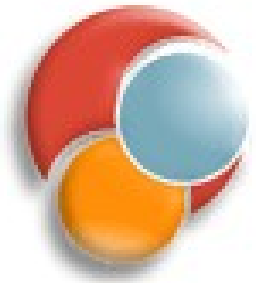


Android y Java para Dispositivos Móviles

Sesión 2: Introducción a los MIDs.
Java para MIDs.

MIDlets.

Puntos a tratar

- Características de los dispositivos
- Arquitectura de J2ME
- Construcción de aplicaciones MIDP
- Características de CLDC
- Temporizadores
- Serialización de objetos
- Acceso a los recursos
- MIDlets

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

tdt televisión digital terrestre mhp

sin interfaz

Blu-ray Disc

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

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

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

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)

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

Aplicaciones locales

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

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)

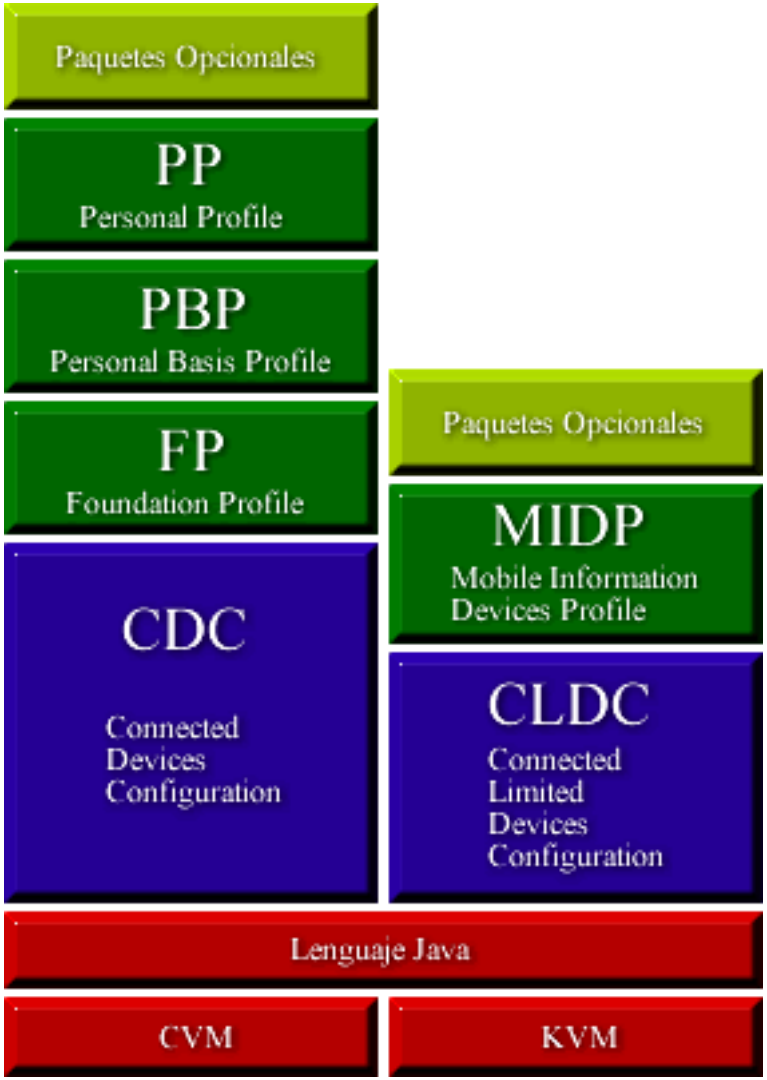
Java 2 Micro Edition

- Edición de la plataforma Java 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

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

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

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`

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

Más paquetes opcionales

- Location API
  - Localización física del dispositivo (GPS)
- Security and Trust Services API
  - Servicios de seguridad: encriptación, identificación, autentificación
- 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)

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 `MANIFEFT.MF` con información sobre la aplicación (algunos datos son replicados del fichero JAD).

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.

Fichero MANIFEST.MF

- Ejemplo de fichero `MANIFEST.MF`:

