TextField	textfielddemo.png	textfield.TextFieldDemo
MIDlet-9	DateField	datefielddemo.png	datefield.DateFieldDemo
MIDlet-10	Ticker	tickerdemo.png	ticker.TickerDemo

Desarrollo y despliegue de MIDlets

1. Creación (etapas de desarrollo)
2. Publicación
3. Descarga
4. Instalación
5. Ejecución
6. Actualización (gestión de versiones)
7. Borrado

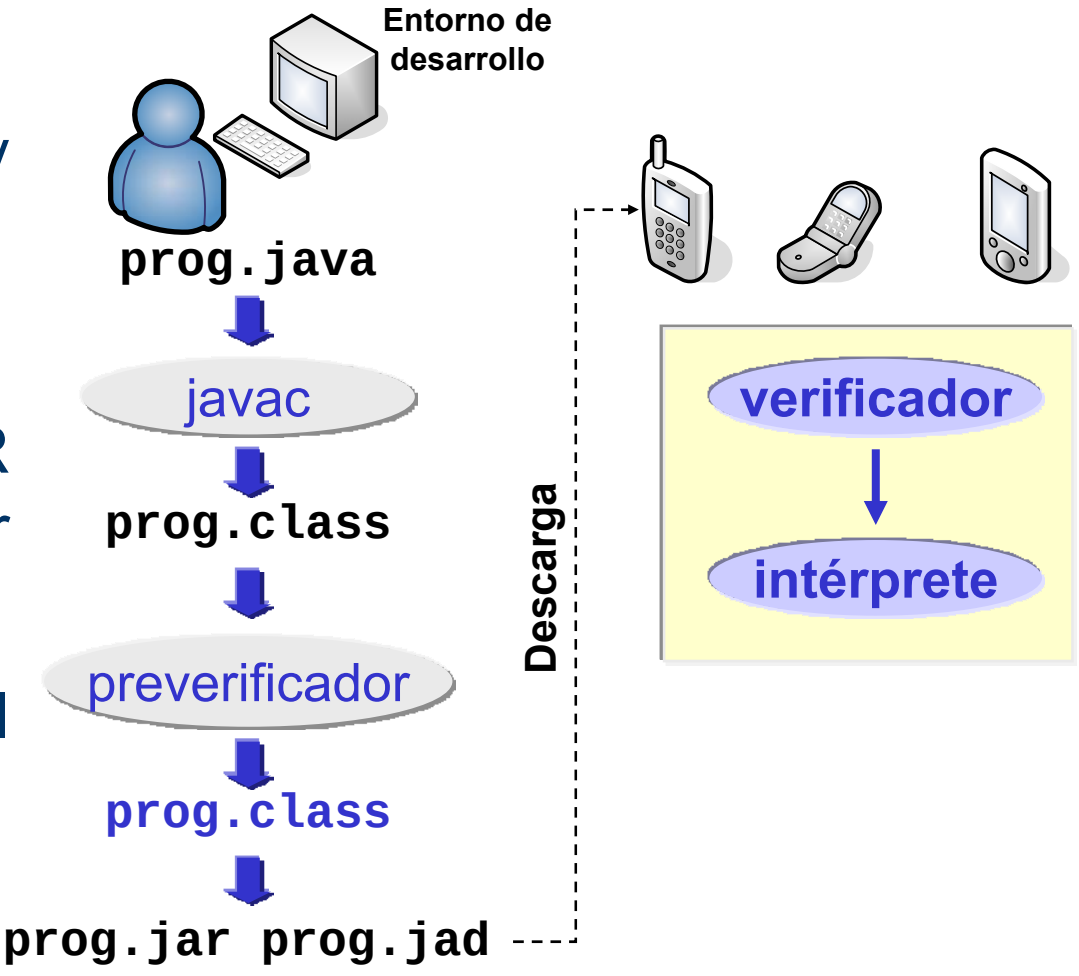
A

M

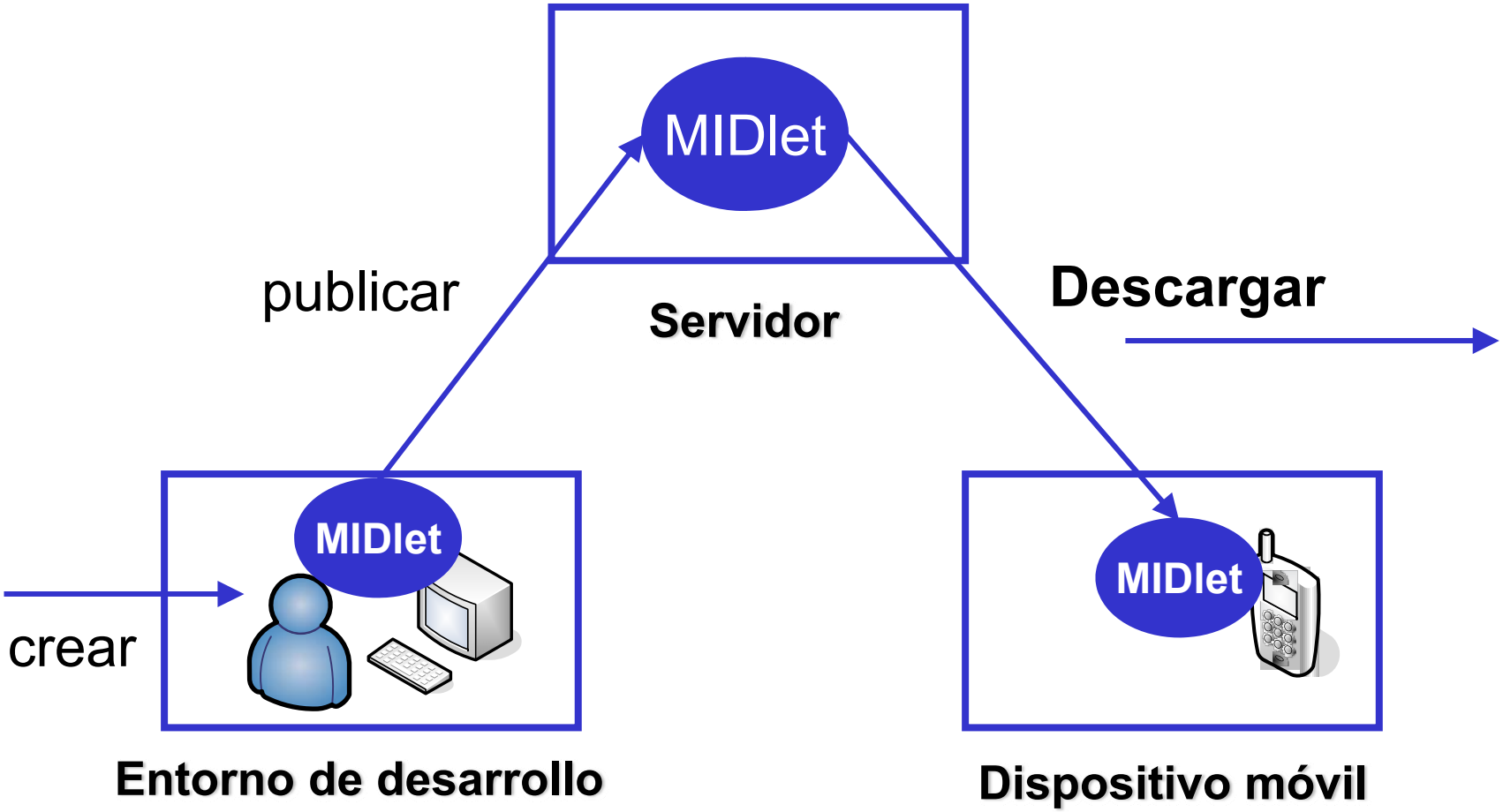
S

1. Creación

- Escribir el código y compilar
- Preverificar el código
- Empaquetar en un JAR y crear el descriptor (JAD)
- Ejecutar en el emulador
- Depurar los fallos



2. Publicación



3. Descarga

- Gestionada por el *Application Management System (AMS)*
- El dispositivo obtiene el MIDlet de alguna fuente:
  - red inalámbrica (Wi-Fi, Bluetooth, UMTS, GPRS, ...)
  - puerto serie
  - IrDA
  - ...
- Negociación sobre capacidades del dispositivo según los requisitos del MIDlet, coste, ...
- Se descarga el MIDlet a la memoria del dispositivo

