PERSISTENCIA DE DATOS Y CONTENIDO MULTIMEDIA

PERSISTENCIA DE DATOS

- Una aplicación para móviles debería, como mínimo, guardar el estado que tenía la última vez que fue usada por el usuario. Para que cuando la vuelva a usar, la encuentre tal y como la dejó.
- Android tiene varias formas de almacenar información permanentemente:
  - Usando un SGBD como SQLite.
  - Por medio de Content Providers.
  - Preferencias y ficheros.
- En esta unidad nos vamos a centrar en las preferencias (*preferences*) y ficheros (*static files*).

PREFERENCIAS

- Las preferencias son una forma sencilla y ligera de guardar información simple, almacenando la información como un tipo de dato primitivo (integer, boolean, String…).
- Esta información se almacena según un par nombre/valor (key/value, name/value).
- Usando las preferencias podemos, entre otras cosas:
  - Guardar el tono de llamada, notificación o mensaje de WatsApp…
  - Establecer el rechazo de llamadas.
  - Activar la ocultación de identidad en las llamadas.
- Las preferencias nos permiten guardar los datos que queramos, dándole el significado que deseemos.

ALMACENAMIENTO DE DATOS SIMPLES

- Para crear un par *nombre/valor* tenemos a nuestra disposición la clase **SharedPreferences**.
- Para conseguir acceso a las preferencias tenemos que invocar al método *getSharedPreferences()*:

```
SharedPreferences misPreferencias = getSharedPreferences("prefs", MODE_PRIVATE);
```

- Los dos parámetros que se le pasan a *getSharedPreferences* tienen el siguiente significado:
  - "prefs" es el nombre asignado al fichero de preferencias. Podemos tener tantos como queramos.
  - MODE_PRIVATE se refiere al modo de creación del fichero de preferencias. De esta manera, las preferencias creadas solo pueden ser accesibles desde la propia apliación. Existen otros modos (MODE_WORLD_READABLE y MODE_WORLD_WRITEABLE), los cuales son totalmente desaconsejados debido a los problemas de seguridad que puedan provocar.

Más información sobre MODE_WORLD_READABLE

ALMACENAMIENTO DE DATOS SIMPLES

- Las preferencias pueden ser compartidas por los distintos componentes de una aplicación, pero no están disponibles para el resto de aplicaciones.
- Una vez tenemos acceso a las preferencias, para crear, modificar o borrar valores utilizaremos el interfaz *SharedPreferencesEditor*.

```
// Se crea el objeto de la clase SharedPreferences
SharedPreferences misPreferencias = getSharedPreferences("prefs", MODE_PRIVATE);
// Se crea el objeto editor que permite modificar los valores del objeto misPreferencias
SharedPreferences.Editor editor = misPreferencias.edit();
```

- La interfaz *SharedPreferencesEditor* proporciona un conjunto de métodos para establecer, modificar o eliminar los pares *nombre/valor* que definirán nuestras preferencias:

```
editor.putString("nombre", "Raistlin");
editor.putString("apellidos","Majere");
editor.putInt("edad",36);
editor.putBoolean("estaVivo", true);
```

- Nuestras preferencias no se guardarán hasta que llamemos al método *commit()* o *apply()*.

ALMACENAMIENTO DE DATOS SIMPLES

- Diferencias entre *commit()* y *apply()* :
  - *commit():* escribe las preferencias de forma síncrona y devuelve un valor booleano indicando el éxito o fracaso de la operación de escritura.
  - *apply():* escribe las preferencias de forma asíncrona y no informa del éxito de la escritura. Debido al funcionamiento asíncrono, *apply()* es el método recomendado para guardar las preferencias.

```
editor.apply();
```

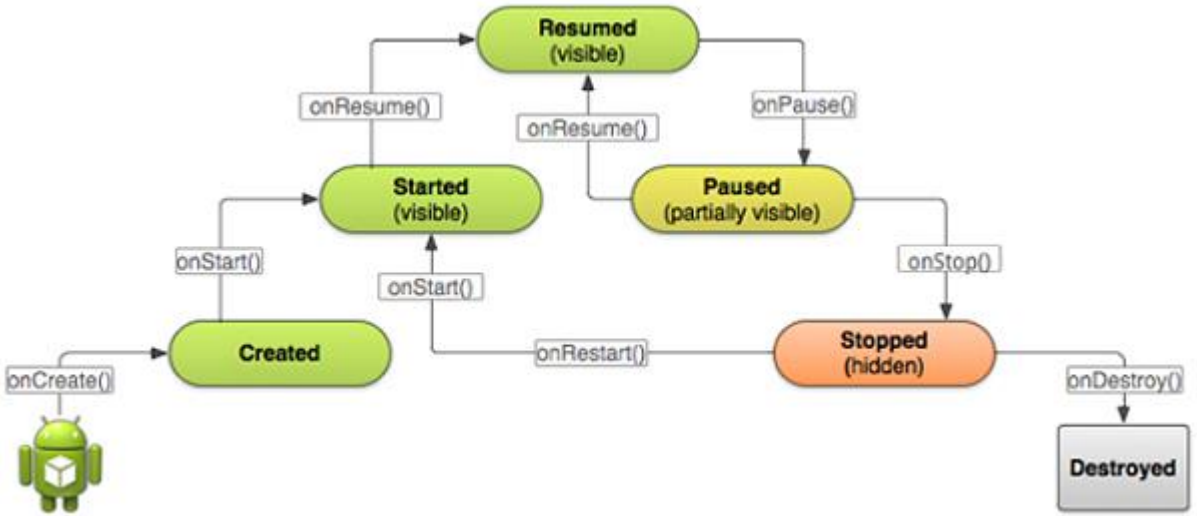
- Para recuperar las preferencias utilizamos el objeto creado *misPreferencias* de la clase *SharedPreferences*:

```
String nombre = misPreferencias.getString("nombre", "en blanco");
String apellidos = misPreferencias.getString("apellidos", "en blanco");
int edad = misPreferencias.getInt("edad", 0);
boolean estaVivo = misPreferencias.getBoolean("estaVivo", true);
```

- El primer parámetro de los métodos get es el nombre asignado con los put (*putString, putInt, etc*), y el segundo parámetro es un valor por defecto usado cuando todavía no se haya establecido un valor para la preferencia correspondiente.

ACTIVIDAD 1

- Implementar una App que le pida al usuario su nombre, apellidos y edad. Dichos valores se deben almacenar en forma de preferencias. Además hay que tener en cuenta los siguientes requisitos:
  - Un botón permitirá mostrar las preferencias en cajas de texto.
  - Las preferencias se guardarán cuando la aplicación se detenga.



Tenéis el código de ejemplo con la solución de esta actividad en drive con el nombre de proyecto "Preferencias01".

SISTEMA DE PREFERENCIAS ANDROID

- Android nos ofrece un potente y ágil armazón para trabajar con las preferencias, lo que nos permite integrar configuraciones de otras aplicaciones en nuestras propias preferencias.
- A partir de la versión de Android 3.0 (Honeycomb, API 11) se dejó de trabajar con *Preference Screen* y se comenzó a utilizar *Preference Fragment*.
- Si queremos abarcar todo el abanico de móviles con sus diferentes versiones de Android, tendremos que programar nuestras Apps para que sean compatibles tanto con *Preference Screen* como con *Preference Fragment*.

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE SCREEN (1)

Para trabajar con *Preference Screen* tenemos que seguir los siguientes pasos:

- Paso 1: Crear un fichero de recursos xml.
  - En el directorio de recurso (res) creamos un directorio que se llame *xml*, y dentro de este directorio (res/xml) es donde creamos el fichero de recursos *xml*.
  - En este ejemplo al fichero lo vamos a llamar *preferencias.xml* y su contenido será el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android">
    <EditTextPreference
        android:key="nickname"
        android:title="Nickname"
        android:summary="Tan solo escribe tu apodo">
    </EditTextPreference>
    <CheckBoxPreference
        android:key="gustanSuperheroes"
        android:title="¿Te gustan los superheroes?"
        android:summary="Para gustos los colores"
        android:defaultValue="true">
    </CheckBoxPreference>
</PreferenceScreen>
```

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE SCREEN (11)

- Paso 1: Crear un fichero de recursos xml.
  - La finalidad del fichero *preferencias.xml* es definir el contenido y el formato que tendrán nuestras preferencias.
  - La etiqueta *PreferenceScreen* es la raíz y contiene el resto de etiquetas.
  - Con *EditTextPreference* creamos una caja de texto. Y el tipo de dato que contendrá será un *String*.
  - *CheckBoxPreference* es un CheckBox y contendrá un tipo de dato boolean.
  - Dentro de cada etiquetas especificamos los **atributos**:
    - *android:key* → El nombre de la preferencia (par nombre/valor).
    - *android:title* → El texto mostrado al usuario para representar la preferencia.
    - *android:summary* → Una descripción más larga de la preferencia, mostrada en un tamaño de fuente más pequeño.
    - *android:defaultValue* → Valor por defecto que se mostrará al usuario. Y que también será asociado como valor de la preferencia en caso de que el usuario no asigne ninguno.

Más información sobre el contenido de Preference Screen y sus atributos

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE SCREEN (III)

- Paso 2: Crear una nueva actividad.
  - Una vez hemos definido la estructura de nuestras preferencias tenemos que crear una nueva actividad, la cual será invocada cada vez que el usuario quiera editar las preferencias de nuestra aplicación.
  - En este ejemplo la actividad se llamará *MisPreferencias* y hereda de *PreferenceActivity*:

```
public class MisPreferencias extends PreferenceActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        …
        …
    }
…
…
}
```

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE SCREEN (III)

- Paso 2: Crear una nueva actividad.
  - Lo siguiente que tenemos que hacer es declarar esta nueva actividad en el archivo *AndroidManifest.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mmc.preferencias02" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.mmc.preferencias02.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.mmc.preferencias02.MisPreferencias"
            android:label="@string/title_activity_mis_preferencias" >
        </activity>
    </application>

</manifest>
```

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE SCREEN (IV)

- Paso 3: Cargar el layout (o interfaz) definido en *res/xml/preferencias.xml*.
  - Tenemos que modificar el método *onCreate()* de la nueva actividad:

```
public class MisPreferencias extends PreferenceActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_mis_preferencias);

        addPreferencesFromResource(R.xml.preferencias);
    }
```

  - La línea que establece la interfaz de usuario está comentada porque ya no la necesitamos, y también podemos borrar el archivo *res/layout/activity_mis_preferencias*.
  - El nuevo layout lo cargamos a través de la línea:

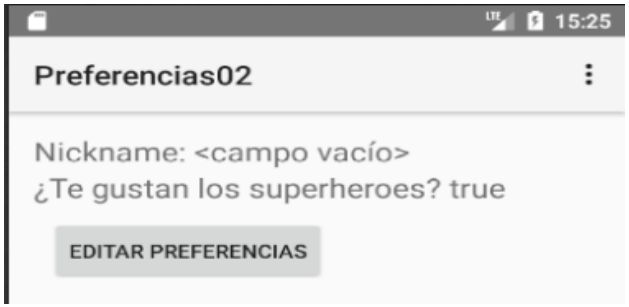
addPreferencesFromResource(R.xml.preferencias);

  - Android Studio nos indica que este método está obsoleto desde la API Level 11 (Honeycomb).

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE SCREEN (V)

- Paso 4: Lanzar la nueva actividad *(MisPreferencias)* para editar las preferencias.
  - La forma de lanzar la nueva actividad será a través de un *Intent*.
  - En este ejemplo dicho *Intent* se activará a través de un botón, aunque podría ser a través de cualquier otro medio, como un menú por ejemplo.
  - El interfaz principal de esta aplicación será el siguiente:



  - El evento *Click* lo definimos a través del atributo onClick (del widget Button):

```
<Button
    …
    android:onClick="editarPreferencias"
    …/>
```

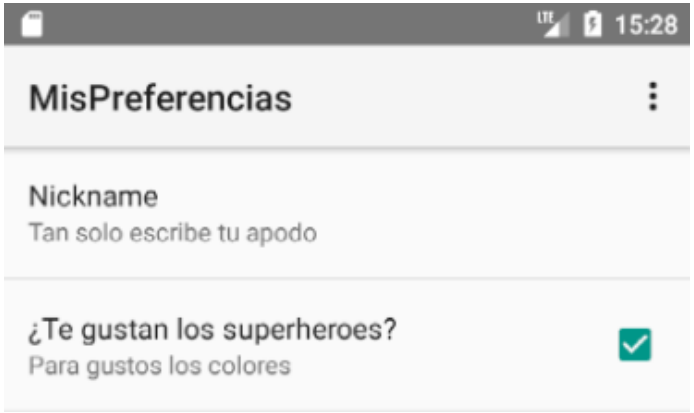
  - El código asociado a dicho botón sería:

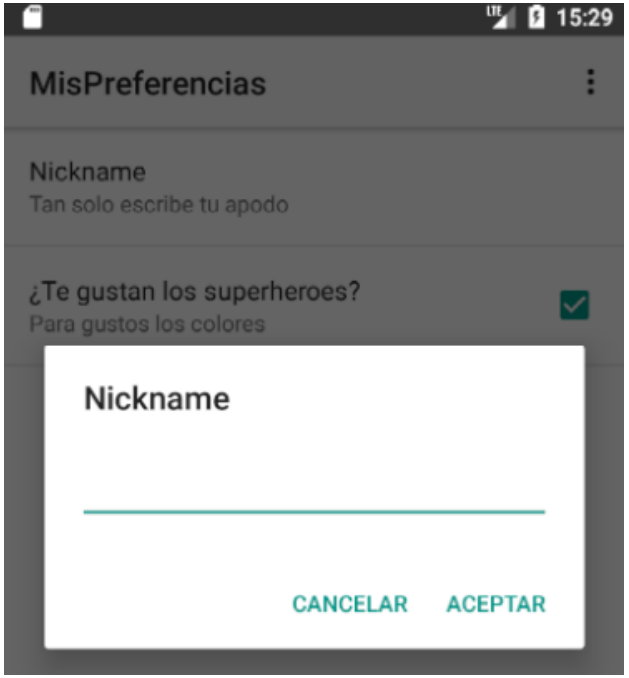
```
public void editarPreferencias(View view){
    startActivity(new Intent(this, MisPreferencias.class));
}
```

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE SCREEN (VI)

- Paso 4: Lanzar la nueva actividad *(MisPreferencias)* para editar las preferencias.
  - Cuando el usuario pulse el botón "*Editar preferencias*" verá la siguiente interfaz:





SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE SCREEN (VII)

- Paso 5: DEBUG, mostrar las preferencias en la actividad principal.
  - Si nos fijamos en el ciclo de vida de una aplicación, veremos que hay que implementar el método *onResume()* si queremos que se muestren las preferencias cada vez que la aplicación esté en primer plano de ejecución:

```
public void onResume(){
    super.onResume();
    String nickname;
    boolean gustan;

    TextView tv_nickname = (TextView) findViewById(R.id.textViewNombre);
    TextView tv_gustar = (TextView) findViewById(R.id.textViewGustar);

    SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);

    nickname = prefs.getString("nickname", "<campo vacío>");
    gustan = prefs.getBoolean("gustanSuperheroes", true);

    tv_nickname.setText("Nickname: " + nickname);
    tv_gustar.setText("¿Te gustan los superheroes? " + new Boolean(gustan).toString());
}
```

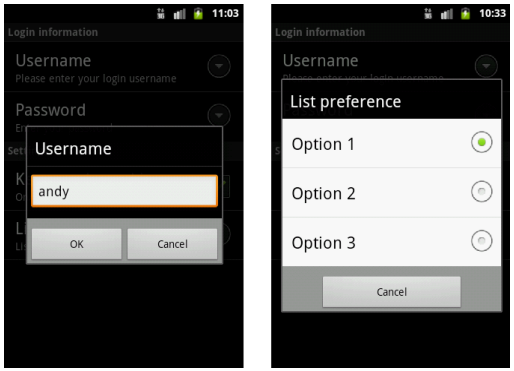
SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE SCREEN (VII)

- Paso 5: DEBUG, mostrar las preferencias en la actividad principal.
  - El método que aquí nos interesa es el *PreferenceManager.getDefaultSharedPreferences(this):*
    - Se le pasa como parámetro tan solo el contexto (this).
    - Es un método estático que pertenece a la clase *PreferenceManager*.
    - Devuelve un objeto de la clase *SharedPreferences*.
  - Al heredar de *PreferenceActivity*, los cambios se escriben automáticamente sin tener que hacer nosotros explicitamente *commit()* o *apply()*.

ACTIVIDAD 2

- Investigar acerca de *PreferenceCategory* y *PreferenceScreen* anidados.
- Basándoos en el ejemplo anterior o implementando una nueva aplicación, diseñar un entorno de preferencias más avanzado que contenga tanto etiquetas *PreferenceCategory* como *PreferenceScreen* anidados.
- Además de los elementos vistos (*CheckBoxPreference* y *EditTextPreference*) probar elementos nuevos como *ListPreference*, *RingtonePreference*, etc.



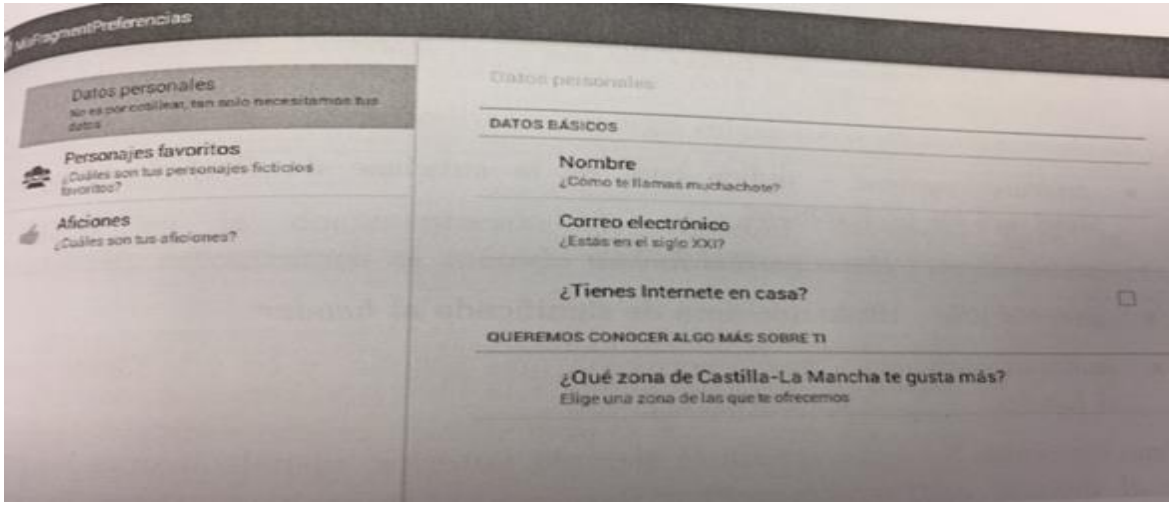
- Esta actividad será evaluable y tendréis que subirla a vuestra carpeta de entregas de la asignatura en google drive, con fecha límite de entrega el 11/05/2017.

