



# **Instituto Politécnico Nacional**

## **Escuela Superior de Computo**



Nombre del Alumno:

- Huerta Cortes Alan Antonio

Asignatura: Application Development For Mobile Devices

Profesor: Cifuentes Álvarez Alejandro Sigfrido

Reporte de la Agenda con Midlets

Fecha de entrega: 25 de febrero del 2019

Grupo: 3CV5

# Índice

<b>1.Objetivo.....</b>	<b>3</b>
<b>2.Conceptos previos.....</b>	<b>3</b>
2.1 ¿Qué es un Midllet?.....	3
2.2 Registros.....	3
2.2.1 Interfaces.....	4
2.2.2Clase RecordStore .....	4
<b>3.Desarrollo.....</b>	<b>5</b>
<b>4.Conclusiones.....</b>	<b>10</b>
<b>5.Bibliografia.....</b>	<b>11</b>

## 1.Objetivo:

Realizar una agenda en Midlets con ayuda de registros para almacenar los datos proporcionados por el usuario, simulando una base de datos en este dispositivo, la cual solo podrá ingresar, actualizar y eliminar datos.

## 2.Conceptos previos

### 2.1 ¿Qué es un Midlet?

Un "MIDlet" es un programa desarrollado con el lenguaje de programación Java para dispositivos embebidos (que se dedican a una sola actividad), más específicamente para la máquina virtual Java Micro Edition (Java ME). Generalmente son juegos y aplicaciones que corren en un teléfono móvil. Está desarrollada bajo la especificación MIDP (Perfil para Información de Dispositivo Móvil).

Requisitos:

Requiere un dispositivo que implemente Java ME y MIDP para correr. Como otros programas desarrollados en Java, tienen la característica "Escribir una vez, ejecutar en cualquier parte" ("Write once, run anywhere"). Para programar MIDlets se puede obtener el Sun Java Wireless Toolkit o NetBeans con la extensión Mobility Pack. Para la versión de distribución son necesarios dos archivos, el archivo .jar conteniendo el bytecode del programa y un archivo .jad que describe los contenidos del archivo .jar.

Un MIDlet tiene que cumplir los siguientes requisitos para poder correr en un teléfono móvil:

- La clase principal necesita ser una subclase de `javax.microedition.midlet.MIDlet`.
- El MIDlet necesita ser empacado dentro de un archivo .jar.
- El archivo .jar necesita ser preverificado usando un preverificador.
- En algunos casos, el archivo .jar necesita ser firmado digitalmente por un proveedor de teléfonos móviles.

Al crear un MIDlet se genera un archivo descriptor con extensión .jad, que contiene todos los recursos que se están utilizando para que la aplicación se ejecute.

### 2.2 Registros

Para que una aplicación pueda almacenar datos en el dispositivo móvil, debe crear un almacén de registros ("RecordStore"). Es una base de datos especial donde la unidad de información son los registros ("Records"). Una vez añadimos un registro a un almacén de registros, a éste se le asigna un identificador único (recordid). Los almacenes de registros pueden ser compartidos por los MIDlets dentro de una

misma Suite de Midlets, pero un MIDlet no puede acceder nunca a un almacén que no haya sido creado por un MIDlet que no esté dentro de su Suite.

Cada almacén de registros tiene un identificador que es una cadena de 1-32 caracteres Unicode, así pues, la combinación (nombre del almacén, MIDlet Suite) identifican unívocamente a un almacén de registros. Un MIDlet perteneciente a una Suite, puede acceder por tanto a cualquier almacén de datos de cualquiera de los otros MIDlets de su propia suite, pero sin embargo, no puede obtener ningún tipo de información de los almacenes de registros de cualquier otra Suite. Igualmente cabe notar que ninguna aplicación en un dispositivo móvil que no sea de Java tampoco puede acceder a los datos almacenados por los MIDlets de J2ME.