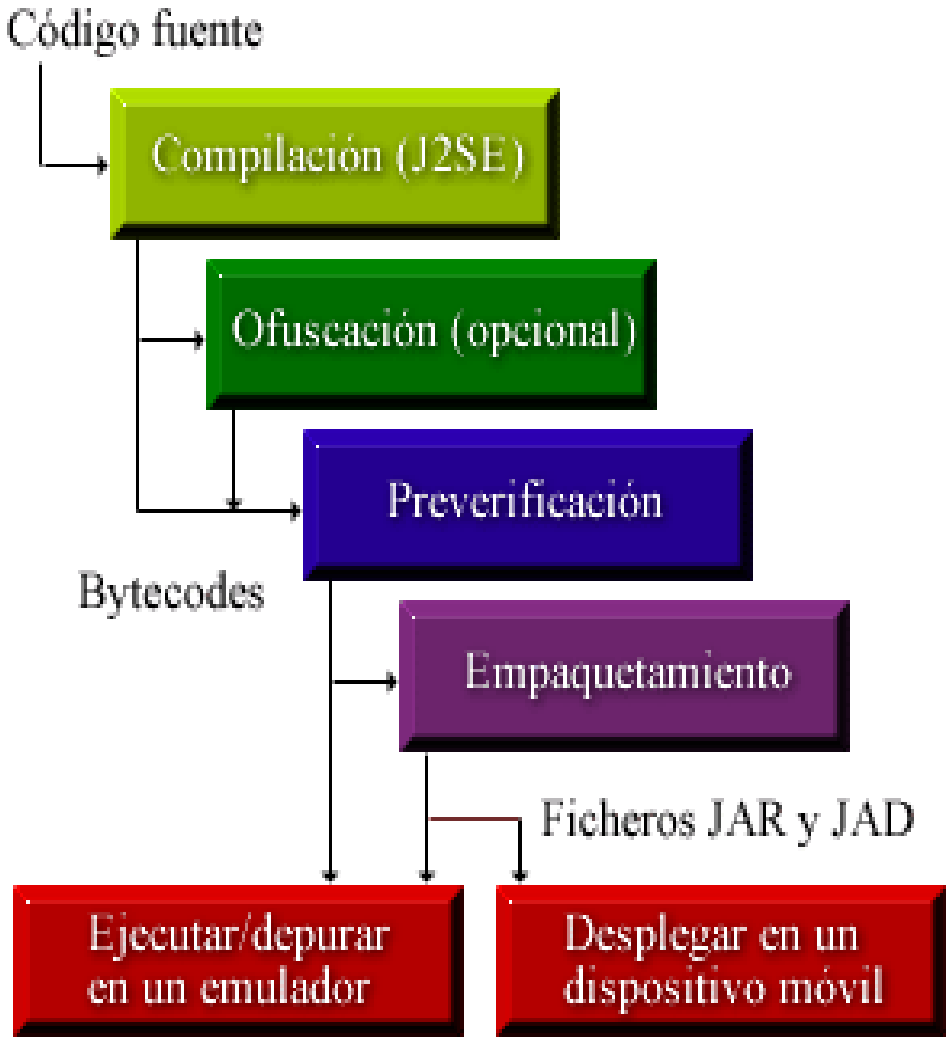
```
MIDlet-Name: SuiteEjemplos
MIDlet-Version: 1.0.0
MIDlet-Vendor: Universidad de Alicante
MIDlet-Description: Aplicaciones de ejemplo para moviles.
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
MIDlet-1: Snake, /icons/snake.png, es.ua.jtech.serpiente.SerpMIDlet
MIDlet-2: TeleSketch,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

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

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

