



FORMACIÓN Y TECNOLOGÍAS JAVA
UNIVERSIDAD DE ALICANTE

PROGRAMACIÓN DE DISPOSITIVOS MÓVILES

Transparencias



Sesión 1: Introducción a los MIDs

Índice



- **Características de los dispositivos**
- **Arquitectura de J2ME**
- **Aplicaciones MIDP**
- **Construcción de aplicaciones**
- **Desarrollo con Eclipse**

Introducción a los MIDs



- **Características de los dispositivos**
- **Arquitectura de J2ME**
- **Aplicaciones MIDP**
- **Construcción de aplicaciones**
- **Desarrollo con Eclipse**

Tipos de dispositivos



- Dispositivos móviles de información
 - MIDs: Mobile Information Devices
 - Teléfonos móviles, PDAs, etc
- Descodificadores de TV (*set top boxes*)
- Electrodomésticos
- Impresoras de red
- Routers
- etc

sin interfaz



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-4

Características de los MIDs



96x65
Monocromo
164kb



101x64
Monocromo
150kb



178x201
4096 colores
1,4mb



128x128
4096 colores
200kb



640x200
4096 colores
8mb



240x320
65536 colores
64mb

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-5

Redes de telefonía celular



- 1G: Red analógica
 - Sólo voz
 - Red TACS en España
 - Distintos países usan distintas redes
 - No permite itinerancia
- 2G: Red digital
 - Voz y datos
 - GSM (*Global System for Mobile communications*) en toda Europa
 - Permite itinerancia
 - Red no IP
 - Protocolos WAP (WSP)
 - Un gateway conecta la red móvil (WSP) a la red Internet (TCP/IP)
 - Conmutación de circuitos (*Circuit Switched Data, CSD*)
 - 9'6kbps
 - Se ocupa un canal de comunicación de forma permanente
 - Se cobra por tiempo de conexión

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-6

Redes de telefonía celular (2)



▪ 2,5G: GPRS (*General Packet Radio Service*)

- Transmisión de paquetes
 - No ocupa un canal de forma permanente
 - Hasta 144kbps teóricamente (40kbps en la práctica)
 - Cobra por volumen de información transmitida
- Se implementa sobre la misma red GSM

▪ 3G: Banda ancha

- Red UMTS (*Universal Mobile Telephony System*)
 - Itinerancia global
- Entre 384kbps y 2Mbps
- Servicios multimedia
 - Videoconferencia, TV, música, etc
- Transmisión de paquetes
- Requiere nueva infraestructura

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDe-7

Paradigmas de programación en móviles



▪ Documentos Web

- Descarga documentos y los muestra en un navegador
- Formato adecuado para móviles (WML, XHTML, ...)
- Requiere conectar a red para descargar cada documento
- Velocidad de descarga lenta
- Documentos pobres (deben servir para todos los móviles)

▪ Aplicaciones locales

- La aplicación se descarga en el móvil
- Se ejecuta de forma local
- Interfaz de usuario más flexible
- Puede funcionar sin conexión (minimiza el tráfico)

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDe-8

Documentos Web



▪ WML (Wireless Markup Language)

- Forma parte de los protocolos WAP (Capa de aplicación, WAE)
- Lenguaje de marcado dirigido a móviles
- Requiere aprender un nuevo lenguaje diferente a HTML
- Documentos muy pobres

▪ iMode

- Documentos escritos en cHTML (HTML compacto)
 - Subconjunto de HTML
 - Propietario de NTT DoCoMo
- Sobre la red japonesa PDC-P (extensión de la red japonesa PDC, similar a GSM, para transmisión de paquetes)
 - En Europa se lanza sobre GPRS

▪ XHTML MP

- Versión reducida de XHTML dirigido a móviles
- A diferencia de cHTML, se desarrolla como estándar

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDe-9

Aplicaciones locales



- **Sistema operativo**
 - Symbian OS, Palm OS, Windows Pocket PC, etc
 - Poco portable
 - Requiere aprender nuevas APIs
- **Runtime Environments**
 - BREW
 - Soportado por pocos dispositivos
 - Requiere aprender una nueva API
 - J2ME
 - Soportado por gran cantidad de dispositivos
 - Existe una gran comunidad de desarrolladores Java

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-10

Conectividad de los MIDs



- **Los dispositivos deben conectarse para descargar las aplicaciones**
 - Over The Air (OTA)
 - Conexión a Internet usando la red móvil (GSM, GPRS, UMTS)
 - Cable serie o USB
 - Conexión física
 - Infrarrojos
 - Los dispositivos deben verse entre si
 - Bluetooth
 - Ondas de radio (10 metros de alcance)
 - Alta velocidad (723kbit/s)

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-11

Introducción a los MIDs



- **Características de los dispositivos**
- **Arquitectura de J2ME**
- **Aplicaciones MIDP**
- **Construcción de aplicaciones**
- **Desarrollo con Eclipse**

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-12

Java 2 Micro Edition



- Edición de la plataforma Java 2 para dispositivos móviles
- Independiente de la plataforma
 - Adecuado para programar dispositivos heterogéneos
- Gran comunidad de desarrolladores Java
 - Los programadores Java podrán desarrollar aplicaciones para móviles de forma sencilla
 - No hace falta que aprendan un nuevo lenguaje
- Consiste en un conjunto de APIs
 - Una sola API es insuficiente para la variedad de tipos de dispositivos existente
 - Cada API se dedica a una distinta familia de dispositivos

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDe-13

Capas de J2ME



- Configuraciones
 - API común para todo un gran conjunto de dispositivos
 - Elementos básicos del lenguaje
- Perfiles
 - API que cubre las características propias de una familia de dispositivos concreta
 - P.ej, para acceder a la pantalla de los teléfonos móviles
- Paquetes opcionales
 - APIs para características especiales de ciertos dispositivos
 - P.ej, para acceder a la cámara de algunos teléfonos móviles

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDe-14

APIs de J2ME



Configuraciones

- CDC: Dispositivos conectados
 - Sobre JVM
- CLDC: Dispositivos conectados limitados
 - Sobre KVM (limitada)
 - Paquetes:
 - java.lang
 - java.io
 - java.util
 - javax.microedition.io

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDe-15

Perfiles CDC



- **Dispositivos con memoria del orden de los MB**
 - Se recomienda por lo menos 2MB
 - PDAs de gama alta
- **Se ejecuta sobre CVM (equivalente a JVM)**
- **FP (Foundation Profile)**
 - Dispositivos sin interfaz: impresoras de red, routers
- **FBP (Foundation Basis Profile)**
 - Dispositivos con interfaz: descodificadores de TV
 - Sólo componentes ligeros de AWT
- **PP (Personal Profile)**
 - Incluye la especificación completa de AWT
 - Dispositivos con interfaz gráfica nativa
 - Adecuado para migrar antiguos sistemas PersonalJava

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-16

CLDC



- **Dispositivos con memoria del orden de los KB**
 - Puede funcionar con sólo 128KB
 - Teléfonos móviles y PDAs de gama baja
- **Se ejecuta sobre KVM (*Kilobyte Virtual Machine*)**
- **Muy limitada, para poder funcionar con escasos recursos**
 - P.ej, no soporta reales (tipos float y double)
- **Perfil MIDP**
 - Dispositivos móviles de información (MIDs)
 - Paquetes:
 - javax.microedition.lcdui
 - javax.microedition.midlet
 - javax.microedition.rms

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-17

Paquetes opcionales



- **Wireless Messaging API (WMA)**
 - Envío y recepción de mensajes cortos (SMS)
- **Mobile Media API (MMAPI)**
 - Multimedia, reproducción y captura de video y audio
- **Bluetooth API**
 - Permite establecer conexiones vía Bluetooth
- **J2ME Web Services**
 - Invocación de servicios web desde dispositivos móviles
- **Mobile 3D Graphics**
 - Permite incorporar gráficos 3D a las aplicaciones y juegos

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-18

Más paquetes opcionales



- **Location API**
 - Localización física del dispositivo (GPS)
- **Security and Trust Services API**
 - Servicios de seguridad: encriptación, identificación, autenticación
- **PDA Optional Packages**
 - Consta de dos librerías:
 - *FileConnection* (FC): librería para acceso al sistema de ficheros (FC)
 - *Personal Information Management* (PIM): librería para el acceso a la información personal almacenada (agenda, contactos, etc)
- **Content Handler API**
 - Integración con el entorno de aplicaciones del dispositivo. Permite utilizar otras aplicaciones para abrir diferentes tipos de contenidos
- **SIP API**
 - Permite utilizar *Session Initiation Protocol*. Este protocolo se usa para conexiones IP multimedia (juegos, videoconferencia, etc)

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-19

JTWI



- **JTWI (*Java Technologies for Wireless Industry*)**
- **Especificación que trata de definir una plataforma estándar para el desarrollo para móviles**
 - Aumentar la compatibilidad entre los dispositivos
- **Las tareas de esta especificación son:**
 - Definir las APIs que deben estar presentes en los dispositivos.
 - CLDC 1.0, MIDP 2.0, WMA 1.1
 - Opcionalmente: CLDC 1.1, MMAPI
 - Evitar que se utilicen APIs adicionales que reducen la compatibilidad.
 - Aclarar aspectos confusos en las especificaciones de estas APIs.

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-20

Introducción a los MIDs



- **Características de los dispositivos**
- **Arquitectura de J2ME**
- **Aplicaciones MIDP**
- **Construcción de aplicaciones**
- **Desarrollo con Eclipse**

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-21

MIDlets



- Las aplicaciones para dispositivos MIDP se denominan *MIDlets*
- Estas aplicaciones se distribuyen como una *suite de MIDlets*, que se compone de:
 - Fichero JAD
 - Fichero ASCII
 - Descripción de la aplicación
 - Fichero JAR
 - Aplicación empaquetada (clases y recursos)
 - Contiene uno o más MIDlets
 - Contiene un fichero `MANIFEST.MF` con información sobre la aplicación (algunos datos son replicados del fichero JAD).

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-22

Fichero JAD



- Ejemplo de fichero JAD:

```
MIDlet-Name: SuiteEjemplos
MIDlet-Version: 1.0.0
MIDlet-Vendor: Universidad de Alicante
MIDlet-Description: Aplicaciones de ejemplo para moviles.
MIDlet-Jar-Size: 16342
MIDlet-Jar-URL: ejemplos.jar
```

- En un dispositivo real es importante que `MIDlet-Jar-Size` contenga el tamaño real del fichero JAR
- Si publicamos la aplicación en Internet, `MIDlet-Jar-URL` deberá apuntar a la URL de Internet donde se encuentra publicado el fichero JAR.

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-23

Fichero MANIFEST.MF



- Ejemplo de fichero `MANIFEST.MF`:

```
MIDlet-Name: SuiteEjemplos
MIDlet-Version: 1.0.0
MIDlet-Vendor: Universidad de Alicante
MIDlet-Description: Aplicaciones de ejemplo para moviles.
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
MIDlet-1: Snake, /icons/snake.png, es.ua.j2ee.serpiente.SerpMIDlet
MIDlet-2: TeleSketch, /icons/ts.png, es.ua.j2ee.ts.TeleSketchMIDlet
MIDlet-3: Panj, /icons/panj.png, es.ua.j2ee.panj.PanjMIDlet
```

- Si el dispositivo real no soporta la configuración o el perfil indicados, se producirá un error en la instalación.

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-24

Software gestor de aplicaciones



- Los dispositivos móviles con soporte para Java tienen instalado un software gestor de aplicaciones
 - AMS: *Application Management Software*
- Gestiona las aplicaciones Java:
 - Descarga
 - Descarga primero el fichero JAD y muestra los datos de la aplicación
 - Si la aplicación es compatible y el usuario acepta, descarga el JAR
 - Instalación
 - Actualización
 - Desinstalación
 - Ejecución
 - Es el contenedor que da soporte a los MIDlets
 - Contiene la KVM sobre la que se ejecutarán las aplicaciones
 - Soporta la API de MIDP
 - Controla el ciclo de vida de los MIDlets que ejecuta

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-25

Introducción a los MIDs



- Características de los dispositivos
- Arquitectura de J2ME
- Aplicaciones MIDP
- Construcción de aplicaciones
- Desarrollo con Eclipse

Programación de Dispositivos Móviles

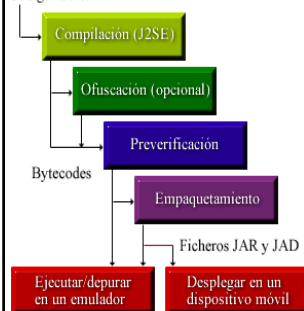
© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-26

Pasos del proceso



Código fuente



- Compilar
 - Utilizar como clases del núcleo la API de MIDP
- Ofuscar (optativo)
 - Reducir tamaño de los ficheros
 - Evitar descompilación
- Preverificar
 - Reorganizar el código para facilitar la verificación a la KVM
 - Comprobar que no se usan características no soportadas por KVM
- Empaquetar
 - Crear ficheros JAR y JAD
- Probar
 - En emuladores o dispositivos reales

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-27

Kits de desarrollo



- Incluyen las APIs necesarias
 - MIDP y APIs adicionales
- Incluyen herramientas que no están en Java 2 SDK
 - Preverificador
- Incluye emuladores para probar las aplicaciones
 - Imitan teléfonos genéricos o modelos reales
- Facilitan el proceso de construcción de aplicaciones
 - Entorno de creación de aplicaciones
- Es necesario contar con Java 2 SDK para compilar y empaquetar

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-28

Sun Wireless Toolkit (WTK)



- Kit de desarrollo genérico.
 - Se puede integrar con emuladores proporcionados por terceros (Nokia, Ericsson, etc).
- Versiones:
 - WTK 1.0.4: Sólo soporta MIDP 1.0
 - WTK 2.0: Sólo soporta MIDP 2.0
 - APIs opcionales: WMA, MMAPI
 - WTK 2.1: Soporta MIDP 1.0 y MIDP 2.0
 - Puede generar aplicaciones JTWI
 - APIs opcionales: WMA, MMAPI, WSA
 - WTK 2.2: Igual que WTK 2.1, añadiendo:
 - APIs opcionales: M3G, Bluetooth
 - WTK 2.5: Igual que WTK 2.2, añadiendo:
 - APIs opcionales: SIP, CHAPI, PDA, SATSA, MPay, SVG, AMS, I18N, y Location API
 - Cumple con Mobile Service Architecture (MSA)

Programación de Dispositivos Móviles

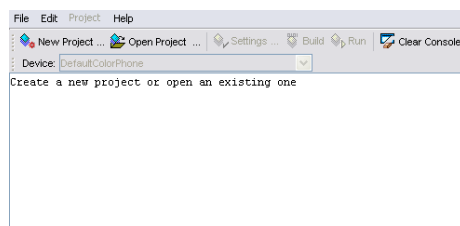
© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-29

Ktoolbar



- WTK contiene la herramienta **ktoolbar** para automatizar la creación de aplicaciones



Programación de Dispositivos Móviles

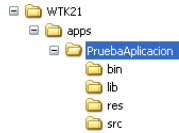
© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-30

Aplicaciones de WTK



- Se almacenan en el directorio `${WTK_HOME}/apps`
- Existe un subdirectorio por aplicación
- Cada aplicación se organiza en los siguientes subdirectorios:



src: Código fuente
res: Recursos (ficheros de datos, imágenes, ...)
lib: Librerías (jar)
bin: Aquí se generan los ficheros JAD y JAR
classes: Clases intermedias generadas (temporal)

