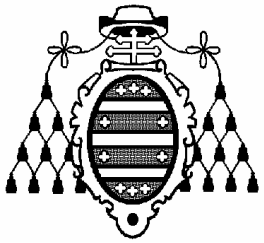
UNIVERSIDAD DE OVIEDO



ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA EN INFORMÁTICA DE OVIEDO

INFORMÁTICA MÓVIL

Práctica 1:
Diseño de aplicaciones con Java ME

(Java Micro Edition)

CURSO 2007-2008

TABLA DE CONTENIDOS

1. OBJETIVOS

Esta sesión práctica pretende introducir al alumno en el desarrollo de aplicaciones en J2ME mediante un sencillo ejemplo de aplicación, que será ampliado a lo largo de esta sesión y posteriores.Al finalizar, el alumno deberá dominar el API de alto nivel para implementación de interfaces de usuario en J2ME y comience a familiarizarse con el uso del API RMS para almacenamiento persistente de datos. Además, se comentarán algunas particularidades del emulador de dispositivos MID, como el acceso por parte del usuario a su sistema de ficheros.

2. HOLA MUNDO

Para comenzar, se procede a la confección del programa inicial típico: el famoso *HolaMundo*. Este programa se limita a mostrar de alguna forma un texto por pantalla. Este ejemplo ya fue visto en la clase de teoría, pero en este caso se utilizará el ejemplo que incluye el propio NetBeans (el cual varía ligeramente respecto al de clase). Para abrir el proyecto que contiene dicho ejemplo, es necesario seleccionar *File→New project* en el menú principal de la herramienta y, en el diálogo emergente, elegir la categoría *Mobility* y el proyecto *MIDP Application*. Dejando las opciones por defecto, creará un MIDlet que implementa un ejemplo tipo *Hola Mundo*. En el cuarto paso del diálogo, selecciona la creación de diferentes configuraciones del proyecto desde los patrones (*templates*) existentes, marcando todos los patrones posibles. Eso permitirá ejecutar la aplicación en los distintos emuladores disponibles.

Aprovecha este ejemplo para aprender cómo ejecutar la aplicación en los distintos emuladores y a depurar paso a paso la aplicación (ver las distintas opciones de la opción *Run* en el menú principal de NetBeans). Es necesario ejecutar las aplicaciones en la mayor cantidad de emuladores y dispositivos reales, pues la creación de software para móviles es complicada debido a la heterogeneidad de dispositivos en el mercado.

Es de vital importancia además, tener en cuenta la configuración y el perfil soportados por un emulador y un móvil en concreto, para conocer qué funcionalidad nos proporciona el API disponible. Las distintas configuraciones de un proyecto permiten modificar la configuración, el perfil y los paquetes opcionales de la plataforma destino. Comprueba los valores de configuración, perfil y paquetes opcionales de cada una de las configuraciones de proyecto disponibles en el midlet de ejemplo (en la pestaña *Projects*, selecciona el nombre del proyecto con el botón derecho del ratón y selecciona el elemento *Properties* del menú desplegable.).

3. CREACIÓN DE UNA APLICACIÓN

Para ilustrar la creación de una aplicación, se creará un pequeño programa que ayudará a ilustrar los pasos típicos de creación de una aplicación con varias pantallas, lista para ser instalada en cualquier MID (Mobile Information Device, esto es, cualquier dispositivo móvil que soporte MIDP).

Para crear estos programas y conocer todos los elementos disponibles en las APIs de Java ME es interesante tener disponible documentación acerca de la jerarquía de clases que ofrecen MIDP, CLDC o CDC, o los distintos paquetes opcionales. NetBeans incluye una ayuda interesante en este sentido (ver en *Help* algunos elementos del menú como *JavaDoc References* o *Quick Start Guide*).

El programa que se propone es una pequeña libreta de direcciones en la que se almacenará información acerca de personas de nuestro entorno (amigos, compañeros de estudios, de

trabajo, etc). Esta utilidad tan sencilla servirá para mostrar la forma de diseñar aplicaciones dentro del entorno de J2ME.

La aplicación constará de dos pantallas:

- **Pantalla principal**: un formulario en el que se podrá navegar a través de la información general de las distintas personas cuyos datos están almacenados en la libreta de direcciones.

- **Pantalla de detalle**: un formulario en el que se podrá consultar la información detallada de una persona concreta.

