APUNTES PROGRAMACIÓN EN ANDROID

José Juan Urrutia Milán

Reseñas

- Curso programación en android desde 0: https://www.youtube.com/playlist?list=PLyvsggKtwbLX06iMtXnRGX5lyjiiMaT2y
- Instalar IDE Android Studio: https://www.youtube.com/watch?v=sILYPMvXDvY&list=PLyvsggKtwbLX06iMtXnRGX5lyjiiMaT2y&index=2
- Documentación de Android Studio: https://developer.android.com/

Siglas/Vocabulario

- **IDE:** Entorno de Desarrollo Integrado.
- **JDK:** Java Development Kit.
- **BBDD:** Bases de Datos.
- **SGBD:** Sistemas Gestores de Bases de Datos.
- **ActionBar:** Barra superior donde se indica el nombre de la app.
- **APK:** Archivo que nos permite instalar programas Android sin necesidad de recurrir a la App Store.

Leyenda

Cualquier abreviatura o referencia será subrayada.
Cualquier ejemplo será escrito en **negrita.**
Cualquier palabra de la que se pueda prescindir irá escrita en cursiva.
Cualquier abreviatura viene explicada a continuación:

Abreviaturas/Referencias:

- 123: Hace referencia a cualquier número.
- nombre: Hace referencia a cualquier palabra/cadena de caracteres.
- a: Hace referencia a cualquier caracter.
- cosa: Hace referencia a cualquier número/palabra/cadena.
- Tipo_var o **…** : Hace referencia a cualquier tipo de variable primitiva o de tipo String.
- nombre_var: Hace referencia a cualquier nombre que se le puede dar a una variable.
- código: Hace referencia a cualquier instrucción. (Se usará para indicar dónde se podrá inscribir código.)
- variable: Hace referencia a cualquier variable.
- condición: Hace referencia a cualquier condición. Entiéndase por condición, una afirmación que devuelve un true o un false. Ej: (variable == 123). *Una variable del tipo boolean puede ser usada como una condición.

Índice

isCheked()

Título V: RadioButton

Título VI: CheckBox

isCheked()

Título VII: Spinner

setAdapter(ArrayAdapter a)

getSelectedItem()

Modificar Spinner (personalizarlo)

Título VIII: ListView

getItemAtPosition(int i)

Eventos ListView

Modificar ListView (personalizarlo)

Título IX: ImageButton y ImageView

Cambiar imagen a ImageView desde parte lógica

Añadir imágenes personalizadas a ImageButton

Quitar fondo de color a ImageButton

Título X: WebView

Uso de WebView

Dar permisos a nuestra app

setWebViewClient(WebViewClient client)

setWebChromeClient(WebChromeClient client)

loadUrl(String url)

Título XI: ScrollView

Título XII: ActionBar - Menú OverFlow

Añadir eventos a menús

Título XIII: ActionBar - ActionButtons

Añadir eventos a botones

Capítulo X: Generar APK con Android Studio

Introducción

Durante este curso, usaremos el IDE de Google Android Studio, en el cual podremos programar en C++, Kotlin y Java aunque nos centraremos más en este último. Necesario tener conocimientos previos de Java.

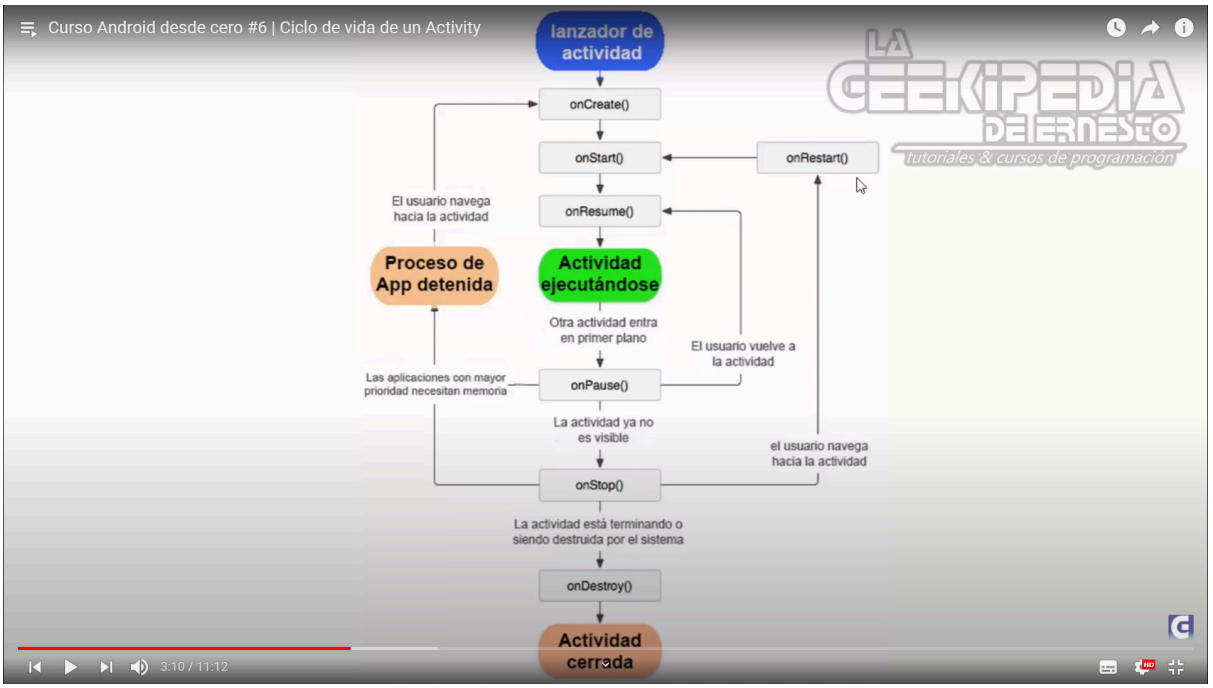
Capítulo I: Conceptos básicos

Título I: Activity

Una activity es lo más básico y más utilizado al desarrollar aplicaciones Android. Podemos decir que cada pantalla o lámina de nuestra app es un activity. Cada activity tiene dos partes: la gráfica y la lógica.
La parte lógica es un archivo .java
La parte gráfica es un archivo XML , parecido al HTML.

Recomendación: ver los primeros vídeos para aprender a utilizar Android Studio.

Un activity tiene un ciclo de vida que se ve influenciado por las acciones del usuario al minimizar o cerrar nuestra app.



Cada de estos ciclos de vida puede ser reconocido como un evento de ventana que ejecutarán el código que se encuentre en el interior de las funciones especificadas: (cada vez que usemos una función de estas,

deberemos indicar que llame primero a la función de la superclase con super.onStart() , por ejemplo) (dentro de nuestro activity):

protected void onCreate(Bundle SavedInstanceState)

Se ejecuta al abrir el activity.
Todos las apps deben llevar este método para que funcionen.

protected void onStart()

Se ejecuta antes de hacer visible el activity.
Se ejecuta cuando el activity vuelve a 1er plano.

protected void onResume()

Se ejecuta al hacer visible el activity (volver a hacerlo visible).
Se ejecuta cuando el activity vuelve a 1er plano.

protected void onPause()

Se ejecuta antes de pausar el activity.

protected void onStop()

Se ejecuta cuando el activity ya no es visible, cuando este se oculta.

protected void onRestart()

Se ejecuta al volver al activity.

protected void onDestroy()

Se ejecuta al cerrar la app.

Con estos métodos, podemos hacer que una aplicación se comporte de forma diferente en 1er y en 2o plano.

Título II: Toast

Librería **android.widget.Toast**

Un Toast es una notificación emergente con la que podemos mostrar información a los usuarios.
Este mensaje aparece en el activity que se está utilizando pero no bloquea sus funciones (el programa no espera a que el Toast finalice).
Los Toast no aceptan interacciones, por lo que al tocarlo no pasa nada.
Los Toast pueden mostrar texto, imágenes o imágenes y texto.

Estructura

Toast.makeText(Context context, String txt, Int duracion).show();
- context es el lugar, pantalla o activity donde queremos mostrar el Toast (**this** sirve).
- txt es el texto queremos mostrar
- duracion es el tiempo durante el que aparecerá el Toast (puede ser **Toast.LENGTH_LONG** o **Toast.LENGTH_SHORT** de 1 y 0.5 segundos respectivamente).

Título III: Dar permisos a nuestra app

Para solicitar al usuario que le de permisos a nuestra app, debemos abrir el archivo que se encuentra en (Project) app>manifests>AndroidManifest.xml
Una vez abierto, debemos crear una nueva etiqueta antes de la etiqueta **<application>** , justo después del "manifest". Esta etiqueta será:
<uses-permission android:name="android.permission.nombre" />
Sustituir **nombre** por el permiso que queramos dar.
Se pueden concadenar:
<uses-permission android:name="android.permission.nombre"
android:name="android.permission.nombre" />
...

