# JAVA2 MICRO EDITION.

# UN PRIMER VISTAZO

**VERSIÓN 1.0**

A medida que evoluciona la tecnología aumenta la posibilidad de añadir funcionalidad a nuestros dispositivos inalámbricos y así satisfacer las necesidades de los usuarios. ¿Por qué conformarse con los paquetes que trae el móvil por defecto cuando lo puedes personalizar?, para esto Java ha desarrollado un paquete llamado J2ME con el objetivo de desarrollar aplicaciones para PDAs, móviles, agendas electrónicas y todo tipo de dispositivos que cumplan unas determinadas especificaciones.

En este documento se pretende iniciar en el manejo de este paquete así como familiarizarle con conceptos tales como MIDP, CLDC, MIDLET, RMS, Perfiles de usuario y muchos otros.

Tutores: Mª Celeste Campo Vázquez y Carlos García Rubio

Autores: Guillermo Diez-Andino Sancho (gdandino@it.uc3m.es )
Rosa Mª García Rioja (rgrioja@it.uc3m.es)

**http://www.it.uc3m.es/pervasive**

# INDICE

# OBJETIVOS

El objetivo de este documento es dar una visión general del lenguaje de programación Java2 Micro Edition, desarrollado por Sun para cubrir las necesidades de dispositivos con recursos muy limitados como teléfonos móviles y PDAs.

En primer lugar se hará una introducción a la tecnología Java en general, para a continuación hacer una breve reseña de tecnologías paralelas existentes en el mercado. Tras ver las principales diferencias con Java2 SE se procederá a considerar los aspectos tanto de instalación como de ejecución de una primera aplicación. Por último se describirán las características principales de los paquetes más destacados en J2ME con breves ejemplos que faciliten su entendimiento.

# INTRODUCCIÓN

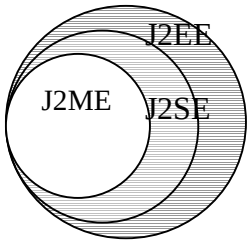
TECNOLOGÍA JAVA2

Si se ha trabajado con Java anteriormente, se conocerá la edición estándar de Java, denominada J2SE (Java 2 Standard Edition). **J2SE** es un conjunto de herramientas y APIs que permiten desarrollar applets y aplicaciones. En los orígenes de Java este kit cubría las necesidades del mercado ya que se podía desarrollar todo lo demandado, pero a medida que pasa el tiempo y los ordenadores se adaptan más a los tiempos, aparecen nuevas exigencias a cubrir.

Hoy en día los productos software tienden a ser distribuidos, a compartir información en distintas ubicaciones físicas, esto es lo que se denomina “enterprice applications”. Sun para cubrir las nuevas necesidades del mercado presenta **J2EE** (Java Enterprice Edition).

La razón por la que no se integró el paquete J2EE en el J2SE que ya era conocido es que las aplicaciones distribuidas suponen tratar tareas específicas de red, operaciones de E/S y manejo de Base de datos con un enfoque totalmente distinto al que requiere una aplicación convencional. Además estas aplicaciones requieren manejar una significante cantidad de almacenamiento, de memoria y tienen una complejidad adicional en el diseño de APIs.

Si J2EE surgió por los nuevos requisitos que presentaban las aplicaciones ni que decir tiene que se precisaba un paquete que diese soporte a las aplicaciones para móviles, PDAs, agendas electrónicas... Estos dispositivos no cuentan con mucha memoria, tienen limitaciones computacionales, los displays son pequeños...y es por esto que **J2ME** resume la funcionalidad de J2SE adaptándola a los requisitos mínimos necesarios que son aplicables a los dispositivos móviles, inalámbricos.



# J2ME Y OTRAS TECNOLOGÍAS INALÁMBRICAS

En esta apartado se comentan algunas de las tecnologías y servicios inalámbricos con el fin de esclarecer lo que no es J2ME, entre ellas se hablará de las más conocidas.

**WAP**
Acrónimo de Wireless Application Protocol, es una tecnología introducida en el mercado en 1995 que permite a los dispositivos inalámbricos recibir datos de Internet y mostrarlos en las pantalla de los mismos. Se piensa en Wap como una tecnología que da soporte a buscadores Web en móviles pero en realidad no es una aplicación sino un protocolo.

**SMS**
Short Messaging Service, es una tecnología que da soporte al envío y recepción de mensajes cortos de texto en dispositivos móviles. Otra característica interesante de SMS es que emplea una mensajería unificada, lo que te permite acceder a mensajes de voz, al correo electrónico y faxes desde un dispositivo móvil.

**I-MODE**
Introducido al mercado por la compañía Japonesa NTT DoCoMo, esta tecnología compite con WAP, ya que de igual modo ofrece un mecanismo de acceso a Internet a través de los dispositivos móviles. I-Mode dispone de un lenguaje de etiquetas similar a HTML denominado compact HTML (cHTML). La mayoría de sus usuarios se encuentran en Japón y otros países asiáticos.

# J2SE vs J2ME

Java2 Micro Edition ha sido creado para adaptarse a las características de los nuevos dispositivos inalámbricos tales como teléfonos móviles y PDAs. Consecuentemente existirán diferencias con la versión estándar de Java destinada a PCs. Algunas de las principales diferencias son las siguientes:

1.- Tipos de datos

J2ME soporta un subconjunto de los tipos de datos de J2SE, los tipos float y double no son soportados por dos razones: la mayoría de dispositivos CLDC no tienen unidad de coma flotante y en segundo lugar es una operación muy costosa.

2.- Preverificación (on-line y off-line)

Al contrario que en J2SE dónde se realiza la verificación del código en tiempo de ejecución, en J2ME parte de la verificación se realiza off-line, es decir, fuera del dispositivo. Esto tiene la finalidad de reducir la carga de la máquina, llevando a cabo el resto de la verificación on-line.

3.- Descriptor y manifiesto

Al empaquetar archivos en J2ME es necesaria la inclusión de unos archivos especiales que contienen información de las aplicaciones que incluyen, estos archivos de denominan manifiesto y descriptor

4.- Nueva librería gráfica

Mediante el paquete lcdui J2ME define un nuevo conjunto de clases para la creación de interfaces gráficas. Estas clases están adaptadas a dispositivos con memorias muy limitadas y pantallas de tamaño reducido.

5.- Desaparición del main

Al contrario que las aplicaciones de la edición estándar de Java, en J2ME no existe una función main como punto de entrada para la ejecución del programa. El equivalente a este main sería el método *start app* (será explicado posteriormente).

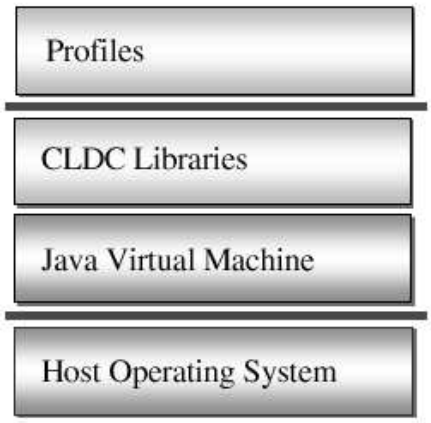
6.- Ausencia del recolector de basura

Con el objetivo de reducir la utilización de memoria, desaparece el recolector de basura en J2ME siendo necesario eliminar de forma explícita todos aquellos elementos que no vayan a ser utilizados más.

# ARQUITECTURA DE J2ME

En este apartado se describen brevemente los conceptos básicos relacionados con J2ME.

La arquitectura de alto nivel de J2ME se muestra en la siguiente figura:




**K VIRTUAL MACHINE**

La K Virtual Machine es una máquina virtual java altamente portable y compacta pensada y diseñada para funcionar en dispositivos con recursos limitados y con conexión a red. El objetivo de la tecnología KVM es el de crear la JVM más pequeña y completa que mantuviese los aspectos más importantes del lenguaje de programación Java y que funcionase en dispositivos con procesadores de 16 o 32 bits y unas

capacidades de unos pocos cientos de KiloBytes (originalmente 128 Kbytes de memoria).

### CONFIGURACIÓN

Este nivel es menos visible a los usuarios, pero es muy importante para los implementadores de perfiles. Define el conjunto mínimo de características de la KVM y las librerías de clases disponibles en una categoría de dispositivos. Representaría una franja horizontal de dispositivos con un mínimo denominador común que los desarrolladores van a poder asumir que tienen todos los dispositivos con esta configuración.

### PERFIL

Este nivel es el más visible a los usuarios y desarrolladores de aplicaciones. Las aplicaciones se desarrollan sobre un determinado perfil que a su vez está implementado sobre una determinada configuración.