En primer lugar, se comenzará a crear la aplicación y, ya que se compone de dos pantallas, la crearemos como una suite de MIDlets (*MIDlet suite*). Una suite de MIDlets consta de los ficheros .class que componen la aplicación y, opcionalmente, dos archivos con metadatos relativos a la misma. Se suele distribuir en dos ficheros, que permitirán la instalación de la aplicación mediante los servicios del AMS (Application Management Software o Gestor de Aplicaciones): uno con extensión .jad (Java Application Descriptor) y otro con extensión .jar (Java ARchive). Ambos ficheros son creados en el subdirectorio *dist* dentro del directorio del proyecto (por defecto, *c:\Documents and Settings\NombreUsuario\NombreProyecto*).

El fichero JAD contiene los atributos que el AMS utilizará para gestionar el ciclo de vida del MIDlet y los atributos específicos de la aplicación que la propia suite de MIDlets utilizará. El fichero JAR es un archivo comprimido basado en una variante del formato zip que incluye:

- Los archivos .class que conforman la suite de midlets.

- Otro opcional, llamado *META-INF*, que incluye un manifiesto (fichero *MANIFEST.MF*). Este fichero contiene metainformación acerca del resto de ficheros incluidos en el fichero JAR y metainformación útil para el AMS y para la suite.

Es inevitable pensar que, al fin y al cabo, tanto el manifiesto (*MANIFEST.MF*) como el descriptor (fichero con extensión JAD) tienen un propósito similar. Esto es absolutamente cierto y, por ello, es necesario definir una política para localizar atributos y para resolver cualquier diferencia entre dos posibles valores distintos para un mismo atributo que se encuentre en ambas localizaciones. De todas formas, la creación del fichero descriptor (JAD) y del fichero comprimido (JAR) de forma automática mediante NetBeans garantizará la coherencia de los metadatos presentes tanto en el fichero descriptor como en el manifiesto.

Para comenzar a crear la suite de MIDlets, crea un proyecto vacío (no basado en el ejemplo *HolaMundo*). Sobre el elemento raíz del proyecto en la pestaña Project, puedes pulsar con el botón derecho y utilizar el elemento *New* del menú desplegable para añadir nuevos Midlets y paquetes Java (entre los que distribuir los distintos .class que conformen las aplicaciones). **Intenta programar la aplicación editando el código fuente y no utilizando el diseñador visual de aplicaciones de NetBeans.**

Un fichero .java contiene, al ser creado por el IDE de NetBeans, el esqueleto básico de un MIDlet:

- Directiva **package** para que el MIDlet quede incluido en el paquete de la aplicación.

Un paquete es un conjunto de ficheros .class que pertenecen a una misma aplicación y que se distribuye típicamente en un fichero JAR. Todos los .class que compondrán nuestra aplicación incorporarán esta directiva.

- Directiva **import** para importar los espacios de nombres típicos en aplicaciones J2ME basadas en MIDP (*javax.microedition.midlet*, para la gestión del ciclo de vida de la aplicación; y *javax.microedition.lcdui*, para la gestión de la interfaz gráfica de la aplicación).

- La definición de la **clase principal** que compone el MIDlet (extiende la clase *javax.microedition.midlet.MIDlet*) y los métodos públicos siempre presentes en un MIDlet:
  - *startApp()*, donde se implementarán las acciones que tienen lugar cuando una aplicación entra en modo **ACTIVO**.
  - *pauseApp()*, método que incluye el código a ejecutar cuando la aplicación pasa a modo **PAUSA**.
  - *destroyApp()*, método encargado de llevar a cabo las acciones necesarias cuando la aplicación pasa a modo **DESTRUIDO**.

3.1. EJERCICIO

Basándose en los ejemplos ya vistos, el primer paso será incluir un formulario como **pantalla principal** de la aplicación, en el que se mostrará (**sin poder modificar**) el **nombre, apellidos y mote** (si lo tiene) de una persona, así como **cuatro botones que permitirán navegar** (avanzar y retroceder dentro de la lista de personas y acceder automáticamente a la primera y la última persona con información almacenada en la futura "base de datos"). También existirá un **botón adicional para acceder a la pantalla de detalle** (aún no creada), en la que se ofrecerá información adicional. **Observa los ejemplos de aplicaciones vistos en clase para recordar cómo construir una aplicación utilizando las clases del API de alto nivel de interfaz de usuario**.

En la pantalla de detalle (segundo formulario de la aplicación) se ofrecerá, además de la información de la persona que se estaba mostrando en el formulario principal, información adicional. Esta información adicional **nombre, apellido1, apellido2, mote, email, número de teléfono de casa, número de teléfono móvil, foto, profesión, edad y afinidad** (un valor entre 0 y 10, que indique lo "bien que nos cae" esa persona).

La novedad en esta pantalla de detalle es que se podrá editar toda la información que aparece en la pantalla y se podrá volver a la pantalla principal de navegación (lo cual implica guardar el estado actual de los datos de la persona en cuestión). No permitirá salir de la aplicación: para ello habrá, previamente, que volver a la pantalla principal de navegación.