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-31

Crear una aplicación



- Pulsar **New Project ...**

Project Name: PruebaAplicacion
MIDlet Class Name: es.ua.j2ee.prueba.MIDletPrueba
Create Project Cancel

- Editar los datos para los ficheros JAD y JAR (**MANIFEST.MF**)

API Selection Required Optional User Defined MIDlets Push Registry Permissions

Key	Value
MIDlet-Jar-Spec	1.0.0
MIDlet-Jar-URL	PruebaAplicacion.jar
MIDlet-Name	PruebaAplicacion
MIDlet-Vendor	Unknown
MIDlet-Version	1.0
MicroEdition-Configuration	CLDC-1.0
MicroEdition-Profiles	MDP-1.0

OK Cancel Down Remove OK Cancel

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-32

Prueba de la aplicación



- Construir la aplicación
 - Pulsar sobre **Project** → **Build**
- Ejecutar en un emulador
 - Seleccionar un emulador del cuadro desplegable
 - Pulsar sobre **Project** → **Run**



- Distribuir la aplicación
 - Pulsar sobre **Project** → **Package** → **Create package**

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-33

Provisionamiento OTA



- Podemos simular la descarga real de la aplicación
- Provisionamiento OTA: *Project > Run via OTA*



Programación de Dispositivos Móviles

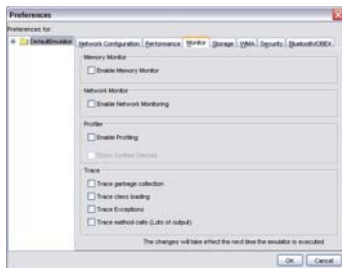
© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-34

Optimización



- Podemos activar monitores para controlar:
 - Tráfico en la red
 - Ocupación de memoria



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-35

Introducción a los MIDs



- Características de los dispositivos
- Arquitectura de J2ME
- Aplicaciones MIDP
- Construcción de aplicaciones
- Desarrollo con Eclipse

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-36

Integración de J2ME y Eclipse



- Eclipse no incluye soporte “de serie” para J2ME
- Tenemos varias opciones
 - Utilizarlo sólo como editor de código
 - Construir las aplicaciones con WTK
 - Utilizar tareas de *Ant* para el desarrollo con J2ME
 - Utilizar librería de tareas Antenna
 - Añadir *plugins* para trabajar con aplicaciones J2ME
 - Como por ejemplo EclipseME

Programación de Dispositivos Móviles

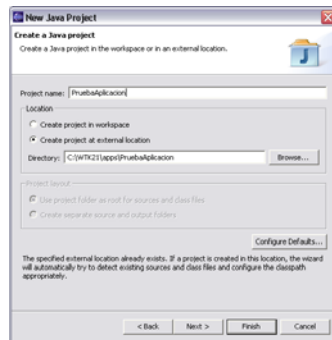
© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-37

Creación de un proyecto



- Asignar un nombre al proyecto
- Utilizar como directorio del proyecto el directorio de la aplicación creada con WTK
- Pulsar sobre *Next >*



Programación de Dispositivos Móviles

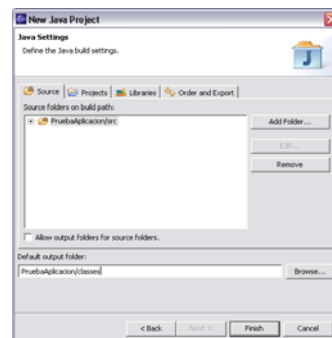
© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-38

Establecer directorios



- Establecer como directorio de fuentes el directorio *src* de la aplicación
- Establecer como directorio de salida el directorio *classes* de la aplicación



Programación de Dispositivos Móviles

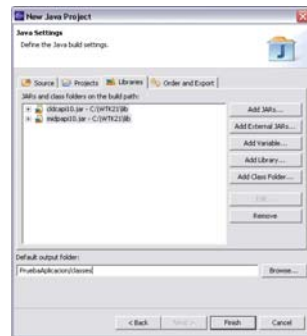
© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-39

Establecer librerías



- Eliminar la librería de clases de J2SE
- Añadir la librería de CLDC (`cldcapi10.jar`)
- Añadir la librería de MIDP (`midpapi10.zip`)



Programación de Dispositivos Móviles

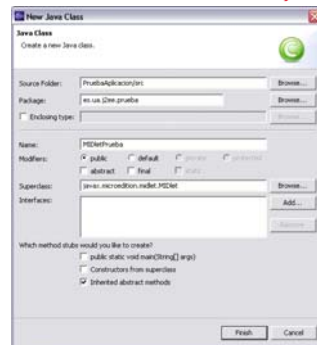
© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-40

Crear un MIDlet



- Crear una clase que herede de `MIDlet`
- Introducir el código necesario en la clase creada
- Crear todas las clases adicionales que sean necesarias para la aplicación
- Grabar el código editado
- Construir la aplicación desde WTK



Programación de Dispositivos Móviles

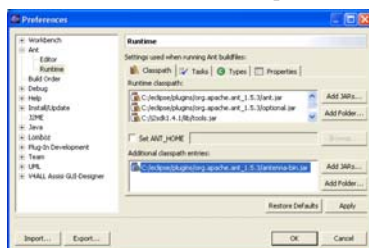
© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-41

Tareas de Antenna



- *Antenna* es una librería de tareas de *Ant* para construir aplicaciones J2ME
- Podemos utilizar esta librería desde Eclipse



Programación de Dispositivos Móviles

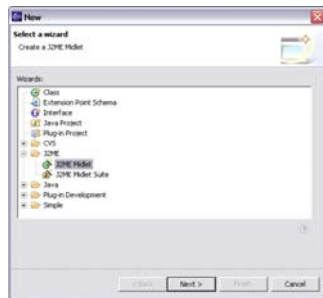
© 2006 Depto. Ciencia Computación e IA

Introducción a los MIDs-42

EclipseME



- *Plug-in* de Eclipse
- Nos permite crear aplicaciones J2ME con este entorno de forma integrada
 - No es necesario utilizar ninguna herramienta externa
- Podemos:
 - Crear una suite de MIDlets
 - Añadir MIDlets a la suite
 - Editar el fichero JAD mediante un editor de JAD incorporado
 - Ejecutar la aplicación directamente en un emulador





Sesión 2: Java para MIDs. MIDlets

Índice



- **Características de CLDC**
- **Números reales**
- **Temporizadores**
- **Serialización de objetos**
- **Acceso a los recursos**
- **MIDlets**

Java para MIDs



- **Características de CLDC**
- **Números reales**
- **Temporizadores**
- **Serialización de objetos**
- **Acceso a los recursos**
- **MIDlets**

Configuración CLDC



- **Características básicas del lenguaje**
 - Mantiene la sintaxis y tipos de datos básicos del lenguaje Java
 - No existen los tipos `float` y `double`
- **Similar a la API de J2SE**
 - Mantiene un pequeño subconjunto de las clases básicas de J2SE
 - Con una interfaz más limitada en muchos casos
 - Excepciones
 - Hilos
 - No soporta hilos de tipo *daemon*
 - No soporta grupos de hilos
 - Flujos básicos de E/S
 - No hay flujos para acceder a ficheros
 - No hay tokenizadores
 - No hay serialización de objetos
 - Destinados principalmente a conexiones de red y memoria

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDe-4

Características ausentes



- **No soporta números reales**
 - No existen los tipos `float` y `double`
- **Desaparece el marco de colecciones**
 - Sólo se mantienen las clases `Vector`, `Stack` y `Hashtable`
- **Desaparece la API de *reflection***
 - Sólo se mantienen las clases `Class` y `Object`
- **Desaparece la API de red** `java.net`
 - Se sustituye por una más sencilla (GCF)
- **Desaparece la API de AWT/Swing**
 - Se utiliza una API adecuada para la interfaz de los dispositivos móviles (LCDUI)

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDe-5

Java para MIDe



- **Características de CLDC**
- **Números reales**
- **Temporizadores**
- **Serialización de objetos**
- **Acceso a los recursos**
- **MIDlets**

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDe-6

CLDC y los números reales



- En CLDC 1.0 no tenemos soporte para números reales
 - Los tipos `float` y `double` no existen
- En muchas aplicaciones podemos necesitar trabajar con este tipo de números
 - P.ej. para cantidades monetarias
- Podemos implementar números de coma fija usando enteros
 - Existen librerías como *MathFP* que realizan esta tarea
- En CLDC 1.1 ya existe soporte para los tipos `float` y `double`

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDP-7

Números reales sobre enteros



- Podemos representar números de coma fija como enteros
 - Consideramos que los últimos N dígitos son decimales
 - Por ejemplo, `1395` podría representar `13.95`
- Podremos hacer operaciones aritméticas con ellos
 - Suma y resta
 - Se realiza la operación sobre los números enteros
 - El resultado tendrá tantos decimales como los operandos
 - $13.95 + 5.20 \rightarrow 1395 + 520 = 1915 \rightarrow 19.15$
 - Multiplicación
 - Se realiza la operación sobre los números reales
 - El resultado tendrá tantos decimales como la suma del número de decimales de ambos operandos
 - $19.15 * 1.16 \rightarrow 1915 * 116 = 222140 \rightarrow 22.2140$

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDP-8

Conversión de números reales a enteros



- Deberemos convertir el entero a real para mostrarlo al usuario

```
public String imprimeReal(int numero) {
    int entero = numero / 100;
    int fraccion = numero % 100;
    return entero + "." + fraccion;
}
```
- Cuando el usuario introduzca un valor real deberemos convertirlo a entero

```
public int leeReal(String numero) {
    int pos_coma = numero.indexOf('.');
    String entero = numero.substring(0, pos_coma - 1);
    String fraccion = numero.substring(pos_coma + 1, pos_coma + 2);
    return Integer.parseInt(entero)*100+Integer.parseInt(fraccion);
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDP-9

Java para MIDs



- Características de CLDC
- Números reales
- Temporizadores
- Serialización de objetos
- Acceso a los recursos
- MIDlets

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDs-10

Temporizadores en los MIDs



- Los temporizadores resultan de gran utilidad en los MIDs
- Nos permiten programar tareas para que se ejecuten en un momento dado
 - Alarmas
 - Actualizaciones periódicas de software
 - Etc
- En CLDC se mantienen las clases de J2SE para temporizadores
 - `Timer` y `TimerTask`

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDs-11

Definir la tarea



- Deberemos definir la tarea que queremos programar
 - La definimos creando una clase que herede de `TimerTask`
 - En el método `run` de esta clase introduciremos el código que implemente la función que realizará la tarea

```
public class MiTarea extends TimerTask {  
    public void run() {  
        // Código de la tarea  
        // ... Por ejemplo, disparar alarma  
    }  
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDs-12

Programar la tarea



- Utilizaremos la clase `Timer` para programar tareas
- Para programar la tarea daremos
 - Un tiempo de comienzo. Puede ser:
 - Un retardo (respecto al momento actual)
 - Fecha y hora concretas
 - Una periodicidad. Puede ser:
 - Ejecutar una sola vez
 - Repetir con retardo fijo
 - Siempre se utiliza el mismo retardo tomando como referencia la última vez que se ejecutó
 - Repetir con frecuencia constante
 - Se toma como referencia el tiempo de la primera ejecución. Si alguna ejecución se ha retrasado, en la siguiente se recupera

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDe-13

Programar con retardo



- Creamos la tarea y un temporizador

```
Timer t = new Timer();
TimerTask tarea = new MiTarea();
```

- Programamos la tarea en el temporizador con un número de milisegundos de retardo

```
long retardo = 10000; // 10 segundos
long periodo = 1000; // 1 segundo
t.schedule(tarea, retardo); // Una vez
t.schedule(tarea, retardo, periodo); // Retardo fijo
t.scheduleAtFixedRate(tarea, retardo, periodo);
// Frecuencia constante
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDe-14

Programar a una hora



- Debemos establecer la hora en la que se ejecutará por primera vez el temporizador
 - Representaremos este instante de tiempo con un objeto `Date`
 - Podemos crearlo utilizando la clase `Calendar`

```
Calendar calendario = Calendar.getInstance();
calendario.set(Calendar.HOUR_OF_DAY, 8);
calendario.set(Calendar.MINUTE, 0);
calendario.set(Calendar.SECOND, 0);
calendario.set(Calendar.MONTH, Calendar.SEPTEMBER);
calendario.set(Calendar.DAY_OF_MONTH, 22);
Date fecha = calendario.getTime();
```

- Programamos el temporizador utilizando el objeto `Date`

```
t.schedule(tarea, fecha, periodo);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDe-15

Java para MIDs



- Características de CLDC
- Números reales
- Temporizadores
- **Serialización de objetos**
- Acceso a los recursos
- MIDlets

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDs-16

Serialización manual



- CLDC no soporta serialización de objetos
 - Conversión de un objeto en una secuencia de bytes
 - Nos permite enviar y recibir objetos a través de flujos de E/S
- Necesitaremos serializar objetos para
 - Hacer persistente la información que contengan
 - Enviar esta información a través de la red
- Podemos serializar manualmente nuestros objetos
 - Definiremos métodos `serialize` y `deserialize`
 - Utilizaremos los flujos `DataOutputStream` y `DataInputStream` para codificar y decodificar los datos del objeto en el flujo

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDs-17

Serializar



- Escribimos las propiedades del objeto en el flujo de salida

```
public class Punto2D {
    int x;
    int y;
    String etiqueta;
    ...
    public void serialize(OutputStream out) throws IOException {
        DataOutputStream dos = new DataOutputStream( out );
        dos.writeInt(x);
        dos.writeInt(y);
        dos.writeUTF(etiqueta);
        dos.flush();
    }
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDs-18

Deserializar



- Leemos las propiedades del objeto del flujo de entrada
- Debemos leerlas en el mismo orden en el que fueron escritas

```
public class Punto2D {  
    ...  
    public static Punto2D deserialize(InputStream in)  
        throws IOException {  
        DataInputStream dis = new DataInputStream( in );  
  
        Punto2D p = new Punto2D();  
        p.x = dis.readInt();  
        p.y = dis.readInt();  
        p.etiqueta = dis.readUTF();  
  
        return p;  
    }  
}
```

Java para MIDs



- Características de CLDC
- Números reales
- Temporizadores
- Serialización de objetos
- Acceso a los recursos
- MIDlets

Recursos en el JAR



- Hemos visto que podemos añadir cualquier tipo de recursos al JAR de nuestra aplicación
 - Ficheros de datos, imágenes, sonidos, etc
- Estos recursos no se encuentran en el sistema de ficheros
 - Son recursos del JAR
- Para leerlos deberemos utilizar el método `getResourceAsStream` de cualquier objeto `Class`:

```
InputStream in =  
    getClass().getResourceAsStream("/datos.txt");
```

- Es importante anteponer el nombre del recurso el carácter `"/"` para que acceda de forma relativa al raíz del JAR

Java para MIDs



- Características de CLDC
- Números reales
- Temporizadores
- Serialización de objetos
- Acceso a los recursos
- MIDlets

Programación de Dispositivos Móviles

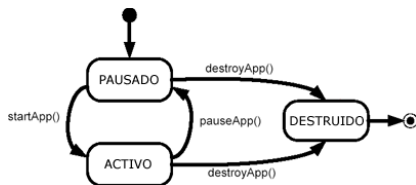
© 2006 Depto. Ciencia Computación e IA

Java para MIDs-22

Ciclo de vida



- La clase principal de la aplicación debe heredar de MIDlet
- Componente que se ejecuta en un contenedor
 - AMS = Software Gestor de Aplicaciones
- El AMS controla su ciclo de vida



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDs-23

Esqueleto de un MIDlet



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

public class MiMIDlet extends MIDlet {
    protected void startApp()
        throws MIDletStateChangeException {
        // Estado activo -> comenzar
    }

    protected void pauseApp() {
        // Estado pausa -> detener hilos
    }

    protected void destroyApp(boolean incondicional)
        throws MIDletStateChangeException {
        // Estado destruido -> liberar recursos
    }
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Java para MIDs-24

Propiedades




- Leer propiedades de configuración (JAD)

```
String valor = getAppProperty(String key);
```

- Salir de la aplicación

```
public void salir() {  
    try {  
        destroyApp(false);  
        notifyDestroyed();  
    } catch ( MIDletStateChangeException e ) { }  
}
```

Programación de Dispositivos
Móviles




Sesión 3:
Interfaz gráfica

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-1

Índice




- Interfaz gráfica
- Componentes de alto nivel
- Imágenes
- Comandos
- Diseño de pantallas

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-2

Interfaz gráfica



- Interfaz gráfica
- Componentes de alto nivel
- Imágenes
- Comandos
- Diseño de pantallas

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-3

Display



- La interfaz gráfica se realizará con la API LCDUI
 - LCDUI = *Limited Connected Devices User Interface*
 - Se encuentra en el paquete `javax.microedition.lcdui`
- El *display* representa el visor del móvil
 - Nos permite acceder a la pantalla
 - Nos permite acceder al teclado
- Cada MIDlet tiene asociado uno y sólo un *display*

```
Display display = Display.getDisplay(midlet);
```
- El *display* sólo mostrará su contenido en la pantalla y leerá la entrada del teclado cuando el MIDlet esté en primer plano

Programación de Dispositivos Móviles

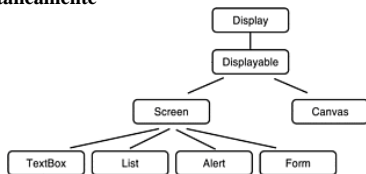
© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-4

Componentes displayables



- Son los elementos que pueden mostrarse en el *display*
- El *display* sólo puede mostrar un *displayable* simultáneamente



- Establecemos el *displayable* a mostrar con

```
display.setCurrent(displayable);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-5

Alto nivel vs Bajo nivel



- Podemos distinguir dos APIs:
 - Alto nivel
 - Componentes predefinidos: listas, formularios, campos de texto
 - Se implementan de forma nativa
 - Aplicaciones portables
 - Adecuados para *front-ends* de aplicaciones corporativas
 - Bajo nivel
 - Componentes personalizables: *canvas*
 - Debemos especificar en el código cómo dibujar su contenido
 - Tenemos control sobre los eventos de entrada del teclado
 - Se reduce la portabilidad
 - Adecuado para juegos

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-6

Interfaz gráfica



- Interfaz gráfica
- Componentes de alto nivel
- Imágenes
- Comandos
- Diseño de pantallas

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-7

Campos de texto



```
TextBox tb = new TextBox("Contraseña",  
    "", 8, TextField.ANY |  
    TextField.PASSWORD);  
  
Display d = Display.getDisplay(this);  
d.setCurrent(tb);
```



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-8

Listas



```
List l = new List("Menu",  
    Choice.IMPLICIT);  
l.append("Nuevo juego", null);  
l.append("Continuar", null);  
l.append("Instrucciones", null);  
l.append("Hi-score", null);  
l.append("Salir", null);  
  
Display d =  
    Display.getDisplay(this);  
d.setCurrent(l);
```



Implícita



Múltiple



Exclusiva

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-9

Formularios

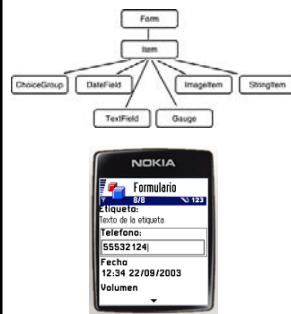


```
Form f = new Form("Formulario");

Item itemEtiqueta = new StringItem(
    "Etiqueta:",
    "Texto de la etiqueta");
Item itemTexto = new TextField(
    "Telefono:", "", 8,
    TextField.PHONENUMBER);
Item itemFecha = new DateField(
    "Fecha",
    DateField.DATE_TIME);
Item itemBarra = new Gauge("Volumen",
    true, 10, 8);

f.append(itemEtiqueta);
f.append(itemTexto);
f.append(itemFecha);
f.append(itemBarra);

Display d = Display.getDisplay(this);
d.setCurrent(f);
```



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-10

Alertas



▪ Mensaje de transición entre pantallas

```
Alert a = new Alert("Error",
    "No hay ninguna nota seleccionada",
    null, AlertType.ERROR);

Display d = Display.getDisplay(midlet);
d.setCurrent(a, d.getCurrent());
```



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-11

Interfaz gráfica



- Interfaz gráfica
- Componentes de alto nivel
- Imágenes
- Comandos
- Diseño de pantallas

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-12

Imágenes en MIDP



- En muchos componentes podemos incluir imágenes
- Las imágenes se encapsulan en la clase `Image`
- Encontramos dos tipos de imágenes

➤ Imágenes mutables:

- Podemos editar su contenido desde nuestra aplicación
- Se crea con:

```
Image img_mut = Image.createImage(ancho, alto);
```

- Al crearla estará vacía. Deberemos dibujar gráficos en ella.

➤ Imágenes inmutables:

- Una vez creada, ya no se puede modificar su contenido
- En los componentes de alto nivel sólo podremos usar este tipo

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-13

Imágenes inmutables



- El único formato reconocido por MIDP es PNG
- Las imágenes inmutables se crean:

➤ A partir de un fichero PNG contenido en el JAR

```
Image img = Image.createImage("/logo.png");
```

➤ A partir de un array de bytes leído de un fichero PNG

- Podemos leer un fichero PNG a través de la red.
- Almacenamos los datos leídos en forma de array de bytes.

```
Image img = Image.createImage(datos, offset, longitud);
```

➤ A partir de una imagen mutable

- Nos permitirá usar en componentes de alto nivel imágenes creadas como mutables, y editadas en el código

```
Image img_inmut = Image.createImage(img_mut);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-14

Interfaz gráfica



- Interfaz gráfica
- Componentes de alto nivel
- Imágenes
- Comandos
- Diseño de pantallas

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-15

Comandos de entrada



- La entrada de usuario se realiza mediante comandos



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-16

Creación de comandos



- Podemos crear comandos y añadirlos a un *displayable*

```
TextBox tb = new TextBox("Login", "", 8, TextField.ANY);
Command cmdOK = new Command("OK", Command.OK, 1);
Command cmdAyuda = new Command("Ayuda", Command.HELP, 1);
Command cmdSalir = new Command("Salir", Command.EXIT, 1);
Command cmdBorrar = new Command("Borrar", Command.SCREEN, 1);
Command cmdCancelar = new Command("Cancelar", Command.CANCEL, 1);

tb.addCommand(cmdOK);
tb.addCommand(cmdAyuda);
tb.addCommand(cmdSalir);
tb.addCommand(cmdBorrar);
tb.addCommand(cmdCancelar);

Display d = Display.getDisplay(this);
d.setCurrent(tb);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-17

Listener de comandos



- Debemos crear un *listener* para dar respuesta a los comandos

```
class ListenerLogin implements CommandListener {
    public void commandAction(Command c, Displayable d) {
        if(c == cmdOK) {
            // Aceptar
        } else if(c == cmdCancelar) {
            // Cancelar
        } else if(c == cmdSalir) {
            // Salir
        } else if(c == cmdAyuda) {
            // Ayuda
        } else if(c == cmdBorrar) {
            // Borrar
        }
    }
}
```

- Registrar el *listener* en el *displayable*

```
tb.setCommandListener(new ListenerLogin());
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-18

Interfaz gráfica



- Interfaz gráfica
- Componentes de alto nivel
- Imágenes
- Comandos
- Diseño de pantallas

Programación de Dispositivos Móviles

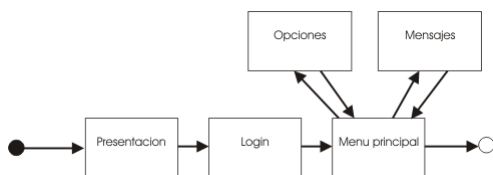
© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-19

Mapa de pantallas



- Cada *displayable* es una pantalla de la aplicación
- Conviene realizar un mapa de pantallas en la fase de diseño de la aplicación



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-20

Capa de presentación



- Conviene seguir un patrón de diseño para realizar la capa de presentación de nuestra aplicación
- Definiremos una clase por cada pantalla
- Encapsularemos en ella:
 - Creación de la interfaz
 - Definición de comandos
 - Respuesta a los comandos
- La clase deberá:
 - Heredar del tipo de *displayable* que vayamos a utilizar
 - Implementar `CommandListener` (u otros listeners) para dar respuesta a los comandos
 - Guardar una referencia al `MIDlet`, para poder cambiar de pantalla

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-21

Creación de la pantalla



```
public class MenuPrincipal extends List implements CommandListener {  
  
    MiMIDlet owner;  
    Command selec;  
    int itemNuevo;  
    int itemSalir;  
  
    public MenuPrincipal(MiMIDlet owner) {  
        super("Menu", List.IMPLICIT);  
        this.owner = owner;  
  
        // Añade opciones al menu  
        itemNuevo = this.append("Nuevo juego", null);  
        itemSalir = this.append("Salir", null);  
  
        // Crea comandos  
        selec = new Command("Seleccionar", Command.SCREEN, 1);  
        this.addCommand(selec);  
        this.setCommandListener(this);  
    }  
    ...  
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-22

Respuesta a los comandos



- En la misma clase capturamos los eventos del usuario

```
...  
public void commandAction(Command c, Displayable d) {  
    if(c == selec || c == List.SELECT_COMMAND) {  
        if(getSelectedIndex() == itemNuevo) {  
            // Nuevo juego  
            Display display = Display.getDisplay(owner);  
            PantallaJuego pj = new PantallaJuego(owner, this);  
            display.setCurrent(pj);  
        } else if(getSelectedIndex() == itemSalir) {  
            // Salir de la aplicación  
            owner.salir();  
        }  
    }  
}  
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Interfaz gráfica-23



Sesión 5: Gráficos avanzados

Índice



- Gráficos en LCDUI
- Contexto gráfico
- Animaciones
- Eventos de entrada
- Gráficos 3D

Gráficos avanzados



- Gráficos en LCDUI
- Contexto gráfico
- Animaciones
- Eventos de entrada
- Gráficos 3D

API de bajo nivel



- Con la API de bajo nivel podremos crear componentes personalizados
 - Adecuado para juegos
 - Se reduce la portabilidad
- Utilizaremos el *displayable* Canvas
 - Consiste en una pantalla vacía
 - Deberemos especificar lo que se mostrará en él
 - Controlaremos la interacción con el usuario a bajo nivel
- Nos permitirá dibujar el contenido que queramos
 - Se hará de forma similar a J2SE
 - Utilizaremos un objeto *Graphics* para dibujar en pantalla

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-4

Creación de un canvas



- Debemos crear una clase que herede de Canvas

```
public class MiCanvas extends Canvas {  
    public void paint(Graphics g) {  
        // Dibujamos en la pantalla  
        // usando el objeto g proporcionado  
    }  
}
```

- Render pasivo
 - No controlamos el momento en el que se dibujan los gráficos
 - Sólo definimos la forma de dibujarlos en el método *paint*
 - El sistema invocará este método cuando necesite dibujar nuestro componente

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-5

Propiedades del canvas



- Según el dispositivo el canvas tendrá distinta resolución
- Podemos consultar la resolución con

```
int ancho = getWidth();  
int alto = getHeight();
```
- El canvas no suele ocupar toda la pantalla
 - Se reserva un área para el dispositivo
 - Cobertura, título de la pantalla, comandos, etc
- En MIDP 2.0 podemos utilizar la pantalla completa

```
setFullScreenMode(true);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-6

Gráficos avanzados



- Gráficos en LCDUI
- Contexto gráfico
- Animaciones
- Eventos de entrada
- Gráficos 3D

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-7

Atributos del contexto



- El objeto `Graphics` representa el contexto gráfico
 - Nos permitirá dibujar contenido en la pantalla
- El contexto tiene asociado atributos
 - Color del lápiz

```
g.setColor(0x00FF99); // Color codificado en 0xRRGGBB
```

- Tipo del lápiz (sólido o punteado)

```
g.setStrokeStyle(Graphics.SOLID); // o Graphics.DOTTED
```

- Fuente de texto

```
g.setFont(fuente); // Utilizamos objetos de la clase Font
```

- Área de recorte

```
g.setClip(x, y, ancho, alto);
```

- Origen de coordenadas

```
g.translate(x,y);
```

Programación de Dispositivos Móviles

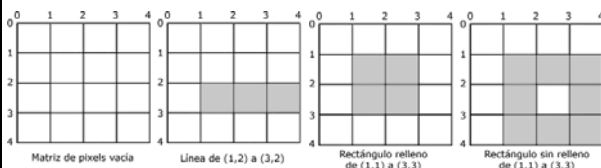
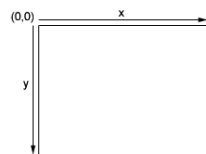
© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-8

Sistema de coordenadas



- La esquina superior izquierda tiene coordenadas (0,0)
 - Las X son positivas hacia la derecha
 - Las Y son positivas hacia abajo
- Las coordenadas corresponden a límites entre píxeles



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-9

Dibujado de primitivas geométricas

Podemos dibujar distintas primitivas geométricas:

- Líneas**

```
g.drawLine(x1, y1, x2, y2);
```
- Rectángulos**

```
g.drawRect(x, y, ancho, alto);
g.fillRect(x, y, ancho, alto);
```
- Rectángulos redondeados**

```
g.drawRoundRect(x, y, ancho, alto, wArco, hArco);
g.fillRoundRect(x, y, ancho, alto, wArco, hArco);
```
- Arcos**

```
g.drawArc(x, y, ancho, alto, iniArco, arco);
g.fillArc(x, y, ancho, alto, iniArco, arco);
```

Rectángulo
Rectángulo redondeado

Arco
Líneas

Programación de Dispositivos Móviles
© 2006 Depto. Ciencia Computación e IA
Gráficos avanzados-10

Puntos anchor

Nos sirven para ubicar elementos en la pantalla

- Lo utilizaremos para texto e imágenes

Especificaremos

- Coordenadas de la pantalla (x,y)
- Qué posición del elemento se ubicará en dichas coordenadas

Esta posición puede ser:

- Para la horizontal:

```
Graphics.LEFT
Graphics.HCENTER
Graphics.RIGHT
```
- Para la vertical

```
Graphics.TOP
Graphics.VCENTER
Graphics.BASELINE
Graphics.BOTTOM
```