Pasos del proceso



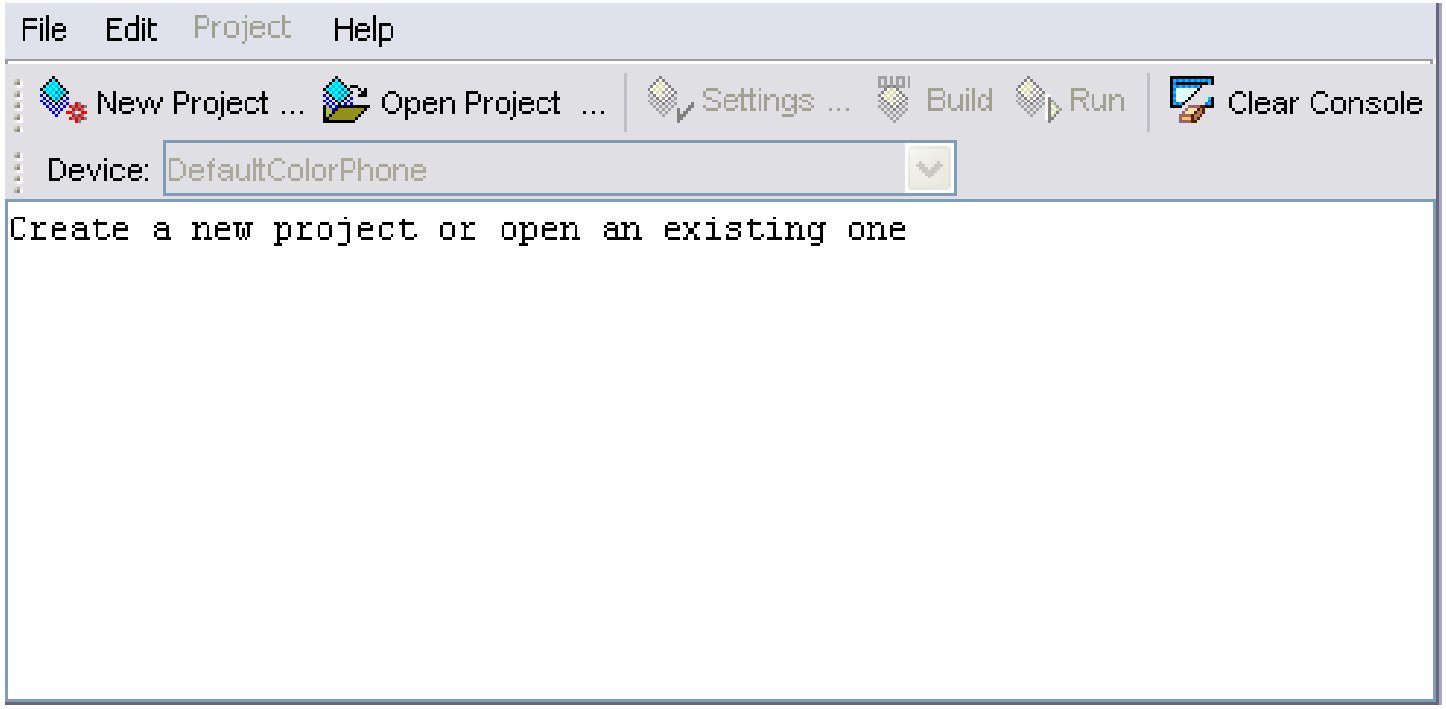
- Compilar
  - Utilizar como clases del núcleo la API de MIDP
- Ofuscar (optativo)
  - Reducir tamaño de los ficheros
  - Evitar descompilación
- Preverificar
  - Reorganizar el código para facilitar la verificación a la KVM
  - Comprobar que no se usan características no soportadas por KVM
- Empaquetar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

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)