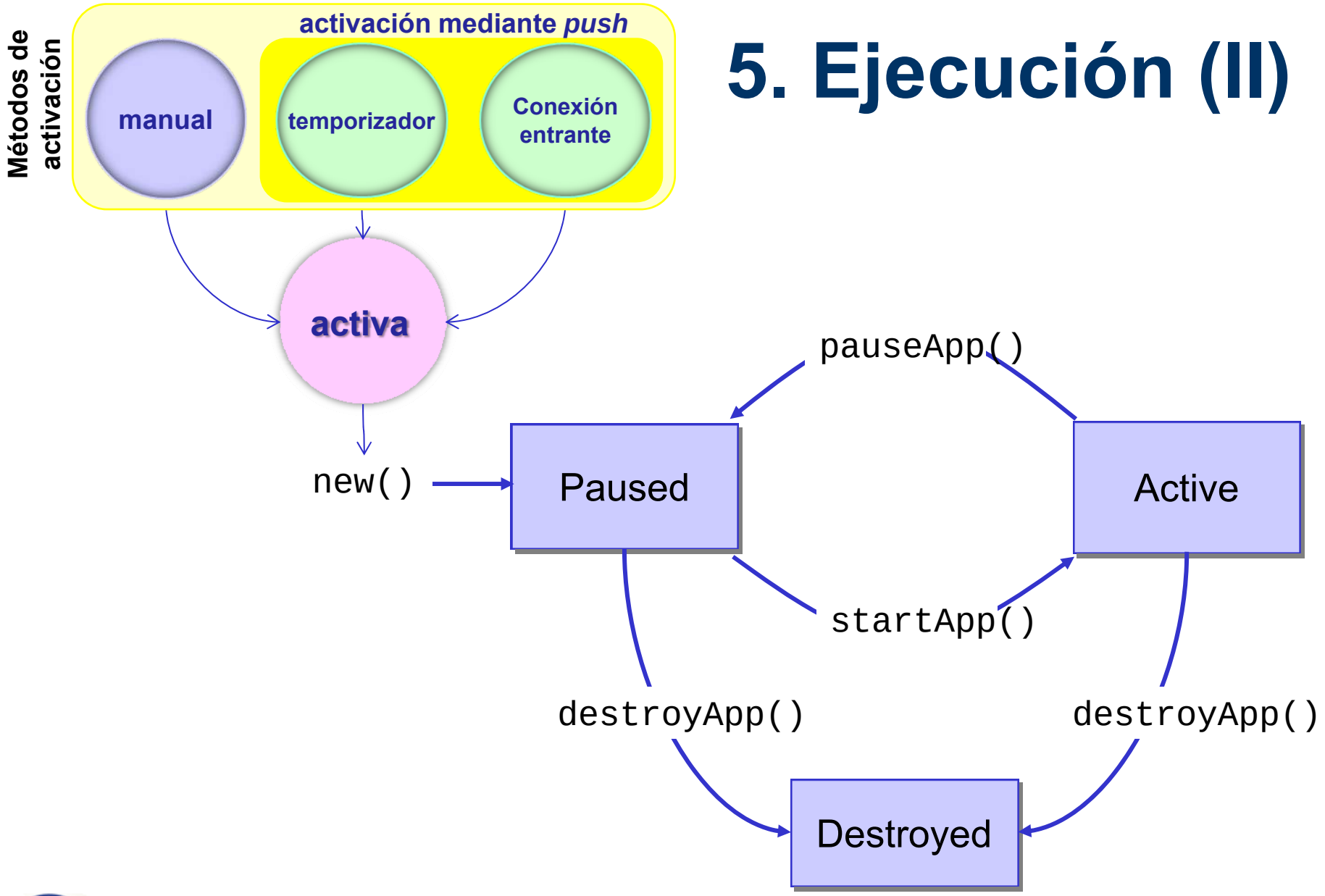
4. Instalación

- Gestionado por el *AMS*
  - Información al usuario sobre el proceso
- Puede comprobar que el MIDlet no vulnera las políticas de seguridad del móvil
- Puede transformar (convertir) el MIDlet de formato "público" a un formato específico del dispositivo:
  - Ejemplo: en PalmOS se transforma a formato PRC.
- El MIDlet queda almacenado en una zona de memoria persistente

5. Ejecución

- Existen dos mecanismos de ejecución:
  - el usuario selecciona el MIDlet y lo ejecuta
  - activación automática (*Push Registry*) en MIDP 2.0
    - *Push Registry* forma parte del AMS ⇨ gestiona las activaciones
    - a través de una alarma o temporizador
    - a través de una conexión entrante (TCP, UDP o **SMS**)
- En este momento, el MIDlet entra en la VM y se invocan los métodos que gestionan su ciclo de vida:
  - **Paused**: Ha sido creado pero aún no se ha ejecutado y en espera.
  - **Active**: En ejecución.
  - **Destroyed**: Ha liberado recursos, destruido hilos y terminado toda su actividad.

5. Ejecución (II)



6. Actualización

- Puede publicarse una nueva versión del MIDlet.
- *AMS* debe gestionar la lista de MIDlets instalados y sus versiones
  - puede así actualizar de versiones más antiguas a más recientes del MIDlet
- Los atributos del MIDlet, incluida la versión, están:
  - En el descriptor del MIDlet (JAD).
  - En el manifiesto del MIDlet contenido en el JAR.

7. Borrado

- El AMS debe permitir al usuario eliminar MIDlets.

- Se borra:
  - MIDlet
  - los registros en memoria permanente escritos por ese MIDlet
  - los recursos asociados al mismo

JAR y Manifiesto

- Incluye los ficheros de clases y otros recursos asociados al MIDlet, por ejemplo imágenes.
- Fichero JAR puede contener un MIDlet Suite
- El manifiesto está incluido en el JAR y contiene información sobre los contenidos del fichero JAR:

Atributos obligatorios	Atributos opcionales

`MIDlet-Name`
`MIDlet-Version`
`MIDlet-Vendor`
`MIDlet-<n> (name, icon, class)`
`MicroEdition-Profile`
`MicroEdition-Configuration`
	`MIDlet-Description`
	`MIDlet-Icon`
	`MIDlet-Info-URL`
	`MIDlet-Data-Size`
	`MIDlet-Permissions`
	`MIDlet-Permissions-Opt`
	`MIDlet-Push-<n>`

- Otros atributos específicos de la aplicación

Descriptor (JAD)

- Permite que el *AMS* verifique si el MIDlet es indicado antes de descargarlo.
- Es un fichero de texto con extensión **`.jad`**.

Atributos obligatorios	Atributos opcionales
`MIDlet-Name`	`MIDlet-<n> (name, icon, class)`
`MIDlet-Version`	`MicroEdition-Profile`
`MIDlet-Vendor`	`MicroEdition-Configuration`
`MIDlet-Jar-URL`	`MIDlet-Description`
`MIDlet-Jar-Size`	`MIDlet-Icon`
	`MIDlet-Info-URL`
	`MIDlet-Data-Size`
	`MIDlet-Permissions`
	`MIDlet-Permissions-Opt`
	`MIDlet-Push-<n>`
	`MIDlet-Install-Notify`
	`MIDlet-Delete-Notify`
	`MIDlet-Delete-Confirm`

- Puede incluir otros atributos específicos de la aplicación

Ejemplo HelloWorld

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

public class HelloWorld extends MIDlet
                        implements CommandListener {
  // Componentes de UI del MIDlet
  private Display display;

  private TextBox mainScreen = null;

  private Command exit = new Command("exit",
   Command.EXIT, 2);
```

Implementa **startApp(), pauseApp(), destroyApp()**

Implementa **commandAction(c,s)**

Gestor de la pantalla y dispositivos de entrada. Uno por MIDlet

Permite introducir y editar texto

Botón de comando que permite ejecutar una acción

Ejemplo HelloWorld (II)

```
 // Constructor sin parámetros
public HelloWorld() {
  display = Display.getDisplay(this);

  mainScreen = new TextBox("Text Box", "Hola Mundo",
                           512,0);

  mainScreen.addCommand(exit);

  mainScreen.setCommandListener(this);
}
```

Obtiene una referencia del display del MIDlet

Comando asociado a `mainScreen`

Se establece `mainScreen` como escuchador de “exit”

Ejemplo HelloWorld (III)

```
// Implementa el método startApp()
public void startApp() {
   display.setCurrent(mainScreen);
 }

// Implementa el método pauseApp()
public void pauseApp() {
 }

// Implementa el método destroyApp()
public void destroyApp(boolean unconditional) {
 }
```

Hace el `TextBox` visible

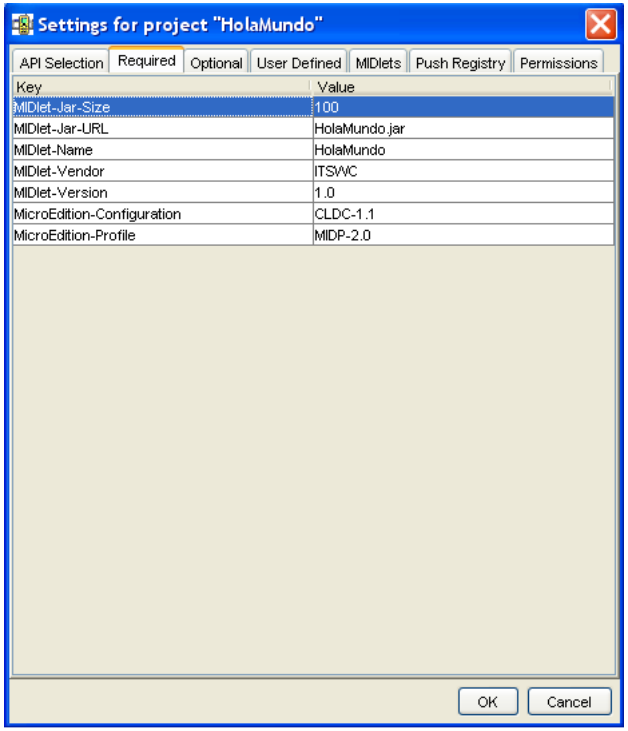
Ciclo de vida del MIDlet

Ejemplo HelloWorld (IV)