Tenéis el código del ejemplo anterior sobre *PreferenceScreen* en drive con el nombre de proyecto "Preferencias02".

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE FRAGMENT (I)

- A partir de Android 3.0, la clase que hereda de *PreferenceActivity* carga unas cabeceras (*headers*), las cuales apuntan a subclases de *PreferenceFragment* que son las encargadas de mostrar las preferencias en la pantalla.
- Así se verían las preferencias utilizando *PreferenceFragment*:



- En la imagen anterior podemos ver los *headers* a la izquierda, y cuando se hace clic sobre alguno de ellos, se muestra a la derecha el conjunto de preferencias que contiene dicho *header*.

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE FRAGMENT (II)

- *Preference Headers* son unos niveles de agrupación superiores que permiten una más clara y mejor visualización de las preferencias.
- Para definir *Preference Headers* hay que crear un fichero de recursos XML en el directorio *res/xml*. En este ejemplo lo vamos a llamar preferences_headers.xml y su contenido será:

```
<?xml version="1.0" encoding="utf-8"?>
<preference-headers xmlns:android="http://schemas.android.com/apk/res/android">
    <header
        android:fragment="com.mmc.preferencefragment01.DatosPersonales"
        android:icon="@drawable/datos_personales"
        android:title="Datos personales"
        android:summary="No es por cotillear, tan solo necesitamos tus datos" />
    <header
        android:fragment="com.mmc.preferencefragment01.PersonajesFavoritos"
        android:icon="@drawable/personajes_favoritos"
        android:title="Personajes favoritos"
        android:summary="¿Cuáles son tus personajes ficticios favoritos?" />
    <header
        android:fragment="com.mmc.preferencefragment01.Aficiones"
        android:icon="@drawable/aficiones"
        android:title="Aficiones"
        android:summary="¿Cuáles son tus aficiones?" />
</preference-headers>
```

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE FRAGMENT (III)

- Cada elemento *header* contiene los siguientes atributos:
  - *android:fragment* → Indica cuál es la subclase de *PreferenceFragment* que mostrará las preferencias.
  - *android:icon* → Icono para el *header*.
  - *android:title* → Título que dota de significado al *header*.
  - *android:summary* → Una descripción más amplia, y en una fuente menor, para el *header*.
- Para cargar los *headers* invocamos al método ***loadHeadersFromResource()***:

```
public class MisFragmentPreferencias extends PreferenceActivity {
…
    @Override
    public void onBuildHeaders(List<Header> target) {
        super.onBuildHeaders(target);
        loadHeadersFromResource(R.xml.preference_headers, target);
    }
…
}
```

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE FRAGMENT (IV)

- Lo siguiente es implementar las clases ***DatosPersonales***, ***PersonajesFavoritos*** y ***Aficiones***.
- Por ejemplo, ***DatosPersonales*** quedaría así:

```
public class DatosPersonales extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.datos_personales);
    }
}
```

- Esta clase hereda de ***PreferenceFragment***, y desde el método *onCreate()* llamamos al método *addPreferencesFromResource()*.
- El contenido del fichero de recursos que se carga son los elementos *PreferenceScreen* que hemos visto en esta unidad.
- El método *addPreferencesFromResource()* de la clase *PreferenceFragment* no está obsoleto como sí lo estaba el de *PreferenceScreen*.

Más información sobre PreferenceFragment

SISTEMA DE PREFERENCIAS ANDROID

PREFERENCE FRAGMENT (V)

- Hay que tener en cuenta que las subclases de *PreferenceActivity* deben sobrescribir el método ***isValidFragment()***, ya que si no lo hacemos se producirá una excepción al ejecutar nuestra aplicación en dispositivos con una versión posterior a *KITKAT*.
- Para evitar que nos de la excepción anterior debemos sobrescribir dicho método:

```
@Override
protected boolean isValidFragment (String fragmentName) {
    if ([NOMBRE_DE_TU_FRAGMENT].class.getName().equals(fragmentName))
       return true;
    return false;
}
```

- En nuestro ejemplo este método quedaría de la siguiente forma:

```
@Override
protected boolean isValidFragment (String fragmentName) {
    if (Aficiones.class.getName().equals(fragmentName)) return true;
        else if (DatosPersonales.class.getName().equals(fragmentName)) return true;
            else if (PersonajesFavoritos.class.getName().equals(fragmentName)) return true;
    return false;
}
```

- Tener en cuenta que la visualización de las *Preference Headers* depende del tamaño y resolución de la pantalla, con lo que no se verán igual de "bonitas" en todos los dispositivos.

ACTIVIDAD 3

- Basándoos en el ejemplo de este apartado (código e imágenes), desarrollar una aplicación que implemente las preferencias a través de *PreferenceFragment* y *Preference Headers*.
- La aplicación debe tener 3 *headers* (Datos Personales, Personajes Favoritos y Aficiones), y cada uno de ellos cargará sus correspondientes preferencias.
- Entre otros, usar los elementos *MultiSelectListPreference* y *ListPreference* dentro de *PreferenceScreen*.
- Recordar que para una correcta visualización debéis modificar la resolución del emulador.

Tenéis el código del ejemplo anterior sobre *PreferenceFragment* en drive con el nombre de proyecto "PreferenceFragment01".

SISTEMA DE PREFERENCIAS ANDROID

COMPATIBILIDAD HACIA ATRÁS

- Si queremos que nuestras Apps funcionen en cualquier teléfono móvil, independientemente de su versión de Android, tendremos que añadir algunas líneas de código.
- Tenemos que crear dos clases que hereden de *PreferenceActivity*, una para las versiones anteriores a **Honeycomb**, y otra para las versiones posteriores.
- Para el ejemplo de la actividad anterior tendríamos que añadir la siguiente clase:

```
public class MisViejasPreferencias extends PreferenceActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);

        addPreferencesFromResource(R.xml.preferencias);
    }
}
```

- El fichero de recursos xml que contiene las preferencias (*res/xml/preferencias.xml*) para las versiones anteriores a **Honeycomb**, tiene que ser diseñado para que agrupe todas las preferencias que existan usando *Preference Headers*. En nuestro caso serían *datos_personales.xml, personajes_favoritos.xml* y *aficiones.xml*.

SISTEMA DE PREFERENCIAS ANDROID

COMPATIBILIDAD HACIA ATRÁS

- Para comprobar la versión del sistema operativo Android, implementamos dentro del botón (o del widget que sea) que carga las preferencias, el siguiente código:

```
public void editarPreferencias(View view){
    if (Build.VERSION.SDK_INT<Build.VERSION_CODES.HONEYCOMB)
        startActivity(new Intent(this, MisViejasPreferencias.class));
    else
        startActivity(new Intent(this, MisFragmentPreferencias.class));
}
```

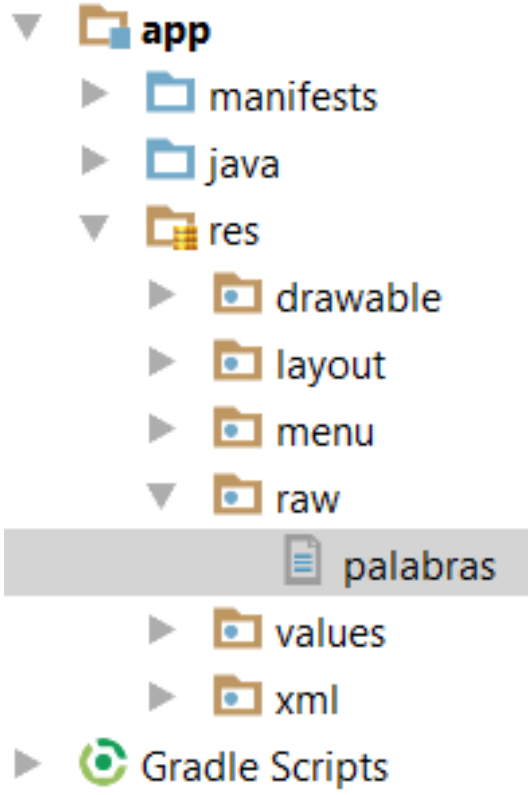
- Este código comprueba si la versión del sistema operativo es anterior a **Honeycomb**. En caso afirmativo se lanzan las "viejas" preferencias, y en caso contrario, se lanzan las "nuevas" preferencias.

FICHEROS ESTÁTICOS

- En ocasiones necesitamos almacenar información pero vemos innecesario utilizar una base de datos o las preferencias. Para ello Android nos ofrece ficheros estáticos, los cuales son empaquetados con la aplicación.
- Los ficheros estáticos sólo admiten operaciones de lectura, no se pueden crear ni modificar en tiempo de ejecución. Para actualizar el contenido de dichos ficheros sería necesario actualizar la aplicación al completo.
- Si en una determinada situación necesitamos tener acceso de lectura/escritura sobre un fichero, Android nos proporciona todos los mecanismos de trabajo con ficheros de Java.

