Introducción a la programación de Dispositivos Móviles.

Dr. Mauricio Arroqui
Dr. Juan Manuel Rodriguez
Ing. Juan Maximiliano Rodriguez

Contenidos

- Threads
- AsyncTask
- Loaders
- AsyncTaskLoader

- Broadcast Receivers
- Almacenamiento

Ejemplo en vivo!!!

Threads

El Main thread debe ser rápido

- El Hardware actualiza la pantalla cada 16 milisegundos
- Un UI thread tiene 16 para hacer todo su trabajo
- Si toma más, la app se tilda o se cuelga

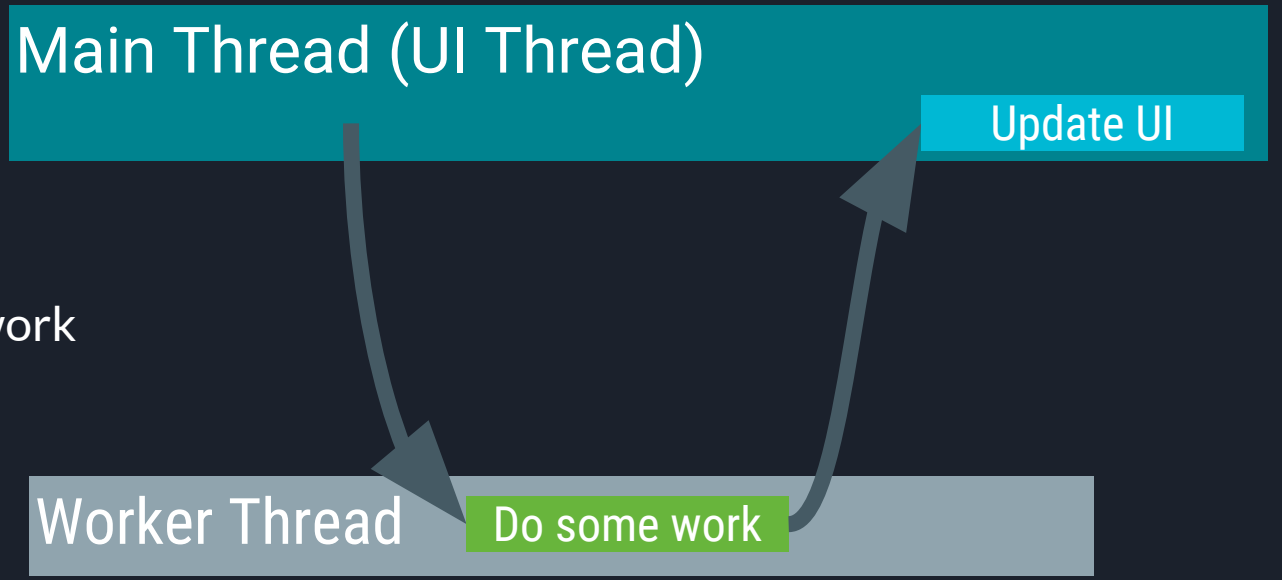
Tareas pesadas?

- Operaciones de red
- Grandes calculo
- subir/bajar archivos
- Procesamiento de imágenes
- Cargar datos