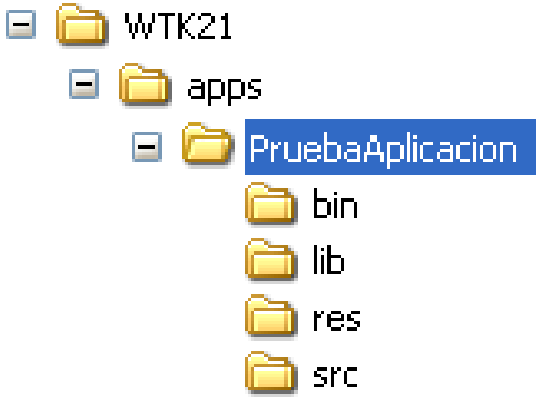
Ktoolbar

- La herramienta principal de WTK (llamada `ktoolbar` en versiones anteriores) nos permite automatizar la creación de aplicaciones



Aplicaciones de WTK

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

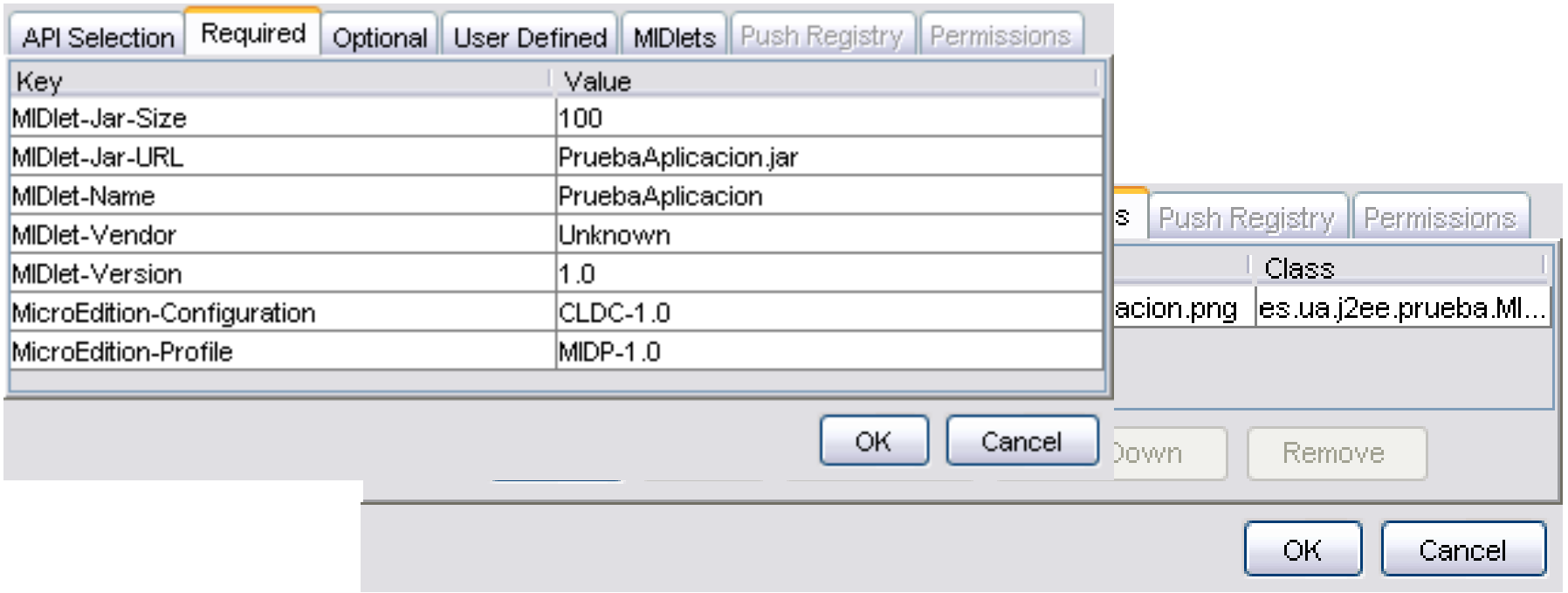


`src`: Código fuente
`res`: Recursos (ficheros de datos, imágenes, ...)
`lib`: Librerías (jar)
`bin`: Aquí se generan los ficheros JAD y JAR
`classes`: Clases intermedias generadas (temporal)

