

Desarrollo de Aplicaciones para Android

Sesión 6: Ficheros y acceso a datos



Puntos a tratar

- Ficheros tradicionales
- Preferencias
- SQLite
- Proveedores de contenidos



Ficheros tradicionales

```
FileOutputStream fos = openFileOutput("fichero.txt", Context.MODE_PRIVATE);  
FileInputStream fis = openFileInput("fichero.txt");
```

- `Context.MODE_PRIVATE` hace que el fichero sea privado a la aplicación
- El fichero se creará si no existe
- `Context.MODE_APPEND` para añadir al final del archivo



Tarjeta de memoria SD

- Utilizaremos:
 - `Environment.getExternalStorageDirectory()`
 - `FileWriter`
- Para probar en el emulador necesitamos
 - Tener creada una SD
 - `mksdcard 512M sdcard.iso`
 - Para copiar fuera del emulador el archivo grabado en la tarjeta:
 - `adb pull /sdcard/fich.txt fich.txt`



Tarjeta de memoria SD

```
try {
    File raiz = Environment.getExternalStorageDirectory();
    if (raiz.canWrite()){
        File file = new File(raiz, "fichero.txt");
        BufferedWriter out = new BufferedWriter(new FileWriter(file));
        out.write("Mi texto escrito desde Android");
        out.close();
    }
} catch (IOException e) {
    Log.e("FILE I/O", "Error en la escritura de fichero: " +
        e.getMessage());
}
```



Preferencias

- `SharedPreferences`: mecanismo que Android ofrece para almacenar opciones y preferencias
- Mucho más mantenible y sencillo que si se usan ficheros tradicionales
- Integrable de forma muy directa con interfaz gráfica de usuario para ajustar los valores de las preferencias
- Aparte de eso las `Activity` guardan el estado de la interfaz de usuario cuando pasan a segundo plano. El manejador `onSaveInstanceState()` está diseñado para guardar el estado de la actividad cuando ésta debe ser terminada.



Designing for Seamlessness

- En la guía oficial,
<http://developer.android.com/guide/practices/design/seamlessness.html>
- Aplicaciones “sin costuras”
 - **Don't Drop Data** →
 - Don't Expose Raw Data
 - Don't Interrupt the User
 - Got a Lot to Do? Do it in a Thread
 - Don't Overload a Single Activity Screen
 - Extend System Themes
 - Design Your UI to Work with Multiple Screen Resolutions
 - Assume the Network is Slow
 - Don't Assume Touchscreen or Keyboard
 - Do Conserve the Device Battery
- Otra actividad puede aparecer en cualquier momento
- Si una actividad acepta entrada de texto, deberá sobrecargar el `onSaveInstanceState()` para guardarlo de manera apropiada
- Ejemplo: cliente durante la redacción de un e-mail.

Guardar preferencias comunes

```
SharedPreferences pref;  
pref = PreferenceManager.getDefaultSharedPreferences(  
    getApplicationContext());  
  
SharedPreferences.Editor editor = pref.edit();  
editor.putBoolean("esCierto", false);  
editor.putString("Nombre", "Boyan");  
editor.commit();  
  
//Lectura de las preferencias:  
boolean esCierto = pref.getBoolean("esCierto", false);  
String nombre = pref.getString("Nombre", "sin nombre");
```

- En la lectura de las preferencias se indica un valor por defecto porque, si la preferencia indicada en la clave no existe, será creada y se devolverá el valor por defecto



Cambio de preferencias

- Listener para actualizar lo que sea necesario cuando ocurra algún cambio desde alguna parte de la aplicación:

```
prefValidacion.registerOnSharedPreferenceChangeListener(  
    new OnSharedPreferenceChangeListener() {  
        @Override  
        public void onSharedPreferenceChanged(  
            SharedPreferences sharedPreferences,  
            String key)  
        {  
            //Ha cambiado algo, emprender las  
            //acciones necesarias.  
        }  
    });
```