El perfil define un conjunto de APIs y características comunes para una franja vertical de dispositivos. Las clases de un perfil permiten el acceso a funcionalidades específicas de los dispositivos como el interfaz gráfico, funcionalidades de red, almacenamiento persistente, etc ...

Las aplicaciones desarrolladas sobre un determinado perfil van a ser portables a cualquier dispositivo que soporte ese perfil.

Cabe destacar que un dispositivo puede soportar varios perfiles y que sobre una configuración pueden existir diversos perfiles.

Actualmente el perfil más utilizado es MIDP (Mobile Information Device Profile).

## CONCEPTOS PREVIOS

### MANIFIESTO

Cuando se empaquetan varias aplicaciones J2ME (MidLets) en un paquete (MidLet Suite) es necesario incluir un fichero de texto denominado manifiesto.

La finalidad del manifiesto es describir el contenido del fichero .JAR con información tal como el nombre, versión, vendedor, etc .. también se incluye en este fichero una entrada por cada MIDlet que lo compone.

Un ejemplo de manifiesto podría ser el siguiente:

```
MIDlet-1: Lista1, /icons/Lista1.png, Lista1
MIDlet-2: Texto1, /Icons/Texto1.png, Texto1
MIDlet-Name: Ejemplos
MIDlet-Vendor: Sun Microsystems
MIDlet-Version: 1.0
```

```
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

**DESCRIPTOR**

Aunque este fichero comparte el mismo formato que el del manifiesto, su finalidad es totalmente diferente. El objetivo de este fichero es proporcionar la información requerida por el Application Management Software (programa que gestiona las descargas de aplicaciones entre otras cosas) para cerciorarse de que el MIDlet suite es adecuado para el dispositivo en el que queremos instalarle. Su extensión es .JAD y al contrario que el manifiesto, éste no se incluye en el paquete.

**MIDLET**

Las aplicaciones que se desarrollan con J2ME e implementan la especificación MIDP para dispositivos móviles se denominan MIDLETs. Los MIDLETS se deben agrupar en un fichero .JAR para que sea posible su distribución (a otros dispositivos, a través de Internet, por ejemplo).

En un fichero .JAR se pueden incluir varios MIDLETs y a este conjunto se le denomina MIDLETSuite.

Por último se presenta la información que se incluye en un MIDLET.JAR:

- Clases MIDlet
- Clases soporte
- Recursos(imágenes,sonido,...)
- Fichero Manifiesto ( fichero .mf)
- Descriptor de aplicación (fichero .jad)

# INSTALACIÓN

WINDOWS

Antes de comenzar a implementar un MIDLET se precisa tener instalado un software determinado que constituirá el entorno desarrollo J2ME. Los componentes están disponibles en http://www.java.sun.com y son los que se presentan a continuación:

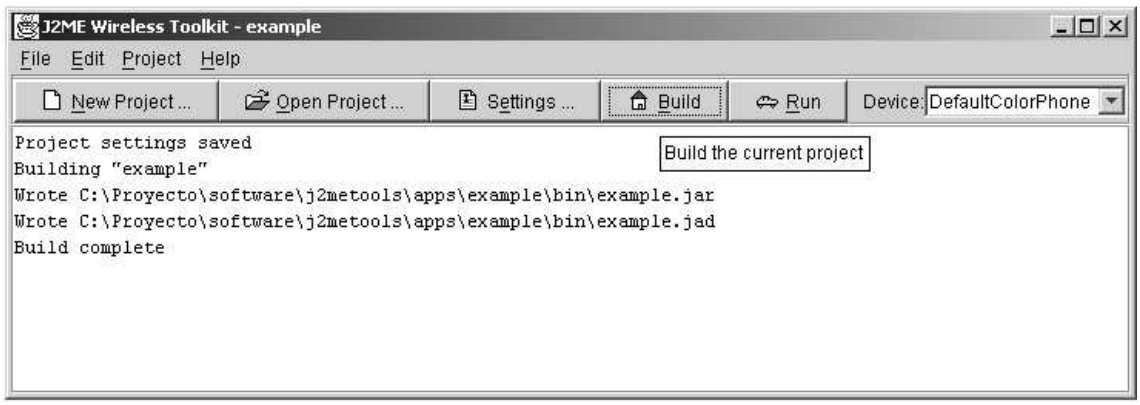
- Java 2 SDK
- Entorno de desarrollo visual (opcional)
- J2ME Wireless Kit

Java 2 SDK, es el Kit de desarrollo estándar de Java que se emplea para diseñar aplicaciones estándares. Este elemento debe ser el primero en instalarse ya que los siguientes se apoyan en él. Se requiere la versión 1.3 o superior que encuentra disponible en el enlace http://java.sun.com/j2se/1.3/download-windows.html.

J2MEWT se puede ejecutar individualmente o como un componente integrado en uno de los siguientes entornos gráficos de desarrollo:

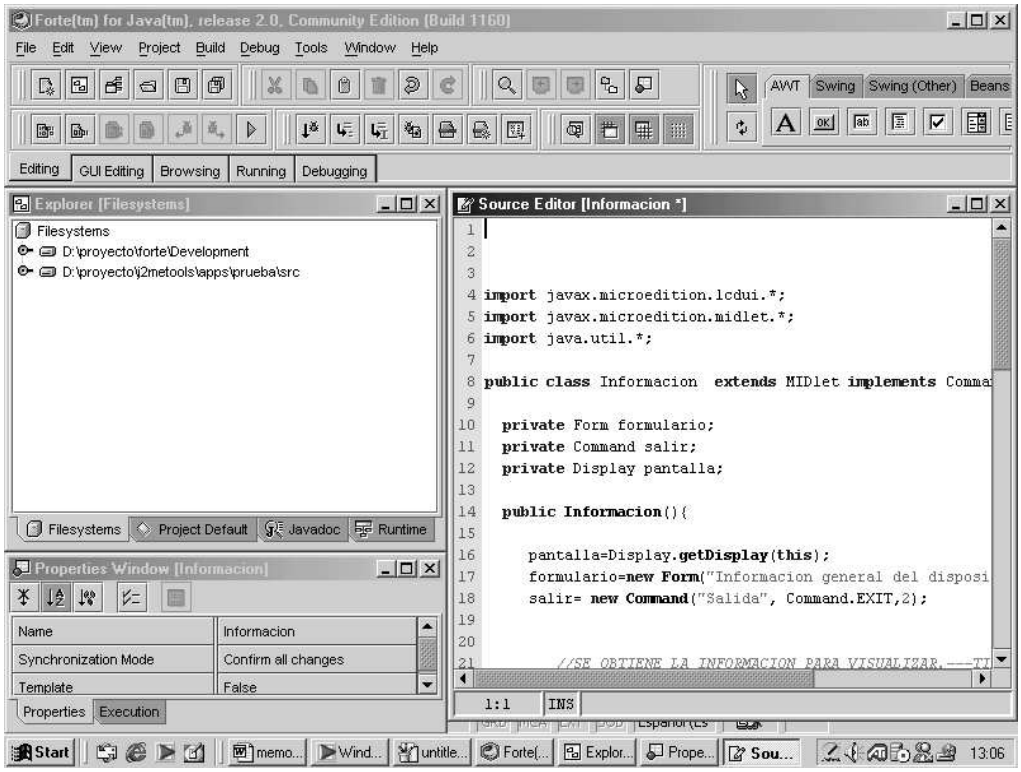
- KToolBar : Es el entorno visual de desarrollo que permite la creación de MIDLETS con J2ME. Se considera el más simple de todos los aquí mencionados ya que ni siquiera cuenta con un editor. Para incluir los fuentes editados con otro programa se debe haber creado un proyecto desde KToolBar, la herramienta crea la siguiente estructura de directorios dentro del proyecto.


En el directorio ***src*** se deben incluir los fuentes para que KToolBar los pueda compilar, pre-verificar y posteriormente emular.



- Forte : Java: Se debe instalar antes que las J2MEWT, éste se puede descargar de la página http://www.cun.com/forte/ffj/buy.html. Durante la instalación hay que seleccionar la versión de JDK que se ha instalado.

- Los requisitos hardware de Forte Java son un Pentium II a 300 MHz, 128 MB y 30 MB de espacio en disco. Este entorno provee un entorno de desarrollo para J2SE Y J2EE. El entorno incluye un editor de texto de GUI, un buscador de clases y un buscador de Web. El buscador de clases proporciona una vista gráfica de las clases que contiene el MIDLET que se está desarrollando. Utilizando este browser se puede navegar fácilmente por los métodos y atributos de las clases, así como analizar la clase compilada almacenada en el .JAR .




- Code Warrior for Java: es un entorno visual que da soporte a J2ME , similar a Forte Java, que cuenta con un editor de textos y un browser de clases. Este software se puede adquirir en http://www.codewarrior.com/ .

- Jbuilder Handheld Express: presenta las mismas opciones que Forte Java y Code Warrior, soporta el entorno de desarrollo J2ME pero sólo para Palms, aunque se espera que en breve se presente la versión que da soporte a móviles y agendas personales. La página en la que se puede adquirir el software y obtener más información es http://www.inprise.com/jbuilder/hhe/ .

J2ME Wireless Kit disponible para dos plataformas: Sun Solaris y Microsoft Windows , se puede descargar en http://java.sun.com/ products/j2mewtoolkit/ download.html . A continuación se va a comentar la instalación en Windows.

La versión Windows de la herramienta requiere un mínimo de 64MB y un espacio en disco de 15MB.

Se deben haber llevado a cabo los pasos anteriores, ya que si en la instalación se detecta que no existe una versión de JDK no se permite continuar con la instalación. El entorno de desarrollo es opcional. Si se desea trabajar con un entorno de desarrollo no se debe seleccionar la opción de “Integrated”, sino la de “Stand Alone”.

Una vez completada la instalación se debe asegurar que todo está correctamente instalado. Desde el promt de DOS vaya al directorio J2MEWTK_DIR\apps\ example\bin\, luego verá que existe un fichero run.bat, ejecútelo y si la instalación tuvo éxito verá la pantalla de inicio de la aplicación ejemplo que incluye el paquete.


Ahora ya está todo listo para comenzar.

# UN PRIMER EJEMPLO

Una vez superada la fase de instalación llegamos a la fase de creación de un primer programa. Para ello basta disponer de un editor de texto cualquiera en el que se irá escribiendo el código fuente que compilaremos bien desde el interfaz gráfico que el Wireless Toolkit ofrece o desde la propia línea de comandos.

A continuación se detallan los pasos para hacer funcionar nuestro "primer programa".

1.- **Escribir el código** fuente del programa, en este caso va a ser el HolaMundo.java.

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

public class Howdy extends MIDlet implements CommandListener{

  private Command exitCommand;
  private Display display;
  private Form screen;

  public Howdy() {

   display=Display.getDisplay(this);
   exitCommand = new Command("Exit",Command.EXIT,2);
   screen = new Form("Hola");
   StringItem strItem = new StringItem(" ","Hola mundo");
   screen.append(strItem);
   screen.addCommand(exitCommand);
   screen.setCommandListener(this);
 }
 public void startApp() throws MIDletStateChangeException{

  display.setCurrent(screen);
 }

 public void pauseApp(){
 }
 public void destroyApp(boolean unconditional){
 }
 public void commandAction (Command c, Displayable s){

   if(c==exitCommand){
        destroyApp(false);
     notifyDestroyed();
   }
  }
}
```

