Ingeniería en Desarrollo de Software
8º Semestre

Programa de la asignatura:
Programación móvil

Unidad 2: Aplicaciones J2ME

Ciudad de México, febrero del 2026

Clave:
15144845

Universidad Abierta y a Distancia de México

Índice

Unidad 2. Aplicaciones J2ME

Presentación de la unidad

Como ya se ha mencionado con anterioridad, la ventaja de la tecnología J2ME es la producción de aplicaciones híbridas, pero antes de entrar en detalle es importante queconsideres la clasificación de las aplicaciones, las cuales se exponen a continuación:

Aplicaciones nativas: Según Lionbrige (2015) las aplicaciones nativas, se desarrollan específicamente para un dispositivo en particular y su sistema operativo, se descarga desde una tienda virtual y se instala en el dispositivo, están escritas en Java, Objective C, o algún otro lenguaje de programación, aprovechan el hardware y el software específico del dispositivo, lo que significa que las aplicaciones nativas pueden aprovechar la última tecnología disponible en los dispositivos móviles y se pueden integrar con aplicaciones propias como el calendario, los contactos y el correo electrónico.

Aplicaciones web: Las aplicaciones web cuentan con formato para teléfonos inteligentes y tabletas, y se accede a través del navegador web de un dispositivo móvil. Al igual que una aplicación web tradicional, una aplicación web móvil está construida con tres tecnologías básicas: HTML (define texto estático e imágenes), CSS (define la presentación y el estilo), y JavaScript (define las interacciones y animaciones), normalmente se descarga desde un servidor web central cada vez que se ejecuta; las aplicaciones desarrolladas en HTML5 también se pueden ejecutar en un dispositivo móvil para su uso sin conexión.

Aplicaciones híbridas: Estas aplicaciones contienen en su interior el navegador web del dispositivo, y utilizan frameworks de desarrollo basados en lenguajes de programación web (html, CSS y JS), con una aplicación híbrida, gran parte o la totalidad de la interfaz de usuario aparece en una ventana del navegador, en conjunto con una aplicación nativa alrededor de la hibrida para dar acceso a la funcionalidad del dispositivo que no está disponible a través del navegador.

La capacidad de combinar aplicaciones web estándar con código nativo puede reducir significativamente el tiempo y costo de desarrollo.

Desde los usuarios, una aplicación híbrida bien diseñada es muy similar a una aplicación nativa, se descarga desde una tienda de aplicaciones, se almacena en el dispositivo móvil y se usa como una aplicación nativa. Desde los desarrrolladores, en lugar de volver a escribir la aplicación completa para cada plataforma móvil, se escribe al menos una parte del código en HTML, CSS y JavaScript, y reutiliza en distintos dispositivos.

Y son las aplicaciones nativas las que se desarrollarán en esta asignatura, a partir de esta unidad y la subsiguiente.

Logros

Al finalizar la unidad lograrás realizar:

- Identificar la instalación de la aplicación de J2ME.
- Relacionar los componentes de J2ME para el desarrollo de una aplicación en un caso.
- Utilizar los tipos de componentes.
- Utilizar el entorno de desarrollo de aplicaciones móviles J2ME.

Competencia específica

- Utiliza el lenguaje J2ME, para el desarrollo de aplicaciones con características generales para dispositivos móviles mediante un entorno en JAVA.

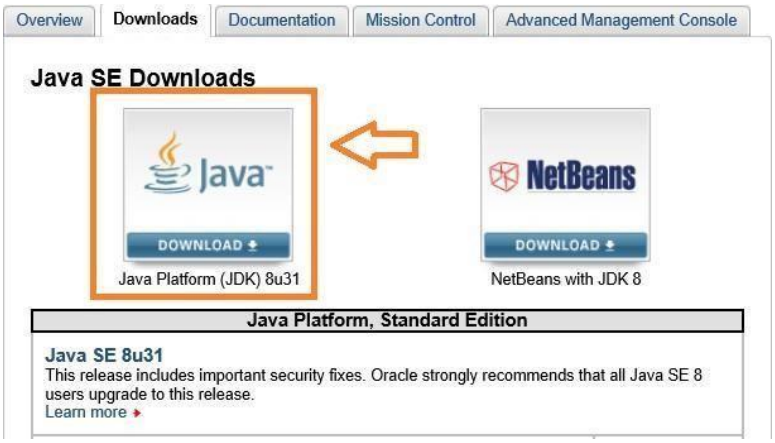
2.1 Instalación de la versión de Oracle J2ME

En el subtema 1.2.1. J2ME, se proporcionó un marco de referencia de esta plataforma de desarrollo, en este tema se expondrá el proceso de instalación del entorno de desarrollo de J2ME, para lo cual es necesario instalar y en el orden que se precisa, la interfaz de programación JAVA SE que establece el marco general de programación en Java, posteriormente el J2ME que establece un marco específico que provee una colección certificada de APIs de desarrollo de software para dispositivos con recursos limitados tales como dispositivos móviles y finalmente la instalación de Wireless Toolkit que proporciona un conjunto de herramientas de construcción, utilerías y un emulador de dispositivo. Los pasos para la instalación son los siguientes:

1. Instalar la última versión de JDK
2. Instalar la última versión de J2ME
3. Instalar la última versión de Wireless Toolkit

A continuación se explicarán cada uno de los pasos anteriores, para la instalación del entorno de desarrollo J2ME.

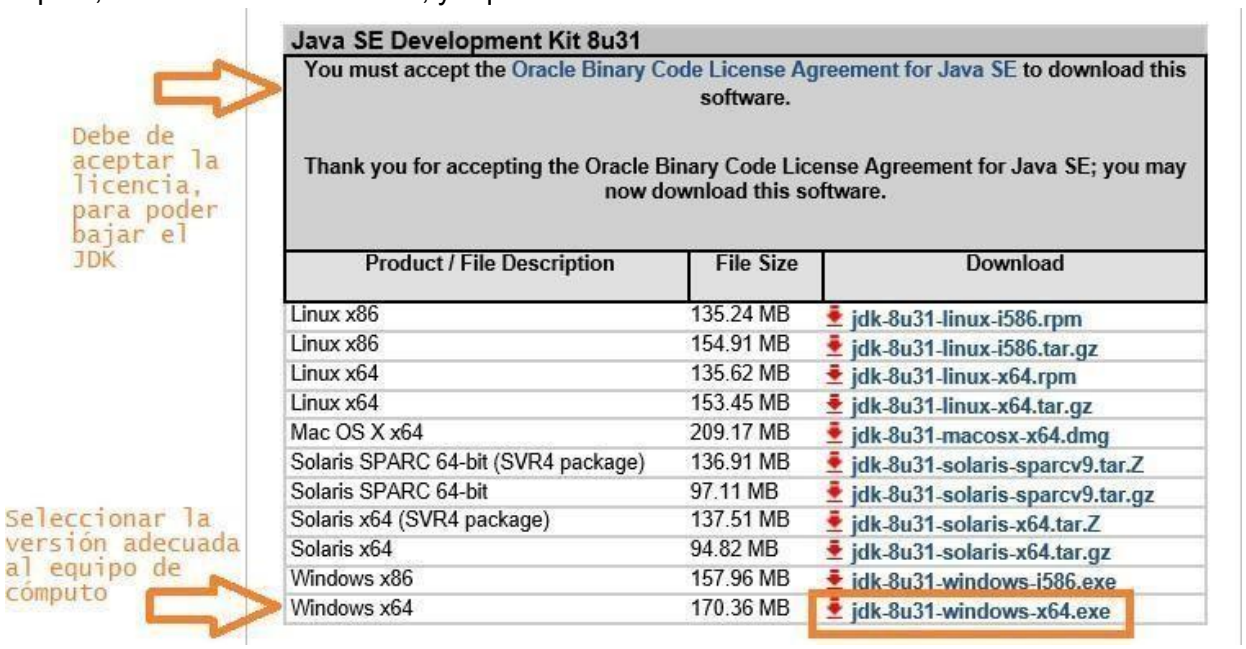
1.-Instalar la última versión de JDK, el Java SE es libre y para obtenerlo únicamente solicita **aceptar** la licencia de uso, para descargar el software.



Plataforma Java SE

La dirección del JDK, es la siguiente:
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
En caso de expirar la dirección anterior, localiza el JDK más reciente en la página oficial de Oracle.

a) Se debe **aceptar** la licencia, **seleccionar** la versión del JDK adecuada al equipo de cómputo, se recomienda a 32 bits, ya que el Wireless Toolkit está a 32 bits.



Java SE Development Kit 8u31

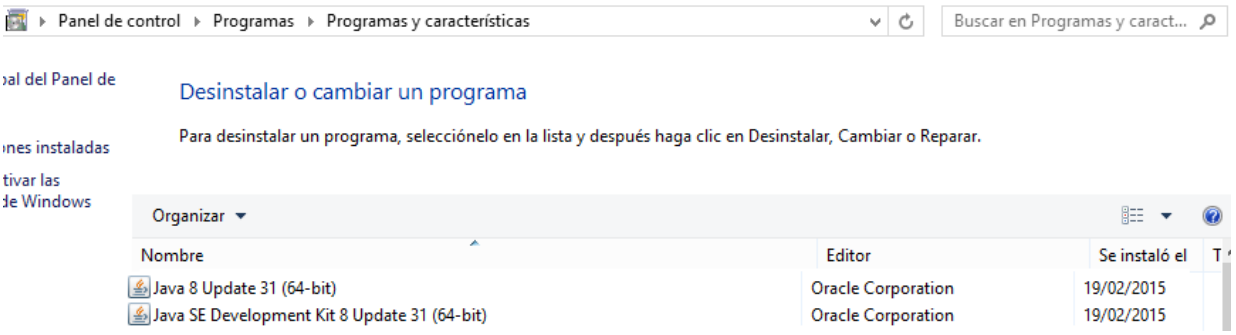
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	135.24 MB	jdk-8u31-linux-i586.rpm
Linux x86	154.91 MB	jdk-8u31-linux-i586.tar.gz
Linux x64	135.62 MB	jdk-8u31-linux-x64.rpm
Linux x64	153.45 MB	jdk-8u31-linux-x64.tar.gz
Mac OS X x64	209.17 MB	jdk-8u31-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	136.91 MB	jdk-8u31-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	97.11 MB	jdk-8u31-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	137.51 MB	jdk-8u31-solaris-x64.tar.Z
Solaris x64	94.82 MB	jdk-8u31-solaris-x64.tar.gz
Windows x86	157.96 MB	jdk-8u31-windows-i586.exe
Windows x64	170.36 MB	jdk-8u31-windows-x64.exe

Versión 8u31 Java SE Development

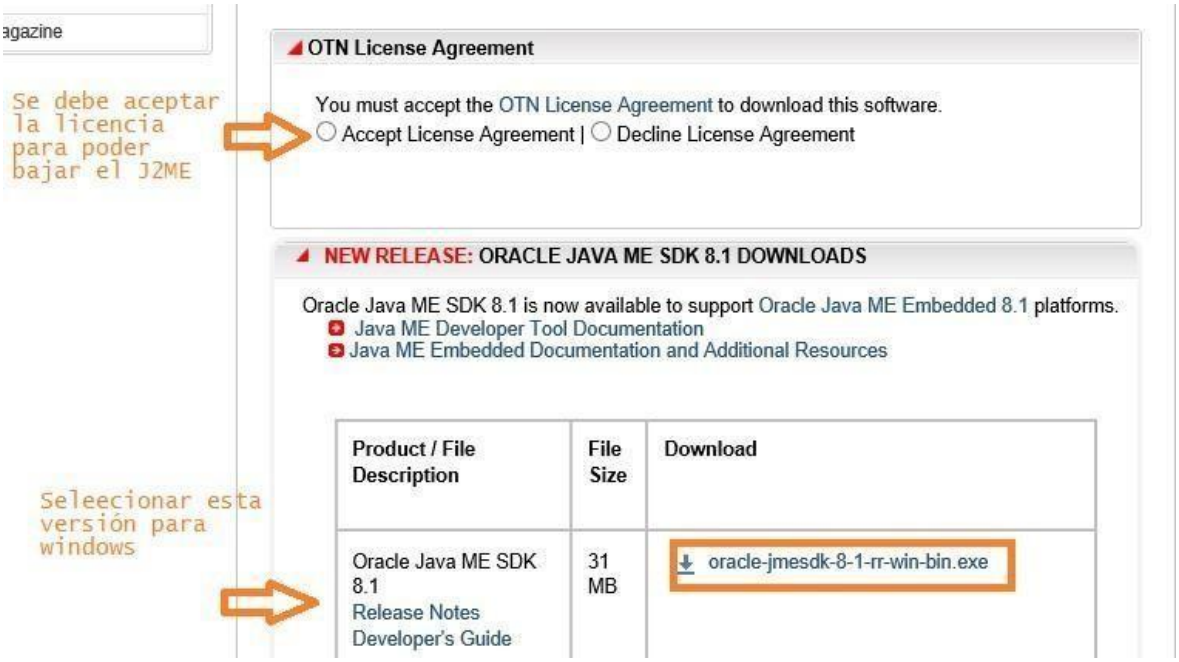
b) **Comprobar** que se haya instalado:



Verificación de instalación

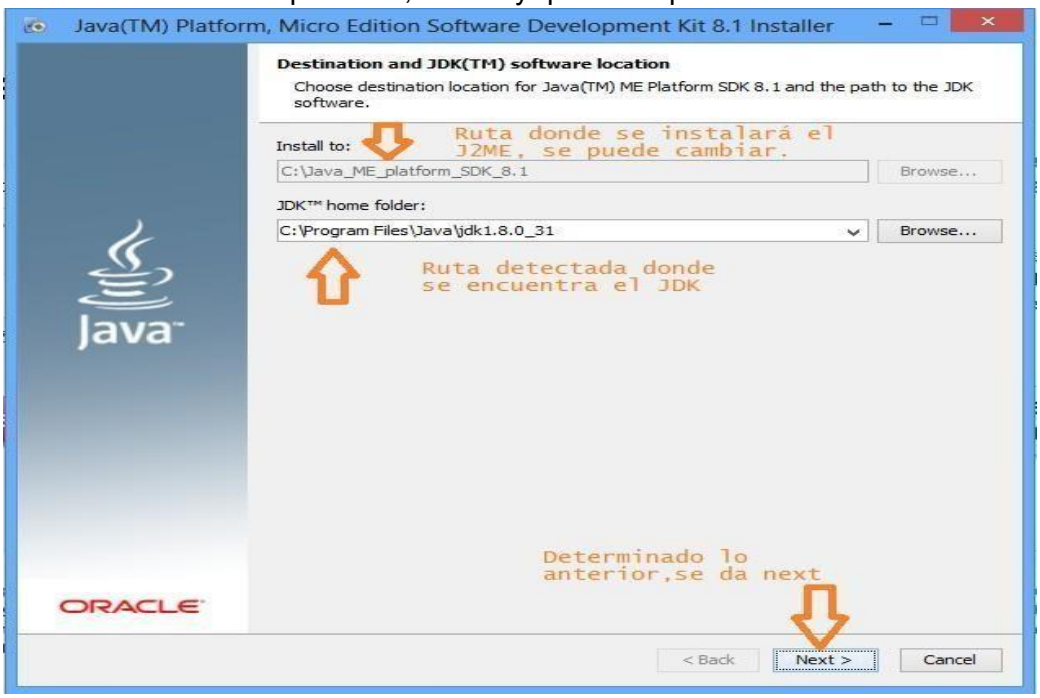
2. Instalar la última versión de J2ME, la cual es libre y únicamente solicita **aceptar** la licencia de uso, para descargar el software. Es importante que identifiques la última versión de J2ME en la página oficial de Oracle, en la actualidad la dirección es: http://www.oracle.com/technetwork/java/embedded/javame/javame-sdk/downloads/javamesdkdownloads-2166598.html
En caso de que la dirección anterior expire, localiza la versión más reciente de J2ME en la página oficial de Oracle.

a) Se debe **aceptar** la licencia, y dependiendo del sistema operativo del equipo de cómputo **seleccionar** la versión adecuada, en este caso se selecciona la versión para Windows.