Crear una aplicación

- Pulsar *New Proyect ...*

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



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*

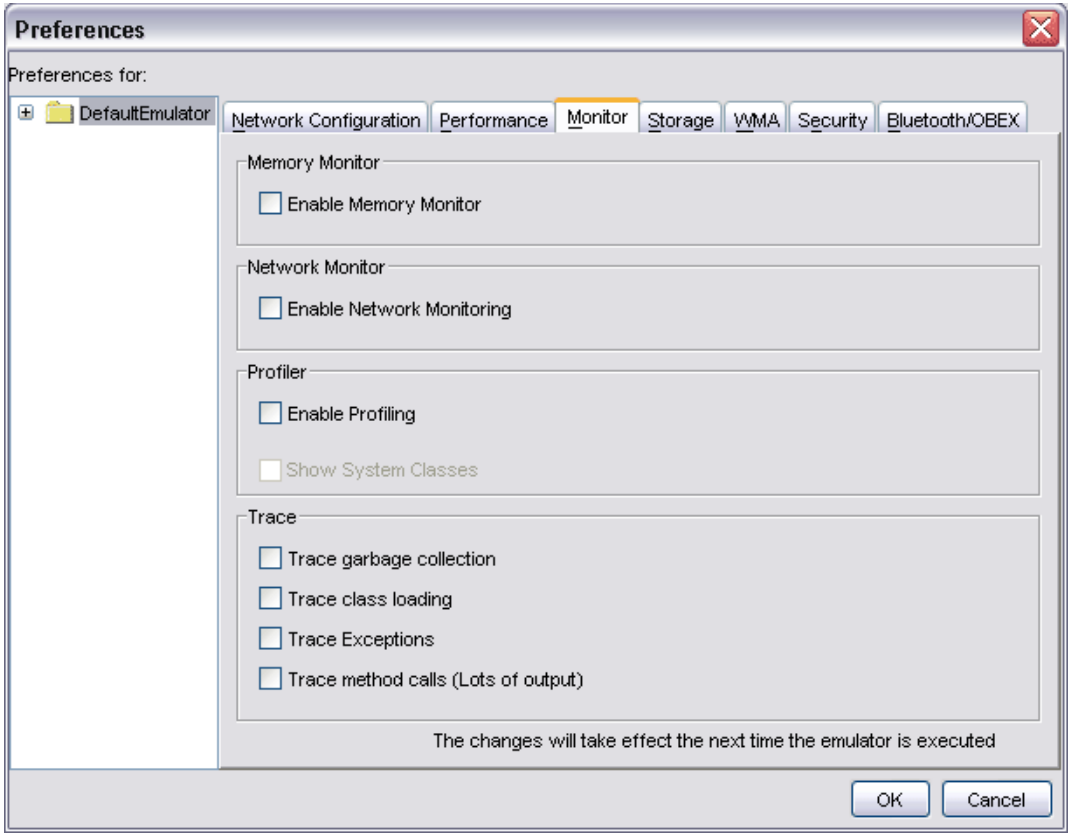
- Distribuir la aplicación
  - Pulsar sobre *Project* → *Package* → *Create package*

Provisionamiento OTA

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

Optimización

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

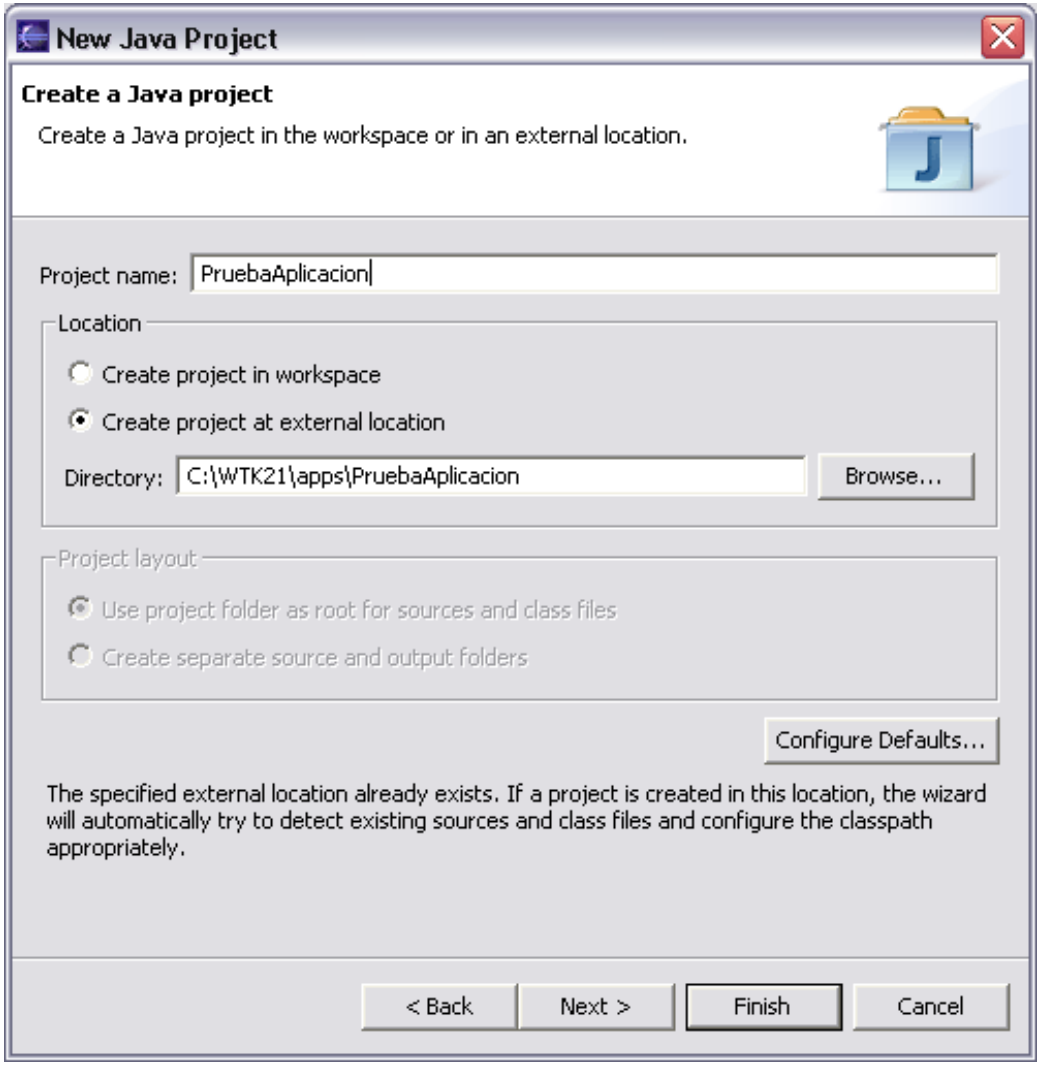


Integración de J2ME y Eclipse