Programación de Dispositivos Móviles
© 2006 Depto. Ciencia Computación e IA
Gráficos avanzados-11

Texto

Dibujamos texto con:

```
g.drawString(cadena, x, y, anchor);
```

Texto de prueba

(0,0) Línea de base

```
g.drawString("Texto de prueba", 0, 0, Graphics.LEFT|Graphics.BASELINE);
```

Podemos necesitar las medidas del texto en píxeles

- La clase `Font` de la fuente utilizada nos proporciona esa información

stringWidth

Ascent

Descent

Texto de prueba

(0,0)

Programación de Dispositivos Móviles
© 2006 Depto. Ciencia Computación e IA
Gráficos avanzados-12

Imágenes



- Podemos dibujar tanto imágenes mutables como inmutables
- Dibujaremos la imagen en pantalla con:

```
g.drawImage(img, x, y, anchor);
```
- Por ejemplo:

```
g.drawImage(img, 0, 0, Graphics.TOP|Graphics.LEFT);
```
- En el caso de las imágenes mutables, editaremos su contenido utilizando su contexto gráfico

```
Graphics offg = img_mut.getGraphics();
```

 - Se utilizará igual que cuando se dibuja en pantalla
 - En este caso los gráficos se dibujan en la imagen en memoria

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-13

Gráficos avanzados



- Gráficos en LCDUI
- Contexto gráfico
- Animaciones
- Eventos de entrada
- Gráficos 3D

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-14

Redibujado



- Para crear una animación tendremos que modificar el contenido de la pantalla con el tiempo
- Debemos solicitar al sistema que redibuje

```
repaint();
```
- Una vez hecho esto, cuando el sistema tenga tiempo redibujará la pantalla invocando nuestro método `paint`
- Si sólo hemos modificado un área, podemos solicitar el redibujado sólo de este área

```
repaint(x, y, ancho, alto);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-15

Técnica del doble buffer



- Para mostrar cada frame de la animación debemos
 - Borrar el frame anterior
 - Dibujar el nuevo frame
- Al hacer esto repetidas veces puede producirse un efecto de “parpadeo” en la pantalla
- Para evitarlo podemos utilizar la técnica del doble buffer
 - Dibujamos todo el contenido en una imagen mutable del mismo tamaño de la pantalla
 - Volcamos la imagen a la pantalla como una unidad
- Muchos dispositivos ya implementan esta técnica
 - Con `isDoubleBuffered()` sabremos si lo implementa el dispositivo
 - Si no lo implementa el dispositivo, deberíamos hacerlo nosotros

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-16

Hilo de la animación



- Creamos un hilo que cada cierto intervalo:
 - Modifique las propiedades de los objetos a dibujar
 - Por ejemplo su posición (x,y)
 - Llame a `repaint` para solicitar el redibujado de la pantalla

```
public void run() {  
    // El rectángulo comienza en (10,10)  
    x = 10; y = 10;  
    while(x < 100) {  
        x++;  
        repaint();  
        try {  
            Thread.sleep(100);  
        } catch (InterruptedException e) {}  
    }  
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-17

Hilo de eventos



- Para poner en marcha el hilo podemos utilizar el evento `showNotify` del `Canvas` por ejemplo
 - Este evento se produce cuando el `Canvas` se muestra
- ```
public class MiCanvas extends Canvas implements Runnable {
 ...
 public void showNotify() {
 Thread t = new Thread(this);
 t.start();
 }
}
```
- Podemos utilizar `hideNotify` para detenerlo
  - En los eventos deberemos devolver el control inmediatamente
    - Si necesitamos realizar una operación de larga duración, crearemos un hilo que la realice como en este caso
    - Si no devolviésemos el control, se bloquearía el hilo de eventos y la aplicación dejaría de responder
      - No actualizaría los gráficos, no leería la entrada del usuario, etc

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-18

---

---

---

---

---

---

---

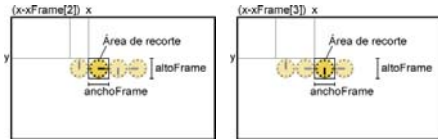
---



## Optimización de imágenes



- Si queremos mostrar una imagen animada necesitamos tener varios frames
  - Para evitar tener varias imágenes, podemos guardar todos los frames en una misma imagen
- Podemos utilizar un área de recorte para seleccionar el frame que se dibuja en cada momento



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-19

## Gráficos avanzados



- Gráficos en LCDUI
- Contexto gráfico
- Animaciones
- Eventos de entrada
- Gráficos 3D

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-20

## Eventos del teclado



- Con el canvas tenemos acceso a los eventos a bajo nivel
  - Sabremos cuando el usuario pulsa o suelta una tecla
- Para dar respuesta a estos eventos debemos sobrescribir los siguientes métodos del `Canvas`

```
public class MiCanvas extends Canvas {
 ...
 public void keyPressed(int cod) {
 // Se ha presionado la tecla con código cod
 }
 public void keyRepeated(int cod) {
 // Se mantiene pulsada la tecla con código cod
 }
 public void keyReleased(int cod) {
 // Se ha soltado la tecla con código cod
 }
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-21

## Códigos de las teclas



- Tenemos definido como constante los códigos de las teclas estándar

➢ Utilizar estos códigos mejora la portabilidad

|                  |   |
|------------------|---|
| Canvas.KEY_NUM0  | 0 |
| Canvas.KEY_NUM1  | 1 |
| Canvas.KEY_NUM2  | 2 |
| Canvas.KEY_NUM3  | 3 |
| Canvas.KEY_NUM4  | 4 |
| Canvas.KEY_NUM5  | 5 |
| Canvas.KEY_NUM6  | 6 |
| Canvas.KEY_NUM7  | 7 |
| Canvas.KEY_NUM8  | 8 |
| Canvas.KEY_NUM9  | 9 |
| Canvas.KEY_POUND | # |
| Canvas.KEY_STAR  | * |

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-22

---

---

---

---

---

---

---

---

---

---

## Acciones de juegos



- Cada tecla tiene asociada una acción de juego

- Las acciones de juego son:

Canvas.LEFT  
Canvas.RIGHT  
Canvas.UP  
Canvas.DOWN  
Canvas.FIRE

- Podemos consultar la acción de juego asociada a una tecla

```
int accion = getGameAction(cod);
```

- Estas acciones mejoran la portabilidad en juegos

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-23

---

---

---

---

---

---

---

---

---

---

## Eventos del puntero



- En dispositivos con puntero podremos recibir estos eventos

```
public class MiCanvas extends Canvas {
 ...
 public void pointerPressed(int x, int y) {
 // Se ha pinchado con el puntero en (x,y)
 }
 public void pointerDragged(int x, int y) {
 // Se ha arrastrado el puntero a (x,y)
 }
 public void pointerReleased(int x, int y) {
 // Se ha soltado el puntero en (x,y)
 }
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-24

---

---

---

---

---

---

---

---

---

---

## Gráficos avanzados



- Gráficos en LCDUI
- Contexto gráfico
- Animaciones
- Eventos de entrada
- Gráficos 3D

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-25

---

---

---

---

---

---

---

---

## Mobile 3D Graphics



- La API *Mobile 3D Graphics* nos permite crear gráficos 3D en los dispositivos móviles
- Soporta dos modos:
  - Modo inmediato
    - Se crean gráficos a bajo nivel
    - Se especifica los vértices, caras y apariencia de los objetos
    - Adecuado para representar datos en 3D
  - Modo *retained*
    - Se crea un grafo con los distintos objetos de la escena 3D
    - Los objetos 3D se cargan de un fichero M3G
    - Adecuado para juegos



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-26

---

---

---

---

---

---

---

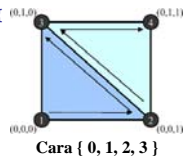
---

## Modo inmediato



- Definimos vértices y caras de los objetos

```
short [] vertexValues = { 0, 0, 0, // 0
 0, 0, 1, // 1
 0, 1, 0, // 2
 0, 1, 1, // 3
 1, 0, 0, // 4
 1, 0, 1, // 5
 1, 1, 0, // 6
 1, 1, 1 // 7 };
int [] faceIndices = { 0, 1, 2, 3,
 7, 5, 6, 4,
 4, 5, 0, 1,
 3, 7, 2, 6,
 0, 2, 4, 6,
 1, 5, 3, 7 };
```



Sin material



Con material



Con textura

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Gráficos avanzados-27

---

---

---

---

---

---

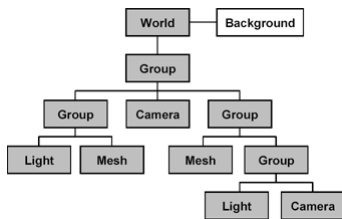
---

---

## Modo retained



- Se construye un grafo de la escena
  - Contiene todos los objetos en distintos grupos

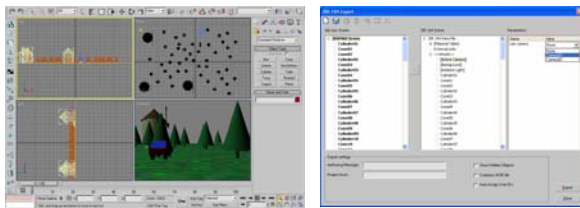


- Cargamos este grafo de un fichero M3G

## Modelado



- Podemos modelar los gráficos 3D con herramientas como 3D Studio MAX
  - A partir de 3DSMAX 7.0 se incluye una herramienta para exportar a ficheros M3G



## Ejemplo de modo *retained*



```
public class Visor3DRetained extends Canvas {

 Graphics3D g3d;
 World mundo;

 public Visor3DRetained() {
 g3d = Graphics3D.getInstance();
 try {
 mundo = (WorldLoader.load("/mundo.m3g"))[0];
 } catch (IOException e) { // Error al cargar mundo }
 }

 protected void paint(Graphics g) {
 try {
 g3d.bindTarget(g);
 g3d.render(mundo);
 } finally {
 g3d.releaseTarget();
 }
 }
}
```



## Sesión 6: Sonido y multimedia

---

---

---

---

---

---

---

### Índice



- Reproductor de medios
- Reproducción de sonido
- Reproducción de video
- Captura

---

---

---

---

---

---

---

### Sonido y multimedia



- Reproductor de medios
- Reproducción de sonido
- Reproducción de video
- Captura

---

---

---

---

---

---

---

## Multimedia en J2ME



- MIDP 1.0 no soporta la reproducción de sonidos
- MIDP 2.0 permite reproducir audio
  - Incorpora subconjunto de MMAPI para audio
    - Secuencias de tonos
    - Ficheros WAV, MIDI, etc
- MMAPI permite
  - Reproducir audio
  - Reproducir video
  - Capturar audio y video
- Los dispositivos MIDP 1.0 y MIDP 2.0 que incorporen MMAPI permitirán realizar todas estas funciones

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-4

---

---

---

---

---

---

---

---

## Reproductor de medios



- Nos permitirá reproducir distintos tipos de medios
- Será un objeto de la clase `Player`

```
Player player = Manager.createPlayer(
 "http://j2ee.ua.es/pdm/sonido.wav");
```

- Para crearlo a partir de un recurso del JAR

```
InputStream in =
 getClass().getResourceAsStream("/musica.mid");
Player player = Manager.createPlayer(in, "audio/midi");
```

- En este caso debemos proporcionar el tipo MIME

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-5

---

---

---

---

---

---

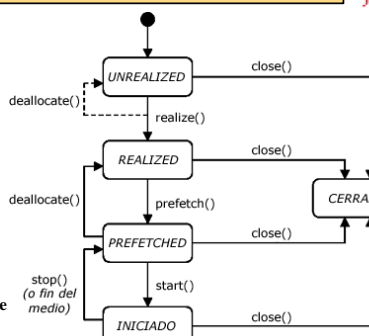
---

---

## Estados



- Nos permiten ajustar la reserva de recursos
- **Unrealized**
  - No ha reservado recursos
  - Tardará en comenzar la reproducción
- **Prefetched**
  - Tiene reservados todos los recursos necesarios para comenzar
  - Puede comenzar de forma instantánea



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-6

---

---

---

---

---

---

---

---

## Controles



- El reproductor de medios es genérico
  - Se podrá utilizar para cualquier tipo de medio
  - Para controlar las características concretas de un determinado tipo de medio utilizaremos controles

- Obtendremos un control con

```
Control control = player.getControl(nombre);
```

- Para poder obtener un control el reproductor deberá estar al menos en estado *realized*

- Por ejemplo, tenemos los controles

```
"VolumeControl"
"ToneControl"
"VideoControl"
"RecordControl"
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-7

---

---

---

---

---

---

---

---

## Sonido y multimedia



- Reproductor de medios
- Reproducción de sonido
- Reproducción de video
- Captura

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-8

---

---

---

---

---

---

---

---

## Reproducción de tonos



- Tono simple

```
Manager.playTone(nota, duracion, volumen);
```

- Secuencia de tonos

- Crear reproductor de tonos

```
Player player =
 Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR);
```

- Obtener control de tonos

```
player.realize();
ToneControl tc =
 (ToneControl)player.getControl("ToneControl");
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-9

---

---

---

---

---

---

---

---

## Reproducción de tonos (II)



### ➤ Establecer secuencia de tonos

```
tc.setSequence(new byte[] {
 ToneControl.VERSION,1,
 ToneControl.TEMPO,30,
 ToneControl.C4,16,
 ToneControl.C4+2,16,
 ToneControl.C4+4,16, //E4
 ToneControl.C4+5,16, //F4
 ...
});
```

### ➤ Comenzar la reproducción

```
player.start();
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-10

---

---

---

---

---

---

---

---

## Reproducción de ficheros



### ▪ Podemos reproducir músicas y sonidos de ficheros

- Los formatos soportados dependen del dispositivo
  - WAV, MIDI, MP3, etc

### ▪ Crear reproductor a partir de URL

```
Player player = Manager.createPlayer(
 "http://j2ee.ua.es/pdm/musica.mid");
```

### ▪ Crear reproductor a partir de recurso en el JAR

```
InputStream in =
 getClass().getResourceAsStream("/musica.mid");
Player player = Manager.createPlayer(in, "audio/midi");
```

### ▪ Comenzar la reproducción

```
player.start();
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-11

---

---

---

---

---

---

---

---

## Control de volumen



### ▪ Podemos obtener un control de volumen

- Estará disponible en los reproductores de audio

```
player.realize();
VolumeControl vol =
 (VolumeControl)player.getControl("VolumeControl");
```

### ▪ Con este control podemos

```
vol.setLevel(volumen);
vol.setMute(true);
```

### ▪ El volumen será un valor entero de 0 a 100

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-12

---

---

---

---

---

---

---

---



## Sonido y multimedia



- Reproductor de medios
- Reproducción de sonido
- Reproducción de video
- Captura

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-13

---

---

---

---

---

---

---

---

## Reproductor de video



- Podemos reproducir videos de distintos formatos
  - Los formatos reconocidos dependen del dispositivo
    - 3GPP, MPEG, etc



- Creamos el reproductor de video con

```
InputStream in =
 getClass().getResourceAsStream("/video.3gp");
Player player = Manager.createPlayer(in, "video/3gpp");
```