FICHEROS ESTÁTICOS

- Para empezar debemos añadir el fichero como recurso (*res/raw*):



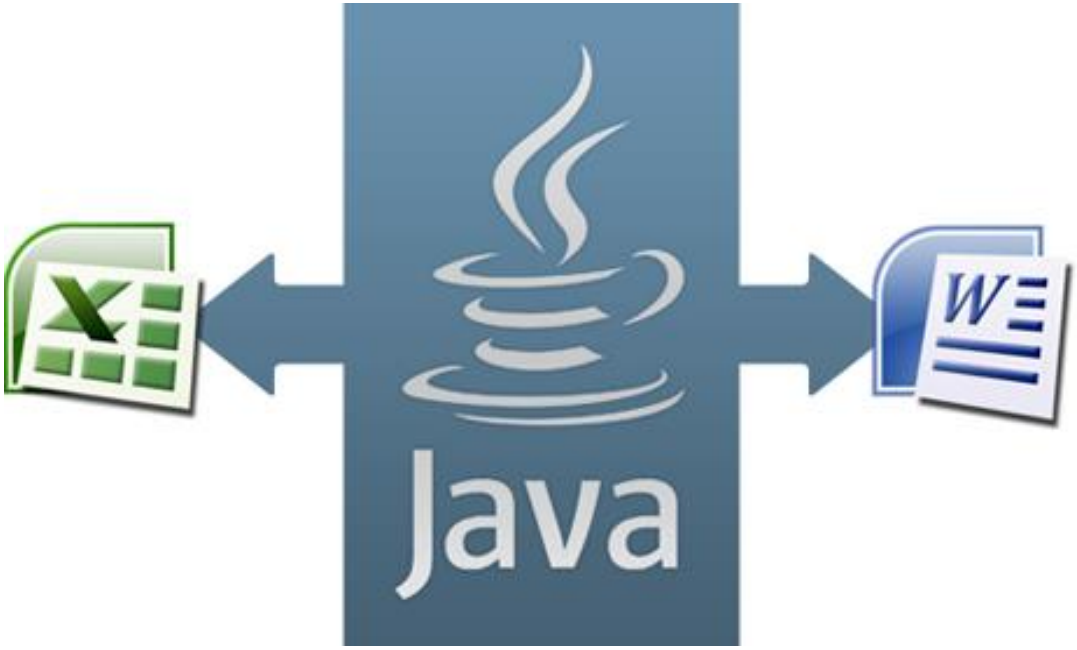
- El contenido del recurso *res/raw/palabras* es un fichero de texto, que podría contener un diccionario, frases, etc.

FICHEROS ESTÁTICOS

- Para conseguir acceso al fichero necesitamos un objeto de la clase *InputStream*:

```
Resources r = getResources();
InputStream in = r.openRawResource(R.raw.palabras);
```

- Desde este momento podemos leer el contenido del fichero usando todas las clases y métodos proporcionados por Java.



ACTIVIDAD 4

- Implementar una sencilla App que lea el contenido de un fichero estático y lo muestre en pantalla.

Tenéis el código del esta actividad en drive con el nombre de proyecto "FicherosEstáticos".

CONTENIDO MULTIMEDIA

- Android proporciona de forma nativa una serie de formatos multimedia, códec y protocolos de red. Sin embargo, hay que tener en cuenta que algunos de ellos, solo son soportados por Android a partir de una determinada versión.
- Protocolos de red usados para el streaming de audio/video:
  - RTSP (RTP, SDP)
  - HTTP/HTTPS streaming progresivo.
  - HTTP/HTTPS streaming en vido (a partir de Android 3.0).

    HTTPS no está soportado para versiones anteriores a Androdi 3.1.
- Audio:
  - AAC LC.
  - FLAC.
  - MP3.
  - MIDI.
  - Ogg Vorbis.

CONTENIDO MULTIMEDIA

- Imagen:
  - JPEG.
  - GIF.
  - PNG.
  - BMP.
  - WebP.
- Vídeo:
  - H.263.
  - H.264 AVC.
  - MPEG-4 SP.
  - VP8.

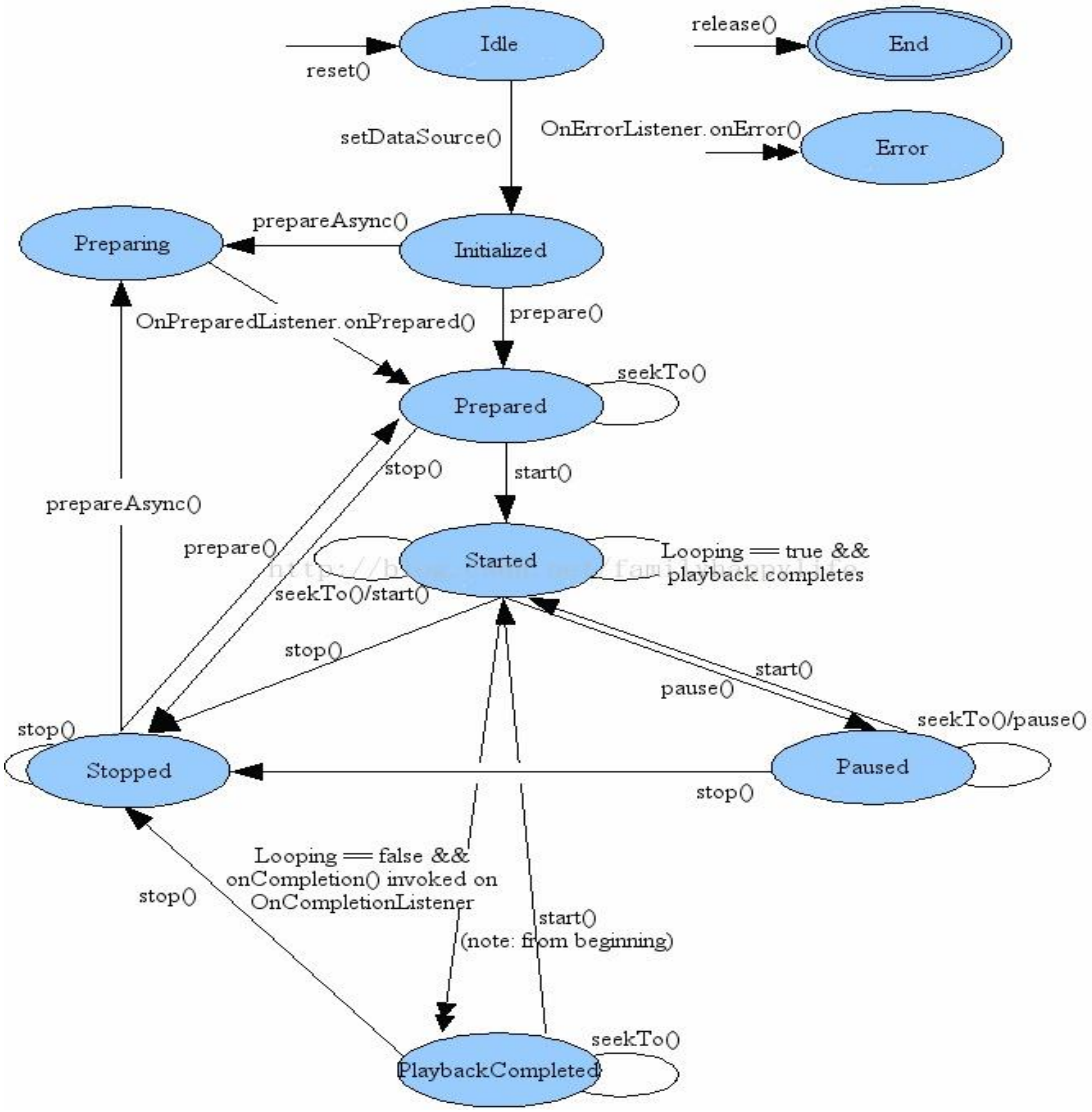
Más información sobre formatos multimedia soportados

REPRODUCCIÓN DE AUIDO Y VIDEO (MediaPlayer)

- La clase **MediaPlayer** es la encargada de controlar la reproducción de audio/vídeo y streams. Dicha reproducción se gestiona como una máquina de estados.
- Podemos identificar los siguientes estados en la reproducción (**MediaPlayer**):
  1. ***Idle***.
  2. Inicialización del reproductor (Media Player) con contenido para reproducir: ***initialized***.
  3. Preparación del Media Player: ***prepared***.
  4. Comienzo de la reproducción: ***started***.
  5. Pausa o parada de la reproducción: ***pause*** or ***stop***.
  6. La reproducción se ha completado: ***completed***.
- Es muy importante tener en cuenta estos estados a la hora de programar nuestras aplicaciones.

Más información sobre la clase MediaPlayer

REPRODUCCIÓN DE AUIDO Y VIDEO (MediaPlayer)



MediaPlayer

INICIALIZACIÓN DEL REPRODUCTOR

- Para poder reproducir contenido multimedia es necesario que estemos en el estado ***prepared***, pero primero hay que pasar por el estado ***idle***, para ello basta con crear un objeto de la clase *MediaPlayer*.

```
MediaPlayer mediaPlayer = new MediaPlayer();
```

- A continuación debemos pasar al estado ***Initialized***:

```
mediaPlayer.setDataSource();
```

- El método ***setDataSource()*** está sobrecargado y admite varias formas de invocarlo. Lo importante es que podemos pasarle contenido multimedia de cuatro formas distintas:
  - A través de un identificador de recurso (audio o vídeo almacenado en el directorio de recursos res/raw).
  - Mediante una ruta local a un archivo almacenado en el sistema, por ejemplo en nuestra tarjeta SD.
  - Usando una URI para reproducir contenido online (streaming).
  - Y desde un Content Provider.

MediaPlayer

PREPARACIÓN DEL CONTENIDO

- Para pasar al estado ***prepared*** tan solo tenemos que invocar al método ***prepare()***.

```
mediaPlayer.prepare();
```

- En este momento podemos comenzar la reproducción. Pero antes debemos conocer otra forma de crear nuestro objeto **MediaPlayer**:

```
MediaPlayer mediaPlayer = MediaPlayer.create();
```

- El método ***create()*** es estático y también está sobrecargado. Permite que pasemos el contenido multimedia a través de las cuatro formas que hemos visto (res/raw, fichero del sitema, URI y Content Provider).
- Sin embargo, hay que tener en cuenta que el objeto devuelto por ***create()*** ya se encuentra en el estado ***prepared***, luego no debemos invocar al método ***prepare()*** otra vez.

