



PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

TEMA 8:

Controles avanzados.

ÍNDICE

1. Vistas de selección: Spinners.
2. Vistas de selección: Listas.
3. Menús.
4. Cuadros de diálogo.
5. Ejercicios de consolidación.



PMDM

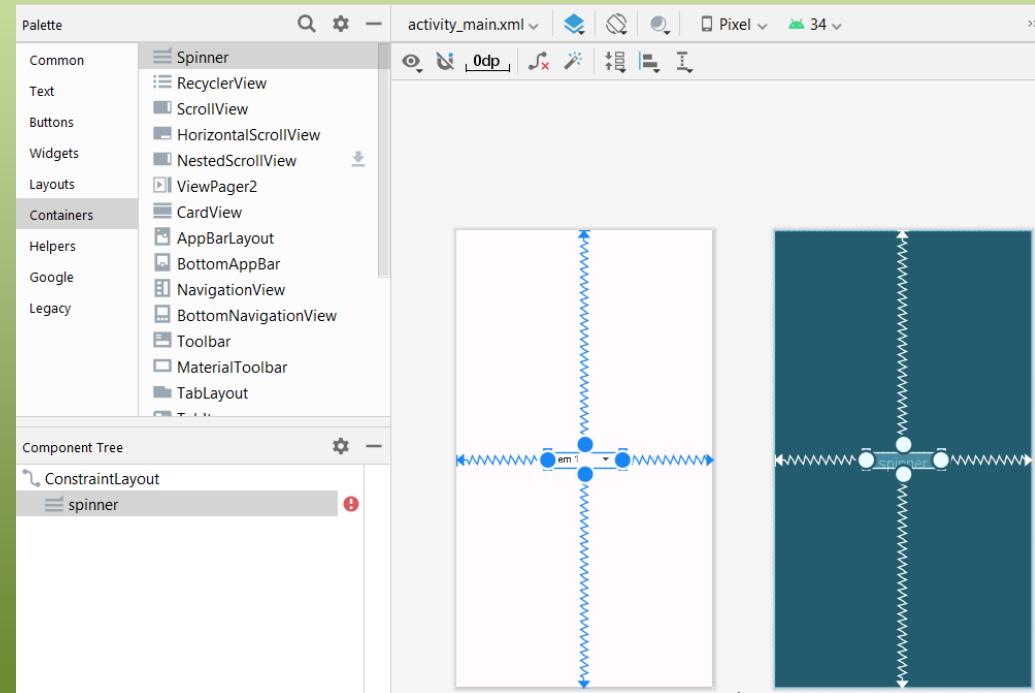
Controles avanzados.

1. Vistas de selección: Spinner



1.- Vistas de selección: Spinner

- Un Spinner es una vista que permite al usuario elegir un valor de una lista de posibles valores (lista despegable)
- Para construir un Spinner bastará con arrastrarlo a nuestro layout:

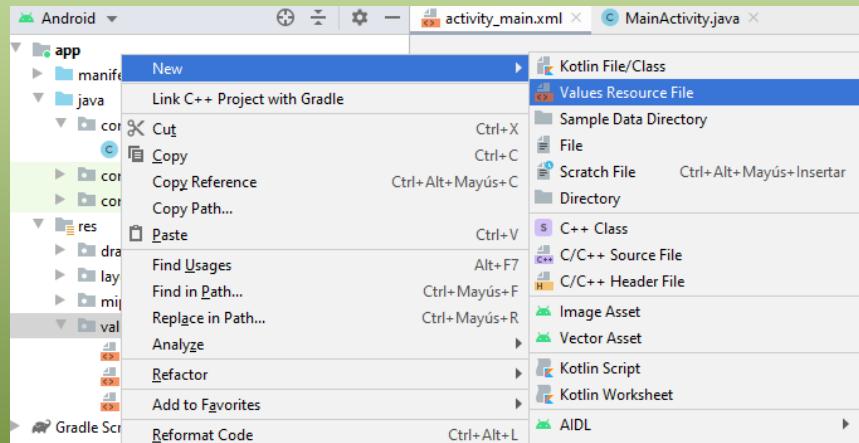


1.- Vistas de selección: Spinner

- A continuación debemos elegir la forma con la que cargar los datos del Spinner:
 - Desde un fichero (a través xml o a través de código).
 - Volcando una estructura de datos desde código (array, listado, mapa,...).
 - Desde una base de datos.
 - ...

1.- Vistas de selección: Spinner

- **Desde un fichero (mediante xml).**
- Para ello, crearemos nuestro fichero *productos.xml* en *res/values* (click derecho en values -> New -> Values Resource File).



1.- Vistas de selección: Spinner

- Y rellenamos el fichero con los datos que queremos que aparezcan en el spinner.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="productos">
        <item>Elige un producto</item>
        <item>Pimentón de la Vera</item>
        <item>Cerezas del Jerte</item>
        <item>Torta del Casar</item>
        <item>Kombucha</item>
    </string-array>
</resources>
```

1.- Vistas de selección: Spinner

- Bastaría con asignar el xml al Spinner mediante el atributo ***android:entries="@array/productos"***
- y ya lo tendríamos funcionando.

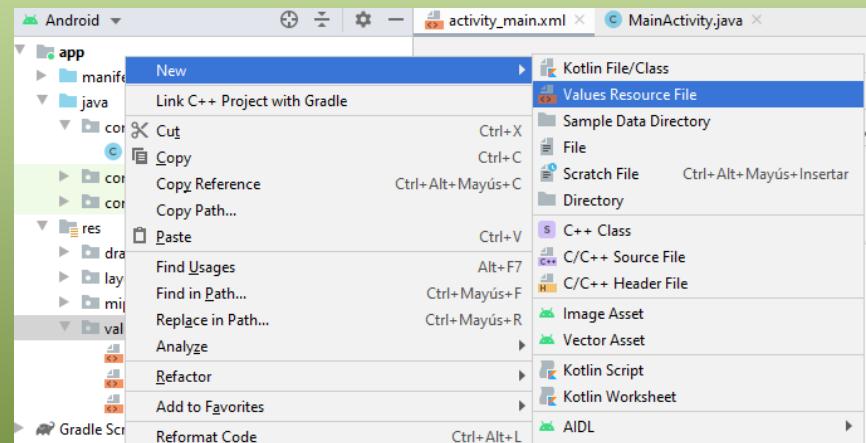
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Spinner
        android:id="@+id/spinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:entries="@array/productos" // Line 13, highlighted with a red box
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

1.- Vistas de selección: Spinner

- **Desde un fichero (a través de código).**
- Deberemos, igual que en el apartado anterior, crear nuestro fichero *productos.xml* en *res/values* (click derecho en *values* -> New -> Values Resource File).



1.- Vistas de selección: Spinner

- Para ello necesitaremos crear un objeto adaptador de la clase *ArrayAdapter*, que asocia cada cadena de nuestro xml a una vista que será mostrada en el Spinner. Utilizaremos el método *createFromResource*, al que le pasaremos el contexto de la aplicación (la actividad en la que estamos), el array del xml, y un identificador de layout para indicar el tipo de vista que queremos asociar a cada ítem de la lista.

1.- Vistas de selección: Spinner

- El identificador de layout que le pasaremos es *android.R.layout.simple_spinner_item*. Se trata de un layout predefinido por el sistema (por eso comienza con *android.R*, a diferencia de los definidos en nuestra aplicación que comienzan con R), que consiste simplemente en una fila que muestra una cadena de texto sencilla.
- Seguidamente llamamos al método *setDropDownViewResource* (de nuestro adaptador) para definir el tipo de layout que se utilizará para mostrar la lista de opciones (el marco de dicha lista). Usaremos el *android.R.layout.simple_spinner_dropdown_item*.

1.- Vistas de selección: Spinner

- Por último asociamos el adaptador al Spinner con el método *adapter* (*setAdapter*) de nuestro spinner.
- Obviamente, todo esto habría que hacerlo sin el *android:entries="@array/productos"* en el xml (para que no lo cargue de manera estática).