Versión 8.1 Oracle Java ME SDK

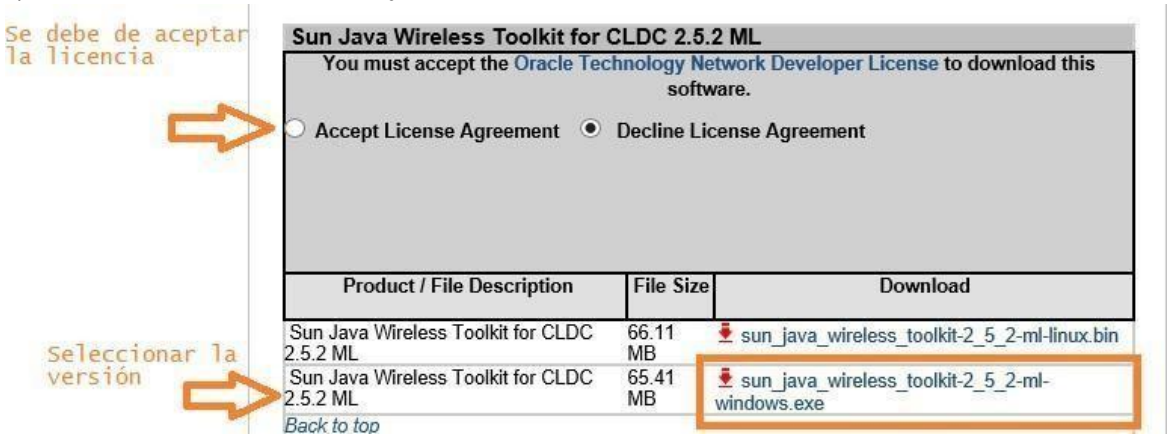
b) Es posible **cambiar** la ruta de instalación del J2ME, ya sea que decida en la ruta donde se encuentra el JDK o cualquier otra, sólo hay que tener presente en dónde se instaló.



Validación de ruta de instalación

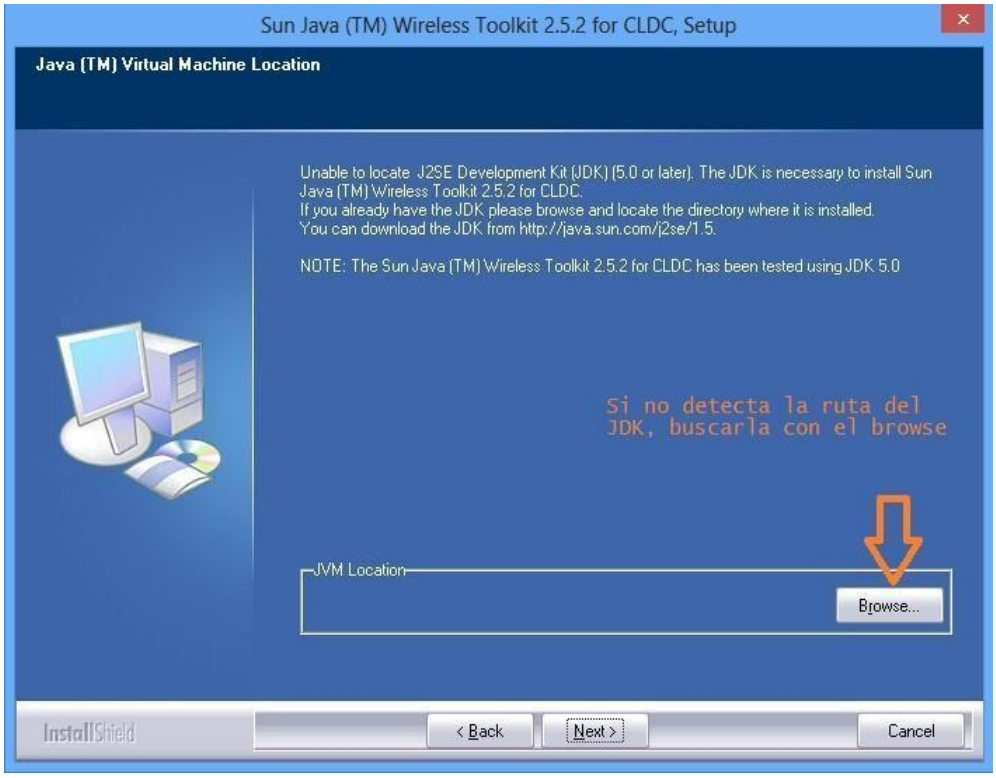
3. Para **instalar** la última versión de J2ME Wireless Toolkit, seleccione **ejecutar** en vez de guardar. La dirección actual es:
http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javame-419430.html#sun_java_wireless_toolkit-2.5.2_01b-oth-JPR
En caso de expirar la dirección anterior, localiza la versión más reciente de Java ME en el sitio oficial de Oracle.

a) Se debe aceptar la licencia y seleccionar la versión.



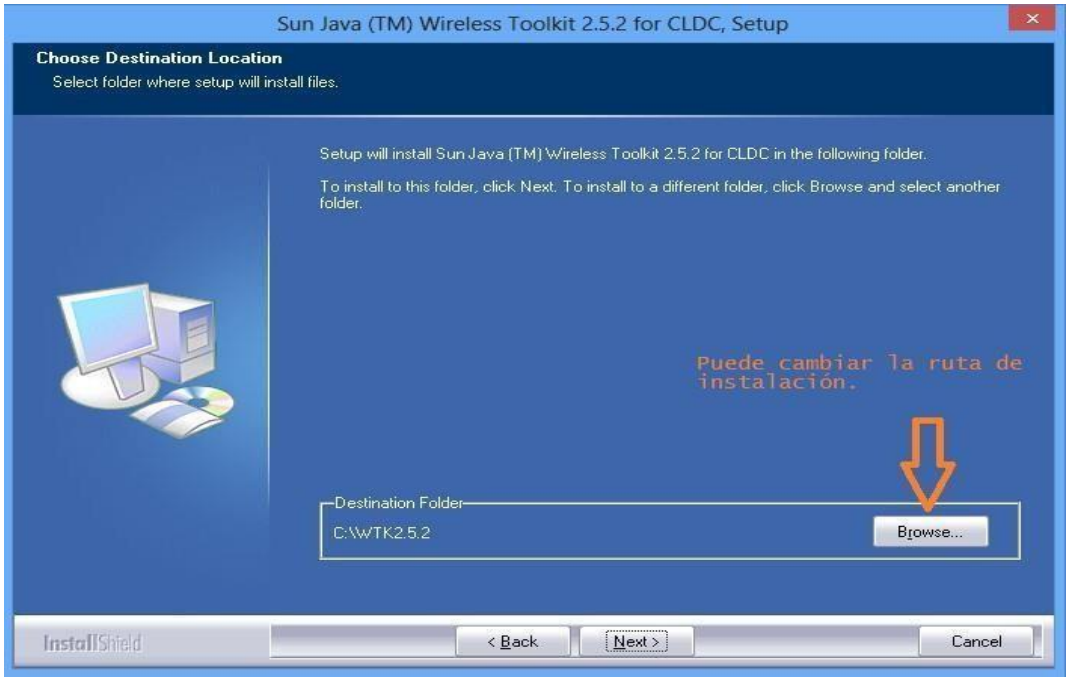
Aceptación de la licencia de Java Wireless Toolkit 2.5.2

b) **Verificar** que Java **detecte** la ruta del JDK.



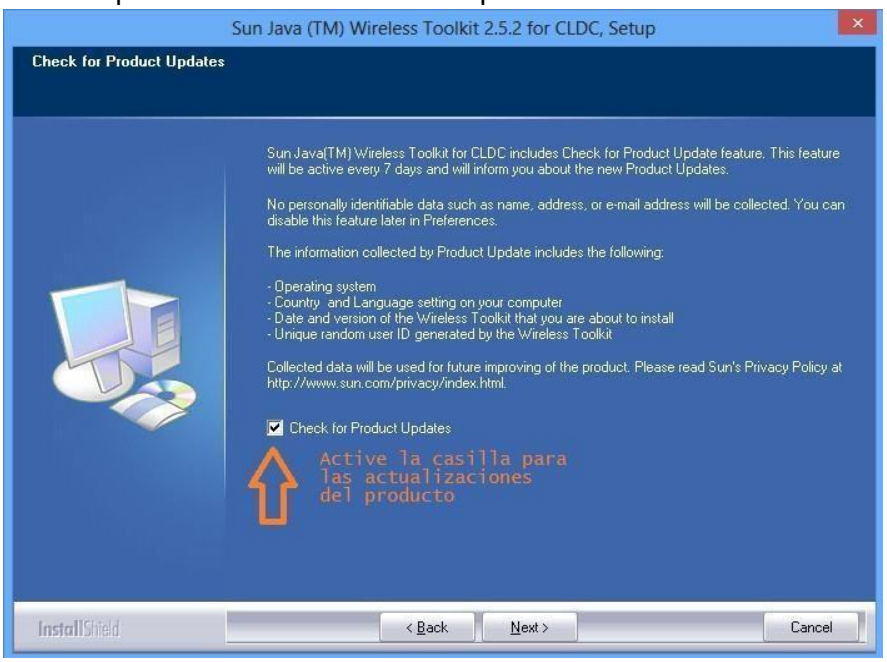
Selección de la ruta del J2SE JDK

c) Si no se detecta la ruta del JDK, puedes **seleccionar** otra ruta de instalación, pero recuerda tenerla presente.



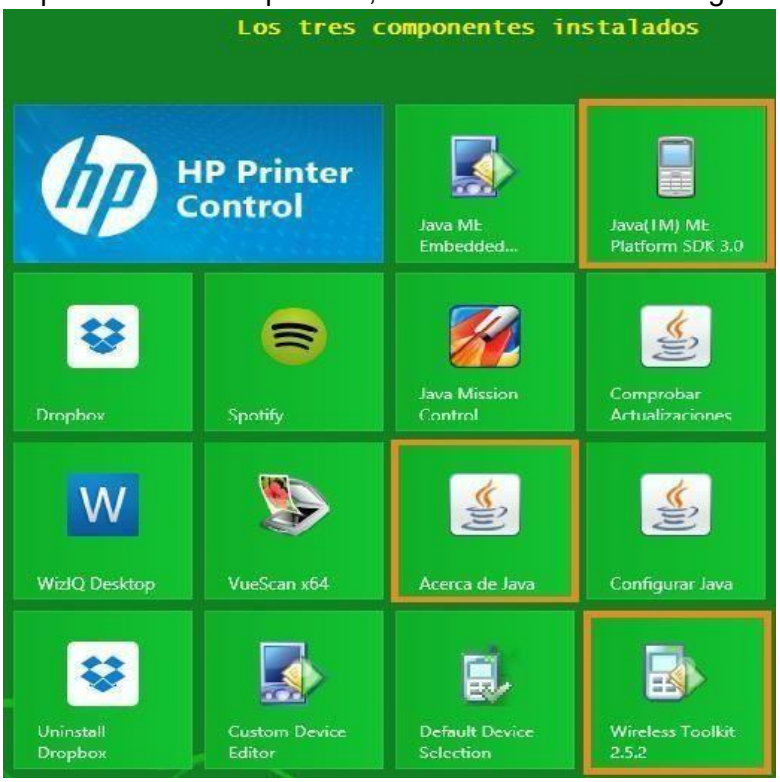
Ruta de instalación del Wireless Toolkit 2.5.2

d) **Activar** la casilla para las actualizaciones del producto.

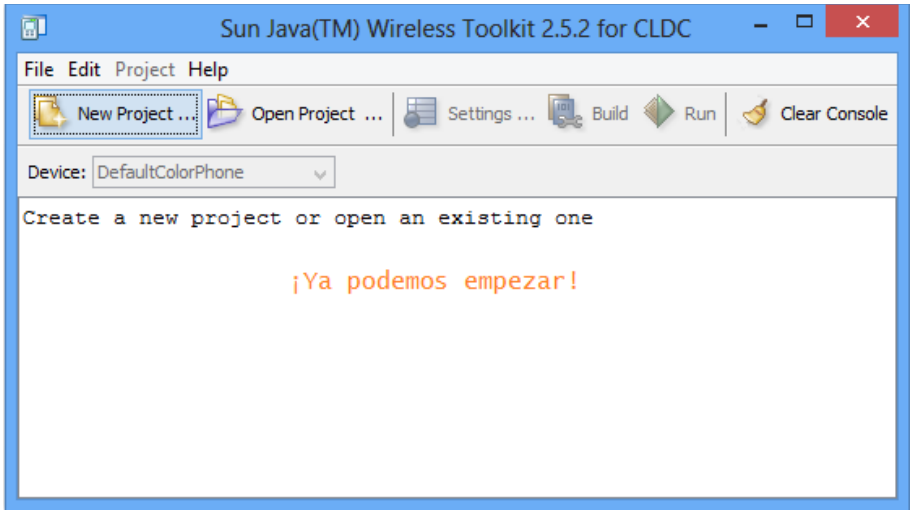


Actualizaciones del producto

Con este último paso se ha concluido con la instalación, puedes saberlo porque aparecen los iconos de las aplicaciones en la pantalla, como se observa en la siguiente imagen:



Pantalla de aplicaciones instaladas



Pantalla de Wireless Toolkit 2.5.2

Definido ya el entorno, se procede a exponer a detalle las interfaces gráficas para el desarrollo de las aplicaciones móviles.

2.2. Interfaces gráficas

Como ya lo revisaste en el tema 2.1. J2ME en la Unidad 1, los perfiles son "bibliotecas de clases específicas, orientadas a implementar funcionalidades de más alto nivel para familias específicas de dispositivos" (Aranaz, 2009, p.27). El perfil a utilizar será el MIDP, el cual define las interfaces de usuario (UI, por sus siglas en inglés de User Interface), imágenes, sonidos y base de datos, y como parte fundamental del MIDlet es el uso de las interfaces gráficas que son aquellas que permiten darle una presentación a la aplicación, éstas se dividen en dos niveles según Gálvez y Ortega (2003, p. 51):

- La UI de **primer** nivel, con la cual será posible integrar componentes tales como: **botones**, **cajas** de **texto** y **formularios**, la implementación de estos elementos está en cada dispositivo, el uso de esta UI le da portabilidad a la aplicación en cuestion.
- Las aplicaciones que utilizan en gran medida la UI de **alto** nivel están diseñadas para aplicaciones de negocios cuyos elementos cliente se ejecutan en los MID. Para estas aplicaciones, la portabilidad entre dispositivos es importante, y la UI de alto nivel proporciona un alto nivel de abstracción y ofrece muy poco control sobre el aspecto y la sensación.

