Articulo de Dispositivos móviles, Cuarta Parte:

Por: Luís Alberto Jumbo Flores
Loja - Valle de Tecnología.

1. Objetivo:

- Proporcionar conocimientos generales sobre la construcción de aplicaciones para dispositivos móviles en J2ME, incorporando mecanismos de persistencia de datos usando RMS (Record Manager Stored).

2. Introducción:

Continuando con la entrega de nuevos artículos, tengo a bien presentarles el siguiente, cuya finalidad es la de incrementar los conocimientos de J2ME, así como el uso de IDE Netbeans.

Hasta ahora hemos visto muchas de la facilidades que presta netbeans en conjunto con J2ME, para el desarrollo de aplicaciones móviles como: creación de pantallas, navegabilidad, e inserción de imágenes, pero los datos que ingresamos a través de un formulario debería persistir en el dispositivo móvil; es este elemento es el que vamos a abordar en el presente artículo.

El Software que se empleó para este trabajo es NetBeans 5.5, (Ud. puede descargar de sitio oficial http://www.**netbeans**.org/**download**s/) deberá descargar los productos: **NetBeans IDE**, **Mobility Pack**, **Profiler and Application Server bundle.**

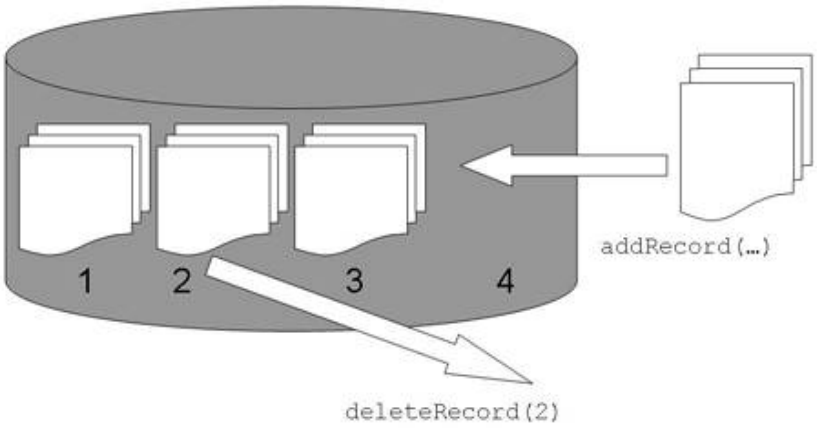
3. Continuación del proyectos anterior

En nuestro artículo anterior creamos pantallas para que acepten el ingreso de datos, pero no incorporamos ningún mecanismo de almacenamiento. Antes de continuar con este artículo vamos a definir algunos conceptos relevantes.

Record Management Store
Al igual que las aplicaciones desktop, los Midlets necesitan facilidades para almacenar permanente datos. La persistencia de datos en los dispositivos móviles se encarga RMS y la clase que la respalda es `RecordStore`, el paquete que contiene esta clase se llama `javax.microedition.rms`.

Los MIDlets leen y escriben registros a través de un identificador llamado recordId. Cuando un nuevo registro es agregado para almacenar registro, esto es asignado a nuevo recordId. MIDlets debería acceder registros en cualquier orden.

Clase RecordStore
Proporciona un acceso completo a métodos que permitan: crear almacenamiento de registros abrir/cerrar almacenamientos, insertar registros, actualizar registros, obtener registros, y borrar registros. Todo el almacenamiento puede interpretarse como un conjunto de espacios a un registro almacenado.



La clase alberga muchos metodos que son usuados se detalla a continuación:
Un método estático llamado openRecordStore es usado para crear almacenamientos. El metodo closeRecordStore sirve para cerrar un Almacenamiento.

Para agregar datos se usa el método addRecord, y para modificar se puede hacer con la llamada al método setRecord .
Para borrar deleteRecord, .
Para obtener el próximo IdRecord se usa el metodo nextRecord.

La clase también dispone de clases exception para manejar la excepciones en el uso de los métodos, esto se explicara mejor con la contruccion de la clase **RMSadmin**

Como el proyectos ya esa creado lo que vamos hacer es a crear una clase llamada RMSadmin, para ello debe ir a la ventana Project en <default package>, click derecho y luego New/JAVA class, en la ventana que se presenta escribe el nombre de la clase en caja correpondiento y luego finísh.