Capítulo II: Componentes (gráficos) periféricos

Título I: IDs

Una vez se añade un componente, lo ideal es añadirle un ID, aunque Android Studio se lo añade automáticamente.
Las IDs no se pueden repetir.
Podemos cambiar cualquier propiedad en el panel de atributos de la derecha.

Título II: Vista Blueprint

A la hora de crear componentes gráficos, es importante usar la vista blueprint para marcar con los puntos que distancia queremos que respete cada componente con el marco. Debemos hacer esto para evitar que se solapen los elementos gráficos.

Título III: Atributos

hint

En el atributo **hint** podemos indicar lo que queramos que se vea en los campos de texto cuando no se haya introducido nada.

textSize

En el atributo **textSize** (dar a mostrar todos los atributos) podemos seleccionar de qué tamaño queremos que sea el texto. Este tamaño se mide en sp.

text

El atributo **text** para mostrar texto en componentes (como un botón) (ver **Hardcoded string should use string resource** del siguiente capítulo).

onClick

En el atributo **onClick** podremos seleccionar una función de nuestro activity para que funcione como receptor del evento que desencadena nuestro botón. Para poder elegir nuestra función como receptora de este evento, esta debe recibir como parámetro un objeto **View**

View es de la **Librería android.view.View;**

```
public void Funcion(View view){}
```

Identificar botón pulsado

Podemos poner en el atributo **onClick** la misma función oyente para múltiples valores y después diferenciarlos dentro de la función con el método **getId()** (devuelve int) sobre nuestro objeto **View**, para luego compararlo con los campos correspondientes de ids: **R.id.nombre** .

textStyle

Aquí podemos seleccionar el estilo de texto que se muestra en el componente (negrita, cursiva...).

gravity

Establece la parte del eje y en el que se podrá escribir en un View (como en un MultilineText) si colocamos **top**, se escribirá de arriba a abajo como un **JTextArea** de Java.

textColor

Establece el color que tendrá el texto del componente (código hexadecimal (#0000ff)).

textColorHint

Establece el color que tendrá el hint del componente (código hexadecimal (#0000ff)).

background

Establece el color que tendrá el fondo del componente (código hexadecimal (#0000ff)).
También puede establecer que el fondo será una imagen (pinchando en los 3 puntos y teniendo la imagen en la carpeta (Project) app>res>drawable). Esta imagen puede no ser 100% opaca, dejando ver el fondo de nuestro activity.

Padding

Indicar valores en **left** y **right** para centrar el texto/hint.

Título IV: Relacionar con parte lógica

Si necesitamos usar varios componentes de la parte gráfica, debemos indicarle a la parte lógica esto de la siguiente manera:
Indicamos que va a ser un objeto privado, indicamos la clase a la que pertenecerán (depende de cada componente gráfico) e indicamos su nombre.
Todos estas clases pertenecen a la:

Librería **android.widget.***

Una vez declarados todos los objetos gráficos que vamos a usar, dentro del método **onCreate()** , los inicializamos mediante un casting a la **Clase en Java** , seguido del método **findViewById(int id)** , donde especificaremos el id del componente gráfico mediante el campo de clase de la clase **R.id** : (primera instrucción en los campos de clase, segunda instrucción dentro de **onCreate()**), (nuestro componente gráfico es un **Number** con id: etiqueta1):

```
private EditText etiqueta;
etiqueta = (EditText)findViewById(R.id.etiqueta1);
```

Obtener texto de componente

Para ello, haremos lo mismo que haríamos en java (etiqueta corresponde a un **EditText** que hace referencia a un **Number**):

String texto = etiqueta.getText().toString();
Para obtener un número:
int numero = Integer.parseInt(etiqueta.getText().toString());

Imprimir texto en componente

(Donde Label es un **TextView** que representa un objeto **TextView**)
Label.setText("Hola mundo");

Hardcoded string should use string resource

Es una advertencia que nos indica que no debemos escribir en la propiedad **text** de un componente el texto que este mostrará sino debemos crear un String dentro del documento:
(Project) app>res>values>strings.xml
Ahí dentro, desarrollaremos todos los String a usar, indicando un nombre (txt) y un valor (Hola):
<string name="txt">Hola</string>
Posteriormente, en el atributo **text** del componente que queramos que muestre el texto, pulsaremos Ctrl+espacio o escribiremos @string/ + el nombre que le hayamos dado al String, en este caso:
@string/txt .

Hacer invisible/visible un componente

Desde Java, ejecutaremos el método **setVisibility()** sobre el objeto que queremos hacer invisible/visible y le pasaremos como parámetro **View.INVISIBLE** o **View.VISIBLE**.

Título V: Establecer icono a app

El que se verá en el escritorio y al minimizar la app.
*Importante que el nombre de la imagen esté en minúsculas y no use caracteres especiales.

Añadir imagen al proyecto y añadirla

Click dcho sobre app (Project) > New > Image Asset
Icon Type: Launcher Icons (Adaptive and Legacy) recomendado.

Foreground Layer:
Recomendable no cambiar el Name.
En Path indicamos la ruta de la imagen.

Background Layer:
Poder cambiar el color de fondo o seleccionar imagen y disminuir fondo para transparencia.

Añadir imagen al lado del nombre de la app (actionBar)
Nos dirigimos al archivo Java y dentro del método **onCreate** escribimos:
getSupportActionBar().setDisplayShowHomeEnabled(true);
getSupportActionBar().setIcon(R.mipmap.ic_Launcher);

Título VI: Establecer color app

Ir a la web https://www.materialpalette.com
Seleccionar 2 colores.
Download>XML y guardar archivo.
Renombrar archivo con: **colors_nombreC1_nombreC2** .
Abrir archivo con bloc de notas (y acomodarlo).
Copiar todas las etiquetas <color.name>
Ir a (Project) app>res>values>colors.xml
Pegar códigos.
Sustituir los valores default (primario, primario oscuro y accent) por los nuevos (dejar solo 3 etiquetas).
Copiar y pegar los **name** de las default en las nuevas.
Borrar las default.

Título VII: Cambiar nombre a la app

Dirigirse al archivo strings.xml (título IV, **Hardcoded string should use string resource**) y cambiar la primera etiqueta.

Título VIII: Cambiar fondo de activity

Para ello, movemos la imagen que queramos usar como fondo a la carpeta (Project) app>res>drawable
*Esta foto debe estar en minúsculas y no puede tener caracteres especiales.
Para cambiar el fondo de nuestro activity, pinchamos en él, nos vamos al apartado donde salen todos los atributos y en el atributo **background** escribimos la referencia a la imagen. (**@drawable/nombre**). (Sustituir **nombre** por el nombre de la imagen.)

Título IX: Aplicaciones multilenguaje

Este es un método que permite distribuir apps por distintos países con diferentes idiomas.
Estos cambios se aplican una vez que el usuario cambia el idioma de su dispositivo en sus ajustes.
Estos cambios son posibles si todo el texto de nuestra aplicación son referencias al archivo **strings.xml** .

Para añadir diferentes idiomas, debemos irnos a la carpeta (Project) app>res>values y crear un nuevo **Values resource file** (es como una copia del archivo strings.xml). Este nuevo archivo se llamará strings también.
En el apartado de **qualifiers**, debemos pasar el **qualifier Locale**, seleccionando el idioma deseado y las regiones deseadas (si es para cualquier ciudadano de habla española, seleccionar español y **Any Region,** de esta forma saldrá el mismo formato para España y para países hispanohablantes como Argentina...)

Una vez tengamos este archivo, copiamos y pegamos el archivo strings al nuevo strings, cambiando el texto de un idioma a otro y manteniendo los **name** .

Título X: Aplicaciones con diseño adaptable

Para crear una aplicación cuyo diseño se adapte a la pantalla del dispositivo, deberemos irnos a la parte de texto de nuestro archivo xml (la parte de código, no gráfica).
Aquí, cambiaremos la etiqueta **android.support.constraint.ConstraintLayout** por una etiqueta **RelativeLayout** (con el mismo contenido que la anterior).
Una vez realizado este pequeño cambio, nuestro diseño se adapta a cualquier pantalla.