- 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

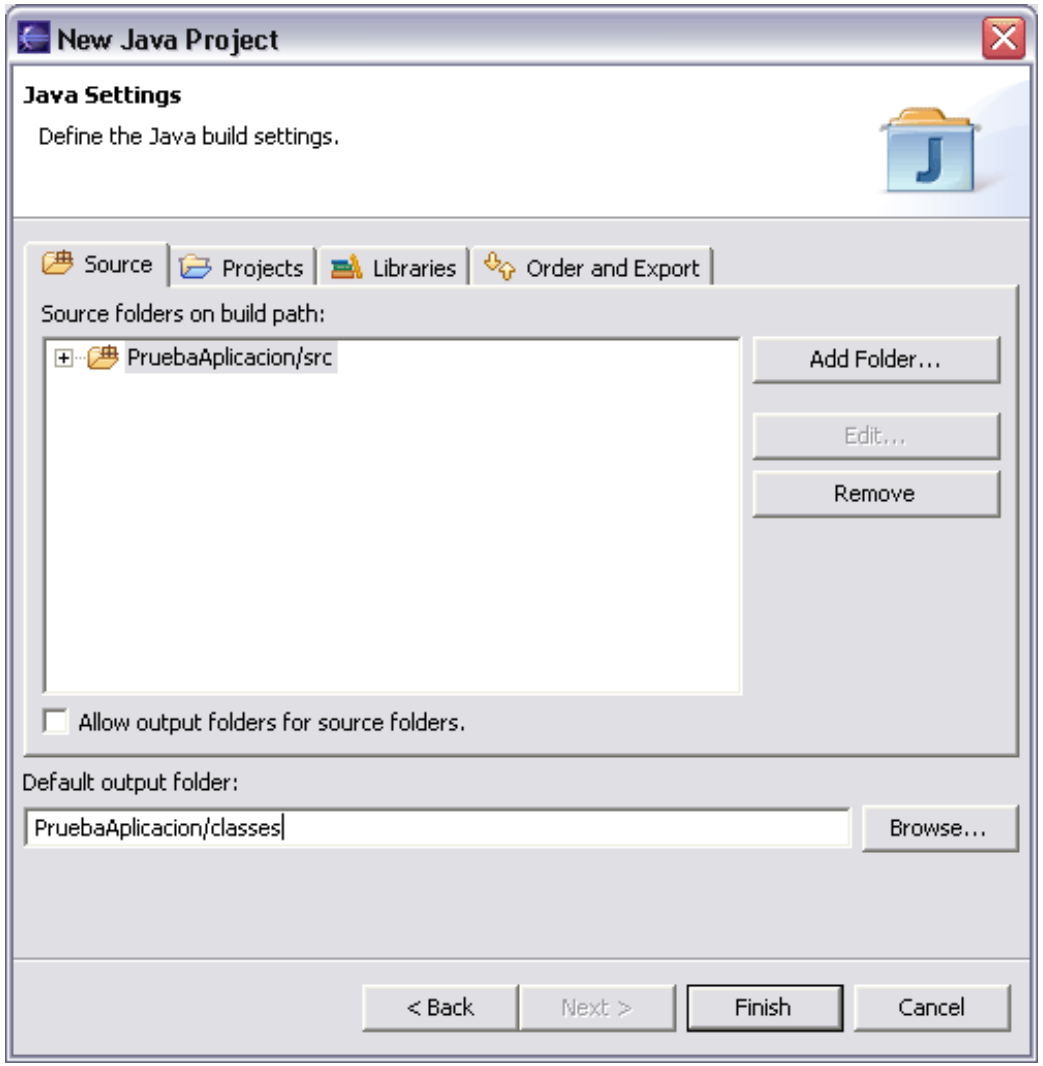
Creación de un proyecto

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



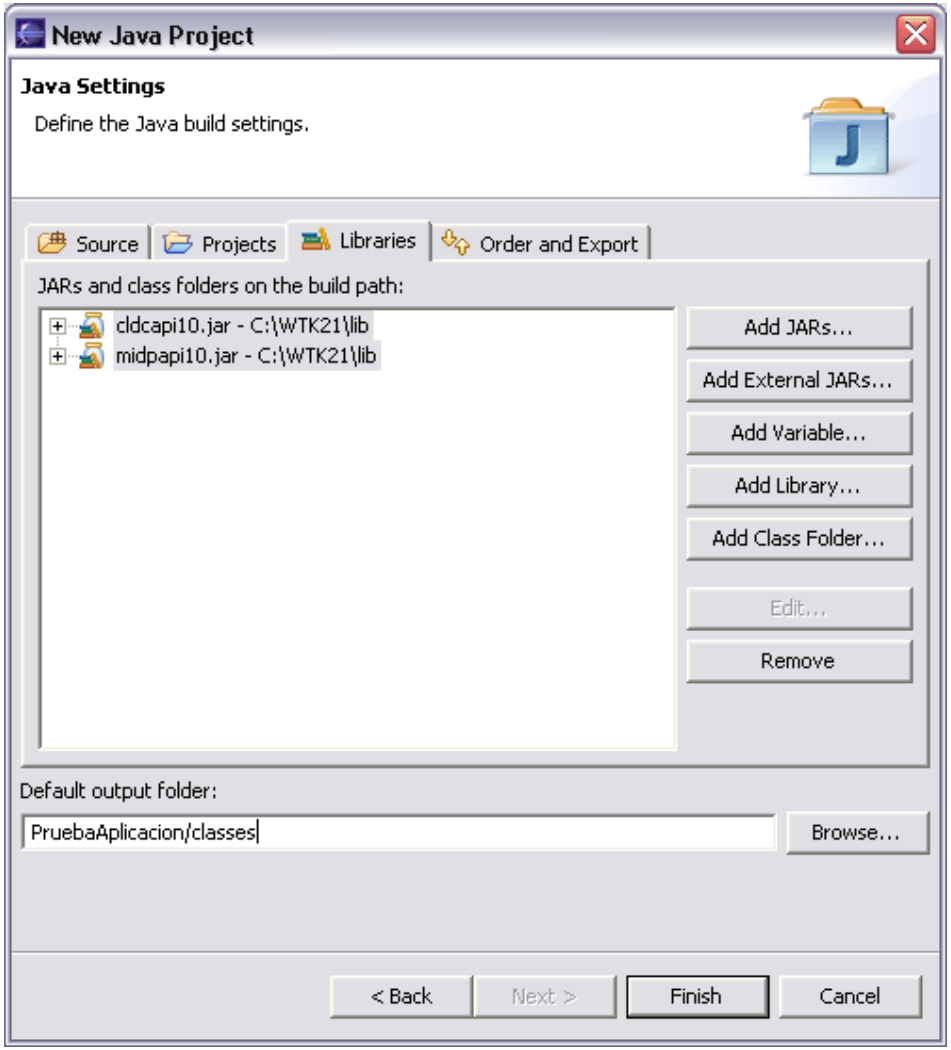
Establecer directorios

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



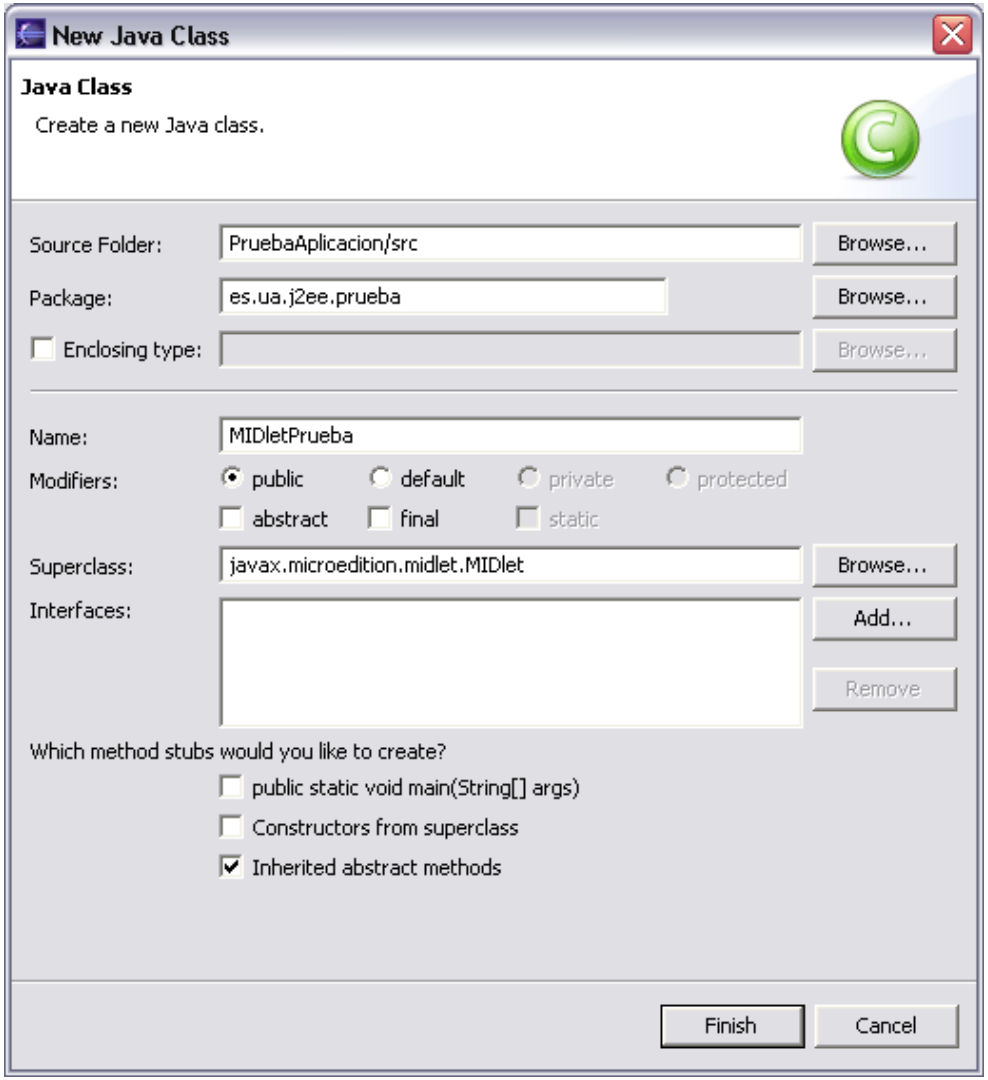
Establecer librerías

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



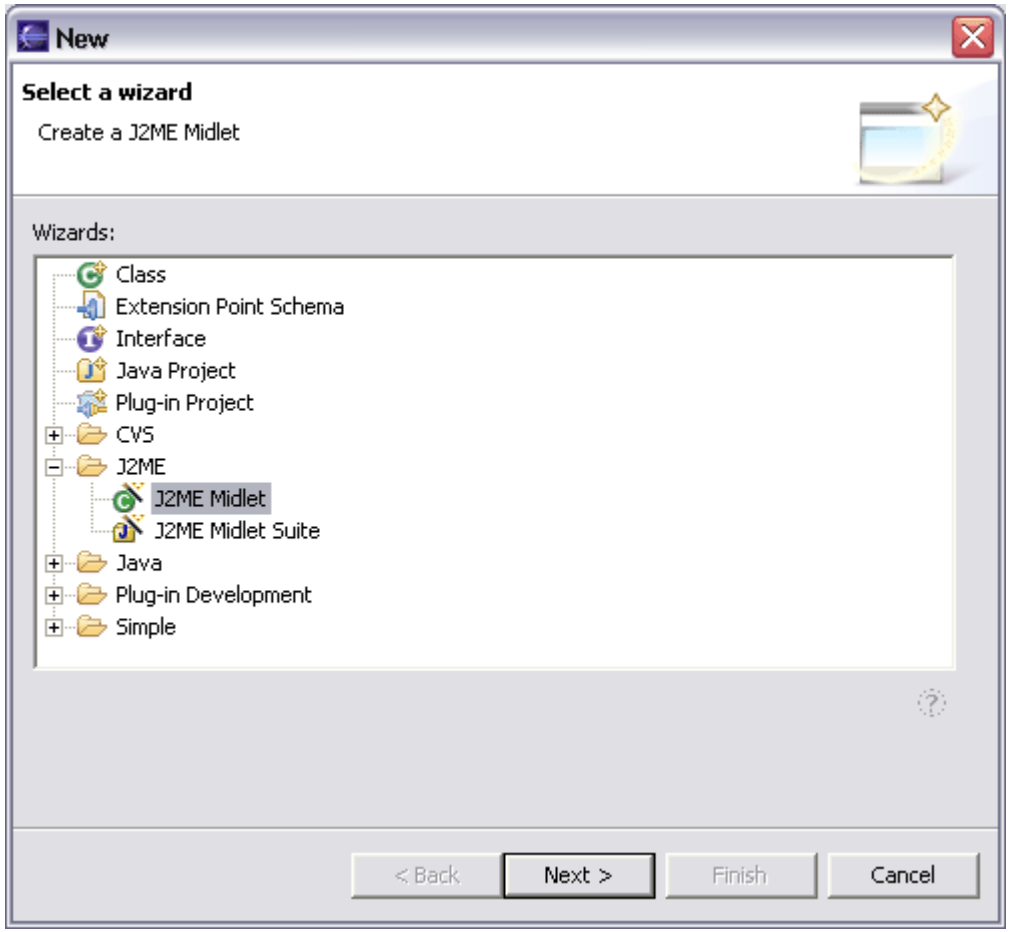
Crear un MIDlet

- 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



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



Configuración CLDC

- Características básicas del lenguaje
  - Mantiene la sintaxis y tipos de datos básicos del lenguaje Java
- 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

Características ausentes

- 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)

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`

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
  }
}
```