Esta clase hereda de la clase MIDLET. En el constructor se capturan las características del display, se crea una instancia de la clase Form , StringItem y del Comando que me permite salir. Luego se añade la cadena y el comando de salida al formulario y se incluye el capturador de eventos del comando, éste tiene una acción asociada que se ejecutará al pulsar sobre esta opción.

2.- **Compilar** desde la línea de comandos:

```
C:\c:\jdk1.3\bin\javac -g:none
                      -bootclasspath %SUNJ2MEHOME%\lib\midpapi.zip
                          -classpath %SUNJ2MEHOME%\lib\kvem.jar
                          HolaMundo.java
```

Al utilizar el compilador de J2SE para compilar el ejemplo es necesario indicar de alguna manera que las librerías CLDC y MIDP se encuentran en un lugar diferente. Esto se hace mediante la opción –bootclasspath.
La opción –g:none se ha empleado para desactivar la posibilidad de incluir información en la depuración, haciendo de este modo los ficheros .class más pequeños.

3.- **Preverificar**

Cuando se trabaja con J2SE la máquina virtual Java (JVM) lleva a cabo un proceso de verificación en tiempo de ejecución. Esto ocupa ciertos recursos que en dispositivos inalámbricos son relativamente escasos. Debido a las restricciones de estos dispositivos se ha creado en J2ME un proceso denominado *preverification* consistente en llevar a cabo parte de la verificación de la KVM off-line, el resto será llevado a cabo como una verificación normal.

El comando utilizado es el siguiente:

```
C:\>%SUNJ2MEHOME%\bin\preverify.exe -classpath %SUNJ2MEHOME%\lib\midpapi.zip
                                    -d c:\Ejemplos c:\Ejemplos
```

4.- **Crear el fichero** .jar

El siguiente paso consiste en empaquetar el Midlet en un MidletSuite, que es un simple fichero .jar. Esto es un requerimiento por parte de muchos dispositivos MIDP. Para realizar este proceso JDK1.3 dispone de la utilidad jar para archivar y comprimir ficheros .class en un paquete.

```
C:\>c:\JDK1.3\bin\jar cmf MANIFIESTO.MF HolaMundo.jar HolaMundo.class
```

5.- **Ejecutar** el Midlet

Una vez dispuesto el Midlet dentro de un paquete llega la fase de ejecución. Para simplificar la línea de comandos establecemos en primer lugar las variables de entorno necesarias.

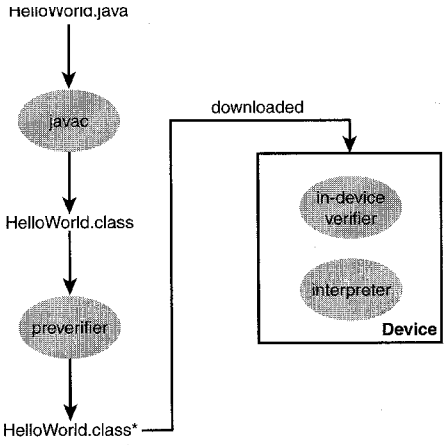
C:\>set CODEPATH=c:\Ejemplos
*C:\>set CLASSPATH=%SUNJ2MEHOME%\lib\kvem.jar;%SUNJ2MEHOME%\lib\kenv.zip;*
*%SUNJ2MEHOME%\lib\lime.jar;%CODEPATH%*

A continuación se lanza el emulador para ejecutar el programa:

***C:**\c:\jdk1.3\bin\java.exe -Dkvem.home=%SUNJ2MEHOME%*
*-classpath %CLASSPATH% com.sun.kvem.midp.Main DefaultGrayPhone*
*-descriptor HolaMundo.jad*

El resultado obtenido es el siguiente:

Finalmente se muestra un esquema de las fases anteriores



# DESCARGA DE UN MIDLET

Una vez creado un MidLet llega la fase de distribución. Existen diversas maneras como puede ser a través de un cable serie conectado al PC, pero la técnica más empleada es a través de Internet. Esto conlleva una serie de pasos adicionales que se describen a continuación.

1.- Seleccionar la aplicación

Los dispositivos MIDP proporcionan mecanismos que permiten a los usuarios descubrir o ser notificados de los MIDlets suites que pueden ser descargados en sus dispositivos. Estos MIDlets son identificados mediante su descriptor.

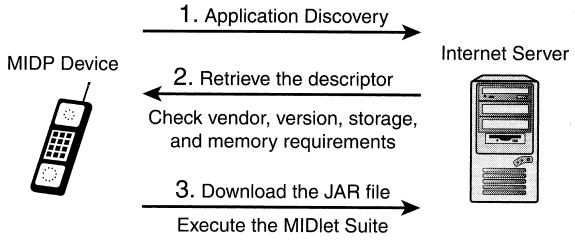
2.- Descargar y verificar el descriptor

Una vez seleccionado el MidLet se procede a la descarga del descriptor en el dispositivo. El AMS (Application Management Software) examinará el nombre, versión, tamaño, versión de CLDC/MIDP para verificar que el entorno de ejecución será el correcto.

3.- Descargar y ejecutar el MIDlet suite

Una vez verificado el descriptor el AMS comenzará a descargar el archivo JAR de la dirección especificada en el atributo MIDlet-Jar-URL. Una vez descargada la aplicación se contrastará el descriptor con la información del manifiesto, para que en caso de que los atributos correspondan el MIDlet suite se cargue y esté listo para su ejecución, en caso contrario será rechazado.

En la siguiente ilustración se muestra el proceso completo de descarga.



# PAQUETES Y CLASES PRINCIPALES

A lo largo de este apartado se comentan los paquetes lcdui y midlet así como sus clases más importantes, siendo el resto explicados en apartados independientes.