- Necesitaremos además un control de video
  - Nos permitirá vincular el video a la pantalla

```
player.realize();
VideoControl vc =
 (VideoControl)player.getControl("VideoControl");
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-14

---

---

---

---

---

---

---

---

## Vincular el video a la pantalla



- Podemos vincular el video de dos formas
  - A un item de un formulario

```
Item item = (Item)vc.initDisplayMode(
 VideoControl.USE_GUI_PRIMITIVE, null);
formulario.append(item);
```

- A una región de un canvas

```
vc.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, canvas);
vc.setVisible(true);
```

- Comenzar la reproducción

```
player.start();
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-15

---

---

---

---

---

---

---

---

## Sonido y multimedia



- Reproductor de medios
- Reproducción de sonido
- Reproducción de video
- **Captura**

---

---

---

---

---

---

---

---

## Captura de medios



- Podemos capturar audio y/o video
  - Audio del micrófono
  - Video de la cámara
- Crear un reproductor de medios con una URL como  
`capture://dispositivo`
- Por ejemplo podemos utilizar  
`capture://audio`  
`capture://video`  
`capture://audio_video`
- Creamos el reproductor con la URL adecuada

```
Player player = Manager.createPlayer("capture://video");
```

---

---

---

---

---

---

---

---

## Mostrar captura



- En el caso de capturar video, podemos mostrarlo mientras se captura
- Esto nos permitirá ver en la pantalla el video que se está capturando
  - Podremos tomar fotografías
- Para mostrar el video capturado haremos lo mismo que para mostrar video de un fichero

```
player.realize();
VideoControl vc =
 (VideoControl)player.getControl("VideoControl");
vc.initDisplayMode(
 VideoControl.USE_DIRECT_VIDEO, canvas);
vc.setVisible(true);
player.start();
```

---

---

---

---

---

---

---

---

## Grabación de video



- Para grabar video utilizaremos un control de grabación

```
RecordControl rc =
(RecordControl)player.getControl("RecordControl");
```

- Establecemos el flujo de salida donde grabar el video

```
ByteArrayOutputStream out = new ByteArrayOutputStream();
rc.setRecordStream(out);
```

- Comenzamos/reanudamos la grabación

```
rc.startRecord();
```

- Detenemos la grabación

```
rc.stopRecord();
```

- Finalizamos la grabación

```
rc.commit();
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-19

---

---

---

---

---

---

---

## Captura de imágenes



- Mientras se reproduce un video podemos capturar imágenes de dicho video

- Capturamos el frame actual con

```
byte [] img_png = vc.getSnapshot(null);
```

- Nos devolverá la imagen codificada en PNG

- Podemos crear la imagen con

```
Image img = Image.createImage(img_png, 0,
img_png.length);
```

- Tomar fotografías

- Reproducir video capturado por la cámara
- Capturar imágenes de dicho video

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Sonido y multimedia-20

---

---

---


---

---

---

---

Programación de Dispositivos  
Móviles



Sesión 7:  
Juegos

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-1

---

---

---

---


---

---

---

---

Índice



- Juegos para móviles
- Desarrollo de juegos
- Motor del juego
- Entrada de usuario
- Componentes de la pantalla

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-2

---

---

---

---


---

---

---

---

Juegos



- Juegos para móviles
- Desarrollo de juegos
- Motor del juego
- Entrada de usuario
- Componentes de la pantalla

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-3

---

---

---

---

---

---

---

---

## Juegos para móviles



- Los juegos Java han tenido un gran éxito
  - Gran parte de los usuarios de móviles están interesados en este tipo de aplicaciones de ocio
  - Java nos permite portar fácilmente los juegos a distintos modelos de móviles con poco esfuerzo
- Es el tipo de aplicación MIDP más difundida



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-4

---

---

---

---

---

---

---

---

## Características de los juegos para móviles



- Normalmente el usuario utiliza el móvil para pasar el rato mientras hace tiempo
  - Cuando está haciendo cola
  - Cuando viaja en autobús
  - Etc
- Por lo tanto estos juegos deberán
  - No requerir apenas aprendizaje por parte del usuario
  - Permitir ser pausados
    - El usuario puede ser interrumpido en cualquier momento
  - Permitir guardar nuestro progreso
    - Cuando tengamos que dejar el juego, no perder nuestros avances

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-5

---

---

---

---

---

---

---

---

## Características de los dispositivos



- Escasa memoria
  - No crear más objetos de los necesarios
  - No cargar excesivo número de recursos, ni imágenes complejas
- CPU lenta
  - Optimizar el código
- Pantalla reducida
  - Los gráficos deben ser suficientemente grandes
- Almacenamiento limitado
  - Almacenar la información mínima al guardar la partida
  - Codificar la información de forma compacta
- Poco ancho de banda
  - Dificultad para implementar juegos de acción en red
- Teclado pequeño
  - El manejo debe ser sencillo
- Posibles interrupciones
  - Permitir modo de pausa

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-6

---

---

---

---


---

---

---

---

Juegos



- Juegos para móviles
- Desarrollo de juegos
- Motor del juego
- Entrada de usuario
- Componentes de la pantalla

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-7

---

---

---

---


---

---

---

---

API de bajo nivel



- Los juegos deben resultar atractivos a los usuarios
  - Deberán tener gráficos personalizados e innovadores
  - Deberemos dar una respuesta rápida al control del usuario
- Utilizaremos por lo tanto la API de bajo nivel
  - Nos permite dibujar gráficos libremente
  - Tenemos acceso completo a los eventos del teclado
- En MIDP 2.0 se incluye una librería para desarrollo de juegos
  - Incorpora clases que nos facilitarán la creación de juegos  
`javax.microedition.lcui.game`

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-8

---

---

---

---


---

---

---

---

Aplicación conducida por los datos



- El motor del juego debe ser lo más genérico posible
- Debemos llevar toda la información posible a la capa de datos
  - El juego normalmente consistirá en varios niveles
  - La mecánica del juego será prácticamente la misma en todos ellos
  - Será conveniente llevar la definición de los niveles a la capa de datos
  - Almacenaremos esta información en un fichero
  - Simplemente editando este fichero, podremos añadir o modificar los niveles del juego

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-9

---

---

---

---

---

---

---

---

## Ejemplo de codificación de niveles



- Vamos a ver como ejemplo un clon del *Frogger*
- El fichero con los datos de niveles estará codificado de la siguiente forma:

```
<int> Numero de fases
Para cada fase
 <UTF> Titulo
 <byte> Número de carriles
Para cada carril
 <byte> Velocidad
 <short> Separación
 <byte> Tipo de coche
```

- Podemos utilizar un objeto `DataInputStream` para deserializar esta información

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-10

---

---

---

---

---

---

---

---

## ¿Y eso no puede hacerlo otro?



- Para realizar tareas anodinas y repetitivas están las máquinas
  - J2ME-Polish incluye un editor de ficheros binarios genérico

| Name        | Count       | Type       | Data                                                        |
|-------------|-------------|------------|-------------------------------------------------------------|
| NumFases    | 3           | int        | 4                                                           |
| Titulo      | 3           | UTF-String | Fase 1                                                      |
| NumCarriles | 3           | byte       | 5                                                           |
| Carriles    | NumCarriles | TrackData  | {3, 100, 2} {2, 100, 0} {1, 100, 1}                         |
| Titulo      | 3           | UTF-String | Fase 2                                                      |
| NumCarriles | 3           | byte       | 5                                                           |
| Carriles    | NumCarriles | TrackData  | {4, 150, 1} {1, 100, 1} {3, 150, 0} {2, 100, 1} {1, 100, 0} |
| Titulo      | 3           | UTF-String | Fase 3                                                      |
| NumCarriles | 3           | byte       | 4                                                           |
| Carriles    | NumCarriles | TrackData  | {4, 150, 0} {3, 100, 1} {2, 100, 0} {1, 100, 2}             |
| Titulo      | 3           | UTF-String | Fase 4                                                      |
| NumCarriles | 3           | byte       | 5                                                           |
| Carriles    | NumCarriles | TrackData  | {3, 150, 2} {2, 150, 1} {1, 150, 0} {1, 100, 1} {2, 75, 2}  |

[www.j2mepolish.org](http://www.j2mepolish.org)

- Además incluye otras herramientas y librerías útiles para el desarrollo de juegos (editor de fuentes, componentes propios, etc) y para hacerlos independientes del modelo de dispositivo

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-11

---

---

---

---

---

---

---

---

## Gestión de recursos



- Conviene centralizar la gestión de los recursos
  - Crear un clase que gestione la carga de estos recursos
- Cargar todos los recursos necesarios durante la inicialización
  - En dispositivos con poca memoria podríamos cargar sólo los recursos de la fase actual
- No tener los recursos dispersos por el código de la aplicación
  - Evitamos que se carguen varias veces accidentalmente
    - Los recursos los carga el gestor de recursos y todos los demás componentes de la aplicación los obtendrán de éste
  - Sabremos qué recursos está utilizando la aplicación simplemente consultando el código del gestor

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-12

---

---

---

---

---

---

---

---

## Gestor de recursos



```
public class Resources {
 public static final int IMG_TIT_TITULO = 0;
 public static final int IMG_SPR_CROC = 1;
 public static final int IMG_BG_STAGE_1 = 2;

 public static Image[] img;

 private static String[] imgNames = {
 "/title.png", "/sprite.png", "/stage01.png" };

 private static void loadCommonImages()
 throws IOException {
 img = new Image[imgNames.length];
 for (int i = 0; i < imgNames.length; i++) {
 img[i] = Image.createImage(imgNames[i]);
 }
 }
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-13

---

---

---

---

---

---

---

---

## Tipos de recursos



- En el gestor de recursos podemos añadir recursos como
  - Imágenes
  - Sonidos
  - Datos
  - Etc
- Podremos acceder a los datos de las fases a través de este gestor
  - Los datos habrán sido leídos del fichero de datos de fases durante la inicialización
  - Deberemos haber creado las estructuras de datos adecuadas en Java para encapsular esta información

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-14

---

---

---

---

---

---

---

---

## Portabilidad



- Centralizar toda la información dependiente del dispositivo en una misma clase
  - Dimensiones de la pantalla
  - Dimensiones de los objetos
  - Posiciones de los objetos
  - Etc
- Podemos añadir estos datos como constantes en una misma clase

```
public class CommonData {
 public static final int NUM_LIVES = 3;
 public final static int SCREEN_WIDTH = 176;
 public final static int SCREEN_HEIGHT = 208;
 public final static int SPRITE_WIDTH = 16;
 public final static int SPRITE_HEIGHT = 22;
 ...
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-15

---

---

---

---

---

---

---

---



## Optimización



- El juego deberá funcionar de forma fluida para que sea jugable
  - Deberemos optimizarlo
  - Primero hacer una implementación clara, después optimizar
- Identificar que operación consume más tiempo
  - Gráficos
    - Volcar sólo la parte de la pantalla que haya cambiado
  - Lógica
    - No crear más objetos que los necesarios, reutilizar cuando sea posible
    - Permitir que se desechen los objetos que no se utilicen, poniendo sus referencias a null

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-16

---

---

---

---

---

---

---

---

## Juegos



- Juegos para móviles
- Desarrollo de juegos
- Motor del juego
- Entrada de usuario
- Componentes de la pantalla

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-17

---

---

---

---

---

---

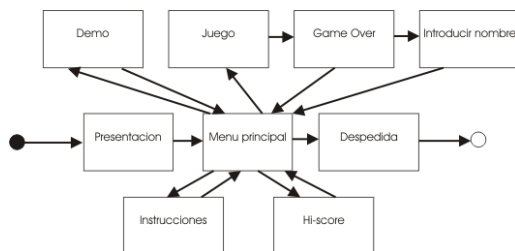
---

---

## Pantallas



- Un juego típico suele constar de las siguientes pantallas



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-18

---

---

---

---

---

---

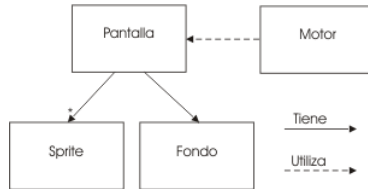
---

---

## Componentes



- En la pantalla de juego, podemos distinguir los siguientes componentes:



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-19

---

---

---

---

---

---

---

---

## Motor del juego



- Vamos a ver cómo implementar la pantalla en la que se desarrolla el juego
- Utilizaremos lo que se conoce como ciclo del juego
  - Bucle infinito
  - En cada iteración:
    - Lee la entrada
    - Actualiza la escena según la entrada e interacciones entre objetos de la misma
    - Redibuja los gráficos de la escena
    - Duerme durante un tiempo para que los ciclos tengan una duración uniforme
  - En la clase `GameCanvas` de MIDP 2.0 encontraremos facilidades para la creación de este ciclo

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-20

---

---

---

---

---

---

---

---

## Ciclo del juego



- Crearemos el ciclo en una clase que herede de `GameCanvas`
  - El ciclo se ejecutará dentro de un hilo
  - Podemos poner en marcha el hilo en el evento `showNotify`

```
Graphics g = getGraphics();
long t1, t2, td;
while(true) {
 t1 = System.currentTimeMillis();
 int keyState = getKeyStates();
 tick(keyState);
 render(g);
 flushGraphics();
 t2 = System.currentTimeMillis();
 td = t2 - t1; td = td < CICLO ? td : CICLO;
 try {
 Thread.sleep(CICLO - td);
 } catch (InterruptedException e) { }
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-21

---

---

---

---

---

---

---

---

## Máquina de estados



- Durante el desarrollo del juego pasaremos por diferentes estados
  - En cada uno se permitirán realizar determinadas acciones y se mostrarán determinados gráficos
  - Según el estado en el que nos encontremos el ciclo del juego realizará tareas distintas



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-22

## Juegos



- Juegos para móviles
- Desarrollo de juegos
- Motor del juego
- Entrada de usuario
- Componentes de la pantalla

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-23

## Entrada en MIDP 1.0



- En MIDP 1.0 podemos utilizar las acciones de juego para responder a los eventos del teclado
- Para conocer las teclas que se presionen deberemos capturar el evento de pulsación del teclado

```
public void keyPressed(int keyCode) {
 int action = getGameAction(keyCode);
 if (action == LEFT) {
 moverIzquierda();
 } else if (action == RIGHT) {
 moverDerecha();
 } else if (action == FIRE) {
 disparar();
 }
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-24

## Entrada en MIDP 2.0



- No hará falta capturar los eventos del teclado
- El método `getKeyStates` nos dirá las teclas pulsadas actualmente
  - Es más apropiado para implementar el ciclo del juego
  - La entrada del usuario se leerá de forma síncrona con el ciclo del juego
- Nos devolverá un entero en el que cada bit codifica la pulsación de una tecla

```
int keyState = getKeyStates();
```

- Podremos saber si una determinada tecla está pulsada utilizando una máscara como la siguiente

```
if ((keyState & LEFT_PRESSED) != 0) {
 moverIzquierda();
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-25

---

---

---

---

---

---

---

---

## Juegos



- Juegos para móviles
- Desarrollo de juegos
- Motor del juego
- Entrada de usuario
- Componentes de la pantalla

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-26

---

---

---

---

---

---

---

---

## Sprites



- Objetos que aparecen en la escena
  - Se mueven o podemos interactuar con ellos de alguna forma



- Lo creamos a partir de la imagen con el mosaico de *frames*

```
Sprite personaje = new Sprite(imagen,
 ancho_frame, alto_frame);
```