Un almacén de registros tiene cero o más registros, donde cada registro es un array de bytes con un identificador único asociado que permite identificarlo unívocamente dentro del almacén. Los identificadores se les asignan a los registros empezando por 1 y dando al siguiente registro insertado un identificador igual a 1 + el identificador dado al último registro añadido anteriormente. El borrado de los identificadores no es relevante en la secuencia dada a la inserción de registros.

### **2.2.1 Interfaces**

**RecordComparator** .- Nos ayudará a comparar dos registros para la obtención ordenada de todos los registros de un almacén.

**RecordEnumeration** .- Nos devolverá una enumeración de los registros de un almacén de datos. Nos servirá para recorrer todos los registros de un almacén.

**RecordFilter** .- A la hora de obtener una enumeración nos permitirá filtrar los registros que recorreremos.

**RecordListener** .- Monitorizador de eventos que usaremos para controlar cambios en los registros.

### **2.2.2 Clase RecordStore**

Esta clase es la clase central del paquete rms, que nos representará una colección de registros. Podremos realizar operaciones para abrir, cerrar y borrar almacenes, así como operar con los registros de éstos, añadiendo, eliminando, modificando y enumerando los registros dentro de un almacén.

Métodos de la clase:

**openRecordStore()** .- Abre el almacén de registros

**closeRecordStore()** .- Cierra el almacén de registros

**deleteRecordStore()** .- Borra el almacén de registros

**getName()** .- Recupera el nombre del almacén de registros

getNumRecords() .- Recuperar el número de registros del almacén

addRecord() .- Añadir un registro al almacén de registros

getRecord() .- Recupera un registro del almacén de registros

deleteRecord() .- Borra un registro del almacén de registros

enumerateRecord() .- Obtiene un enumeration del almacén de registros

### 3. Desarrollo

El desarrollo se compone en la codificación de la agenda con registros.

1. Se desarrolla la clase que maneja los registros, en esta clase se implementan los métodos para gestionar los registros de manera eficiente.

```
1  import java.io.*;
2  import javax.microedition.rms.*;
3  public class RMSOps {
4      RecordStore rs;
5      public RMSOps() { }
6      public boolean abrir(String nombreZona) {
7          try {
8              rs = RecordStore.openRecordStore(nombreZona, true);
9              return true;
10         } catch (RecordStoreException e) {
11             e.toString();
12             return false;
13         }
14     }
15     public boolean adicionarRegistro(ByteOutputStream baos){
16         byte[] mensaje; mensaje = baos.toByteArray();
17         try {
18             rs.addRecord(mensaje, 0, mensaje.length);
19             return true;
20         } catch (RecordStoreException e) {
21             e.toString(); return false;
22         }
23     }
24     public boolean adicionarRegistro(String valor) {
25         byte[] mensaje;
26         mensaje = valor.getBytes();
27         try {
28             rs.addRecord(mensaje, 0, mensaje.length);
29             return true;
30         } catch (RecordStoreException e) {
31             e.toString(); return false;
32         }
33     }
```

```

34 public String listarRegistro() {
35     byte[] reg = new byte[120];
36     int tam;
37     String buffer = "";
38     try {
39         for (int i = 1; i <= rs.getNumRecords(); i++){
40             tam = rs.getRecordSize(i);
41             reg = rs.getRecord(i);
42             buffer += "\n" + i + new String(reg, 0, tam);
43         }
44     } catch (RecordStoreException e) {
45     }
46     return buffer;
47 }
48
49 public String[] listarRegistro(int r){
50     byte[] reg = new byte[120];
51     ByteArrayInputStream bais;
52     DataInputStream dis;
53     String[] datos= new String[6];
54     try{
55         reg= rs.getRecord(r);
56         bais = new ByteArrayInputStream(reg);
57         dis = new DataInputStream(bais);
58         datos[0] = dis.readUTF();
59         datos[1] = dis.readUTF();
60         datos[2] = dis.readUTF();
61         datos[3] = dis.readUTF();
62         datos[4] = dis.readUTF();
63         datos[5] = dis.readUTF();
64     } catch (RecordStoreException e){
65     } catch (IOException e){
66         e.printStackTrace();
67     }
68     return datos;
69 }
70
71 public boolean cerrar(){
72     try{
73         rs.closeRecordStore();
74         return true;
75     } catch (RecordStoreException e){
76         e.toString();
77         return false;
78     }
79 }
80
81 public boolean eliminar(int registro){
82     try{
83         rs.deleteRecord(registro);
84         return true;
85     } catch (RecordStoreException e){
86         e.toString();
87         return false;
88     }
89 }