Background threads

Ejecutar tareas pesadas en un Thread en background

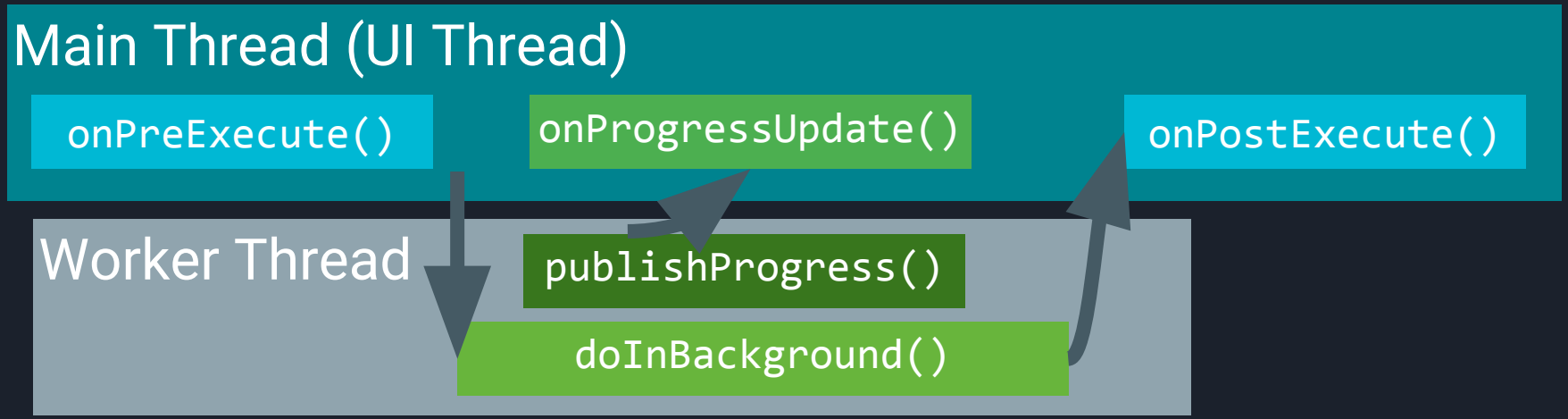
- AsyncTask
- Loader Framework
- Services



AsyncTask

Que es AsyncTask?

Usar AsyncTask para implementar tareas básicas en background



```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

Para arrancar:

```
new DownloadFilesTask().execute(url1, url2, url3);
```

Limitaciones de AsyncTask

- Cuando el dispositivo cambia la configuración, la activity es destruida
- AsyncTask no se puede conectar más a la Activity
- Una nueva AsyncTask es creada por cada cambio de configuración
- AsyncTasks viejas quedan ahi!!!
- La App puede quedarse sin memoria y crashear.

Cuando usar AsyncTask?

- Tareas cortas o que pueden ser interrumpidas
- Tareas que no necesitan reportar nada a la UI o al usuario
- Tareas de baja prioridad que pueden no terminar
- Sino usar AsyncTaskLoader

Ejemplo ahora con AsyncTask

3er Entregable

Loaders

Qué es un Loader?

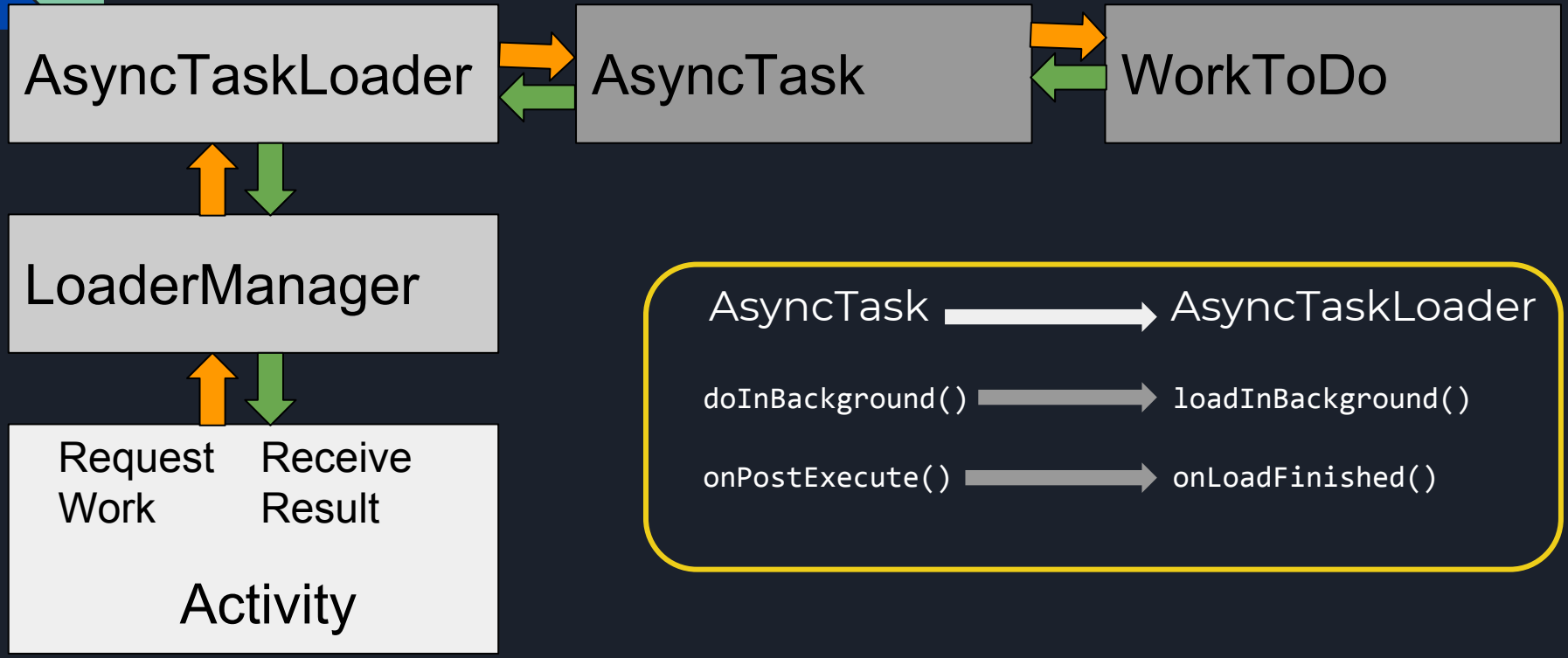
- Prové una forma asíncrona para cargar información
- **Se reconecta a la activity si la configuración cambia**

- Los Callbacks son implementados en la Activity
- Tipos de Loaders
  - AsyncTaskLoader, CursorLoader

Por qué usar Loaders?

- Ejecuta fuera del UI thread
- El LoaderManager maneja los cambios de configuración
  - Administra múltiples loaders: loader de base de datos, de internet, de AsyncTask
- Está implementado eficientemente por el framework
- Los usuarios no tienen que esperar hasta que los datos sean cargados

Implementar AsyncTaskLoader (Visión general)



Pasos para crear una AsyncTaskLoader

1. Sub clase de AsyncTaskLoader (Otro ejemplo completo!)
2. Implementar el constructor
3. `loadInBackground()`
4. `onStartLoading()`

Broadcast Receivers

Qué es un broadcast receiver?

- Escucha “intents” enviados por sendBroadcast()
  - Estando en background
- Los “intents” pueden ser enviados:
  - Por el sistema, cuando ocurre un evento que puede cambiar el comportamiento de una app
  - Por otra aplicación, incluyendo ella misma

NOTA: Las apps Android pueden enviar o recibir mensajes broadcast del sistema Android como de otra app, similar al patrón de diseño publish-subscribe

Los Broadcast receiver siempre responden

- Responden aún cuando la aplicación está cerrada
- Son independientes de cualquier activity
- Cuando un broadcast intent es recibido y enviado a onReceive(), tiene 5 segundos para ejecutar, luego el “receiver” es destruido

Broadcasts de sistema

- Automáticamente son enviados cuando ciertos eventos ocurren

- Cuando el sistema Bootea
  - `android.intent.action.BOOT_COMPLETED`
- Cuando el wifi cambia de estado
  - `android.net.wifi.WIFI_STATE_CHANGED`

Custom broadcasts

- Enviar un custom intent como un broadcast
  - `sendBroadcast()` method—asynchronous
  - `sendOrderedBroadcast()`—synchronously
  - `android.example.com.CUSTOM_ACTION`

sendBroadcast()

- Todos los “receiver” de broadcast son ejecutados sin orden
- Pueden ser ejecutados en paralelo
- + Eficiente
- Usado para enviar custom broadcasts

sendOrderedBroadcast()

- Enviado a un “receiver” a la vez
- Receiver puede propagar el resultado al siguiente receiver o abortar el broadcast
- El orden es controlado con android:priority macheando los intent filter
- Los “receivers” con misma prioridad ejecutan en orden arbitrario

Pasos para crear un broadcast receiver

- Subclass `BroadcastReceiver`
- Implementar on`Receive()` method
- Registrarse para recibir broadcast
  - Estaticamente, en el AndroidManifest
  - Dinamicamente, con `registerReceiver()`

File > New > Other > BroadcastReceiver