La UI de usuario de segundo nivel, le da al programador control sobre los recursos del dispositivo ya que controla eventos tal como el rastreo de las pulsaciones de las teclas, la UI es usada para la creación de juegos donde el usuario tiene control sobre lo que aparece en la pantalla.
En el siguiente subtema se revisarán las características de las UI de primer nivel y segundo nivel y su uso en las aplicaciones.

2.2.1 Características

Como se especificó anteriormente, la UI de alto nivel proporciona un alto nivel de abastracción, y se manifiesta en las siguientes características:

- El dibujo real en la pantalla del MID se realiza mediante la implementación de las clases específicas para ello, tales como la clase **Screen**. Las aplicaciones no definen el aspecto visual (por ejemplo, forma, color, tipo de letra, etc.) de los componentes.
- La navegación, desplazamiento y otra interacción primitiva está encapsulada por la aplicación, y la aplicación no es consciente de estas interacciones.
- Las aplicaciones no pueden acceder a los dispositivos de entrada concretos como teclas individuales específicas.

En concreto, cuando se utiliza el API de alto nivel, se supone que la implementación subyacente va a hacer la adaptación necesaria para el hardware del dispositivo, la interfaz de usuario y el estilo nativo. Las clases que proporcionan el API de alto nivel son las subclases de la clase **screen** como se puede observar más adelante en la imagen Jerarquía de clases de la clase Display.

Por otro lado, las características de la UI de bajo nivel, teniendo en cuenta que proporcionan muy poca abstracción, son usadas para diseñar aplicaciones que necesitan una colocación precisa y control de los elementos gráficos, así también, necesitan tener acceso a los eventos de entrada de bajo nivel, que cuentan con las características especiales del dispositivo, el ejemplo típico de una aplicación de este tipo sería un juego. Las características de la UI se exponen a continuación:

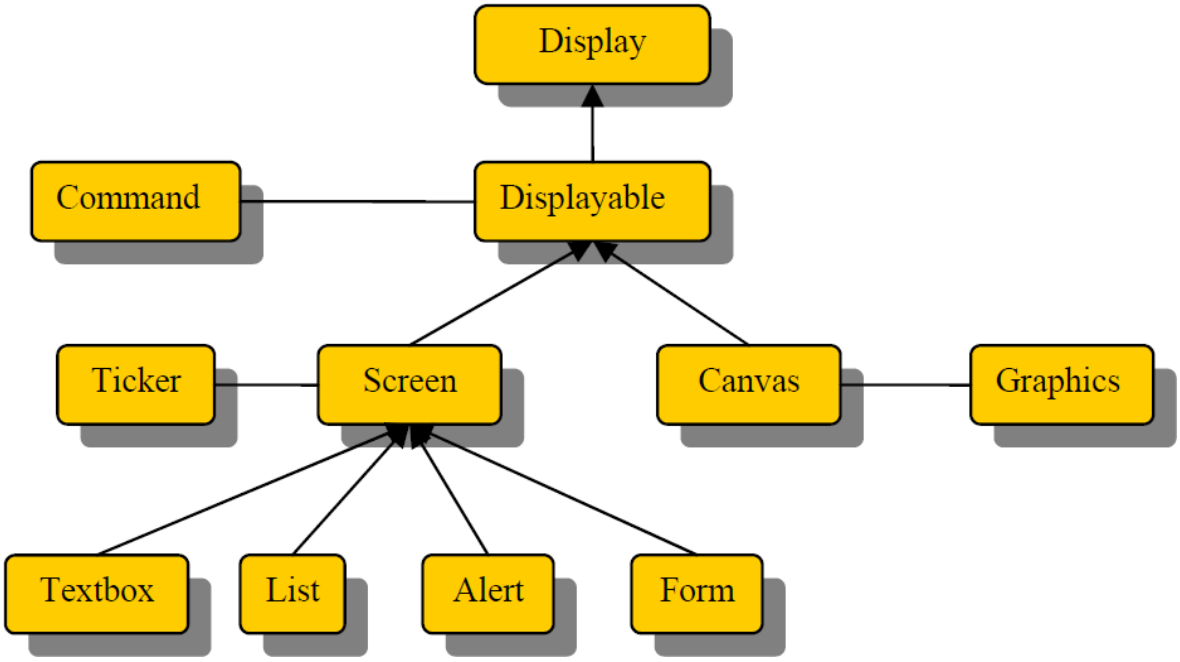
• Tiene un **control** total de lo que se dibuja en la pantalla.
• **Escucha** los eventos primitivos como el pulsado y el soltado de las teclas.
• Tiene **acceso** con teclas concretas y a otros dispositivos de entrada.

Las clases que proporcionan la API de bajo nivel son **Canvas** y **Graphics**. Las aplicaciones que se programan con la UI de bajo nivel no garantizan que sea portátil, por lo que la UI **proporciona** los **medios** para acceder a los detalles que son específicos de un dispositivo en particular. Cabe mencionar que las interfaces están contenidas en el siguiente paquete: **Package javax.microedition.lcdui**

Gálvez y Ortega (2011) señalan que este paquete engloba el interfaz de usuario con pantalla LCD, con los elementos gráficos a utilizar en las aplicaciones.
Considerando el uso de las interfaces gráficas, sus características principales y su ubicación, es importante conocer los componentes que las conforman, y es el tema siguiente a abordar.

2.2.2. Componentes

Las clases de las UI, de alto y bajo nivel son los componentes de abstracción central de la UI del MIDP, es un objeto Displayable, que encapsula gráficos específicos del dispositivo generado por la entrada del usuario. Sólo un objeto **Displayable** puede ser visible a la vez y el usuario puede ver e interactuar sólo con los contenidos de ese displayable, el objeto se deriva de la clase Display, según imagen siguiente.



Jerarquía de clases de la clase Display (Gálvez y Ortega, 2011,p 52)

Display: La sintaxis de esta clase se define de la siguiente forma (Sun Microsystems, 2006a)**:**

```
public class Display extends Object
```

La clase **Display** representa el controlador de los dispositivos de visualización y de los dispositivos de entrada al sistema, es en donde el MIDlet coloca su interfaz incluye métodos para recuperar propiedades del dispositivo y para requerir los objetos que se mostrarán en el dispositivo.

Hay una instancia de Display por MIDlet y la aplicación puede obtener una referencia a esa instancia llamando al método getDisplay (). La aplicación puede llamar al método getDisplay () en cualquier momento durante su ejecución (startApp, pauseApp, destroyApp), a continuación, se expone un ejemplo:

```
// se obtiene el dispostivo de la pantalla
display = Display.getDisplay(this);
```

Los métodos de la clase **Display**, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método
void	`callSerially(Runnable r)` Interface Runnable Hace que el objeto r sea `Runnable` con el método `run()`, que llama más tarde, en serie con el flujo de eventos, poco después de la finalización del ciclo de repintado.
boolean	`flashBacklight(int duration)` Pide un efecto de parpadeo de luz de fondo del dispositivo.
int	`getBestImageHeight(int imageType)` Regresa la mejor altura de la imagen de un tipo de imagen dada.
int	`getBestImageWidth(int imageType)` Regresa el mejor ancho de la imagen para un tipo de imagen dada.
int	`getBorderStyle(boolean highlighted)` Regresa el estilo de trazo utilizado para el dibujo del perímetro en función del estado del componente (resaltado / no, seleccionado).
int	`getColor(int colorSpecifier)` Regresa uno de los colores de la combinación de colores de interfaz de usuario de alto nivel, en forma `0x00RRGGBB` basada en `colorSpecifier`.
Displayable	`getCurrent()` Obtiene el Displayable actual, objecto para este MIDlet.
static Display	`getDisplay(MIDlet m)` Obtiene el Display, objeto que es único para este MIDlet.
boolean	`isColor()` Obtiene información sobre el soporte de color del dispositivo.
int	`numAlphaLevels()` Obtiene el número de los niveles de transferencia alpha soportado por esta implementación.
int	`numColors()`

	Obtiene el número de colores (Si `isColor()` es `true`) o los niveles de grises (Si `isColor()` es `false`) que pueden ser representados en el dispositivo.
`void`	`setCurrent(Alert alert, Displayable nextDisplayable)` Pide que esta `Alert` sea actual, y que `nextDisplayable` sea la actual después de que la alerta sea despedida.
`void`	`setCurrent(Displayable nextDisplayable)` Pide que una `Displayable` diferente sea visible en la pantalla.
`void`	`setCurrentItem(Item item)` Solicitudes al `Display` que contiene este Item para que sea el actual, desplaza la `Displayable` para que este `Item` esté visible, y posiblemente asignándole la atención al `Item`.
`boolean`	`vibrate(int duration)` Solicitudes de operación de vibrador del dispositivo.

Métodos de la clase Display (Sing y Knudsen 2005, p.401).

Displayable (Sun Microsystems, 2006b)
La sintaxis de esta clase es la siguiente:

```
public abstract class Displayable extends Object
```

Displayable es la clase de donde se **derivan** todas las **UIs** de usuario de un MIDlet y como no es posible hacer uso de la clase directamente por ser abstracta, se hace uso de las clases derivadas que son Canvas y Screen según imagen anterior -Jerarquía de clases de la clase Display.

Un objeto displayable se puede colocar en la pantalla, puede tener un título, ticker[1], comandos y un **listener** (método que escucha eventos) asociado a él. Los contenidos que se muestran y su interacción con el usuario se definen por las subclases.
A menos que se especifique lo contrario por una subclase, el estado por defecto de los objetos Displayables de nueva creación es el siguiente:
• No es visible en la pantalla
• No hay **Ticker** asociado a este **Displayable**
• El título es **nulo**;
• No hay **comandos presentes**; y
• No hay **CommandListener** presente.

[1] Una pieza de texto que corre continuamente a través de la pantalla

Ejemplo

```
//La clase commandAction tiene como parámetros el objeto c de la Clase Command y el objeto d de la clase Displayable
public void commandAction(Command c, Displayable d) {…. }
//Muestra el contenido en mensaje1 en el objeto de de la clase Displayable
display.setCurrent(mensaje1,d);  // Muestra un mensaje de pantalla
```

Los métodos de la clase Displayable, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método
void	**addCommand**(Command cmd) Agrega commando al objeto Displayable.
int	**getHeight**() Obtiene la altura en pixeles del área desplegable disponible para la aplicación.
Ticker	**getTicker**() Obtiene el ticker usado por el objeto Displayable.
String	**getTitle**() Obtiene el título del objeto Displayable.
int	**getWidth**() Obtiene el ancho en pixeles del área disponible del objeto desplegable para la aplicación.
boolean	**isShown**() Checa si el Displayable está actualmente visible en la pantalla.
void	**removeCommand**(Command cmd) Remueve un commando desde el Displayable.
void	**setCommandListener**(CommandListener l) Establece un oyente para los commandos del Displayable, reemplazando cualquier previo definido CommandListener.
void	**setTicker**(Ticker ticker) Establece un ticker para ser usado con el Displayable, reemplazando cualquier ticker previo
void	**setTitle**(String s) Establece el título del Displayable.
protected void	**sizeChanged**(int w, int h) La implementación llama este método cuando el área disponible del Displayable está siendo cambiado.

Métodos de la clase Displayable (Sing y Knudsen 2005, p.401)

Command public class **Command** extends Object (Oracle midp 2.0,2014)

La clase Command es un constructor que encapsula la información semántica de una acción. El comportamiento que el comando activa se encapsula en este objeto. Esto significa que el comando sólo contiene información acerca del "comando", no de la acción que está sucediendo cuando el comando se activa. La acción se define en un CommandListener asociado con el Displayable. Los objetos comandos se muestran en la interfaz de usuario y la forma en que se presentan pueden depender de la información semántica contenida dentro del comando.

Los comandos pueden implementarse en cualquier UI que tenga una semántica para la activación de una sola acción. Esto, por ejemplo, puede ser un botón suave, elemento de un menú, o alguna otra UI. Por ejemplo, una interfaz de voz puede presentar estos comandos como etiquetas de voz.

El mapeo UI también puede depender del número total de los comandos. Por ejemplo, si una aplicación solicita comandos más abstractos que se pueden asignar a los botones físicos disponibles en un dispositivo, el dispositivo puede utilizar una interfaz humana alternativa, como un menú. Por ejemplo, los comandos abstractos que no se pueden asignar a los botones físicos se colocan en un menú y la etiqueta **Menú** se le asigna a uno de los botones programables.

Un comando contiene cuatro piezas de información: una etiqueta corta, una etiqueta larga opcional, un tipo, y una prioridad. Una de las etiquetas se utiliza para la representación visual de la orden, mientras que el tipo y la prioridad indican la semántica del comando.

Ejemplo:

```
// Se declaran los objetos tipo Command, para que puedan
accederse desde cualquier método de la clase.
private Command cmdExit;
private Command cmdHelp
//Se crean los objetos y se les asigna un contenido
cmdExit = new Command("Salir", Command.EXIT, 1);
cmdHelp = new Command("Ayuda", Command.HELP, 2);
//Se agregan los commandos al formulario
formulario.addCommand(cmdExit);
formulario.addCommand(cmdHelp);
//Uso de los comandos
```

```
f (c == cmdExit) {
destroyApp(true);
notifyDestroyed();
}else if (c == cmdHelp) {…}
```

Los métodos de la clase **Command**, se exponen a continuación:

Tipo que devuelve el método	*Método y uso del método
int	**getCommandType**() Obtiene el tipo de comando.
String	**getLabel**() Obtiene la etiqueta corta del comando.
String	**getLongLabel**() Obtiene la etiqueta larga del comando.
int	**getPriority**() Obtiene la prioridad del comando.

Métodos de la clase Command (Sing y Knudsen 2005, p.399)

Canvas

public abstract class **Canvas** extends Displayable (Oracle midp 2.0,2014)

La clase **Canvas** es una clase base para escribir aplicaciones que se necesitan para manejar eventos de bajo nivel y emitir llamadas de gráficos para dibujar en la pantalla. Las aplicaciones para diversos juegos hacen uso intensivo de esta clase, desde una perspectiva de desarrollo de aplicaciones, la clase Canvas es **intercambiable** con la clase estándar Screen, por lo que una aplicación puede **mezclar canvas** con **screen**, según sea necesario. Por ejemplo, una pantalla de **List** puede ser usada para seleccionar la pista de un juego de carreras, y una subclase de Canvas implementaría el juego real.

La clase Canvas proporciona al desarrollador métodos para manejar las acciones del juego, los eventos claves y eventos de puntero (si es compatible con el dispositivo). También proporciona métodos para identificar las capacidades y el mapeo de las teclas de las acciones del juego. La clase Canvas permite tener un **Listener** para los comandos, siendo abstracta se hace uso de ella a través de otras subclases, por ejemplo su método **paint(),** es abastracto, por lo que hay que implementarlo en la subclase que lo mande a llamar.

