



Manual de uso WebSphere Studio Device Developer 5.6

**Desarrollando Aplicaciones
J2ME**

**Florina Almenárez Mendoza
Celeste Campo Vázquez
Rubén Lagar Ferreiro**

Contenido

1. INTRODUCCIÓN.....	4
2. INSTALACIÓN.....	4
3. COMENZANDO CON WSDD	6
3.1. Proyecto J2ME	7
3.2. MIDlet Suite	9
4. ENTORNO DE DESARROLLO	11
4.1. Editor del código fuente.....	13
4.2. Usando Imágenes u otros recursos	15
4.3. Configurando RMS	17
4.4. Depuración de aplicaciones	18
5. UNA APLICACIÓN SENCILLA DE EJEMPLO: HOLA MUNDO	24
6. AÑADIENDO NUEVOS EMULADORES	28

Lista de Figuras

Figura 1. Bienvenida a la instalación del WSSD	4
Figura 2. Confirmación de parámetros de instalación	5
Figura 3. Proceso de copia de archivos	5
Figura 4. Fin de la Instalación	6
Figura 5. Bienvenida al WSSD	7
Figura 6. Selección de Librerías Java	8
Figura 7. Información detallada de implementaciones específicas	8
Figura 8. Configuración final del proyecto	9
Figura 9. Añadir JARs	9
Figura 10. Creación de un nuevo <i>MIDlet Suite</i>	10
Figura 11. Ventana Principal del Entorno de desarrollo	10
Figura 12. Ventana de la perspectiva <i>Debug</i>	11
Figura 13. Ventanas de la perspectiva Java <i>Browsing</i>	12
Figura 14. Opciones de los proyectos	13
Figura 15. Métodos de un objeto	13
Figura 16. Indicaciones del Editor	14
Figura 17. Búsqueda de problemas en el código	15
Figura 18. Código de un único método	15
Figura 19. Selección del fichero <code>jxeLinkOptions</code>	16
Figura 20. Ventana <i>Inclusion/Exclusion</i>	16
Figura 21. Añadir recursos	17
Figura 22. Perspectiva <i>Debug</i>	18
Figura 23. Menú <i>Run</i> en perspectiva <i>Debug</i>	19
Figura 24. Añadiendo <i>watchpoint</i> a dos campos	20
Figura 25. Propiedades del <i>Watchpoint</i>	20
Figura 26. Estado de los hilos de ejecución (<i>thread</i>)	21
Figura 27. Contenido de las Variables	22
Figura 28. Tipos de Interrupción	23
Figura 29. Expresiones de depuración	24
Figura 30. Propiedades del Objeto <i>Form</i>	24
Figura 31. Propiedades del objeto <i>StringItem</i>	25
Figura 32. Configuración para la ejecución	26
Figura 33. Pestaña <i>Common</i>	27
Figura 34. Apariencia del Emulador	27
Figura 35. Opciones de ejecución y depuración	28
Figura 36. Configuración de dispositivos	28
Figura 37. Configurando la ejecución con el nuevo emulador	29

1. Introducción

Este manual explica el funcionamiento del programa *WebSphere Studio Device Developer* (WSDD) 5.6, desde su instalación en Windows hasta el desarrollo y pruebas de aplicaciones J2ME sobre dispositivos móviles.

El entorno de desarrollo cuenta con un editor de texto que resalta las palabras clave de Java, un editor gráfico para incluir elementos de las interfaces gráficas, y un depurador para examinar los errores.

Además, permite añadir distintos emuladores para probar las aplicaciones, como los de la serie 40, 60, 80 de *Nokia*, *Pocket PC*, *Wireless Toolkit*, etc.

2. Instalación

El proceso de instalación en Windows es bastante sencillo, casi automático, como la mayoría de programas que se instalan sobre Windows.

Lo primero es introducir el CD de WSDD 5.6 en el ordenador, automáticamente se arranca la máquina virtual de java y se ejecuta el programa de instalación sobre ella (ver Figura 1).

💡 En caso de que esto no ocurra, podemos abrir el explorador de Windows y ejecutar el fichero **autorun** o **windows\setupWin32**.



Figura 1. Bienvenida a la instalación del WSSD

A continuación, debemos confirmar que deseamos instalar WSDD haciendo clic en el botón **Next**. Después se nos pide aceptar la licencia del mismo, y tras la confirmación, tenemos que indicar el directorio destino de la instalación, y los accesos directos que queremos crear en el Menú Inicio. Por último, se muestra un resumen de los parámetros elegidos antes de confirmar el comienzo de la instalación (ver Figura 2).

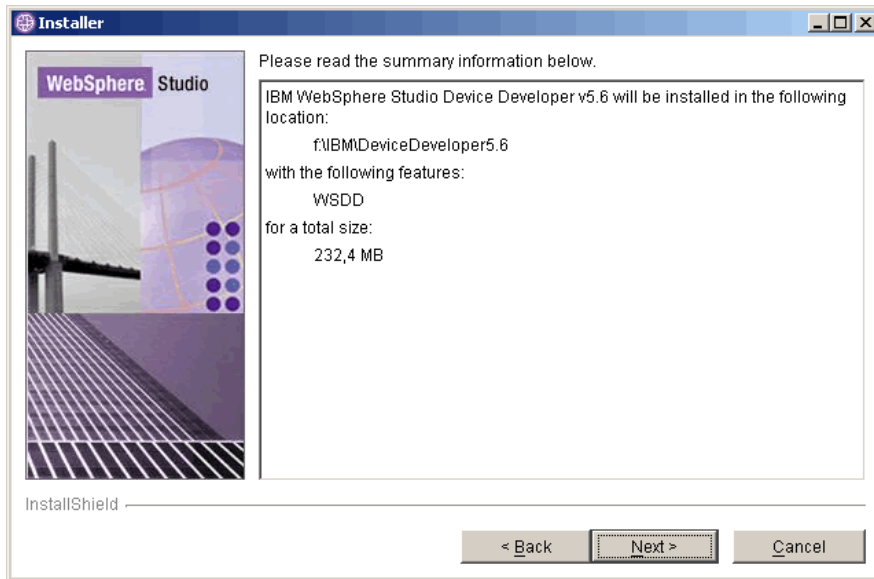


Figura 2. Confirmación de parámetros de instalación

Después de la confirmación, comienza la copia de archivos (ver Figura 3), y una vez finalizado este proceso, se presenta una pantalla de información sobre la correcta instalación del programa (ver Figura 4).

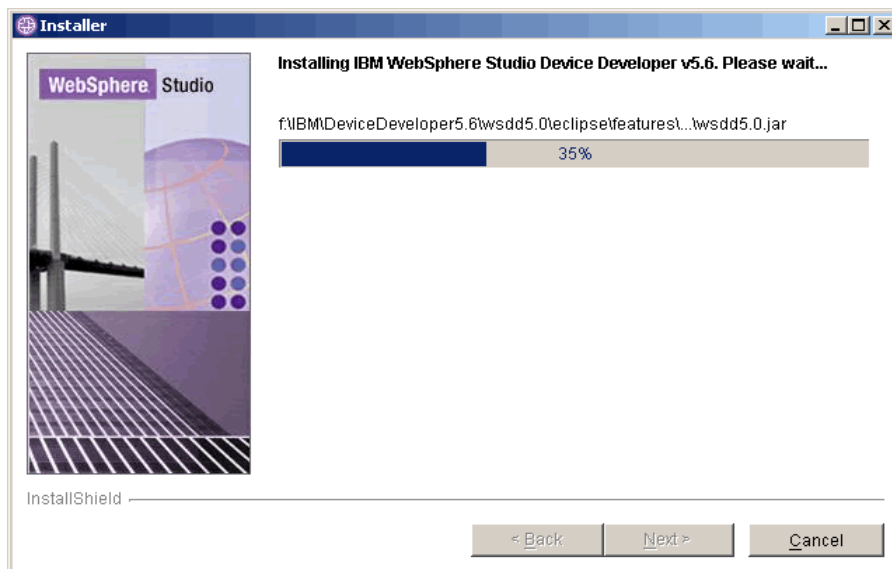


Figura 3. Proceso de copia de archivos

A partir de este momento, podemos utilizar el entorno de desarrollo desde el menú de inicio o el acceso directo del escritorio.

💡 Si miramos las propiedades del icono de ejecución en el escritorio, podremos ver una opción: `-data "C:\Archivos de programa\IBM\DeviceDeveloper5.6\workspace"`, la cual indica el espacio de trabajo, por defecto, donde se almacenarán nuestros proyectos.

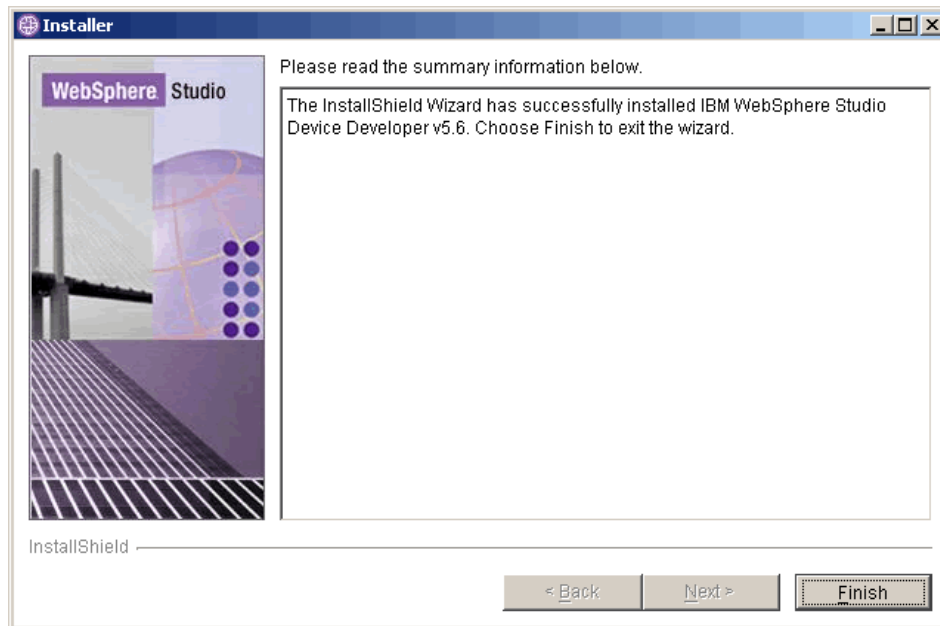


Figura 4. Fin de la Instalación

3. Comenzando con WSDD

Cuando iniciamos WSDD, lo primero que nos presenta es una pantalla de bienvenida en la que se describe brevemente el propósito de éste programa (ver Figura 5). Dicha pantalla presenta varias opciones para comenzar a utilizar el programa:

- La primera permite desarrollar aplicaciones J2ME genéricas, de forma que podremos crear un proyecto desde el principio.
- La segunda es la indicada para crear *MIDlets* (ya que crea automáticamente la estructura básica común de los mismos). De esta forma es el mismo entorno de programación el que nombra las clases y métodos necesarios en todo *MIDlet*.
- Las dos últimas opciones permiten actualizar el WSDD con nuevas versiones o parches desde la página oficial o consultar la ayuda online.

Además, disponemos de algunas opciones activas en el menú principal para configurar el entorno de programación o las típicas acciones de abrir proyectos anteriores y ayuda.

💡 Las aplicaciones se pueden crear directamente desde el menú principal: `File -> New -> Project... -> J2ME`.

A continuación, veremos las dos opciones disponibles para crear aplicaciones J2ME.