MediaPlayer

CONTROLANDO LA REPRODUCCIÓN

- Para controlar la reproducción, tenemos a nuestra disposición principalmente tres métodos:
  - ***start();*** `mediaPlayer.start();`
  - ***pause();*** `mediaPlayer.pause();`
  - ***stop();*** `mediaPlayer.stop();`
- En el diagrama de estados podemos observar que tenemos un estado ***stopped*** y un estado ***paused***.
- Para regresar al estado ***started*** desde ***paused*** tan solo tenemos que llamar al método ***start()***, sin embargo, desde el estado ***stopped*** no podemos pasar directamente al estado ***started***, sino que debemos pasar por ***prepared***, para ello, tal como indica el diagrama habrá que invocar primero al método ***prepare()*** y a continuación ***start()***.
- Además de estos tres métodos, la clase **MediaPlayer** nos ofrece una enorme funcionalidad para controlar nuestra reproducción: ***getCurrentPosition(), getDuration(), isLooping(), isPlaying(), seekTo(), selectTrack(), etc.***

MediaPlayer

FINALIZACIÓN DE LA REPRODUCCIÓN

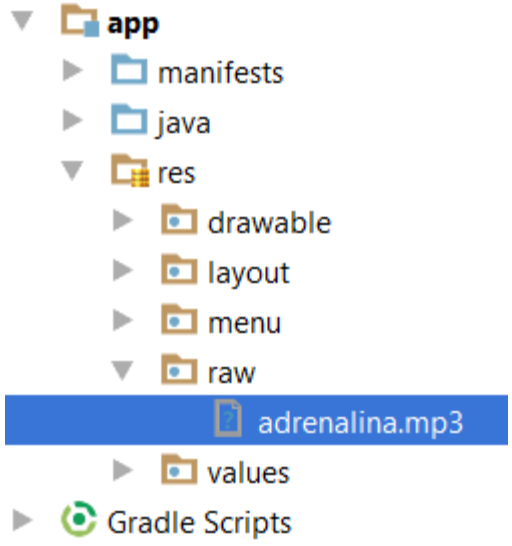
- Una vez haya acabado la reproducción y no se necesite el objeto **MediaPlayer**, debemos invocar el método ***release()***, para liberar todos los recursos asociados a dicha reproducción.

```
mediaPlayer.release();
```

MediaPlayer

CASO PRÁCTICO

- Vamos a ver todo este funcionamiento con un sencillo reproductor de audio en el que vamos a reproducir una única canción. El código completo lo tenéis en el fichero *MediaPlayer01* en la carpeta del Drive.
- El contenido multimedia lo vamos a cargar desde el directorio de recursos (res).
- Lo primero será crear un directorio dentro de "res" que se llame "raw".
- Después tenemos que copiar la canción dentro del directorio "raw" (*MediaPlayer01\app\src\main\res\raw*).



MediaPlayer

CASO PRÁCTICO

- Esta App va a estar compuesta por tres widget *Button* (play, stop y pause) y un widget *TextView* que nos permita mostrar en qué estado estamos, es decir, nos servirá de debug.

- Usaremos la propiedad *onClick* del widget Button para invocar los distintos métodos:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/play"
    android:id="@+id/playButton"
    android:onClick="play"
    android:layout_alignParentTop="true"
    android:layout_alignParentStart="true" />
```

MediaPlayer

CASO PRÁCTICO

- Lo primero es inicializar y preparar el reproductor. En este caso usaremos el método ***create()***, aunque también veremos algún ejemplo con ***setDataSource()***.

```
public class MainActivity extends Activity{
    MediaPlayer mediaPlayer;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mediaPlayer = MediaPlayer.create(this, R.raw.adrenalina);
    }
}
```

MediaPlayer

CASO PRÁCTICO

- El código asociado al Button PLAY y PAUSE quedaría así:

```
public void play(View view){
    TextView t = (TextView) findViewById(R.id.textView);
    if (mediaPlayer.isPlaying()){
        t.setText("Ya estás escuchando mÚsica, ¿qué más quieres
chaval?");
    }
    else {
        mediaPlayer.start();
        t.setText("Tu MP está parado, tranqui que le hago un
start()");
    }
}
public void pause(View view){
    TextView t = (TextView) findViewById(R.id.textView);
    if (mediaPlayer.isPlaying()) {
        mediaPlayer.pause();
        t.setText("Acabas de pausar tu MP");
    }
    else {
        t.setText("Tu MP no está en ejecuciÓn, luego no lo puedes
pausar");
    }
}
```

MediaPlayer

CASO PRÁCTICO

- Dentro del método que controla el STOP tenemos que tener cuidado con las excepciones que puede provocar el método ***prepare()***.

```
public void stop(View view) throws IOException {
    TextView t = (TextView) findViewById(R.id.textView);
    if (mediaPlayer!=null && mediaPlayer.isPlaying()) {
        mediaPlayer.stop();

        try {
            mediaPlayer.prepare();
            t.setText("La música estaba sonando pero acabas de hacer un
stop() y un prepare() a tu MP");
        } catch (IOException e) {
            e.printStackTrace();
        } catch (IllegalStateException e) {
            e.printStackTrace();
        }
    }
    else {
        t.setText("La música no suena, el MP está parado, ¿por qué haces
un stop()?");
    }
}
```

MediaPlayer

CASO PRÁCTICO

- Faltaría la parte del estado *Completed*, es decir, cuando se ha terminado la reproducción.
- Es muy importante que liberemos todos los recursos asociados a nuestro reproductor una vez que hayamos terminado con él.
- El método ***onCreate()*** quedaría de la siguiente forma añadiéndole el control del estado *Completed* con el método y listener para dicho estado:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mediaPlayer = MediaPlayer.create(this, R.raw.adrenalina);

    mediaPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener()
{
        @Override
        public void onCompletion(MediaPlayer mp) {
            TextView t = (TextView) findViewById(R.id.textView);
            mediaPlayer.release();
            t.setText("La reproducción ha terminado, acabo de hacer un
release()");

        }
    });
}
```

CONTROLES ESTANDAR (MediaControler)

- Para facilitarnos la vida a la hora de controlar nuestras reproducciones, Android incluye la clase ***MediaController***.
- Esta clase nos ofrece los botones típicos de reproducción: play, pause, rewind, fast forward y la barra de progreso.

- ***MediaController*** también se encarga de sincronizar los controles con el estado de nuestro ***MediaPlayer***.
- ***MediaController*** nos ofrece la interfaz típica al que todo usuario está acostumbrado cuando reproduce vídeo o audio en su teléfono móvil.

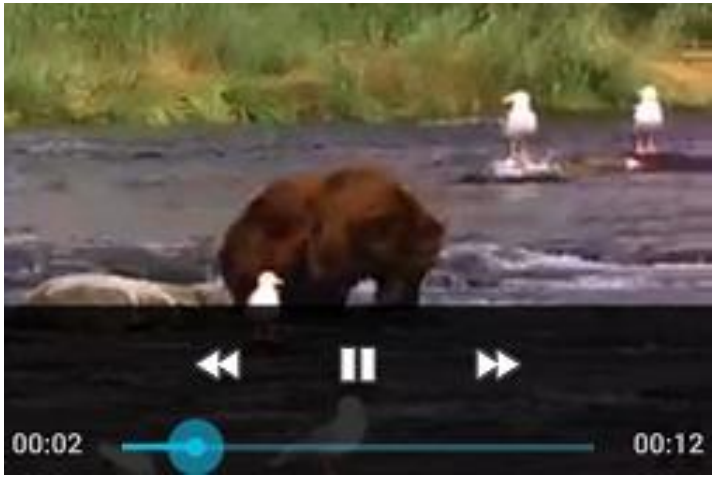
CONTROLES ESTANDAR (MediaControler)

- Aunque podemos insertar el ***MediaController*** desde la paleta (sección *Expert*), la forma correcta de trabajar con esta clase es instanciarla desde el código:

```
MediaController mediaContoller = new MediaController(Context context);
```

- No debemos olvidar que para nosotros es un widget, con lo que tendremos que hacer la siguiente importación:

```
import android.widget.MediaController;
```



CONTROLES ESTANDAR (MediaControler)

- Si queremos usar ***MediaController*** con la clase ***MediaPlayer***, tenemos que implementar el interfaz *MediaController.MediaPlayerControl*. En concreto los métodos que tenemos que implementar son:

```
public boolean canPause() {}

public boolean canSeekBackward() {}

public boolean canSeekForward() {}

public int getAudioSessionId() {}

public int getBufferPercentage() {}

public int getCurrentPosition() {}

public int getDuration() {}

public boolean isPlaying() {}

public void pause() {}

public void seekTo() {}

public void start() {}
```

CONTROLES ESTANDAR (MediaControler)

- Además necesitamos invocar a otros tres métodos de la clase ***MediaController***: uno para especificar el ***MediaPlayer*** que se controlará, otro para especificar la vista que determinará el ancho del control, y otro para mostrar el control en la pantalla.

```
mediaController.setMediaPlayer(MediaController.MediaPlayerControl player);
mediaController.setAnchorView(View view);
mediaController.show(int timeout);
```

Más información sobre la clase MediaController.