Ejemplo:

```
//Se crea la clase NuevaCanvas que hereda de Canvas
class NuevaCanvas extends Canvas
     implements CommandListener {
    //Constructor de la clase NuevaCanvas
    NuevaCanvas(){
    }
    protected void paint(Graphics g) {
        //se define las caracteristicas gráficas
    }
    public void commandAction( Command c, Displayable d) {
    }
  } // fin de la definición de NuevaCanvas
```

Los métodos de la clase **Canvas**, se exponen a continuación:

Tipo que devuelve el método	***Método y uso del método**
int	**getGameAction**(int keyCode) Obtiene la accion de juego asociada con el código de tecla del dispositivo.
int	**getKeyCode**(int gameAction) Obtiene un código de tecla que corresponde a la accion de juego especificado en el dispositivo.
String	**getKeyName**(int keyCode) Obtiene una cadena de información para una tecla.
boolean	**hasPointerEvents**() Comprueba si la plataforma admite eventos de presionar y liberar el puntero
boolean	**hasPointerMotionEvents**() Comprueba si la plataforma admite eventos de movimiento del puntero.
boolean	**hasRepeatEvents**() Comprueba si la plataforma puede generar eventos repetidos cuando una tecla se mantiene presionada.
protected void	**hideNotify**() La implementación llama a `hideNotify()` después de que el `Canvas` ha sido removido de la pantalla.
boolean	**isDoubleBuffered**() Comprueba si el método `Canvas` es doblemente almacenado por la implementación.
protected void	**keyPressed**(int keyCode)

	Llamado cuando una tecla es presionada.
protected void	**keyReleased**(int keyCode Llamado cuando una tecla es liberada.
protected void	**keyRepeated**(int keyCode) Llamado cuando una tecla es repetida (pulsada).
protected abstract void	**paint**(Graphics g) Pinta el Canvas.
protected void	**pointerDragged**(int x, int y) Llamado al arrastrar el puntero.
protected void	**pointerPressed**(int x, int y) Llamado al presionar el puntero.
protected void	**pointerReleased**(int x, int y) Llamado al liberar el puntero.
void	**repaint**() Pide un repintado de todo el Canvas.
void	**repaint**(int x, int y, int width, int height) Pide un repintado para una región específica del Canvas.
void	**serviceRepaints**() Forza cualquier solicitud pendiente para ser aplicado inmediatamente.
void	**setFullScreenMode**(boolean mode) Controla ya sea si el lienzo está en modo de pantalla completa o en modo normal.
protected void	**showNotify**() La aplicación llama showNotify() inmediatamente antes de que el Canvas se haga visible en la pantalla.
protected void	**sizeChanged**(int w, int h) Llamado cuando el área dibujable del Canvas está siendo cambiado.

Métodos de la clase Canvas (Sing y Knudsen 2005, p.398)

Graphics

public class **Graphics** extends Object (Oracle midp 2.0,2014)

La clase proporciona una capacidad simple de representación geométrica en 2D, las primitivas de dibujo que se proveen son para texto, imágenes, líneas, rectángulos y arcos, también con relleno sólido y esquinas redondeadas.

Se proporciona un modelo de color de 24 bits, con 8 bits para cada uno de los componentes rojo, verde y azul de un color, no todos los dispositivos son compatibles con el valor de color de un total de 24 bits y por lo tanto se asignan colores solicitados por la aplicación en los colores disponibles en el dispositivo, esta información se proporciona en

la clase Display para la obtención de las características del dispositivo, tales como el color disponible y el número de niveles de gris diferentes disponibles. Las solicitudes también pueden utilizar getDisplayColor () para obtener el color real que se muestra para un color solicitado. Esto permite que las aplicaciones adapten su comportamiento a un dispositivo sin comprometer la independencia de dispositivos.

Ejemplo:

```
// Definición de clase Graphics que se crea en el método paint
protected void paint(Graphics g) {
        int w = getWidth();
        int h = getHeight();
        g.setGrayScale(255);
        g.fillRect( 0,0,w,h );
        g.setGrayScale(15*16);
        h = Math.min( w, h );
        long mf = 100000000;
         int ang1 = (int)(( (n1*(36000*mf/lngResultado))
+50*mf)/(100*mf)) ;
        int orig=90;
        //Formato RGB
        g.setColor(255,200,200 );
        g.fillArc(0,0,h,h, orig, (int)ang1);
        //g.setGrayScale(13*16);
        g.setColor(200,200 , 255);
        g.fillArc(0,0,h,h, (int)(orig+ang1), (int)(360-ang1) );

        g.setGrayScale(0);
        g.drawString("A "+n1+"  ", h/2, h/2-10,
                Graphics.BASELINE|Graphics.RIGHT);
        g.drawString("  B "+n2, h/2, h/2-10,
                Graphics.BASELINE|Graphics.LEFT);
        g.drawString("Total "+(n1+n2), h/2, h/2,
                Graphics.TOP|Graphics.HCENTER);
    }
```

Los métodos de la clase Graphics, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método
void	**clipRect**(int x, int y, int width, int height) Intersecta el recorte actual con el rectángulo especificado.
void	**copyArea**(int x_src, int y_src, int width, int height, int x_dest, int y_dest, int anchor) Copia el contenido del área rectangular (x_src, y_src, width, height) a un área destino, cuyo punto de anclaje identificado se

	encuentra en `(x_dest, y_dest)`.
void	**drawArc**`(int x, int y, int width, int height, int startAngle, int arcAngle)` Dibuja el contorno de un arco circular o elíptico que cubre el rectángulo especificado, utilizando el color actual y estilo de trazo.
void	**drawChar**`(char character, int x, int y, int anchor)` Dibuja el carácter especificado utilizando el tipo de letra y el color actual.
void	**drawChars**`(char[] data, int offset, int length, int x, int y, int anchor)` Dibuja el carácter especificado utilizando el tipo de letra y el color actual.
void	**drawImage**`(Image img, int x, int y, int anchor)` Dibuja la imagen especificada mediante el punto de anclaje.
void	**drawLine**`(int x1, int y1, int x2, int y2)` Dibuja una línea entre las coordenadas `(x1,y1)` y `(x2,y2)` usando el color y estilo actual.
void	**drawRect**`(int x, int y, int width, int height)` Dibuja el contorno del rectángulo especificado utilizando el estilo y color actual
void	**drawRegion**`(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor)` Copia una región de la imagen de origen especificada a una ubicación dentro del destino, posiblemente transformando (rotación y reflexión) los datos de imagen usando la función transformación.
void	**drawRGB**`(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height, boolean processAlpha)` Representa una serie de valores RGB + transparencia independientes del dispositivo en una región determinada.
void	**drawRoundRect**`(int x, int y, int width, int height, int arcWidth, int arcHeight)` Dibuja el contorno del rectángulo de esquinas redondeadas especificado utilizando el estilo y color actual.
void	**drawString**`(String str, int x, int y, int anchor)` Dibuja la `String` especificada usando la fuente y color actual.
void	**drawSubstring**`(String str, int offset, int len, int x, int y, int anchor)` Dibuja la `String` especificada usando la fuente y color actual.
void	**fillArc**`(int x, int y, int width, int height, int startAngle, int arcAngle)` Llena un círculo o arco elíptico cubriendo el rectángulo especificado.

void	**fillRect**(int x, int y, int width, int height) Llena el rectángulo especificado con color actual.
void	**fillRoundRect**(int x, int y, int width, int height, int arcWidth, int arcHeight) Llena la esquina redondeada especificada del rectángulo con el color actual.
void	**fillTriangle**(int x1, int y1, int x2, int y2, int x3, int y3) Llena el triángulo especificado con el color actual.
int	**getBlueComponent**() Obtiene el componente azul del color actual.
int	**getClipHeight**() Obtiene la altura del área recortable.
int	**getClipWidth**() Obtiene la anchura del área recortable actual.
int	**getClipX**() Obtiene el desplazamiento de la zona de recorte actual X, en relación con el origen del sistema de coordenadas de este contexto gráfico.
int	**getClipY**() Obtiene el desplazamiento de la zona de recorte actual Y, en relación con el origen del sistema de coordenadas de este contexto gráfico.
int	**getColor**() Obtiene el color actual.
int	**getDisplayColor**(int color) Obtiene el color que será desplegado si el color especificado es requerido requested.
Font	**getFont**() Obtiene la fuente actual.
int	**getGrayScale**() Obtiene el valor de la escala de grises actual del color que está siendo utilizado para las operaciones de dibujo.
int	**getGreenComponent**() Obtiene el componente verde del color actual.
int	**getRedComponent**() Obtiene el componente rojo del color actual.
int	**gettrokeStyle**() Obtiene el estilo usado para las operaciones de dibujo.
int	**getTranslateX**() Obtiene la coordenada X del origen trasladado del contexto gráfico.
int	**getTranslateY**() Obtiene la coordenada Y del origen trasladado del contexto gráfico.

void	**setClip**(int x, int y, int width, int height) Establece el Clip actual para el rectángulo especificado por las coordenadas dadas.
void	**setColor**(int RGB) Establece el color actual para el valor especificado de RGB.
void	**setColor**(int red, int green, int blue) Establece el color actual para los valores especificados de RGB.
void	**setFont**(Font font) Establece la fuente de todas las operaciones subsecuentes de despliegue de texto.
void	**setGrayScale**(int value) Establece el sistema de escala de grises actual para ser usado para todas las operaciones subsecuentes de formación de imágenes.
void	**SettrokeStyle**(int style) Establece el estilo usado para dibujar líneas, arcos y rectángulos y rectángulos redondeados.
void	**translate**(int x, int y) Traslada el origen del contexto gráfico del punto (x, y) en el sistema de coordenadas actual.

Métodos de la clase Graphics (Sing y Knudsen 2005, p.404).

Screen

public abstract class **Screen** extends Displayable (Oracle midp 2.0,2014)

Es la superclase común de todas las clases de interfaz de usuario de alto nivel, los contenidos que se muestran y su interacción con el usuario se definen por las subclases Form, Alert, List,TextBox.

Se usan los métodos definidos de la subclase, con los cuales la aplicación puede cambiar el contenido de un objeto Screen, mientras se muestra al usuario, si esto ocurre, y el objeto Screen es visible, la pantalla se **actualizará** automáticamente. Es decir, la aplicación actualizará la pantalla en el momento oportuno sin esperar ninguna nueva acción de la aplicación. Por **ejemplo**, en el caso de que un objeto de **List** se muestre actualmente, y cada elemento de la lista es **visible**. Si la aplicación **inserta** un **nuevo** elemento al principio de la lista, se muestra de inmediato, y los otros elementos se reordenarán adecuadamente, no hay necesidad de que la aplicación llame a otro método para actualizar la pantalla.

En el MIDP 2.0 los cuatro métodos definidos de Screen de lectura /escritura del **ticker** y las propiedades fueron trasladados a la superclase Displayable, la semántica de estos métodos no han cambiado.

Al ser una súper clase derivada de Displayable, sus métodos son:

Métodos heredados de la clase javax.microedition.lcdui.Displayable

```
addCommand, getHeight, getTicker, getTitle, getWidth, isShown,
removeCommand, setCommandListener, setTicker, setTitle,
sizeChanged
```

Los métodos de la clase Screen (Sing y Knudsen 2005, p.408)

Ticker

public class **Ticker** extends Object (Oracle midp 2.0,2014)

La clase Ticker es un objeto que proporciona el desplazamiento de texto en la parte superior de la pantalla. Un Ticker se asocia con el display, no con screen. Se puede colocar un Ticker en una pantalla utilizando el método Screen.setTicker (Ticker t).

Ejemplo:

```
//Se declara la variable del tipo Ticker
private Ticker t;
//Se crea el ticker con el texto a desplegar
        t = new Ticker("Este es un ejemplo del uso del Ticker");
//Se establece el Ticker en la pantalla
        setTicker(t);
```

Uso del Ticker

Los métodos de la clase **Ticker**, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método
String	**Obtienetring**() Obtiene la cadena actual que está siendo colocada por el ticker.
void	**Establece tring**(String str) Establece la cadena para ser desplegada por el ticker.

Los métodos de la clase Ticker (Sing y Knudsen 2005, p.410)

TextBox

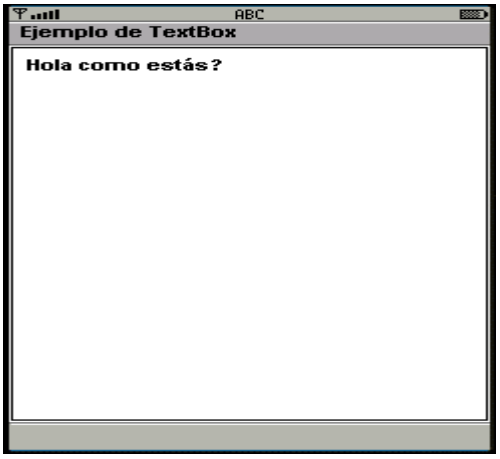
public class **TextBox** extends Screen (Oracle midp 2.0,2014)

La clase TextBox es un Screen que permite al usuario introducir y editar texto, tiene un tamaño máximo, que es el número máximo de caracteres que se pueden almacenar en el objeto en cualquier momento (según la capacidad especificada –ver ejemplo de Textbox-) Este límite se aplica cuando se construye la instancia TextBox, cuando el usuario está editando texto dentro del cuadro de texto, así como cuando el programa de aplicación llama a métodos en el cuadro de texto que modifica su contenido. El tamaño máximo es la capacidad máxima almacenada y no está relacionada con el número de caracteres que se pueden visualizar en cualquier momento dado. El número de caracteres de la imagen y su disposición en filas y columnas se determinan por el dispositivo.

Ejemplo:

```
//Se crea un objeto TextBox  con título y mensaje
  TextBox t = new textBox("Ejemplo de TextBox",
   "Hola como estás?", 256, 0); //se especifica el tamaño máximo de caracteres -256
  display.setCurrent(t);
```

Y el resultado es:



Ejemplo de TextBox

Los métodos de la clase **TextBox**, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método
void	**delete**(int offset, int length) Borra los caracteres desde TextBox.
int	**getCaretPosition**()

	Obtiene la posición actual de la entrada.
int	**getChars**(char[] data) Copia el contenido de TextBox en un arreglo de caracteres comenzando en el índice cero.
int	**getConstraints**() Obtiene la entrada actual de las limitaciones del TextBox.
int	**getMaxSize**() Regresa el tamaño máximo (número de caracteres) que puede ser almacenado en el TextBox.
String	**Obtienetring**() Obtiene el contenido del TextBox como una cadena de valor.
void	**insert**(char[] data, int offset, int length, int position) Inserta un subrango de caracteres en el contenido de TextBox.
void	**insert**(String src, int position) Inserta una cadena en el contenido del TextBox.
void	**setChars**(char[] data, int offset, int length) Establece el contendio del TextBox de un arreglo de caracteres, remplazando el contenido previo.
void	**setConstraints**(int constraints) Establece la entrada de restricciones del TextBox.
void	**setInitialInputMode**(String characterSubset) Establece un tipo de entrada que debería ser usada cuando el usuario inicialice le edición del TextBox.
int	**setMaxSize**(int maxSize) Establece el tamaño máximo (número de caracteres) que pueden ser contenidas en el TextBox.
void	**setString**(String text) Establece el contenido del TextBox como una cadena de valor, reemplazando el contenido previo.
void	**setTicker**(Ticker ticker) Establece un ticker para usarlo con el Displayable, reemplazando cualquier ticker previo.
void	**setTitle**(String s) Establece el título del Displayable.
int	**size**() Establece el número de caracteres que están actualmente almacenadas en el TextBox

Los métodos de la clase TextBox (Sing y Knudsen 2005, p.409)

Alert

public class **Alert** extends Screen (Oracle midp 2.0,2014)

Una alerta es una pantalla que muestra mensajes al usuario y espera durante un cierto período de tiempo antes de proceder al siguiente `Displayable,` puede contener una cadena de texto y una imagen. El uso de Alert es informar al usuario acerca de los errores y otras condiciones excepcionales.