Los principales paquetes de J2ME son los siguientes:

- javax.microediton.lcdui
- javax.microediton.midlet
- javax.microediton.rms
- javax.microediton.io

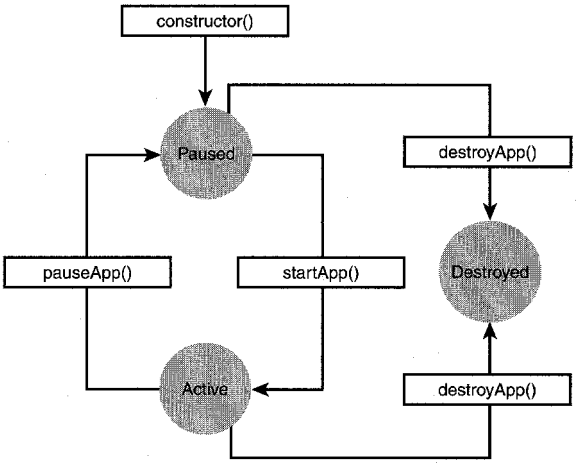
Se incluye una breve descripción de ciertos métodos en los casos que se considere relevante, ya que no hay que perder de vista que el objetivo de este documento es iniciar, no ilustrar de forma detallada, con lo que si en algún momento desea información más concreta remítase a la ayuda online del perfil MIDP.

### JAVAX.MICROEDITON.MIDLET

Un MIDlet es una aplicación de MIDP. La aplicación debe heredar de esta clase para que le permita gestionar el control del software.

Un MIDLET tiene 3 estados que son activo, pasivo y destruido, éstos se corresponden con los siguientes métodos:

- **startApp( )**- La aplicación inicialmente se encuentra en el estado "pausa".A este método se le puede llamar varias veces si entre medias se llama al método pauseApp()
- **destroyApp( )**- Mediante este método, la aplicación libera recursos.
- **pauseApp( )** - Este método permite intercambiar el MIDLET que se está ejecutando, pasándolo de activo a pasivo.



El administrador del entorno de ejecución de la aplicaciones es el que llama directamente a estos métodos, ya que un MIDLET no cuenta con el método "main".

El método de esta clase permite al gestor software de la aplicación crear, iniciar, pasar al estado de pausa y destruir el MIDlet. Un MIDlet es el conjunto de clases diseñadas para ser ejecutadas y controladas por la aplicación vía la interfaz. Se pueden tener varios MIDlets a la vez, y seleccionar cual está activo en un tiempo concreto llamando al metodo startApp( ) y ponerlo en pausa con la llamada a otro método, el pauseApp().

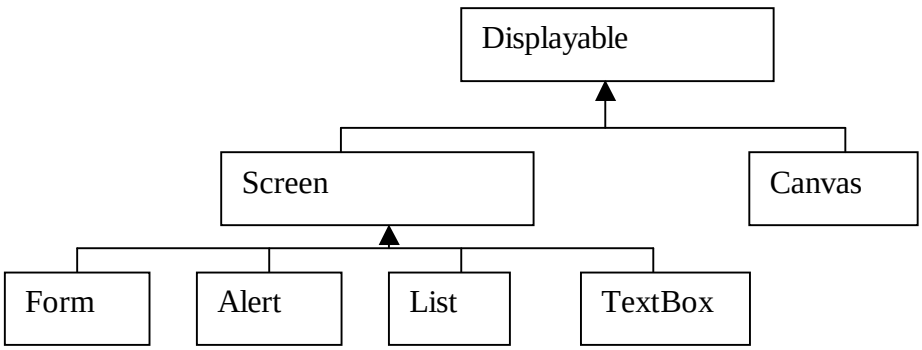
```
public void startApp() throws MIDletStateChangeException{
   display.setCurrent(list);
 }

 public void pauseApp(){

 }

 public void destroyApp(boolean unconditional){
  list=null;
  exitCommand=null;
  display=null;
 }
```

En este ejemplo se presentan únicamente los métodos anteriormente comentados. En el método startApp( ) se establece la pantalla de esa clase. Hay que capturar la excepción dentro del mismo o lanzarla. En destroyApp (boolean unconditional), se ponen a nulo los elementos instanciados en la clase, para liberar memoria ya que J2ME carece del recolector de basura.

## JAVAX.MICROEDTION.LCDUI

En el paquete **javax.microedtion.lcdui** se encuentran las clases que nos permiten desarrollar una interfaz de usuario.

La clase pública **Display** representa el gestor de las entradas y salidas del dispositivo del sistema. Por cada MIDlet se crea una instancia de Display y la aplicación hace referencia a esta instancia con la llamada al método **getDisplay().** La aplicación llama a este método desde **startApp()**. La clase Displayable es una superclase de todas las pantallas que se pueden poner en el Display.



La clase Displayable tiene dos descendientes directos , **Screen** y **Canvas**.

La principal abstracción de la interfaz de usuario de MIDP es la pantalla, instancia de la clase **Screen**. Una pantalla es un objeto que encapsula las especificaciones gráficas del dispositivo. Este objeto captura todos los eventos que tienen lugar cuando el usuario está navegando por su pantalla.

La aplicación puede cambiar de pantalla invocando al método *Display.setCurrent (Displayable).*

Esta clase presenta la posibilidad de incluir un título para la pantalla y un Ticker, para ello cuenta con los métodos *getTitle(), setTitle(String), getTicker(), setTicker(Ticker).*

Para incluir elementos (deben ser del tipo Item) dentro de una pantalla hace falta un contenedor, que viene representado por la clase **Form**.

Los métodos principales de esta clase, que son los que permiten trabajar con Items, realizan funciones tales como la inserción, borrado y obtención de un Item.
Se presentan a continuación:

- int append(Item elemento)
- void insert(int index,Item elemento)
- get(Item)
- set(int index, Item elemento)
- delete(int index)

Otro elemento que se puede incluir en un objeto Screen es un mensaje informativo para el usuario, teniendo la posibilidad de definir la duración del mismo en pantalla, esto se consigue instanciando la clase **Alert**.

**Alert(String titulo,String texto, Image imagen, AlertType a)**

El primer parámetro que se incluye en el constructor de esta clase es el título que se asocia al mensaje a visualizar. En el parámetro texto se incluye el contenido a mostrar por pantalla, el mensaje propiamente dicho. El parámetro imagen es la imagen que se va a incluir en el mensaje, aunque se puede obviar poniéndolo a nulo. Por último el parámetro AlertType indica el tipo al que corresponde este mensaje, éstos pueden ser:

- Alarm
- Confirmation
- Error
- Info
- Warning

El método más destacable de esta clase es el que permite determinar el tiempo que permanecerá en pantalla el mensaje que es **setTimeout(int time).**

A continuación se muestra como crear un objeto Alert, su asignación de tiempo y la visualización en pantalla.

```
Alert conversion= new Alert("Conversion €","Euros:"+ String.valueOf(euros) ,null,
                                    AlertType.INFO);
 conversion.setTimeout(Alert.FOREVER);
 pantalla.setCurrent(conversion);
```

También puede ser deseable incluir una lista con opciones que se puedan seleccionar dentro de la pantalla, esto se consigue utilizando la clase List. Hay tres tipos básicos de lista: exclusiva, implícita y múltiple.

**List(String titulo,int tipolista,String[] Elementos, Image[] imágenes)**

Los elementos que se van a incluir en la lista así como las imágenes que se relacionan con los mismos se pasan al método como un array.

A continuación se presenta una clase que crea una lista y la inicializa con los valores "Opcion A","Opcion B","Opcion C".

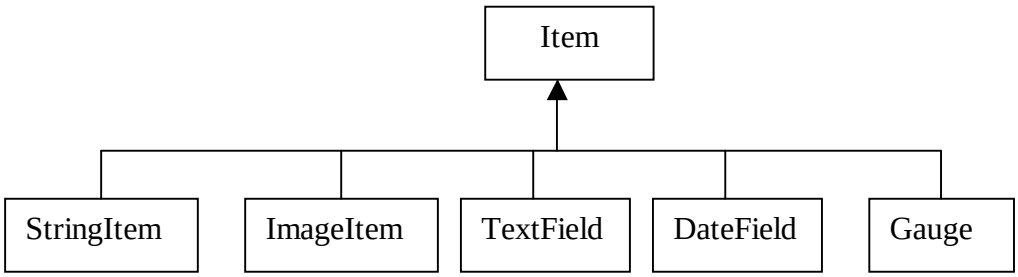
```
public class Lista1 extends MIDlet implements CommandListener{
 private List list;
 private Command exitCommand = new Command("Salir",Command.EXIT,1);

 private String[] options={"Opcion A","Opcion B","Opcion C"};
 private Display display;

 public Lista1(){
  list=new List("Elija una opción",List.IMPLICIT,options,null);
  list.addCommand(exitCommand);
  list.setCommandListener(this);

  display=Display.getDisplay(this);
 }
```

La clase TextBox implementa un componente de texto que rellena toda la pantalla. El constructor nos permite incluir el título, un texto inicial, el tamaño máximo de caracteres que se pueden incluir en ese TextBox, así como las restricciones del tipo de datos que se pueden incluir.