En esta primera versión se crearán simplemente los elementos gráficos que compondrán la pantalla principal de navegación, sin darles valores que mostrar ni definir su comportamiento.

PISTAS

- Intenta realizar una versión mínima del software, pruébala y añade nueva funcionalidad poco a poco.

- De momento, no se incluirá la foto de los contactos.

- Sólo puede haber dos *Command*s en pantalla:

    - **En la pantalla principal**, uno será usado para salir de la aplicación; los otros *Command*s conformarán un menú. El *Command* para salir será del tipo *CommandEXIT*; los *Command*s del menú desplegable, del tipo *Command.SCREEN*. Puedes jugar con la prioridad de los Commands del menú deplegable para asociar distintas teclas de acceso rápido del teclado del móvil.
    - **En la pantalla de detalle**, habrá un *Command* para volver a la pantalla principal guardando los datos introducidos (*Command.OK*) y otro para volver a la pantalla principal sin guardarlos (*Command.CANCEL*).

- Recuerda cómo atender a los eventos de los *Command*s (interfaz *CommandListener*).

- Hay que limitar las posibilidades de entrada en los TextField utilizando las restricciones al llamar al constructor (*ANY*, *PHONENUMBER*, etc.).

- En los ejemplos de clase, aparece un ejemplo de aplicación con más de un formulario que muestra cómo hacer visible un formulario de entre todos los disponibles para la aplicación (método *setCurrent()*).

- Prueba a actualizar (o no, según selecciones guardar o cancelar) la información mostrada de la pantalla principal a partir de la información introducida en la pantalla de detalle.

4. NAVEGACIÓN ENTRE DISTINTOS CONTACTOS

La información relativa a cada una de las personas será modelada mediante una nueva clase llamada *Contacto*, que será implementada de la siguiente forma:

```
class Contacto {
   String nombre, apellido1, apellido2, email, numTfnoCasa, numTfnoMovil, mote;
   String foto, profesion;
   int edad, afinidad;
}
```

Para introducir esta clase en la aplicación, se creará su propio fichero .java, siendo necesario añadirlo a la suite para "enlazarlo" con el único fichero .java que hasta el momento tenía la aplicación. De esta manera, se pretende ilustrar cómo una aplicación puede estar compuesta de distintos ficheros .java (.class tras la compilación a bytecodes), sin que necesariamente cada uno de ellos implemente un MIDlet. Esto se llevará a cabo pulsando con el botón derecho sobre el elemento raíz del proyecto (o sobre el paquete Java apropiado) en la pestaña *Project*, y seleccionado *New→Java class*.

Llamaremos a la nueva clase *Contacto* y la declararemos como se mostró anteriormente. Además, crearemos un constructor que admita como parámetros 4 cadenas de texto, correspondientes a los miembros *nombre*, *apellido1*, *apellido2* y *mote* de la clase. Es muy

interesante que el constructor inicialice el resto de miembros de la clase con valores por defecto (por ejemplo, con la cadena vacía o con valor 0, en el caso de valores enteros). De esta forma, se facilita mostrar valores no inicializados expresamente en la pantalla de detalle. Opcionalmente, podríamos sobrecargar el constructor con tantas versiones consideremos necesarias, aunque de momento será suficiente con este único constructor.

Entre los miembros de la clase MIDlet creada en el ejercicio anterior (*ContactosMIDlet*), añadiremos:

```
private static final int MAX_CONTACTOS=10;
private Contacto[] listaContactos=null;
```

Y en la primera línea del constructor del MIDlet, justo al principio, introduciremos la línea:

```
listaContactos = new Contacto[MAX_CONTACTOS];
```

De esta manera, agregaremos en memoria una lista de diez contactos. A continuación de la línea anterior, introduciremos el código para inicializar los datos de diez personas, que serán mostrados posteriormente en pantalla:

```
int i;
for (i=0;i<listaContactos.length;i++)
{
        listaContactos[i] =new Contacto("Pepe"+i.toString(), "Lopez"+i.toString(), "Perez"+i.toString(), "Pepin"+i.toString());
}
```

Añade el código necesario para que los botones nos hagan navegar entre los datos de las 10 personas y que el recorrido sea circular (cuando se ven los datos del décimo contacto, si se avanza, se accede a los datos del primer contacto; y lo mismo en el orden inverso). También se podría decidir un comportamiento alternativo, y que no tenga efecto el comando de avance cuando se estén visualizando los datos del décimo contacto, ni tampoco el comando de retroceso cuando se estén visualizando los datos del último contacto.