Título XI: Aplicaciones que no rotan con el dispositivo

Para crear una aplicación que se mantenga siempre en vista vertical u horizontal, debemos dirigirnos a su archivo (Project) app>manifests>AndroidManifest.xml y dentro de la primera etiqueta **activity** y después del **atributo name**, debemos crear otro atributo llamado **screenOrientation** al que le asignaremos el valor **“portrait”** (para que siempre esté en vertical) o **“landscape”** (para que esté siempre en horizontal):

…
<activity android:name=”.nombre”
android:screenOrientation=”portrait”> *// o “landscape”*
…

Título XII: Evitar que se reinicie el activity al girar el dispositivo

Si giramos nuestro dispositivo con un activity en curso, este se reinicia desde 0 ya que tiene que adaptarse a la nueva resolución. Para evitar esto, debemos hacer lo siguiente:

Debemos dirigirnos al archivo (Project)
app>manifests>AndroidManifest.xml
Dentro del código, buscamos la etiqueta **activity** que contenga al activity que no queremos que se reinicie y después del apartado **name** y antes de su cierre, colocamos lo siguiente:

android:configChanges=”orientation|screenSize”

quedando así la etiqueta completa:

<activity android:name=”nombre”
android:configChanges=”orientation|screenSize” />

Capítulo III: Componentes gráficos / View / Control

(A = Apartado), (CJ = Clase en Java)

La mayoría de los Views que se verán a continuación comparten métodos similares entre ellos como: **getText(), **setText(String s)...**

Título I: EditText: PlainText / Password / E-mail / MultilineText / Number

A: Text
CJ: EditText

PlainText: similar al JTextField de Java.
Password: similar al JPasswordField de Java.
E-mail: no visto en el curso.
MultilineText: similar al JTextArea de Java.
Number: similar al JTextField pero sólo acepta números.

PlainText y **Password** funcionan igual, la principal diferencia es que **Password** pone asteriscos en lugar de las palabras a diferencia del **EditText**.

*Los **PlainText** se crean con el atributo **text** relleno por defecto.

getText()

Devuelve un objeto referente al contenido del **EditText**.
Recomendable poner después .toString().

Limitar cantidad de caracteres máximo

Para ello, nos dirigiremos al **atributo maxLength** y le fijamos el valor que queramos (en caracteres).

Abrir teclado de forma automática

Si queremos que el teclado se abra de forma automática sobre nuestro Plain Text (o similar) una vez que ocurre una acción (como que pulsemos sobre un botón y este Plain Text esté vacío), debemos ejecutar las siguientes líneas de código en la parte lógica de nuestra app: (donde input es un objeto **EditText**:)

```
input.requestFocus();
InputMethodManager imm = (InputMethodManager)
getSystemService(this.INPUT_METHOD_SERVICE);
imm.showSoftInput(input,
InputMethodManager.SHOW_IMPLICIT);
```

Título II: Button

A: Widgets

CJ: Button

Un botón. (Mirar capítulo II, título III, onClick).

Cambiar diseño

Podemos cambiar el diseño de nuestro botón moviendo la imagen que queramos que aparezca por el botón a la carpeta (Project) app>res>drawable. *Esta imagen debe tener el nombre en minúsculas y sin caracteres especiales.

Posteriormente, nos dirigimos a nuestro botón y le damos un tamaño según las proporciones de la imagen (si esta es cuadrada, un botón cuadrado.).

A continuación, nos dirigimos al atributo **background** y le damos un valor que haga referencia a la imagen de la carpeta drawable: **@drawable/nombre** (sustituir **nombre** por el nombre de la imagen.)

Esto también se puede hacer desde la parte lógica:

(el objeto **boton** hace referencia a un **Button** de la parte gráfica):
boton.setBackgroundResource(R.drawable.nombre);

Título III: TextView

A: Text
CJ: TextView
Una etiqueta de texto (como un JLabel de Java).

setText(String s)

Establece el texto del **TextView**

Título IV: RadioGroup

A: Widgets
Un Radio Group es un container con forma cuadrada donde podremos agregar **RadioButton**s. Al estar metidos todos los **RadioButton** en el mismo **RadioGroup**, sólo podrá haber un botón activado al mismo tiempo.

(Atributo) layout_height

Seleccionar **wrap_content** para ajustar el tamaño del **RadioGroup** a la cantidad de **RadioButton** que este contenga.

isCheked()

Devuelve true si está seleccionado, false si no.

Título V: RadioButton

A: Widgets
CJ: RadioButton
Botones de radio (como los JRadioButton). Si estos se encuentran en un mismo **RadioGroup**, sólo podrá haber uno seleccionado a la vez.

Título VI: CheckBox

A: Widgets
CJ: CheckBox
Botones de check (como los JCheckBox).

isCheked()
Devuelve true si está seleccionado, false si no.

Título VII: Spinner

A: Widgets
CJ: Spinner
Selector de objetos (como los JComboBox).

Para añadir las opciones a nuestro **Spinner**, lo debemos hacer desde la parte lógica.
Dentro del método **onCreate**, establecemos el puente de comunicación con la parte gráfica, creamos un array de tipo String con los objetos que se mostrarán en el **Spinner** , creamos un objeto de la clase genérica **ArrayAdapter** (a la que le pasaremos como clase el tipo de dato que vayamos a meter en el **Spinner** , en este caso String), pasándole al método constructor tres datos: el activity del **Spinner**, el tipo de separación que tendrá nuestro **Spinner** (**android.R.layout.simple_spinner_item o android.R.layout.simple_spinner_dropdown_item** (este último deja un poco de espacio entre los elementos)) y el array que queremos que se muestre en el **Spinner**.
Posteriormente, usaremos el método **setAdapter(ArrayAdapter a)** sobre nuestro objeto **Spinner**. (nuestro **spinner** se llama selector):
String[] elementos = {“elemento1”, “elemnto2”, “elemento3”};
ArrayAdapter <String> adaptador = new ArrayAdapter <String> (this, android.R.layout.simple_spinner_item, elementos);
selector.setAdapter(adaptador);

setAdapter(ArrayAdapter a)

Establece un **ArrayAdapter** en el que poder indicar los elementos del **Spinner**.

getSelectedItem()

Devuelve el objeto seleccionado. Recomendable usar .toString() después para recibir un String.

Modificar Spinner (personalizarlo)

Para modificar nuestro **Spinner**, debemos crear un nuevo archivo .xml en la ruta: (Project) app>res>layout recomendable darle el nombre de: spinner_item_**nombre** .
En el código XML, borraremos todo el código menos la primera línea. A continuación, abriremos una nueva etiqueta, escribiremos "TextView", pulsamos Ctrl + espacio y seleccionamos layout_width . Una vez generado todo, copiamos la línea de layout_width y la pegamos cambiando width por height. El resultado será similar al siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:android="https://schemas.android.com/apk/res/android" />
```

A continuación, añadiremos todos los atributos que queramos antes de la última línea respetando el formato:

android:nombre_atributo="valor_atributo"

por ejemplo:

android:background="#0000ff" *color del fondo*

android:textSize=”24sp”	*tamaño del texto*
android:padding=”10sp”	*separación de elementos*
android:textColor=”#0000ff”	*color del texto*

Una vez hayamos seleccionado todos los atributos, nos vamos a la parte lógica de nuestro programa (.java) y en el constructor de nuestro objeto **ArrayAdapter**, cambiamos el tipo de spinner (como **android.R.layout.simple_spinner_item**) por nuestro spinner: **R.layout.spinner_item_nombre** .

Título VIII: ListView

A: Containers

CJ: ListView

Lista de elementos (como el Spinner pero en este se muestran varios a la vez).

Al igual que el Spinner, se modifica desde la parte lógica.
Al igual que el Spinner, sus atributos de la parte visual se modifican de forma similar.

Para añadir un **ListView**, al igual que con el Spinner, necesitaremos un array unidimensional de Strings en el que introduciremos los elementos de nuestro **ListView**.
Y al igual que con **Spinner**, creamos un objeto **ArrayAdapter** al que le pasamos por parámetros: un contexto (activity (this)), un tipo de **ListView** (como el defecto (**android.R.layout.simple_list_item**)) o uno que creemos (ver **Modificar ListView**) y el array de elementos que declaramos antes.
una vez creado el **ArrayAdapter**, lo ejecutaremos sobre nuestro **ListView** con el método **setAdapter(ArrayAdapter a)** .
(nuestro objeto **ListView** se llama lista y el array elementos):

ArrayAdapter <String> adaptador = new ArrayAdapter <String> (this, android.R.layout.simple_list_item, elementos);
lista.setAdapter(adaptador);

getItemAtPosition(int i)

Devuelve el objeto que se encuentra en la posición i.
(Podemos usar este método con el entero **i** de dentro del evento del **ListView** que se ve a continuación:)

Eventos ListView

Para ello, deberemos ejecutar el método **setOnItemClickListener()** sobre nuestro objeto **ListView**, como si de un evento Java se tratara. Dentro del método tendremos que escribir una clase anónima (ver apuntes de Java) pero Android Studio nos ofrece la siguiente por defecto:

```
lista.setOnItemClickListener(new
AdapterView.OnItemClickListener()){
    public void onItemClick(AdapterView <?> adapterView,
    View view, int i, long l){
        código;     que se ejecuta al pulsar sobre el ListView
    }
}
```

(El parámetro **int i** hace referencia al elemento pulsado (0, 1, 2...))

Modificar ListView (Personalizarlo)

Crearemos un documento dentro de (Project) app>res>layout y crearemos un nuevo documento .xml . Recomendable que tenga esta estructura de nombre: **list_item_nombre** .
En el código XML, borraremos todo el código menos la primera línea. A continuación, abriremos una nueva etiqueta, escribiremos "TextView", pulsamos Ctrl + espacio y seleccionamos layout_width . Una vez generado todo, copiamos la línea de layout_width y la

pegamos cambiando width por height. El resultado será similar al siguiente:

<?xml version=”1.0” encoding=”utf-8”?>
<TextView
android:layout_width=”match_parent”
android:layout_height=”wrap_content”
xmlns:android=”https://schemas.android.com/apk/res/android” />

A continuación, añadiremos todos los atributos que queramos antes de la última línea respetando el formato:

android:nombre_atributo=”valor_atributo”

por ejemplo:

android:background=”#0000ff”	*color del fondo*
android:textsize=”10sp”	*tamaño de texto*
android:padding=”10sp”	*separación entre elementos*
android:textColor=”#0000ff”	*color del texto*

Título IX: ImageButton y ImageView

A: Images, Windgets
CJ: ImageView para ambos
ImageView: Una vista que nos permite poner imágenes. Funciona de forma similar a **ImageButton**, salvo que no es un botón.
ImageButton: Un botón que es capaz de mostrar una imagen (o una imagen interactiva). (Recomendable que las imágenes sean de 50x50 px).
*El nombre de las imágenes debe estar en minúsculas y no pueden tener caracteres especiales.
Android Studio pone a la disposición del usuario multitud de imágenes al colocar nuestro **ImageButton**.

Cambiar imagen a ImageView desde parte lógica

(nuestro objeto **ImageView** se llama **imagen**):
imagen.setImageResource(R.drawable.nombre);

o, si no sabemos el id podemos hacer lo siguiente:
(donde nombre es el nombre del archivo y drawable la carpeta en la que se encuentra este archivo).
int id = getResources().getIdentifier("nombre", "drawable", getPackageName());
imagen.setImageResource(id);

Añadir imágenes personalizadas a ImageButton

Si queremos añadir una imagen personalizada a nuestro **ImageButton**, deberemos dirigirnos a la carpeta (Project) app>res>mipmap o app>res>drawable (drawable recomendable). Copiaremos todas las imágenes que queramos usar en los **ImageButton** y las pegaremos sobre esta carpeta.
A continuación, añadiremos nuestro **ImageButton**, seleccionando nuestras imágenes.

Quitar fondo de color a ImageButton

Si aparece nuestra imagen .png en el botón y sigue saliendo la apariencia del botón, nos dirigiremos al **atributo background**, pincharemos en los tres puntos, en la sección **Color** y en la pestaña **android** seleccionaremos el color **transparent** .

Android Studio pide al programador que sea necesario rellenar el atributo **contentDescrip de nuestro **ImageButton** con un texto que haga una pequeña descripción del botón. Para ello, crearemos el texto en nuestro documento **strings.xml** y pondremos la referencia en este atributo (ver Capítulo II, título IV, **Hardcoded string should use string resource**).

Esta descripción no se mostrará al usuario por lo que si estamos desarrollando un programa pequeño, podemos poner cualquier texto o la referencia al texto **@string/app_name** .

No es necesario crear una conexión directa entre los **ImageButton** y la parte lógica, ya que podemos crear eventos en funciones gracias al método **onClick** (ver Capítulo II, título III, **onClick**).

Título X: WebView

Librería android.webkit.WebView

A: Widgets

CJ: WebView (Recomendable dejarlo con el modificador de acceso default (no poner public o private))

Un Container que nos permite acceder a sitios web sin salir de nuestra aplicación.

Uso de WebView

Primero, debemos indicar si este va a abrir el sitio web dentro de la app o fuera con la ayuda de google chrome con el método **setWebViewClient(...)** o **setWebChromeClient(...)** y posteriormente indicar con el método **loadUrl(...)** la url a cargar.
(Métodos explicados a continuación).
Para que todo esto funcione, necesitamos que el usuario le dé permisos a la app para navegar por internet (ver **Dar permisos a nuestra app).

Dar permisos a nuestra app

Para solicitar al usuario que le de permisos a nuestra app, debemos abrir el archivo que se encuentra en (Project) app>manifests>AndroidManifest.xml

Una vez abierto, debemos crear una nueva etiqueta antes de la etiqueta **<application>** , justo después del "manifest". Esta etiqueta será:
<uses-permission
android:name="android.permission.INTERNET" />
Una vez escrito esto, ya estará todo resuelto.

setWebViewClient(WebViewClient client)

Para acceder a un sitio web sin salir de nuestra app.
(como client podemos pasar un objeto **WebViewClient** con constructor default; es decir: **new WebViewClient()**)

setWebChromeClient(WebChromeClient client)

Para acceder a un sitio abriendo Chrome, saliendo de nuestra app.
(como client podemos pasar un objeto **WebChromeClient** con constructor default; es decir: **new WebChromeClient ()**)

loadUrl(String url)

Carga la url introducida por el usuario (introducir antes método **setWeb...Client()**).

Título XI: ScrollView

A: Containers

Similar al **JScrollPane** de Java. Permite desplazarnos por la lámina de la app.
Los **ScrollView** cuentan con **LinearLayout** verticales que son los que nos permiten introducir elementos dentro del **ScrollView**.

Título XII: ActionBar - Menú OverFlow

Librería Menu: android.view.*;

Este es un menú desplegable que puede mostrar opciones o acciones. Aparece reflejado como tres puntos en la parte izquierda del ActionBar.
En este caso, crearemos un nuevo documento .xml para ponerlo sobre el documento que Android Studio genera de manera automática.

1. Por lo que iremos a la carpeta (Project) app>res y crearemos un nuevo **Android Resource File** . Lo nombramos como queramos *este nombre debe ir en minúsculas y sin caracteres especiales . En **Resource Type** indicamos **Menu** .

2. Añadimos tantos **Menu Item** a nuestro nuevo archivo.xml como opciones queramos añadir.

3. La acción de mostrar y ocultar el menú no se hace de forma automática, por lo que lo tendremos que programar:
   3.1. Crearemos una nueva función pública que devuelve un **boolean** y la nombraremos **onCreateOptionsMenu(Menu menu)** .
   3.2. Dentro de este método, agregaremos las siguientes líneas de código quedándonos de esta manera:

```
public boolean onCreateOptionsMenu(Menu menu){
        getMenuInflater().inflate(R.menu.nombre, menu);
        return true
}
```

Sustituir **nombre** por el nombre que le dimos al archivo en el punto 1.

Añadir eventos a menús

1. Para ello, debemos crear un método público que devuelve un **boolean** llamado **onOptionsItemSelected(MenuItem m)** .
*Si declaramos el método **getItemId()** sobre nuestro objeto **MenuItem m** , obtendremos un int con su id (**R.id.nombre**);

2. Para finalizar, debemos indicarle a nuestro método que devolverá lo que devuelve la clase padre:
return super.onOptionsItemSelected(m);

Título XIII: ActionBar - Action Buttons

Para crear un ActionButton, necesitamos una imagen .png de 24 x 24 px con nombre en minúsculas sin caracteres especiales que almacenaremos en la carpeta (Projects) app>res>drawable .

1. Al igual que con los Menús de OverFlow, iremos a la carpeta (Project) app>res y crearemos un nuevo **Android Resource File** . Lo nombramos como queramos *este nombre debe ir en minúsculas y sin caracteres especiales . En **Resource Type** indicamos **Menu** .

2. Agregaremos un **Menu Item** , agregamos en el **atributo icon** la referencia a la imagen (**@drawable/nombre**) y en **showAsAction** pondremos **always**.
*Si mezclamos Menús de OverFlow con Action Buttons, deberemos poner al menú en el **atributo showAsAction** : **ifRoom .**

3. Es necesario agregar un método para que nuestros Action Buttons aparezcan.
3.1. Crearemos una nueva función pública que devuelve un **boolean** y la nombraremos **onCreateOptionsMenu(Menu menu)** .
3.2. Dentro de este método, agregaremos las siguientes líneas de código quedándonos de esta manera:

```
public boolean onCreateOptionsMenu(Menu menu){
        getMenuInflater().inflate(R.menu.nombre, menu);
        return true
}
```

Sustituir **nombre** por el nombre que le dimos al archivo en el punto 1.

Añadir eventos a botones

1. Para ello, debemos crear un método público que devuelve un **boolean** llamado **onOptionsItemSelected(MenuItem m)** .
*Si declaramos el método **getItemId()** sobre nuestro objeto **MenuItem m** , obtendremos un int con su id (**R.id.nombre**);

2. Para finalizar, debemos indicarle a nuestro método que devolverá lo que devuelve la clase padre:
return super.onOptionsItemSelected(m);

Capítulo IV: Comunicación entre activitys

Para ello, necesitaremos manejar objetos **Intent**, un objeto de acción que se usa para solicitar una acción de otro componente de nuestra aplicación. Este representa la "intención de hacer algo".

Librería: **android.content.Intent**

Título I: Creación de activity

Para ello, iremos a la ventana Project de Android Studio, pulsaremos click derecho sobre la carpeta app y pulsamos en New>Activity>El tipo de activity que queramos , en este caso, **Empty Activity**. Una vez hecho, se nos crearán dos nuevos documentos uno con la parte lógica de nuestro activity y otro con la parte gráfica.

Título II: Cambio de activity

Para ello, deberemos crear una instancia de la clase **Intent**, al cual le pasaremos como parámetros el activity donde nos encontramos (podemos usar **this**) y el activity al que nos queremos dirigir (importante hacer referencia al archivo .class). Posteriormente, ejecutaremos este cambio con el método **startActivity(Intent i)**:

```
Intent i = new Intent(this, Activity2.class);
startActivity(i);
```

*Si la activity de la que venimos no se va a ejecutar más en el transcurso de la app o apenas se va a usar, es interesante cerrarla del todo con un **finish()** :

```
startActivity(i);
finish();
```

finish()

*Disponemos del método **finish()** que lo que hace es cerrar el activity actual, volviendo al activity anterior que conserva los datos que el usuario introdujo (como texto en un **EditText**).
finish(); *No hace falta usarlo sobre ningún objeto*.
Si nos encontramos en el último activity, se cerrará la app.

Título III: Paso de parámetros: Escritura

Para pasar datos de un activity a otra, debemos indicar sobre nuestro objeto **Intent** con el método **putExtra(String key, … value)** (... hace referencia a que acepta multitud de valores (int, float, String...)) dos parámetros siguiendo el formato **key-value**, indicando el nombre del key en un String y el value en un dato de los múltiples que acepta la sobrecarga de constructores del método **putExtra()**.
(Si queremos pasar un String de una activity a otra por un **Intent** i):
i.putExtra("Nombre", "Juan");

De esta forma, pasamos a nuestro siguiente activity el parámetro "Nombre" que almacena el valor: "Juan".

Una vez enviados todos los parámetros, efectuaremos el cambio de activity con el método **startActivity(Intent i)** visto en el título anterior.

Título IV: Paso de parámetros: Lectura

Una vez en el activity que queramos que reciba el String enviado anteriormente, almacenaremos en una variable del mismo tipo del dato que se envió (en este caso un String) el método **getIntent()** seguido del método **get...Extra(String key)** (Sustituir … por el tipo de dato que se recibe (Int, Float, String...)) donde indicaremos como **key** el String que se empleó al escribir el dato enviado. En este caso:

String dato = getIntent().getStringExtra("Nombre");

De esta forma, el valor “Juan” se almacenará dentro de la variable dato.

Título V: Control de botones especiales

Podemos controlar los botones de Android (atrás, menú y aplicaciones en 2º plano) desde la parte lógica de Java para suprimir algunas funciones o cambiarlas. De esta forma, si queremos inutilizar el botón atrás para que, al ser pulsado, no nos envíe a un activity anterior, debemos sobreescribir el método público que no devuelve nada **onBackPressed()** e introducirle el método **finish()**:

```
@Override
public void onBackPressed(){
        finish();
}
```

*Esto se tendrá que hacer en cada activity en la que lo queramos modificar.

Capítulo V: Almacenamiento de datos

Título I: Clase SharedPreferences

La clase **SharedPreferences** se utiliza para guardar información limitada permanentemente en el dispositivo, como nivel actual en un juego, colores de fondo, configuraciones en los juegos…

Guardar datos

1. Creamos un objeto de tipo **SharedPreferences** que lo igualamos al método **getSharedPreferences** al que le pasamos el nombre que recibirá el archivo que almacena los datos y un modificador.
2. Creamos un objeto de la clase **SharedPreferences.Editor** que lo igualamos al objeto anterior **.edit()** .
3. Escribimos con la ayuda del objeto editor y con el método **putString(String key, String value)** indicando la preference que se guardará en formato **key-value** .
4. Confirmamos que queremos guardar la preferencia con el método **commit()** sobre el editor:

```
SharedPreferences preferencias = getSharedPreferences("datos", Context.MODE_PRIVATE);
SharedPreferences.Editor editor = preferencias.edit();
editor.putString("nombre", "Juan");
editor.commit();
```

Leer datos

1. Para ello, crearemos un objeto perteneciente a **SharedPreferences** que hará referencia al archivo donde las referencias se guardan.
2. A continuación, almacenaremos en la variable que elijamos el objeto que había en la **key** indicada gracias al método

getString(String key, String s) sobre nuestro objeto **SharedPreferences**:

```
SharedPreferences preferencias = getSharedPreferences("datos", Context.MODE_PRIVATE);
String nombre = preferencias.getString("nombre", "");
```

Título II: Almacenamiento en ficheros

Los ficheros que crea un programa sólo pueden ser accesibles por ese programa. No pueden ser leídos por otras aplicaciones o por el usuario android.
Los ficheros que se crean en una aplicación son eliminados una vez que la aplicación es eliminada.

Escribir en fichero

Para ello, necesitamos un Stream:
(Dentro de un try catch:)
1. Creamos un objeto **OutputStreamWriter** y le pasamos al constructor el método **openFileOutput(String archivo, int mode)** donde especificamos el archivo y la constante **Activity.MODE_PRIVATE** .
2. Posteriormente, escribimos los datos deseados con el método **write(String datos)** sobre nuestro objeto **OutputStreamWriter**.
Limpiamos el buffer con el método **flush()** sobre el mismo objeto.
Cerramos el stream con **close()**

```
OutputStreamWriter stream = new OutputStreamWriter(openFileOutput("archivo.txt", Activity.MODE_PRIVATE));
stream.write("texto");
stream.flush();
stream.close();
```

Leer fichero

Para ello, necesitaremos la ayuda de un Stream y de un Buffer:
(Dentro de un try catch:)
1. Creamos un objeto **InputStreamReader** al que le pasamos como parámetros al constructor el método **openFileInput(String archivo)** al que le pasamos como archivo el nombre del archivo al que queremos acceder. (consultar método **fileList()**).
2. A continuación, creamos un objeto **BufferedReader** al que le pasamos al constructor el objeto **InputStreamReader** anteriormente creado.

```
InputStreamReader stream = new
InputStreamReader(openFileInput("archivo.txt"));
BufferedReader buffer = new BufferedReader(stream);
```

3. Podemos usar el método **readLine()** sobre nuestro objeto **BufferedReader** para obtener la lectura de la primera línea del documento. Cada vez que lo hagamos, este accederá a una siguiente línea. Si no existe ninguna línea, este devolverá **null**.
(Ingeniárselas para leer el archivo con todo lo mencionado y bucles).

*4. Al final, debemos cerrar el Stream y el Buffer:

```
buffer.close();
stream .close();
```

fileList()

Devuelve un array (String) con los ficheros que nuestra aplicación ha guardado.

Título III: Almacenamiento en SD

Primero, deberemos darle a nuestra app el permiso: **android.permission.WRITE_EXTERNAL_STORAGE** (consultar Capítulo I, título III).

Escribir en SD

(Todo dentro de un try catch)

1. Primero, debemos obtener la ruta de nuestra tarjeta SD. Esto lo haremos creando un objeto **File** que lo igualaremos al método estático **getExternalStorageDirectory()** de la clase **Environment** .
2. Una vez tengamos la ruta de nuestra SD, debemos crear otro objeto **File** que hará referencia al archivo en el que queremos escribir, indicando al constructor la ruta (el objeto **File** anterior con el método **.getPath()**) de la SD y el nombre del archivo.
3. Creamos un objeto **OutputStreamWriter** al que le pasamos al método constructor el método **openFileOutput()** al que le tenemos que indicar un String (el nombre del archivo) y la siguiente constante: **Activity.MODE_PRIVATE**.
4. Escribiremos la información deseada dentro del archivo con el método **write(String s)** sobre nuestro Stream.
5. Limpiamos el Stream con el método **flush()**.
6. Cerramos el Stream con el método **close()**.

```
File tarjetaSD = Environment.getExternalStorageDirectory();
File archivo = new File(tarjetaSD.getPath(), “archivo.txt”);
OutputStreamWriter stream = new
OutputStreamWriter(openFIleOutput(“archivo.txt”,
Activity.MODE_PRIVATE));
stream.write(“Hola”);
stream.flush();
stream.close();
```

Leer SD

(Todo dentro de un try catch)

1. Localizamos la ruta de la tarjeta SD con un objeto de tipo **File** igualándolo al método estático **getExternalStorageDirectory()** de la clase **Environment** .
2. Creamos un objeto **File** que haga referencia al archivo deseado, pasándole como parámetros al constructor la ruta de la SD (el objeto anterior .**getPath()**) y el nombre del archivo en un String.
3. Creamos un objeto de la clase **InputStreamReader** al que le pasamos por el constructor el método **openFileInput(String s)** (s = nombre del archivo.)
4. Creamos un buffer con la ayuda de la clase **BufferedReader** al que le pasamos al constructor nuestro Stream.
5. Usaremos el método **readLine()** sobre nuestro buffer las veces que queramos para extraer la información del archivo. Cuando haya pasado por todas las líneas, devuelve **null** .
6. Cerramos el Buffer.
7. Cerramos el Stream.

