T1-Programacion-multimedia-y-dis...

bloodyraintatii

Programación multimedia y dispositivos móviles

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España

Accede al documento original

Programación multimedia y dispositivos móviles

Tema 1: Fundamentos de aplicaciones móviles

Dispositivos móviles

→ Pequeños, poco peso, pantalla, teclado (físico o táctil), capacidad de procesamiento elevada (no tanto como un ordenador), memoria, conexión a Internet, GPS, Bluetooth…

→ Tipos de dispositivos móviles:

- Smartphones → los teléfonos inteligentes de hoy en día.
- PDA → asistente digital diseñado para servir como agenda digital. En desuso.
- Tablets → casi las mismas funciones que los smartphones pero más grandes.

→ Limitaciones: la batería se agotará más rápido cuantas más reacciones realice, como tener el GPS activo.

Sistemas operativos móviles

→ Los dispositivos móviles necesitan un sistema operativo para poder funcionar. Estos están diseñados específicamente para los dispositivos móviles: no tienen la misma potencia de cálculo, ya que tienen que optimizar los recursos (como RAM o batería).

→ Sistemas operativos existentes:

- Android → basado en un Kernel de Linux. Desarrollado por Google. Diseñado para dispositivos móviles con pantalla táctil, como smartphones, tablets, relojes inteligentes, automóviles, televisores.
- iOS → desarrollado por Apple Inc. Diseñado para iPhone (iPhone iOS) y después usado en otros dispositivos como iPod touch y iPad. No se permite su instalación en hardware de terceros.

→ Para desarrollar aplicaciones Android hay herramientas disponibles en todas las plataformas: Windows, GNU/Linux y MacOS.

→ Para desarrollar aplicaciones para iOS debemos tener un ordenador Mac con MacOS porque el framework de desarrollo para iOS solo está disponible ahí.

Evolución de Android

→ Los creadores originales fueron Android Inc en 2003 y el sistema operativo fue comprado por Google en 2005. En 2007 Google liberó el sistema operativo bajo la licencia de código abierto Apache.

→ Las versiones de Android recibían el nombre inglés de dulces en orden alfabético hasta la versión 9.0.

Limitaciones de uso de una aplicación móvil

- Uso de energía limitado.
- Baja capacidad de cómputo en comparación a los ordenadores.
- Memoria RAM bastante reducida.
- Reducido espacio de almacenamiento interno, aunque se puede ampliar con tarjetas de memoria o la nube.

- Tienen un teclado integrado, normalmente táctil, con funcionalidades muy reducidas.
- Las pantallas son pequeñas.
- Existe el riesgo de obtener malware y poner en riesgo los datos del usuario.
- Si el dispositivo se queda sin cobertura, se desconecta de cualquier operación importante que esté en proceso o se necesite realizar.

Ciclo de vida de una aplicación móvil Android

→ Hay que conocer cuándo una aplicación se está usando, cuándo finaliza, cuándo le pasa el control a otra aplicación, etc., y se necesitará liberar memoria continuamente.

→ Estados de una aplicación:

- Activa → se está usando por el usuario.
- Pausada → la actividad no tiene el foco pero está activa.
- Parada → la actividad ya no es visible y se deben guardar los datos para usarlos posteriormente.

→ Eventos que representan los ciclos de vida de una app Android:

- onCreate(Bundle) → momento en el que se crea la actividad. Se pueden recuperar los datos si se han guardado anteriormente.
- onStart() → la actividad pasa a estar en pantalla aunque no necesariamente de forma visible.
- onRestart() → anterior a onStart(), tras venir de una llamada a onStop().
- onResume() → la actividad va a empezar a responder a la interacción del usuario.
- onPause() → la actividad va a dejar de responder a la interacción del usuario.
- onStop() → la actividad ha pasado completamente a segundo plano.
- onDestroy() → la actividad va a ser destruida y va a liberar sus recursos.

Frameworks para el desarrollo de aplicaciones móviles

→ Los frameworks para desarrollar apps se pueden dividir en 3 categorías:

- Frameworks nativos → permiten desarrollar apps móviles para un SO de forma nativa, usando todo el potencial de las mismas. Ej.: Android Studio / XCode (iOS). Para el desarrollo nativo es necesario conocer lenguajes de programación como Java, Kotlin y XML para Android, o Swift y Objective-C para iOS.
- Frameworks híbridos → reduce costes y tiempos, facilita el aprendizaje y desarrollo de apps móviles, y se pueden crear webs con estos entornos de trabajo. Tienen una curva de aprendizaje más suave que el desarrollo nativo.
- Frameworks webs → además de crear apps web, permite crear apps móviles mediante tecnología web.

→ La mejor opción para desarrollar apps móviles es usar frameworks nativos para aprovechar al máximo las características de los dispositivos móviles. Al usar frameworks que nos permitan crear una app, al exportarla a diferentes SOs, estos nos la limitarán bastante pudiendo ofrecer menos funcionalidades.