- Podremos animar este *sprite* por la pantalla

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-27

---

---

---

---

---

---

---

---

## Animación de los sprites



### ▪ Podemos mover el *sprite* por la pantalla

#### ➢ Situar en una posición absoluta

```
personaje.setPosition(x, y);
```

#### ➢ Desplazar respecto la posición actual

```
personaje.move(dx, dy);
```

### ▪ Podemos cambiar el *frame* del *sprite* para animarlo

```
personaje.setFrame(indice);
```

#### ➢ Podemos establecer una secuencia de frames para la animación

```
this.setFrameSequence(new int[]{ 4, 5, 6});
```

#### ➢ Para cambiar al siguiente frame de la secuencia actual llamaremos a

```
this.nextFrame();
```

#### ➢ Para volver a disponer de la secuencia completa llamaremos a

```
personaje.setFrameSequence(null);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-28

---

---

---

---

---

---

---

---

## Colisiones de los sprites



### ▪ Muchas veces necesitaremos conocer cuando dos *sprites* “chocan” entre ellos. Por ejemplo

#### ➢ Cuando el *sprite* de un enemigo toque a nuestro personaje, perderemos una vida

#### ➢ Cuando una de nuestras balas impacten contra un enemigo, el enemigo morirá

### ▪ Esta información la obtendremos mediante cálculo de colisiones

```
personaje.collidesWith(enemigo, false);
```

#### ➢ El rectángulo (*bounding box*) que se utilizará para calcular las colisiones tendrá como tamaño el tamaño de los *frames* de la imagen

#### ➢ Podremos cambiar este rectángulo con

```
this.defineCollisionRectangle(x, y, ancho, alto);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-29

---

---

---

---

---

---

---

---

## Fondo



### ▪ Los *sprites* se moverán sobre un escenario de fondo

#### ➢ El escenario muchas veces es más grande que la pantalla

#### ➢ No es conveniente crear una imagen con todo el fondo ya que será demasiado grande

### ▪ Podemos construir el fondo como un mosaico

#### ➢ Necesitaremos una imagen con los elementos básicos

#### ➢ Compondremos el fondo a partir de estos elementos



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-30

---

---

---

---

---

---

---

---

## Mosaico de fondo



- MIDP 2.0 nos proporciona la clase `TiledLayer` con la que crear el mosaico de fondo

```
TiledLayer fondo = new TiledLayer(columnas, filas,
 imagen, ancho, alto);
```

### ➤ Donde

- `(columnas x filas)` serán las dimensiones del mosaico en número de celdas
- `(ancho x alto)` serán las dimensiones en píxeles de cada elemento del mosaico

### ➤ Por lo tanto, el fondo generado tendrá unas dimensiones en píxeles de `(columnas*ancho) x (filas*alto)`

### ➤ Para fijar el tipo de elemento de una celda utilizamos

```
fondo.setCell(columna, fila, indice);
```

### ➤ Con el índice indicamos el elemento que se mostrará en dicha celda

- Los elementos de la imagen se empezarán a numerar a partir de 1
- Con 0 especificamos que la deje vacía (con el color del fondo)
- Utilizaremos valores negativos para crear animaciones

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-31

---

---

---

---

---

---

---

---

## Pantalla



- En la pantalla deberemos mostrar todos los elementos de la escena

### ➤ Fondo

### ➤ Sprites

- Podemos considerar que todos estos elementos son capas

### ➤ Tanto `Sprite` como `TiledLayer` derivan de la clase `Layer`

### ➤ Según el orden en el que se dibujen estas capas veremos que determinados objetos taparán a otros

- Podemos utilizar la clase `LayerManager` para crear esta estructura de capas

```
LayerManager escena = new LayerManager();
escena.append(personaje);
escena.append(enemigo);
escena.append(fondo);
```

- La primera capa que añadamos será la que quede más cerca del observador, y tapará a las demás capas cuando esté delante de ellas

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-32

---

---

---

---

---

---

---

---

## Volcado de los gráficos



- Si tenemos una escena de gran tamaño, podemos hacer que sólo se muestre un determinado recuadro

```
escena.setViewWindow(x, y, ancho, alto);
```

- Esto nos permitirá implementar fácilmente *scroll* en el fondo

### ➤ Haremos que el visor se vaya desplazando por la escena conforme el personaje se mueve

- Para volcar los gráficos en la pantalla utilizaremos

```
escena.paint(g, x, y);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Juegos-33

---

---

---

---

---

---

---

---



## Sesión 9: Almacenamiento persistente

---

---

---

---

---

---

---

---

### Índice



- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

---

---

---

---

---

---

---

---

### Almacenamiento persistente



- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

---

---

---

---

---

---

---

---

## RMS



- **RMS = *Record Management System***
  - Nos permite almacenar datos de forma persistente
  - Esta API se encuentra en `javax.microedition.rms`
- **No se especifica la forma en la que se guardan realmente los datos**
  - Deben guardarse en cualquier memoria no volátil
- **Los datos se guardan en almacenes de registros**
  - Un almacén de registros contiene varios registros
  - Cada registro contiene
    - Un identificador
    - Un *array* de *bytes* como datos

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-4

---

---

---

---

---

---

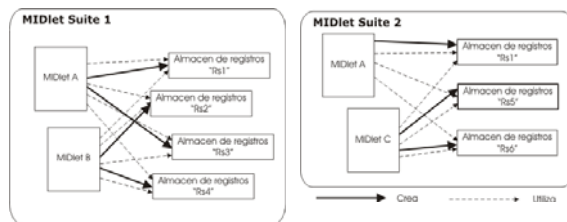
---

---

## Almacenes de registros



- **Un MIDlet puede crear y acceder a varios almacenes**
- **Los almacenes son privados de cada *suite***



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-5

---

---

---

---

---

---

---

---

## Operaciones con los almacenes



- **Abrir/crear un almacén**

```
RecordStore rs =
 RecordStore.openRecordStore(nombre, true);
```

- **Cerrar un almacén**

```
rs.closeRecordStore();
```

- **Listar los almacenes disponibles**

```
String [] nombres = RecordStore.listRecordStores();
```

- **Eliminar un almacén**

```
RecordStore.deleteRecordStore(nombre);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-6

---

---

---

---


---

---

---

---



Propiedades de los almacenes


- Nombre

String nombre = rs.getName();
- Fecha de modificación

long timestamp = rs.getLastModified();
- Versión

int version = rs.getVersion();
- Tamaño

int tam = rs.getSize();
- Tamaño disponible

int libre = rs.getSizeAvailable();

Programación de Dispositivos Móviles
© 2006 Depto. Ciencia Computación e IA
Almacenamiento persistente-7

---

---

---


---

---

---

---

---

Almacenamiento persistente


- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

Programación de Dispositivos Móviles
© 2006 Depto. Ciencia Computación e IA
Almacenamiento persistente-8

---

---

---


---

---

---

---

---

Conjunto de registros


- Cada almacén contendrá un conjunto de registros
- Cada registro tiene
  - Identificador
    - Valor entero
  - Datos
    - Array de bytes
- El identificador se autoincrementará con cada inserción
- Deberemos codificar los datos en binario para añadirlos en un registro
  - Utilizar objetos `DataInputStream` y `DataOutputStream`
  - Podemos utilizar los métodos de serialización de los objetos

| Identificador | Datos           |
|---------------|-----------------|
| 1             | A4 5D 12 09 ... |
| 2             | 32 3E 1A 98 ... |
| 3             | FE 26 3B 45 ... |

Programación de Dispositivos Móviles
© 2006 Depto. Ciencia Computación e IA
Almacenamiento persistente-9

---

---

---

---

---

---

---

---

## Añadir datos



### ▪ Codificar los datos en binario

```
ByteArrayOutputStream baos =
 new ByteArrayOutputStream();
DataOutputStream dos =
 new DataOutputStream(baos);
dos.writeUTF(nombre);
dos.writeInt(edad);
byte [] datos = baos.toByteArray();
```

### ▪ Añadir los datos como registro al almacén

```
int id = rs.addRecord(datos, 0, datos.length);
0
rs.setRecord(id, datos, 0, datos.length);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-10

---

---

---

---

---

---

---

---

## Consultar y borrar datos



### ▪ Leemos el registro del almacén

```
byte [] datos = rs.getRecord(id);
```

### ▪ Descodificamos los datos

```
ByteArrayInputStream bais =
 new ByteArrayInputStream(datos);
DataInputStream dis = DataInputStream(bais);
String nombre = dis.readUTF();
String edad = dis.readInt();
```

### ▪ Eliminar un registro

```
rs.deleteRecord(id);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-11

---

---

---

---

---

---

---

---

## Almacenamiento persistente



### ▪ Almacenes de registros

### ▪ Registros

### ▪ Consultas de registros

### ▪ Listener del registro

### ▪ Optimización de consultas

### ▪ Patrón de diseño adaptador

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-12

---

---

---

---

---

---

---

---

## Enumeración de registros



- Normalmente no conoceremos el identificador del registro buscado *a priori*
  - Podremos recorrer el conjunto de registros para buscarlo
  - Utilizaremos un objeto `RecordEnumeration`

```
RecordEnumeration re =
 rs.enumerateRecords(null, null, false);
```

- Recorremos la enumeración

```
while(re.hasNextElement()) {
 int id = re.nextRecordId();
 byte [] datos = rs.getRecord(id);
 // Procesar datos obtenidos
 ...
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-13

---

---

---

---

---

---

---

---

## Ordenación de registros



- Creamos un comparador

```
public class MiComparador implements RecordComparator {
 public int compare(byte [] reg1, byte [] reg2) {
 if(/* reg1 es anterior a reg2 */) {
 return RecordComparator.PRECEDES;
 } else if(/* reg1 es posterior a reg2 */) {
 return RecordComparator.FOLLOWS;
 } else if(/* reg1 es igual a reg2 */) {
 return RecordComparator.EQUIVALENT;
 }
 }
}
```

- Obtenemos la enumeración

```
RecordEnumeration re =
 rs.enumerateRecords(new MiComparador(), null, false);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-14

---

---

---

---

---

---

---

---

## Filtrado de registros



- Creamos un filtro

```
public class MiFiltro implements RecordFilter {
 public boolean matches(byte [] reg) {
 if(/* reg nos interesa */) {
 return true;
 } else {
 return false;
 }
 }
}
```

- Obtenemos la enumeración

```
RecordEnumeration re =
 rs.enumerateRecords(null, new MiFiltro(), false);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-15

---

---

---

---

---

---

---

---

## Almacenamiento persistente



- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-16

---

---

---

---

---

---

---

---

## Listener



- Nos permite “escuchar” cambios en el registro

```
public class MiListener implements RecordListener {
 public void recordAdded(RecordStore rs, int id) {
 // Añadido un registro con identificador id a rs
 }
 public void recordChanged(RecordStore rs, int id) {
 // Modificado el registro con identificador id en rs
 }
 public void recordDeleted(RecordStore rs, int id) {
 // Eliminado el registro con identificador id de rs
 }
}
```

- Registrar el listener

```
rs.addRecordListener(new MiListener());
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-17

---

---

---

---

---

---

---

---

## Almacenamiento persistente



- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-18

---

---

---

---

---

---

---

---

## Consultas



- Necesitamos realizar consultas en el almacén
  - Buscar registros que cumplan ciertos criterios
- Podemos utilizar una enumeración con un `RecordFilter`
- Esto nos forzará a recorrer todos los registros del almacén
  - Deserializar cada registro
  - Comprobar si los datos cumplen los criterios de la búsqueda
- Si tenemos almacenado un gran volumen de datos, hará que las consultas sean lentas

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-19

## Índices



- Podemos optimizar las consultas creando un almacén de índices
  - Tendremos un índice por cada registro almacenado
  - Los índices contendrán sólo los datos por los que se suele realizar la búsqueda
  - Además contendrán una referencia al registro donde se encuentra almacenado el dato al que corresponde

```
public class Cita {
 Date fecha;
 String asunto;
 String descripcion;
 String lugar;
 String contacto;
 boolean alarma;
}
```

```
public class IndiceCita {
 int id;
 Date fecha;
 boolean alarma;
}
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-20

## Almacenamiento persistente



- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Almacenamiento persistente-21

## Adaptador



- Para implementar el acceso a RMS conviene utilizar el patrón de diseño adaptador
  - Interfaz adaptada al dominio de nuestra aplicación, que encapsula una API genérica y nos aísla de ella
- Por ejemplo, para nuestra aplicación de citas
  - En RMS tenemos un método `getRecord`
  - En nuestro adaptador tenemos un método `getCita`
- Desde nuestra aplicación siempre accederemos al registro a través del adaptador

---

---

---

---

---

---

---

---

## Ejemplo de adaptador



```
public class AdaptadorRMS {

 public final static String RS_DATOS = "rs_datos";
 RecordStore rsDatos;

 public AdaptadorRMS() throws RecordStoreException {
 rsDatos = RecordStore.openRecordStore(RS_DATOS, true);
 }

 public int addCita(Cita cita)
 throws IOException, RecordStoreException {
 ByteArrayOutputStream baos = new ByteArrayOutputStream();
 DataOutputStream dos = new DataOutputStream(baos);
 cita.serialize(dos);
 byte[] datos = baos.toByteArray();
 int id = rsDatos.addRecord(datos, 0, datos.length);
 return id;
 }
}
```

---

---

---

---

---

---

---

---

## Clave primaria



- Necesitamos una clave primaria para poder referenciar cada registro
  - Podemos utilizar el identificador del registro en RMS
  - Deberemos guardarnos una referencia a este ID al leer los datos para posteriormente poderlo modificar o eliminar

```
public Cita getCita(int id)
 throws RecordStoreException, IOException {
 byte[] datos = rsDatos.getRecord(id);
 ByteArrayInputStream bais =
 new ByteArrayInputStream(datos);
 DataInputStream dis = new DataInputStream(bais);
 Cita cita = Cita.deserialize(dis);
 cita.setRmsID(id);
 return cita;
}
```

---

---

---

---

---

---

---

---



## Sesión 10: Conexiones de red

---

---

---

---

---

---

---

### Índice



- Marco de conexiones genéricas
- Conexión HTTP
- Envío y recepción de datos
- Conexiones a bajo nivel
- Mensajes SMS
- Bluetooth
- Servicios Web

---

---

---

---

---

---

---

### Conexiones de red



- Marco de conexiones genéricas
- Conexión HTTP
- Envío y recepción de datos
- Conexiones a bajo nivel
- Mensajes SMS
- Bluetooth
- Servicios Web

---

---

---


---

---

---

---

GCF



- GCF = Generic Connection Framework**
  - Marco de conexiones genéricas, en `javax.microedition.io`
    - Permite establecer conexiones de red independientemente del tipo de red del móvil (circuitos virtuales, paquetes, etc)
- Cualquier tipo conexión se establece con un único método genérico**