La aplicación puede establecer el tiempo de alerta para ser infinita con **`setTimeout`** `(Alert.FOREVER)` en cuyo caso la alerta se considera que es modal y la aplicación proporciona una característica que permite al usuario "despedir" la alerta, con lo cual se muestra el siguiente displayable como si el tiempo de espera hubiera expirado inmediatamente.

Una alerta puede tener un `AlertType` asociada con ella, para proporcionar una indicación de la naturaleza de la alerta, la aplicación puede utilizar este tipo para reproducir un sonido apropiado cuando la alerta se presenta al usuario, ver `AlertType.playSound ().`

Una alerta puede contener una imagen opcional, la imagen puede ser mutable o inmutable.

Ejemplo:

```
// Se declara la variable de tipo Alert con valor nulo
private Alert myAlert = null
// Se crea el objeto de tipo Alert con un título
myAlert = new Alert("MIDlet Alert")
// Se agrega una cadena de caracteres a alertString
String[] alertString = { " Alert String" }
myAlert.setTimeout(Alert.FOREVER)
```

Y el resultado es:

Ejemplo de Alert

Los métodos de la clase **Alert**, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método
void	**addCommand**(Command cmd) Similar a Displayable.addCommand(javax.microedition.lcdui.Command), cuando la aplicación agrega la primera vez un comando para un Alert, el DISMISS_COMMAND , se elimina automáticamente.
int	**getDefaultTimeout**() Obtiene la hora por defecto para mostrar en el objeto Alert.
Image	**getImage**() Obtiene la Image usada en el objeto Alert.
Gauge	**getIndicator**() Obtiene el indicador de actividad para el objeto Alert.
String	**Obtienetring**() Obtiene la cadena de texto usada en el objeto Alert.
int	**getTimeout**() Obtiene la hora del objeto Alert a mostrar.
AlertType	**getType**() Obtiene el tipo del objeto Alert.
void	**removeCommand**(Command cmd) Similar a Displayable.removeCommand(javax.microedition.lcdui.Command), sin embargo cuando la aplicación remueve el último comando de un objeto Alert, un DISMISS_COMMAND está implícitamente agregado.
void	**setCommandListener**(CommandListener l) Lo mismo a Displayable.setCommandListener(javax.microedition.lcdui.CommandListener) pero con semántica adicional.
void	**setImage**(Image img) Establece el objeto Image usado en el objeto Alert.
void	**setIndicator**(Gauge indicator) Establece un indicador de actividad para el objeto Alert.
void	**setString**(String str) Establece la cadena de texto usada en el objeto Alert.
void	**setTimeout**(int time) Establece la hora para el cual el objeto Alert mostrará.
void	**setType**(AlertType type) Establece el tipo del objeto Alert.

Los métodos de la clase Alert (Sing y Knudsen 2005, p.395)

List

public class **List** extends Screen implements Choice (Oracle midp 2.0,2014)

Es una **Screen** que contiene una lista de opciones, la mayor parte de su comportamiento es común con la clase `ChoiceGroup` y su API. Los diferentes tipos de lista, en particular, se definen en la interfaz **Choice.** Cuando una lista está presente en la pantalla, el usuario puede interactuar con ella mediante la selección de elementos y desplazarse entre ellos.

La noción de una operación de selección en un elemento de la lista es central para la interacción del usuario con la lista. En los dispositivos que tienen una tecla dedicada "seleccionar" o "ir", la operación de selección se lleva a cabo con esa tecla. Los dispositivos que no tienen una tecla dedicada deben proporcionar otro medio para hacer la operación de selección, por ejemplo, usando una tecla de función.

Ejemplo:

```
// Declaración del objeto ejemploList
    private List ejemploList;
//Creación del objeto List
private void createList() {
        ejemploList = new List("Selecciona ejemplo", List.IMPLICIT);
        for (int i = 0; i < ejemplo.length; i++) {
            ejemploList.append(ejemplo[i], null);
        }
```

Y el resultado es:

Ejemplo de List

Los métodos de la clase **List**, se exponen a continuación:

Tipo que devuelve el método	*Método y uso del método
int	**append**(String stringPart, Image imagePart) Agrega un elemento en List.
void	**delete**(int elementNum) Borra el elemento referenciado por elementNum.
void	**deleteAll**() Borra todos los elementos de la lista.
int	**getFitPolicy**() Obtiene las políticas preferidas de la aplicación para adecuar el contenido del elemento al espacio de la pantalla.
Font	**getFont**(int elementNum) Obtiene las fuentes preferidas de la aplicación para dibujar el elemento especificado por Choice.
Image	**getImage**(int elementNum) Obtiene el objeto Image como parte del elemento referenciado por elementNum.
int	**getelectedFlags**(boolean[] selectedArray_return) Consulta el estado del objeto List y regresa el estado de todos los elementos en un arreglo booleano selectedArray_return.
int	**getelectedIndex**() Regresa el índice de un elemento en el objeto List seleccionado.
String	**getString**(int elementNum) Obtiene la parte de String del elemento referenciado por elementNum.
void	**insert**(int elementNum, String stringPart, Image imagePart) Inserta un elemento en el objeto List, priorizando el elemento especificado.
boolean	**isSelected**(int elementNum) Obtiene un valor booleano indicando que el elemento está seleccionado.
void	**removeCommand**(Command cmd) Lo mismo que Displayable.removeCommand pero con semántica adicional.
void	**set**(int elementNum, String stringPart, Image imagePart) Establece el String y la Image como partes del elemento referenciado por elementNum, reemplazando el contenido previo del elemento.
void	**setFitPolicy**(int fitPolicy) Establece las políticas preferidas de la aplicación para el montaje del contenido de elementos Choice para el espacio de pantalla disponible.

void	**setFont**(int elementNum, Font font) Establece la fuente preferida de la aplicación para dibujar el elemento especificado de Choice.
void	**setSelectCommand**(Command command) Establece el Command a ser usado mediante una acción de selección IMPLICIT List.
void	**setSelectedFlags**(boolean[] selectedArray) Establece el estado seleccionado de todos los elementos de List.
void	**setSelectedIndex**(int elementNum, boolean selected) Establece el estado seleccionado de un elemento.
void	**setTicker**(Ticker ticker) Establece un ticker para usar con el Displayable, reemplazando cualquier ticker previo.
void	**setTitle**(String s) Establece el título del Displayable.
int	**size**() Obtiene el número de elementos en List.

Los métodos de la clase List (Sing y Knudsen 2005, p.407)

Form

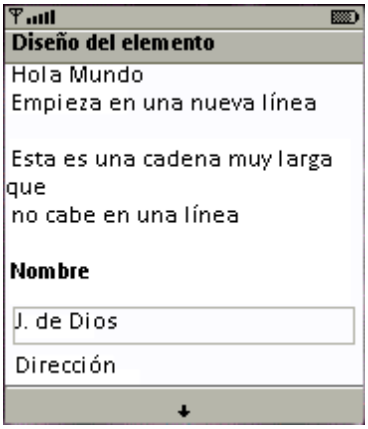
public class **Form** extends Screen (Oracle midp 2.0,2014)

Un formulario es un **Screen** que contiene una **mezcla arbitraria** de los **objetos**: imágenes, campos de texto de solo lectura, campos de texto editables, campos de fecha editables, gauges, grupos de selección, y elementos personalizados, en general, cualquier subclase de la clase de Item puede estar contenida dentro de un formulario, la aplicación se encarga de la disposición, de recorrido, y el desplazamiento en el formulario.

Ejemplo:

```
// Se crea el objeto form
Form form = new Form("Diseño del elemento");
// Se agregan elementos al formulario
form.append("Hola");
form.append("Mundo");
form.append("\nEmpieza  en una nueva línea\n");
form.append("Esta es una cadena muy larga que no cabe en una línea");
form.append(new TextField("Nombre", "J. de Dios", 32, TextField.ANY));
form.append("Dirección");
form.append(new TextField(null, null, 32, TextField.ANY));
//Despliega el formulario
 display.setCurrent(form);
```

Y el resultado es:



Ejemplo de form

Los métodos de la clase **Form**, se exponen a continuación:

Tipo que devuelve el método	*Método y uso del método
int	**append**(Image img) Agrega un elemento tipo `Image` al `Form`.
int	**append**(Item item) Agrega un `Item` al `Form`.
int	**append**(String str) Agrega un item tipo `String` al `Form`.
void	**delete**(int itemNum) Borra el `Item` referenciado por `itemNum`.
void	**deleteAll**() Borra todos los elementos del `Form`, dejando cero elementos.
Item	**get**(int itemNum) Obtiene el elemento en una posición dada.
int	**getHeight**() Regresa la altura en pixeles del área desplegable disponible para los elementos.
int	**getWidth**() Regresa la anchura en pixeles del área desplegable disponible para los elementos.
void	**insert**(int itemNum, Item item) Inserta un elemento en el Form priorizando el elemento especificado.
void	**set**(int itemNum, Item item) Establece el elemento referenciado por itemNum para especificar el elemento reemplazando cualquier elemento previo.
void	**setItemStateListener**(ItemStateListener iListener)

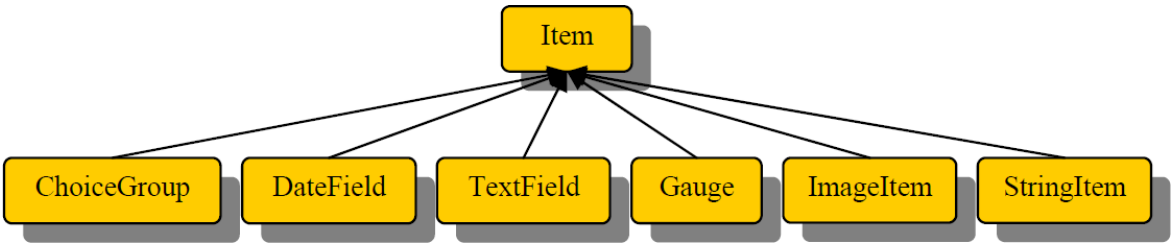
	Establece el `ItemStateListener` para el `Form`, reemplazando cualquier `ItemStateListener` previo.
`int`	**`size`**`()` Obtiene el tamaño de los elementos en el `Form`.

Los métodos de la clase Form (Sing y Knudsen 2005, p.403)

Item

public abstract class **Item** extends Object (Oracle midp 2.0,2014)

Item es una superclase para los componentes que se agregan a un formulario. Todos los objetos tipo item tienen un **campo etiqueta**, que es una cadena que se adjunta al ítem, la etiqueta normalmente se muestra cerca el componente en la pantalla. La etiqueta debe colocarse en la misma fila horizontal que el item o directamente sobre él. La aplicación debe intentar **distinguir cadenas** de etiquetas de otro tipo de contenido textual, posiblemente mediante la visualización de la etiqueta en una fuente diferente, alineación con un margen diferente, o añadiendo dos puntos a la misma si se coloca en la misma línea que otro contenido de la cadena. Si la pantalla es desplegable, la aplicación debe tratar de **mantener** la **etiqueta visible** al mismo tiempo que el item.

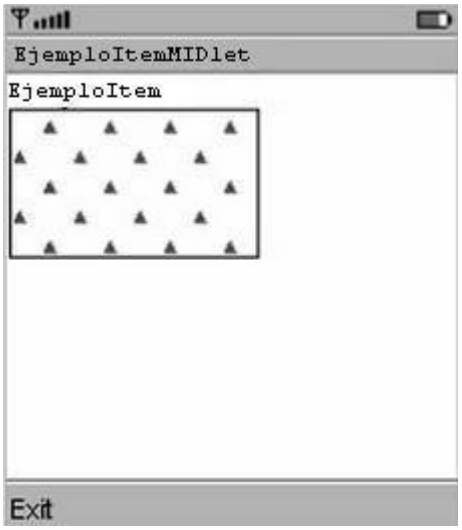


Jerarquía de clases de la clase Item (Gálvez y Ortega, 2011,p 52)

Ejemplo:

```
//se crea un formulario para contener items
Form form = new Form("EjemploItemMIDlet");
//Se agrega el objeto EjemploItem al formulario
form.append(new EjemploItem("EjemploItem"));
//Despliega el formulario
Display.getDisplay(this).setCurrent(form);
```

Y el resultado es:



Ejemplo de uso de Item

Los métodos de la clase **Item**, se exponen a continuación:

Tipo que devuelve el método	*Método y uso del método
void	**addCommand**(Command cmd) Agrega un contexto sensitivo Command al item.
String	**getLabel**() Obtiene la etiqueta de este objeto Item.
int	**getLayout**() Obtiene la disposición de las directivas usadas para colocar el elemento.
int	**getMinimumHeight**() Obtiene la altura mínima para este Item.
int	**getMinimumWidth**() Obtiene la anchura mínima para este Item.
int	**getPreferredHeight**() Obtiene la altura preferida de este Item.
int	**getPreferredWidth**() Obtiene la anchura preferida de este Item.
void	**notifyStateChanged**() Causa que el contexto notifique el cambio de estado del Item ItemStateListener .
void	**removeCommand**(Command cmd) Remueve el contexto sensitivo del command del item.
void	**setDefaultCommand**(Command cmd) Establece el Command por defecto para este Item.
void	**setItemCommandListener**(ItemCommandListener l)

	Establece un oyente para `Commands` en este `Item`, reemplazando cualquier otro establecido previamente `ItemCommandListener`.
`void`	**`setLabel`**`(String label)` Establece la etiqueta para este `Item`.
`void`	**`setLayout`**`(int layout)` Establece la disposición de las directivas para este `item`.
`void`	**`setPreferredSize`**`(int width, int height)` Establece el ancho y alto preferido para este `Item`.

Los métodos de la clase Item (Sing y Knudsen 2005, p.406)

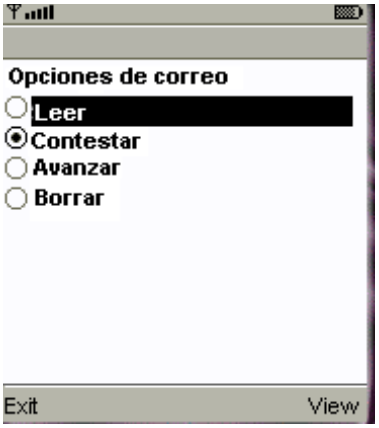
ChoiceGroup

public class **ChoiceGroup** extends Item implements Choice (Oracle midp 2.0,2014)
ChoiceGroup es una clase que incluye un grupo de elementos seleccionables destinados a ser colocados en un formulario. El grupo puede ser creado con una modalidad que requiera una única opción o múltiples opciones. La aplicación es responsable de proporcionar la representación gráfica de estos modos y debe proporcionar visualmente diferentes gráficos para los diferentes modos. Por ejemplo, podría utilizar **botones de radio** para el modo de elección individual y **casillas de verificación** para el modo de elección múltiple.