```
    /*
     * El MIDlet implementa el método escuchador
     * correspondiente del interfaz CommandListener
    */
    public void commandAction(Command c, Displayable s) {
        if (c == exit) {
            destroyApp(true);
            notifyDestroyed();
        }
    }
}
```

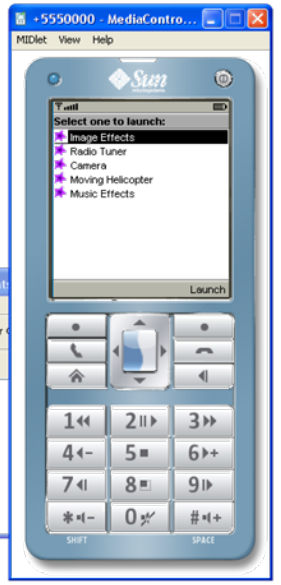
Ejemplo HelloWorld JAR y JAD

```
MIDlet-Name: HolaMundo
MIDlet-Version: 1.0
MIDlet-Vendor: ITSWC
MIDlet-1:   HolaMundo,   /hola.png,
  uc3m.it.swc.HolaMundo
MIDlet-2:  HolaCanvas,  /canvas.png,
  uc3m.it.swc.HolaCanvas
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.1
MIDlet-Description: Mi primer MIDlet
MIDlet-Jar-URL: HolaMundo.jar
MIDlet-Jar-Size: 100
```

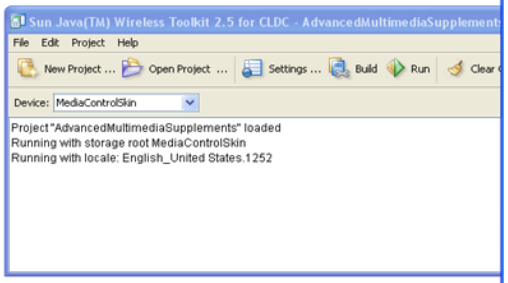


Desarrollo de MIDlets

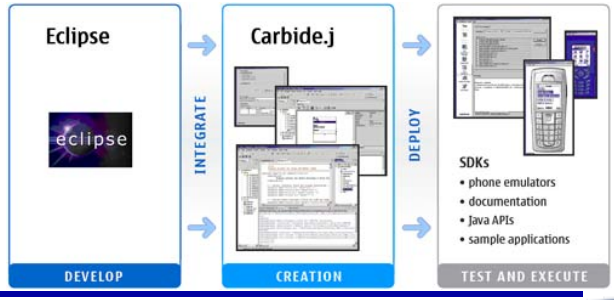
- **SDK de Java 2** ⇨ compilar aplicaciones J2ME



- **Kits de desarrollo**
  - Sun Wireless Toolkit



  - Nokia ⇨ carbide.j
    - Eclipse, Sun One Studio, Borland JBuilder



  - Siemens ⇨ Siemens Mobile Toolkit

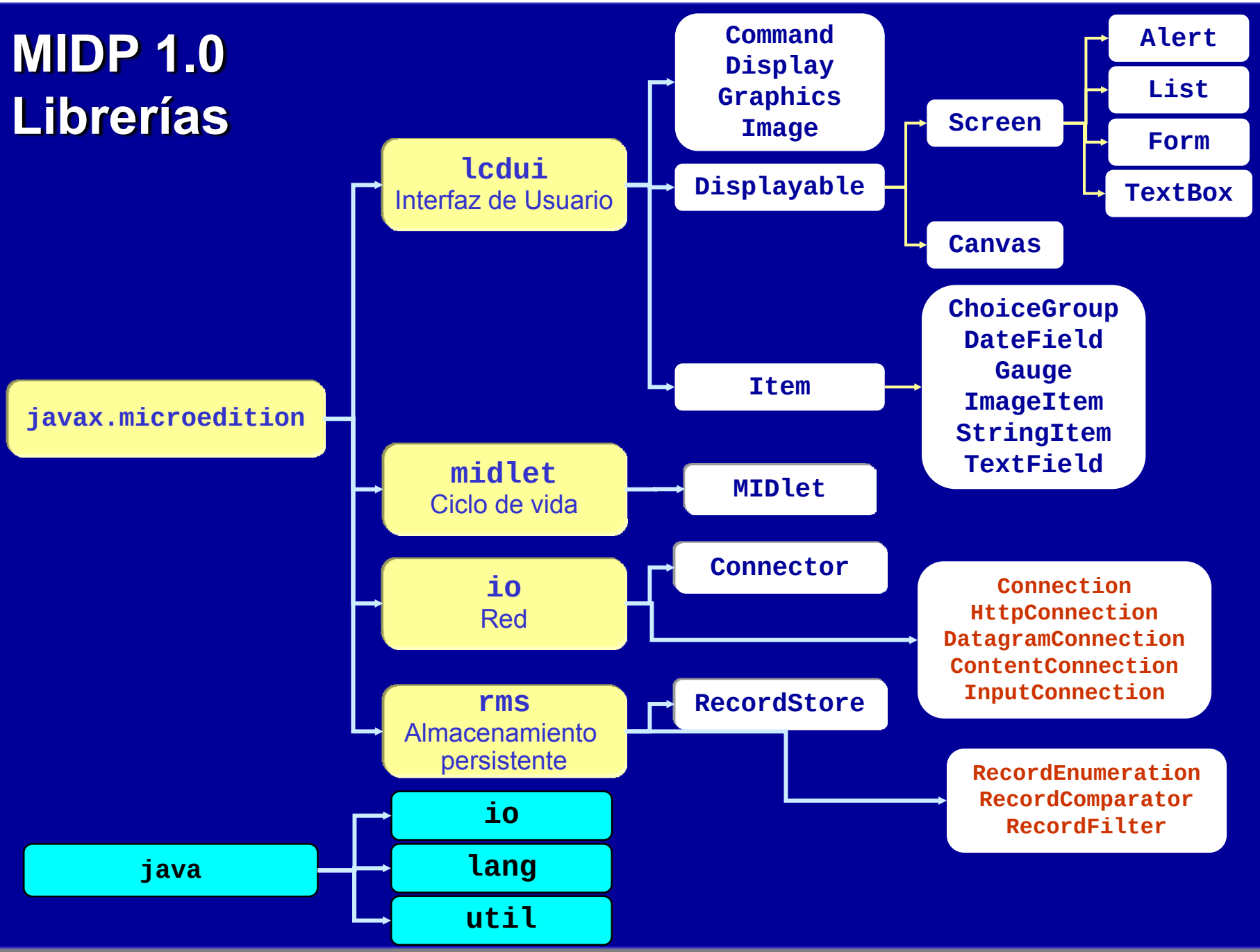
Desarrollo de MIDlets (II)

- **Entornos de desarrollo integrados**
  - **Java ME Platform SDK 3.0**
  - **Eclipse**
    - Plugins: **EclipseME**, SIPTech J2ME, Wirelesoft VistaMax, …
    - EasyEclipse Mobile Java
    - Requiere integración con emuladores (WTK, Nokia, …)
  - **NetBeans** ⇨ WTK, Plugin J2ME, Mobility Pack 4.1
  - Websphere Studio Device Developer (**WSDD**), IBM

Índice

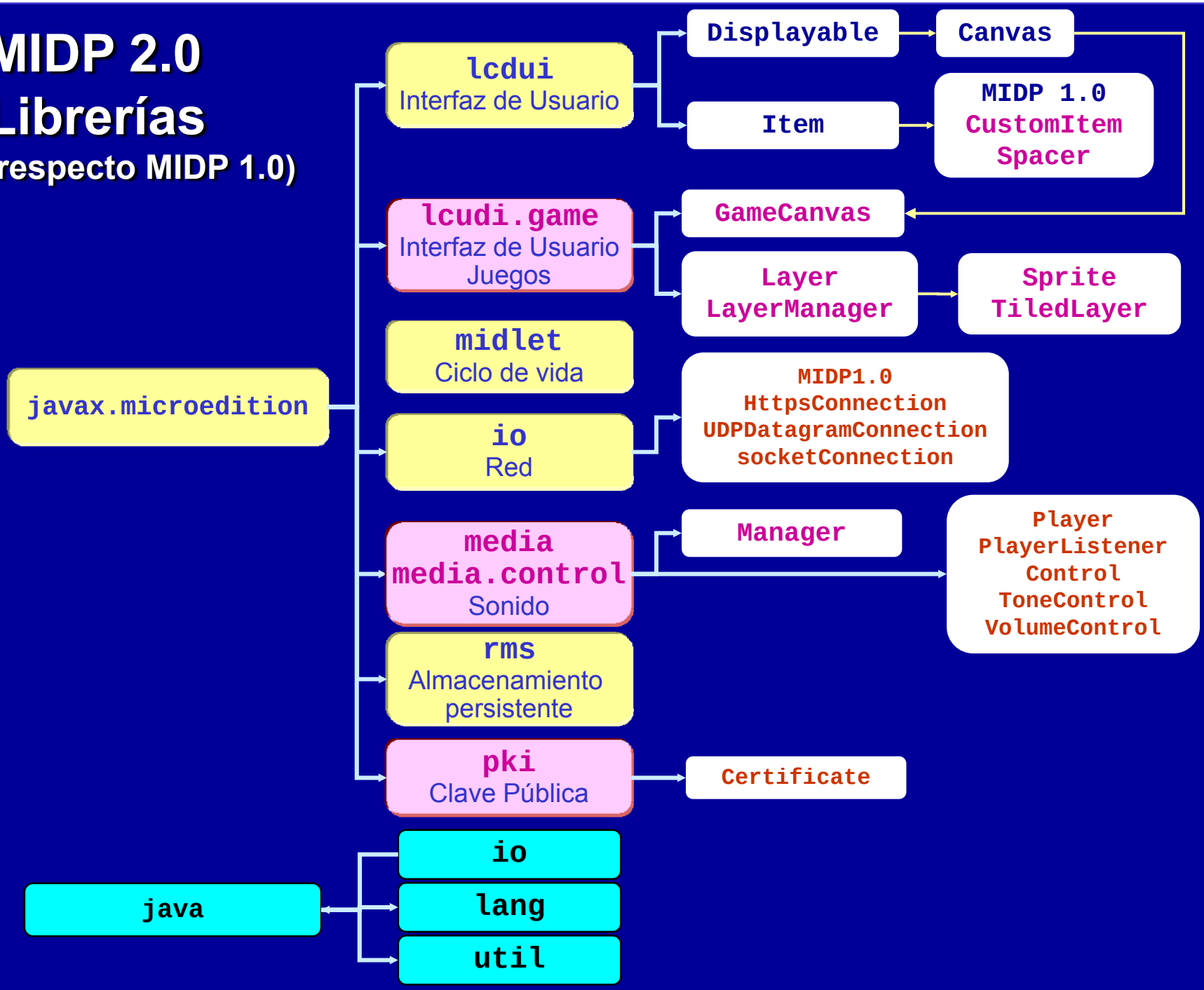
- Introducción
- MIDlets
  - Conceptos básicos
  - Desarrollo y despliegue
- **Librerías de MIDP**
- Interfaz de usuario
- Almacenamiento persistente
- Conectividad

MIDP 1.0 Librerías



MIDP 2.0 Librerías

(respecto MIDP 1.0)



MIDlet
javax.microedition.midlet.MIDlet

- Clase abstracta base para todos los MIDlets:
  - Constructor: **`protected MIDlet()`**
  - `protected abstract void startApp() throws MIDletStateChangeException`
  - `protected abstract void pauseApp()`
  - `protected abstract void destroyApp(boolean unconditional) throws MIDletStateChangeException`
  - **`public final void notifyDestroyed()`**
    - Comunica al AMS que el MIDlet ha limpiado la memoria y ha terminado.
  - **`public final void notifyPaused()`**
    - Comunica al AMS que el MIDlet está en pausa.
  - **`public final String getAppProperty(String key)`**
    - Se le llama para obtener las propiedades del MIDlet (descriptor JAD)

Índice

- Introducción
- MIDlets
  - Conceptos básicos
  - Desarrollo y despliegue
- Librerías de MIDP
- **Interfaz de usuario**
- Almacenamiento persistente
- Conectividad

Interfaz de usuario

- API de alto nivel:
  - Muy portable
  - Orientada a “screen” y “widget”
  - Las aplicaciones que usan este API deberían funcionar en todos los dispositivos
  - No hay acceso a todas las funciones del dispositivo
  - Más sencillo y menos potente que AWT
- API de bajo nivel:
  - Primitivas de dibujo ⇨ control gráfico de la pantalla a nivel de píxel
  - Eventos de teclado
  - Más flexibilidad, menos portabilidad, mejor “experiencia del usuario”
- Nueva API ⇨ **LWUIT**
  - Interfaces gráficas al estilo de `Swing`

Interfaz de usuario gráfico

- **Paquete**:
  - **`javax.microedition.lcdui`**

- **Clases** básicas:
  - **`Displayable`**:
    - información a ser visualizada
    - elemento funcional que encapsula la forma específica en que un dispositivo permite la realización de gráficos y el manejo de las entradas de usuario