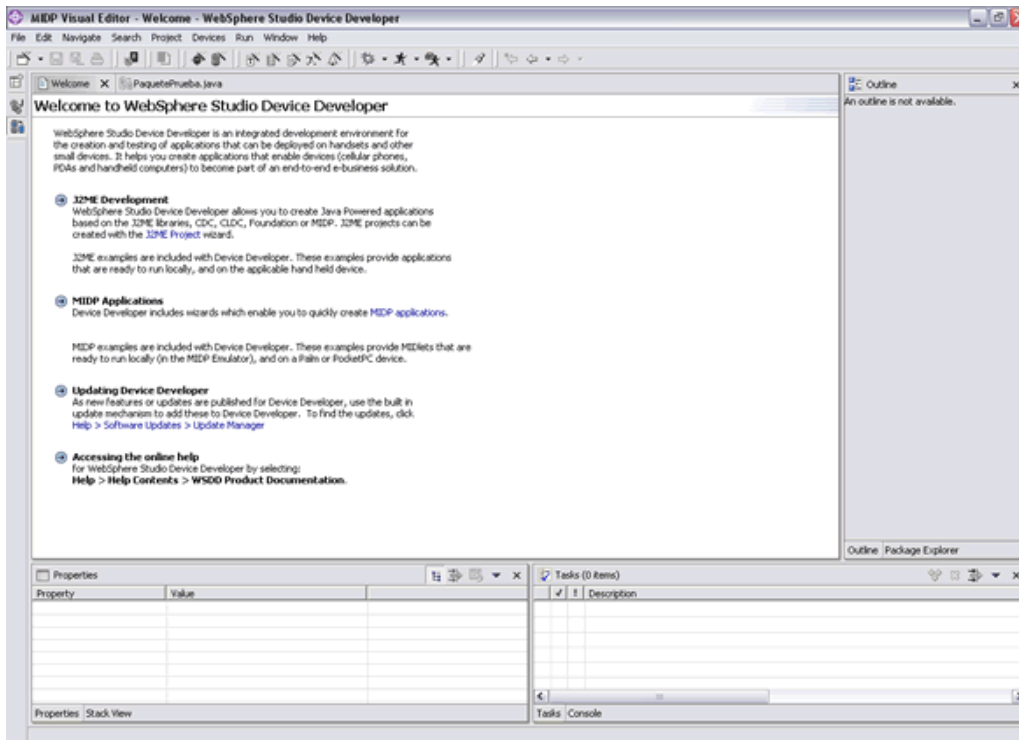


Figura 5. Bienvenida al WSSD

3.1. Proyecto J2ME

Como se mencionó previamente, esta opción es más general, ya que nos permite crear aplicaciones para la configuración CDC o CLDC de J2ME. Al seleccionar esta opción, se pide el nombre del proyecto que vamos a crear, utilizándose después dicho nombre para identificar los archivos propios del mismo. Además, podemos cambiar el directorio por defecto para almacenar nuestro proyecto.

El siguiente paso es elegir las librerías de clases java con las que trabajaremos. Así en función de lo que vamos a desarrollar y para qué dispositivo, elegiremos CDC, CLDC, CLDC 1.1 (en pruebas), *Foundation Profile*, MIDP, MIDP 2.0... En cada opción se nos indica en el panel de la derecha las plataformas sobre las que funcionará nuestra aplicación. Por ejemplo, si elegimos MIDP 2.0, la aplicación podrá correr sobre Windows x86 o sobre PocketPC ARM (ver Figura 6).

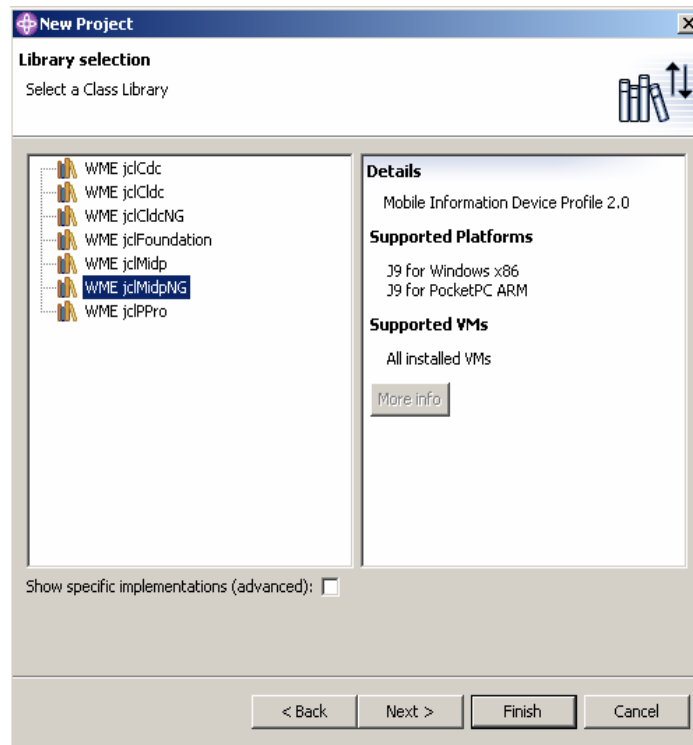


Figura 6. Selección de Librerías Java

Además si seleccionamos la opción "show specific implementations (advanced)", nos permitirá ver información sobre las implementaciones específicas de cada librería, de forma que podremos asegurarnos de elegir las librerías adecuadas si queremos programar una aplicación específica para un entorno concreto (ver Figura 7).

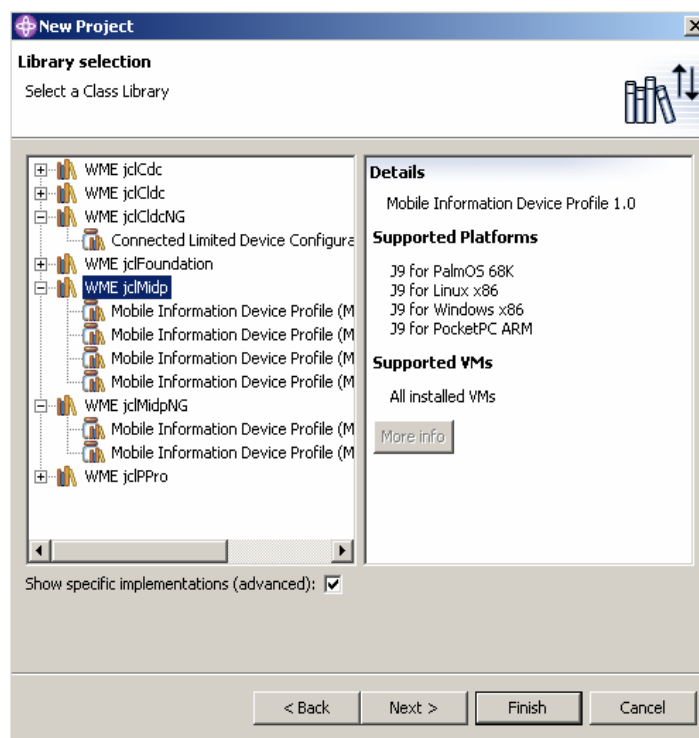


Figura 7. Información detallada de implementaciones específicas

En la pantalla final de configuración se nos permite elegir los contenidos iniciales de nuestro proyecto desde alguna carpeta del sistema (primera pestaña), o simplemente dejarlo igual para crear un proyecto nuevo con los contenidos por defecto. Además, se pueden añadir otros proyectos (segunda pestaña), librerías o archivos jar/zip (tercera pestaña) ya desarrollados (ver Figura 8, Figura 9).

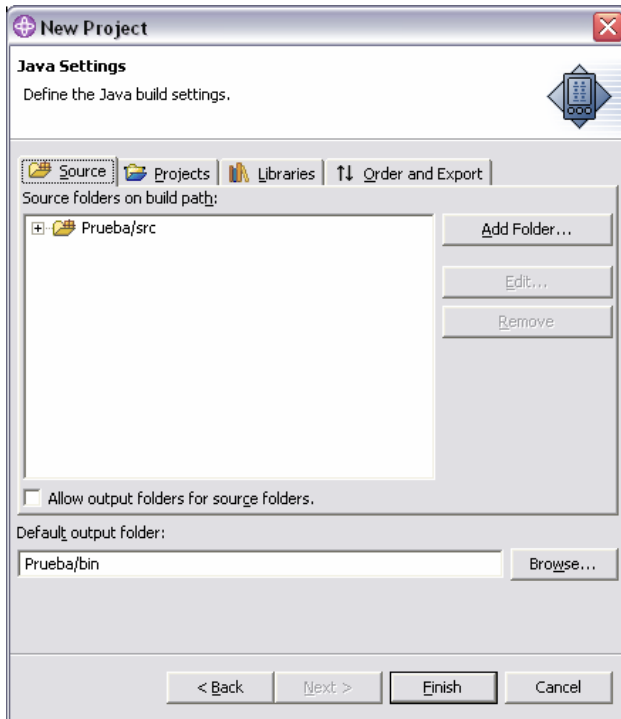


Figura 8. Configuración final del proyecto

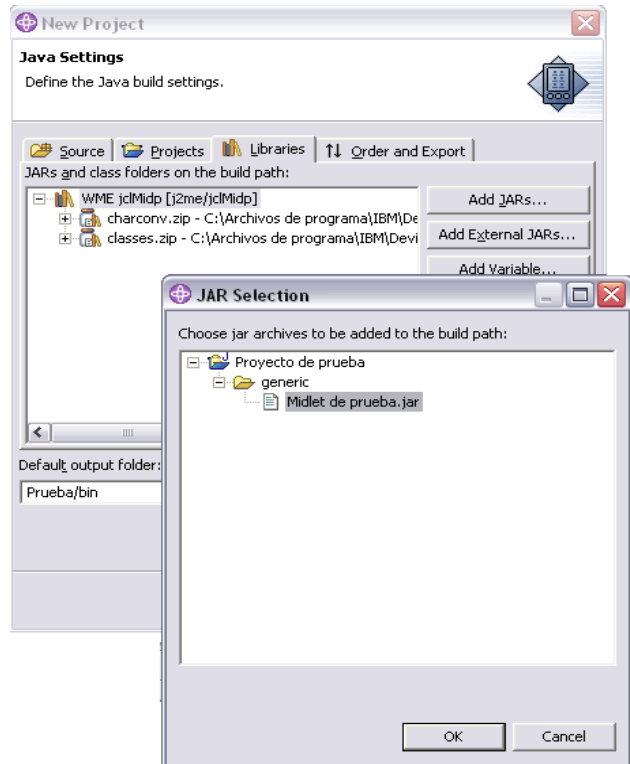


Figura 9. Añadir JARs

Al hacer click en **Finish**, hemos creado nuestro proyecto preparado para empezar a crear nuestras clases.

3.2. MIDlet Suite

El otro asistente básico permite crear *MIDlet* Suite, generando el esqueleto básico de un *MIDlet* de forma automática. Similar a la opción anterior, tenemos que elegir el nombre del proyecto (ver Figura 10), el nombre del *MIDlet*, la versión de MIDP, y el nombre de la clase. El asistente, se encarga de crear la clase básica, entre otras cosas.

Se puede incluir un *MIDlet* ya existente (hacer clic en **Browse**), o crear un proyecto nuevo. En el último caso, tenemos que indicar el nombre del nuevo proyecto, el nombre del paquete (**Package**), y el nombre de la clase. La convención es que empiece por minúscula, al contrario que el nombre de la clase, que se espera empiece por mayúscula.