Las restricciones son las siguientes:

- ANY - no impone restricciones en el tipo de texto a incluir
- EMAILATOR - sólo se permite incluir direcciones de e-mail
- NUMERIC – los datos han de ser de tipo numérico
- PASSWORD – este tipo enmascara los caracteres que se introducen
- PHONENUMBER – permite incluir exclusivamente números de teléfono
- URL – permite la inserción de URLs

En el API de MIDP también están incluidos los componentes que derivan de la clase Item y ChoiceGroup.




La clase **StringItem** representa un elemento que contiene una cadena de texto, y cuenta con dos de métodos para coger y devolver el valor de la cadena.

Para representar imágenes se cuenta con la clase **ImageItem**.

A continuación se presenta un ejemplo en el que se trabaja con la clase StringItem e ImageItem. Primero se crean y se inicializan la pantalla, el formulario, los comandos y se crea un elemento StringItem a cuyo constructor se le pasa la cadena que contiene, en este caso "Bienvenido". A continuación se crea una imagen indicando dónde se encuentra y posteriormente se le pasa al constructor de ImageItem, junto con la ubicación que se quiere que ocupe en la pantalla. Ya para finalizar se añade el objeto frase y dibujito, ambas instancias de las clases StringItem e ImageItem, al formulario mediante la invocación al método append(Item).

```
//constructor que inicializa los valores de los atributos

public EjemploItem(){

   Image dibu=null;
   pantalla=Display.getDisplay(this);
   salir= new Command ("Salida",Command.EXIT,2);
   formulario= new Form("Programa 1");
   StringItem frase = new StringItem(" ","Bienvenido ");

   try{
     dibu= Image.createImage("/icons/Duke.png");
   }catch (IOException e){ System.out.println("error"+e);
   }
   ImageItem dibujito=new ImageItem("",dibu,ImageItem.LAYOUT_DEFAULT,"");
   formulario.append(frase);
   formulario.append(dibujito);
   formulario.addCommand(salir);
   formulario.setCommandListener(this);
  }
```

**TextField** presenta una ventana de texto que se incluye en un formulario. A diferencia de la clase TextBox, se puede añadir a un formulario junto con otros elementos y no ocupa toda la pantalla.

Este ejemplo se va a comentar a la vez que se muestra el código. Dentro de la clase que se ha creado se define un elemento de tipo Form y otro de tipo TextField y se inicializan en el constructor.

```
private Form formulario;
private TextField pesetas;
```

En el caso del TextField, se indica el titulo que queremos que aparezca en el TextField, el valor inicial que en este caso es nulo, la restricciones de los tipos de datos que se admiten en este elemento, son las mismas que las de TextBox y en este caso se indica que sólo se admiten datos de tipo numérico.

```
public Euros(){

  pantalla=Display.getDisplay(this);
  formulario=new Form("Conversor de euros");
  pesetas = new TextField("Pts a convertir","",5,TextField.NUMERIC);
```

Ahora ya solo queda añadírselo al formulario y para ello lo hacemos como se muestra a continuación

```
  formulario.append(pesetas);
}
```

**DateField** es un componente editable que presenta información de tipo fecha y hora en un formulario. En el constructor se añade la etiqueta que se le asigna y el modo que viene definido por unas constantes que son:

- DATE – Tipos de datos de fecha
- TIME – para horas
- DATE_TIME – para horas y fechas

La Clase **Gauge** implementa un gráfico de barras para visualizar valores representados en rangos de porcentajes.

La clase **ChoiceGroup** es como la de JDK, representa una lista de elementos que el usuario puede seleccionar y que cada uno de ellos tiene una acción asociada.

Una vez comentada la clase Screen ,se dará una breve descripción de la clase **Canvas** y **Graphics** ya que no presentan ninguna diferencia sobre las de JDK.

La clases **Canvas** se emplea para las aplicaciones en las que se necesita capturar eventos y realizar llamadas gráficas para dibujar en el display. Esta clase es

intercambiable con la clase Screen, por ejemplo en un formulario se puede incluir un elemento List y que la opción seleccionada realice un dibujo, utilizando para ello la clase Canvas.

Para dibujar cualquier elemento es imprescindible invocar al método *paint(Graphics dibujo)* de la clase Canvas. El objeto Graphics define una región que se considera inválida, con la llamada al método paint() se consigue que se pinten todos los pixels de esa región.

La clase Graphics provee la capacidad de renderizar figuras geométricas 2D, permite dibujar textos , imágenes, líneas, rectángulos y arcos, para ello cuenta con los métodos:

- drawString()
- drawImage (Image img, int x, int y, int anchor)
- drawLine(int x1, int y1, int x2, int y2)
- drawRect(int x, int y, int width, int height)
- drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)
- drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)[1]

Los rectángulos y los arcos se pueden rellenar con la llamada al método fillRect(int x, int y, int width, int height) y fillArc(int x, int y, int width, int height, int startAngle, int arcAngle).

[1]A diferencia de Java Estándar no existe el método drawOval( ), con lo que para dibujar un óvalo se emplea el método drawArc( ) y se especifica que el ángulo es de 360 º

# ALMACENAMIENTO DE DATOS

El almacenamiento y recuperación de datos de forma persistente es una característica propia de prácticamente todos los lenguajes de programación.
J2ME MIDP define una simple base de datos de registros denominada record management system (RMS) con el objetivo de poder almacenar información una vez que el MIDlet finalice.

La unidad de almacenamiento básica dentro de RMS es el *record* que es almacenado en una base de datos especial denominada *record store*. Un record (registro) como su nombre indica es un simple array de bytes, mientras que un record store es un fichero binario que contiene una colección de registros.

El API empleado para la creación y manejo de estos registros es descrito en el paquete javax.microedition.rms; este paquete define dos clases, tres interfaces[2] y cinco excepciones.

RecordStore, RecordEnumeration
RecordComparator, RecordFilter, RecordListener
InvalidRecordIDException, RecordStoreException, RecordStoreFullException, RecordStoreNotFoundException, RecordStoreNotOpenException.

A continuación se describen las clases disponibles en el API así como sus funciones de manejo básicas.

RecordStore

Algunas de las normas básicas a la hora de crear un RecordStore son las siguientes:

- Los nombres no deben superar los 32 caracteres, cabe mencionar que se distingue entre mayúsculas y minúsculas.
- Dentro de un MIDlet suite no deben existir dos record store con el mismo nombre
- Los RecordStores creados dentro de un MIDlet suite no son accesibles a otros.

Las funciones básicas de manejo de record store se muestran a continuación:

. **openRecordStore**: abre un record store existente, en caso contrario crea uno nuevo.

Ej. *RecordStore fichero = null;*
*fichero = RecordStore.openRecordStore("fichero_prueba",true);*

. **closeRecordStore**: cierra un record store

Ej. *fichero.closeRecordStore();*

[2] En la especificación MIDP, RecordEnumeration está definida como un interfaz, aunque no es necesario implementarlo ya que lo hace MIDP. A efectos del programador se trata como una clase más.

. **deleteRecordStore**: borra un record store

. **getName**: obtiene el nombre de un record store

. **getNumRecords**: obtiene el número de registros dentro de un record store

Existen una serie de métodos adicionales para la obtención de la información de cabecera sobre el record store, las más empleadas son: getLastModified, getNextRecordID, getNumRecords, getVersion. Es posible asimismo obtener otros tipos de información sobre el record store, esta información no está disponible en la cabecera, pero se puede acceder a ella mediante implementación, para ello se dispone de los métodos: getSizeAvailable, getSize, listRecordStores. Este último método merece mención especial ya que ofrece la posibilidad de obtener un array con los nombres de todos los record stores que pertenecen a un MIDlet suite.

Con respecto al manejo de registros se dispone de los siguientes métodos:

. **addRecord**: añade un nuevo registro, éste debe encontrarse como un array de bytes, el valor devuelto por esta función es un entero denominado record ID

. **getRecord**: se obtienen los datos almacenados en un determinado registro

. **setRecord**: este método se emplea para establecer los datos de un registro.

. **deleteRecord**: eliminar un registro de un record store

Cada vez que se crea un nuevo registro en un fichero se le asigna un identificador único dentro del mismo, record ID, éste será utilizado para localizarle dentro del fichero de manera única, una vez eliminado un registro, este permanece a NULL, su identificador con.