Ejemplo:

```
// declaración de la variable Choice group
private ChoiceGroup cgEmail;
// Creación del objeto choice group exclusivo (radio)
  cgEmail = new ChoiceGroup("Opciones de correo",
Choice.EXCLUSIVE);
// Agrega opciones
 cgEmail.append("Leer", null);
 replyIndex = cgEmail.append("Contestar", null);
 cgEmail.append("Avanza", null);
 cgEmail.append("Borrar", null);
```

Y el resultado es:



Ejemplo de ChoiceGroup

Los métodos de la clase **ChoiceGroup**, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método
int	**append**(String stringPart, Image imagePart) Agrega un elemento al ChoiceGroup.
void	**delete**(int elementNum) Borra el elemento referenciado por elementNum.
void	**deleteAll**() Borra todos los elementos de este ChoiceGroup.
int	**getFitPolicy**() Obtiene las políticas preferidas para montar el elemento Choice en el espacio de la pantalla disponible.
Font	**getFont**(int elementNum) Obtiene la fuente preferida de la aplicación para representar el elemento especificado de este Choice.
Image	**getImage**(int elementNum) Obtiene la parte de Image del elemento referenciado por elementNum.
int	**getSelectedFlags**(boolean[] selectedArray_return) Obtiene las banderas seleccionadas del ChoiceGroup y regresa el estado de todos los elementos en el arreglo booleano. selectedArray_return.
int	**getSelectedIndex**() Regresa el número indexado de un elemento en ChoiceGroup que está seleccionado.
String	**getString**(int elementNum) Obtiene la parte String del elemento referenciado por elementNum.
void	**insert**(int elementNum, String stringPart,

	Image imagePart) Inserta un elemento en ChoiceGroup priorizando el elemento especificado.
boolean	**isSelected**(int elementNum) Obtiene un valor booleano del elemento seleccionado.
void	**set**(int elementNum, String stringPart, Image imagePart) Establece las partes de string e Image del elemento referenciado por elementNum, reemplazando el contenido previo del elemento.
void	**setFitPolicy**(int fitPolicy) Establece las preferencias de la aplicación para ajustar el contenido del elemento Choice, para el espacio disponible de la pantalla.
void	**setFont**(int elementNum, Font font) Establece las fuentes preferidas de la aplicación para generar un elemento especificado del Choice.
void	**setSelectedFlags**(boolean[] selectedArray) Intenta establecer el estado seleccionado de cada elemento en ChoiceGroup.
void	**setSelectedIndex**(int elementNum, boolean selected) Para los objetos multiples de ChoiceGroup, se establece el estado de un elemento individual.
int	**size**() Regresa el número de elementos en el ChoiceGroup.

Los métodos de la clase ChoiceGroup (Sing y Knudsen 2005, p.398)

DateField

public class **DateField** extends Item (Oracle midp 2.0,2014)

Un DateField es un **componente editable** para presentar la información de la fecha y la hora (calendario) que se coloca en un formulario, el valor para este campo se puede establecer o no inicialmente. Si el valor no se establece entonces la UI para el campo lo muestra y no es un valor válido y el resultado de la funcion getDate () es nulo.

La instancia de DateField puede ser configurada para aceptar la información de fecha u hora o ambos. Este modo de configuración de entrada se realiza con DATE, TIME o DATE_TIME con campos estáticos de esta clase. Modo de entrada DATE permite fijar sólo información de fecha y TIME sólo información de tiempo (horas, minutos). DATE_TIME permite configurar tanto la hora como los valores de fecha.

Ejemplo:

```
//Se crean los objetos date, time y ambos
private DateField dateDateField = new DateField("fecha:", DateField.DATE);
private DateField timeDateField = new DateField("hora:", DateField.TIME);
```

```
private DateField bothDateField = new DateField("ambos:", DateField.DATE_TIME);
public DateFieldMIDlet() {
    //despliega el contenido
    display = Display.getDisplay(this);
    //crea el objeto now para obtener el valor actual
    java.util.Date now = new java.util.Date();
   //obtiene los valores actuales de fecha, hora y ambos
    dateDateField.setDate(now);
    timeDateField.setDate(now);
    bothDateField.setDate(now);
  }
```

Y el resultado es:

Ejemplo de DateField

Los métodos de la clase **DateField**, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método
Date	**getDate**() Regresa la fecha del campo.
int	**getInputMode**() Obtiene el modo entrada del campo fecha.
void	**setDate**(Date date) Establece un nuevo valor para este campo.
void	**setInputMode**(int mode) Establece el modo entrada para el campo fecha.

Los métodos de la clase DateField (Sing y Knudsen 2005, p.400)

TextField

public class **TextField** extends Item (Oracle midp 2.0,2014)

Un TextField es un **componente** de **texto editable** que se puede colocar en un formulario, este puede ser un texto que se utiliza como el valor inicial, tiene un tamaño máximo, el cual es el número máximo de caracteres que se pueden almacenar en el objeto en cualquier momento, este límite se aplica cuando se construye la instancia TextField, cuando el usuario está editando texto dentro del TextField, así como cuando el programa de aplicación llama a métodos en el TextField que modifica su contenido. El tamaño máximo es la capacidad máxima almacenada y no está relacionado con el número de caracteres que se pueden visualizar en cualquier momento dado.

Ejemplo:

```
// Se crea el objeto textfield
private TextField textfield = new TextField("Nombre:", "", 30,
TextField.ANY);
//Se implementa el método para capturar el nombre
  public TextFieldCapture() {
    display = Display.getDisplay(this);
    //se agrega el objeto al formulario
    form.append(textfield);
  }
```

Y el resultado es:

Ejemplo de TextField

La implementación puede poner un límite en el tamaño máximo, y el tamaño máximo asignado, en realidad puede ser menor que el que solicite la aplicación. El valor asignado en realidad se reflejará en el valor devuelto por el método `getMaxSize()`.

Los métodos de la clase **TextField**, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método

void	**delete**(int offset, int length) Borra los caracteres del TextField.
int	**getCaretPosition**() Obtiene la posición de la entrada actual.
int	**getChars**(char[] data) Copia el contenido de TextField en un arreglo de carateres empezando en el índice cero.
int	**getConstraints**() Obtiene las restricciones de la entrada actual de TextField.
int	**getMaxSize**() Regresa el tamaño máximo (número de caracteres) que pueden ser almacenadas en el objeto TextField.
String	**getString**() Obtiene el contenido del TextField como una cadena de valor.
void	**insert**(char[] data, int offset, int length, int position) Inserta un subrango de un arreglo de caracteres dentro del contenido del TextField.
void	**insert**(String src, int position) Inserta una cadena en el contenido del TextField.
void	**setChars**(char[] data, int offset, int length) Establece el contenido del TextField de un arreglo de caracteres, reemplazando el contenido previo.
void	**setConstraints**(int constraints) Establece la entrada de restricciones del objeto TextField.
void	**setInitialInputMode**(String characterSubset) Establece la implementación oculta en modo entrada que se utiliza cuando el usuario inicia la edición del TextField.
int	**setMaxSize**(int maxSize) Establece el máximo tamaño (número de caracteres) que pueden ser contenidos en el TextField.
void	**setString**(String text) Establece el contenido en TextField como una cadena de valor reemplazando el contenido previo.
int	**size**() Obtiene el número de caracteres almacenados actualmente en TextField.

Los métodos de la clase TextField (Sing y Knudsen 2005, p.410)

Gauge

public class **Gauge** extends Item (Oracle midp 2.0,2014)

Es una clase que implementa una pantalla gráfica, tal como un gráfico de barras, de un valor entero. El Gauge contiene un **valor** que se encuentra entre **cero** y un **valor máximo**, la aplicación puede controlar el **valor actual** y el **valor máximo**. El rango de valores especificados por la aplicación puede ser mayor que el número de estados visuales distintos posibles en el dispositivo, por lo que más de un valor puede tener la misma representación visual.

Por ejemplo, considerando un objeto Gauge que tiene un rango de valores de cero a 99, y se ejecuta en un dispositivo que muestra el valor aproximado del Gauge utilizando un conjunto de una a diez barras. El dispositivo podría mostrar una barra para valores cero al nueve, dos barras para los valores de diez al 19, tres barras para los valores de 20 a 29, y así sucesivamente.

Ejemplo:

```
//Se declara la variable de tipo Gauge
private Gauge gauge;
//Implementación del método GaugeInteractive
public J2MEGaugeInteractive() {
display = Display.getDisplay(this);
//Se crea el objeto de tipo Gauge
 gauge = new Gauge("Evalúa la película: ", true, 5, 1);
//Se agrega el formulario
form.append(gauge);
```

Y el resultado es:

Ejemplo de Gauge

Un Gauge puede ser interactivo o no interactivo, las aplicaciones pueden establecer o recuperar el valor del Gauge en cualquier momento. Independientemente del modo de interacción, también la aplicación puede cambiar la apariencia visual del gráfico de barras solo si el Gauge se creó en modo interactivo.

Los métodos de la clase **Gauge**, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método
void	**addCommand**(Command cmd) Agrega un contexto sensitivo, tipo Command al elemento.
int	**getMaxValue**() Obtiene el valor máximo del objeto Gauge.
int	**getValue**() Obtiene el valor actual del objeto Gauge.
boolean	**isInteractive**() Indica si el usuario permitió el cambio del valor de Gauge.
void	**setDefaultCommand**(Command cmd) Establece el Command por default para este Item.
void	**setItemCommandListener**(ItemCommandListener l) Establece un oyente para el elemento, reemplazando cualquier ItemCommandListener previo.
void	**setLabel**(String label) Establece una etiqueta para el Item.
void	**setLayout**(int layout) Establece las directrices de diseño para este elemento.
void	**setMaxValue**(int maxValue) Establece el valor máximo del objeto Gauge.

void	**setPreferredSize**(int width, int height) Establece el ancho preferido y la altura para este Item.
void	**setValue**(int value) Establece el valor actual del objeto Gauge.

Los métodos de la clase Gauge (Sing y Knudsen 2005, p.403)

ImageItem

public class **ImageItem** extends Item (Oracle midp 2.0,2014)

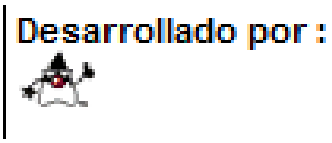
Es un elemento que puede **contener** una **imagen**.

Cada objeto `ImageItem` contiene una referencia a un objeto imagen. Esta imagen puede ser mutable o inmutable. El valor null se puede especificar para el contenido de imagen de un `ImageItem`. Si esto ocurre (y si la etiqueta también es nula) `el ImageItem` no ocupará ningún espacio en la pantalla.

Ejemplo:

```
//Se crea un  objeto de ImageItem
imageItem= New ImageItem( "Desarrollado por : ",
Image.createImage("/duke.gif"),
ImageItem. LAYOUT_DEFAULT,"Duke");
```

Y el resultado es:



Uso del ImageItem

ImageItem contiene directrices de diseño que fueron definidos originalmente en **MIDP 1.0.** Estas directivas de diseño se han trasladado a la clase de **Item** y ahora se aplican a todos los elementos. Las declaraciones se dejan en `ImageItem` para fines de compatibilidad fuente.

El parámetro `altText` especifica una cadena que se mostrará en lugar de la imagen si la imagen excede la capacidad de la pantalla. El parámetro altText puede ser nulo.

Los métodos de la clase **ImageItem**, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método
int	**getAppearanceMode**() Regresa el modo apariencia de ImageItem.
Image	**getImage**() Obtiene la imagen contenida en ImageItem, o null si no hay imagen.
int	**getLayout**()

	Obtiene las directivas de diseño utilizados para la colocación de la imagen.
void	**setAltText**(String text) Establece el texto alternative de ImageItem, o nulo si el texto alternative no fue proporcionado.
void	**setImage**(Image img) Establece el objeto Image contenido en ImageItem.
void	**setLayout**(int layout) Establece las directrices del diseño .

Los métodos de la clase ImageItem (Sing y Knudsen 2005, p.406)

StringItem

public class **StringItem** extends Item (Oracle midp 2.0,2014)

Un item o elemento puede contener una cadena, un StringItem sólo **visualiza**; el usuario no puede editar el contenido. Tanto la etiqueta y el contenido textual de un StringItem pueden ser **modificados** por la aplicación. La representación visual de la etiqueta puede ser diferente de la de los contenidos textuales.

Por ejemplo

```
// El StringItem no es editable
stringItem=new StringItem ("Tu usuario: ", "WXP_890);
```

Y el resultado es :

Uso del StringItem

Los métodos de la clase **StringItem**, se exponen a continuación:

Tipo que devuelve el método	Método y uso del método
int	**getAppearanceMode**() Regresa el modo aparciencia de StringItem.
Font	**getFont**() Obtiene la fuente preferida de la aplicacón para dibujar el contenido de StringItem.
String	**getText**() Obtiene el texto contenido en el StringItem, o nulo si el StringItem está vacío.
void	**setFont**(Font font) Establece la fuente preferida por la aplicación para dibujar el StringItem.
void	**setPreferredSize**(int width, int height)

	Establece el ancho y altura preferidos para este `Item`.
void	**setText**(String text) Establece el texto contenido del `StringItem`.