```
class MainActivity : AppCompatActivity() {

    private lateinit var spinnerProductos: Spinner

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        spinnerProductos = findViewById<Spinner>(R.id.spinnerProductos)

        val adaptador = ArrayAdapter.createFromResource(context: this, R.array.productos, android.R.layout.simple_spinner_item)
        adaptador.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        spinnerProductos.adapter = adaptador
    }

}
```

1.- Vistas de selección: Spinner

- Para manejar los eventos relacionados con el Spinner debemos implementar la interface *AdapterView.OnItemSelectedListener*.

```
class MainActivity : AppCompatActivity(), AdapterView.OnItemSelectedListener {  
  
    private lateinit var spinnerProductos: Spinner  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        spinnerProductos = findViewById<Spinner>(R.id.spinnerProductos)  
  
        val adaptador = ArrayAdapter.createFromResource(context: this, R.array.productos, android.R.layout.simple_spinner_item)  
        adaptador.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)  
        spinnerProductos.adapter = adaptador  
        spinnerProductos.onItemSelectedListener = this  
    }  
}
```

1.- Vistas de selección: Spinner

- Al seleccionar un ítem de la lista, se ejecuta el método *onItemSelected*, que recibirá la propia vista asociada al ítem pulsado, la posición del ítem en el Spinner y su identificador.

```
override fun onItemSelected(parent: AdapterView<*>?, view: View?, position: Int, id: Long) {
    //Se invoca cuando se selecciona un ítem de la lista
    if(position == 2) {
        Toast.makeText(context: this, text: "Ha seleccionado Cerezas del Jerte", Toast.LENGTH_SHORT).show()
    } else {
        Toast.makeText(context: this, text: "Ha seleccionado cualquier posición, excepto la 2", Toast.LENGTH_SHORT).show()
    }
}

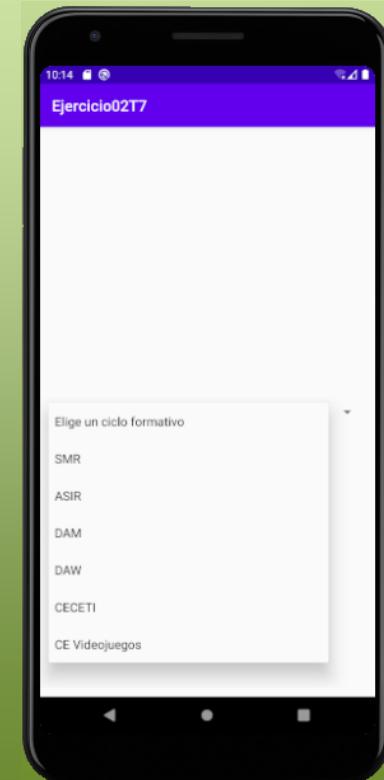
override fun onNothingSelected(parent: AdapterView<*>?) {
    //Se invoca cuando no se selecciona nada
}
```

EJERCICIOS

- **Ejercicio 01:** Crea un proyecto en Android Studio en cuya actividad nos encontremos un Spinner.
- El Spinner contendrá productos gastronómicos de Extremadura y se cargará mediante un fichero. Este fichero lo especificaremos en el xml de nuestro activity.

EJERCICIOS

- **Ejercicio 02:** Crea un proyecto en Android Studio en cuya actividad nos encontremos un Spinner.
- El Spinner contendrá la oferta formativa del IES Valle del Jerte. Se cargará mediante un fichero (a través de código Kotlin).
- Mostrará un Toast cuando seleccionemos cualquiera de las opciones del Spinner (“Elige un ciclo formativo” no cuenta como opción).

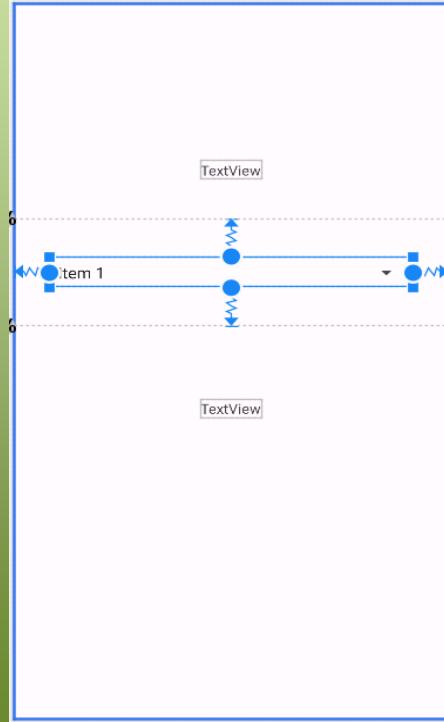


1.- Vistas de selección: Spinner

- **Desde Estructura de Datos (listado).**
- Todo sería igual: hay que utilizar un adaptador para conectar los datos y capturar los eventos de selección implementando la interface *AdapterView.OnItemSelectedListener*.

1.- Vistas de selección: Spinner

- El layout constará de dos TextView y, en el medio, un Spinner que lo redimensionaremos para que ocupe el ancho del dispositivo.



1.- Vistas de selección: Spinner

- También volveremos a utilizar los 2 layouts ya predefinidos por Android:
 - *android.R.layout.simple_spinner_item* -> Para mostrar cada elemento desplegable.
 - *android.R.layout.simple_spinner_dropdown_item* -> Para mostrar el marco de los elementos desplegables.
- Para crear el adaptador de la clase ArrayAdapter ya no tenemos que utilizar el método `createFromResource` (como ocurría anteriormente al obtener los datos de un XML), sino que lo haremos así:
 - `val adaptador: ArrayAdapter<String> = ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, nombre_del_listado)`

1.- Vistas de selección: Spinner