```
public class CustomReceiver extends BroadcastReceiver {
    public CustomReceiver() {
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO: This method is called when the BroadcastReceiver
        //  is receiving an Intent broadcast.
        throw new UnsupportedOperationException("Not yet implemented");
    }
}
```

Registro en el Manifest Android

- <receiver> dentro de <application>
- <intent-filter> registra receiver para intents específicos

```
<receiver
   android:name=".CustomReceiver"
   android:enabled="true"
   android:exported="true">
      <intent-filter>
 <action android:name="android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
</receiver>
```

Registro dinámico

- En onCreate() o onResume()
- Usar registerReceiver() y pasar como parámetro el intent filter
- Se debe eliminar el registro (desregistrar) en onDestroy() o onPause()

```
registerReceiver(mReceiver, mIntentFilter)
unregisterReceiver(mReceiver)
```

Intents disponibles

- ACTION_TIME_TICK
- ACTION_TIME_CHANGED
- ACTION_TIMEZONE_CHANGED
- ACTION_BOOT_COMPLETED
- ACTION_PACKAGE_ADDED
- ACTION_PACKAGE_CHANGED
- ACTION_PACKAGE_REMOVED
- ACTION_PACKAGE_RESTARTED
- ACTION_PACKAGE_DATA_CLEARED
- ACTION_PACKAGES_SUSPENDED
- ACTION_PACKAGES_UNSUSPENDED
- ACTION_UID_REMOVED
- ACTION_BATTERY_CHANGED
- ACTION_POWER_CONNECTED
- ACTION_POWER_DISCONNECTED
- ACTION_SHUTDOWN

Implement onReceive()

```
@Override
public void onReceive(Context context, Intent intent)
{
    String intentAction = intent.getAction();
    switch (intentAction){
        case Intent.ACTION_POWER_CONNECTED:
            break;
        case Intent.ACTION_POWER_DISCONNECTED:
            break;
    }
}
```

Custom broadcasts

- El emisario y el receptor deben acordar un nombre único para el intent (nombre de acción)
- Definir en la activity y broadcast receiver

```
public static final String ACTION_CUSTOM_BROADCAST =
        "com.example.android.ACTION_CUSTOM_BROADCAST";
```

Enviar custom broadcasts

```
Intent customBroadcastIntent =
    new Intent(ACTION_CUSTOM_BROADCAST);

LocalBroadcastManager.getInstance(this)
    .sendBroadcast(customBroadcastIntent);
```

Destroy!

```
@Override
protected void onDestroy() {
    super.onDestroy();
    LocalBroadcastManager.getInstance(this)
        .unregisterReceiver(mReceiver);

}
```

Local Broadcast Manager

- Para hacer broadcasts dentro una app
- No se necesitas cuestiones de seguridad ya que no hay comunicación entre apps

```
LocalBroadcastManager.sendBroadcast()

LocalBroadcastManager.registerReceiver()
```

Registrar un local broadcast manager