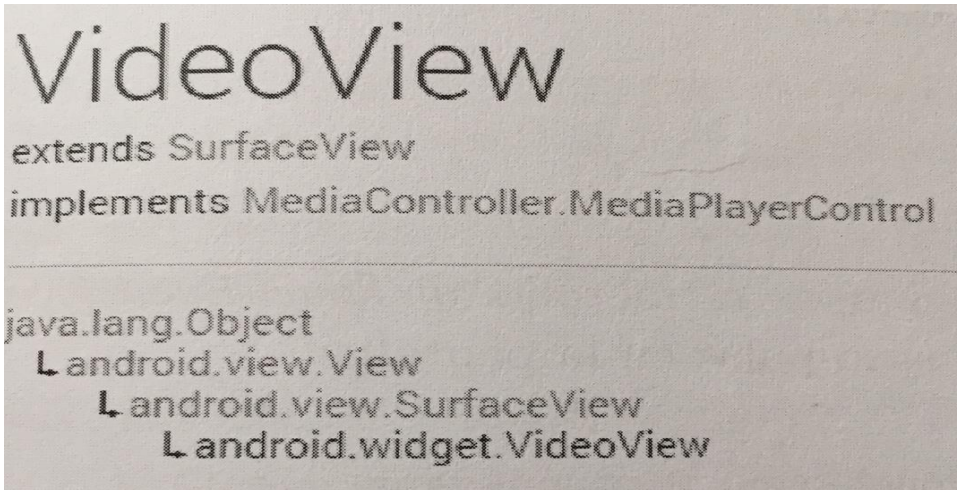
ACTIVIDAD 5

- Desarrollar una App que reproduzca una canción teniendo en cuenta los siguientes requisitos:
  - Utilizar la clase *MediaPlayer*.
  - Usar el método *setDataSource()* para cargar el contenido a reproducir. La canción a reproducir estará en el directorio de recursos *res/raw*.
  - Para controlar la reproducción utilizar la clase *MediaController*.
  - Usar un *listener* para comenzar la reproducción cuando el contenido multimedia esté preparado (*MediaPlayer.OnPreparedListener*).

- Esta actividad será evaluable y tendréis que presentarla y subirla a vuestra carpeta de entregas de la asignatura en google drive, con fecha límite de entrega el 25/05/2017.

REPRODUCCIÓN DE VIDEO

- La reproducción de vídeo, en comparación con la de audio, requiere de algún paso extra.
- Lo primero que necesitamos es un contenedor donde poder reproducir dicho vídeo.
- Tenemos dos posibilidades:
  - Usar la clase **VideoView**. Esta clase dispone de un conjunto de métodos que nos permiten cargar contenido desde distintas fuentes y controlar la reproducción.
  - Crear nosotros mismos el contenedor y controlar la reproducción usando la clase **MediaPlayer**. Esto se hace a través de la clase **SurfaceView**.



REPRODUCCIÓN DE VIDEO

- La clase *VideoView* hereda de *SurfaceView* e implementa *MediaController.MediaPlayerControl*. Esto nos da mucha ventaja porque no tenemos que preocuparnos por diseñar el contenedor donde reproduciremos el vídeo, y tenemos a nuestra disposición todos los métodos vistos hasta ahora.
- Comenzamos declarando las variables de las clases **VideoView** y **MediaController**:

```
public class MainActivity extends Activity {

    VideoView videoView;
    MediaController mediaController;
...
}
```

REPRODUCCIÓN DE VIDEO

- Dentro del método *onCreate()*:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Obtenemos la referencia al widget VideoView
    videoView = (VideoView) findViewById(R.id.videoView);
    // Creamos el objeto MediaController
    mediaController = new MediaController(this);
    // Establecemos el ancho del MediaController
    mediaController.setAnchorView(videoView);
    // Al contenedor VideoView le añadimos los controles
    videoView.setMediaController(mediaController);
    // Cargamos el contenido multimedia (el vídeo) en el VideoView
    videoView.setVideoURI(Uri.parse("android.resource://" + getPackageName() + "/" +
R.raw.magia));

    // Registramos el callback que será invocado cuando el vídeo esté cargado y
    // preparado para la reproducción
    videoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
        @Override
        public void onPrepared(MediaPlayer mp) {
            mediaController.show(10000);
            videoView.start();
        }
    });
}
```

REPRODUCCIÓN DE VIDEO

- Y para terminar programamos el método *onTouchEvent()*, para que se muestre el **MediaControl** cuando el usuario pulse en la pantalla:

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    mediaController.show();
    return false;
}
```

- El método *setVideoURI()* lo utilizamos para cargar el contenido a reproducir:

```
videoView.setVideoURI(Uri.parse("android.resource://" + getPackageName() + "/" +
R.raw.magia));
```

- Mediante un String se indica el recurso: (“android.resource://” + getPackageName() + “/” + R.raw.magia).
- Con el método *Uri.parse()* se convierte ese string a un objeto de la clase ***Uri***, que es lo que espera el método *setVideoURI*.

Tenéis el código completo del ejemplo en la carpeta compartida del drive en el fichero “VideoView”.

STREAMING DE AUDIO Y VIDEO

- El streaming en Android es una tarea relativamente sencilla.
- Se puede reproducir cualquier contenido publicado en Internet (videos de YouTube, Vimeo, etc).
- El contenido que nosotros reproduciremos está alojado en un hosting web, es decir, un alojamiento web donde se puede almacenar cualquier tipo de fichero.
- Dicho hosting tiene asociado un nombre de dominio para poder acceder a él. En nuestro caso trabajaremos con el dominio *mim.zz.mu*, y para acceder al contenido de este capítulo la URL es http://mim.zz.mu/ut4_multimedia/.
- Si dicha URL hubiera dejado de estar operativa, en Google podremos encontrar multitud de sitios para poder reproducir contenido online.

STREAMING DE AUDIO Y VIDEO

- Vamos a trabajar sobre los dos últimos ejemplos para reproducir vídeo y para audio.
- Para dotar a nuestra App con esta funcionalidad sólo vamos a tener que modificar 2 líneas de código.
- Lo primero que tenemos que hacer es incluir el permiso necesario para que nuestra aplicación pueda acceder a Internet. En el manifiesto de Android añadimos la siguiente línea:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mmc.streamingvideo" >

    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.mmc.streamingvideo.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

STREAMING DE AUDIO Y VIDEO

- Y lo segundo es modificar el origen del contenido a reproducir:
  - Para el vídeo, tendríamos que cambiar la línea:

```
videoView.setVideoURI(Uri.parse("android.resource://" + getPackageName() + "/" +
R.raw.magia));
```

Por esta:

```
videoView.setVideoURI(Uri.parse("http://mim.zz.mu/ut4_multimedia/ma
gia.webm"));
```

  - Y para el audio, cambiamos la línea:

```
mediaPlayer.setDataSource(this, Uri.parse("android.resource://" + getPackageName()
+ "/" + R.raw.metodo_para_escapar));
```

Por:

```
mediaPlayer.setDataSource(this,
Uri.parse("http://mim.zz.mu/ut4_multimedia/cirujano/04_aire.ogg"));
```

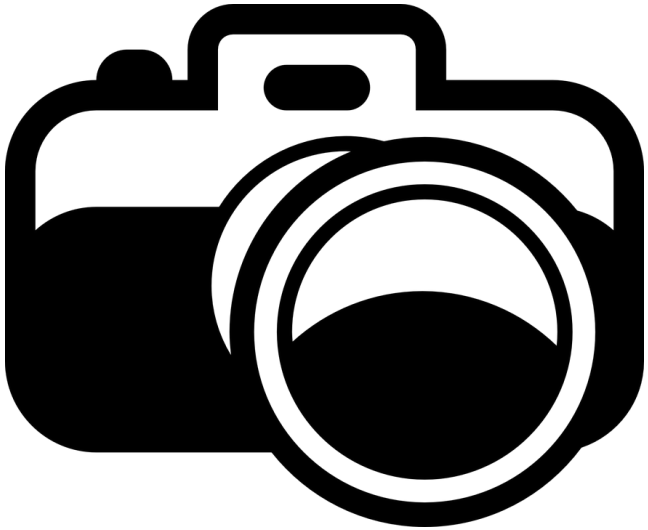
ACTIVIDAD 6

- Desarrollar una App que haga un streaming de audio desde la URL: http://min.zz.mu/ut4_multimedia/cirujano (o desde otra ubicación adecuada).

La solución para esta actividad la podréis encontrar en la carpeta compartida de drive con el nombre *"StreamingAudio"*.

CAPTURA DE FOTOS

- Hoy en día, cualquier móvil de gama media, lleva integrada una cámara de calidad más que aceptable.
- Trabajar con la cámara de fotos/vídeo en Android es relativamente fácil. Para capturar fotos dentro de nuestra aplicación disponemos, básicamente, de dos mecanismos:
  - Usar un *intent*, delegando a la aplicación nativa de Android todo el trabajo sucio.
  - Programar nosotros mismos una aplicación que controle directamente la cámara de fotos.
- La primera opción es la más adecuada en la mayoría de los casos, y es la que vamos a ver en esta sección.



CAPTURA DE FOTOS

- El siguiente código muestra como lanzar el *intent*:

```
static final CAPTURA_IMAGEN = 1;
public void hacerFoto (){
    Intent hacerFotoIntent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if (hacerFotoIntent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(hacerFotoIntent, CAPTURA_IMAGEN);
    }
}
```

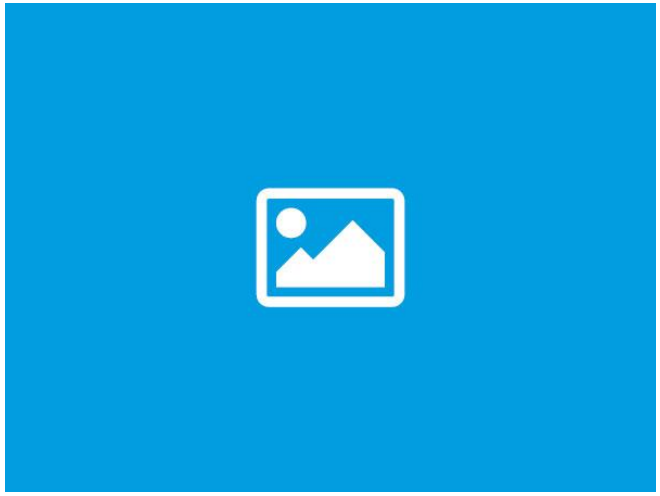
- La comprobación del *if* es importante, ya que el método *resolveActivity()* devuelve la actividad que debe manejar el *intent* que vamos a lanzar. Así evitamos que nuestra aplicación falle inesperadamente, en caso de que en el móvil donde se ejecute nuestra App no disponga de una aplicación capaz de manejar la cámara.