Vamos a incorporar algunas propiedades y métodos que incorporan mecanismos para insertar, consultar y listar registros

Abriendo y cerrando un RecordStore.

Primero debemos incorporar una propiedad privada llamada rs de la clase RecordStore,

```
    private RecordStore rs;
```

Y debemos incorporar un método para que crear y/o abrir el recordstore, a través del siguiente código

```
    private void CrearRecordStore(){
        try{
            rs=RecordStore.openRecordStore("myRs",true);
        }
        catch(RecordStoreException e){
            //throw(new RecordStoreException(e));
        }
    }
```

Se usa el método openRecordStore, con parámetros que representan el nombre de RecordStore, y el valor trae, que indica, la construcción del recordstore si no existiera.
Además debemos agregar un método que permita cerrar al objeto RecordStore, les presentamos el código siguiente:

```
    private void CerrarRecordStore(){
        try{
            rs.closeRecordStore();
        }
        catch(RecordStoreException e )
        {
```

```
            // throw(e);
        }
    }
```

Obtener y Listar Registros.

Es importante crear mecanismos de consulta para recuperar registros, es por ello que debemos incorporar dos métodos que se muestran:

```
    public String[] ObtenerRegistro(int idRecord)
    {
        String[] datos= new String[3];
        byte[] nombreB= new byte[20];
        byte[] numeroB= new byte[15];
        byte[] direccB= new byte[30];
        byte[] Data= new byte[nombreB.length+numeroB.length+direccB.length];
        try
        {
            Data=rs.getRecord(idRecord);
        }
        catch(RecordStoreException ex )
        {
            //luego veremos que hacemos que esta exception
        }
        for (int i = 0; i<nombreB.length;i++){
            nombreB[i]=Data[i];
        }
        for (int i = 0; i<numeroB.length;i++){
            numeroB[i]=Data[nombreB.length+ i];
        }
        for (int i = 0; i<direccB.length;i++){
            direccB[i]=Data[nombreB.length+numeroB.length+ i];
        }
        datos[0]= nombreB.toString();
        datos[0]= numeroB.toString();
        datos[0]= direccB.toString();
        return datos;
    }
```

Se usa el método getRecord, para obtener el registro a través del IdRecord el tipo de dato que retorna es un arreglo de byte, luego coloca en los arreglos nombreB y numeroB, para posteriormente convertirlos en string, y almacenarlos en el arreglo de strings llamado datoRet.

```
    public String[] ObtenerToDosDatos()
    {
        String Todos[] = null;
        int totalReg=0;
        try
        {
            byte[] byteTemB= new byte[20];
            byte[] Dato= new byte[65];
            //byte[] numeroB= new byte[15];
            totalReg= rs.getNumRecords();//total de registros
            Todos= new String[totalReg];
            String datoTem;
            char charTem[]= new char[20];
            int idRecord=-1;
            for (int i = 1; i<=totalReg;i++){//
                Dato= rs.getRecord(i);//obtiene el registro
                 for(int j=0; j<byteTemB.length;j++)
                {
                    byteTemB[j]= Dato[j];
                    charTem[j]=(char) Dato[j];
```

```
            }
            datoTem=String.valueOf(charTem);
            Todos[i-1]= datoTem;
        }
    }
    catch(RecordStoreException Ex)
    {
        //poner codigo de excpecion que debe ejecutarse
        int i=0;
    }
    return (String[])Todos;
}
```

El método ObtenerTodosDatos(), primero obtiene el numero total de registros, para crear el arreglo total de datos que van devolverse Todos. Luego utilizamos un forma para recorrer el total de registros; por cada iteración, se usa el método getNextRecordID(), para obtener Idrecord, que se usará en el método getRecord(idRecor); luego se pasa byte a byte en el arreglo byteTemB. Por último se asigna al arreglo convirtiéndose primero en string antes de la asignación.