```
Connection con = Connector.open(url);
```
- Según la URL podemos establecer distintos tipos de conexiones**

|                                          |              |
|------------------------------------------|--------------|
| <code>http://jee.ua.es/pdm</code>        | HTTP         |
| <code>datagram://192.168.0.4:6666</code> | Datagramas   |
| <code>socket://192.168.0.4:4444</code>   | Sockets      |
| <code>comm:0;baudrate=9600</code>        | Puerto serie |
| <code>file:/fichero.txt</code>           | Ficheros     |

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-4

---

---

---

---

---

---


---

---

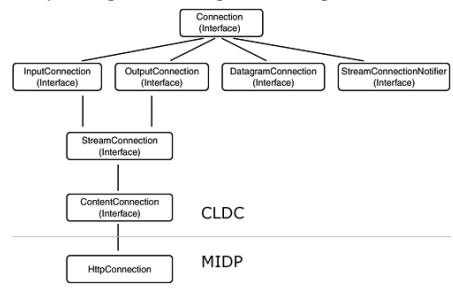
---

---

Tipos de conexiones



- En CLDC se implementan conexiones genéricas
- En MIDP y APIs opcionales se implementan los protocolos concretos



```

graph TD
 C[Connection Interface] --> IC[InputConnection Interface]
 C --> OC[OutputConnection Interface]
 C --> DC[DatagramConnection Interface]
 C --> SCN[StreamConnectionNotifier Interface]
 IC --> SC[StreamConnection Interface]
 OC --> SC
 SC --> CC[ContentConnection Interface]
 CC --> HC[HttpConnection]

```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-5

---

---

---

---

---

---


---

---

---

---

Conexiones de red



- Marco de conexiones genéricas
- Conexión HTTP**
- Envío y recepción de datos
- Conexiones a bajo nivel
- Mensajes SMS
- Bluetooth
- Servicios Web

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-6

---

---

---

---

---

---

---

---

---

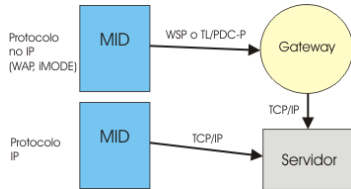
---



## Conexión HTTP



- El único protocolo que se nos asegura que funcione en todos los móviles es HTTP
  - Funcionará siempre de la misma forma, independientemente del tipo de red que haya por debajo



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-7

---

---

---

---

---

---

---

---

## Leer de una URL



- Abrimos una conexión con la URL

```
HttpConnection con = (HttpConnection)Connector.open(
 "http://j2ee.ua.es/index.htm");
```

- Abrimos un flujo de entrada de la conexión

```
InputStream in = con.openInputStream();
```

- Podremos leer el contenido de la URL utilizando este flujo de entrada

- Por ejemplo, en caso de ser un documento HTML, leeremos su código HTML

- Cerramos la conexión

```
in.close();
con.close();
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-8

---

---

---

---

---

---

---

---

## Mensaje de petición



- Podemos utilizar distintos métodos

```
HttpConnection.GET
HttpConnection.POST
HttpConnection.HEAD
```

- Para establecer el método utilizaremos:

```
con.setRequestMethod(HttpConnection.GET);
```

- Podemos añadir cabeceras HTTP a la petición

```
con.setRequestProperty(nombre, valor);
```

- Por ejemplo:

```
c.setRequestProperty("User-Agent",
 "Profile/MIDP-1.0 Configuration/CLDC-1.0");
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-9

---

---

---

---

---

---

---

---

## Mensaje de respuesta



- A parte de leer el contenido de la respuesta, podemos obtener

- Código de estado

```
int cod = con.getResponseCode();
String msg = con.getResponseMessage();
```

- Cabeceras de la respuesta

```
String valor = con.getHeaderField(nombre);
```

- Tenemos métodos específicos para cabeceras estándar

```
getLength()
getType()
getLastModified()
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-10

---

---

---

---

---

---

---

---

## Conexiones de red



- Marco de conexiones genéricas
- Conexión HTTP
- Envío y recepción de datos
- Conexiones a bajo nivel
- Mensajes SMS
- Bluetooth
- Servicios Web

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-11

---

---

---

---

---

---

---

---

## Enviar datos



- Utilizar parámetros

- GET o POST

- Parejas <nombre, valor>

```
HttpConnection con = (HttpConnection)Connector.open(
 "http://j2ee.ua.es/registra?nombre=Pedro&edad=23");
```

- No será útil para enviar estructuras complejas de datos

- Añadir los datos al bloque de contenido de la petición

- Debemos decidir la codificación a utilizar

- Por ejemplo, podemos codificar en binario con  
DataOutputStream

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-12

---

---

---

---

---

---

---

---

## Tipos de contenido



- Para enviar datos en el bloque de contenido debemos especificar el tipo MIME de estos datos

➤ Lo establecemos mediante la cabecera `Content-Type`

```
con.setRequestProperty("Content-Type", "text/plain");
```

➤ Por ejemplo, podemos usar los siguientes tipos:

|                                                |                 |
|------------------------------------------------|-----------------|
| <code>application/x-www-form-urlencoded</code> | Formulario POST |
| <code>text/plain</code>                        | Texto ASCII     |
| <code>application/octet-stream</code>          | Datos binarios  |

---

---

---

---

---

---

---

---

## Codificación de los datos



- Podemos codificar los datos a enviar en binario

➤ Establecemos el tipo MIME adecuado

```
con.setRequestProperty("Content-Type",
 "application/octet-stream");
```

➤ Utilizaremos un objeto `DataOutputStream`

```
DataOutputStream dos = con.openDataOutputStream();
dos.writeUTF(nombre);
dos.writeInt(edad);
dos.flush();
```

- Si hemos definido serialización para los objetos, podemos utilizarla para enviarlos por la red

---

---

---

---

---

---

---

---

## Leer datos de la respuesta



- Contenido de la respuesta HTTP

➤ No sólo se puede utilizar HTML

➤ El servidor puede devolver contenido de cualquier tipo

➤ Por ejemplo, XML, ASCII, binario, etc

- Si el servidor nos devuelve datos binarios, podemos descodificarlos mediante `DataInputStream`

```
DataInputStream dis = con.openDataInputStream();
String nombre = dis.readUTF();
int precio = dis.readInt();
dis.close();
```

- Podría devolver objetos serializados

➤ Deberíamos deserializarlos con el método adecuado

---

---

---

---

---

---

---

---

## Conexiones de red



- Marco de conexiones genéricas
- Conexión HTTP
- Envío y recepción de datos
- **Conexiones a bajo nivel**
- Mensajes SMS
- Bluetooth
- Servicios Web

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-16

---

---

---

---

---

---

---

---

## Conexiones a bajo nivel



- A partir de MIDP 2.0 se incorporan a la especificación conexiones de bajo nivel
  - Sockets
  - Datagramas
- Nos permitirán aprovechar las características de las nuevas redes de telefonía móvil
- Podremos acceder a distintos servicios de Internet directamente
  - Por ejemplo correo electrónico
- Su implementación es optativa en los dispositivos MIDP 2.0
  - Depende de cada fabricante

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-17

---

---

---

---

---

---

---

---

## Sockets



- **Establecer una comunicación por sockets**

```
SocketConnection sc = (SocketConnection)
Connector.open("socket://host:puerto");
```

- **Abrir flujos de E/S para comunicarnos**

```
InputStream in = sc.openInputStream();
OutputStream out = sc.openOutputStream();
```

- **Podemos crear un socket servidor y recibir conexiones entrantes**

```
ServerSocketConnection ssc = (ServerSocketConnection)
Connector.open("socket://:puerto");
SocketConnection sc =
(SocketConnection) ssc.acceptAndOpen();
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-18

---

---

---

---

---

---

---

---

## Datagramas



- Crear conexión por datagramas

```
DatagramConnection dc = (DatagramConnection)
Connector.open("datagram://host:puerto");
```

- Crear un enviar paquete de datos

```
Datagram dg = dc.newDatagram(datos, datos.length);
dc.send(dg);
```

- Recibir paquete de datos

```
Datagram dg = dc.newDatagram(longitud);
dc.receive(dg);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-19

---

---

---

---

---

---

---

---

## Conexiones de red



- Marco de conexiones genéricas
- Conexión HTTP
- Envío y recepción de datos
- Conexiones a bajo nivel
- Mensajes SMS
- Bluetooth
- Servicios Web

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-20

---

---

---

---

---

---

---

---

## Conexión de mensajes



- Con WMA podremos crear conexiones para enviar y recibir mensajes de texto SMS

- Utilizaremos una URL como

```
sms://telefono:[puerto]
```

- Creamos la conexión

```
MessageConnection mc = (MessageConnection)
Connector.open("sms://+34555000000");
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-21

---

---

---

---

---

---

---

---

## Envío de mensajes



### ▪ Componemos el mensaje

```
String texto =
 "Este es un mensaje corto de texto";
TextMessage msg = mc.newMessage(mc.TEXT_MESSAGE);
msg.setPayloadText(texto);
```

### ▪ El mensaje no deberá pasar de 140 bytes

- Si se excede, podría ser fraccionado
- Si no puede ser fraccionado, obtendremos un error

### ▪ Enviamos el mensaje

```
mc.send(msg);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-22

---

---

---

---

---

---

---

---

## Recepción de mensajes



### ▪ Creamos conexión de mensajes entrantes

```
MessageConnection mc = (MessageConnection)
 Connector.open("sms://:4444");
```

### ▪ Recibimos el mensaje

```
Message msg = mc.receive();
```

### ▪ Esto nos bloqueará hasta la recepción

- Para evitar estar bloqueados, podemos utilizar un listener
- Con un `MessageListener` se nos notificará de la llegada de mensajes

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-23

---

---

---

---

---

---

---

---

## Conexiones de red



### ▪ Marco de conexiones genéricas

### ▪ Conexión HTTP

### ▪ Envío y recepción de datos

### ▪ Conexiones a bajo nivel

### ▪ Mensajes SMS

### ▪ Bluetooth

### ▪ Servicios Web

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-24

---

---

---

---

---

---

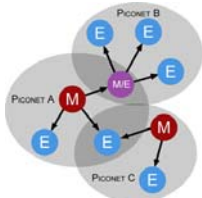
---

---

## Redes bluetooth



- Las redes bluetooth son redes “ad hoc”
  - La red se crea dinámicamente
  - Tenemos la capacidad de “descubrir” dispositivos
  - Conecta pequeños dispositivos (sustituye al cable)
- Topología



Piconet: 1 maestro con 7 esclavos como máximo

Scatternet: Conexión de varias piconets

Programación de Dispositivos Móviles

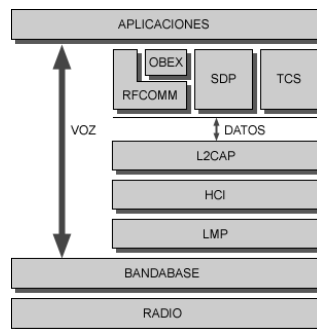
© 2006 Depto. Ciencia Computación e IA

Conexiones de red-25

## Protocolos bluetooth



- **L2CAP**
  - Protocolo a bajo nivel
  - Transmisión de paquetes
  - Sin control de flujo
- **RFCOMM**
  - Puerto serie sobre radio
  - Realiza control de flujo
- **SDP**
  - Descubrimiento de dispositivos



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-26

## Servicios



- Los servicios se identifican mediante un **UUID**
  - *Universal Unique Identifier*
  - Clave de 128 bits, única en el tiempo y en el espacio
  - Se puede generar con herramientas como `uuidgen`
- Podemos buscar dispositivos y explorar los servicios que ofrecen
  - Los servicios se buscarán mediante su UUID
  - Tipos de búsqueda de dispositivos:
    - GIAC:** General.
      - Encuentra tanto dispositivos descubribles GIAC como LIAC.
    - LIAC:** Limitada. Para búsquedas acotadas.
      - Sólo encuentra dispositivos descubribles LIAC.

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-27

## Publicar un servicio (servidor)



- Generar UUID para nuestro servicio

```
public final static String UUID =
 "000000000000010008000123456789ab";
```

- Hacemos nuestro dispositivo local descubrible

```
LocalDevice ld = LocalDevice.getLocalDevice();
ld.setDiscoverable(DiscoveryAgent.GIAC);
```

- Crear servicio

```
StreamConnectionNotifier scn =
 (StreamConnectionNotifier)
 Connector.open("btspp://localhost:" + UUID);
```

- Aceptar conexiones entrantes

```
StreamConnection sc =
 (StreamConnection)scn.acceptAndOpen();
InputStream is = sc.openInputStream();
OutputStream os = sc.openOutputStream();
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-28

---

---

---

---

---

---

---

---

## Descubrir dispositivos y servicios (cliente)



- Obtener agente de descubrimiento

```
LocalDevice ld = LocalDevice.getLocalDevice();
DiscoveryAgent da = ld.getDiscoveryAgent();
```

- Crear un objeto DiscoveryListener

```
deviceDiscovered(RemoteDevice rd, DeviceClass dc);
inquiryCompleted(int tipo);
servicesDiscovered(int transID, ServiceRecord[] srvs);
serviceSearchCompleted(int transID, int estado);
```

- Comenzar búsqueda de dispositivos

```
da.startInquiry(DiscoveryAgent.GIAC, mListener);
```

- Buscar servicios de un dispositivo

```
da.searchServices(null, new UUID[]{new UUID(UUID,false)},
 (RemoteDevice)obtenerDispositivoRemoto(),mListener);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-29

---

---

---

---

---

---

---

---

## Conectar a un servicio (cliente)



- Una vez descubiertos los servicios de nuestro entorno, habremos obtenido varios objetos `ServiceRecord` con cada uno de estos servicios

```
ServiceRecord rs =
 (ServiceRecord)obtenerServicioRemoto();
```

- Obtener URL de conexión al servicio

```
String url = rs.getConnectionURL(
 ServiceRecord.NOAUTHENTICATE_NOENCRYPT, true);
```

- Establecer la conexión

```
StreamConnection sc =
 (StreamConnection)Connector.open(url);
InputStream is = sc.openInputStream();
OutputStream os = sc.openOutputStream();
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Conexiones de red-30

---

---

---

---

---

---

---

---

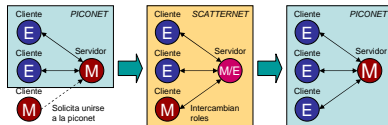


## Roles maestro/esclavo



- Quien realiza la conexión (cliente) actuará como maestro
  - Indicando `;"master=true"` en la URL de quien publica el servicio (servidor) podemos forzar a que sea éste quien se comporte como maestro