```
class MainActivity : AppCompatActivity(), AdapterView.OnItemSelectedListener {

    private lateinit var spinnerCiudades: Spinner
    private lateinit var textViewResultado: TextView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        spinnerCiudades = findViewById<Spinner>(R.id.spinnerCiudades)
        textViewResultado = findViewById<TextView>(R.id.textViewResultado)

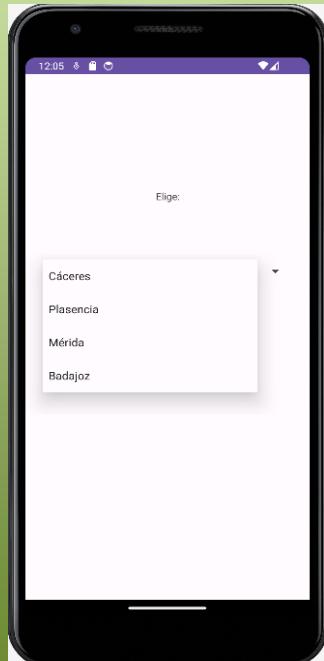
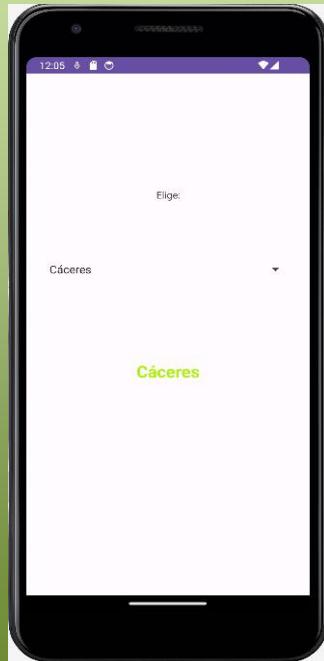
        //LISTADO DE CIUDADES
        val ciudadesList: List<String> = listOf("Cáceres", "Plasencia", "Mérida", "Badajoz")

        //SPINNER
        val adaptador: ArrayAdapter<String> = ArrayAdapter<String>( context: this, android.R.layout.simple_spinner_item, ciudadesList)
        adaptador.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        spinnerCiudades.adapter = adaptador
        spinnerCiudades.onItemSelectedListener = this
    }

    override fun onItemSelected(parent: AdapterView<*>?, view: View?, position: Int, id: Long) {
        textViewResultado.setText(spinnerCiudades.selectedItem.toString())
    }

    override fun onNothingSelected(parent: AdapterView<*>?) {
        TODO( reason: "Not yet implemented")
    }
}
```

1.- Vistas de selección: Spinner



EJERCICIOS

- **Ejercicio 03:** Crea un proyecto en Android Studio que contenga seis Actividades: **ActividadPrincipal**, **Pregunta1**, **Pregunta2**, **Pregunta3**, **Pregunta4** y **Resultados**.
- Este proyecto consistirá en crear un Spinner con cinco posibles respuestas a la pregunta de ¿Cuál es el nombre del hardware de la figura?
- Las respuestas correctas valen 2,5, las incorrectas -1 y las que no se contesten valdrán 0.

EJERCICIOS

- El resultado sería parecido al de las siguientes imágenes:



EJERCICIOS

- Pista:
 - Para implementar la funcionalidad del botón SALIR de la aplicación, bastará con llamar al método finish(). Este sería el código:

```
btnSalir.setOnClickListener { it: View!  
    finish()  
}
```

PMDM

Controles avanzados.

2. Vistas de selección: Listas



2.- Vistas de selección: Listas

- Las listas son vistas de tipo ListView, por lo que podemos introducirlas directamente en un layout como haríamos normalmente con cualquier tipo de vista en Android.
- En caso que nuestra actividad solo contuviera una lista y nada más, podemos hacer que nuestra actividad herede de ListActivity en vez de AppCompatActivity (más adelante veremos un ejemplo de este caso concreto)

2.- Vistas de selección: Listas

- Las ListView, en función del Layout que les apliquemos mediante el adaptador, pueden ser de tres tipos:
 - Listas Implícitas: *android.R.layout.simple_list_item_1*
 - Listas de Selección Única: *android.R.layout.simple_list_item_single_choice*
 - Listas de Selección Múltiple: *android.R.layout.simple_list_item_multiple_choice*

Gladiador
Spiderman
Torrente
Blade
Rocky
Spiderman
Torrente
Blade

Gladiador	<input type="radio"/>
Spiderman	<input type="radio"/>
Torrente	<input checked="" type="radio"/>
Blade	<input type="radio"/>
Rocky	<input type="radio"/>
Spiderman	<input type="radio"/>
Torrente	<input type="radio"/>
Blade	<input type="radio"/>

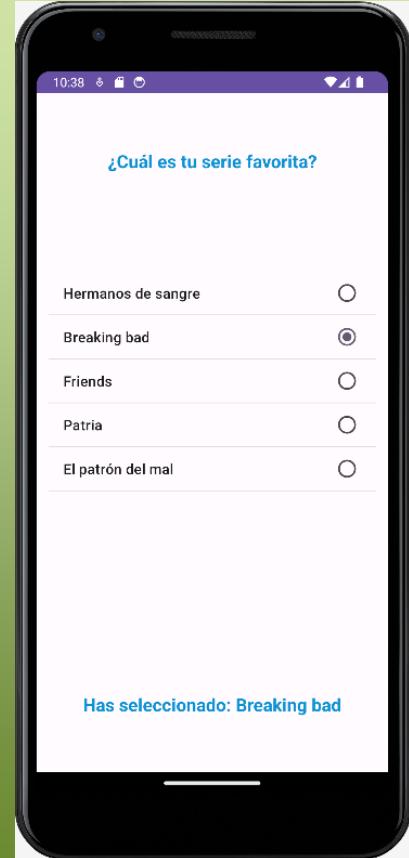
Gladiador	<input type="checkbox"/>
Spiderman	<input type="checkbox"/>
Torrente	<input checked="" type="checkbox"/>
Blade	<input type="checkbox"/>
Rocky	<input checked="" type="checkbox"/>
Spiderman	<input type="checkbox"/>
Torrente	<input checked="" type="checkbox"/>
Blade	<input type="checkbox"/>

2.- Vistas de selección: Listas

- Al igual que los Spinner, los ListView pueden obtener sus datos de una estructura de programación (listado, array...), de un fichero XML, de una base de datos, etc... Se haría igual que lo visto en el punto anterior con los Spinner.
- También podemos crear nosotros un Layout, y asignarlo, y así no depender únicamente de los layouts por defecto que nos proporciona *android.R.layout*.
- Veamos un ejemplo:

EJERCICIOS

- **Ejercicio 04:** Crea un proyecto en Android Studio en cuya actividad nos encontraremos un ListView.
- El ListView contendrá series de televisión y el usuario tendrá que elegir su serie favorita. Dicho ListView se cargará mediante un ArrayList.



2.- Vistas de selección: Listas