```

```

86 public boolean check(int registro){
87     try{
88         rs.getRecord(registro);
89         return true;
90     } catch (RecordStoreException e){
91         e.toString();
92         return false;
93     }
94 }
95 public boolean editRegistro(ByteOutputStream baos,int indice){
96     byte[] mensaje; mensaje = baos.toByteArray();
97     try {
98         rs.setRecord(indice,mensaje, 0, mensaje.length);
99         return true;
100     } catch (RecordStoreException e) {
101         e.toString(); return false;
102     }
103 }
104 }

```

2. Posteriormente se crea la plantilla en donde se ingresarán los datos y donde se le dará acción al formulario de agregar, eliminar y listar contactos.

#### Declaración de las variables visuales y comandos del midlet

```

3 public class Forma extends Form implements CommandListener {
4     Command g, s, lista,select;
5     TextField nom, ape, tel, cel, dom;
6     Principal p;
7     Alert a;
8     String foto = "/4.png";
9     public Forma(Principal mid) {
10         super("Agenda");
11         p = mid;
12         //foto = "";
13         nom = new TextField("Nombre:", "", 30, TextField.ANY);
14         ape = new TextField("Apellido:", "", 30, TextField.ANY);
15         tel = new TextField("Telefono:", "", 30, TextField.ANY);
16         cel = new TextField("Celular:", "", 30, TextField.ANY);
17         dom = new TextField("Domicilio:", "", 30, TextField.ANY);
18         s = new Command("Salir", Command.EXIT, 1);
19         g = new Command("Guardar", Command.SCREEN, 1);
20         lista = new Command("Contactos", Command.EXIT, 2);
21         select = new Command("Seleccionar Foto", Command.EXIT, 3);
22         append(nom);
23         append(ape);
24         append(tel);
25         append(cel);
26         append(dom);
27         addCommand(g);
28         addCommand(s);
29         addCommand(lista);
30         addCommand(select);
31         setCommandListener(this);
32         setRegistro();