GUI para preferencias

- `res/xml/preferencias.xml`
- Las `key` deben coincidir con las claves de las preferencias

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android">
  <PreferenceCategory android:title="Validar DNI en:">
    <CheckBoxPreference
      android:title="en el campo"
      android:summary="Validará la introducción de números y una letra"
      android:key="validacampo"></CheckBoxPreference>
    <CheckBoxPreference
      android:title="al pulsar"
      android:summary="Comprobará también que la letra sea la correcta"
      android:key="validaboton"></CheckBoxPreference>
  </PreferenceCategory>
  <PreferenceCategory android:title="Otras preferencias:">
    <CheckBoxPreference android:enabled="false"
      android:title="Otra, deshabilitada"
      android:key="otra"></CheckBoxPreference>
  </PreferenceCategory>
</PreferenceScreen>
```



GUI para preferencias

- Para mostrar el XML en una actividad:

```
public class Preferencias extends PreferenceActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        addPreferencesFromResource(R.xml.preferencias);  
    }  
}
```

```
startActivity(new Intent(this, Preferencias.class);
```



GUI para preferencias





SQLite

- Gestor de base de datos relacional
- Open Source
- Útil para aplicaciones pequeñas
- No requiere la instalación aparte de un gestor de base de datos.
- Android da soporte a SQLite



SQLite. Creación

- Crear la base de datos

```
private static final String CREATE_DB = "CREATE TABLE " + TABLE_NAME +  
    "(" + COLUMNAS[0] + " INTEGER PRIMARY KEY, "  
    + COLUMNAS[1] + " TEXT, "  
    + COLUMNAS[2] + " NUMBER)";
```

```
SQLiteDatabase db;  
db.execSQL(CREATE_DB);  
db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
```



SQLiteOpenHelper

- Patrón de diseño para abrir y/o actualizar la versión de la base de datos

```
private static class MiOpenHelper extends SQLiteOpenHelper {  
    MiOpenHelper(Context context) {  
        super(context, DATABASE_NAME, null, DATABASE_VERSION);  
    }  
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL(CREATE_DB);  
    }  
    @Override  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        Log.w("SQL", "onUpgrade: eliminando tabla si existe, y creándola de nuevo");  
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);  
        onCreate(db);  
    }  
}
```

```
MiOpenHelper openHelper = new MiOpenHelper(this.context);  
this.db = openHelper.getWritableDatabase();
```



SQLite. Insertar y eliminar

- Insertar datos con una sentencia compilada
 - Las sentencias compiladas evitan las inyecciones (accidentales o malintencionadas) de código SQL.

```
private static final String INSERT = "insert into " + TABLE_NAME +  
    "(" + COLUMNAS[1] + ", " + COLUMNAS[2] + ") values (?, ?)";
```

```
SQLiteStatement insertStatement;  
insertStatement = db.compileStatement(INSERT);  
insertStatement.bindString(1, "Alicante");  
insertStatement.bindLong(2, 12345);  
insertStatement.executeInsert();  
  
db.delete(TABLE_NAME, null, null);
```


SQLite. Query

- Los resultados de las query se recorren con un cursor.

```
List<String> list = new ArrayList<String>();
Cursor cursor = db.query(TABLE_NAME, COLUMNAS,
    null, null, null, null, null);
if (cursor.moveToFirst()) {
    do {
        list.add(cursor.getString(1));
    } while (cursor.moveToNext());
}
if (cursor != null && !cursor.isClosed()) {
    cursor.close();
}
```



SQLite Data Helper

- Un Data Helper es un patrón de diseño que separa el código de acceso a la fuente de datos en una clase con métodos para insertar, modificar, eliminar datos, etc.
- Lo programamos como una clase propia.