CAPTURA DE FOTOS

Obtención de Thumbnails:

- Por defecto, la foto es devuelta como un *thumbnail*, que es una miniatura de la imagen (más ligera) que sirven para una mejor organización y visualización.
- Dicho *thumbnail* va integrado dentro del propio *intent* como *datos extras*, el cual recibimos a través del método *onActivityResult()*.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == CAPTURA_IMAGEN_THUMBNAIL && resultCode == RESULT_OK) {
        Bundle extras = data.getExtras();
        Bitmap imageBitmap = (Bitmap) extras.get("data");
        imageView.setImageBitmap(imageBitmap);
    }
}
```



CAPTURA DE FOTOS

Guardar Fotos a Tamaño Completo:

- Para guardar la foto a tamaño real hay que proporcionar una ruta completa donde almacenarla.
- Normalmente, todas las fotos se almacenan en un directorio de acceso público para el resto de aplicaciones. Para obtener dicho directorio tenemos el método *getExternalStoragePublicDirectory()*, pasándole como argumento ***DIRECTORY_PICTURES***.
- Al tratarse de un directorio de acceso público, es necesario otorgar permisos de escritura y lectura a nuestras aplicaciones (*READ_EXTERNAL_STORAGE* y *WRITE_EXTERNAL_STORAGE*). El permiso de escritura lleva impícito el permiso de lectura.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mmc.haciendofotos01" >

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

    …

</manifest>
```

CAPTURA DE FOTOS

Guardar Fotos a Tamaño Completo:

- Si no queremos que cada vez que hagamos una foto, esta sobrescriba la anterior, habrá que pensar un mecanismo para nombrar las nuevas fotos.
- Una buena idea es hacerlo en base a la fecha en que se tomó la foto, tal como muestra el siguiente código:

```
private File createImageFile() throws IOException {
    // Create an image file name
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
    String imageFileName = "JPEG_" + timeStamp + "_";
    File storageDir = Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES);

    File image = File.createTempFile(
            imageFileName,  /* prefix */
            ".jpg",         /* suffix */
            storageDir      /* directory */
    );

    // Save a file: path for use with ACTION_VIEW intents
    fotoPath = "file:" + image.getAbsolutePath();
    return image;
}
```

CAPTURA DE FOTOS

Guardar Fotos a Tamaño Completo:

- Una vez hecho esto, solo nos queda crear e invocar el *intent*:

```
private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Ensure that there's a camera activity to handle the intent
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        // Create the File where the photo should go
        File photoFile = null;
        try {
            photoFile = createImageFile();
        } catch (IOException ex) {
            // Error occurred while creating the File
            //...
        }
        // Continue only if the File was successfully created
        if (photoFile != null) {
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT,
                    Uri.fromFile(photoFile));
            startActivityForResult(takePictureIntent, CAPTURA_IMAGEN_TAMAÑO_REAL);
        }
    }
}
```

CAPTURA DE FOTOS

Guardar Fotos a Tamaño Completo:

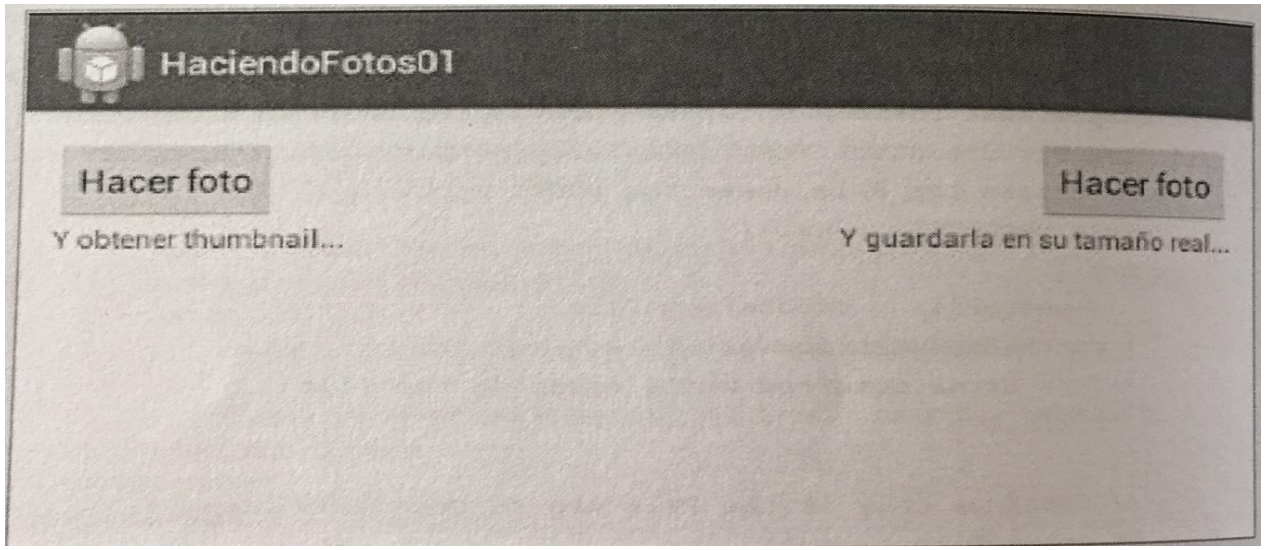
- En la línea de código:

```
takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT,Uri.fromFile(photoFile));
```

- La foto (en tamaño real) se añade como *dato extra* al *intent* con el método *putExtra()*.
- ***MediaStore*** es el proveedor de contenido de los archivos multimedia, y con su constante EXTRA_OUTPUT, se indica el *intent* que contiene la URI que se ussará para almacenar la foto.
- En este caso no se devuelve el *thumbnail*, y los datos del *intent* recibido serán **null**. Luego no se puede hacer *data.getExtras()* como en el ejemplo de "*Obtención de thumbnails*".
- La foto todavía no estará en la galería, deberás buscar la foto donde tu dispositivo almacene los archivos.

ACTIVIDAD 7

- Implementar una App con la siguiente interfaz:



- El botón de la izquierda hará una foto y devolverá un *thumbnail*, el cual será mostrado en un widget *ImageView* (que debe estar debajo de los botones).
- El botón de la derecha hará una foto y la guardará a tamaño completo en el sistema de archivos del dispositivo. Cuando lo haga deberá mostrarse un mensaje (*Toast*) indicando la ruta donde se ha guardado.

La solución para esta actividad la podréis encontrar en la carpeta compartida de drive con el nombre *"HaciendoFotos01"*.

TRATAMIENTO DE IMÁGENES ESCALADAS

- Cargar imágenes de gran tamaño en memoria dinámica es muy costoso, y la memoria en cualquier tipo de dispositivo informático es un recurso muy valioso y escaso.
- Las aplicaciones cada vez demandan más memoria dinámica, con lo que los dispositivos se van quedando "obsoletos" rápidamente si queremos utilizarlos sacando el mayor partido de las nuevas aplicaciones.
- Para visualizar las fotos en el móvil la resolución no es tan importante, con lo que si nuestra aplicación va a trabajar con imágenes, cada vez que las mostremos deberemos escalarlas a un tamaño razonable para no quedarnos sin memoria (excepción "out of memory").

TRATAMIENTO DE IMÁGENES ESCALADAS

- Vamos a ver la solución que aportan los desarrolladores de Android (http://developer.android.com/):

```
private void setPic(){
    //Cogemos las dimensiones de la Vista
    int targetW = mImageView.getWidth();
    int targetH = mImageView.getHeight();

    //Cogemos las dimensiones del bitmap
    BitmapFactory.Options bmOptions = new BitmapFactory.Options();
    bmOptions.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(mCurrentPhotopath, bmOptions);
    int photoW = bmOptions.outWidth;
    int photoH = bmOptions.outHeight;

    //Determinamos cuánto vamos a escalar la imagen
    int scaleFactor = Math.min(photoW/targetW, photoH/targetH);

    //Decodificamos el fichero de la imagen en un Bitmap del tamaño necesario para
llenar la vista
    bmOptions.inJustDecodeBounds = false;
    bmOptions.inSampleSize = scaleFactor;
    bmOptions.inPurgeable = true;

    Bitmap bitmap = BitmapFactory.decodeFile(mCurrentPhotopath, bmOptions);
    mImageView.setImageBitmap(bitmap);
}
```

TRATAMIENTO DE IMÁGENES ESCALADAS

- En la parte del código donde se obtienen las dimensiones de la imagen:
  - ***BitmapFactory.Options*** es una clase estática con un conjunto de campos o variables para trabajar con imágenes (*Bitmap*):

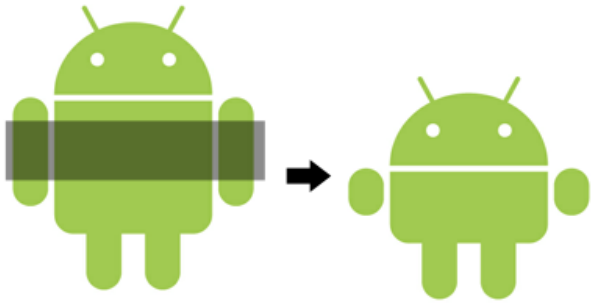
```
BitmapFactory.Options bmOptions = new BitmapFactory.Options();
```