```
File tarjetaSD = Environment.getExternalStorageDirectory();
File archivo = new File(tarjetaSD.getPath(), “archivo.txt”);
InputStreamReader stream = new
InputStreamReader(openFileInput(“archivo.txt”));
BufferedeReader buffer = new BufferedReader(stream);
String primeraLinea = buffer.readLine();
buffer.close();
stream.close();
```

Capítulo VI: Acceso a BBDD

En este curso, trabajaremos con el SGDB **SQLite**.
Este presenta un tamaño pequeño , no necesita servidor, poca configuración, es transaccional y es de código libre.
La principal clase de este capítulo será **SQLiteOpenHelper** y sus métodos principales: **onCreate()** , **onUpgrade()**.
Estos dos últimos métodos hará falta sobreescribirlos para crear y actualizar nuestra BBDD.

Título I: Creación de tabla dentro de BBDD

Librerías a usar:

android.database.sqlite.*;

1. Para crear una BBDD, deberemos crear una nueva clase Java que herede de **SQLiteOpenHelper** (en esta caso, se va a llamar **AdminSQLite**). Para ello, deberemos sobreescribir los métodos y el constructor (en este último, nombramos al constructor de la superclase pasándole los 4 parámetros):

```
public AdminSQLite(Context context, String name,
SQLiteDatabase.CursorFactory factory, int version){
        super(context, name, factory, version);
}
public void onCreate(SQLiteDatabase basedatos)
public void onUpgrade(SQLiteDatabase basedatos, int i, int il)
```

2. A continuación, debemos crear nuestra tabla dentro del método **onCreate()** así como sus campos de clase. Esto lo haremos con lenguaje SQL (sencillo):

```
public void onCreate(SQLiteDatabase basedatos){
```

```
        basedatos.execSQL("create table articulos(codigo int
        primary key, descripcion text, precio real)");
}
```

Donde **articulos, codigo, descripcion** y **precio** son nombres que nosotros damos.
int, **text** y **real** son tipos de datos (int, String, float)
Y con **primary key**, indicamos que **codigo** va a ser una clave de la tabla.

Con esto, ya tendríamos nuestra tabla creada dentro de nuestra base de datos.

Título II: Escribir datos en BBDD

1. Primero, debemos abrir nuestra base de datos creando un objeto de la clase que sobreescribimos (en este caso **AdminSQLite**) al que le pasaremos como constructor el **context this**, en **name** el nombre de nuestra BBDD **"administracion"** (por ejemplo), en **factory** un **null** y en la **version** un **1**.
2. Creamos un objeto que nos permita escribir y leer en la BBDD. Este será perteneciente a **SQLiteDatabase** y lo igualaremos al objeto **AdminSQLite** creado anteriormente **.getWritableDatabase()**.

3. Creamos un registro con la ayuda de **ContentValues** (usamos constructor default ()).
Establecemos con la ayuda del método **put(String key, … value)** los diferentes campos que queramos escribir (... hace referencia a cualquier dato primitivo o String) (**key** se corresponde con los campos de clase de la tabla a la que accedamos).
4. Añadimos los datos del registro a nuestra base de datos escribiendo sobre nuestro objeto **SQLiteDatabase** el método **insert()** al que le pasamos el nombre de la tabla a la que queremos acceder (en este

caso, **“articulos”**(la tabla creada en el título I)), un **null** y el registro formado en el punto 3.
5. Cerramos la BBDD con **close()**.

```
AdminSQLite admin = new AdminSQLite(this, “administracion”, null, 1);
SQLiteDatabase basedatos = admin.getWritableDatabase();

ContentValues registro = new ContentValues();
registro.put(“codigo”, 1);
registro.put(“descripcion”, “asdfasdfasdf”);
registro.put(“precio”, 20);

basedatos.insert(“articulos”, null, registro);
basedatos.close();
```

Título III: Consultar datos de BBDD

1. Primero, debemos abrir nuestra base de datos creando un objeto de la clase que sobreescribimos (en este caso **AdminSQLite**) al que le pasaremos como constructor el **context this**, en **name** el nombre de nuestra BBDD **“administracion”** (por ejemplo), en **factory** un **null** y en la **version** un **1**.
2. Creamos un objeto que nos permita leer y escribir la BBDD. Este será perteneciente a **SQLiteDatabase** y lo igualaremos al objeto **AdminSQLite** creado anteriormente **.getWritableDatabase()**.

3. Creamos un objeto **Cursor** que igualamos al objeto **SQLiteDatabase** creado anteriormente y a su método
rawQuery(String sql, selectionArgs s);
en **sql**, indicamos que seleccione los campos **descripcion** y **precio** de la tabla **articulos** donde el campo **codigo** sea igual a y el dato que queramos

(**select descripcion, precio from articulos where codigo=1**)
en **s,** indicamos un **null**.

*Si indicamos **select ***, obtendremos todos los campos.
Si solo queremos indicar que obtengamos el articulo que tiene un mayor **codigo, ejecutaremos lo siguiente:
select * from articulos where codigo = (select max(codigo) from articulos)

4. Si el **codigo** indicado en el punto 3 existía, la información se almacenará dentro del objeto **Cursor** si no, no.
Podemos ejecutar el método **moveToFirst()** sobre nuestro objeto **Cursor**. Este devolverá **true** si el **codigo** existía y **false** si no.

5. Para ver los elementos obtenidos en el punto 3 (**descripcion** y **precio**), debemos ayudarnos de dos variables que almacenarán el objeto **Cursor .getString(int i).** En **i** indicamos el elemento que queremos obtener (en el punto 3, se ordenan en una especie de array, siendo **descripcion** la posición 0 y **precio** la 1).

6. Cerramos nuestra BBDD con **close()**.

```
AdminSQLite admin = new AdminSQLite(this, “administracion”, null, 1);
SQLiteDatabase basedatos = admin.getWritableDatabase();
Cursor fila = basedatos.rawQuery(“select descripcion, precio from articulos where codigo=1”, null);
if(fila.moveToFirst()){
        String descripcion = fila.getString(0);
        String precio = fila.getString(1);
}
basedatos.close();
```

Título IV: Eliminar datos de BBDD

1. Primero, debemos abrir nuestra base de datos creando un objeto de la clase que sobreescribimos (en este caso **AdminSQLite**) al que le pasaremos como constructor el **context this**, en **name** el nombre de nuestra BBDD **"administracion"** (por ejemplo), en **factory** un **null** y en la **version** un **1**.
2. Creamos un objeto que nos permita leer y escribir la BBDD. Este será perteneciente a **SQLiteDatabase** y lo igualaremos al objeto **AdminSQLite** creado anteriormente **.getWritableDatabase()**.

3. (Queremos borrar el **codigo**=1)
Para borrar un dato, debemos ejecutar el método **delete()** sobre nuestro objeto de tipo **SQLiteDatabase**, al que le pasaremos como parámetros un String con el nombre de la tabla ("**articulos**"), otro String con la condición a evaluar (**"codigo=1"**) y un **null**.
El método **delete()** devuelve un int indicando el número de datos que ha borrado de la BBDD. Si este no borra nada, devuelve 0.

4. Cerrar la BBDD.

```
AdminSQLite admin = new AdminSQLite(this, "administracion", null, 1);
SQLiteDatabase basedatos = admin.getWritableDatabase();
int amount = basedatos.delete("articulos", "codigo=1", null);
basedatos.close();
```

Título V: Modificar datos de BBDD

1. Primero, debemos abrir nuestra base de datos creando un objeto de la clase que sobreescribimos (en este caso **AdminSQLite**) al que le pasaremos como constructor el **context this**, en **name** el nombre de nuestra BBDD **"administracion"** (por ejemplo), en **factory** un **null** y en la **version** un **1**.

2. Creamos un objeto que nos permita leer y escribir la BBDD. Este será perteneciente a **SQLiteDatabase** y lo igualaremos al objeto **AdminSQLite** creado anteriormente **.getWritableDatabase()**.

3. Creamos un registro con la ayuda de **ContentValues** (usamos constructor default ()).
Establecemos con la ayuda del método **put(String key, … value)** los diferentes campos que queramos escribir (... hace referencia a cualquier dato primitivo o String) (**key** se corresponde con los campos de clase de la tabla a la que accedamos).

4. Para cambiar un dato, debemos ejecutar el método **update()** sobre nuestro objeto **SQLiteDatabase**, al cual le pasaremos en un String la tabla (**“articulos”**), nuestro objeto **ContentValues** con las modificaciones y un String con el lugar donde va a ejercer estas modificaciones (**“codigo=1”**). También pasaremos un **null.**
Update devuelve un int con el número de elementos modificados. Si este no cambia nada, devuelve 0.

5. Cerramos la BBDD con **close**.