```
public class DataHelper {  
    private static final String DATABASE_NAME = "mibasededatos.db";  
    private static final int DATABASE_VERSION = 1;  
    private static final String TABLE_NAME = "ciudades";  
    private static final String[] COLUMNAS = {"_id","nombre","habitantes"};  
    private static final String INSERT = "insert into " + TABLE_NAME +  
        "("+COLUMNAS[1]+",""+COLUMNAS[2]+") values (?,?)";  
    private static final String CREATE_DB = "CREATE TABLE " + TABLE_NAME +  
        "("+COLUMNAS[0]+" INTEGER PRIMARY KEY, "  
        +COLUMNAS[1]+" TEXT, "  
        +COLUMNAS[2]+" NUMBER)";  
    private Context context;  
    private SQLiteDatabase db;  
    private SQLiteStatement insertStatement;  
    ...  
}
```



SQLite Data Helper

```
...
public DataHelper(Context context) { //Constructor
    this.context = context;
    MiOpenHelper openHelper = new MiOpenHelper(this.context);
    this.db = openHelper.getWritableDatabase();
    this.insertStatement = this.db.compileStatement(INSERT);
}

private static class MiOpenHelper extends SQLiteOpenHelper {
    MiOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_DB);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w("SQL", "onUpgrade: eliminando tabla si existe, y creándola de nuevo");
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
...
```



SQLite Data Helper

```
...
public long insert(String name, long number) {
    this.insertStatement.bindString(1, name);
    this.insertStatement.bindLong(1, number);
    return this.insertStatement.executeInsert();
}
public int deleteAll() {
    return db.delete(TABLE_NAME, null, null);
}
public List<String> selectAllNombres() {
    List<String> list = new ArrayList<String>();
    Cursor cursor = db.query(TABLE_NAME, COLUMNAS,
        null, null, null, null, null);
    if (cursor.moveToFirst()) {
        do {
            list.add(cursor.getString(1));
        } while (cursor.moveToNext());
    }
    if (cursor != null && !cursor.isClosed()) {
        cursor.close();
    }
    return list;
}
}
```



Proveedores de contenidos

- Todos heredan de la clase `ContentProvider`
- Proporcionan una interfaz para publicar y consumir datos
- Identifican la fuente de datos con una URI que empieza por `content://`
- Hay proveedores nativos y podemos crear proveedores propios
- Es una forma estándar para compartir fuentes de datos entre aplicaciones diferentes
- Se deben declarar en el `AndroidManifest.xml`



Proveedores nativos

- Algunos proveedores de contenido nativos son los que se utilizan para acceder a los contactos, colección de música, etc:
 - Browser, CallLog, ContactsContract, MediaStore, Settings, UserDictionary.
- Necesitan tener permisos en el AndroidManifest.xml, por ejemplo, para los contactos:

```
...  
<uses-sdk android:minSdkVersion="8" />  
<uses-permission android:name="android.permission.READ_CONTACTS"/>  
</manifest>
```



Query a un proveedor

- Las query nos devuelven un cursor que debemos recorrer:

```
ContentResolver.query(  
    Uri uri,  
    String[] projection,  
    String selection,  
    String[] selectionArgs,  
    String sortOrder)
```

- Acceder a la lista completa de contactos:

```
ContentResolver cr = getContentResolver();  
Cursor cursor = cr.query(ContactsContract.Contacts.CONTENT_URI,  
    null, null, null, null);
```



Notificación de cambios

- Un cursor puede ser notificado de los cambios que ocurran en un ContentProvider:

```
cursor.setNotificationUri(cr, ContactsContract.Contacts.CONTENT_URI);
```

- Muy útil para que un componente gráfico actualice el contenido que muestra, de manera automática

Adaptador

- Ejemplo de adaptador de un `Cursor` a un componente gráfico `ListView`

```
ListView lv = new (ListView)findViewById(R.id.ListView01);
SimpleCursorAdapter adapter = new SimpleCursorAdapter(
    getApplicationContext(),
    R.layout.textviewlayout,
    cursor,
    new String[]{
        ContactsContract.Contacts._ID,
        ContactsContract.Contacts.DISPLAY_NAME},
    new int[]{
        R.id.TextView1,
        R.id.TextView2});
lv.setAdapter(adapter);
```

- Hay un layout que define el aspecto de la fila:



Adaptador

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <TextView android:id="@+id/TextView1"
        android:textStyle="bold"
        android:ems="2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </TextView>
    <TextView android:id="@+id/TextView2"
        android:textStyle="bold"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </TextView>
</LinearLayout>
```