```
class MainActivity : AppCompatActivity(), AdapterView.OnItemClickListener {  
  
    private lateinit var listViewSeries: ListView  
    private lateinit var textViewResultado: TextView  
  
    private lateinit var arrayListSeries: ArrayList<String>  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        listViewSeries = findViewById<ListView>(R.id.listViewSeries)  
        textViewResultado = findViewById<TextView>(R.id.textViewResultado)  
  
        //ARRAYLIST DE SERIES  
        arrayListSeries = arrayListOf()  
        arrayListSeries.add("Hermanos de sangre")  
        arrayListSeries.add("Breaking bad")  
        arrayListSeries.add("Friends")  
        arrayListSeries.add("Patria")  
        arrayListSeries.add("El patrón del mal")  
  
        //LISTA  
        val adaptador: ArrayAdapter<String> =  
            ArrayAdapter<String> (context: this, android.R.layout.simple_list_item_single_choice, arrayListSeries)  
        listViewSeries.adapter = adaptador  
        listViewSeries.setOnItemClickListener = this  
  
    }  
  
    override fun onItemClick(parent: AdapterView<*>?, view: View?, position: Int, id: Long) {  
        val msgResultado: String = "${resources.getString(R.string.respuesta_text)} ${listViewSeries.getItemAtPosition(position)}"  
        textViewResultado.setText(msgResultado)  
    }  
}
```

- **Importante:** Para que se quede marcada la selección hecha en el listView, tenemos que añadir ***android:choiceMode="singleChoice"*** a las propiedades del ListView en el activity_main.xml.

2.- Vistas de selección: Listas

- Si quisiéramos utilizar la selección múltiple y mostrar todo aquello que el usuario seleccione, deberíamos modificar tanto el layout como el xml:

```
val adaptador: ArrayAdapter<String> =
    ArrayAdapter<String>( context: this, android.R.layout.simple_list_item_multiple_choice, arrayListSeries)

        android:choiceMode="multipleChoice"
```

- Y, además, modificar el método *onItemClick()*:

```
override fun onItemClick(parent: AdapterView<*>?, view: View?, position: Int, id: Long) {
    var cadenaSeries: String = ""

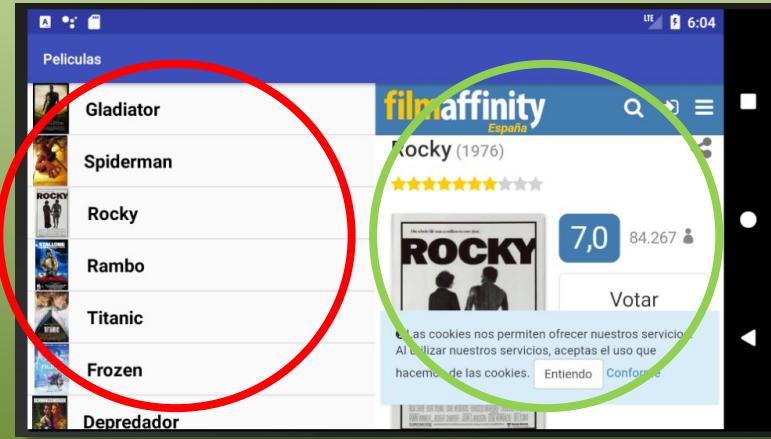
    val seriesSeleccionadas: SparseBooleanArray = listViewSeries.checkedItemPositions

    for(i: Int in 0 .. until < seriesSeleccionadas.size()){
        if(seriesSeleccionadas.valueAt(i) == true) {
            cadenaSeries = "$cadenaSeries - ${listViewSeries.getItemAtPosition(seriesSeleccionadas.keyAt(i))}"
        }
    }
    val msgResultado: String = "${resources.getString(R.string.respuesta_text)} $cadenaSeries"
    textViewResultado.setText(msgResultado)
}
```

- Para finalizar el apartado vamos a crear, paso a paso, un ListView personalizado.
- Este ListView estará enlazado con una WebView.
- Una WebView no es más que un recurso que nos permite integrar contenido web en nuestras apps.

2.- Vistas de selección: Listas

LISTVIEW



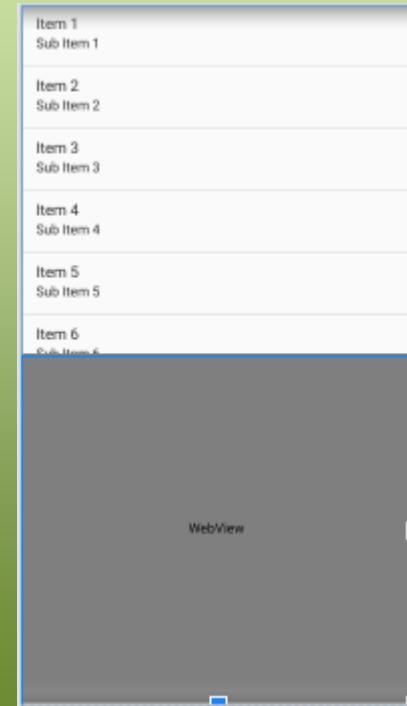
WEBVIEW

- 1.- Diseñamos el layout de la actividad:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

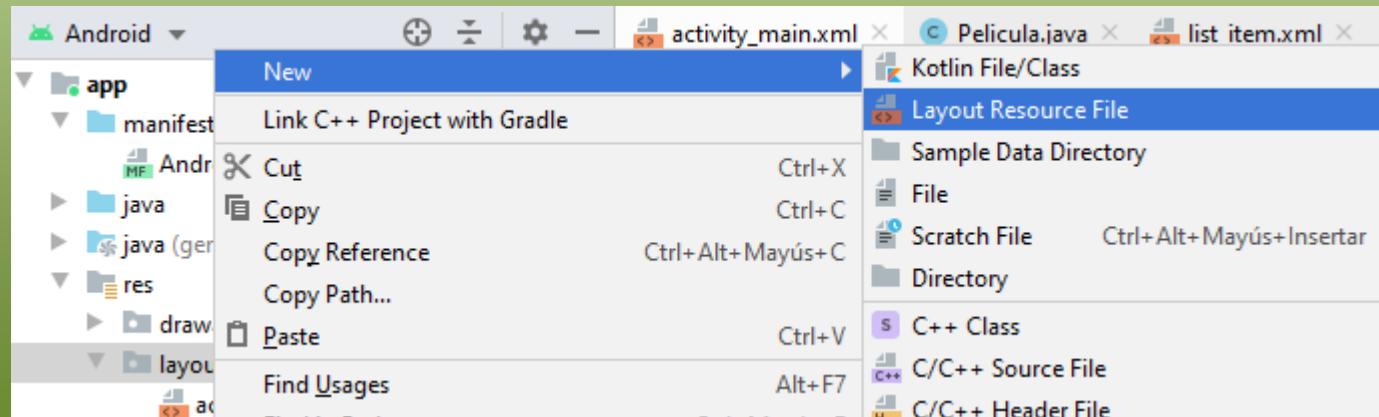
    <ListView
        android:id="@+id/listViewPeliculas"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="50" />

    <WebView
        android:id="@+id/webViewPeliculas"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="50" />
```



2.- Vistas de selección: Listas

- 2.- Creamos el layout de la fila de la lista, para personalizar nuestro ListView. (app -> res -> layout -> click derecho -> New -> Layout Resource File).