```
AdminSQLite admin = new AdminSQLite(this, “administracion”, null, 1);
SQLiteDatabase basedatos = admin.getWritableDatabase();
ContentValues registro = new ContentValues();
registro.put(“codigo”, 1);
registro.put(“descripcion”, “asdfasdfasdf”);
registro.put(“precio”, 20);
int cantidad = basedatos.update(“articulos”, “codigo=1”, null);
basedatos.close();
```

Capítulo VII: Layouts

Todos los layouts se añaden al igual que los componentes; arrastrándolos a nuestra vista diseño. También hará falta especificar coordenadas con los bordes en la vista blueprint.

Título I: LinearLayout

Este alinea todos los campos en una única dirección, de manera vertical u horizontal. Se puede especificar la dirección de este con el atributo **orientation**.

Título II: TableLayout

Permite acomodar los componentes por filas y columnas. Cada **TableLayout** tiene un conjunto de componentes **TableRow** que es el que agrupa los componentes por cada fila.
Un **TableRow** es prácticamente igual a un **LinearLayout** horizontal.
Para añadir componentes a un **TableLayout**, deberemos agregarlos directamente en el Component Tree dentro de su respectivo **TableRow**.
*Podemos seleccionar con shift todos los elementos de una misma fila y pulsar en el botón “Distribute Weigths Evenly” para que estos elementos ocupen la fila entera.

layout_span

Este atributo nos permite establecer al componente cuantas columnas ocupará.

layout_weight

Este atributo nos permite establecer al componente cuantos elementos estirados ocupará (para cuando **layout_span** no nos sirve).

Título III: FrameLayout

Este layout coloca unas vistas encima de otras. Suele usarse para mostrar un único control en su interior a modo de contenedor para un elemento sustituible como una imagen.
A medida que se vayan añadiendo vistas, estas se irán superponiendo a las anteriores.
*Es interesante mezclar este componente con el método **setVisibility()** (Consultar Capítulo II, título IV, **Hacer invisible/visible un componente**).

Capítulo VIII: Audio

Principales clases:
MediaPlayer: Archivos de audio muy largos.
SoundPool: Archivos de audio muy cortos (como pulso de botones); tamaño máximo = 1MB.
*Los archivos de audio deben tener el nombre en minúsculas, no pueden usar caracteres especiales y deben ser .mp3 .
Recomendable llamar a los audios **sound_nombre** .

Agregar audios a proyecto

Copiamos los archivos dentro de su carpeta.
Nos dirigimos a la carpeta (Project) app>res , daremos click dcho sobre esta , New>Directory y lo nombramos: raw .
Click dcho sobre la nueva carpeta y pegamos los audios.

Título I: Reproducción con SoundPool

Preparar el audio

Dentro del método **onCreate()** ya que esto sólo crea el sonido a reproducir:
1. Debemos crear un objeto **SoundPool** al que le pasaremos por el constructor el máximo de reproducciones simultáneas (int), el tipo de Stream de audio (**AudioManager.STREAM_MUSIC**) y la calidad de reproducción (ya no se implementa pero se sigue solicitando así que introducir un 1).

2. Cargamos nuestra pista de audio dentro del objeto **SoundPool**:
Creamos una variable int que la igualamos al objeto anterior al que le aplicamos el método **load()** y le pasamos como argumentos el activity donde se va a ejecutar (**this**), el archivo .mp3 (**R.raw.sound_nombre**) y la prioridad del audio (no se implementa, colocar un 1).

sp = new SoundPool(123, AudioManager.STREAM_MUSIC, 1**);**
numero = sp.load(this, R.raw.sound_nombre, 1);

(numero y sp fueron declarados fuera del **onCreate()** para poder ser usados al ejecutar el audio).

Reproducir el audio

Ejecutar sobre nuestro objeto **SoundPool** el método **play()**:
sp.play(int a, int b, int c, int d, int e, int f);

- a: Wl ID del sonido (el int creado anteriormente, en este caso, **numero**).
- b: El volumen por el altavoz izqdo (sirve un 1).
- c: El volumen por el altavoz dcho (sirve un 1).
- d: La prioridad del sonido (sirve un 1).
- e: Si va a repetirse en bucle (-1 = si, 0 = no, 1 = se repite instantaneo).
- f: La velocidad de reproducción (0 para default).

sb.play(nombre, 1, 1, 1, 0, 0);

Título II: Reproducción con MediaPlayer

1. Creamos un objeto **MediaPlayer** que igualaremos al método estático perteneciente a **MediaPlayer create()** al que le pasaremos como argumentos un activity (**this**), y la ruta del archivo (**R.raw.sound_nombre**).
1'. Creamos un objeto **MediaPlayer** con el constructor default y luego (dentro de un try catch) le ejecutamos el método **setDataSource(String s)** donde **s** es la ruta del archivo .mp3 (ver título III, **Iniciar grabación**, 1.) para indicarle qué archivo debe reproducir. A continuación, le ejecutamos el método **prepare()** para que compruebe todo.

2. Empezamos a reproducir el sonido con el método **start()** sobre nuestro objeto **MediaPlayer**. (este método no es necesario que esté dentro de un try catch)
*Si vamos a crear un programa que recree una pista de audio, es interesante crear un arreglo de **MediaPlayer** en vez de un solo objeto.

```
MediaPlayer mp = MediaPlayer.create(this,
R.raw.sound_nombre);
mp.start();
```

////

```
MediaPlayer mp = new MediaPlayer();
try{
        mp.setDataSource(archivo);
        mp.prepare();
}catch(Exception e){
        código;
}
mp.start();
```

isPlaying()

Devuelve **true** si la canción se está reproduciendo, **false** si no.

pause()

Pausa la canción.

start()

Empieza o reanuda desde donde se pausó la canción.

stop()

Detiene la canción (si se aplica **start**, empieza de 0).

Este método “borra” el contenido del objeto al que se le aplique por lo que pierde el archivo que se le dió en un inicio. Si queremos volver a reproducir este archivo, debemos volver a asignarlo:
(obj **MediaPlayer** = **cancion):**
cancion.stop();
cancion = MediaPlayer.create(this, R.raw.sound_nombre);

Si se le aplicó el método **stop** a un objeto **MediaPlayer**, este será igual a **null**.

setLooping(boolean b)

Establece que la canción se repetirá en bucle (**b = true**) o no (**b = false**).

Título III: Grabación con MediaRecorder

info: https://developer.android.com/reference/android/media/MediaRecorder

Iniciar grabación

1. Debemos crear el archivo de audio donde se almacenará el audio grabado. Esto lo hacemos igualando una variable String al método estático **getExternalStorageDirectory()** de la clase **Environment** a la vez del método **getAbsolutePath**, concadenándolo con el nombre del archivo y su extensión (que puede ser .mp3 o .3gp).

2. A continuación, creamos nuestro objeto **MediaRecorder** con el constructor default.

3. Después, indicamos a nuestro objeto **MediaRecorder** que va a empezar a capturar audio con el método **setAudioSoruce()** al cual le pasamos como argumento el campo estático **MIC** de la clase **MediaRecorder.AudioSource** .

4. Indicamos qué formato queremos que tenga el audio de salida con el método **setOutputFormat()** al que le pasamos el campo estático **THREE_GPP** de la clase **MediaRecorder.OutputFormat** .(Da igual que este formato sea .3gp y el del archivo .mp3, este se transforma en el proceso).

5. Realizamos el encoder de nuestro audio con el método **setAudioEncoder()** al que le pasamos como parámetro el campo estático **AMR_NB** de la clase **MediaRecorder.OutputFormat** .

6. Indicamos al audio que se guardará en el archivo creado en el punto 1: declaramos el método **setOutputFile()** y le pasamos como argumento la variable String creada en el punto 1.

7. (Dentro de un try catch)
Ejecutamos los métodos **prepare()** y **start()** para que prepare el dispositivo y comience a grabar el audio.

```
String archivo =
Environment.getExternalStorageDirectory().getAbsolutePath() +
“archivo.mp3”;
MediaRecorder grabacion = new MediaRecorder();
grabacion.setAudioSource(MediaRecorder.AudioSource.MIC);
grabacion.setOutputFormat(MediaRecorder.OutputFormat.THR
EE_GPP);
grabacion.setAudioEncoder(MediaRecorder.OutputFormat.AMR
_NB);
grabacion.setOutputFile(archivo);
try{
        grabacion.prepare();
        grabacion.start();
}catch(IOException e){
        código;
```

}

Detener grabación

1. Pausamos la grabación.
2. Liberamos la grabación.