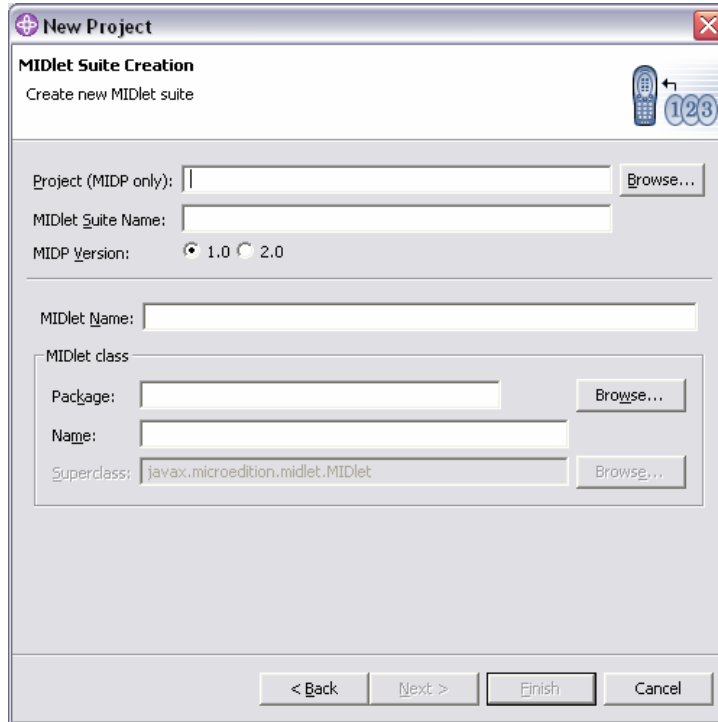


Figura 10. Creación de un nuevo *MIDlet Suite*

Una vez elegimos todos estos datos de configuración, se nos presenta una pantalla como la mostrada en la Figura 11.

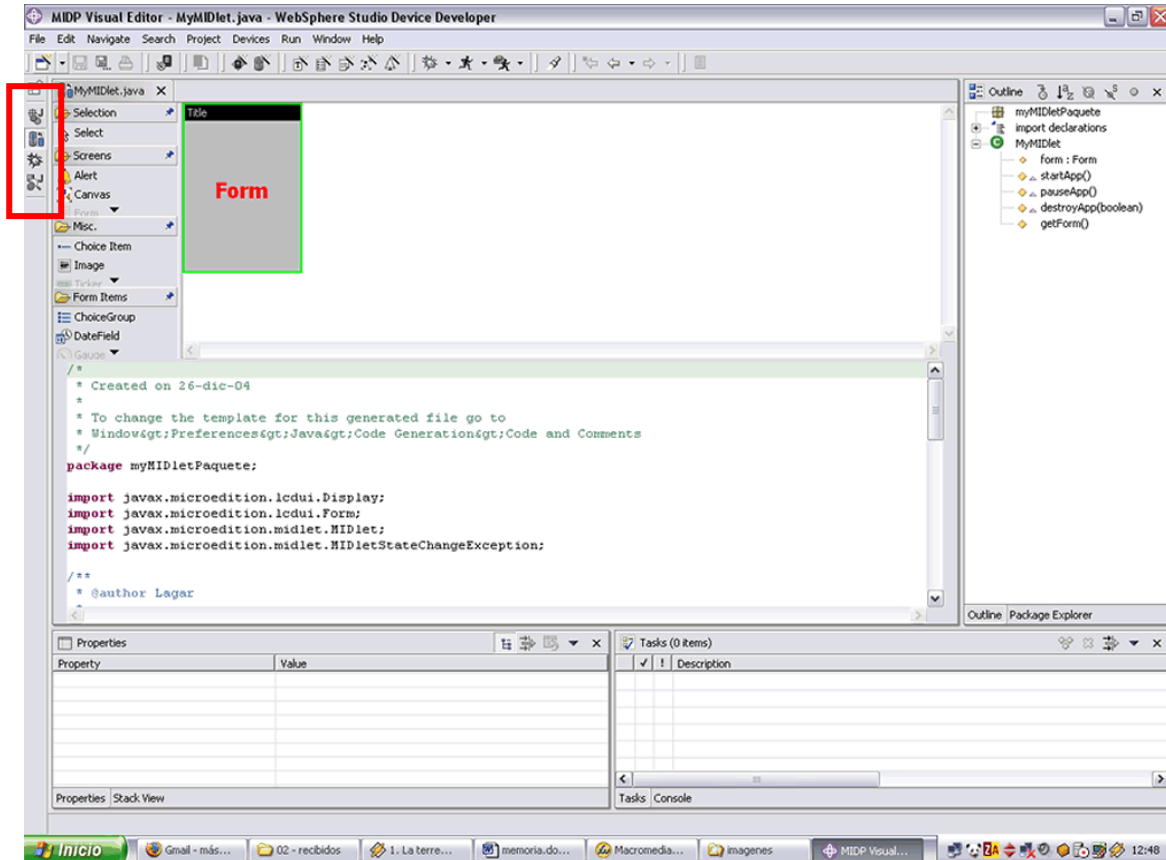


Figura 11. Ventana Principal del Entorno de desarrollo

4. Entorno de desarrollo

Las distintas configuraciones para ver nuestra aplicación se denominan **perspectivas**, las cuales a su vez tienen distintas zonas (Show view). La que nos aparece en la imagen es la denominada perspectiva *MIDP Visual Editor* (📱). En ella tenemos como ventana principal un editor visual, en el que podemos colocar los elementos que queramos en nuestro *MIDlet*. En el centro se encuentra la ventana con el código fuente de nuestra clase (sección 0), la cual contiene la estructura básica del *MIDlet* (`startApp()`, `pauseApp()` y `destroyApp()`) ya que el asistente lo ha creado por nosotros. La parte inferior muestra en el lado izquierdo una ventana *Properties*, que nos indicará y nos permitirá editar las propiedades de los elementos visuales, y en el lado derecho *Tasks* en la cual se muestran mensajes de error, advertencia, o información útil con su descripción. Finalmente a la derecha se resume la estructura de la clase que estamos editando (ventana *Outline*), el paquete al que pertenece, las declaraciones de importación, sus métodos y propiedades.

Aparte de esta perspectiva, existen otras que podemos elegir en el menú principal, *Windows*. Además, se puede cambiar de perspectiva utilizando los botones mostrados en la barra de herramientas a la izquierda del todo de la ventana, resaltado con un cuadro en la Figura 11.

La perspectiva *Java* (🌐), cambia las ventanas inferiores por una consola de salida de Java, de forma que se pueden observar los mensajes que arroje la misma.

La perspectiva *Debug* (⚙️) nos facilitará las cosas cuando depuremos nuestra aplicación (ver Figura 12), cuya utilidad es detallada en la sección 4.4.

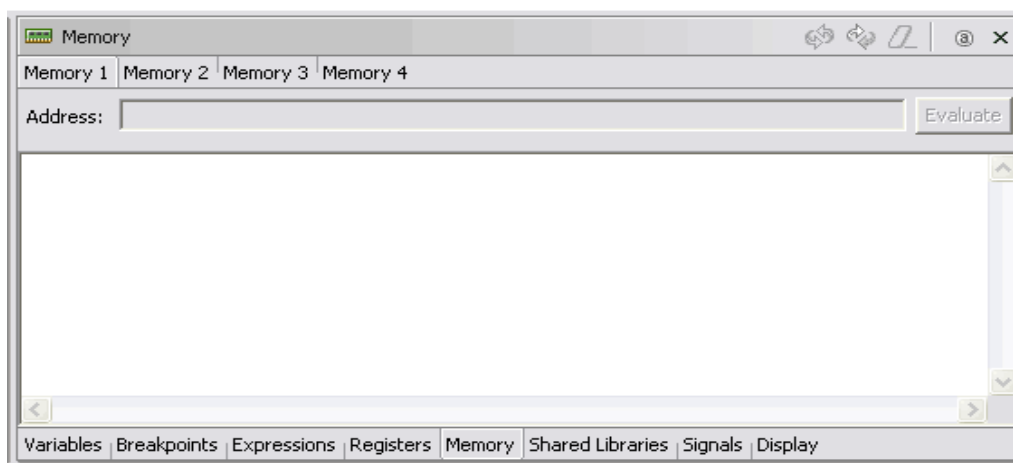


Figura 12. Ventana de la perspectiva *Debug*

Por último, la última perspectiva básica es *Java Browsing* (🌐), en la cual podemos navegar por la estructura de nuestros proyectos java. Así, tenemos

una ventana con todos nuestros proyectos, otra con los paquetes, con los tipos definidos, y con los miembros de las clases anteriores (ver Figura 13).

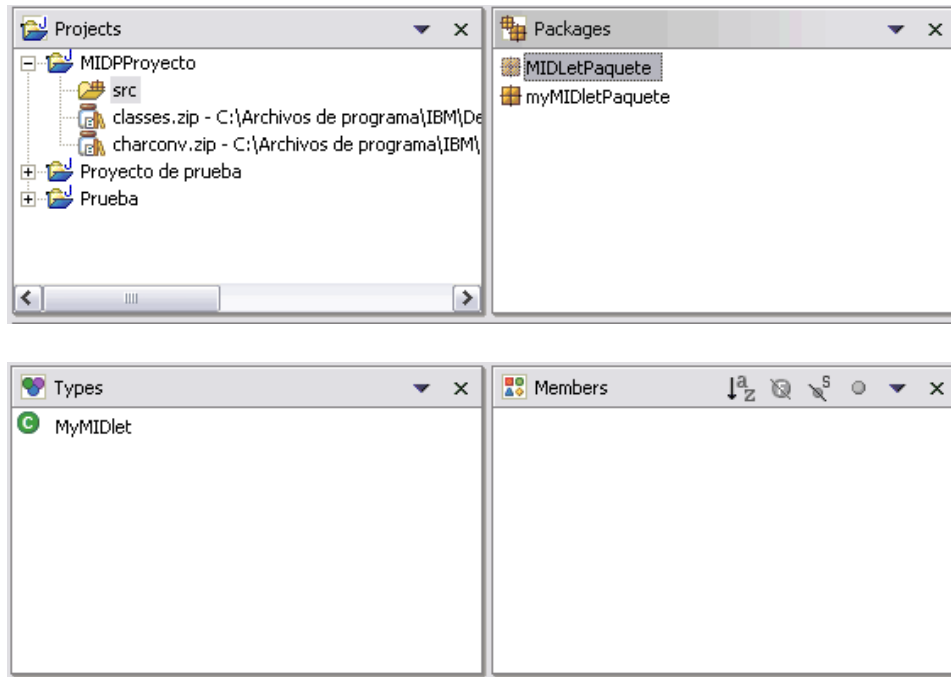


Figura 13. Ventanas de la perspectiva Java Browsing

Como vemos, prácticamente todas las herramientas que necesitamos para crear nuestras aplicaciones las tenemos disponibles en el panel principal de la aplicación, mediante una perspectiva u otra. Además, también podemos recurrir a las barras de herramientas como accesos más rápidos a otras funciones. Por último, existen los menús superiores más típicos, como *File*, *Edit*, *Search*, ... que en este caso cumplen funciones generales comunes a la mayoría de las aplicaciones, pero otros, como las opciones *Project*, *Run* o *Devices* cumplen funciones específicas del WSDD.

Además, en caso de tener más de un proyecto, podemos navegar entre ellos usando la ventana *Package Explorer*, la cual nos presenta todos los proyectos existentes y los elementos que los componen. Esta ventana puede abrirse desde el menú: *Window* -> *Show view* -> *Package Explorer*. Las acciones básicas (abrir, cerrar, generar la documentación, construir) que pueden ser realizadas sobre los proyectos se encuentran en el menú principal: *Project* (ver Figura 14).