Agregar registros.
El Ejemplo no estaría completo si no insertáramos registros, para ello añadimos el siguiente método en nuestra clase:

```
public void  NuevoReg(String Nombre,String Numero,String Direcc){

    int recordId=0;
    int logRecNomb=20;// longitud de registro del campo nombre
    int logRecNume=15;// longitud de registro del campo numero
    int logRecDire=30;// longitud de registro del campo direccion

    byte[] nombreB= new byte[logRecNomb];
    byte[] numeroB= new byte[logRecNume];
    byte[] direccB= new byte[logRecDire];

    byte[] Data= new byte[logRecNomb+logRecNume+logRecDire];
    nombreB= Nombre.getBytes();//permite obtener bytes de nombre
    numeroB= Numero.getBytes();//permite obtener bytes de numero
    direccB= Direcc.getBytes();//permite obtener bytes de numero
    for(int i=0; i<nombreB.length;i++)//Recorrer los byte del arreglo nombreB y pasa a Data
    {
        Data[i]=nombreB[i];
    }
    for(int i=0; i<numeroB.length;i++)// recorre los byte del arreglo nombreB y pasa a Data
    {
        Data[logRecNomb+i]=numeroB[i];
    }
    for(int i=0; i<direccB.length;i++)//Recorre los byte del arreglo nombreB y pasa a Data
    {
        Data[logRecNomb+logRecNume+i]=direccB[i];
    }
    try
    {
        recordId= rs.addRecord(Data,0,Data.length);
        //return recordId;
    }
    catch(RecordStoreException ex)
    {
        //thrown RecordStoreException(ex.getMessage());
    }
```

```
    }
```

NuevoReg, dispone de dos parámetros de tipo string, estos parámetros se convierten en bytes y son asignadas a los arreglos nombreB, numeroB; el arreglo Data unifica los dos datos para que puedan ser insertados a través de método addRecord(Data, 0, Data.length),

Para terminar con la clase vamos a modificar el constructor de la clase con la siguiente línea de código

```
CrearRecordStore();//dentro del constructor se crea el RecordStore
```

Borrar Registros.

Además de los métodos implementados deberíamos construir, una función que permita borrar algún registros, o todos, para ésta tarea, debemos incluir el siguiente código en nuestra clase.

```
public void BorrarRecord(int IdRecord)
{
    try
    {
        rs.deleteRecord(IdRecord);// toma el id del Registro y se borra el mismo
    }
    catch(RecordStoreException Ex)
    {
        int i=0;
    }
}
public void BorrarTodos()
{
    try
    {
        CerrarRecordStore();
        rs.deleteRecordStore("myRs");
        CrearRecordStore();

    }
    catch(RecordStoreException Ex)
    {
        int ip=0;
    }
}
```

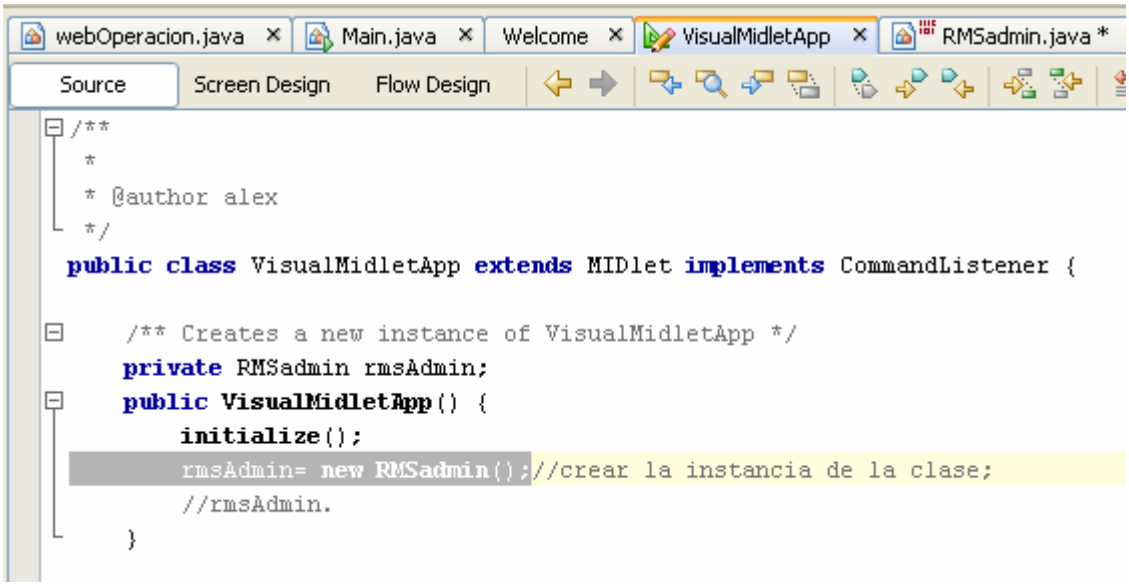
Los dos métodos eliminan registros, el primero acepta como parámetro, el IdRecord, para eliminar un registro determinado. El segundo metodo elimina todo los registros.