  - **`Display`**:
    - Selecciona qué objeto **`Displayable`** se muestra al usuario

Clase `Displayable`

- Existen tres categorías de objetos `Displayable`:
  - **`Screen`** con estructura predefinida:
    - **`Alert`**, **`List`** o **`TextBox`** (subclases de **`Screen`**)
    - Encapsulan componentes de interfaces complejos que las aplicaciones no pueden enriquecer con nuevos componentes.
  - **`Screen`** genérico:
    - **`Form`** (subclase de **`Screen`**)
    - Las aplicaciones pueden llenar esta pantalla con texto, imágenes u otros componentes (objetos `Item`) de interfaz gráfico.
  - **`Canvas`** (API de bajo nivel):
    - Usuario tiene control total sobre los componentes del `display` y puede acceder a eventos de bajo nivel.

Clase `Display`

- Métodos para controlar la visualización de objetos **`Displayable`** y obtener propiedades del dispositivo
  - color, número de colores, vibración, etc.
- Sólo uno por MIDlet (*singleton*).
- Obtener el objeto `Display`:
  - `static Display getDisplay(MIDlet m)`
- Obtener el **`Displayable`** que se está visualizando:
  - `Displayable getCurrent()`
- Establecer el **`Displayable`** a visualizar:
  - `void setCurrent(Displayable nextDisplayable)`
  - `void setCurrent(Alert alert, Displayable nextDisplayable)`

Eventos y su gestión

- Mismo modelo que AWT:
  - Fuentes de eventos y escuchadores (*listeners*) de evento
- Gestión en el mismo hilo en el que se produce el evento
- Eventos de alto nivel:
  - **`CommandAction(Command c, Displayable d)`**
    - `CommandListener`
    - Fuente: `Displayable`
  - **`ItemStateChanged(Item i)`**
    - `ItemStateListener`
    - Fuente: `Form`
    - `Item` interactivos: `Gauge`, `ChoiceGroup`, `TextField`, …
- Eventos de bajo nivel:
  - Relacionados con pulsaciones de teclas (`keyPressed`, `keyReleased`, `keyRepetead`), de puntero (`pointerPressed`, `pointerReleased`), . . .

Clase Command

- Un objeto **command** tiene tres atributos:

  `Command(String label, int Type, int priority)`

  1. *Label*: `String` representando el significado del comando, lo que la aplicación muestra al usuario.
  2. *Type*: `BACK`, `CANCEL`, `HELP`, `EXIT`, `ITEM`, `OK`, `SCREEN` y `STOP`.
  3. *Priority:* Entero que indica la importancia del comando. Mayor cuanto menor sea el número.

- Añadir comando a un `Displayable`:

  `addCommand(Command)`

- Eliminar comando de un `Displayable`:

  `removeCommand(Command)`

Imágenes

- Clase **`Image:`**
  - Imágenes inmutables:
    - No se pueden modificar
    - Generadas a partir de un fichero (recurso, descargado,…)
    - Tipo de imágenes en un **`Alert`**, **`List`** o **`Form`**
  - Imágenes mutables:
    - Se pueden modificar
- ¿Cómo se crean?
  - Inmutables
    - `createImage(String nombre)`
    - `createImage(byte[] data, int offset, int longitud)`
    - `createImage(Image imagen)`
    - `createImage(InputStream stream)`
  - Mutables
    - `createImage(int ancho, int alto)`

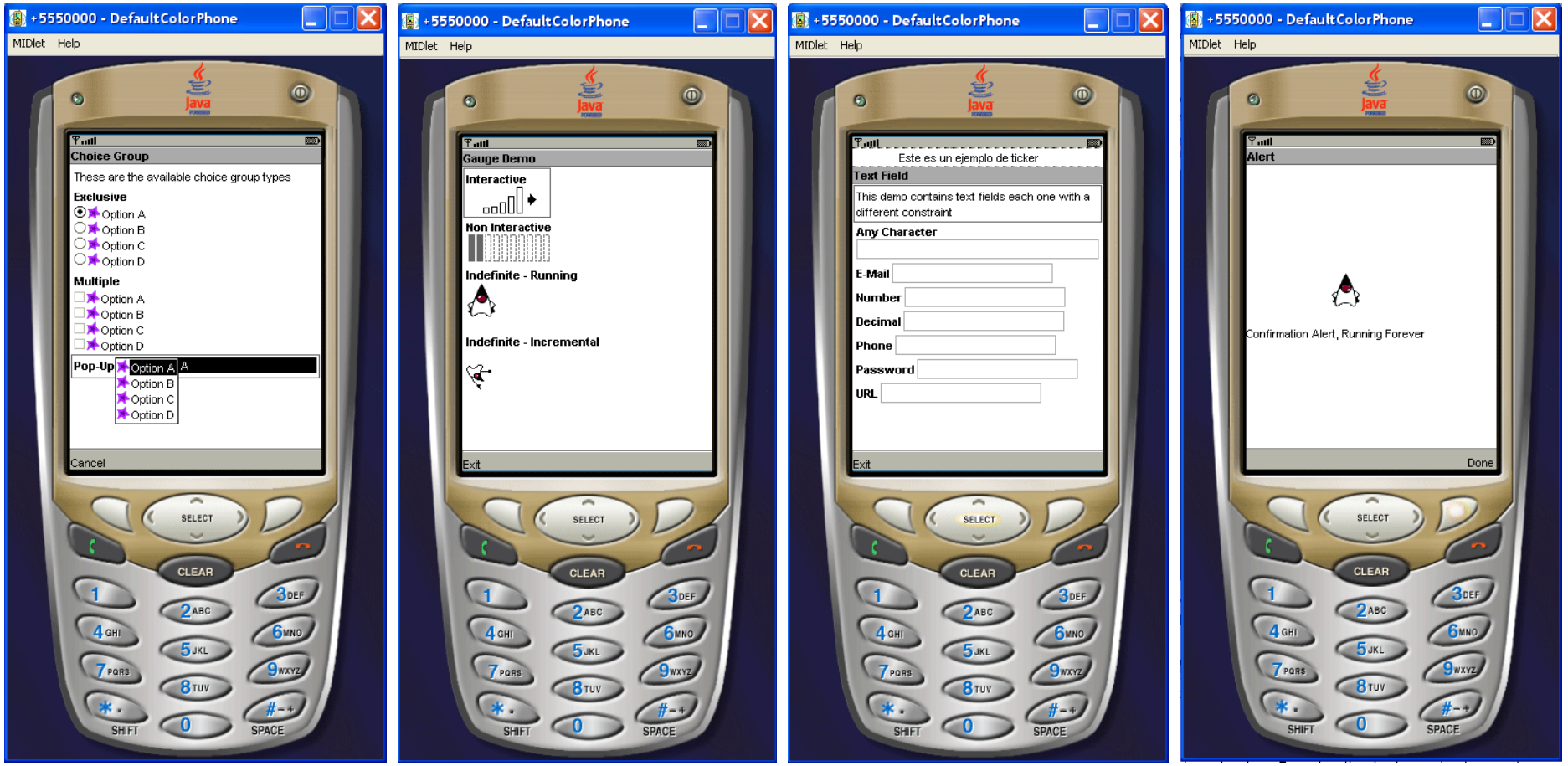
API UI de alto nivel

- Un objeto de tipo `Ticker` (marquesina) consiste en un texto que se desplaza continuamente a través de la pantalla
  - `Ticker(String texto)`
- **`Alert`** permite visualizar datos durante un cierto tiempo (*timeout*) antes de pasar a otra pantalla
  - *timeout* en milisegundos