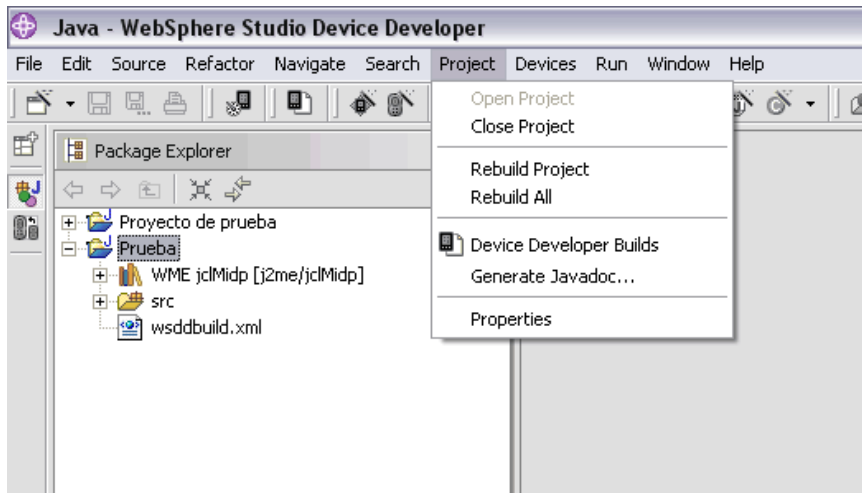


Figura 14. Opciones de los proyectos

4.1. Editor del código fuente

Una de las ventanas principales del WSSD es el editor del código fuente. Este editor está siempre presente, aunque cambiemos entre las distintas perspectivas.

Como su nombre lo indica, en este editor se muestra el código fuente de la clase con todos sus elementos. Pero además incluye diversas funcionalidades para ayudar al desarrollador, entre ellas tenemos:

- Utiliza distintos colores para diferenciar entre palabras reservadas, comentarios, cadenas de texto, y nombres de variables. De este modo podemos encontrar e identificar más fácilmente una línea de texto o una instrucción.
- Muestra los campos y métodos pertenecientes al objeto al que estamos haciendo referencia según vamos escribiendo (ver Figura 15). De esta forma, podemos elegir el que queramos moviéndonos por la lista con los cursores, y pulsando <enter>.
- Además, en la lista de métodos de los objetos, se nos indican los parámetros necesarios en la llamada al mismo, y el tipo del valor que devuelve, junto con una breve descripción (ver Figura 15).

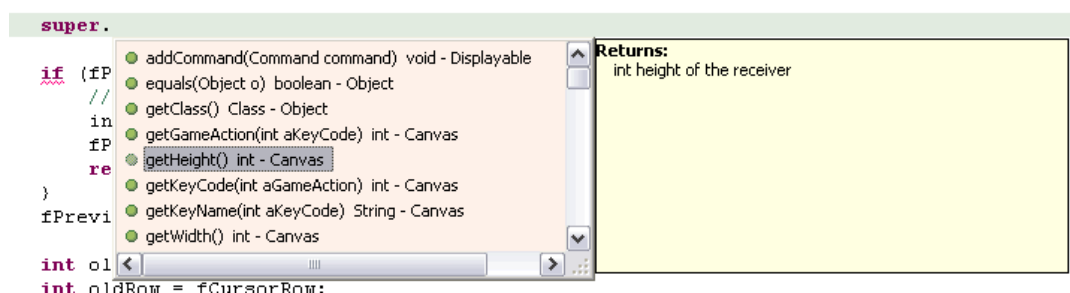


Figura 15. Métodos de un objeto

- Incluye una función automática para cerrar paréntesis y corchetes, de forma que es casi imposible que se nos olvide hacerlo alguna vez, lo que por otra parte es un fallo bastante común, y que en el caso de tener muchas sentencias anidadas cuesta identificar. Para esto, también dispone de un resaltado sobre fondo gris de la pareja (ya sea paréntesis o corchete) del elemento sobre el que tenemos el cursor.
- En caso de cometer algún error de sintaxis (❌), remarca la parte de código errónea en rojo. Errores como no incluir un punto y coma al final de línea, escribir una variable de forma errónea o no declarada previamente, llamar un método no definido, etc.
- Además, utiliza advertencias (⚠️) que son subrayados amarillos. Las advertencias no serán causa de error, pero ayudan a mejorar el código; por ejemplo, importar alguna clase que no se utilice nunca, definir métodos privados que nunca se llaman dentro de la propia clase, etc.
- Utiliza un resaltado azul asociado a elementos de información útil (ℹ️), que podrían contribuir a una mejora de la estructura del código; por ejemplo, la ausencia de un método `get` para una variable `protected`.

Todas estas indicaciones se muestran en el margen izquierdo del editor, con su símbolo correspondiente al lado de la línea asociada al mensaje. Además, aparecen en la ventana `Tasks`, con su descripción. Si colocamos el cursor sobre la palabra subrayada, también muestra la descripción como un texto alternativo (ver Figura 16).



Tasks (3 items)				
✓ !	Description	Resource	In Folder	Location
⚠️	The import javax.microedition.lcdui.Graphics is never used	HolaMundo.java	MIDPProyecto/src/myMIDletPaquete	line 18
ℹ️	Field is missing a corresponding getter method.	HolaMundo.java	MIDPProyecto/src/myMIDletPaquete	MIDlet
❌	The method repaint() is undefined for the type MiForm	MiForm.java	MIDPProyecto/src/myMIDletPaquete	line 39

Figura 16. Indicaciones del Editor

Además de tener un indicador en el margen izquierdo, en el margen derecho también nos aparece un código de colores para ayudarnos a localizar estos mensajes. En la parte superior existe un pequeño cuadrado, que en caso de que tengamos este tipo de alertas, se mostrará del color de la más grave (rojo, amarillo o azul).

También aparecen pequeñas marcas del color de los mensajes. Así, se escala el tamaño del archivo fuente en el margen derecho, y las marcas se colocan a la altura de la línea que provoca el mensaje



en esta escala. Si hacemos clic sobre ellas, nos llevarán a la palabra que está provocando el mensaje al que hace referencia.

Estos mensajes de error, pueden localizarse también muy fácilmente con los botones `Go to next problem`, y `Go to previous problem` (ver Figura 17), situados en la barra superior en la parte derecha.

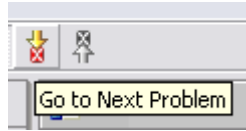


Figura 17. Búsqueda de problemas en el código

- Finalmente, indicar que si tenemos abierta la ventana Outline, podemos presionar el botón `Show source of selected element only`. De esta forma el editor sólo nos mostrará el código fuente del elemento que tengamos seleccionado (ver Figura 18), haciéndonos mucho más legible el código de métodos largos y complejos.

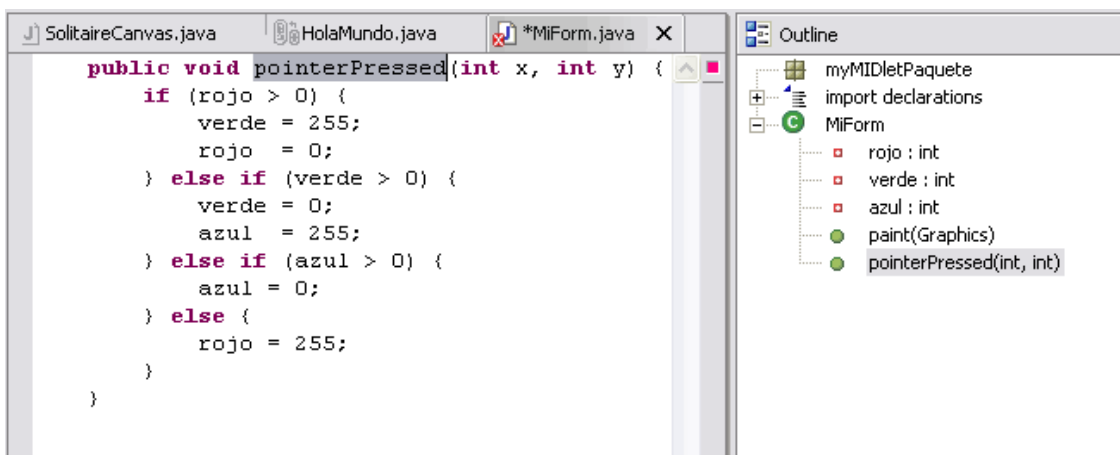


Figura 18. Código de un único método

4.2. Usando Imágenes u otros recursos

Dentro de un *MIDlet* es posible usar imágenes u otro tipo de recursos, por ejemplo, sonidos. Estos recursos deben ser empaquetados con la aplicación, por lo tanto, en *WebSphere* para añadir recursos debemos seguir los siguientes pasos:

- a) Copiar las imágenes en la carpeta `bin` del proyecto. Pueden estar dentro de alguna subcarpeta.
- b) Hacer doble clic sobre el fichero `jxeLinkOptions` que se encuentra dentro de la subcarpeta `nombre-MidletsInfo` (ver Figura 19) El fichero se abre en el editor.

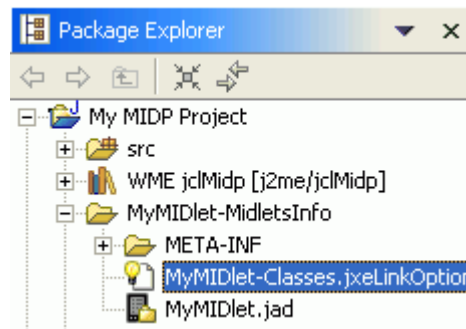


Figura 19. Selección del fichero `jxeLinkOptions`

- c) En la parte inferior del editor se encuentran varias pestañas. Seleccionamos la pestaña In/Exclusion (ver Figura 20).
- d) Muestra un formulario, en el menú desplegable debemos seleccionar Include Resources y hacer clic en el botón New.
- e) Aparece un cuadro de diálogo (ver Figura 21), en el cual debemos escribir el nombre de la imagen y aceptar. En caso de que se encuentre dentro de una subcarpeta debemos escribir: subcarpeta/nombreimagen.png. Este procedimiento se realiza para cada uno de los recursos.
- f) Finalmente, no olvide guardar los cambios realizados en el archivo.