Los métodos de la clase StringItem (Sing y Knudsen 2005, p.409)

2.3 Desarrollo de aplicaciones para dispositivos en J2ME

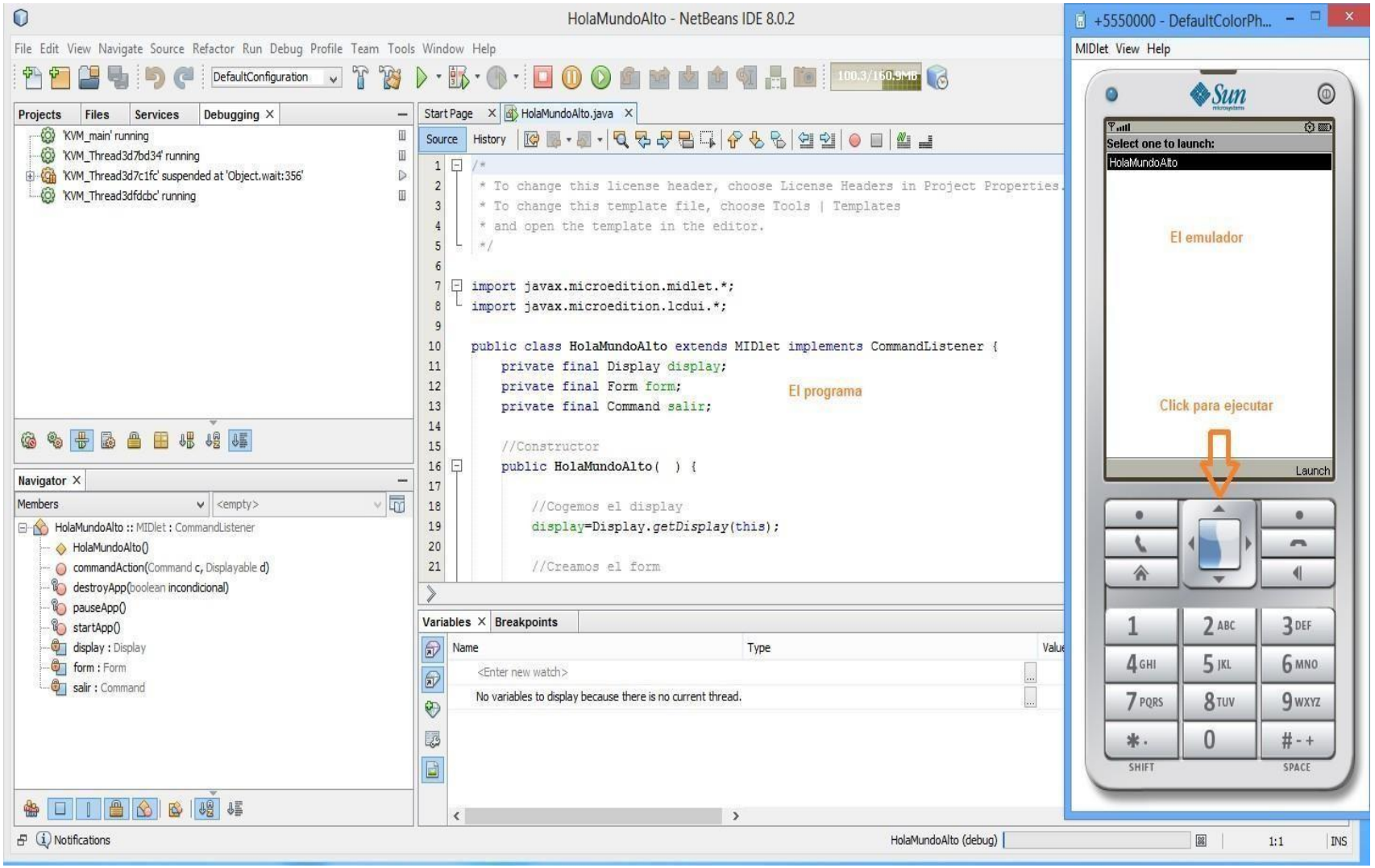
Después de revisar las APIs de alto y bajo nivel, sus respectivas clases y métodos, es importante probar su funcionamiento, el cual se va a desarrollar con el famoso programa hola mundo y con la herramienta Netbeans, es recomendable consultar el documento U3. *Instalando Netbeans*, en donde encontrarás una guía para la instalación del Java SE, Wireless Toolkit y Netbeans con sus respectivos plugins. Después de haber realizado todo el proceso ya estamos en condiciones de probar el hola mundo con la UI de alto nivel, cuyo código es el siguiente:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class HolaMundoAlto extends MIDlet implements
CommandListener {
    private final Display display;
    private final Form form;
    private final Command salir;
    //Constructor
    public HolaMundoAlto(  ) {
      //se toma el display
      display=Display.getDisplay(this);
       //Se crea el  formulario
        form = new Form ("Este es un ejemplo ");
        form.append("Hola Mundo\n");
      //Se crea el comando  salir
        salir=new Command("Salir",Command.EXIT, 3);
         // se añade el comando al formulario y se activa al
oyente
      form.addCommand(salir);
```

```
        form.setCommandListener(this);
    }
     //Método que se llama cuando se pasa de Pausado a Activo
    protected void startApp(  ) {
        display.setCurrent(form);
    }
    //Método que se llama cuando se pasa de Activo a Pausado
    protected void pauseApp(  ) {
    }
     //Método que se llama cuando se destruye el midlet
    protected void destroyApp(boolean incondicional) {
    }
    //Método para el tratamiento de datos de teclado
    public void commandAction(Command c, Displayable d) {
   //Se comprueba si se sale o muestra una alerta
      if (c ==salir) {
        destroyApp(true);
        notifyDestroyed();
        } else System.out.println("Otro comando pulsado");
           }
}
```

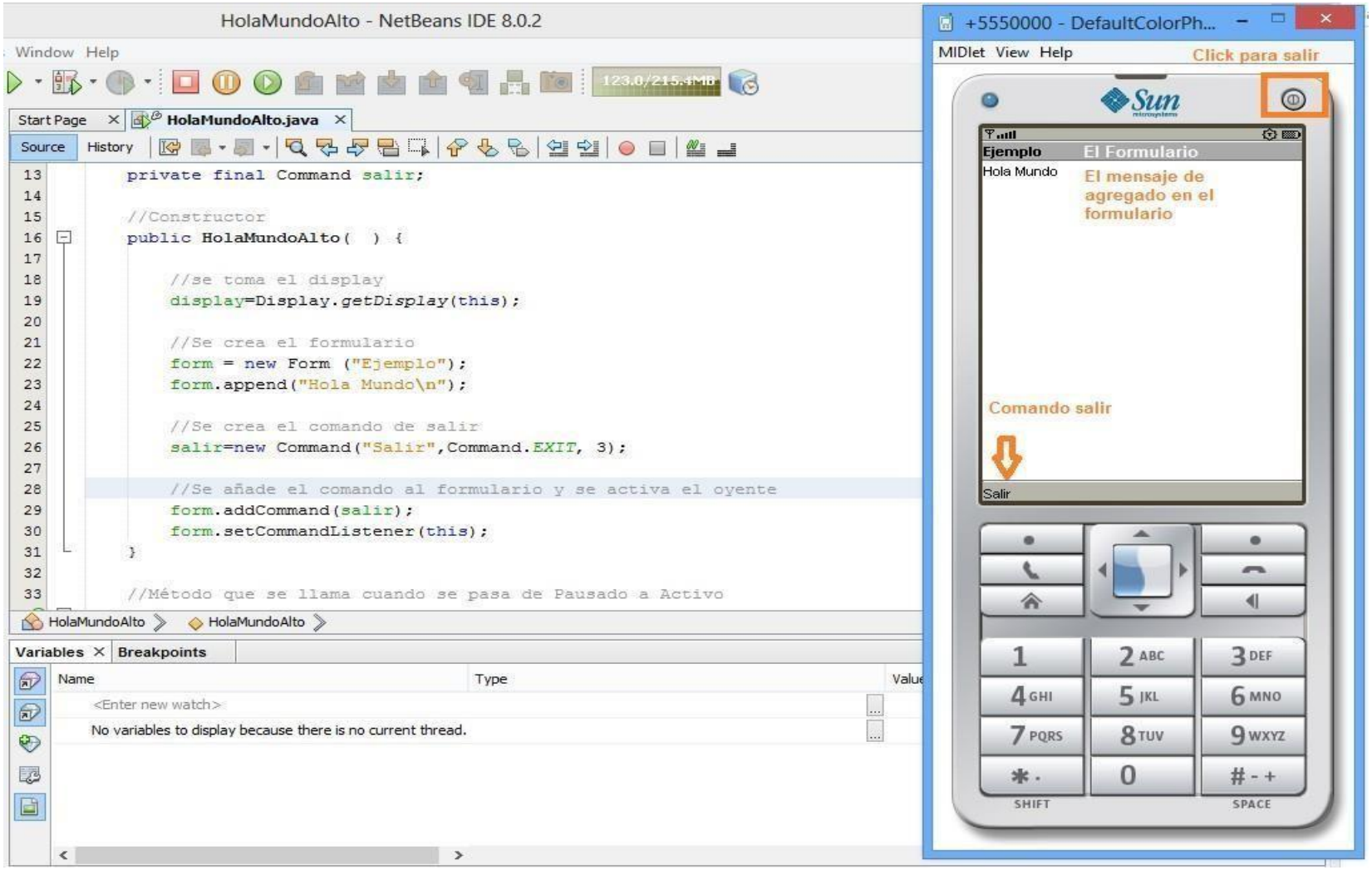
Codigo del MIDlet Hola Mundo con API de alto nivel, basado en http://leo.ugr.es/J2ME/MIDP/graficos.htm

Y se debe observar el emulador:



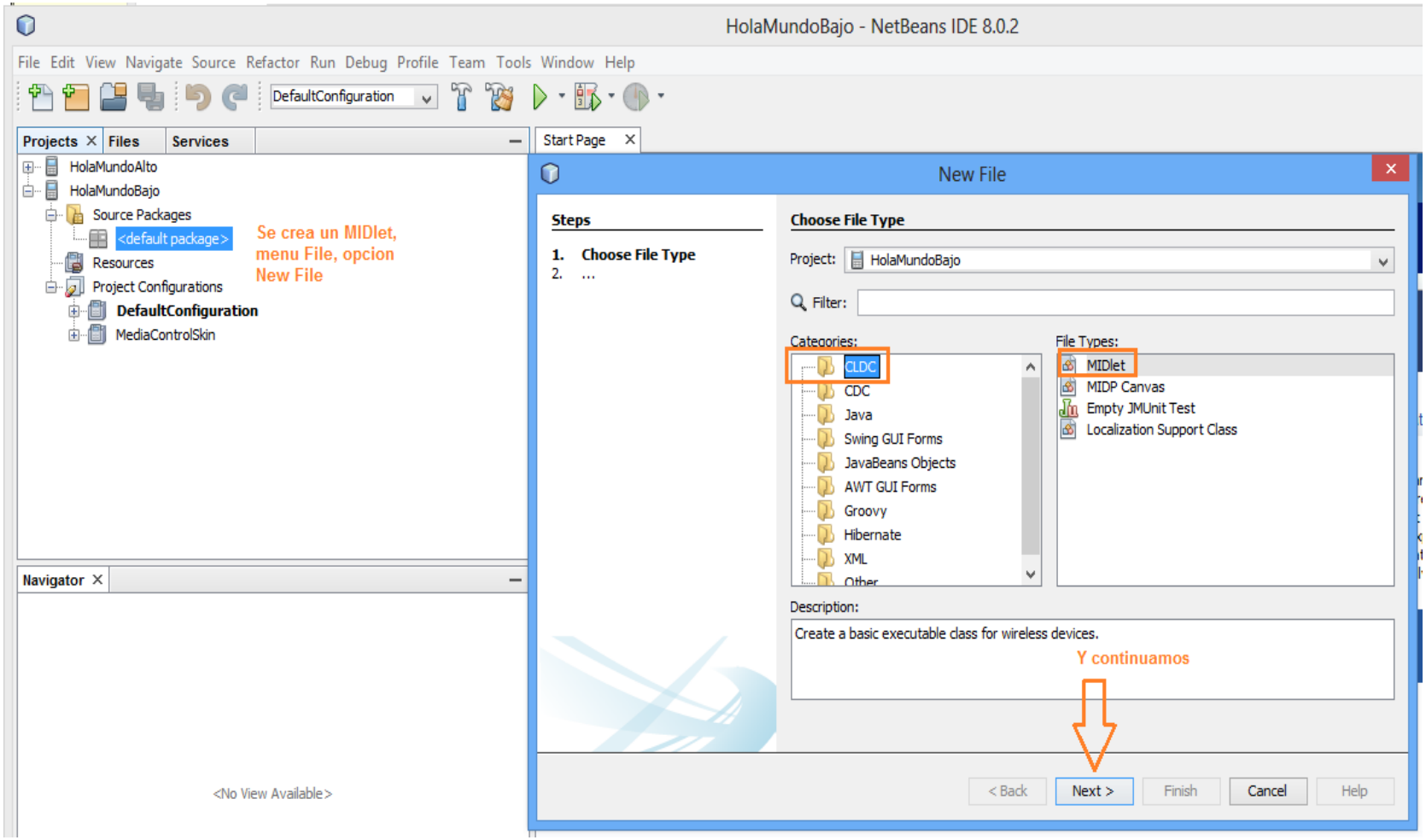
MIDlet y emulación del MIDlet

Y después de ejecutar el Midlet, observarás la siguiente pantalla:



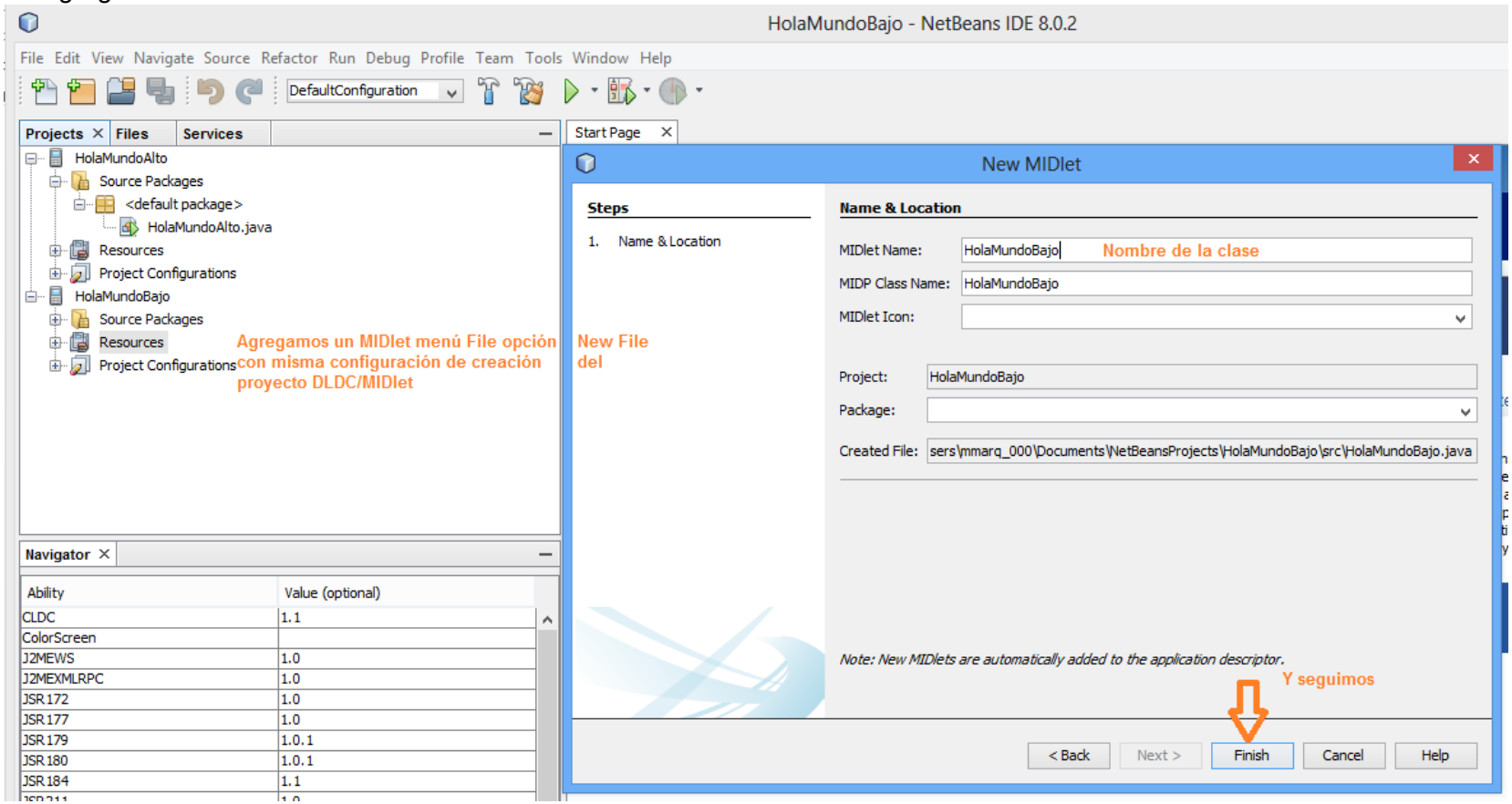
Identificación de elementos de la API de alto nivel

Observar que el comando salir no está asociado a ninguna tecla y por eso no es funcional. Se prueba ahora con la API de bajo nivel. Se crea un nuevo proyecto, de categoría CLDC y tipo de archivo MIDlet:



Creación del proyecto del uso de la API de bajo nivel

Se agrega un archivo MIDLet



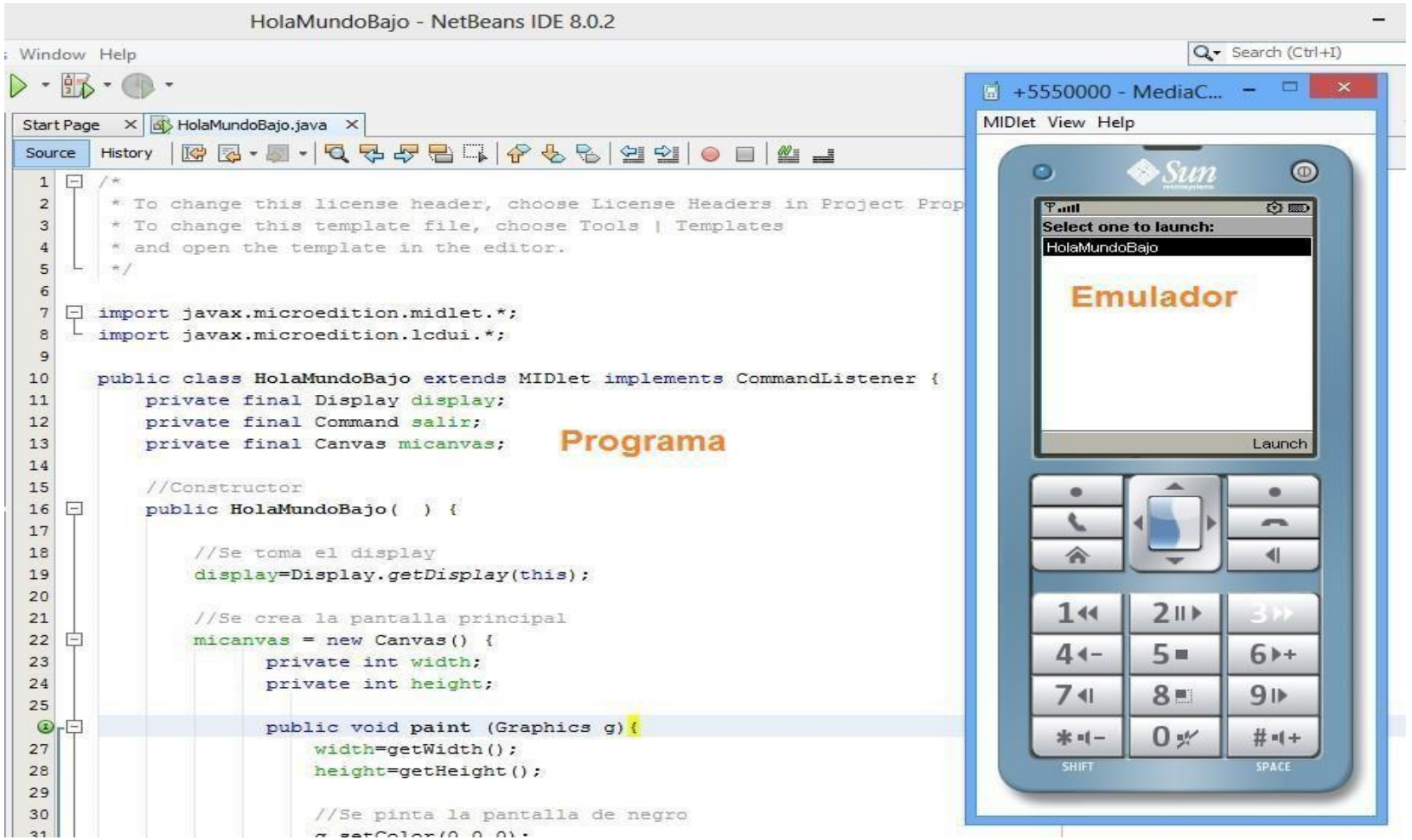
Creación del MIDlet HolaMundoBajo

Se sustituye el esqueleto del MIDlet por el siguiente código:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class HolaMundoBajo extends MIDlet implements
CommandListener {
    private final Display display;
    private final Command salir;
    private final Canvas micanvas;
    //Constructor
    public HolaMundoBajo(  ) {
      //Se toma el display
      display=Display.getDisplay(this);
      //Se crea la pantalla principal
      micanvas = new Canvas() {
            private int width;
            private int height;
            public void paint (Graphics g){
              width=getWidth();
              height=getHeight();
             //Se pinta la pantalla de negro
              g.setColor(0,0,0);
              g.fillRect(0,0,width,height);
              //Se dibuja el mensaje y en blanco
              g.setColor(255,255,255);
              g.setStrokeStyle(Graphics.SOLID);
              g.drawString("Hola en API de bajo
Nivel",width/2,height/2,(Graphics.BASELINE | Graphics.HCENTER));
            }//fin del método paint
```

```
        };
      //Se crea el comando  salir
            salir=new Command("Salir",Command.EXIT, 3);
             //Se añade el comando al Canvas y  se activa el oyente
      micanvas.addCommand(salir);
      micanvas.setCommandListener(this);
    }
     //Método que se llama cuando se pasa de Pausado a Activo
    protected void startApp(  ) {
        display.setCurrent(micanvas);
    }
     //Método que se llama cuando se pasa de Activo a Pausado
    protected void pauseApp(  ) {
    }
     //Método que se llama cuando se destruye el midlet
    protected void destroyApp(boolean incondicional) {
    }
    //Método para el tratamiento de datos de teclado
   public void commandAction(Command c, Displayable d) {
      //Verifica si se sale o muestra una alerta
      if (c ==salir) {
         destroyApp(true);
         notifyDestroyed();
         } else System.out.println("Otro comando pulsado");
   }
}
```