  - temporizador infinito (**`Alert.FOREVER`**) hasta pulsar una tecla
- **`List`** Implementa el interfaz **`Choice`**
  - Los tipos de listas son definidos en el interfaz `Choice`
    - `EXCLUSIVE:` un elemento seleccionado simultáneamente
    - `IMPLICIT:` el elemento que está "enfocado" es el que se selecciona implícitamente
    - `MULTIPLE:` se puede seleccionar cualquier número de elementos
- **`TextBox`** permite al usuario introducir y editar texto

API UI de alto nivel (II)

- Algunas restricciones de entrada de un `TextBox`:
  - **`UNEDITABLE`**: El texto no se puede editar
  - **`SENSITIVE`**: El texto no debe almacenarse
  - **`NON_PREDICTIVE`**: Indica el formato de entrada
  - **`INITIAL_CAPS_WORD`**: Mayúscula la letra inicial de cada palabra
  - **`INITIAL_CAPS_SENTENCE`**: Mayúscula la primera letra de cada frase
- `Form` contiene un número arbitrario de componentes (*items*)
  - El dispositivo controla la posición y el desplazamiento
  - Un `Item` sólo puede colocarse en un `Form`
- `Item` superclase de:
  - **`StringItem`** (etiqueta)
  - **`TextField`** (introducir texto)
  - **`Gauge`** (diagrama de barras)
  - **`ChoiceGroup`** (implementa la interfaz `Choice`, no `IMPLICIT`, añade `POPUP`)
  - **`ImageItem`**
  - **`DataField`** (fechas y horas)
  - **`Spacer`** (espacio en blanco)

Ejemplos API UI de alto nivel



Ver código: **`UIExample.zip`** (proyecto ejemplo WTK 2.5)

API UI de bajo nivel
Clase Canvas

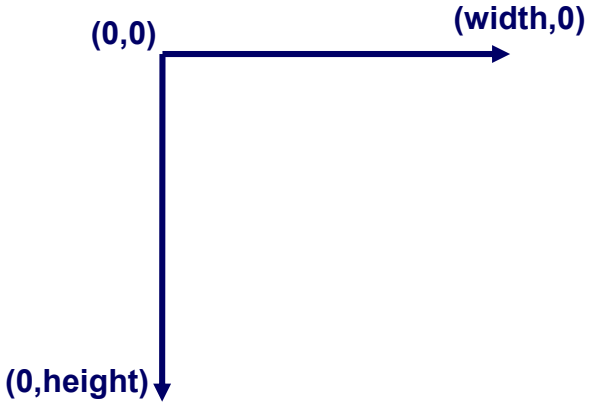
- Subclase abstracta de `Displayable` que permite realizar interfaces gráficos de bajo nivel en MIDP
- Necesario obtener el tamaño del `display` y programar teniendo en cuenta estas dimensiones:
  - **`int getWidth()`**, **`int getHeight()`**
- Método **`void paint(Graphics g):`**
  - Debe pintar todos los píxeles de la pantalla.
- Métodos para la gestión de eventos a bajo nivel
  - Entradas teclado
  - Puntero pantalla táctil
  - Cuando se visualiza el `Canvas` en el `display`:
    - **`showNotify:`** antes de visualizarlo
    - **`hideNotify:`** después de visualizarlo

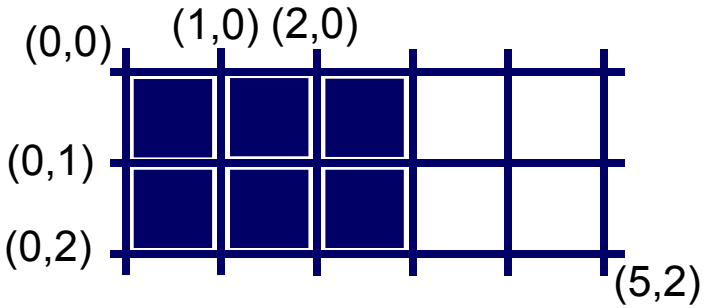
API UI de bajo nivel
Clase Canvas (II)

- Una clase que extienda `Canvas`:
  - Debe implementar obligatoriamente el método (abstracto) **`paint`**.
  - No es necesario que implemente todos los métodos relacionados con eventos a bajo nivel:
    - No son métodos abstractos y su implementación por defecto es vacía (no hacen nada).
    - El desarrollador sólo debe implementar los métodos correspondientes a los eventos que quiere gestionar.

API UI de bajo nivel
Clase `Graphics`

- Similar a `java.awt.Graphics`: geometría bidimensional
- Se pasa como parámetro al método **`paint`** de **`Canvas`**
- Sistema de coordenadas empieza en el extremo superior izquierdo





- Métodos para dibujar
  - `void drawLine(int x1, int y1, int x2, int y2)`
  - `void drawImage(Image img, int x, int y, int anchor)`
  - `void fillRect(int x, int y, int width, int height)`

Ejemplo API UI de bajo nivel

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

public class LineCanvasExample extends MIDlet {
  private Display display;
  public LineCanvasExample() {
    display=Display.getDisplay(this);
  }
  public void startApp() throws MIDletStateChangeException {
    display.setCurrent(new LineCanvas());
  }
  public void pauseApp() {
  }
  public void destroyApp(boolean unconditional) {
  }
```