```
Connector.open("btspp://localhost:" + UUID + ";" + "master=true");
```

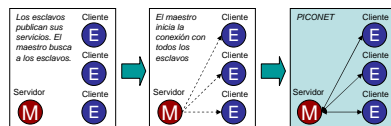


- Esto no funciona en dispositivos que no soporten el intercambio de roles maestro/esclavo

## Conexión punto-a-multipunto



- Para poder hacer conexiones punto-a-multipunto en estos casos deberemos:
  - Abrir varios dispositivos que publiquen servicios.
    - En este caso, éstos serán los “clientes”.
  - Iniciar la conexión a todos estos clientes desde un único maestro
    - Este maestro será el “servidor”, ya que es quien coordinará las múltiples conexiones de los clientes.



## Seguridad en bluetooth



- Podemos forzar la utilización de diferentes tipos de seguridad en las conexiones bluetooth:
  - Autenticación (`;"authenticate=true"`)
    - Los usuarios de los móviles deben conocerse
    - Se resuelve mediante “emparejamiento” (*pairing*)
    - Los usuarios de los móviles que se conectan deben introducir un mismo código secreto en sendos dispositivos
  - Autorización (`;"authorize=true"`)
    - Quien solicita la conexión a un servicio debe tener autorización
    - Si no está en una lista de dispositivos de confianza, se preguntará al usuario si acepta la conexión
  - Encriptación (`;"encrypt=true"`)
    - Los datos se transmiten encriptados
    - Requiere estar autenticado

## Conexiones de red

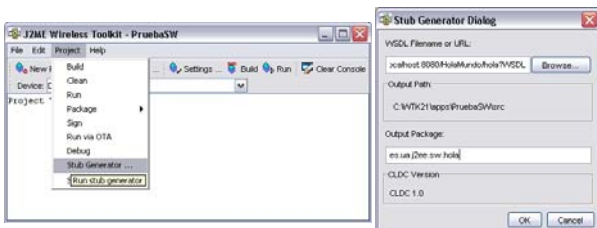


- Marco de conexiones genéricas
- Conexión HTTP
- Envío y recepción de datos
- Conexiones a bajo nivel
- Mensajes SMS
- Bluetooth
- Servicios Web

## Servicios web



- Podemos acceder a servicios web desde dispositivos mediante Web Services API
  - Los servicios deben ser de tipo `document/literal`
- Podemos generar un stub mediante WTK 2.2



## Invocación de servicios



- Utilizamos el stub para acceder al servicio

```
HolaMundoIF hola = new HolaMundoIF_Stub();
try {
 String saludo = hola.saluda("Miguel");
} catch (RemoteException re) { // Error }
```

- Los servicios web requieren
  - Gran cantidad de memoria
  - Gran cantidad de procesamiento
  - Gran cantidad de tráfico por la red (XML)
- Esto los hace poco adecuados para los dispositivos actuales

## Programación de Dispositivos Móviles



### Sesión 11: Aplicaciones corporativas

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-1

---

---

---

---

---

---

---

---

## Índice



- Registro push
- Seguridad
- Front-end de aplicaciones corporativas
- Integración con aplicaciones corporativas
- Arquitectura MVC
- Modo sin conexión

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-2

---

---

---

---

---

---

---

---

## Aplicaciones corporativas



- Registro push
- Seguridad
- Front-end de aplicaciones corporativas
- Integración con aplicaciones corporativas
- Arquitectura MVC
- Modo sin conexión

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-3

---

---

---

---

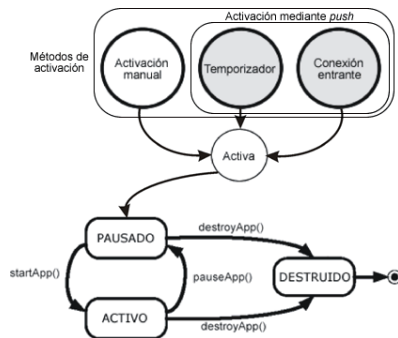
---

---

---

---

## Activación por push



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-4

## Conexiones entrantes



- Podemos hacer que la aplicación se active cuando se produzca una conexión entrante
  - Sockets, datagramas, mensajes, bluetooth
- Normalmente en el móvil no tendremos una IP fija, por lo que los sockets y los datagramas no son adecuados
- Podemos registrar la conexión push de dos formas:
  - Estática, en el fichero JAD

```
MIDlet-Push-1: sms://:4444,
es.ua.j2ee.sms.MIDletRecibirSMS, *
```

- Dinámica, utilizando la API de PushRegistry

```
PushRegistry.registerConnection(url, nombreClaseMIDlet,
remitentesPermitidos);
```

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-5

## Temporizadores



- Podemos hacer que la aplicación se active a una determinada hora
- Registraremos un temporizador push con
 

```
long t = PushRegistry.registerAlarm(
 midlet.getClass().getName(), fecha.getTime());
```
- Sólo podemos registrar un temporizador push
- La aplicación no tendrá constancia de que se ha activado mediante push
  - Podemos guardar en RMS información sobre la hora del temporizador
  - Si la aplicación se activa a esta hora, consideramos que ha sido mediante push
- Las conexiones push sólo serán efectivas cuando nuestra aplicación esté cerrada
  - Cuando esté abierta será responsabilidad de nuestro MIDlet responder a los temporizadores y a la conexiones entrantes

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-6

## Aplicaciones corporativas



- Registro push
- Seguridad
- Front-end de aplicaciones corporativas
- Integración con aplicaciones corporativas
- Arquitectura MVC
- Modo sin conexión

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-7

---

---

---

---

---

---

---

---

## Seguridad



- Para garantizar la seguridad, las aplicaciones MIDP se ejecutan dentro de una caja de arena (*sandbox*)
  - Entorno restringido
  - Evitar que causen daños a otras aplicaciones del dispositivo
- Están restringidas a acceder únicamente a los recursos de la *suite* de la aplicación
  - Sólo puede utilizar clases Java de su propia suite
  - Sólo puede leer recursos estáticos contenido dentro de su suite
  - Sólo puede acceder a almacenes de registros creados por MIDlets de su misma suite
  - No pueden acceder al sistema de ficheros del móvil, si necesitan almacenar datos debe hacerlo mediante RMS
  - Sólo pueden usar la API Java (MIDP), nunca a la API nativa

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-8

---

---

---

---

---

---

---

---

## Seguridad en MIDP 2.0



- Operaciones sensibles
  - Establecer conexiones de red
  - Registrar activación por push
- Debemos solicitar permiso para realizar estas operaciones
  - Lo haremos en el fichero JAD mediante el atributo `MIDlet-Permissions`
- Al instalar la aplicación se asigna a un dominio
  - Según el dominio se concederán ciertos permisos
- El dispositivo asignará dominios que otorguen permisos a las aplicaciones que sean de confianza

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-9

---

---

---

---

---

---

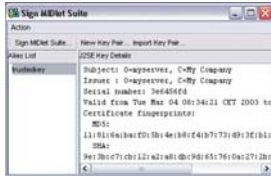
---

---

## Firmar aplicaciones



- Se recomienda que se utilicen firmas y certificados para decidir el dominio que se le asignará a cada aplicación
- Cada dispositivo tendrá almacenado un conjunto de firmas
  - Deberemos utilizar una de estas firmas para que nuestra aplicación sea de confianza
- Podemos utilizar WTK para realizar pruebas con MIDlets firmados



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-10

---

---

---

---

---

---

---

---

## Aplicaciones corporativas



- Registro push
- Seguridad
- **Front-end de aplicaciones corporativas**
- Integración con aplicaciones corporativas
- Arquitectura MVC
- Modo sin conexión

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-11

---

---

---

---

---

---

---

---

## Front-ends



- En un PC normalmente accedemos a las aplicaciones web mediante un navegador, a través de una interfaz HTML
- En dispositivos móviles podemos utilizar un paradigma similar, con lenguajes como WML o cHTML
- Sin embargo, la utilización de aplicaciones J2ME aporta las siguientes ventajas:
  - Interfaz de usuario flexible
  - Permiten trabajar sin conexión
  - Se conectan mediante protocolo HTTP estándar, no necesitaremos conocer el tipo de red subyacente

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-12

---

---

---

---

---

---

---

---

## Optimizaciones



- **Reducir el tráfico en la red**
  - Validar datos en el cliente
  - Mantener copias de los datos en local (RMS)
- **Operaciones de larga duración**
  - Accesos a RMS, conexiones de red
  - Realizar siempre desde un hilo
  - Proporcionar información al usuario sobre el progreso
  - Permitir interrumpir si es posible
- **Personalización**
  - Guardar las preferencias del usuario en el móvil
  - Recordar login y password para futuras sesiones

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-13

## Aplicaciones corporativas



- **Registro push**
- **Seguridad**
- **Front-end de aplicaciones corporativas**
- **Integración con aplicaciones corporativas**
- **Arquitectura MVC**
- **Modo sin conexión**

Programación de Dispositivos Móviles

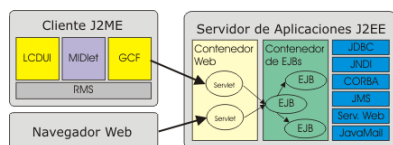
© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-14

## Comunicación con el servidor



- El MIDlet cliente utilizará:
  - GCF para comunicarse con el servidor web
  - LCDUI para la interfaz con el usuario
  - RMS para almacenar datos de forma local en el móvil
- En la aplicación web J2EE utilizaremos:
  - Un servlet que se comunique con el cliente J2ME
  - Podemos definir otro servlet para acceder mediante una interfaz web
  - Podemos reutilizar desde ambos servlets la misma lógica de negocio implementada mediante EJBs



Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-15

## Codificación de los datos



- En la comunicación con el servidor (Servlet) se debe acordar una codificación de los mensajes que ambos entiendan.
- **Binario**
  - Mensajes compactos y fáciles de analizar.
  - Alto acoplamiento.
  - Podemos utilizar la serialización de objetos definida en MIDP
    - Asegurarse de que el objeto es compatible con J2ME y J2EE
    - Tanto en el cliente como en el servidor se deberán utilizar los mismos métodos de serialización
- **XML**
  - Mensajes extensos y complejos de analizar por un móvil.
  - Bajo acoplamiento.

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-16

---

---

---

---

---

---

---

---

## Mantenimiento de sesiones



- Las sesiones normalmente se mantienen con elementos que gestionan los navegadores web como las cookies
- Para poder utilizar sesiones deberemos implementar en nuestro cliente alguno de los métodos existentes
  - Cookies
  - Reescritura de URLs
- Las cookies en algunos casos son filtradas por gateways
  - Será más conveniente utilizar reescritura de URLs

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-17

---

---

---

---

---

---

---

---

## Reescritura de URLs



- En el lado del servidor debemos obtener la URL reescrita

```
String url_con_ID = response.encodeURL(url);
```
- Se adjunta un identificador a dicha URL que identifica la sesión en la que nos encontramos
- Devolvermos la URL al cliente
  - Por ejemplo, mediante una cabecera HTTP

```
response.setHeader("URL-Reescrita", url_con_ID);
```
- La próxima vez que nos conectemos al servidor deberemos utilizar la URL reescrita
  - De esta forma el servidor sabrá que la petición la realiza el mismo cliente y podrá mantener la sesión

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-18

---

---

---

---

---

---

---

---



## Aplicaciones corporativas



- Registro push
- Seguridad
- Front-end de aplicaciones corporativas
- Integración con aplicaciones corporativas
- **Arquitectura MVC**
- Modo sin conexión

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-19

---

---

---

---

---

---

---

---

## MVC para aplicaciones J2ME



- Podemos aplicar el patrón de diseño Modelo-Vista-Controlador a las aplicaciones J2ME
- En esta arquitectura distinguimos:
  - Modelo
    - Datos de la aplicación
  - Vista
    - Presentación de la aplicación
    - Pantallas de nuestro MIDlet
  - Controlador
    - Controla el flujo de la aplicación
    - Decide qué pantalla mostrar y qué operaciones realizar en cada momento

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-20

---

---

---

---

---

---

---

---

## Modelo



- Tenemos aislados los datos del resto de la aplicación
  - Nos facilitará implementar la posibilidad de trabajar en modo *online* y modo *offline*
- Podemos dividir el modelo en dos subsistemas:
  - Modelo local
    - Accede a los datos almacenados localmente para trabajar *offline*
    - Puede utilizar un adaptador RMS para acceder a estos datos
  - Modelo remoto
    - Podemos definir un proxy para acceder al servidor
    - El proxy encapsula la conexión con el servidor para acceder a sus funcionalidades, proporcionándonos una interfaz local
- Podemos utilizar el patrón de diseño fachada para integrar estos dos subsistemas
  - Proporcionamos una interfaz única que nos dé acceso a ellos
  - Reduce la complejidad subyacente, aísla al resto de la aplicación

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-21

---

---

---

---

---

---

---

---

## Aplicaciones corporativas



- Registro push
- Seguridad
- Front-end de aplicaciones corporativas
- Integración con aplicaciones corporativas
- Arquitectura MVC
- Modo sin conexión

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-22

---

---

---

---

---

---

---

---

## Tipos de aplicaciones



- Según la forma de conectarse, podemos distinguir varios tipos de aplicaciones:
  - *Thin*
    - Todo el procesamiento se realiza en el servidor
    - Este tipo de aplicaciones son por ejemplo a las que accedemos mediante un navegador
    - Siempre necesitamos conexión para acceder a ellas
  - *Thick*
    - Aplicaciones dedicadas
    - Se instalan en el cliente para realizar una tarea concreta
    - Necesitan trabajar de forma coordinada con el servidor
  - *Standalone*
    - Todo el procesamiento se realiza en el cliente
    - Por ejemplo calculadora, bloc de notas, juegos, etc
    - Pueden conectarse eventualmente para actualizar datos, normalmente a petición del usuario

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-23

---

---

---

---

---

---

---

---

## Replica de datos



- Vamos a centrarnos en las aplicaciones *thick*
- Para permitir que estas aplicaciones trabajen sin conexión deberemos replicar los datos del servidor
  - Mantendremos una copia local de los datos
- El modelo de réplica se caracteriza por
  - ¿Se replican todos los datos o sólo una parte de ellos?
  - ¿Las estructuras de datos se replican fielmente o no?
  - ¿Los datos son de lectura/escritura o de sólo lectura?
  - ¿Los mismos datos pueden ser compartidos y replicados por muchos usuarios?
  - ¿Los datos tienen fecha de caducidad?

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-24

---

---

---

---

---

---

---

---

## Sincronización de datos



- Los datos en el cliente y en el servidor deberán ser consistentes
  - Deberemos sincronizar los datos para que los cambios hechos en cliente o servidor se actualicen en el otro lado
- Podemos distinguir tres formas de envío de datos:
  - El cliente descarga datos del servidor
    - Mantenemos una caché de datos
  - El cliente envía datos no compartidos al servidor
  - El cliente envía datos compartidos con otros usuarios al servidor
    - Este es el caso más problemático
    - Varios clientes pueden modificar sus copias locales concurrentemente y causar conflictos en la actualización de datos

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-25

---

---

---

---

---

---

---

---

## Caché de datos



- Debemos decidir cuando actualizar la caché
  - Si conocemos la fecha de caducidad podemos utilizar esta información
  - Si no la conocemos podemos conectar periódicamente al servidor o a petición del usuario
- Podemos utilizar *timestamps* para conocer qué datos no se han descargado todavía
  - A cada dato que se añade en el servidor se le asignará un *timestamp* superior al del anterior dato
  - El cliente conocerá el *timestamp* del último dato descargado
  - Cuando solicite datos al servidor, enviará este *timestamp* para que el servidor nos devuelva todos los datos posteriores
  - Recibiremos el servidor el *timestamp* correspondiente al último dato devuelto actualmente

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-26

---

---

---

---

---

---

---

---

## Enviar datos al servidor



- Si modificamos o creamos datos en el cliente deberemos actualizar los cambios en el servidor
- Podemos añadir a los datos almacenados localmente un *flag* que indique si el dato está pendiente de ser actualizado en el servidor
- Será conveniente que la granularidad de los datos sea lo más fina posible
  - Almacenar menor cantidad de datos juntos en un mismo registro
  - De esta forma actualizaremos sólo la porción modificada, y no toda la estructura

Programación de Dispositivos Móviles

© 2006 Depto. Ciencia Computación e IA

Aplicaciones corporativas-27

---

---

---

---

---

---

---

---