Un pequeño ejemplo de escritura en un fichero se muestra a continuación:

```
public void Escribir(){
  String nombre1="Pepe", nombre2="Juan";
  byte[] dato1,dato2;
  int id = 0;

  dato1=nombre1.getBytes();
  dato2=nombre2.getBytes();
  try{
   id=fichero.addRecord(dato1,0,dato1.length);
   System.out.println("Escribiendo ...");
  }catch (RecordStoreException e){
    e.printStackTrace();
  }
  System.out.println("El id del primer registro es: " + id);
 }
```

Por último comentar que una opción interesante a la hora de navegar entre los registros, es la clase RecordEnumeration que ofrece las siguientes funciones: hasNextElement,

hasPreviousElement, nextRecord, nextRecordId, numRecords, previousRecord, previousRecordId, rebuild y reset.

# COMUNICACIONES DE RED MEDIANTE J2ME MIDP

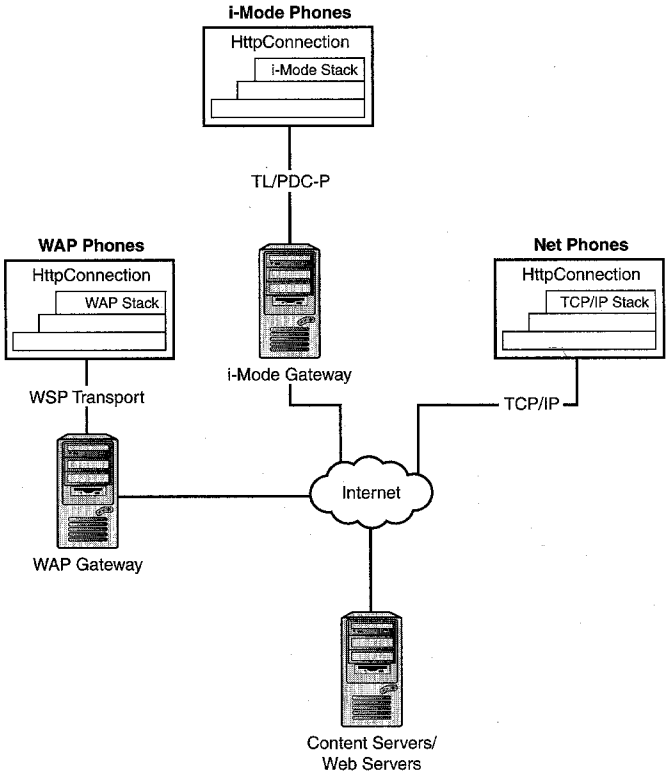
La capacidad de movilidad y conectividad de los dispositivos móviles quedaría mermada si no se dispusiese de un mecanismo para poder acceder a datos remotos y redes corporativas, en resumen, Internet.

El funcionamiento de red en J2ME tiene que ser muy flexible para soportar una gran variedad de dispositivos, para ello se introduce un nuevo concepto, "marco de conexión general" consistente en abstraer el funcionamiento de red y los ficheros de entrada/salida.

Todas las clases e interfaces se encuentran en el paquete **javax.microedition.io**.

J2ME soporta las siguientes formas de comunicación:

. HTTP
. Sockets
. Datagramas
. Puerto serie
. Fichero



Todas estas formas de comunicación son creadas mediante el método Connector.open(), al que se le especificará el tipo de comunicación.

*static Connection open(String connectString, int mode, boolean timeouts)*

*Ejemplo*: *Connection hc = Connector.open("http://www.uc3m.es")*
*Connection dc = Connector.open("datagram://www.uc3m.es:9000")*
*Connection sc = Connector.open("comm:0;baudrate=9000")*

En los ejemplos anteriores se está creando en primer lugar una comunicación http, a continuación mediante datagramas para finalmente llevar a cabo una comunicación serie. En estos ejemplos no se ha especificado ninguno de los dos parámetros posibles: modo y timeout.

El parámetro *mode* permite seleccionar el tipo de acceso, existen tres posiblidades: READ, READ_WRITE y WRITE. El valor tomado por defecto en caso de que no se seleccione ningún modo es de lectura/escritura. Hay que tener en cuenta que si el modo de acceso no es soportado por el protocolo, se lanza una IllegalArgumentException. En caso de que el parámetro *timeouts* esté a true genera una excepción de tipo InterruptedIOException, indicando que se ha superado el tiempo de espera en la apertura de la comunicación.

Cabe destacar que de todas las implementaciones MIDP existentes, MotoSDK es la única que soporta los tres protocolos de red (http, udp, tcp/ip), aunque el J2ME Wireless Toolkit de Sun sólo soporta comunicación http, existe una característica no documentada que permite ejecutar un socket o programa que utilice udp estableciendo para ello la variable de entorno ENABLE_CLDC_PROTOCOLS = INTUITIVE_TOOLKIT.

# XML

XML (Xtensible Markup Language) en los últimos años se ha convertido en un estándar para el intercambio de datos. La finalidad de xml es hacer los datos portables a través de diferentes aplicaciones.

Muchas de las aplicaciones desarrolladas hoy en día conllevan un intercambio de datos constante con fuentes variadas. Estos datos pueden encontrarse en diversos formatos por lo que se hace necesaria una conversión. El enfoque de xml es evitar este paso intermedio generando para ello los llamados "documentos xml".

Un documento xml es un fichero que contiene información categorizada mediante etiquetas; éstas definen las estructuras de los datos así como los elementos que los componen.

Existen una serie de parsers basados en Java. Los más destacados son TinyXML, NanoXML y /Elfred. Estos fueron originalmente escritos en J2SE, aunque ya se encuentran varias versiones existentes para J2ME.

Los ficheros necesarios (así como sus tamaños) para poder utilizar alguno de estos parsers son los siguientes:

Tiny XML → tinyxml_event.jar 10KB
NanoXML → nanoxml_tree.jar 9KB
/Elfred SAX → aelfred_event.jar 18KB

Para el caso del parser de SAX se deben soportar ocho métodos*: endDocument(), startElement(), endElement(), characters(), ignorableWhitespace(), processingInstruction() y setDocumentLocator().*

Un pequeño ejemplo que utiliza el parser NanoXML's SAX sería el siguiente:

```
try{
   Parser parser = ParserFactory.makeParser("nanoxml.sax.SAXParser");
  DemoHandler miHandler = new DemoHandler();
  Parser.setDocumentHandler(miHandler);
 Parser.parse(urlstring);
 ResultString = miHandler.getResultString();
}catch(Exception e){
    System.out.println(e);
}
```

# UNA APLICACIÓN DE DESCARGA

Una vez vistas las principales características y funcionalidades de J2ME se va a proceder a la realización desde cero de una aplicación que dada la URL de una página Web la descargue y almacene en un fichero para su posterior visualización.

El Midlet está estructurado en tres clases:

- **Conexión**: clase encargada de establecer la conexión, descargar la página Web y devolverla como un array de bytes.
- **Fichero**: gestiona todos los aspectos relacionados con el almacenamiento en fichero de las páginas Web.
- **Comunicación**: clase principal con los interfaces que gestiona la descarga de las páginas.

A continuación se van a ir comentando cada una de ellas, así como sus correspondientes métodos.

Primero se comenta la clase Conexión que como su propio nombre indica en esta clase se establece la conexión con el servidor Web indicado en una ventana de texto.

Para trabajar con conexiones es J2ME se debe incluir el paquete javax.microedition.io, el resto de los paquetes depende de la operativa interna, en nuestro caso se han incluido las siguientes:

```
import javax.microedition.io.*;
import java.io.*;
import java.util.*;
```

La clase conexión cuenta con los siguientes atributos:

- *_con* conexión propiamente dicha
- *_sal* flujo de datos de recepción.
- *_datos* representa la información en flujo de bytes que se ha descargado
- *_dirección* URL de la página a la que nos queremos conectar.

En el constructor de la clase se inicializa el atributo _dirección al valor del parámetro que recibe el método.

```
public class Conexion {

   private StreamConnection _con = null;
   private InputStream _sal = null;
   private StringBuffer _datos;
   private String _direccion;

 public Conexion(String direc){
    _direccion = new String(direc);
  }
```

Para poder establecer la comunicación se debe abrir la comunicación indicando la URL a la que se quiere conectar, esto se hace mediante la llamada la método open(URL) de la clase Conector, se debe hacer un cast para indicar que la comunicación es de tipo http, ya que se dispone de distintos tipos de comunicación[3].

En caso de que la dirección introducida no siga el formato correcto un excepción del tipo IllegalArgumentException será lanzada. Si por el contrario el formato es correcto, pero el servidor no está operativo o la dirección introducida no existe la excepción será ConnectionNotFoundException.