Ejemplo API UI de bajo nivel (II)

```
class LineCanvas extends Canvas {
  public void paint(Graphics g) {
    // Tamaño del área de dibujo
    int width=this.getWidth();
    int height=this.getHeight();
    // Fondo de la pantalla blanco
    g.setColor(255,255,255); //0xFFFFFF
    g.fillRect(0,0,width,height);
    // Líneas en negro
    g.setColor(0,0,0);
    // Dibujamos dos líneas (ejes)
    g.drawLine(0,height,width,0);
    g.drawLine(0,0,width,height);
  }
 }
}
```

API UI de bajo nivel
Ejemplo `Canvas` y `Graphics` (III)

Ver código: **`LineCanvasExample.zip`** (proyecto WTK)

LineTest**`Canvas.zip`** (proyecto WTK)

API UI de bajo nivel
Eventos

- Eventos de teclado
  - Métodos que gestionan los eventos
    - `void keyPressed(int keyCode)`
    - `void keyReleased(int keyCode)`
    - `void keyRepeated(int keyCode)`
  - Cada tecla está definida por un código `KEY_NUM1`,..., `KEY_NUM9`, `KEY_STAR` (*) y `KEY_POUND` (#).
- Acciones de juego
  - Se definen una serie de códigos: `UP`, `DOWN`, `RIGHT`, `LEFT`, `FIRE`, `GAME_A`, `GAME_B`, `GAME_C` y `GAME_D`.
  - Mapeo a teclas con los métodos: `getKeyCode(int gameAction)` y `getGameAction(int keyCode)`
- Eventos de puntero
  - Métodos que gestionan los eventos
    - `void pointerPressed(int x, int y)`
    - `void pointerReleased(int x, int y)`
    - `void pointerDragged(int x, int y)`

API para juegos

- Introducida en MIDP 2.0 para mejorar el soporte de desarrollo de juegos 2D
- Principal ventaja: el `display` puede dividirse en capas de forma que se puede tratar cada una de ellas de forma independiente
- Paquete:
  - **`javax.microedition.lcdui.game`**
- Proporciona cinco nuevas clases:
  - **`GameCanvas`**
  - **`Layer`**
  - **`LayerManager`**
  - **`Sprite`**
  - **`TiledLayer`**

API para juegos
Clase Canvas

- Para programar juegos en MIDP 1.0 se empleaba la clase **Canvas**, y los programas tenían una estructura similar a la siguiente:

```
public class JuegoCanvas extends Canvas implements Runnable {
  public void run() {
    while (true) {
      // Modifica el estado del juego
      repaint();
      // Espera un tiempo
    }
  }
  public void paint(Graphics g) {
    // Pintar la pantalla
  }
  protected void keyPressed(int keyCode) {
    // Responder a eventos de pulsación de teclas
  }
}
```

API para juegos
Clase `GameCanvas`

- Problemas en MIDP 1.0:
  - Tres hilos distintos:
    - Controla el estado del juego
    - Pinta la pantalla
    - Gestiona eventos
  - Animación e interacción defectuosa
- **`GameCanvas`** permite:
  - Tener un único hilo para controlar el juego, pintar en pantalla y gestionar eventos
  - Construir sobre un buffer lo que se quiere representar en pantalla antes de visualizarlo: "*off-screen buffer*"
  - Emplear un mecanismo de "*polling*" para saber el estado de las teclas

API para juegos
Clase GameCanvas (II)

- Con **GameCanvas** los programas tienen una estructura similar a la siguiente:

```
public class JuegoCanvas extends GameCanvas
                                  implements Runnable {
  public void run() {
    Graphics g = getGraphics();
    while (true) {
      // Se actualiza el estado del juego
      int keyState = getKeyStates();
      // Responde a eventos del teclado
      // Pintar en pantalla
      flushGraphics();
      // Esperar un tiempo
    }
  }
}
```

Devuelve el objeto **Graphics** correspondiente al "off-screen buffer".

Devuelve un entero (1 pulsada y 0 no pulsada).

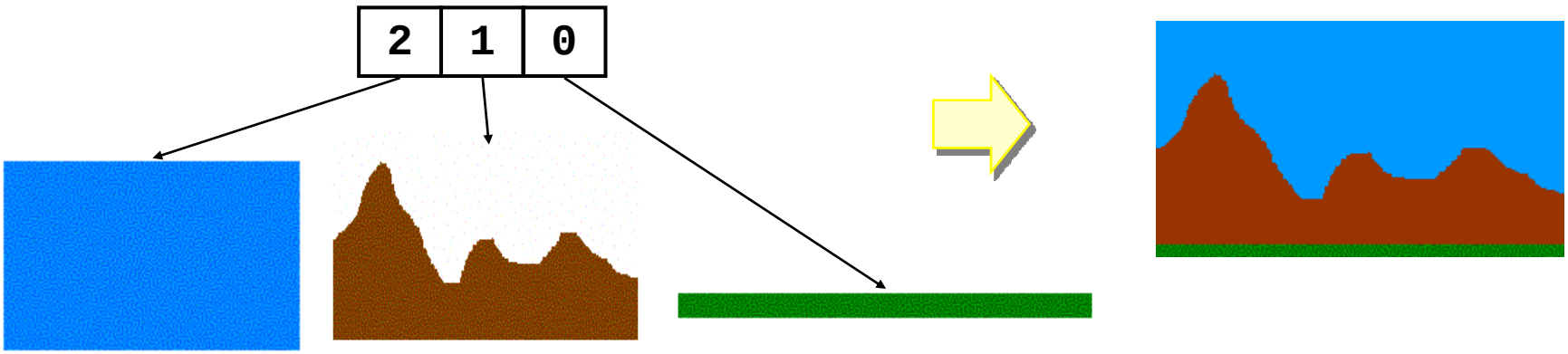
Permite volcar el contenido del "off-screen buffer" en la pantalla del dispositivo

API para juegos
Clase Layer

- Clase abstracta que representa un elemento visual
- Tiene tres propiedades:
  - Posición
  - Tamaño
  - Si es visible o no
- Métodos de esta clase permiten obtener los valores de estas propiedades.
- El programador puede extender esta clase para ofrecer una funcionalidad específica:
  - Siempre debe realizar una implementación del método **`paint`**
- API MIDP 2.0 define dos subclases de **`Layer`**:
  - **`TiledLayer`**
  - **`Sprite`**

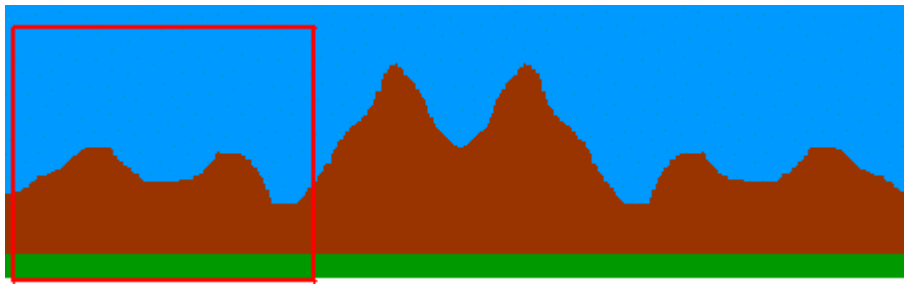
API para juegos
Clase LayerManager

- Clase que permite trabajar con varios objetos **`Layer`** en una aplicación
- Mantiene una lista ordenada de **`Layers`**:
  - Para cada uno de ellos se puede obtener sus propiedades
  - Los diferentes **`Layer`** se insertan en la lista según su profundidad:
    - Posición 0 indica que es el **`Layer`** más próximo al usuario
    - Última posición es el **`Layer`** sobre el que se superponen todos los demás



API para juegos
Clase `LayerManager` (II)

- Ventana visible:
  - Se puede indicar qué parte del **`LayerManager`** se presenta en pantalla
  - Muy útil cuando se quieren generar barridos o panorámicas de una escena
  - La ventana visible se establece a través del método:
    - **`setViewWindow(int x, int y, int width, int height)`**

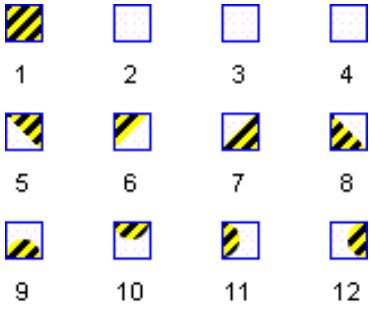


Ventana visible

API para juegos
Clase `TiledLayer`

- Clase que extiende de **`Layer`**
- Representa un elemento visual compuesto por un conjunto de "baldosas"
- Nos permite obtener como un rompecabezas donde podremos colocar las diferentes piezas en distintas posiciones

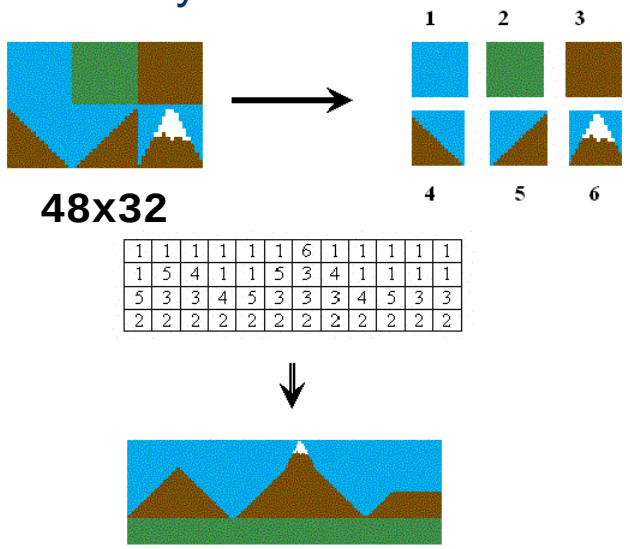
64x48



16x16

API para juegos
Clase `TiledLayer` (II)

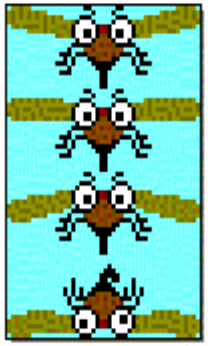
- Cuando se crean las baldosas se crea también una matriz de un determinado número de posiciones:
  - Cada posición hace referencia a una de las baldosas.
  - Las posiciones de la matriz empiezan en 0 y las baldosas se numeran a partir del 1.