Código del MIDlet de HolaMundoBajo, basado en http://leo.ugr.es/J2ME/MIDP/graficos.htm
Se observará la siguiente pantalla :



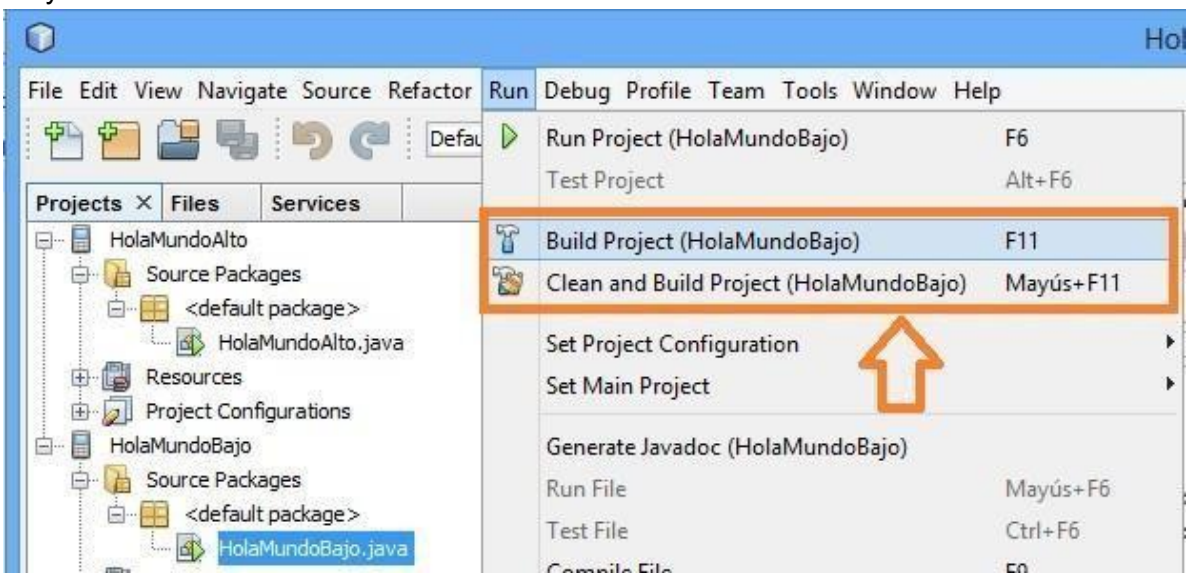
MIDlet y emulación del MIDlet

Y después de ejecutar el MIDlet se debe observar lo siguiente:

Identificación de elementos de la API de bajo nivel

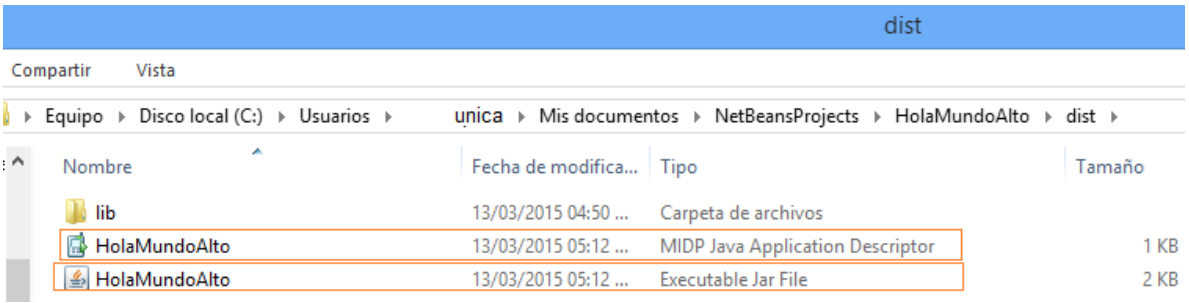
Ya se probó el uso de las dos APIs, sólo falta identificar los archivos JAR y JAD que son los que se copian en el teléfono, una vez que se da el visto bueno de que la aplicación es funcional.

Una vez que se compila, se corrige, se corre la aplicación y se está seguro que no hay errores, se construye el proyecto con la opción F11 o la opción limpiar y construir Mayús+F11.



Opciones para construir el proyecto

Los archivos JAD y JAR, se encuentran en el directorio **dist**, después de comprobar el buen funcionamiento de la aplicación lo que sigue es pasar estos archivos al teléfono móvil y probarlo, el procedimiento depende del dispositivo en cuestión, pero en general consiste en pasar estos archivos a una carpeta del dispostivo y ejecutar el archivo .**JAR. Por ejemplo:**



Archivos JAR y JAD

La diferencia entre las dos APIs en la programación se aprecian con estos dos ejemplos básicos en donde en el primer ejemplo se utiliza un formulario en donde es posible agregar elementos y en el segundo un lienzo donde es posible dibujar, ya dependerá de nuestras necesidades de usar una API o la otra. Con la instalación de dos ambientes de desarrollo, experimentado con uno de ellos, concocer las

interfaces gráficas de usuario para la progrmación de los MIDlets, lo que procede es probar la funcionalidad de los métodos en la programación de la propia aplicación.

En la siguiente y última unidad toca conocer el ambiente de desarrollo de Android y experimentar con ella.

Cierre de la unidad

¿Qué te ha parecido la experiencia de la programación con Netbeans? ¿Fácil? ¿Difícil? Como todo, solo es cuestión de práctica y mientras más practiques más utilidad le verás a esta rama del desarrollo de aplicaciones.

Se experimentó con la instalación de dos plataformas de desarrollo J2ME y Netbeans, de las cuales se selecciono trabajar con Netbeans por la facilidad y el ambiente amigable e intuitivo. Se señaló la importancia de las dos APIs con que trabaja en J2ME, la de alto nivel y la de bajo nivel, conociendo sus respectivas clases y métodos y se comprobó el uso de las APIs con dos ejemplos básicos, en los cuales se pudo observar la diferencia entre las interfaces de programación de aplicaciones.

La presente unidad deja una experiencia en la programación, la cual servirá para abordar la unidad 3 Aplicaciones en plataforma Android.

Para saber más

Es necesario que practiques y revises otras fuentes de consulta para ampliar tu conocimiento, se te sugieren los siguientes recursos:

- La documentación de las clase del MIDP 2.0

Oracle. (s. f.). *J2ME Reference Implementation — MIDP 2.0*.
http://docs.oracle.com/javame/config/cldc/ref-impl/midp2.0/jsr118/index-all.html

- Para documentación, soporte y videotutoriales acerca del uso de Netbeans y aplicaciones móviles.

Oracle. (s. f.). *NetBeans | Herramienta de desarrollo de aplicaciones*.
https://www.oracle.com/latam/application-development/netbeans/

- Ejemplos de aplicaciones en J2ME para experimentar las clases de la UIs.

Universidad Carlos III de Madrid. (s. f.). *Práctica introductoria a J2ME*.
https://www.it.uc3m.es/celeste/docencia/j2me/tutoriales/midp1_0//PracticaIntroJ2ME/

Fuentes de consulta básicas

Aranaz Tudela, J. (2009). *Desarrollo de aplicaciones móviles sobre la plataforma Android de Google*. Universidad Carlos III de Madrid, Escuela Politécnica Superior.

Cuello, J., & Vittone, J. (2013). *Diseñando apps para móviles* (pp. 20–32). José Vittone.

Gálvez, S., & Ortega, L. (2003). *Java a tope: J2ME (Java 2 Micro Edition)*. Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga.

Li, S., & Knudsen, J. (2005). *Beginning J2ME: From novice to professional* (3rd ed.). Apress.

Lionbridge. (n. d.). *Mobile web apps vs. mobile native apps: How to make the right choice*. https://www.lionbridge.com/

Oracle Corporation. (2014). *MIDP 2.0*. http://docs.oracle.com/javame/config/cldc/ref-impl/midp2.0/jsr118/index-all.html

Paredes Velasco, M., Santacruz Valencia, L., & Domínguez Mateos, F. (2012). *Programación multimedia y dispositivos móviles*. RA-MA.

Sun Microsystems. (2006a). *All classes*. https://docs.oracle.com/javame/config/cldc/ref-impl/midp2.0/jsr118/allclasses-noframe.html

Sun Microsystems. (2006b). *Class Display*. http://docs.oracle.com/javame/config/cldc/ref-impl/midp2.0/jsr118/javax/microedition/lcdui/Display.html

Sun Microsystems. (2006c). *Class Displayable*. http://docs.oracle.com/javame/config/cldc/ref-impl/midp2.0/jsr118/javax/microedition/lcdui/Displayable.html