Una vez establecida la conexión sin problemas, lo siguiente es crear un flujo de datos de recepción para poder leer del destino.

```
public void abrir() throws ConnectionNotFoundException,
                    IllegalArgumentException,
                    IOException{

   _con = (HttpConnection)Connector.open(_direccion);
   _sal = _con.openInputStream();
 }
```

El siguiente paso es la descarga de la pagina, en este caso se hace leyendo carácter a carácter mediante el flujo de datos creado para esta conexión concreta; los datos son almacenados en el atributo _datos. Si se produce algún error en la lectura se lanza la excepción IOException.

El tratamiento dado a las excepciones dentro de las clases Fichero y Conexion es que en caso de que se produzca una dentro de la propia clase será propagada hacia el exterior para que la clase principoal sea la encargada de gestionarlas de forma adecuada, es decir, que el tratamiento de las excepciones sea independiente de las clases que las generan.

```
public void descargar() throws IOException{

   _datos= new StringBuffer();
   int _ch;

   while ((_ch = _sal.read())!=-1){
     _datos.append((char)_ch);
   }

 }//fin de private readpagina
```

El método getpagina permite acceder a los datos descargados devolviendo una array de bytes con el contenido descargado.

[3] Ver apartado Comunicaciones de red mediante J2ME

```
 public StringBuffer getpagina(){
   return _datos;
 }
```

Cuando ya se han realizado todas las operaciones necesarias se debe cerrar la conexión para liberar recursos asociados con el flujo de datos y la conexión, la conexión finaliza.

```
  public void cerrar() throws IOException{
    _sal.close();
    _con.close();
  }
}
```

A continuación se va a comentar la clase Fichero, que almacena en un Record Strore la página descargada.

Para poder manejar el RecordStore se precisa el paquete javax.microedition.rms.
El record store es una colección de registros localizados por un identificador propio de cada registro. Por simplificar el ejemplo se ha considerado como únicos atributos el nombre del fichero (inicializado en el constructor) y el Record Store del fichero..

```
public class Fichero{
  private RecordStore _fichero =null;
  private String _nombre= null;

  public Fichero(String nomfich){
    _nombre=new String(nomfich);

  }
```

Lo primero es indicar qué fichero queremos abrir y si se desea que se cree en caso de que no exista, esto se logra mediante la llamada al método openRecordStrore. Al tratar de abrir el fichero se pueden producir errores porque éste no se encuentre o bien debido a la falta de espacio en el dispositivo.

```
  public void abrir() throws RecordStoreException,
                             RecordStoreFullException,
                             RecordStoreNotFoundException{

    _fichero = RecordStore.openRecordStore(_nombre,true);
}
```

La operación de borrado de fichero también puede interesar implementarla, ya que se ha como se ha visto, algunos errores pueden ser producidos por la falta de espacio.

```
public void borrar() throws RecordStoreNotFoundException,
                            RecordStoreException{
```

```
    _fichero.deleteRecordStore(_nombre);

}
```

El método que se presenta a continuación es el encargado de añadir un registro al fichero, en nuestro caso tendríamos un único registro que contiene la página descargada.

Este método nos devuelve el atributo _id que contiene el identificador del registro insertado, éste será necesario para la posterior recuperación del fichero.

```
public int insertar(byte[] datos) throws RecordStoreFullException,
                    RecordStoreNotOpenException,
                    RecordStoreException{
  int _id = 0;

  _id=_fichero.addRecord(datos,0,datos.length);
  return _id;
}
```

El siguiente mégodo recupera el archivo descargado accediendo al RecordSotre mediante el identificador, _id, para finalmente devolver un array de bytes.

Las excepciones que se lanzan en éste método se deben a que no se puede cargar el registro o el identificador no tiene asociado ningún registro.

```
public byte[] Visualizar(int id) throws RecordStoreNotOpenException,
                                        InvalidRecordIDException,
                                RecordStoreException{
   byte[] datos=null;

   datos=_fichero.getRecord(id);
   return datos;
}
```

Una vez que ya no se van a realizar más operaciones sobre el Record Store se cierra mediante el método cerrar.

```
public void cerrar() throws RecordStoreNotOpenException,
                    RecordStoreException{

    _fichero.closeRecordStore();

}
}//Cierre de la clase Fichero
```

Creadas ya las clases Comunicación y Fichero, será necesaria una clase que controle el flujo de programa  e instancia a Fichero y Conexión, ésta es la clase Comunicación.

Los paquetes a incluir son los que se indican a continuación.

javax.microedition.midlet.*;
javax.microedition.lcdui.*;
javax.microedition.io.*;
javax.microedition.rms.*;
java.io.*;
java.util.*;

Los atributos de esta clase son los elementos que se añaden al objeto Displayable, como Form, TextBox, TextField, Command, etc.

```
public class Comunicacion extends MIDlet implements CommandListener{

  private Command salir, volver, ir;
  private Display pantalla;
  private Form formulario;
  private TextField direccion,guardar;
  private TextBox pagina=null;
  private TextBox editor=null;

  private Conexion _conexion;
  private Fichero _fichero;

  public Comunicacion(){
    pantalla=Display.getDisplay(this);
    inicia_formulario();
  }
```

En el constructor se indica cuál es la pantalla activa para a continuación llamar al método inicia_formulario.

Como se ve a continuación hay tres métodos denominados inicia_X, donde X es una instancia de algún hijo de la clase Screen. En estos métodos se inicializan los valores de los elementos de X (formulario,TexBoxt, List,..) y se añaden a éste.

En la pantalla se va a visualizar el formulario inicial, el Textbox que muestra el contenido de la página descargada y el TextBox que muestra el fichero.

```
  public void inicia_formulario(){
    formulario=new Form("Conectarse a...");
    direccion = new TextField("Introduzca la URL","",256,TextField.URL);
    guardar=new TextField("nombre de l fichero a guardar","",256,TextField.URL);
    salir = new Command("Salida", Command.EXIT,2);
    ir = new Command("Ir a ", Command.OK,2);

    formulario.append(direccion);
    formulario.append(guardar);
    formulario.addCommand(salir);
    formulario.addCommand(ir);
```

```
    formulario.setCommandListener(this);
}
```

```
public void inicia_pagina(){

  salir = new Command("Salida", Command.EXIT,2);
  volver = new Command("Ver Fichero", Command.OK,2);

  cargarpagina();
  pagina.addCommand(volver);
  pagina.addCommand(salir);
  pagina.setCommandListener(this);
}
```

```
public void inicia_editor(){

  volver = new Command("Volver", Command.EXIT,2);
  salir = new Command("Menu", Command.EXIT,2);

  editor=new TextBox("Visualiza RMS",tratarfichero(),tratarfichero().length(),0);

  editor.addCommand(volver);
  editor.addCommand(salir);
  editor.setCommandListener(this);
}
```

A continuación se presentan los métodos que permiten la ejecución del midlet y que modifican los estados del mismo a lo largo de su ciclo de vida.
La funcionalidad de ellos es la misma que la comentada en el ejemplo HolaMundo, cabe destacar que en el método destroyApp se igualan todos los elementos de la pantalla a nulo, para liberar memoria ,ya que J2ME no cuenta con el recolector de basura de J2SE.

```
 public void startApp() throws MIDletStateChangeException{
   pantalla.setCurrent(formulario);
  }

 public void pauseApp(){
 }
 public void destroyApp(boolean unconditional){
   direccion = null;
   formulario=null;
   pantalla=null;
   editor = null;
   dirección = null;
   guardar = null;
   salir = null;
  volver = null;
```

```
   ir = null;
}
```

Hasta aquí ya se dispone de los interfaces y métodos necesarios para poder ejecutar el midlet, pero falta lo que aporta la funcionalidad a la aplicación. Como se indicó la principio de este apartado, mediante este ejemplo se pretende descargar una página Web y almacenarla en un fichero. Para ello se han implementado los métodos cargarpagina y tratarfichero.

El cargar página se establece la conexión y se descarga la página haciendo uso de la clase Conexión. El contenido de la página se obtiene mediante el método getpagina y se visualiza en el TextBox página.

Como en la clase Conexión no se han capturado las excepciones sino que sólo se han lanzado, ahora hay que capturarlaa y asociarlas funcionalidad, en éste ejemplo simplemente se visualiza el error.

```
public void cargarpagina(){
    Conexion _conecta=null;
    String _texto;
    try{
      _conecta= new Conexion(direccion.getString());
      _conecta.abrir();
      _conecta.descargar();
      _texto =_conecta.getpagina().toString();

      pagina=new TextBox("Pagina cargada",_texto,_texto.length(),0);
      _conecta.cerrar();

   }catch(ConnectionNotFoundException co){

     System.out.println("Error: "+co.getMessage());

   }catch(IllegalArgumentException ia){

     System.out.println("Error: "+ia.getMessage());

   }catch (IOException e){

     System.out.println("Se ha producido un error de I/O");

   }

 }//fin de private readpagina
```

El método tratarfichero abre un fichero en el que almacena la información obtenida en la descarga de la página de Internet para devolvérselo a la aplicación que lo va a tratar, aquí se emplea una instancia de la clase Fichero y sus métodos.