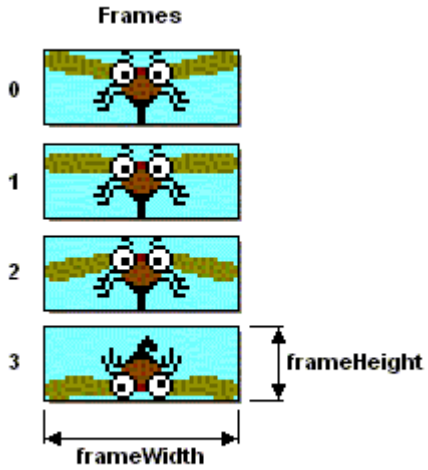



```
Image image = Image.createImage("/board.png");
TiledLayer tiledLayer = new TiledLayer(12, 4, image, 16, 16);
```

API para juegos
Clase `Sprite`

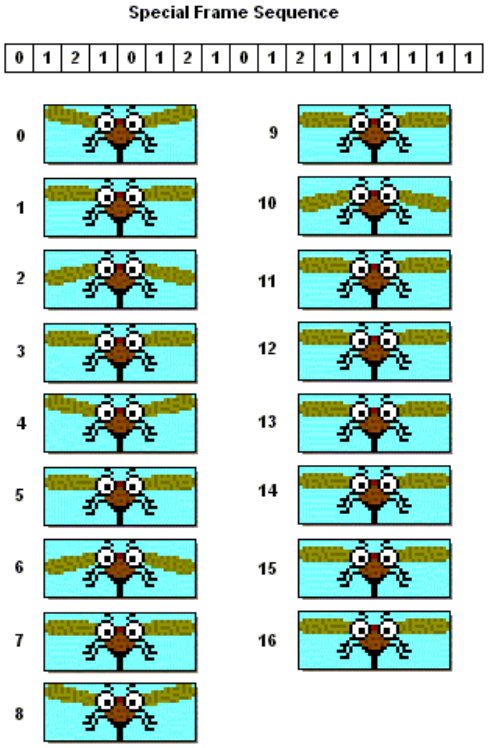
- Clase que extiende de **`Layer`**
- Divide una imagen en una serie de *frames* que se pueden reproducir en una determinada secuencia y de esta manera realizar animaciones

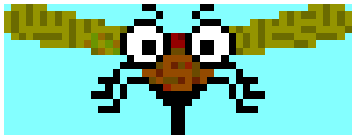





```
Image image = Image.createImage("/board.png");
Sprite sprite = new Sprite(image, frameWidth, frameHeight);
```

API para juegos
Clase `Sprite` (II)





```
int[] sequence = {0,1,2,1,0,1,2,1,0,1,2,1,1,1,1,1,1}
sprite.setFrameSequence(sequence);
```

API para juegos
Ejemplo LayerManager

`Games-WormGame` (proyecto WTK) **`Games-PushPuzzle`** (proyecto WTK) **`LayerManagerExample.zip`**

LightWeight User Interface Toolkit (LWUIT)

- Proporciona interfaces sofisticadas similar a `Swing`
- LWUIT ofrece
  - un conjunto de componentes: `Form`, `Label`, `Button`, `CheckBox`, `ComboBox`, `TextArea`, ...
  - "*layouts*" flexibles: `FlowLayout`, `BorderLayout`, `BoxLayout`, `GridLayout`
  - tema y estilos: `Style`, `UIManager.getInstance().setThemeProps`
  - transiciones animadas entre pantallas: `CommonTransitions`
  - un simple mecanismo de gestión de eventos: interfaces *listeners* (`ActionListener`)
- Librería externa que debe ser añadida al MIDlet Suite
  - `http://lwuit.java.net/`

Ejemplo HelloWord LWUIT

```
import javax.microedition.midlet.*;
import com.sun.lwuit.*;
import com.sun.lwuit.events.*;

public class HelloLWUITMidlet extends MIDlet
                              implements ActionListener {

 public void startApp() {
    Display.init(this);
    Form f = new Form("Hello, LWUIT!");
    f.show();
    Command exitCommand = new Command("Exit");
    f.addCommand(exitCommand);
    f.setCommandListener(this);
 }

 public void pauseApp() {}
 public void destroyApp(boolean unconditional) {}
 public void actionPerformed(ActionEvent ae) {
           notifyDestroyed();
 }
}
```

Índice

- Introducción
- MIDlets
  - Conceptos básicos
  - Desarrollo y despliegue
- Librerías de MIDP
- Interfaz de usuario
- **Almacenamiento persistente**
- Conectividad

Almacenamiento persistente

- Definido en el paquete **`javax.microedition.rms`**
- API independiente del dispositivo
- Base de datos sencilla orientada a registros (RMS)
  - Los **registros** (**`Record`**) se guardan en almacenes de registro y se identifican unívocamente mediante un identificador de tipo **`int`**
    - array de bytes de tamaño variable
  - Los **almacenes de registros** (**`RecordStore`**) se comparten entre MIDlets de un mismo MIDlet suite

Registro	Datos
1	Datos 1
2	Datos 2
...	...

- Soporta enumeración, ordenamiento y filtrado
- Actualización atómica de registros

Clase RecordStore

- Un *record store* es una colección de *records*
- Reglas:
  - Nombre único en un mismo MIDlet Suite
  - El nombre puede ser una combinación de 32 caracteres ("*case sensitive*")
  - Se almacenan en el mismo espacio de nombres de un MIDlet Suite
- Operaciones:
  - **Crear/abrir:** `static RecordStore openRecordStore (String recordStoreName, boolean createIfNecessary)`
  - **Cerrar** (después de utilizarse): `void closeRecordStore()`
  - **Borrar** (debe estar cerrado): `static void deleteRecordStore(String recordStoreName)`
- La cabecera proporciona la siguiente información sobre
  - Número de registros, versión, fecha de la última modificación, etc.

RMS
Interfaz `RecordEnumeration`

- Después de borrar `records` sus identificadores ya no son consecutivos
- Para recorrerlos se proporciona la clase **`RecordEnumeration`**:
  - Lista doblemente enlazada ⇨ cada nodo representa un `record`.
  - Se obtiene a través del método:
    - `RecordEnumeration enumerateRecords(RecordFilter f, RecordComparator comparator, boolean keepUpdated)`
  - Métodos:
    - **`void reset()`**: puntero al primer elemento de la lista.
    - **`int nextRecordId()`**: ID del siguiente elemento de la lista.
    - **`int previousRecordId()`**: ID del anterior elemento de la lista.
  - Se define como interfaz pero los fabricantes deben realizar una implementación de ella: para los desarrolladores es una clase.

Interfaces `RecordFilter` y `RecordComparator`

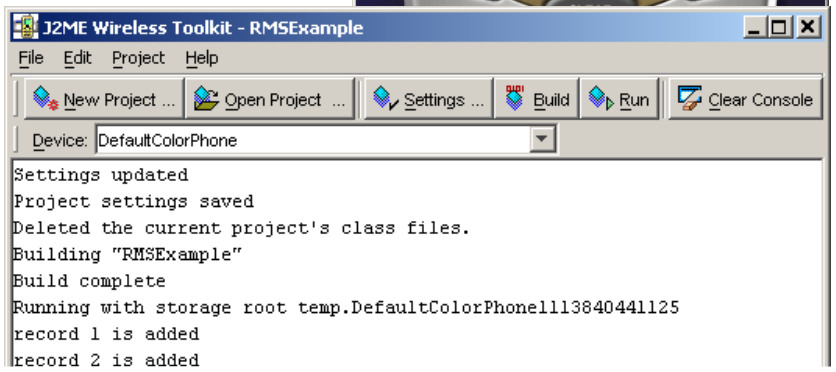
- Permiten ordenar o clasificar los `records` en un `RecordStore` según algún criterio
- Interfaz **`RecordFilter`**:
  - **`boolean matches(byte[] candidate)`**
  - La aplicación determina si el `record` (**`candidate`**) verifica el criterio de selección
- Interfaz **`RecordComparator`**:
  - **`int compare(byte[] rec1, byte[] rec2)`**
  - Clasificación de records por algún criterio (ej. fecha creación)
  - Devuelve el orden de `records` (**`rec1`** y **`rec2`**):
    - **`PRECEDES`**, **`FOLLOWS`**, **`EQUIVALENT`**