Proveedores propios

- Heredaremos de `ContentProvider`
- Sobrecargaremos
 - `onCreate()`
 - `delete(...)`
 - `insert(...)`
 - `query(...)`
 - `update(...)`
 - `getType(Uri)` : el tipo MIME del contenido
- Declararemos una URI:
 - `content://es.ua.jtech.daa.proveedores/ciudades`
- Lo declararemos en el `AndroidManifest.xml`



Proveedores propios

- Declaración en el `AndroidManifest.xml`:

```
<provider
    android:name="CiudadesProvider"
    android:authorities="es.ua.jtech.daa.proveedores" />
```

- Coincidiendo con el nombre de la clase:

```
package es.ua.jtech.daa.proveedores;

import android.content.ContentProvider;

public class CiudadesProvider extends ContentProvider {
    ...
}
```

URI del proveedor propio

- La URI se declara como constante pública:

```
public static final Uri CONTENT_URI = Uri.parse(
    "content://es.ua.jtech.daa.proveedores/ciudades");
```

- Diferenciar entre acceder a una fila o a múltiples por medio de un “1” ó “2”:
 - Todas las filas:
content://es.ua.jtech.daa.proveedores/ciudades/1
 - Una fila (la 23):
content://es.ua.jtech.daa.proveedores/ciudades/2/23

UriMatcher

- Para diferenciar fácilmente entre el acceso a una o a múltiples filas nos declaramos una constante estática privada `UriMatcher` dentro del proveedor:

```
private static final UriMatcher uriMatcher;  
static{  
    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);  
    uriMatcher.addURI("es.ua.jtech.daa.proveedores", "ciudades", ALLROWS);  
    uriMatcher.addURI("es.ua.jtech.daa.proveedores", "ciudades/#", SINGLE_ROW);  
}
```



Tipo MIME

- Devolveremos un tipo MIME con
 - Prefijo `vnd.`
 - Base del dominio, por ejemplo `ua`
 - `.cursor`
 - `.dir/` si son todas las filas o `.item/` si es uno sólo
 - Texto identificativo que indica el nombre de la clase seguido de `content` : si la clase es `CiudadesProvider`, sería `ciudadesprovidercontent`

Tipo MIME

- Devolvemos un tipo MIME u otro si es una fila o todas, usando para ello el `UriMatcher`:

```
@Override
public String getType(Uri uri) {
    switch(uriMatcher.match(uri)){
        case ALLROWS: return "vnd.ua.cursor.dir/ciudadesprovidercontent";
        case SINGLE_ROW: return "vnd.ua.cursor.item/ciudadesprovidercontent";
        default: throw new IllegalArgumentException("URI no soportada: "+uri);
    }
}
```




Notificar cambios

- Para que la notificación de cambios funcione nuestro proveedor propio tendrá que notificarlos explícitamente tras haber terminado de realizarlos.
- La notificación se realiza al ContentResolver
- La notificación se realiza acerca de determinada URI:

```
context.getContentResolver().notifyChange(uri, null);
```



Proveedor propio

```
package es.ua.jtech.daa.proveedores;

public class CiudadesProvider extends ContentProvider {
    //Campos típicos de un ContentProvider:
    public static final Uri CONTENT_URI = Uri.parse(
        "content://es.ua.jtech.daa.proveedores/ciudades");
    private static final int ALLROWS = 1;
    private static final int SINGLE_ROW = 2;
    private static final UriMatcher uriMatcher;
    static{
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI("es.ua.jtech.daa.proveedores", "ciudades", ALLROWS);
        uriMatcher.addURI("es.ua.jtech.daa.proveedores", "ciudades/#", SINGLE_ROW);
    }

    public static final String DATABASE_NAME = "mibasededatos.db";
    public static final int DATABASE_VERSION = 2;
    public static final String TABLE_NAME = "ciudades";
    private static final String[] COLUMNAS = {"_id", "nombre", "habitantes"};
    private static final String CREATE_DB = "CREATE TABLE " + TABLE_NAME +
        "(" + COLUMNAS[0] + " INTEGER PRIMARY KEY, "
        + COLUMNAS[1] + " TEXT, "
        + COLUMNAS[2] + " NUMBER)";
    private Context context;
    private SQLiteDatabase db;
    ...
}
```