```
LocalBroadcastManager.getInstance(this)
  .registerReceiver( mReceiver,
      new IntentFilter(ACTION_CUSTOM_BROADCAST));
```

Seguridad!!!

- Los Receivers cruzan los límites de las aplicaciones
- Hay que asegurar que el namespace para intent es único y te pertenece
- Otras apps pueden enviar broadcasts a tu receiver—usar permisos para control
- Otras apps pueden responder a broadcast que tu app envie
- Permisos de acceso pueden ser forzados desde el emisario o el receptor

Control de permisos emisor

- `void sendBroadcast (Intent intent,`
  `String receiverPermission)`

- Los receivers deben solicitar permisos con `<uses-permission>` en el AndroidManifest.xml

Almacenamiento

Opciones de Almacenamiento

- Shared Preferences— Pares Clave-Valor de datos primitivos privados
- Internal Storage—Datos privados en la memoria del dispositivo
- External Storage—Datos públicos en el dispositivo o en almacenamiento externo
- SQLite Databases—Datos estructurados en base de datos privada
- Content Providers— Ayuda a una App a acceder a datos guardados por el mismo, por otras Apps y además provee una forma de compartir datos con otras Apps

Archivos

Sistema de archivos de Android

- Almacenamiento externo -- Directorios públicos
- Almacenamiento interno -- Directorios privados solo para la app

Las Apps pueden navegar la estructura de directorios

Estructura y operatoria similar a Linux y java.io

Almacenamiento interno

- Siempre disponible
- Usa el sistema de archivos del dispositivo
- Solo la tu app puede acceder, a nos ser que explicitamente sea configurable como lectura o escritura
- Cuando una app se desinstala, el sistema borra todos los archivos de la app en almacenamiento interno

Almacenamiento externo

- No siempre disponible, puede ser removida
- Usa el sistema de archivos del dispositivo o un almacenamiento externo como una tarjeta SD
- Todos lo pueden leer
- Cuando se desinstala una app, el sistema no borra los archivos privados de la app

Cuando usar almacenamiento interno/externo

Interno es mejor cuando

- Se quiere estar seguro que ni el usuario, ni otras apps tengan acceso a los archivos de la app

Externo es mejor cuando

- No se requieren restricciones de acceso
- Se quiere compartir con otras apps
- Se permite al usuario acceder con una computadora

Almacenamiento interno

Almacenamiento interno

- Directorios de almacenamiento permanente—getFilesDir()
- Directorios de almacenamiento temporario—getCacheDir()

Ejemplo 1:

```
File file = new File(
    context.getFilesDir(),
filename);
```

Ejemplo 2:

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

Almacenamiento externo

Almacenamiento externo

- En el dispositivo o en la SD
- Configurar los permisos en el Android Manifest
  - El permiso de escritura incluye el de lectura

```
<uses-permission
   android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
    android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Siempre hay que chequear la disponibilidad del almacenamiento

```
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
```

Acceder a los directorios del almacenamiento público

- Get a path getExternalStoragePublicDirectory()
- Create file

```
File path = Environment.getExternalStoragePublicDirectory(

            Environment.DIRECTORY_PICTURES);

File file = new File(path, "DemoPicture.jpg");
```

Cuánto espacio queda?

- Si no hay suficiente espacio, throws IOException
- Si se sabe el tamaño del archivo, se puede chequear
  - getFreeSpace()
  - getTotalSpace().
- Si no se sabe cuanto tamaño se necesita
  - try/catch IOException

Shared Preferences

Qué son Shared Preferences?

- Lectura y escritura de pocas cantidades de datos primitivos como pares clave/valor en un archivo del en el dispositivo
- La clase SharedPreference provee una API para leer, escribir y manejar estos datos

- Guardar datos en onPause()
  Recuperarlos en onCreate()

Shared Preferences Y Saved Instance State

- Un bajo número de pares clave/valor
- Los datos son privados a la app

Shared Preferences vs Saved Instance State

- Persiste datos entre sesiones de usuario, aún cuando la app es cerrada o reiniciada, o el dispositivo reboteado
- Los datos deben ser recordados entre sesiones, como las “settings” del usuario o el puntaje de en un juego
- El uso común es para preferencias del usuario

- Mantiene datos de estado de la instancia de una “activity” dentro de una sesión de usuario
- Los datos no deben ser recordados entre sesiones, como la pestaña seleccionada o el estado de la “activity”
- El uso común es para recrear el estado después de que un dispositivo es rotado

getSharedPreferences()

```
private String sharedPrefFile =
    "com.example.android.hellosharedprefs";

mPreferences =
    getSharedPreferences(sharedPrefFile,
                         MODE_PRIVATE);
```

SharedPreferences.Editor

```
@Override
protected void onPause() {
    super.onPause();
    SharedPreferences.Editor preferencesEditor =
        mPreferences.edit();
    preferencesEditor.putInt("count", mCount);
    preferencesEditor.putInt("color", mCurrentColor);
    preferencesEditor.apply();
}
```