Tema 2: Construcción de proyectos Android

Arquitectura del sistema operativo Android

- Capa de aplicaciones → aplicaciones que forman la base de Android: cliente de correo electrónico, programa de gestión de SMS, etc. Están escritas en Java.
- Capa del marco de aplicaciones → simplifica la reutilización de componentes por el usuario.
- Capa de bibliotecas → capa escrita en C/C++ y usada por los componentes del SO, como las aplicaciones de gráficos, bases de datos, etc.
- Capa de runtime → conjunto de bibliotecas que proporcionan muchas de las funciones del SO.
- Núcleo Linux → la principal; es el núcleo o kernel que va a proporcionar todo lo relacionado con la gestión de los recursos del dispositivo: memoria, almacenamiento, comunicaciones de red, etc.

Diagrama de aplicaciones

Diagrama de Armazón de aplicaciones

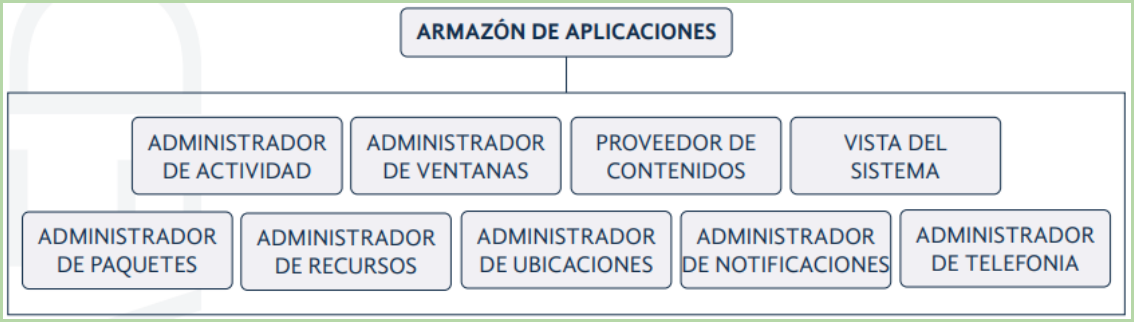
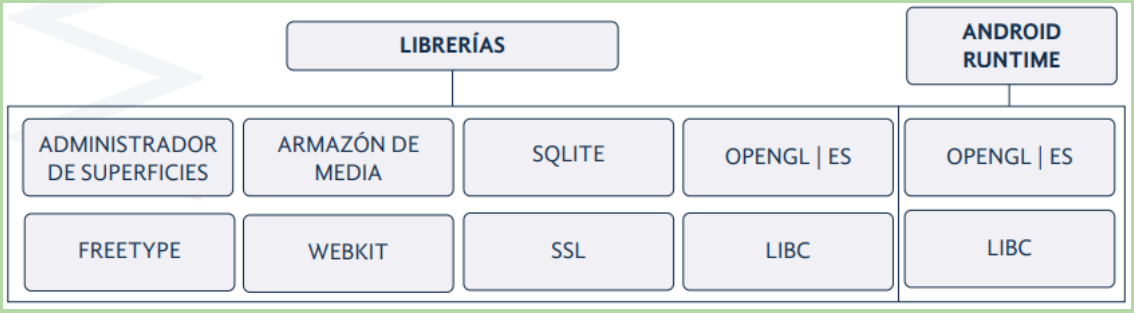


Diagrama de librerías



Tema 7: Multimedia y bases de datos

Reproducción de sonido

→ Bibliotecas de terceros mediante dos clases:

- MediaPlayer → para archivos de audio largos, como música de fondo.
- SoundPool → para archivos de audio muy cortos, como efectos de botones. Tamaño de archivo máximo 1Mb.

→ Estas clases se encargarán de realizar las conversiones necesarias para que los ficheros puedan ser reproducidos correctamente en el dispositivo.

→ Formatos → .mp3, .wav, .midi, .mp4, etc.

→ Los audios deben ir en la carpeta *raw* dentro de *res*.

→ Crear un objeto MediaPlayer → `MediaPlayer mp = MediaPlayer.create(this, R.raw.sonido);`

- play() → inicia.
- pause() → pausa o vuelve a iniciar.
- stop() → detiene.
- release() → libera los recursos. Hay que llamarlo después de stop().

Bases de datos

→ Vamos a usar bases de datos embebidas → la bbdd es un fichero dentro de la aplicación. Si desinstalamos la aplicación, la bbdd se pierde.

→ Como SGBD usaremos SQLite. Android incorpora una API para desarrollar más fácilmente las tareas necesarias.

→ Debemos heredar de la clase SQLiteOpenHelper, que tiene un constructor y dos métodos:

- onCreate() → crearemos las tablas de la bbdd.
- onUpgrade() → haremos las operaciones necesarias para actualizar la bbdd. Debemos realizar una copia de seguridad de la bbdd para volver a volcarla, ya que el método se llamará automáticamente al actualizar su versión.