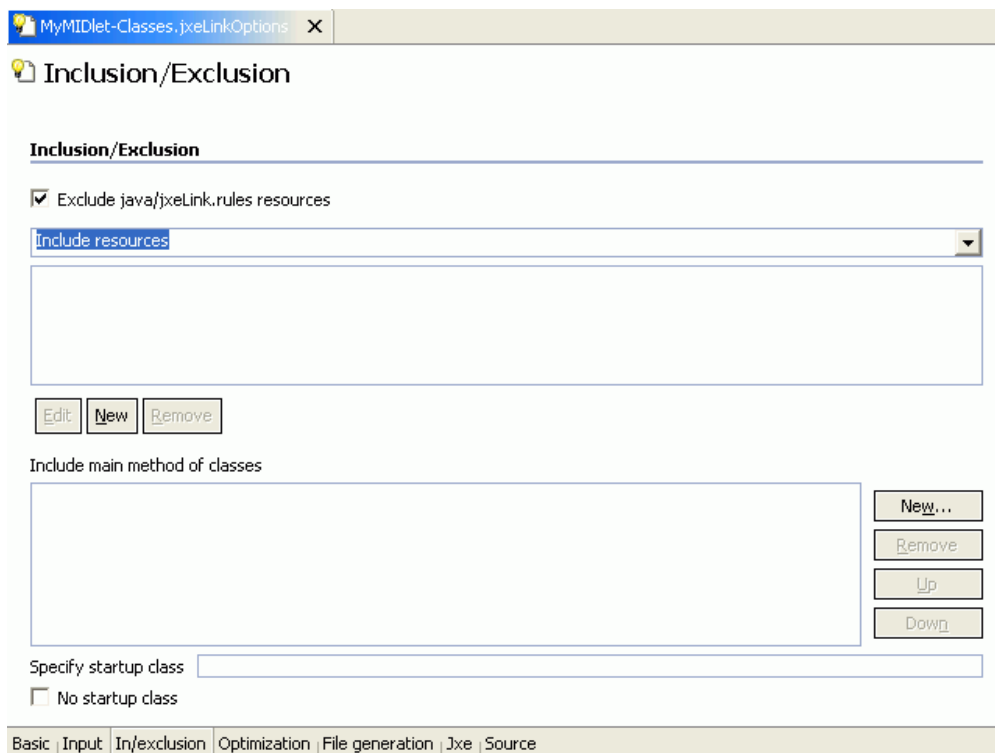


Figura 20. Ventana Inclusion/Exclusion

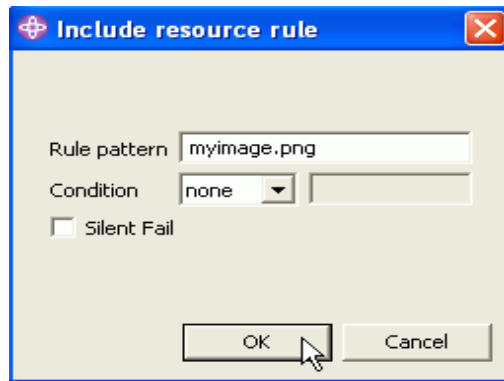


Figura 21. Añadir recursos

💡 No olvide que las imágenes tienen que estar en formato PNG.

4.3. Utilización de RMS

Cuando se trabaja con RMS es necesario tener presente varias recomendaciones, la primera es que siempre hay que cerrar los Record Store porque sino podemos perder la información que estamos almacenando en ellos. Recordar también que cuando se almacena información en un Record Store se obtiene el identificador (`recordID`) de ese record, y siempre que queramos obtener, modificar o borrar esta información debemos acceder a ella utilizando el identificador. Por último, si queremos borrar un Record Store es recomendable eliminar primero todos los records existentes en el Record Store.

Además de estas recomendaciones, a la hora de trabajar con almacenamiento persistente en los emuladores del WebSphere Device Developer es necesario tener en cuenta las siguientes consideraciones:

- a) Los RecordStore del simulador (ive-2.1) se almacenan en el sistema de ficheros en el directorio "recordStores" en el HOME del usuario (por ejemplo, en Windows XP es `c:\Documents and Settings\nombreusuario`). Por lo que podéis borrar el contenido de esta carpeta si queréis eliminar los RecordStore para hacer pruebas.
- b) El WebSphere presenta el siguiente problema: cuando se elimina un Record Store (`deleteRecordStore`) no se eliminan los record asociados a ese Record Store, de tal manera que si creáis un nuevo Record Store con el mismo nombre, os van a aparecer nuevos records que teníais almacenados antes. Por lo que, para eliminar un recordStore debéis eliminar todos los records (`deleteRecord`) del RecordStore antes de eliminar el RecordStore en sí.
- c) Además, fijaros que los `recordID` que proporciona el WebSphere cuando añadís records a los Record Store son consecutivos y no se resetean nunca, es decir:

1. Creo un RecordStore la primera vez.

2. Inserto un record, el recordID asignado es 1.
3. Inserto otro record, el recordID asignado es 2.
4. Borro record 1 y 2.
5. Borro RecordStore.
6. Creo otro RecordStore.
7. Inserto un record, el recordID asignado es 3.
8. Inserto un record, el recordID asignado es 4.
9. ... y así sucesivamente.

El problema que plantea esto (si no borráis los record dentro del Record Store) es que cuando leéis desde el recordId=1 os aparece todos los record que habíais añadido a lo largo de las sucesivas ejecuciones del programa.

4.4. Depuración de aplicaciones

Como se mencionó anteriormente, la perspectiva *Debug* nos ofrece distintas posibilidades de depuración.

Lo primero que debemos hacer para encontrar errores que se nos resisten en nuestra aplicación, es seleccionar la clase elegida y cambiar a esta perspectiva (ver Figura 22)¹.

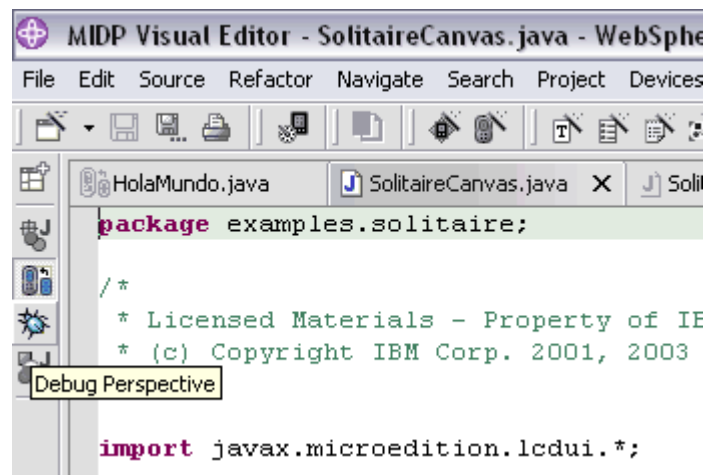


Figura 22. Perspectiva *Debug*

Al cambiar a esta perspectiva, podemos comprobar que la opción *Run* del menú tiene muchas más opciones que en las demás perspectivas (ver figura 5.2), indicadas para la depuración de la aplicación, como *Resume* (continuar), *Suspend* (pausar), *Terminate* (finalizar), *Add Breakpoint* (añadir punto de parada), etc. Estas opciones nos permiten añadir puntos

¹ Para mostrar las opciones disponibles, se utilizará como ejemplo el MIDlet Solitaire que incluye WSSD.

donde se detendrá la ejecución de la aplicación para comprobar el estado de la misma. Además, podemos continuar o parar en cualquier momento, por ejemplo, cuando se haya detectado el error, o simplemente para ver cómo evoluciona el programa.

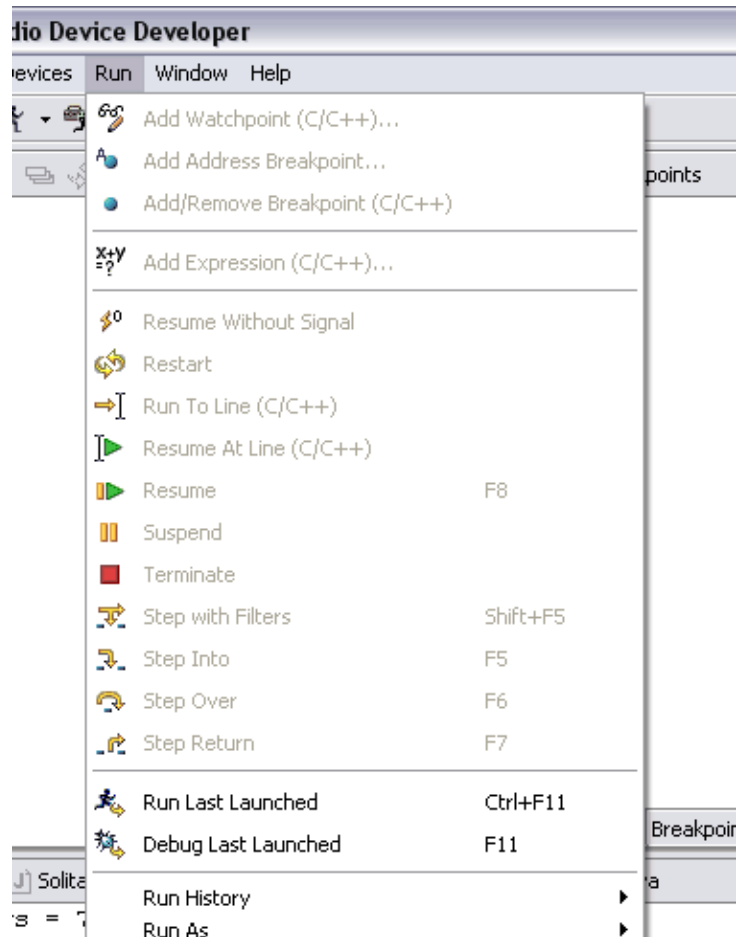


Figura 23. Menú Run en perspectiva Debug

El proceso de depuración comienza al establecer los puntos críticos que deseamos examinar (*breakpoints*). Para ejecutar y depurar la aplicación, hacemos clic en Run -> Debug.

💡 Al ejecutar la aplicación se pueden presentar problemas si tenemos instalado un *firewall* local en la máquina. Esto se debe a que para realizar la depuración se abre una conexión a un puerto TCP, de forma que si el *firewall* niega esta conexión, el *WebSphere* nos devolverá un mensaje de error y finalizará la aplicación.

Las opciones de configuración pueden verse en la sección 5, ya que en esta sección nos interesa mostrar los distintos tipos de *breakpoints* que podemos utilizar en función de nuestras necesidades:

- **Watchpoint.** Nos permite elegir un campo de una clase, de forma que cuando se acceda a dicho campo la ejecución se interrumpirá para poder ver qué es exactamente lo que ocurre con él. Para añadirlo (o

removerlo), hay que colocar el cursor sobre la declaración del campo elegido, e ir al menú Run -> Add/Remove -> Watchpoint. En el ejemplo se han colocado dos puntos, los cuales son indicados con el símbolo de unas gafas junto a la declaración del campo (ver Figura 24).

```

final static private int gCols = 7;

private byte fBoard[] [];
private int fNumberOfColors;
private int fPegWidth;
private int fPegHeight;
private int fSelectedPegCol;
private int fSelectedPegRow;
// where the "cursor" is currently located

```

Figura 24. Añadiendo watchpoint a dos campos

Una vez hecho esto, en la ventana superior derecha nos aparece un listado de los puntos que tenemos en toda nuestra clase, con su tipo y propiedades. En este caso, si hacemos clic sobre uno de ellos con el botón derecho del ratón, y elegimos propiedades, se puede configurar su comportamiento (ver Figura 25). Así, podemos elegir si activarlo o no, qué hacer en caso de alcanzar el punto (detener únicamente el hilo que provoca la parada, o toda la máquina virtual), y qué condiciones deben cumplirse para que se pare (acceso al campo, o modificación).