2.- Vistas de selección: Listas

- 3.- Diseñamos el layout de la fila de la lista.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:id="@+id/imageViewPeliculas"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:scaleType="fitCenter"
        android:src="@android:drawable/btn_star_big_on" />

    <TextView
        android:id="@+id/textViewTitulo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:textColor="@color/blue"
        android:textSize="@dimen/textSize24dp"
        android:textStyle="bold"
        android:paddingLeft="10dp"
        android:paddingTop="10dp"/>
</LinearLayout>
```



2.- Vistas de selección: Listas

- 4.- Creación de la clase, cuyos objetos serán quienes formen la lista.
- Para crear la clase, app -> java -> com.(...) -> Click derecho -> New -> Kotlin Class/File.

```
class Pelicula (private var imagen: Int, private var titulo: String, private var direccionWeb: String) {  
  
    fun setImagen(img: Int) {  
        imagen = img  
    }  
  
    fun getImagen(): Int {  
        return imagen  
    }  
  
    fun setTitulo(tit: String) {  
        titulo = tit  
    }  
  
    fun getTitulo(): String {  
        return titulo  
    }  
  
    fun setDireccionWeb(dirWeb: String) {  
        direccionWeb = dirWeb  
    }  
  
    fun getDireccionWeb(): String {  
        return direccionWeb  
    }  
}
```

- 5.- Programamos la Activity principal.
- En ella creamos un ArrayList y lo rellenamos de objetos de la clase que creamos en el paso anterior:

```
class MainActivity : AppCompatActivity() {  
  
    private lateinit var listViewPeliculas: ListView  
    private lateinit var webViewPeliculas: WebView  
    private lateinit var arrayListPeliculas: ArrayList<Pelicula>  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        //Referencia de componentes de la Activity  
        listViewPeliculas = findViewById<ListView>(R.id.listViewPeliculas)  
        webViewPeliculas = findViewById<WebView>(R.id.webViewPeliculas)  
  
        //ARRAYLIST  
        arrayListPeliculas = arrayListOf()  
        arrayListPeliculas.add(  
            Pelicula(R.drawable.superman, resources.getString(R.string.superman_text), direccionWeb: "https://www.filmaffinity.com/es/film730631.html"))  
        arrayListPeliculas.add(  
            Pelicula(R.drawable.batman, resources.getString(R.string.batmant_text), direccionWeb: "https://www.filmaffinity.com/es/film943383.html"))  
        arrayListPeliculas.add(  
            Pelicula(R.drawable.capitanamerica, resources.getString(R.string.capitan_america_text), direccionWeb: "https://www.filmaffinity.com/es/film942015.html"))  
        arrayListPeliculas.add(  
            Pelicula(R.drawable.hulk, resources.getString(R.string.hulk_text), direccionWeb: "https://www.filmaffinity.com/es/film182008.html"))  
        arrayListPeliculas.add(  
            Pelicula(R.drawable.ironman, resources.getString(R.string.ironman_text), direccionWeb: "https://www.filmaffinity.com/es/film854941.html"))  
        arrayListPeliculas.add(  
            Pelicula(R.drawable.spiderman, resources.getString(R.string.spiderman_text), direccionWeb: "https://www.filmaffinity.com/es/film218379.html"))  
    }  
}
```

2.- Vistas de selección: Listas

- 6.- Ahora necesitamos un adaptador para que podamos rellenar el ListView con los elementos de nuestro ArrayList.
- Como el ArrayList lo hemos llenado de objetos con varios atributos, no podemos utilizar un ArrayAdapter como en ocasiones anteriores.
- Lo que tenemos que hacer es crear un objeto adaptador de la clase BaseAdapter justo debajo del tratamiento del ArrayList y sobreescibir sus 4 métodos:

2.- Vistas de selección: Listas

- **fun getCount(): Int** -> Retorna el número de elementos del ArrayList.
- **fun getItem (position: Int): Any** -> Retornará el elemento del ArrayList que indique la posición.
- **fun getItemId (position: Int): Long** -> Simplemente nos devolverá la posición, ya que ésta coincide con el Id.
- **fun getView(position: Int, convertView: View?, parent: ViewGroup?)**: En este método recibimos la celda de la lista (convertView). La deberemos inflar (si convertView es null) y luego asignar los atributos del objeto a los elementos de la interfaz.

2.- Vistas de selección: Listas

```
//ADAPTADOR
val baseAdapter = object: BaseAdapter() {
    override fun getCount(): Int {
        return arrayListPeliculas.size
    }

    override fun getItem(position: Int): Any {
        return arrayListPeliculas.get(position)
    }

    override fun getItemId(position: Int): Long {
        return position.toLong()
    }

    override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View {
        //INFLAR LA VISTA (si convertView no es null, utilizamos convertView)
        val view: View
        if(convertView == null) {
            view = layoutInflater.inflate(R.layout.list_view_row_layout, parent, attachToRoot: false)
        } else {
            view = convertView
        }

        //CAPTURAR LOS ELEMENTOS LIST ITEM
        val imageViewPeliculas: ImageView = view.findViewById<ImageView>(R.id.imageViewPeliculas)
        val textViewTitulo: TextView = view.findViewById<TextView>(R.id.textViewTitulo)

        //ACTUALIZAR LOS ELEMENTOS LIST ITEM
        val pelicula: Pelicula = arrayListPeliculas.get(position)
        imageViewPeliculas.setImageResource(pelicula.getImagen())
        textViewTitulo.setText(pelicula.getTitulo())

        return view
    }
}
```

2.- Vistas de selección: Listas

- 7.- Por último no olvides establecerle el adaptador al listView:

```
listViewPeliculas.adapter = baseAdapter
```

2.- Vistas de selección: Listas

- 8.- Sólo nos queda establecer un setOnItemClickListener al listView con el que cargaremos en el WebView la url (con el método **loadUrl**) que contenga el objeto de la lista que hayamos pulsado.