Bien ahora que tenemos listo nuestra clase debemos crear un instancia de la clase creada como propiedad de midlet, para ello debemos ir a la ventana *Project* hacemos doble click, en el nodo *VisualMidletApp,* luego debemos elegir el modo *Source,* y veremos el código fuente de nuestra aplicación, y sobre la el constructor de clase agregamos la siguiente línea de código:

```
private RMSadmin rmsAdmin;
```

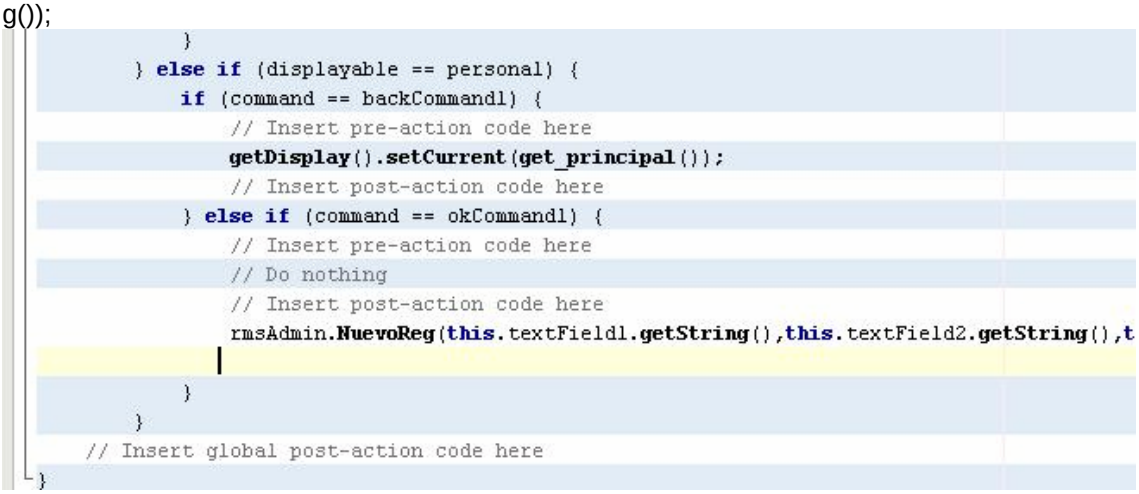
y luego en al constructor de la clase agregamos lo siguiente:

```
rmsAdmin= new RMSadmin();
```



Usando métodos de la clase RMSadmin.
Insertando Registros
Como ahora estanos en modo Source, debemos cambiar a modo Flow Desing. en este modo seleccionamos *personal[Form],* hacemos click derecho y luego edit. Luego en la partes inquiera de nuestra pantalla en la venta inspector, debemos localizar el nodo correspondiente a *personal[Form],* normalmente suele estar bajo el nodo *Screens,*
Dentro de ello expandimos *Assinged Command,* y hacemos click derecho en *okCommand1,* y elegimos Go To Source, e insertamos la siguientes linea de codigo