Al igual que en el método anterior se capturan las excepciones y se les asocia la acción de visualizar un mensaje de error.

```
public String tratarfichero(){
String texto= null;

  try{
    _fichero=new Fichero(guardar.getString());
    _fichero.abrir();
    int identificador=_fichero.insertar(pagina.getString().getBytes());
    _fichero.cerrar();
    _fichero.abrir();

    byte[] contenido= _fichero.Visualizar(identificador);
    texto =new String(contenido);
    _fichero.cerrar();

   }catch(RecordStoreNotFoundException e1){
     System.out.println("Error: "+e1.getMessage());
   }catch(RecordStoreFullException e2){
     System.out.println("Error: "+e2.getMessage());
   }catch(RecordStoreNotOpenException e3){
     System.out.println("Error: "+e3.getMessage());
   }catch(InvalidRecordIDException e4){
     System.out.println("Error: "+e4.getMessage());
   }catch(RecordStoreException e5){
     System.out.println("Error: "+e5.getMessage());
   }
  return texto;
}
```

Para finalizar con este ejemplo, sólo queda capturar los eventos que se producen cuando se selecciona una "opción" (un command), para ello está e l método command Action.

Dado que existen tres elementos que lanzan eventos, debido a que el formulario, la pantalla y el editor tienen commands añadidos, lo primero que hay que hacer es ver de dónde provienen los eventos. Esto se comprueba mediante *instanceof* que informa del tipo de objeto que contiene ese command. Una vez que ya se conoce se comprueba qué "opción " es la solicitada para ejecutar el código asociado a dicha opción.

```
public void commandAction(Command c, Displayable s){

if (s instanceof Form) {
   if(c== salir){
    destroyApp(false);
    notifyDestroyed();
   }
   if(c== ir){
     inicia_pagina();
     pantalla.setCurrent(pagina);
```

```
    }
   }
   else if (s instanceof TextBox) {
     TextBox obj = (TextBox) s;
     if (obj == pagina) {
       if(c== volver){
         inicia_editor();
         pantalla.setCurrent(editor);
       }
       if(c== salir){
         destroyApp(false);
         notifyDestroyed();
       }
     }
     if (obj == editor) {
       if(c== volver){
         pantalla.setCurrent(formulario);
       }
     if(c== salir){
       destroyApp(false);
       notifyDestroyed();
      }
     }
   }

}

}
```

Si se siguen los pasos anteriores y se ejecuta la aplicación se observa un interfaz como el que se presenta, en la primera pantalla se solicita la URL y el fichero en el que se desea almacenar. A continuación está la pantalla con la página recién descargada, y en caso de seleccionar la opción Ver Fich, se obtendrá la pantalla Visualiza RMS que contendrá el fichero a visualizar.

Una vez creado todo el código comienza la fase de compilación, a la que seguirán la de preverificación, creación del .jar y ejecución, para ello basta con escribir lo siguiente:

**Compilación**

```
%COMPI%javac.exe -g:none -bootclasspath %RUTAJ2ME%\lib\midpapi.zip -d
%RUTAFUENTES%\tmpclasses -classpath %RUTAFUENTES%
%RUTAFUENTES%\*.java
```

Para compilar los archivos .java (Conexión.java, Fichero.java y Comunicación.java) basta con escribir el comando anterior. Se han utilizado variables de entorno para simplificar el comando. %COMPI% es la ruta donde se encuentra instalado el jkd1.3, %RUTAJ2ME% indica el directorio base de el j2me wireless toolkit y por último en %RUTAFUENTES% como su nombre indica contiene la ruta de los fuentes. Las opciones empleados son –g:none para no incluir información de depuración, -

bootclasspath para indicar la ruta de las clases j2me a utilizar, con -d directorio indicamos dónde queremos que nos deje el compilador las clases recién compiladas y por último –classpath que será la ruta dónde se encuentran los fuentes a compilar.

**Preverificación**

```
%RUTAJ2ME%\bin\preverify.exe -classpath
%RUTAJ2ME%\lib\midpapi.zip;%RUTAFUENTES%\tmpclasses -d
%RUTAFUENTES%\classes %RUTAFUENTES%\tmpclasses
```

Como ya se ha mencionado anteriormente, J2ME realiza parte de la preverificación "off-line", con este comando podemos llevarlo a cabo. Las opciones y rutas son las mismas que en la compilación salvo el directorio tmpclasses que será donde se almacenarán las clases preverificadas.

**Creación del archivo jar**

```
%COMPI%jar cmf %RUTAFUENTES%\META-INF\MANIFEST.MF
%RUTAFUENTES%\Comunicacion.jar -C %RUTAFUENTES%\classes .
```

Una vez compiladas y preverificadas las clases se pasa al empaquetamiento mediante la utilidad jar, para ello basta indicarle las clases a empaquetar así como los recursos adicionales a incluir, en este caso será el manifiesto que será distribuido con el MidLet Suite. Las opciones cmf indican que se desea empaquetar (c) incluyendo un manifiesto (m) para obtener un fichero (f) que en este caso será Comunicación.jar.

El manifiesto incluido en esta aplicación será el siguiente:

```
MIDlet-1: Comunicacion, Comunicacion.png, Comunicacion
MIDlet-Name: Comunicacion
MIDlet-Vendor: Sun Microsystems
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

**Ejecución**

```
set CLASSPATH=%RUTAJ2ME%\lib\kvem.jar; %RUTAJ2ME%\lib\kenv.zip;
%RUTAJ2ME%\lib\lime.jar;%RUTAFUENTES%

%COMPI%java.exe -Dkvem.home=%RUTAJ2ME% -classpath %CLASSPATH%
com.sun.kvem.midp.Main DefaultGrayPhone -descriptor Comunicacion.jad
```

Como se ha mencionado anteriormente es necesario incluir un archivo denominado descriptor para se establece mediante la opción –descriptor, en nuestro caso es el archivo Comunicación.jad, con DefaultGrayPhone se indica el simulador a utilizar (el j2me wireless toolkit proporciona varios con diferentes tamaños de pantalla), finalmente con –Dkvem.home=<nombre> se establece la ruta para buscar las clases de la aplicación y sus recursos.

El fichero descriptor del ejemplo es el siguiente:

```
MIDlet-1: Comunicacion, Comunicacion.png, Comunicacion
MIDlet-Jar-Size: 4390
MIDlet-Jar-URL: Comunicacion.jar
MIDlet-Name: Comunicacion
MIDlet-Vendor: Sun Microsystems
MIDlet-Version: 1.0
```

Finalmente se muestran los resultados de la ejecución.

# GLOSARIO

**CDLC**: Acrónimo de Connected Device Limited Configuration, es una de las configuración existentes para J2ME.

**Configuración**: Definición de la características comunes de una franja horizontal de dispositivos.

**Descriptor:** Fichero con extensión .JAD que contiene información sobre el MIDlet Suite. Este información es requerida por el gestor de instalación de aplicaciones para cerciorarse que es adecuado para la instalación en dicho dispositivo.

**JAR**: Extensión de los ficheros que empaquetan un conjunto de MIDlets haciendo posible su distribución a través de la red.

**Manifiesto**: Fichero que describe el contenido de un fichero .JAR.

**MIDLET**: Aplicación desarrollada en J2ME que implementa MIDP para dispositivos móviles.

**MIDLET suite**: Agrupación de MIDlets

**MIDP**: acrónimo de Mobile Information Device Profile es uno de los perfiles existentes hoy en día.

**Perfil**: Definición de un conjunto de APIs y características de la franja vertical del mercado

**RMS**: Record Management System , es una base de datos simple definida por J2ME MIDP

# BIBLIOGRAFÍA


- [SAMS 2001] Yu Feng y Dr. Jun Zhu. "Wireless Java Programming with J2ME", Junio 2001.
- [SAMS] Michael Morrison. "Wirele ss Java with J2ME in 21 Days", Junio 2001.
- Application for Mobile Information Device. [White Paper]
- J2ME Technology for creating Mobile Devices. [White Paper]
- CLDC Specification. Version 1.0a
- J2ME Wireless Toolkit User's Guide. Version 1.0
- Mobile Information Device Profile (JSR-37). Draft 0.9


DIRECCIONES DE INTERÉS


<http://www.java.sun.com>

<http://java.sun.com/j2se/1.3/download-windows.html>.

<http://www.cun.com/forte/ffj/buy.html>

<http://www.codewarrior.com/>

<http://www.inprise.com/jbuilder/hhe/>

<http://java.sun.com/ products/j2mewtoolkit/ download.html>