  - La variable ***inJustDecodeBounds*** con valor ***true*** permite obtener los valores que determinan el tamaño original de la imagen:

```
bmOptions.inJustDecodeBounds = true;
```

  - Con el ***decodeFile()*** tan solo se obtiene en ***bmOptions*** información relativa a la imagen (dimensiones reales):

```
BitmapFactory.decodeFile(mCurrentPhotopath, bmOptions);
int photoW = bmOptions.outWidth;
int photoH = bmOptions.outHeight;
```



TRATAMIENTO DE IMÁGENES ESCALADAS

- Una vez que se han calculado las dimensiones que debe tener la imagen para visualizarse en el ***ImageView***, se escala la imagen con el nuevo tamaño:
  - En este caso se establece ***inJustDecodeBounds*** con valor ***false*** para devolver con ***decodeFile()*** la imagen escalada:

```
bmOptions.inJustDecodeBounds = false;
```

  - Factor de escalada calculado previamente:

```
bmOptions.inSampleSize = scaleFactor;
```

  - El campo ***inPurgeable*** se ha quedado obsoleto desde la API 21. Desde Lollipop en adelante es ignorado. Y para versiones anteriores (hasta Kitkat), si este campo es ***true***, el sistema se puede deshacer de la memoria usada para alojar el ***bitmap*** en caso de que la necesite:

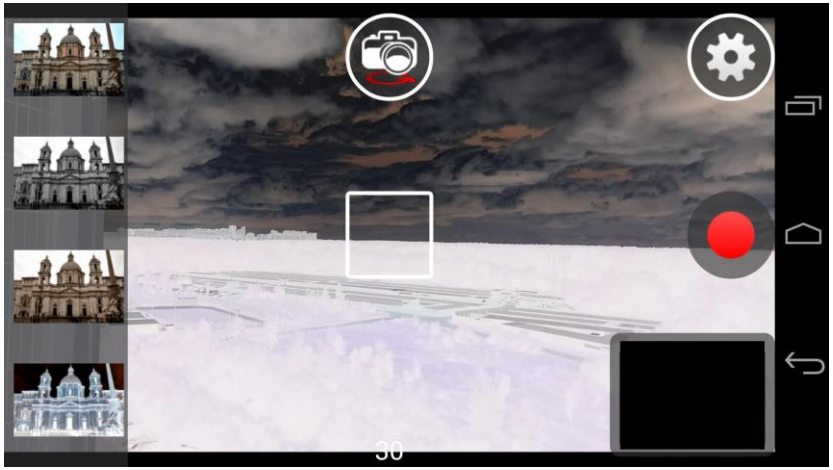
```
bmOptions.inPurgeable = true;
```

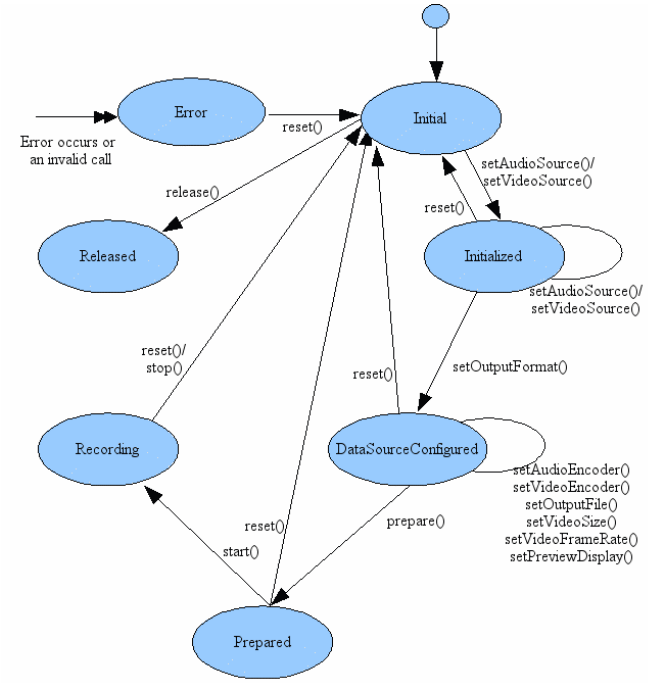
  - Finalmente se invoca a ***decodeFile()***, que devuelve un ***bitmap*** con la imagen escalada. Ya solo queda visualizarla donde corresponde, en este caso en el widget ***ImageView***:

```
Bitmap bitmap = BitmapFactory.decodeFile(mCurrentPhotopath, bmOptions);
mImageView.setImageBitmap(bitmap);
```

CAPTURA DE VIDEO

- Al igual que ocurre con la captura de fotos, Android ofrece dos posibilidades para grabar video:
  - Usar ***intents*** para delegar la captura a la aplicación de grabación por defecto.
  - Usar la clase ***MediaRecorder*** si lo que se pretende es reemplazar la aplicación nativa de Android para grabación de vídeo.
- La primera es la más sencilla y la más apropiada en la mayoría de ocasiones, y es la que vamos a ver a continuación.





MediaRecorder state diagram

CAPTURA DE VIDEO

- Lo primero que hay que hacer es crear el ***intent*** con la acción ***MediaStore.ACTION_VIDEO_CAPTURE***.
- A continuación lanzamos el intent, lo que iniciará la aplicación de grabación del sistema.

```
private final static int GRABAR_VIDEO = 1;

// Código asociado al botón COMENZAR_GRABACIÓN
public void comenzarGrabacion(View view){
    // Creación del intent
    Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
    // El vídeo se grabará en calidad baja (0)
    intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 0);
    // Limitamos la duración de la grabación a 5 segundos
    intent.putExtra(MediaStore.EXTRA_DURATION_LIMIT, 5);
    // Nos aseguramos de que haya una aplicación que pueda manejar el intent
    if (intent.resolveActivity(getPackageManager()) != null) {
        // Lanzamos el intent
        startActivityForResult(intent, GRABAR_VIDEO);
    }
}
```

CAPTURA DE VIDEO

- En el ejemplo se añaden unos datos extras y, opcionales, para personalizar la grabación:
  - ***MediaStore.EXTRA_VIDEO_QUALITY*** permite establecer la calidad de la grabación del vídeo. Hay dos posibles valores: 0 para baja calidad, y 1 para alta calidad. Por defecto, los vídeos se graban en alta calidad.

```
intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 0);
```

  - ***MediaStore.EXTRA_DURATION_LIMIT*** establece, en segundos, la duración máxima de la grabación.

```
intent.putExtra(MediaStore.EXTRA_DURATION_LIMIT, 5);
```

  - ***MediaStore.EXTRA_OUTPUT*** para elegir dónde se guardará el vídeo. Por defecto, los vídeos se almacenan en la galería.
- Una vez que el usuario haya terminado su grabación, el ***intent*** devuelto contendrá. como dato extra. una ***Uri*** al vídeo grabado.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == GRABAR_VIDEO && resultCode == RESULT_OK) {
        VideoView = (VideoView) findViewById(R.id.videoView);
        videoView.setVidvideoVieweoURI(data.getData());
        videoView.start();
    }
}
```

ACTIVIDAD 9

- Desarrollar una app que permita grabar vídeo. El funcionamiento será el siguiente:
  - Un botón para comenzar la grabación.
  - Un widget *VideoView* donde se visualizará el vídeo.

La solución para esta actividad la podréis encontrar en la carpeta compartida de drive con el nombre *"GrabandoVideo"*.

CAPTURA DE VIDEO

- Puede que haya móviles que no tengan la cámara disponible. Para evitar este tipo de situaciones, las cuales dejarían nuestra App en mal lugar, conviene anunciar en Google Play que nuestra aplicación depende de la existencia de la cámara en el dispositivo.
- Para ello en el *AndroidManifest.xml* escribimos:

```
<uses-feature
    android:name="android.hardware.camera"
    android:required="true" />
```

- Si nuestra App usa la cámara, pero no es estrictamente necesarioa para su funcionamiento, se puede establecer ***android:required*** a ***false***. De esta manera, Google Play permitirá que dispositivos sin cámara descarguen nuestra App.
- Es nuestra responsabilidad comprobar la disponibilidad de la cámara en tiempo de ejecución llamando al método ***hasSystemFeature(PackageManager.FEATURE_CAMERA)***. Y en caso de que no lo esté, habrá que deshabilitar las partes de nuestro código que dependan de la cámara.
- Todo esto también es aplicable a Apps que usen la cámara de fotos.

ALMACENAMIENTO DE CONTENIDO MULTIMEDIA (EN LA GALERÍA

- La mayoría de usuarios si una foto o vídeo no está en la galería de Android, no saben cómo encontrarlo.
- Los vídeos y thumbnails se almacenan en la galería directamente, mientras que las fotos a tamaño completo no.
- El siguiente código (extraído de http://developer.android.com/) muestra como añadir una foto a la galería. Sin embargo puede utilizarse para cualquier tipo de fichero.

```
private void galleryAddPic(){
    Intent mediaScanIntent = new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
    File f = new File(mCurrentPhotoPath);
    Uri contentUri = Uri.fromFile(f);
    mediaScanIntent.setData(contentUri);
    this.sendBroadcast(mediaScanIntent);
}
```

- Para conseguirlo se usa un ***intent*** con una acción de la clase ***MediaScanner*** (**ACTION_MEDIA_SCANNER_SCAN_FILE**).