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

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
```

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);
```

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 descodificar los datos del objeto en el flujo

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();
  }
}
```

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;
  }
}
```

Recursos en el JAR

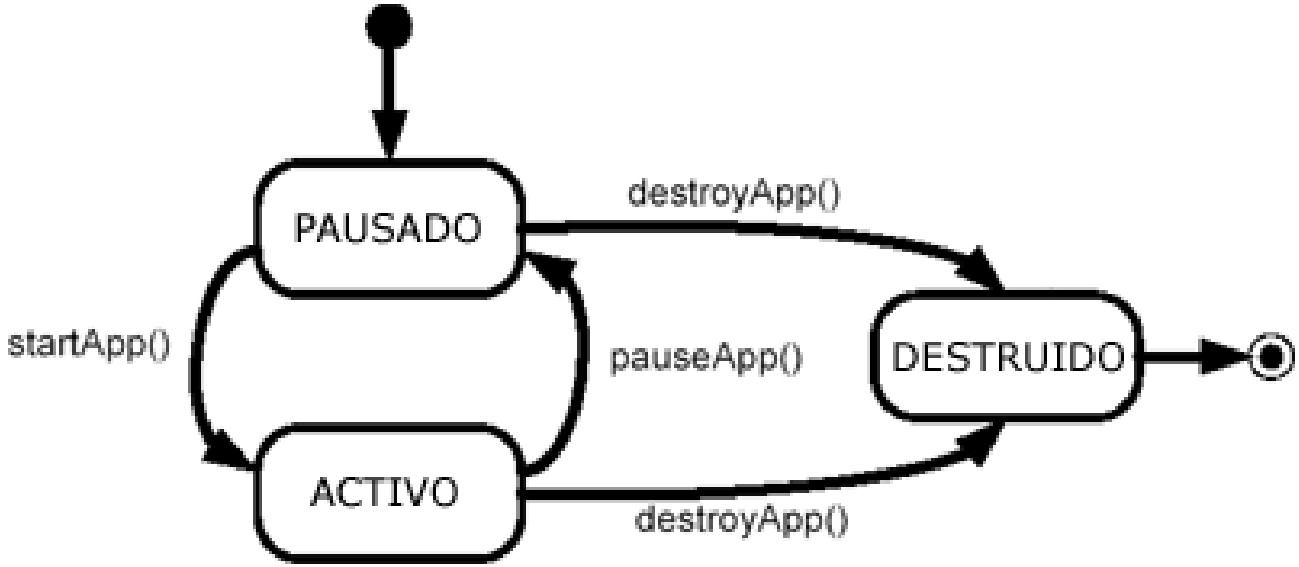
- Hemos visto que podemos añadir cualquier tipo de recursos al JAR de nuestra aplicación
  - Ficheros de datos, imágenes, sonidos, etc
- Estos recursos no se encuentran en el sistema de ficheros
  - Son recursos del JAR
- Para leerlos deberemos utilizar el método `getResourceAsStream` de cualquier objeto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

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



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
   }
}
```

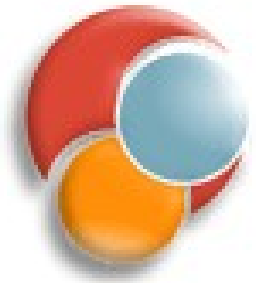
Propiedades

- Leer propiedades de configuración (JAD)

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

- Salir de la aplicación

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



¿Preguntas...?