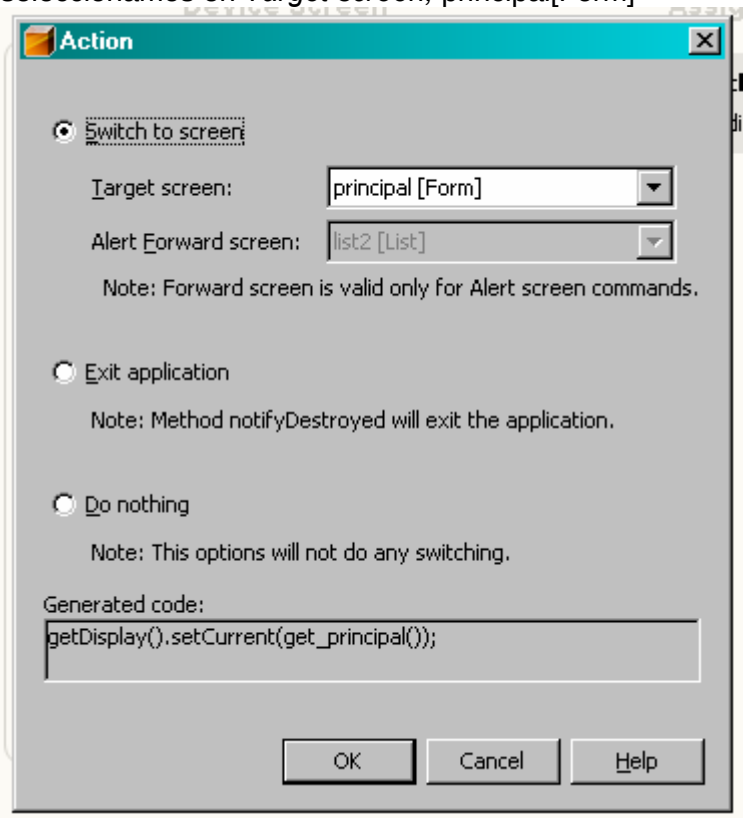
rmsAdmin.NuevoReg(this.textField1.getString(),this.textField2.getString(),this.textField3.getString());



La pantalla *personal[form]* se encargara de insertar registro.

Listando todos los Registros.
Muy bien, ahora vamos a modo Flow Screen y hacemos click derecho en *listado[Formt],* y elegimos Delete(vamos a usuario un nuevo scren llamado List), luego arrastramos un screen de la venta de la parte derecha, llamado List, en la propiedades *Title*, escribimos “Listado de Contactos”, y en Instante Name escribimos “list2”. Luego hacemos doble-click en la *list2[List].* Arrastramos un *Back Command*, luego hacemos

click en edit del back command, en la en ventana elegimos el primer option y seleccionamos en T*arget screen,* principal[Form]



En la venta Inspector, expandimos Ítems/StringItem2/Assigned Commands, hacemos click derecho en *itemcommand2,* y elegir *Go To Source, debe copiar el siguiente codigo:*

getDisplay().setCurrent(get_list2Todos());

```
    public void commandAction(Command command, Item item) {
    // Insert global pre-action code here
        if (item == stringItem2) {
            if (command == itemCommand2) {
/*                  // Insert pre-action code here
                getDisplay().setCurrent(get_listado());
                // Insert post-action code here*/
                getDisplay().setCurrent(get_list2Todos());
            }
        } else if (item == stringItem3) {
            if (command == itemCommand3) {
                // Insert pre-action code here
                exitMIDlet();
                // Insert post-action code here
            }
```

Tal como se muestra en le grafico debemos comentar las tres lineas anteriores. Luego de copiar esa línea de código vemos que la linea muestra un mensaje de error Como podemos ver que se muestra un error y eso es por que el mento get_List2Todos(), no existe, para ello demos copiar el siguiente método:

```
public List get_list2Todos()
{
    String contactos[]=null;
    boolean flags[];

    if (list2==null)
```

```
        {
            list2 = new List("Contactos", Choice.IMPLICIT, new String[0], new Image[0]);
        }
        else
        {
            list2.deleteAll();
        }
            contactos= rmsAdmin.ObtenerToDosDatos();
            flags=new boolean[contactos.length];

            for(int i= 0; i<contactos.length;i++)
            {
                this.list2.insert(i,contactos[i],null);

            }
            list2.addCommand(get_backCommand2());

            list2.setCommandListener(this);
            list2.setSelectedFlags(flags);
        //}
        return list2;
}
```

Si todo va bien, compila la Aplicación. Ejecuta.

Nota Uds. Pueden implementar al opción para borrar todo los registros, o también para incorporara nuevos métodos, a la Clase RMSadmin.java, para ello puedes visitar los link escritos al final del artículo.

Saludos

Direcciones

http://www.ibm.com/developerworks/library/wi-**rms**/
http://www.microjava.com/articles/techtalk/rms
http://developers.sun.com/mobility/midp/articles/databaserms/