```
grabacion.stop();
grabacion.release();
```

*Es conveniente que una vez que se haya terminado la grabación, borremos su contenido con **grabacion = null;** , pero no necesario.

**Para que el usuario pueda grabar audio, debemos programar el mensaje de permitir acceso a micrófono y este lo deberá aceptar.

Pedir permisos al usuario

(Dentro de **onCreate()**:)

```
if (ContextCompat.checkSelfPermission(getApplicationContext(),
Manifest.permission.WRITE_EXTERNAL_STORAGE) !=
PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(getApplicationContext(),
Manifest.permission.RECORD_AUDIO) !=
PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(MainActivity.this,
new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE,
Manifest.permission.RECORD_AUDIO}, 1000);
}
```

Para pedir permisos de almacenamiento de archivos y reproducción de audio.

También deberemos ir al archivo manifest y solicitar los permisos: **WRITE_EXTERNAL_STORAGE**, **RECORD_AUDIO** . (En este caso). (Consultar capítulo III, título X, **Dar permisos a nuestra app**).

Título IV: Voz a texto

Para ello, usaremos la herramienta Dictation de google, la cual se puede usar aunque no estemos conectados a internet, no hace falta pedir permisos de micrófono y es capaz de reconocer el idioma en el que le estamos hablando.

1. Para comenzar, es recomendable escribir el siguiente código dentro de una función que se reproducirá cuando el usuario de click sobre un botón o algo similar. Fuera de la función pero en la misma clase, especificaremos la siguiente variable:
private static final int REQ_CODE_SPEECH_INPUT = 100;

2. A continuación, dentro de la función especificaremos el siguiente código (cuando esta función se reproduzca, aparecerá una pestaña con un micrófono donde se podrá hablar y el texto indicado):

```
Intent intent = new
Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODE
L, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE,
Locale.getDefault());
intent.putExtra(RecognizerIntent.EXTRA_PROMPT, texto);

try{
      startActivityForResult(intent,
REQ_CODE_SPEECH_INPUT);
} catch (ActivityNotFoundException e){
      código;
```

}

3. Posteriormente, sobre escribiremos la función **onActivityResult()** pasándole a la función padre los tres parámetros y especificando el siguiente código en el que obtendremos el texto que el usuario dictó en un String:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data){
        super.onActivityResult(requestCode, resultCode, data);

        if(requestCode == REQ_CODE_SPEECH_INPUT){
                if(resultCode==RESULT_OK && data!=null){
                        ArrayList<String> texto = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
                        String texto_dictado = texto.get(0);
                }
        }
}
```

El código total quedaría así:

```
private static final int REQ_CODE_SPEECH_INPUT = 100;

public void funcion(){
        Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE,
Locale.getDefault());
```

```
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
"Hable:");

        try{
                startActivityForResult(intent,
        REQ_CODE_SPEECH_INPUT);
        } catch (ActivityNotFoundException e){
            código;
        }
}

protected void onActivityResult(int requestCode, int resultCode,
Intent data){
        super.onActivityResult(requestCode, resultCode, data);

        if(requestCode == REQ_CODE_SPEECH_INPUT){
            if(resultCode==RESULT_OK && data!=null){
                ArrayList<String> texto =
data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESUL
TS);
                String texto_dictado = texto.get(0);
            }
        }
}
```

Título V: Texto a voz

Para convertir texto a voz, usaremos una herramienta que debemos copiar y pegar en una nueva clase de Java de nuestro proyecto.

1. Primero, copiamos y pegamos el siguiente código (e importamos los paquetes necesarios) en una nueva clase Java de nuestro proyecto llamada TTSManager.java:

```
public class TTSManager{
	private TextToSpeech mTts = null;
	private boolean isLoaded = false;

	public void init(Context context){
		try{
			mTts = new TextToSpeech(context,
onInitListener);
		}catch(Exception e){
			e.printStackTrace();
		}
	}

	private TextToSpeech.OnInitListener onInitListener = new
TextToSpeech.OnInitListener(){
		@Override
		public void onInit(int status){
			Locale spanish = new Locale(“es”, “ES”);
			if(status == TextToSpeech.SUCCESS){
				int result = mTts.setLanguage(spanish);
				isLoaded = true;

				if(result ==
TextToSpeech.LANG_MISSING_DATA || result ==
TextToSpeech.LANG_NOT_SUPPORTED){
					Log.e(“error”, “Este lenguaje no esta
permitido”);
				}
			}else{
				Log.e(“error”, “Fallo al Inicializar!”);
			}
		}
	};
```

```
    public void shutDown(){
        mTts.shutdown();
    }

    public void addQueue(String text){
        if(isLoaded){
            mTts.speak(text, TextToSpeech.QUEUE_ADD,
null);
        }else{
            Log.e("error", "TTS Not Initialized");
        }
    }

    public void initQueue(String text){
        if(isLoaded){
            mTts.speak(text, TextToSpeech.QUEUE_FLUSH,
null);
        }else{
            Log.e("error", "TTS Not Initialized");
        }
    }
}
```

2. Una vez copiado el anterior código, nos dirigimos a nuestro activity.
Una vez allí, creamos la siguiente variable en los campos de clase:
TTSManager ttsManager = null;

3. Nos dirigimos al método **onCreate()** e inicializamos la variable que creamos en el punto 2:
ttsManager = new TTSManager();
ttsManager.init(this);

4. El siguiente código es conveniente meterlo dentro de una función, ya que será el que se ejecute cuando queramos que nuestra aplicación reproduzca texto:

```
String texto = “Hola”; //Texto que queramos reproducir
ttsManager.initQueue(texto);
```

5. Posteriormente, sobreescribimos el método **onDestroy** , pasándole los parámetros al método padre para cerrar nuestra salida de voz cuando la aplicación se cierre:

```
@Override
protected void onDestroy(){
        super.onDestroy();
        ttsManager.shutDown();
}
```

Capítulo IX: Fotos y Vídeos

Título I: Tomar fotos y guardarlas

Permisos de cámara y guardado

(Todo explicado en vídeo nº46)

1. Para trabajar con la cámara, debemos pedirle permiso al usuario. Esto lo hacemos mediante el archivo (Project) app>manifests>AndroidManifest.xml donde solicitaremos los permisos de cámara y almacenamiento: (Consultar capítulo I, título III).

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

2. También deberemos pegar el siguiente código entre el cierre de la etiqueta **activity** y **application** :
En la línea 9/11, deberemos clicar para decirle a Android Studio que queremos generar el documento @xml/file_paths

```
<provider
   android:name="android.support.v4.content.FileProvider"
   android:authorities="com.example.android.fileprovider"
   android:exported="false"
   android:grantUriPermissions="true">
   <meta-data

android:name="android.support.FILE_PROVIDER_PATHS"
      android:resource="@xml/file_paths">
```

</meta-data>
</provider>

3. Dentro de este documento file_paths, nos iremos a la parte de texto del xml y sustituiremos la etiqueta **PreferencesScreen** por una etiqueta **paths** (que tenga la misma url).
Dentro de esta etiqueta path, creamos otra llamada **external-path** con dos atributos, un **name** y un **path** .
En **name** indicaremos **my_images**
En **path** indicaremos: **“Android/data/”** + (el nombre de una de las carpetas que hay dentro de (Project) app>java) + **“/files/Pictures”**
De esta forma, todas las fotos que se guarden se guardarán dentro de esta carpeta.
El código del archivo **file_paths** se vería así:

```
<?xml version=”1.0” encoding=”utf-8”?>
<paths
xmlns:android=”https://schemas.android.com/apk/res/android”>
        <external-path
                name=”my_images”
                path=”Android/data/nombre/files/Pictures” />
</paths>
```

4. Además, deberemos pegar el siguiente código en la parte lógica de nuestra app para mostrar las etiquetas de permitir al usuario: (Dentro de **onCreate()**):

```
if (ContextCompat.checkSelfPermission(MainActivity.this,
Manifest.permission.WRITE_EXTERNAL_STORAGE) !=
PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(MainActivity.this,
```

```
Manifest.permission.CAMERA) !=
PackageManager.PERMISSION_GRANTED) {
   ActivityCompat.requestPermissions(MainActivity.this, new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE,
Manifest.permission.CAMERA}, 1000);
}
```

(Importar paquetes necesarios)

(para el resto del capítulo (tomar fotos y guardarlas, consular vídeo 46, min 17:00), ya que todo es un copy-paste).

Título II: Grabación de vídeo

(consultar vídeo 47, min 2:30, ya que todo es un copy-paste).

Capítulo X: Generar APK con Android Studio

Para ello, una vez que tengamos nuestra app terminada, nos dirigiremos al menú superior,
Build>Generate Signed Bundle / APK…

1. Seleccionamos la opción APK, siguiente.
2. Key Store Path explicado en vídeo nº67 (3:20 - 7:00), siguiente.
3. Seleccionamos **debug** si es un APK de pruebas, **release** si es un APK de lanzamiento de una app. Seleccionamos también la **V2**, que nos da una APK ya firmada. Finish.
4. El APK tarda unos segundos en crearse. Una vez creado, aparece en la carpeta:
C:/Users/**nombre_usuario**/AndroidStudioProjects/**nombre_app**/app/release/

(El archivo .json no es de importancia).

Una vez obtengamos este archivo, ya lo podemos copiar a diferentes dispositivos Android para poder instalar nuestra app.