Nota: apply() guarda asincrónicamente y seguro

Getting data in onCreate()

```
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
    mCount = mPreferences.getInt("count", 1);
    mShowCount.setText(String.format("%s", mCount));

    mCurrentColor = mPreferences.getInt("color", BLUE);
    mShowColor.setBackgroundColor(mCurrentColor);

    mNewText = mPreferences.getString("text", "");
```

SQLite Database

Usar SQLite database

- Versátil y sencilla de implementar
- Datos estructurados que necesita almacenar de forma persistente
- Acceda, busque y cambie los datos frecuentemente
- Almacenamiento principal para datos de usuarios o aplicaciones
- Cache y hacer disponibles los datos obtenidos de la nube
- Los datos pueden representarse como filas y columnas

SQLiteOpenHelper

Todas las interacciones con la base de datos pasan por SQLiteOpenHelper

- Ejecuta las solicitudes
- Maneja la base de datos
- Separa datos e interacción de la app
- Mantiene a aplicaciones complejas manejables

Cursors

Cursors

- Tipo de datos comúnmente utilizado para los resultados de las consultas
- Puntero a una fila de datos estructurados ...
- ... pensar en ello como una serie de filas
- La clase Cursor proporciona métodos para mover el cursor y obtener datos
- SQLiteDatabase siempre presenta los resultados como Cursor

Operaciones comunes de un Cursor

- getCount()—Número de filas en un cursor
- getColumnNames()—Arreglo de strings con los nombres de columna
- getPosition()—posición actual del cursor
- getString(int column), getInt(int column), ...
- moveToFirst(), moveToNext(), ...
- close() libera todos los recursos e invalida el cursor

Procesando un Cursor

```
// Store results of query in a cursor
Cursor cursor = db.rawQuery(...);
try {
    while (cursor.moveToNext()) {
         // Do something with data
     }
} finally {
    cursor.close();
}
```

Content Values

ContentValues

- Una instancia de ContentValues
  - Representa una fila de la tabla
  - Almacena datos como pares clave/valor
  - La clave es el nombre de la columna
  - Valor es el valor del campo
- Usado para pasar una fila como parámetro entre métodos

ContentValues

```
ContentValues values = new ContentValues();

// Inserts one row.
// Use a loop to insert multiple rows.
values.put(KEY_WORD, "Android");
values.put(KEY_DEFINITION, "Mobile operating system.");


db.insert(WORD_LIST_TABLE, null, values);
```

Implementing SQLite

Para implementar SQLite se necesita ...

- Crear modelo de datos
- Subclass SQLiteOpenHelper
  - Crear constantes para las tablas
  - onCreate()—crear SQLiteDatabase con las tablas
  - onUpgrade(), método opcional
  - Implementar query(), insert(), delete(), update(), count()
- En la MainActivity, crear una instancia de SQLiteOpenHelper
- Llamar a los métodos de SQLiteOpenHelper para trabajar con la base de datos

Modelo de datos

- Clase con getters y setters
- Un "item" de dato (para la base de datos, una fila)

```
public class WordItem {
    private int mId;
    private String mWord;
    private String mDefinition;
   ...
}
```

Subclass SQLiteOpenHelper

```
public class WordListOpenHelper extends SQLiteOpenHelper {

    public WordListOpenHelper(Context context) {
       super(context, DATABASE_NAME, null, DATABASE_VERSION);
       Log.d(TAG, "Construct WordListOpenHelper");
    }
}
```

Constantes para las tablas

```
private static final int DATABASE_VERSION = 1;

// Has to be 1 first time or app will crash.

private static final String DATABASE_NAME = "wordlist";

private static final String WORD_LIST_TABLE = "word_entries";

// Column names...

public static final String KEY_ID = "_id"

public static final String KEY_WORD = "word";

public static final String KEY_DEFINITION = "definition";
```