```
listViewPeliculas.setOnItemClickListener(AdapterView.OnItemClickListener { parent, view, position, id ->
    val peliculaSeleccionada: Pelicula = arrayListPeliculas.get(position)

    //Para que se cargue en el webView, no en un navegador a parte
    webViewPeliculas.webViewClient = WebViewClient()
    webViewPeliculas.loadUrl(peliculaSeleccionada.getDireccionWeb())

})
```

2.- Vistas de selección: Listas

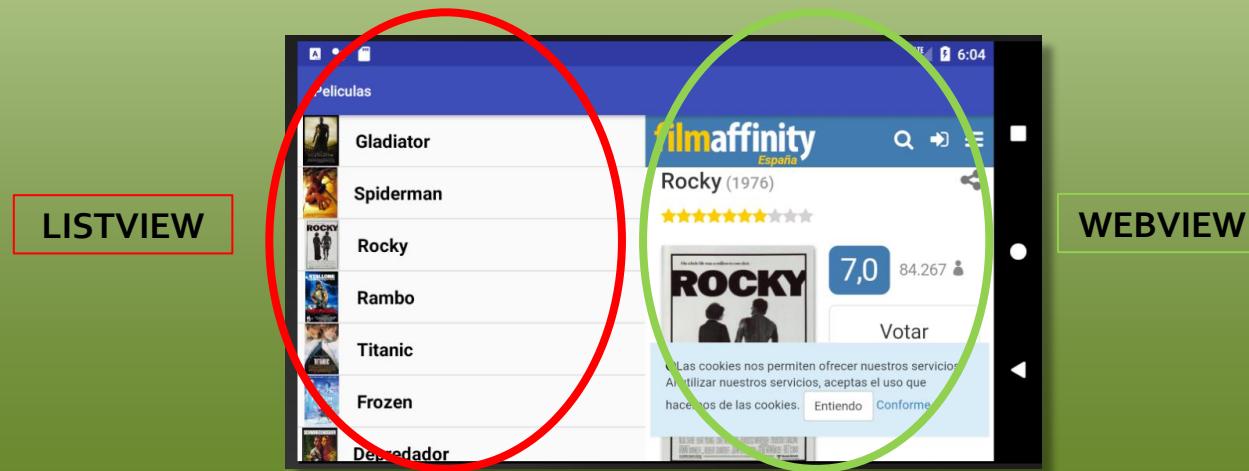
- No olvides darle permiso de INTERNET a la aplicación en el *AndroidManifest.xml*.

```
ty_main.xml × list_view_row_layout.xml × Pelicula.kt × MainActivity.kt × AndroidManifest.xml ×
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.ejercicio5t8">

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

2.- Vistas de selección: Listas

- **Ejercicio 05:** Crea un proyecto en Android Studio formado por un ListView y una WebView:



PMDM

Controles avanzados.



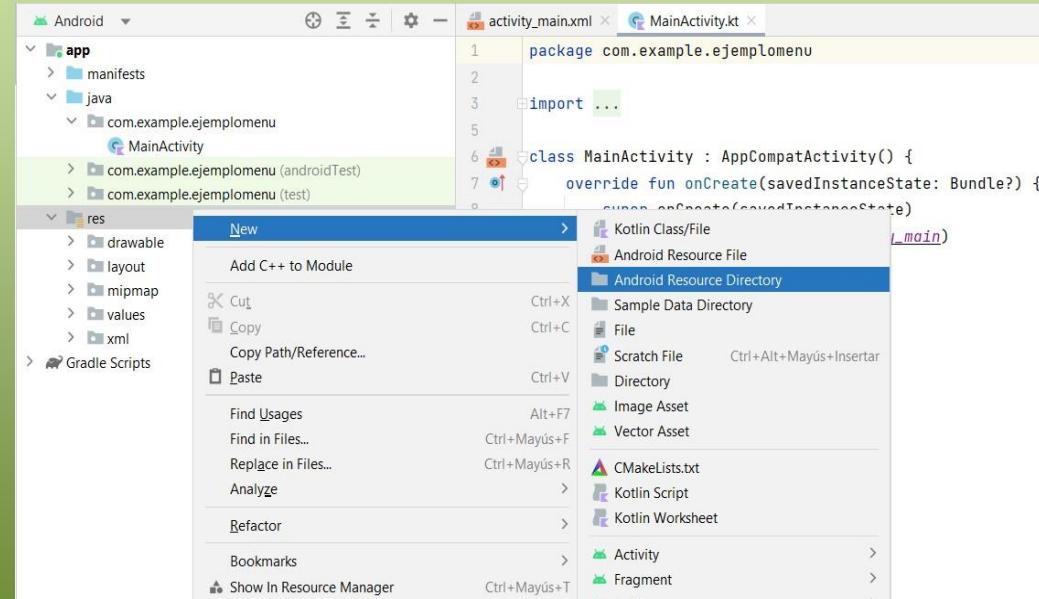
3. Menús

3.- Menús

- Tipos de menús:
 - Menú.
 - ActionBar.
 - Contextual.

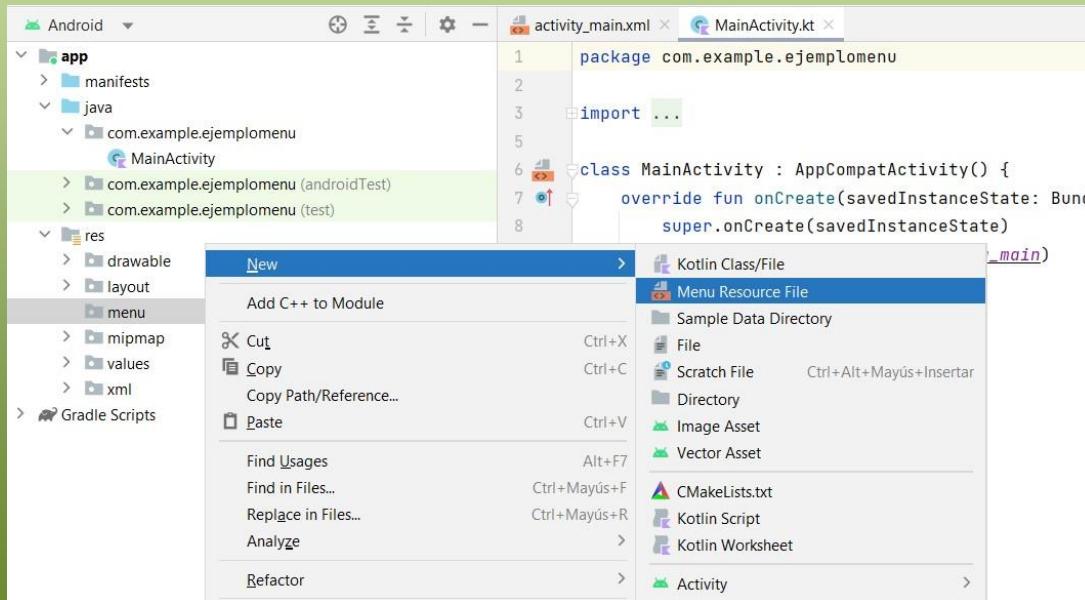
Menú

- Para crear un menú en nuestra aplicación Android comenzaremos creando la carpeta "menu" dentro de la carpeta de recursos (res) de nuestro proyecto:



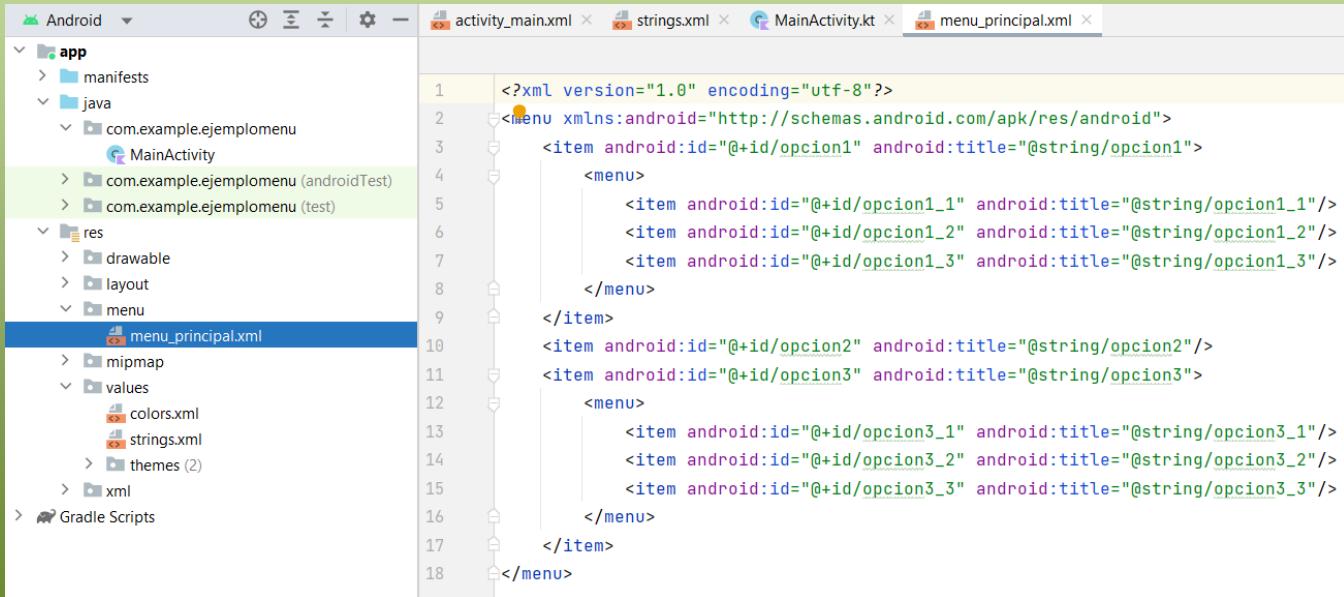
3.- Menús

- Luego, definiremos un archivo de recursos XML dentro de la carpeta “menu” del proyecto:



3.- Menús

- Podemos editar de forma manual nuestro archivo de menu XML:



The screenshot shows the Android Studio interface with the project structure on the left and the XML code for the menu on the right. The project structure includes files like activity_main.xml, strings.xml, MainActivity.kt, and menu_principal.xml. The menu_principal.xml file is selected and shown in the code editor. The XML code defines a menu with three main items, each having a submenu with three items.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/opcion1" android:title="@string/opcion1">
        <menu>
            <item android:id="@+id/opcion1_1" android:title="@string/opcion1_1"/>
            <item android:id="@+id/opcion1_2" android:title="@string/opcion1_2"/>
            <item android:id="@+id/opcion1_3" android:title="@string/opcion1_3"/>
        </menu>
    </item>
    <item android:id="@+id/opcion2" android:title="@string/opcion2"/>
    <item android:id="@+id/opcion3" android:title="@string/opcion3">
        <menu>
            <item android:id="@+id/opcion3_1" android:title="@string/opcion3_1"/>
            <item android:id="@+id/opcion3_2" android:title="@string/opcion3_2"/>
            <item android:id="@+id/opcion3_3" android:title="@string/opcion3_3"/>
        </menu>
    </item>
</menu>
```

3.- Menús

- Con las etiquetas **menú** e **ítem** construiremos nuestros menús.

```
<menu>
```

```
    <item android:id="@+id/opcion1" android:title="Opcion 1"/>
```

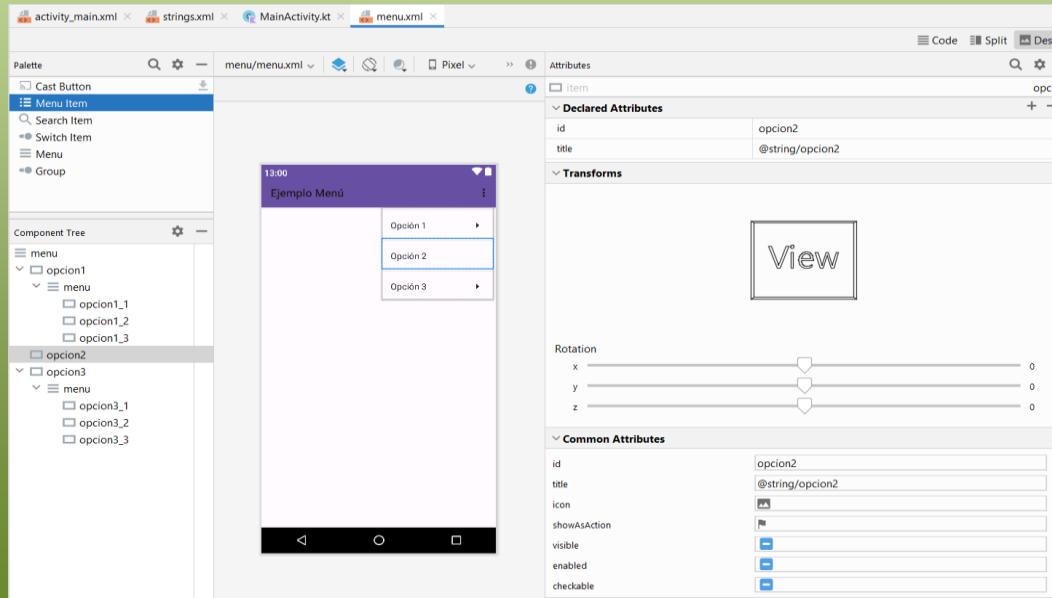
```
    ...
```

```
    </menu>
```

- Si queremos hacer un submenú, bastará con crear un **<menu>** dentro de un **<item>**.

3.- Menús

- También podemos crear nuestro menú de forma automática en vista Diseño:



3.- Menús

- Una vez definido el menú en el fichero XML, tendremos que implementar la función *onCreateOptionsMenu()* de la actividad que queremos que lo muestre.
- En esta función deberemos “inflar” el menú obteniendo una referencia al *inflater* mediante el método *menuInflater (getMenuInflater())* y posteriormente generar la estructura del menú llamando a su método *inflate()*, pasándole como parámetro el ID del menú definido en XML, que en nuestro caso será *R.menu.menu_principal*.
- Por último devolveremos el valor true para confirmar que debe mostrarse el menú.