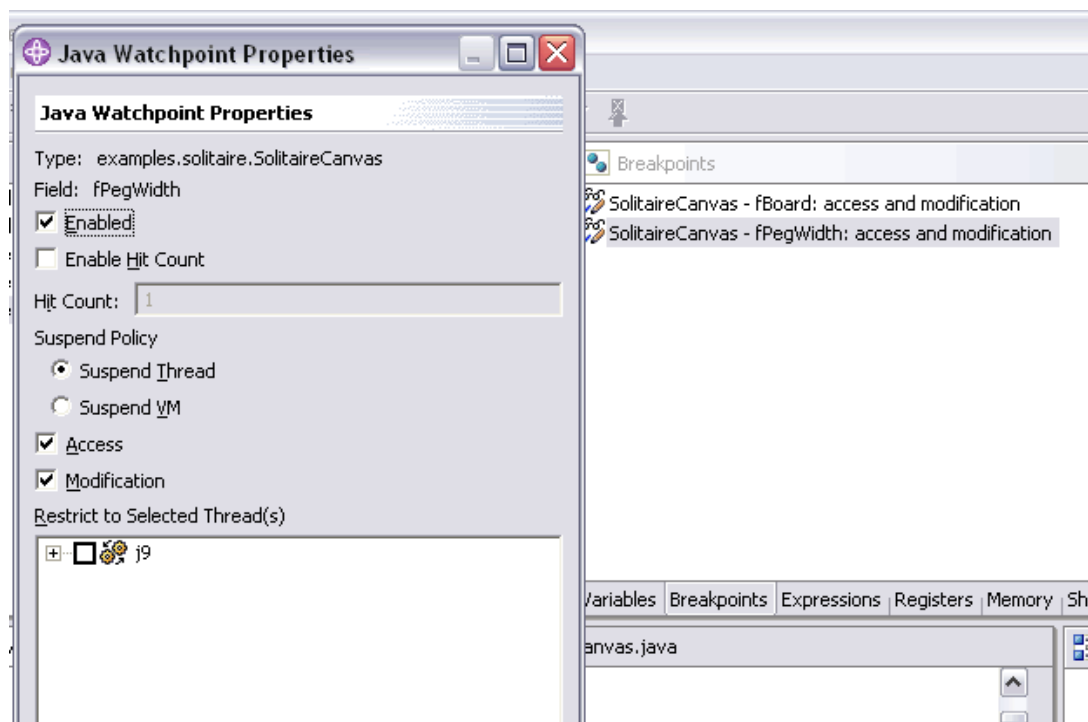


Figura 25. Propiedades del Watchpoint

Cuando ya tenemos todo listo, ejecutamos la aplicación en modo de depuración. En este caso, nada más lanzarla se detiene, ya que hemos colocado los puntos de parada en campos que se inicializan al crear el *MIDlet*.

En la ventana de código fuente, se nos resalta la línea en la que se accede al campo para poder ver exactamente cuál es la operación que se va a realizar sobre él. En este momento disponemos de las siguientes opciones para verificar el estado de la aplicación:

- a) En la ventana Debug se muestra el estado de los *threads* activos, y en qué momento se ha provocado la parada (ver Figura 26). Además, se indica la pila de procesos que se están ejecutando en el momento de la misma.

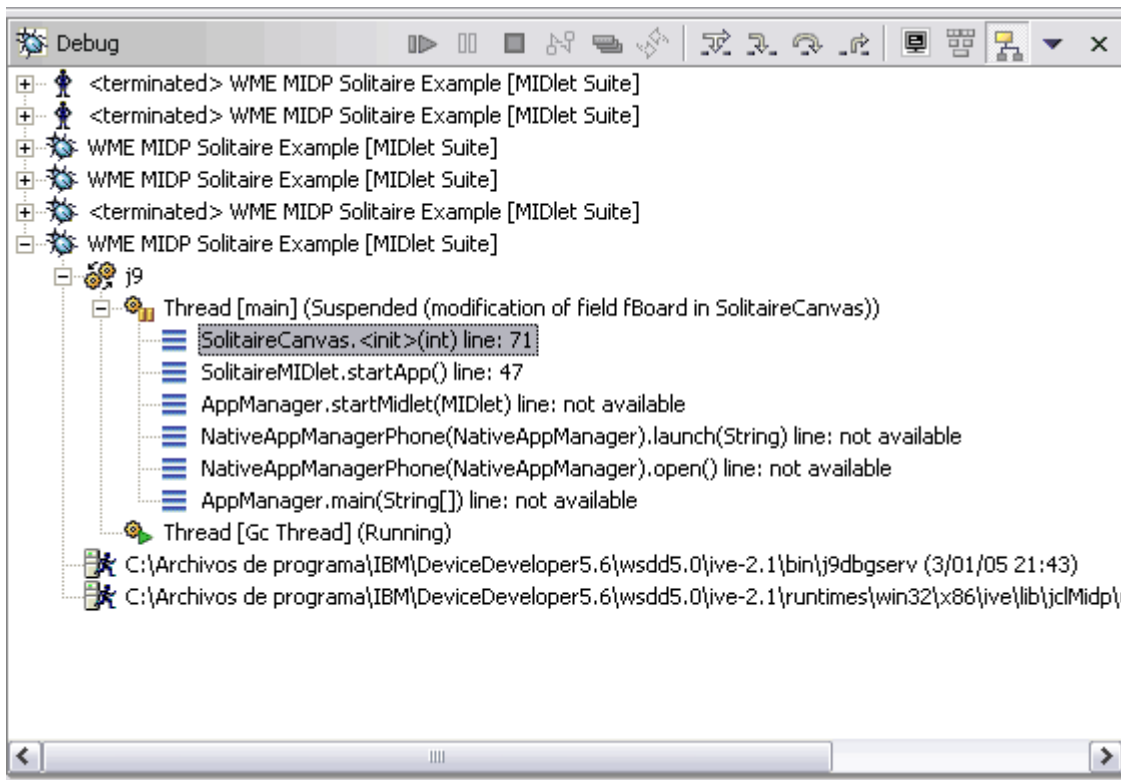


Figura 26. Estado de los hilos de ejecución (thread)

En el ejemplo vemos que el método que ha provocado la interrupción es `SolitaireCanvas.<init>`, en la línea 71. Además, este método ha sido llamado por el `SolitaireMIDlet` en su rutina de inicialización (`startApp`). Si hacemos doble clic sobre cualquiera de estos mensajes, el editor nos llevará directamente a la línea de la clase donde se llamó al método indicado.

Además, nos proporciona la causa de la interrupción, para identificar el punto en el que se ha parado, en caso de que tengamos varios. En la imagen se ve que la causa ha sido la modificación del campo `fBoard` en la clase `SolitaireCanvas`.

- b) En la ventana superior derecha, tenemos varias pestañas de información. Además de la de `Breakpoints`, existe una llamada `Variables` (ver Figura 27). En ella se muestra el nombre de todas

las variables que existen en memoria en ese momento, y su valor.

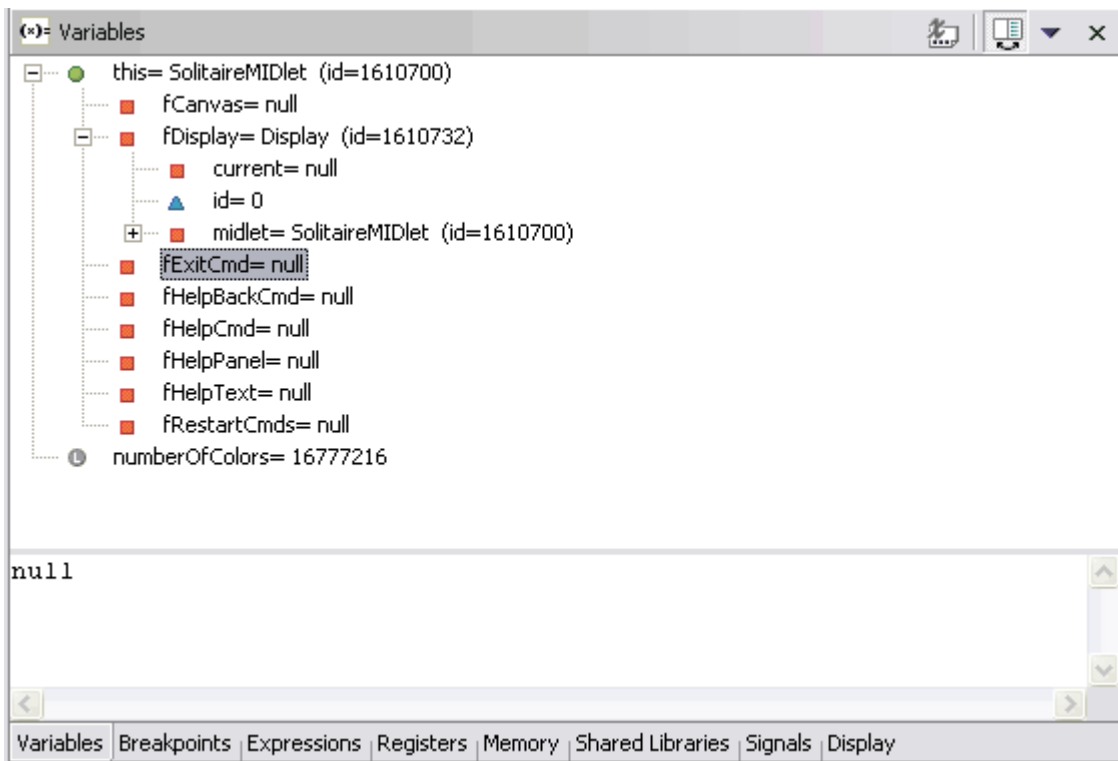


Figura 27. Contenido de las Variables

Se puede ver en la imagen que aparecen todos los campos agrupados por clase a la que pertenecen. Además, si uno de los campos es a su vez un objeto, se pueden desplegar también sus respectivos campos para comprobar el estado de dicho objeto. Como vemos, prácticamente todos los campos están puestos a `null`, ya que, como hemos dicho, se han colocado los puntos de interrupción al comienzo de la aplicación.

En el menú `Run` nos aparecen las opciones de continuar habilitadas: `parar` o `avanzar` paso a paso. Si pulsamos `F6` (`Step Over`), la ejecución avanzará hasta la siguiente línea de código. Si pulsamos `F7` (`Step Return`), la ejecución irá hacia atrás, de forma que volverá a la función anterior que realizó la llamada a la actual (es decir, es como incluir un `return` en el punto del método en el que estamos).

La diferencia entre los menús `Step Over` y `Step Into`, se observa cuando se llama a un método desde la siguiente línea. Así, el primero de ellos ejecutará la siguiente línea como una operación única, y podremos ver el resultado que devuelve. Sin embargo, el segundo nos permite continuar con la depuración y la ejecución paso a paso dentro del método al que se llama.

F8 (Resume) reanuda la ejecución, que continuará hasta que se vuelva a encontrar otro punto de interrupción. Y por último, la opción `Terminate`, finalizará la aplicación.

Combinando estos menús de avance o retroceso por la aplicación, con la ventana `Variables`, podemos comprobar con exactitud la evolución del valor de los campos de cualquier clase activa.

- **Breakpoint.** Nos permite añadir un punto de interrupción en cualquier línea de código que se ejecute en la aplicación.
- **Method Breakpoint.** Añade un punto de interrupción en el método al que pertenece la línea actual. Así, cuando se llame al método desde cualquier parte de la aplicación, ésta se detendrá en la primera línea del mismo.
- **Java Exception Breakpoint.** La ejecución se detendrá cuando ocurra una excepción Java del tipo elegido. Es posible elegir si se detendrá la ejecución cuando la excepción se captura (con un bloque de código `try/catch`), cuando no se haga, o en ambos casos.

En la Figura 28 se muestran todos los tipos de interrupciones posibles, con el icono que las identifica.

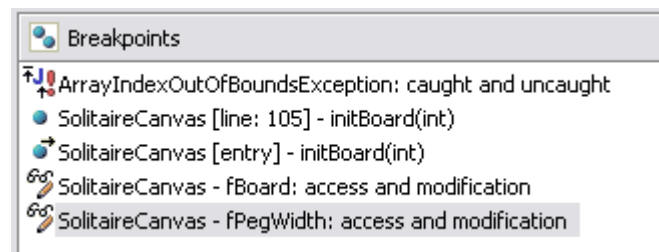


Figura 28. Tipos de Interrupción

Por último, destacar que en la pestaña llamada `Expressions` (ventana superior derecha) se pueden añadir expresiones (de cualquier tipo válido en Java) que se evaluarán según avanza la ejecución, de forma que podemos ver el resultado de operaciones complejas sobre campos sencillos (ver Figura 29).