→ Debemos crear un método para cerrar la conexión a la bbdd.

→ Para almacenar la bbdd debemos crear un objeto de tipo SQLiteDatabase.

Insertar valores en bases de datos

→ Usamos el método getWritableDatabase() para crear una referencia a la bbdd en modo escritura.

→ Para insertar datos podemos usar:

- Método execSQL(), pasándole las sentencias *insert*.
- Método creado por nosotros insert() al que le pasamos el nombre de la tabla y los valores

```
public void insertarUsuarios()
{
    // Obtengo los datos en modos de escritura
    bd = getWritableDatabase();
    // Si hemos abierto correctamente la base de datos
    if(bd != null)
    {
        long newRowId;
        //Insertamos 5 usuarios de ejemplo
        for (int i = 1; i <= 5; i++)
        {
            try
            {
                // Creo un ContentValues con los valores a insertar
                ContentValues values = new ContentValues();
                values.put("codigo", i);
                values.put("nombre", "Usuario" + i);
                // Inserta la nueva fila, devolviendo el valor de la clave
                // primaria de la nueva fila
                newRowId = bd.insert( table: "Usuarios",  nullColumnHack: "", values);
            }
            catch (Exception e)
            {
                newRowId = -1;
                System.out.println("Error -> " + e.toString());
            }
        }
    }
```

que queramos insertar en un objeto de tipo *ContentValues*. Con el método put() indicamos al objeto los valores a almacenar, indicando el nombre del atributo y su valor.

→ Cerramos la conexión a la bbdd al final con el método close().

Borrar valores en bases de datos

→ Usamos el método getWritableDatabase() para crear una referencia a la bbdd en modo escritura.

→ Para borrar datos podemos usar:

- Método execSQL(), pasándole las sentencias *delete*.
- Método creado por nosotros delete() al que le pasamos el nombre de la tabla, la condición o condiciones para el borrado y los valores de las condiciones. Se pueden unir las condiciones con AND y OR.

```
public void eliminarUsuario() throws Exception
{
    // Obtengo los datos en modos de escritura
    bd = getWritableDatabase();
    // Si hemos abierto correctamente la base de datos
    try
    {
        // Defino la parte del WHERE
        String selection = "codigo > ? AND nombre = ?";
        // Valores para borrar en orden
        String[] deleteArgs = { "10", "María" };
        bd.delete( table: "Usuarios", selection, deleteArgs);
    }
    catch (Exception e)
    {
        throw new Exception(e.toString());
    }
}
```

Actualizar valores en bases de datos

→ Usamos el método getWritableDatabase() para crear una referencia a la bbdd en modo escritura.

→ Para actualizar datos podemos usar:

- Método execSQL(), pasándole las sentencias *update*.
- Método creado por nosotros update() al que le pasamos el nombre de la tabla, los nuevos valores de los elementos con un objeto *ContentValues*, la condición o condiciones y los valores de las condiciones. Se pueden unir las condiciones con AND y OR.

```
bd = getWritableDatabase();
// Si hemos abierto correctamente la base de datos
try
{
    // Creo un ContentValues con los valores a insertar
    ContentValues values = new ContentValues();
    values.put("codigo", codigo);
    values.put("nombre", nombrenuevo);
    // Defino la parte del WHERE
    String selection = "nombre = ?";
    // Valores para borrar en orden
    String[] updateArgs = { nombreantiguo };
    bd.update( table: "Usuarios", values, selection, updateArgs);
}
catch (Exception e)
```

Consultar valores en bases de datos

→ Usamos el método getReadableDatabase() para crear una referencia a la bbdd en modo lectura.

→ SQLite nos proporciona la clase Cursor para seleccionar los valores de una tabla sin usar una sentencia SELECT de SQL.

- Nombre de la tabla.
- Columnas que queremos que devuelva. Se puede dejar a null si queremos todas.
- Columnas del WHERE.

OFERTAS BLACK FRIDAY

msi

Esta oferta

es como una doble victoria tuya: disfrútala ahora porque no pasa dos veces.

Ver ofertas

HASTA -40%

Tu viejo portátil ya dio lo que tenía que dar. Pásate a MSI: rápido, potente y sin dramas. Lo enciendes y estás listo para todo. Aprovecha las ofertas y despídete del modo “se cuelga cada dos por tres”.

- Valores del WHERE.
- Valores del GROUP BY.
- Valores del HAVING.
- Orden del ORDER BY.

→ Esto nos devolverá un objeto de tipo Cursor, que recorreremos:

- Primer valor → moveToFirst().
- Siguientes valores → moveToNext() → devuelve true si existe o false si no hay más elementos.

→ Para obtener los valores → getString, getInt, etc. pasando el índice del campo que queramos recuperar.

→ Para obtener el número total de elementos obtenidos por la consulta → getCount() en la clase Cursor.

→ También se puede ejecutar la consulta SQL con el método rawQuery(), pasándole la consulta, y nos devuelve un objeto Cursor.