Proveedor propio

```
...
@Override
public boolean onCreate() {
    this.context = getContext();
    MiOpenHelper openHelper = new MiOpenHelper(this.context);
    this.db = openHelper.getWritableDatabase();
    return true;
}

private static class MiOpenHelper extends SQLiteOpenHelper {

    MiOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_DB);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
        Log.w("SQL", "onUpgrade: eliminando tabla si ésta existe,"+
            " y creándola de nuevo");
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
...
```



Proveedor propio

```
...
@Override
public String getType(Uri uri) {
    switch(uriMatcher.match(uri)){
        case ALLROWS: return "vnd.ua.cursor.dir/ciudadesprovidercontent";
        case SINGLE_ROW: return "vnd.ua.cursor.item/ciudadesprovidercontent";
        default: throw new IllegalArgumentException("URI no soportada: "+uri);
    }
}

@Override
public Cursor query(Uri uri, String[] projection, String selection,
    String[] selectionArgs, String sortOrder) {
    return db.query(TABLE_NAME, COLUMNAS, selection, selectionArgs,
        null, null, null);
}
...
```



Proveedor propio

```
...
@Override
public Uri insert(Uri uri, ContentValues values) {
    long id = db.insert(TABLE_NAME, COLUMNAS[1], values);
    if(id > 0 ){
        Uri uriInsertado = ContentUris.withAppendedId(CONTENT_URI, id);
        context.getContentResolver().notifyChange(uriInsertado, null);
        return uriInsertado;
    }
    throw new android.database.SQLException(
        "No se ha podido insertar en "+uri);
}
...
```



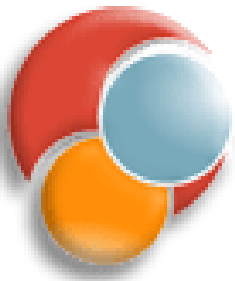
Proveedor propio

```
...
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    int changes = 0;
    switch(uriMatcher.match(uri)){
        case ALLROWS:
            changes = db.delete(TABLE_NAME, selection, selectionArgs);
            break;
        case SINGLE_ROW:
            String id = uri.getPathSegments().get(1);
            Log.i("SQL", "delete the id "+id);
            changes = db.delete(TABLE_NAME,
                "_id = " + id +
                    (!TextUtils.isEmpty(selection) ?
                        " AND (" + selection + ')' :
                        ""),
                selectionArgs);
            break;
        default: throw new IllegalArgumentException("URI no soportada: "+uri);
    }
    context.getContentResolver().notifyChange(uri, null);
    return changes;
}
...
```



Proveedor propio

```
...
@Override
public int update(Uri uri, ContentValues values, String selection,
                  String[] selectionArgs) {
    int changes = 0;
    switch(uriMatcher.match(uri)){
        case ALLROWS:
            changes =db.update(TABLE_NAME, values, selection,
                              selectionArgs);
            break;
        case SINGLE_ROW:
            String id = uri.getPathSegments().get(1);
            Log.i("SQL", "delete the id "+id);
            changes =  db.update(TABLE_NAME, values,
                                "_id = " + id +
                                (!TextUtils.isEmpty(selection) ?
                                 " AND (" + selection + ')'
                                 : ""),
                                selectionArgs);
            break;
        default: throw new IllegalArgumentException("URI no soportada: "+uri);
    }
    context.getContentResolver().notifyChange(uri, null);
    return changes;
}
}
```



¿Preguntas...?