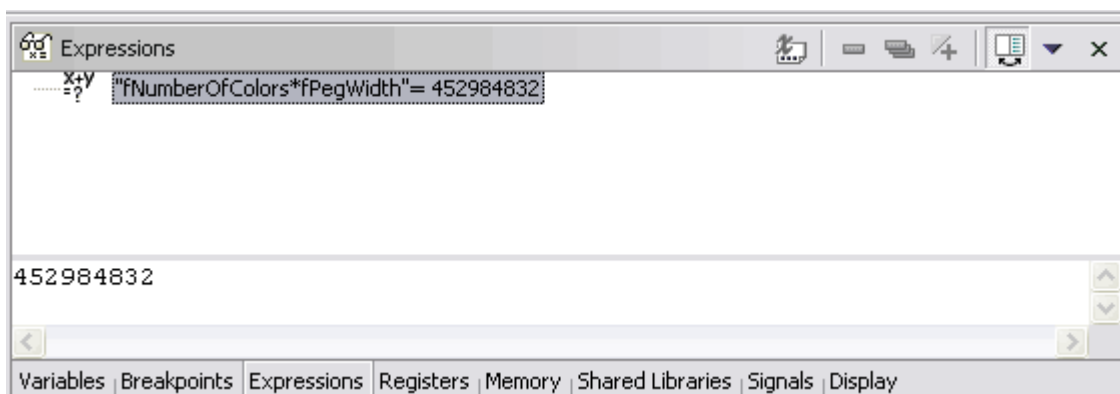


Figura 29. Expresiones de depuración

5. Una aplicación sencilla de ejemplo: Hola Mundo

Para comenzar a familiarizarnos con el *WebSphere*, crearemos un *MIDlet* muy sencillo que nos salude con la frase "hola mundo" cuando pulsemos un botón.

Para ello seleccionamos la opción crear un *MIDlet Suite* (File -> New -> Project -> J2ME -> MIDlet Suite). Como vimos anteriormente, automáticamente se crea un *MIDlet* con un *Form* como contenedor principal de los elementos.

Ahora, debemos crear la interfaz gráfica, que para este caso, será suficiente con un campo de texto. Para ello, elegimos el elemento *StringItem* del menú de la izquierda dentro de *Form Items* y lo colocamos sobre nuestro panel inicial (*Form*).

El siguiente paso es editar los campos de los elementos que hemos creado: el formulario y el campo de texto. Haciendo clic sobre cada uno de ellos, en la ventana *Properties* nos aparecen todas las propiedades que podemos editar (ver Figura 30).

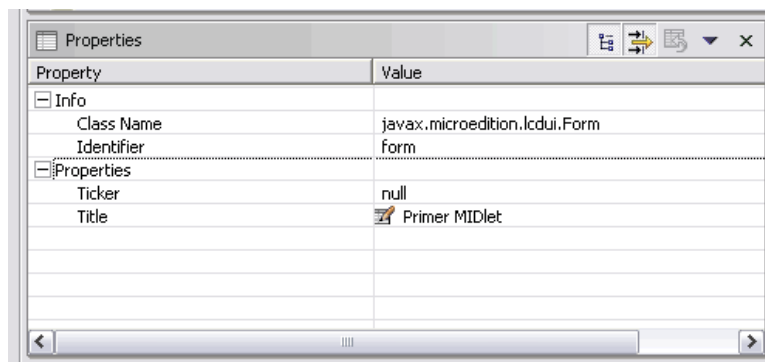


Figura 30. Propiedades del Objeto Form

Para el formulario, hemos indicado únicamente su título (ver Figura 30). Sin embargo, en el campo de texto hemos editado tanto su etiqueta como su contenido (ver Figura 31), de forma que la etiqueta nos indique visualmente cuál es nuestro elemento y el contenido lo ponemos vacío para rellenarlo más adelante.

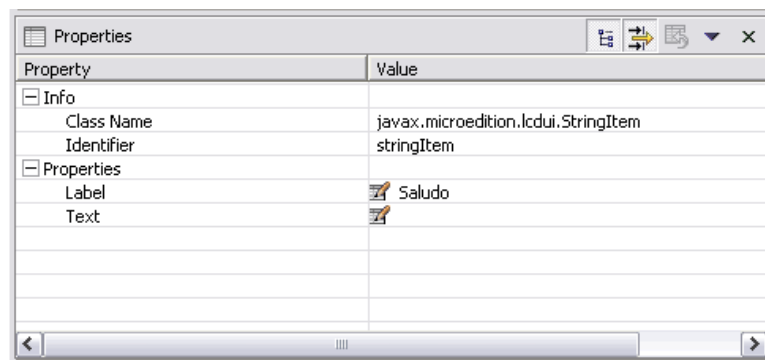


Figura 31. Propiedades del objeto `StringItem`

Ahora debemos crear la funcionalidad de nuestra aplicación mediante código. Para ello, debemos crear un botón de acción, tipo `Command`; y le decimos que el `CommandListener` es la misma clase:

```
okCommand = new Command("saluda", Command.OK, 1);
form.addCommand(okCommand);
form.setCommandListener(this);
```

Implementamos la función `commandAction` necesaria que nos muestre el mensaje "HOLA MUNDO!!", y ya tenemos escrito todo nuestro MIDlet.

```
public void commandAction(Command c, Displayable d) {
    if (c==okCommand) {
        if (this.stringItem.getText().equals("")) {
            this.stringItem.setText("HOLA MUNDO!!");
        } else {
            this.stringItem.setText("");
        }
    }
}
```

El siguiente paso es **simularlo**. Para ello, se puede depurar (Run -> Debug) o ejecutarlo directamente (Run -> Run). Nos aparecerá un menú como el mostrado en la Figura 32.

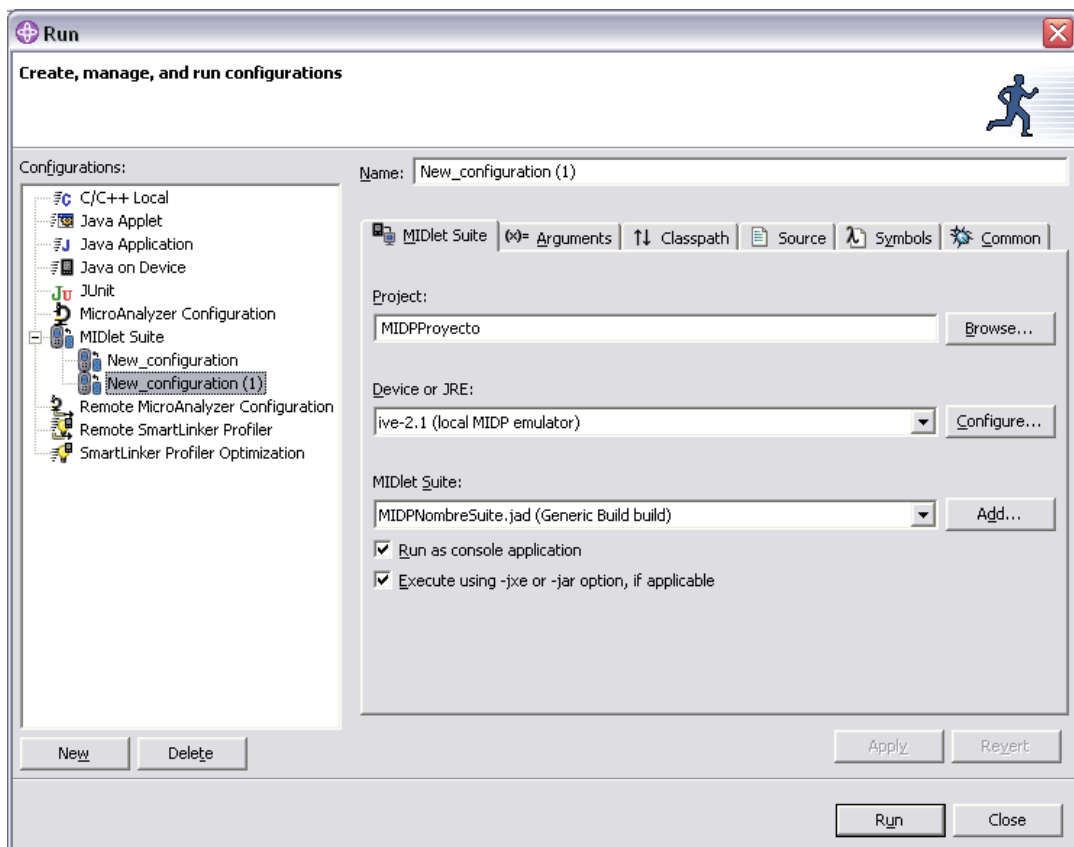


Figura 32. Configuración para la ejecución

Desde esta ventana podemos crear, gestionar y ejecutar lo que se denomina **configuraciones**. Cada una de estas configuraciones está asociada a un tipo de aplicación, como se observa en el listado de la izquierda. Así, podemos tener configuraciones que se refieren a C/C++ Application, Java Applet, Java on Device, o **MIDlet Suite**, siendo ésta última la que nos interesa.

Para crear una nueva configuración, hacemos clic sobre **MIDlet Suite**, y luego clic en el botón **New** (también se puede hacer doble clic), y se abre el panel de la derecha con las distintas opciones de configuración.

La primera pestaña, **MIDlet Suite**, nos permite elegir el proyecto al que pertenece la aplicación que queremos ejecutar (**Browse...**), el dispositivo donde deseamos simularlo (por defecto sólo incluye uno local) y los *suites* (.jad) disponibles en el mismo. Los dispositivos pueden ser previamente configurados como se muestra en la sección 6.

En la pestaña **Arguments**, permite indicar los parámetros que queramos pasarle a la máquina virtual de java. En las siguientes pestañas se nos permite cambiar el **classpath**, el **path** hasta los ficheros fuentes, y otras opciones similares. Lo más recomendable para este tipo de aplicaciones es dejar todos estos parámetros en sus valores por defecto, ya que es el mismo *WebSphere* el que se encarga de colocar los ficheros en una estructura de directorios que nos permita su correcta ejecución.

Por último en la pestaña final, **Common**, podemos hacer que la configuración que hemos creado aparezca en las secciones **Debug** o **Run** del editor, de forma que las tengamos más accesibles para sucesivas ejecuciones (ver Figura 33). Además, también existe la posibilidad de cambiar la perspectiva al ejecutar o depurar la aplicación.