Interfaz `RecordListener`

- Monitorizar cambios en `RecordStores`
- Gestión de eventos mediante los métodos:
  - `void recordAdded(RecordStore recordStore, int recordId)`
  - `void recordChanged(RecordStore recordStore, int recordId)`
  - `void recordDeleted(RecordStore recordStore, int recordId)`
- Para añadir y borrar `listeners`:
  - `void addRecordListener(RecordListener listener)`
  - `void removeRecordListener(RecordListener l)`

Ejemplo RMS

```
public void startApp() throws MIDletStateChangeException {
   RecordStore rs = null;
   try {
     rs = RecordStore.openRecordStore("file1", true);
     byte data[] = new byte[4];
     for ( int j=0; j<2; j++) {
       int i = rs.getNextRecordID();
       ByteArrayOutputStream baos = new ByteArrayOutputStream();
       DataOutputStream outputStream = new DataOutputStream(baos);
       outputStream.writeInt(i);
       byte[] b = baos.toByteArray();
       System.out.println("record " + rs.addRecord(data,0,4) +
                          " is added");
     }
   } catch (Exception e) {}
   finally{
     try {
          rs.closeRecordStore();
     } catch (Exception e) {}
   }
   destroyApp(true);
   notifyDestroyed();
 }
```



Ver código: **`RecordStoreTest.java`** – **`RMSExample.zip`** (proyecto WTK)

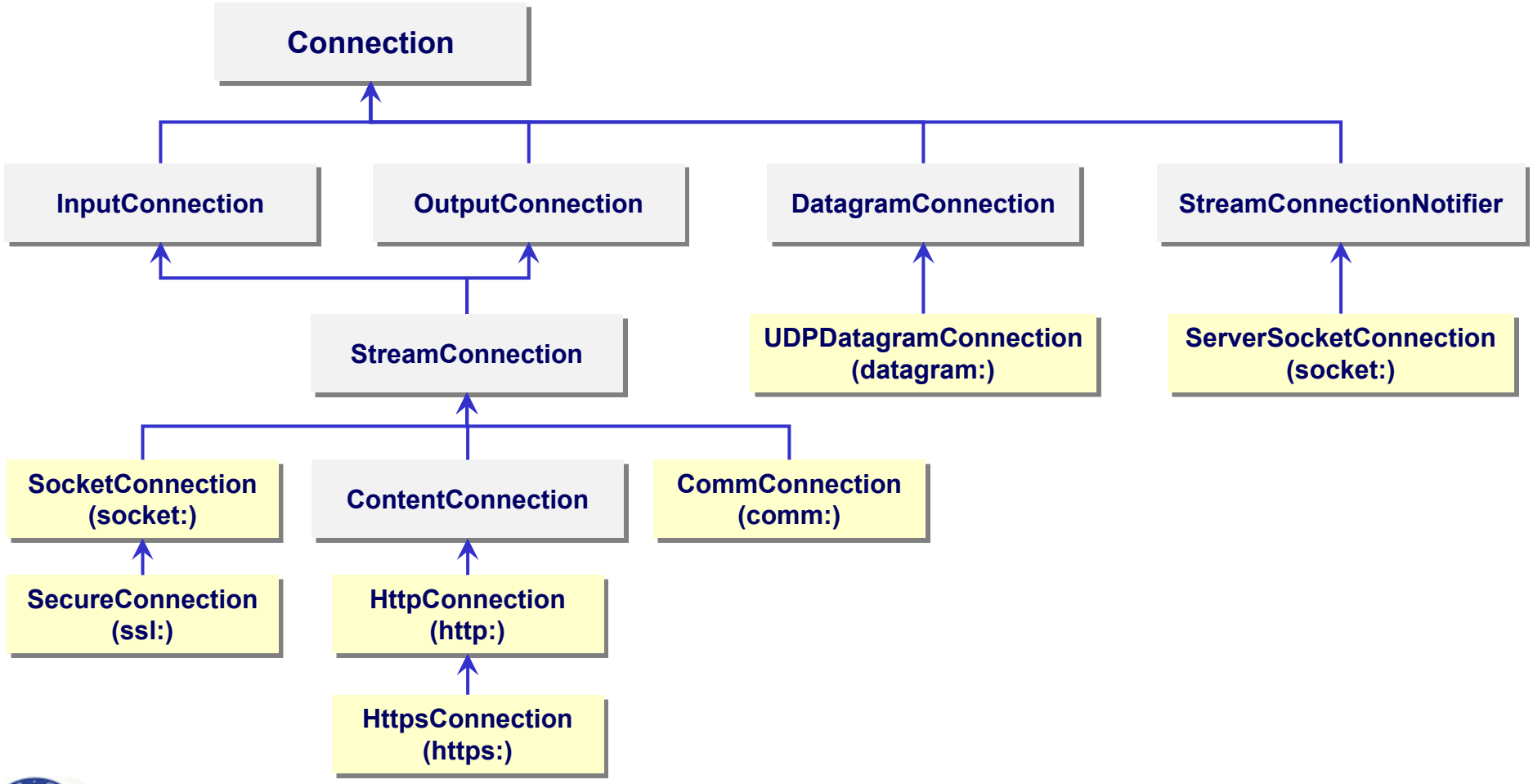
Índice

- Introducción
- MIDlets
  - Conceptos básicos
  - Desarrollo y despliegue
- Librerías de MIDP
- Interfaz de usuario
- Almacenamiento persistente
- **Conectividad**

Conectividad

- Implementa el *Generic Connection Framework* de CLDC definido en el paquete **`javax.microedition.io`**:
  - requiere soporte de conexiones HTTP (RFC 2616) cliente
- Añade e implementa el interfaz **`HttpConnection`**, hereda directamente del interfaz **`ContentConnection`**
- La implementación del interfaz **`DatagramConnection`**, definido en CLDC es opcional, pero recomendable
- Clase `Connector`
  - método open:
    - **`Connector.open("http://www.it.uc3m.es/")`**

Interfaces de conexión en MIDP



Clase HTTPConnection

- Tres estados de la conexión: *Setup*, *connected* y *closed*
- Transición de *Setup* a *Connected* motivada por cualquier método que requiera enviar o recibir datos.

Métodos	Estado
`setRequestMethod` `setRequestProperty`	*Setup*

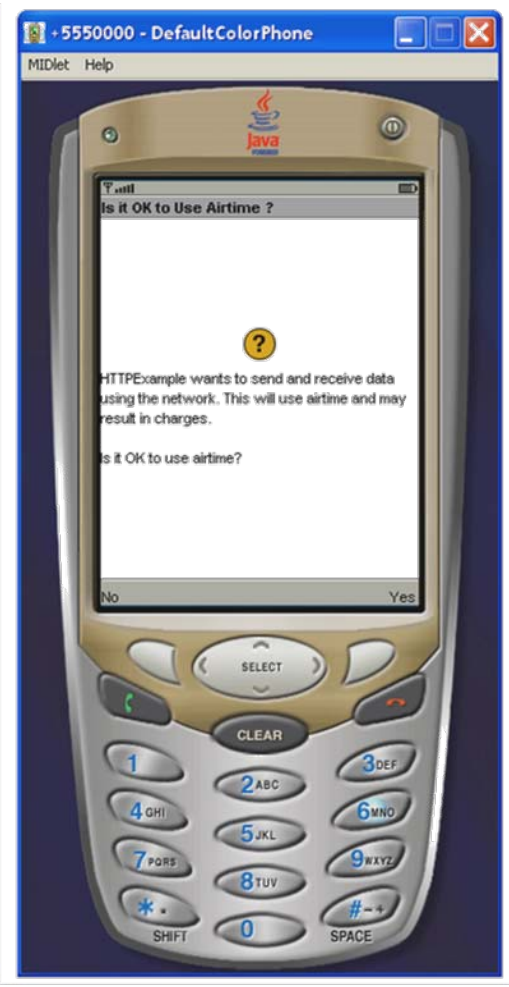
`openInputStream`
`openDataInputStream`
`getLength, getType, getEncoding`
`getHeaderField`
`getResponseCode`
`getResponseMessage`
`getHeaderFieldInt`
`getHeaderFieldDate`
`getExpiration`
`getDate`
`getLastModified`
`getHeaderFieldKey`
	Connected

`close`
`getRequestMethod`
`getRequestProperty`
`getURL, getProtocol, getHost, getFile, getRef, getPort, getQuery`
	Setup o connected

Ejemplo HTTPConnection

```
private void download (String url) throws IOException {
   StringBuffer sb = new StringBuffer();
   InputStream is = null;
   HttpConnection c = null;
   TextBox t = null;
   try {
     long len = 0 ;
     int ch = 0;
     c = (HttpConnection)Connector.open(url);
     is = c.openInputStream();
     while ((ch = is.read()) != -1) {
       sb.append((char)ch); }
     t = new TextBox("Hola...", sb.toString(), 1024, 0);
   } finally {
       if (is != null) is.close();
       if (c != null) c.close();
   }
```

Ejemplo `HTTPConnection`



Ver código: **`HTTPTest.java`** – **`HTTPExample.zip`** (proyecto WTK)