Comprueba también que los valores numéricos no excedan los límites permitidos: edad entre 0 y 99 años y afinidad entre 0 y 10. Muestra una alerta cuando alguno de los dos valores exceda el rango y, obviamente, no actualices el contacto actual en memoria con ninguno de los valores en el formulario de detalle. Recuerda que no sólo se trata de crear un objeto *Alert*, sino que debes hacer que sea el objeto *Displayable* que se muestre en pantalla para que aparezca (igual que ocurre en los formularios). Juega con el temporizador de la alerta para que ésta sea modal o no modal, según desees, y con el tipo de alerta para utilizar distintos sonidos de aviso. No utilices ninguna imagen para la alerta (parámetro ***null*** en el constructor de *Alert*).

Llegado este instante, efectúa una copia del fichero *.java* para guardar el código de este ejemplo, pues a continuación se destruirá parte de lo realizado.

5. AÑADIENDO UN MIDLET A LA SUITE

A continuación, vamos a añadir un nuevo MIDlet a la suite. Dicho MIDlet va a constar de un único formulario, idéntico al formulario de detalle del MIDlet creado en el apartado anterior. Su comportamiento será idéntico, pero añadirá en un *RecordStore* los datos de un contacto, creando el *RecordStore* si es necesario. El nombre del nuevo MIDlet puede ser, por ejemplo, *AnadeContacto*.

Rellena el esqueleto del nuevo MIDlet de manera que presente un aspecto idéntico al del formulario de detalle del primer MIDlet. En esta ocasión, los dos *Command* servirán para guardar un nuevo contacto y para salir del MIDlet. Recuerda que la clase que modela a nuestro nuevo MIDlet *AnadeContacto* debe implementar la interfaz *CommandListener*.

El formulario debe comprobar también los valores introducidos de edad y afinidad. Almacenará los datos en un *RecordStore* que se abrirá como "*misContactos*". Para escribir la información, usa (además de la clase RecordStore) las clases *java.io.ByteArrayOutputStream* y *java.io.DataOutputStream*, tal y como se mostró en clase de teoría.

De esta forma, la pantalla inicial al ejecutar en un dispositivo MID la suite será una pantalla estándar de selección del MIDlet que se desea ejecutar de entre todos los que componen la suite. Al salir del MIDlet, el AMS (Application Management Software, explicado en los apuntes) vuelve a mostrar de nuevo la pantalla de selección de MIDlet.

Observa, tras ejecutar el MIDlet *AnadeContacto*, que NetBeans crea un nuevo archivo llamado *run_by_class_storage_mis#Contactos.db*. Este archivo simula el *RecordStore* que se habría creado en la memoria de nuestro móvil real. En concreto, puesto que venimos utilizando el emulador *Default Color Phone*, incluido en el *Wireless Toolkit 2.2*, el directorio en el que debería estar dicho archivo es:

```
C:\Documents and
Settings\USUARIO\.netbeans\5.5\emulators\wtk22_win\emulator\wtk22\appdb\DefaultColorPhone
```

Es posible que en ese directorio aparezca un archivo llamado *in.use*, que indica que el *RecordStore* está en uso. Hasta que no se cierre el acceso al *RecordStore*, este archivo no será borrado y ningún otro MIDlet podrá acceder a él.

6. MODIFICANDO EL MIDLET CONTACTOSMIDLET

A continuación, vamos a modificar el MIDlet inicial para que, en lugar de mostrar/modificar los datos de contactos tomados de un vector de elementos de la clase *Contacto*, utilice la información almacenada en el *RecordStore*.

Podría enfocarse este problema desde dos perspectivas:

- La primera, abriría el *RecordStore misContactos*, detectaría el número de registros y daría un tamaño al vector *listaContactos* igual a ese número. Posteriormente, se recorrería el *RecordStore*, asignando el contenido de cada registro a cada elemento del vector. Al salir de la aplicación, se borraría el *RecordStore* misContactos, creando uno nuevo con la información en el vector. Con este enfoque, puede ser útil un nuevo constructor de la clase *Contacto* que

admita como único parámetro un registro. Intenta implementar este enfoque, guardando el resultado como *ContactosMIDlet-v3.java*.

- La segunda perspectiva, sustituiría el vector *listaContactos* por un único objeto de la clase *Contacto* que almacenará el contacto actual, realizando la navegación sobre el propio *RecordStore*. Utiliza los métodos de las clases *RecordStore* y *RecordEnumeration* vistos en clase de teoría para recorrer el *RecordStore* y para modificar el contenido de un registro en concreto (método *RecordStore.setRecord*). Intenta implementar este enfoque, guardando el resultado como *ContactosMIDlet-v4.java*.