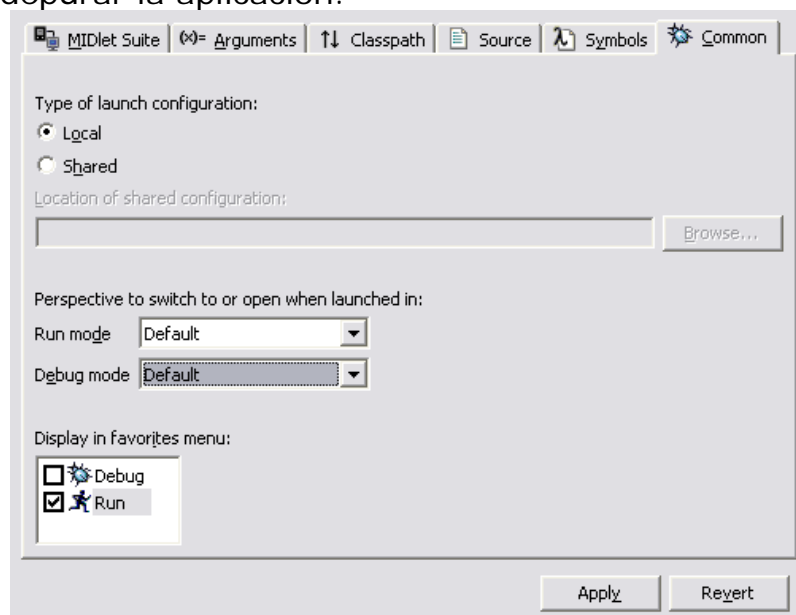


Figura 33. Pestaña Common

Cuando terminemos de realizar nuestra configuración, hacemos clic en **Apply** para que se nos guarden los cambios, y por último sobre **Run**. Se nos abrirá una ventana (ver Figura 34) en la que podemos comprobar si el comportamiento es el deseado en el dispositivo elegido.



Figura 34. Apariencia del Emulador

Si hemos incluido la configuración creada en los menús de **Run** o **Debug**, podremos acceder a ella con más comodidad haciendo clic sobre ellos utilizando la barra de herramientas. Además, con los comandos abreviados **F11** y **Ctrl-F11**, podemos depurar y ejecutar (respectivamente) la última configuración utilizada (ver Figura 35).

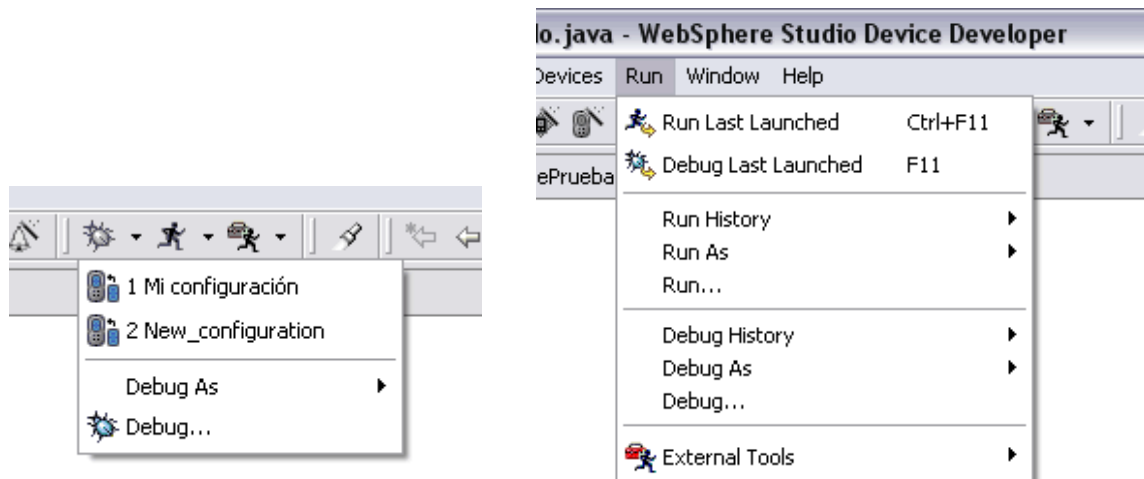




Figura 35. Opciones de ejecución y depuración

6. Añadiendo nuevos emuladores

Esta sección describe como añadir nuevos emuladores para probar nuestras aplicaciones, con el objetivo de simularlas en un entorno similar al de destino. Como ejemplo utilizaremos el MIDP SDK de la *Nokia Series 60*, pero el procedimiento es similar para todos los emuladores, *Nokia Series 80*, *PalmOS*, *Wireles Toolkit*, etc.

- a) Lo primero que tenemos que hacer, es descargar el emulador deseado (en este caso [Nokia Series 60 MIDP SDK²](#)) e instalarlo en el ordenador.

 En la mayoría de los casos los emuladores se encuentran integrados dentro de un SDK, por lo que habría que descargar el SDK completo.

- b) Seleccionamos la opción **Device -> Configure** () y hacemos clic sobre **UEI Emulator Devices** (Ver Figura 36).

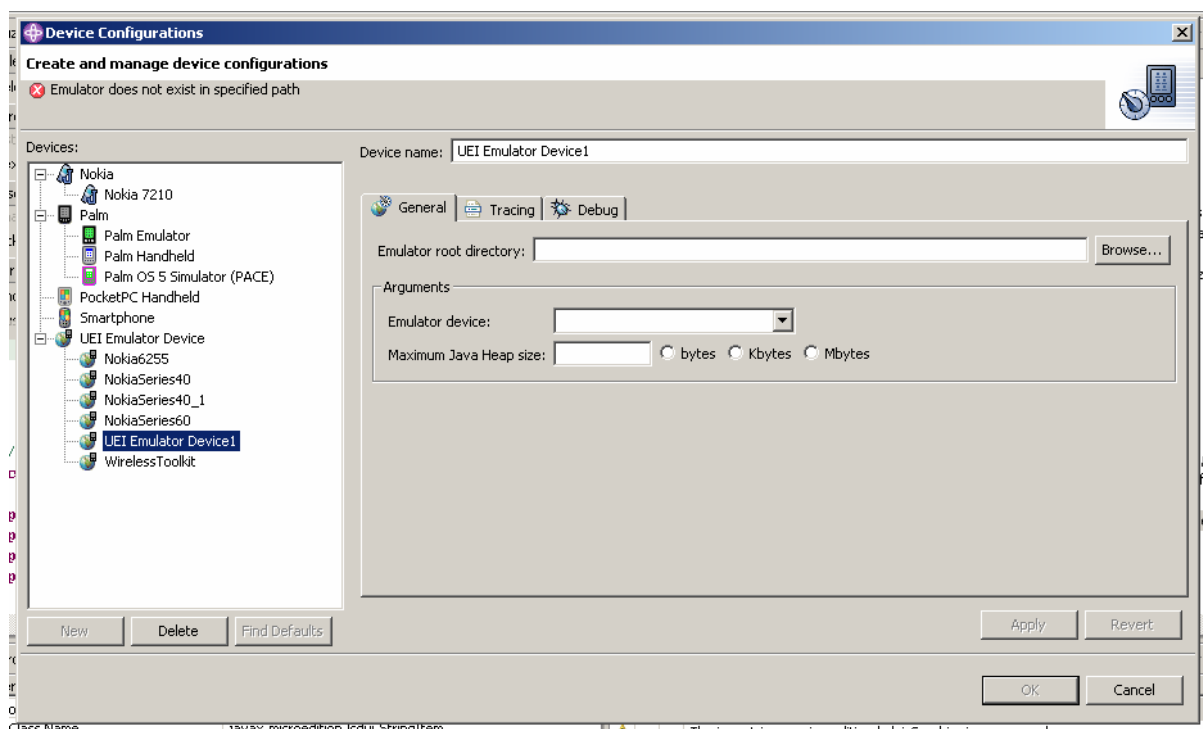


Figura 36. Configuración de dispositivos

- c) Hacemos clic en el botón **New**. Las opciones que debemos configurar en este punto son:
- En la pestaña **General**: Nombre del dispositivo, directorio raíz donde se encuentra instalado el emulador, por ejemplo, "C:\Nokia\Devices\Series_60_MIDP_Concept_SDK". Por último, el

² El SDK de la Series 60 de Nokia nos permite crear aplicaciones para los teléfonos de Nokia 7650, 3600, 3650, 6600, 7610, entre otros.

nombre del dispositivo (asigna uno por defecto) y el tamaño máximo.

- En la pestaña *Tracing*: Podemos seleccionar de forma opcional los parámetros de traza que serán pasados al emulador cuando el *MIDlet* es ejecutado.
- En la pestaña *Debug*: Opciones de conexión como el mecanismo de transporte, el nodo y el puerto. Estas opciones no deben modificarse.

d) Aceptamos la configuración.

Ahora para ejecutar nuestra aplicación podremos configurar su ejecución en el nuevo emulador (ver Figura 37).

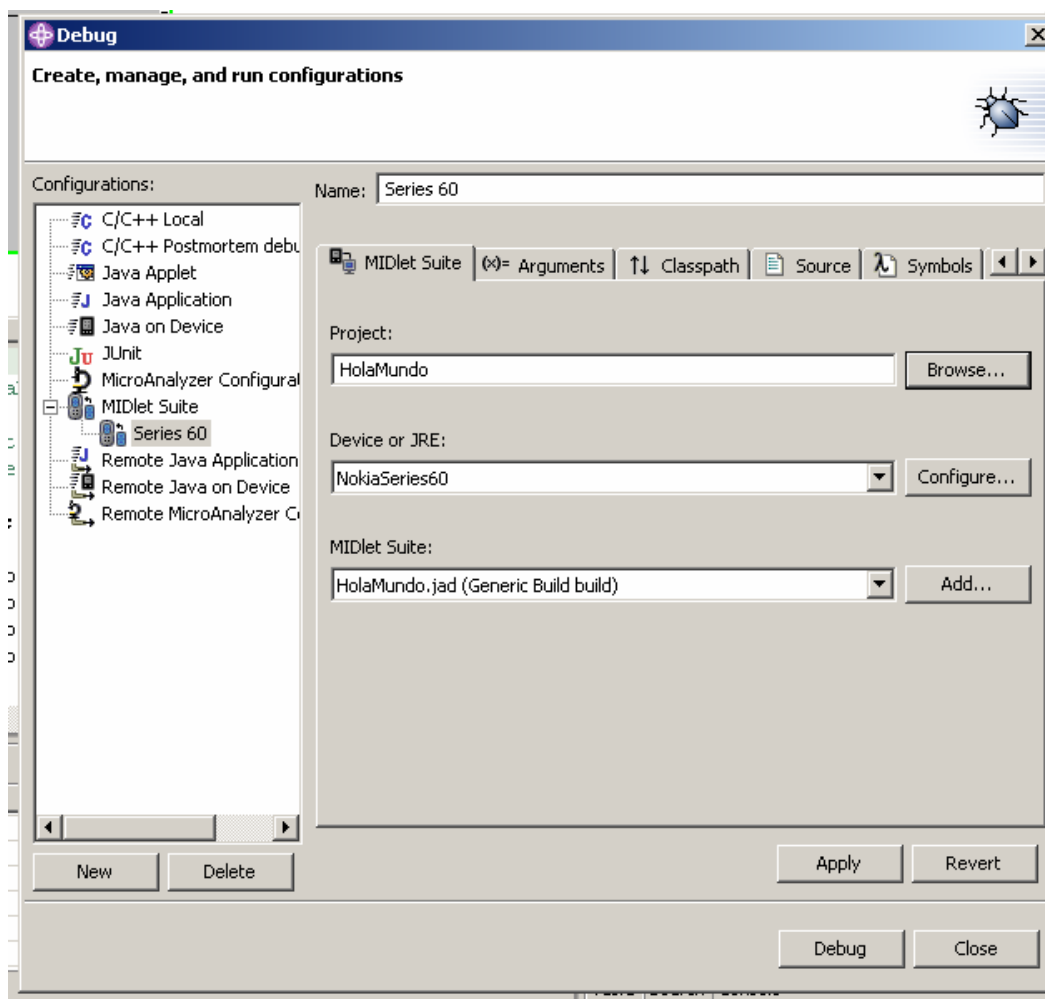


Figura 37. Configurando la ejecución con el nuevo emulador