Definir query para la creación de la base de datos

- Una consulta para crear la base de datos
- Habitualmente definido como una constante de cadena

```
private static final String WORD_LIST_TABLE_CREATE =
            "CREATE TABLE " + WORD_LIST_TABLE + " (" +
                    KEY_ID + " INTEGER PRIMARY KEY, " +
                    // will auto-increment if no value passed
                    KEY_WORD + " TEXT );";
```

onCreate()

```
@Override
public void onCreate(SQLiteDatabase db) { // Creates new database
  // Create the tables
  db.execSQL(WORD_LIST_TABLE_CREATE);
  // Add initial data
  ...
}
```

onUpgrade()

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion,
                                         int newVersion) {
    // SAVE USER DATA FIRST!!!
    Log.w(WordListOpenHelper.class.getName(),
         "Upgrading database from version " + oldVersion + " to "
         + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + WORD_LIST_TABLE);
        onCreate(db);
    }
```

Operaciones de base de datos

- query()
- insert()
- update()
- delete()

Ejecutar queries

- implementar el método query() en la helper class
- query() puede tomar y retornar cualquier tipo de dato que la UI necesite
- Solo proveer soporte para aquellos queries que la app necesite
- Usar los insert, delete, and update a conveniencia

Métodos para ejecutar queries

- SQLiteDatabase.rawQuery()
  Usar cuando los datos están bajo tu control y solo son provistos por tu app

- SQLiteDatabase.query()
  Usar para todos los otros queries

SQLiteDatabase.rawQuery()

```
rawQuery(String sql, String[] selectionArgs)
```

- El primer parámetro es la cadena de consulta SQLite
- El segundo parámetro contiene los argumentos
- Úsalo solo si tus datos son proporcionados por la aplicación y bajo tu control total

Ejemplo SQLiteDatabase.rawQuery()

```
String query = "SELECT  * FROM " + WORD_LIST_TABLE +
        " ORDER BY " + KEY_WORD + " ASC ";

cursor = mReadableDB.rawQuery(queryString, null);
```

SQLiteDatabase.query()

```
Cursor query (boolean distinct,  String table,
              String[] columns, String selection,
              String[] selectionArgs, String groupBy,
              String having, String orderBy,String
limit);
```

Ejemplo SQLiteDatabase.query()

```
String[] columns = new String[]{KEY_WORD};
String where =  KEY_WORD + " LIKE ?";
searchString = "%" + searchString + "%";
String[] whereArgs = new String[]{searchString};
cursor = mReadableDB.query(WORD_LIST_TABLE, columns,
where,
         whereArgs, null, null, null);
```

insert()

```
long insert(String table, String nullColumnHack,
            ContentValues values)
```

- Primer parámetro es el nombre de la tabla
- Segundo parámetro es un `String nullColumnHack`.
  - Solución que le permite insertar filas vacías
  - Usa `null`
- Tercer parámetro tiene que ser un ContentValues con valores para la fila
- Retorna el id de la nueva fila insertada

Ejemplo insert()

```
newId = mWritableDB.insert(
    WORD_LIST_TABLE,
    null,
    values);
```

delete()

```
int delete (String table,
            String whereClause, String[] whereArgs)
```

- El primer parámetro es el nombre de la tabla
- El segundo parámetro es la cláusula WHERE
- El tercer parámetro son argumentos a la cláusula WHERE

Ejemplo delete()

```
deleted = mWritableDB.delete(
    WORD_LIST_TABLE,
    KEY_ID + " =? ",
    new String[]{String.valueOf(id)});
```

update()

```
int update(String table, ContentValues values,
        String whereClause, String[] whereArgs)
```

- Primer parámetro es el nombre de la tabla
- Segundo parámetro tiene que ser un ContentValues con valores nuevos para la fila
- El tercer parámetro es la cláusula WHERE
- El cuarto parámetro son argumentos a la cláusula WHERE

Ejemplo update()

```
ContentValues values = new ContentValues();
values.put(KEY_WORD, word);

mNumberOfRowsUpdated = mWritableDB.update(
                WORD_LIST_TABLE,
                values, // new values to insert
                KEY_ID + " = ?",
                new String[]{String.valueOf(id)});
```

AsyncTask! Que guarde el contador en sharedPreferences...

3er Entregable

Listo por HOY!