```

## Acción de los comandos.

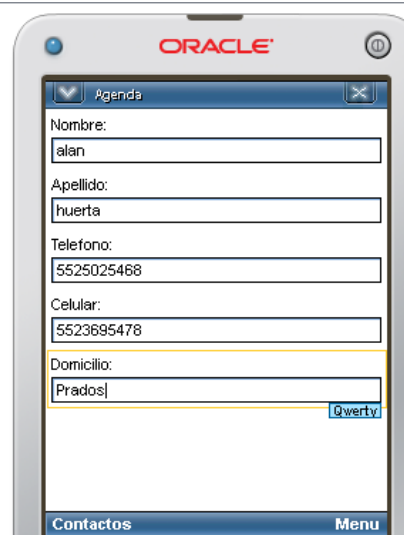
```
public void commandAction(Command co, Displayable di) {
35     if (co == s) {
36         p.destroyApp(true);
37     }
38     else if (co == g) {
39         RMSOps rmso = new RMSOps();
40         rmso.abrir("ZonaJavaZone");
41         try {
42             ByteArrayOutputStream baos = new ByteArrayOutputStream();
43             DataOutputStream dos = new DataOutputStream(baos);
44             dos.writeUTF(foto);
45             dos.writeUTF(nom.getString());
46             dos.writeUTF(ape.getString());
47             dos.writeUTF(tel.getString());
48             dos.writeUTF(cel.getString());
49             dos.writeUTF(dom.getString());
50             dos.close();
51             rmso.adicionarRegistro(baos);
52             rmso.cerrar();
53             } catch (Exception ioe) {
54                 ioe.printStackTrace();
55             }
56             a = new Alert("NOTA", "Guardado", null, AlertType.CONFIRMATION);
57             a.setTimeout(5000);
58             p.d.setCurrent(a, this);
59             p.lc.constructList();
60             p.d.setCurrent(p.lc);
61             nom.setString("");
62             ape.setString("");
63             tel.setString("");
64             cel.setString("");
65             dom.setString("");
66         }
67     else if(co == lista){
68         p.lc.constructList();
69         p.d.setCurrent(p.lc);
70     }
71     else if(co == select){
72         p.d.setCurrent(p.lf);
73     }
74 }

public void setRegistro(){
75     RMSOps rmso = new RMSOps();
76     rmso.abrir("ZonaJavaZone");
77     try {
78         ByteArrayOutputStream baos = new ByteArrayOutputStream();
79         DataOutputStream dos = new DataOutputStream(baos);
80         dos.writeUTF("/4.png");
81         dos.writeUTF("");
82         dos.writeUTF("");
83         dos.writeUTF("");
84         dos.writeUTF("");
85         dos.writeUTF("");
86         dos.close();
87         rmso.adicionarRegistro(baos);
88         rmso.cerrar();
89         } catch (Exception ioe) {
90             ioe.printStackTrace();
91         }
92     }
93 }
94 }
```



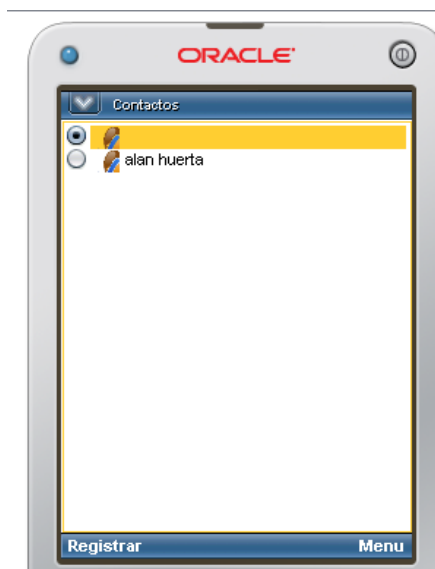
### 3. Ejecución del proyecto.

- Ingresando Datos para su almacenamiento en la agenda



A screenshot of a mobile application interface titled 'Agenda'. The form contains the following fields: 'Nombre:' with the value 'alan', 'Apellido:' with the value 'huerta', 'Telefono:' with the value '5525025468', 'Celular:' with the value '5523695478', and 'Domicilio:' with the value 'Prados'. A 'Qwerty' keyboard icon is visible next to the 'Domicilio' field. At the bottom of the screen, there are two buttons: 'Contactos' and 'Menu'.

- Datos almacenados Correctamente (aunque se cierre la aplicación deben permanecer los datos cuando se vuelva a ejecutar)



- Eliminación de los contactos.



## 4.Conclusiones

La Realización de esta Practica se logro satisfactoriamente ya que el código proporcionado por el profesor a cargo fue de mucha ayuda para manipular de forma correcta los registros y me di cuenta como simular un repositorio de datos en los midlets lo cual es de gran ayuda para manipular datos.

## 5. Bibliografía

<http://flanagan.ugr.es/J2ME/MIDP/persistencia.htm>

<http://leo.ugr.es/J2ME/persistencia.pdf>

<http://www.jtech.ua.es/apuntes/ajdm2010/sesiones/sesion02-apuntes.pdf>

<http://www.jtech.ua.es/apuntes/ajdm2010/sesiones/sesion02-apuntes.html>