```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {  
    val inflater: MenuInflater = menuInflater  
    inflater.inflate(R.menu.menu_principal, menu)  
    return true  
}
```

3.- Menús

- Construido el menú, la implementación de cada una de las opciones se incluirá en la función *onOptionsItemSelected()* de la actividad que mostrará el menú. Esta función recibe como parámetro el ítem de menú que ha sido pulsado por el usuario, cuyo ID podemos recuperar con el método *itemId* (*getItemId()*). Según este ID podremos saber qué opción ha sido pulsada y ejecutar unas acciones u otras.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {  
    when(item.itemId) {  
        R.id.opcion1 -> {  
            Toast.makeText(applicationContext, text: "Pulsada opción 1", Toast.LENGTH_SHORT).show()  
            return true  
        }  
        R.id.opcion1_1 -> {  
            Toast.makeText(applicationContext, text: "Pulsada opción 1_1", Toast.LENGTH_SHORT).show()  
            return true  
        }  
        R.id.opcion2 -> {  
            Toast.makeText(applicationContext, text: "Pulsada opción 2", Toast.LENGTH_SHORT).show()  
            return true  
        }  
        R.id.opcion3 -> {  
            Toast.makeText(applicationContext, text: "Pulsada opción 3", Toast.LENGTH_SHORT).show()  
            return true  
        }  
        R.id.opcion3_3 -> {  
            Toast.makeText(applicationContext, text: "Pulsada opción 3_3", Toast.LENGTH_SHORT).show()  
            return true  
        }  
    }  
    else -> return super.onOptionsItemSelected(item)  
}
```

3.- Menús

- Nos quedaría insertar la barra de herramientas donde se mostrará el menú en la actividad que queremos que lo muestre.
- En primer lugar, debemos definir un componente **Toolbar** en el XML de la actividad:

The screenshot shows the Android Studio XML Editor with three tabs at the top: `activity_main.xml`, `MainActivity.kt`, and `menu_principal.xml`. The `activity_main.xml` tab is selected, displaying the following XML code:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbarMainActivity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
        android:theme="?attr/actionBarTheme"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

3.- Menús

- Por último, insertaremos la barra de herramientas en el método `onCreate()` de la actividad, utilizando el método `setSupportActionBar()`:

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    setSupportActionBar(findViewById(R.id.toolbarMainActivity))  
}
```

EJERCICIOS

- **Ejercicio 06:** Crea un proyecto en Android Studio el que crees un Menú con 3 ítems, y en uno de estos ítem crea un submenú con otros 3 items.
● En función de la opción que elija el usuario, un TextView mostrará la opción elegida.

3.- Menús

ActionBar

- Es un menú de iconos.
- Para añadir un ActionBar, bastará con insertar un ícono al ítem que queramos que aparezca en nuestro ActionBar y darle un valor a la propiedad ***showAsAction***. Esta propiedad puede tomar los siguientes valores:
 - ***ifRoom***. Se mostrará en el ActionBar sólo si hay espacio disponible.
 - ***withText***. Se mostrará el texto de la opción junto al ícono en el caso de que éste se esté mostrando como botón de acción (sin ícono si se muestra en el menú de overflow).
 - ***never***. La opción siempre se mostrará como parte del menú de overflow (sin ícono).
 - ***always***. La opción siempre se mostrará como botón de acción. Este valor puede provocar que los elementos se solapen si no hay espacio suficiente para ellos.

3.- Menús

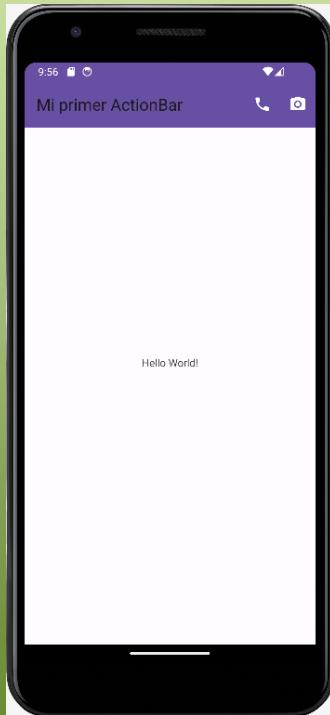
- Ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/opcionTelefono"
        android:icon="@drawable/ic_phone"
        android:title="@string/telefono_text"
        app:showAsAction="ifRoom" />

    <item
        android:id="@+id/opcionCamara"
        android:icon="@drawable/ic_cam"
        android:title="@string/camara_text"
        app:showAsAction="ifRoom" />

</menu>
```



EJERCICIOS

- **Ejercicio 07:** Crea un proyecto en Android Studio en el que crees una Activity con un ActionBar. Este ActionBar tendrá 2 iconos (con la imagen que quieras). Un ítem cambiará el color de fondo de la activity y el otro mostrará un Toast.

3.- Menús

Contextual

- Es el que aparece al dejar el dedo pulsado sobre algún elemento de la interfaz.
- Para crearlo, lo primero que tenemos que hacer es registrar a qué vista (por ejemplo a un textView) le vamos a dar un menú, con una llamada al método *registerForContextMenu()*:

```
private lateinit var textViewContextual: TextView

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    textViewContextual = findViewById<TextView>(R.id.textViewContextual)
    registerForContextMenu(textViewContextual)
}
```

3.- Menús

- Luego crearemos un menú en Android definiendo un archivo de recursos XML dentro de la carpeta “menu” del proyecto (exactamente igual que como hacíamos con los menús normales)

The screenshot shows the Android Studio interface. On the left, the project structure is visible under the 'app' folder, including 'manifests', 'java' (with a file 'MainActivity'), and 'res' (containing 'drawable', 'layout', 'menu', and 'mipmap'). The 'menu' folder contains a file named 'menu_contextual.xml'. In the main editor area, the code for this XML file is being typed:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/opcionRojo"
          android:title="@string/rojo_text"/>
    <item android:id="@+id/opcionVerde"
          android:title="@string/verde_text"/>
</menu>
```

3.- Menús

- Después hay que sobrescribir el método `onCreateContextMenu()` e inflar el menú que hayas creado en el XML.

```
override fun onCreateContextMenu(menu: ContextMenu?, v: View?, menuInfo: ContextMenu.ContextMenuItem?) {  
    super.onCreateContextMenu(menu, v, menuInfo)  
    val inflater: MenuInflater = menuInflater  
    inflater.inflate(R.menu.menu_contextual, menu)  
}
```

3.- Menús

- Por último, para responder a la selección del elemento del menú contextual, se programa el método `onContextItemSelected()`.

```
override fun onContextItemSelected(item: MenuItem): Boolean {
    when(item.itemId) {
        R.id.opcionRojo -> {
            textViewContextual.setText(R.string.rojo_text)
            return true
        }
        R.id.opcionVerde -> {
            textViewContextual.setText(R.string.verde_text)
            return true
        }
        else -> return super.onContextItemSelected(item)
    }
}
```

EJERCICIOS

- **Ejercicio 08:** Crea un proyecto en Android Studio en el que crees una Activity con un TextView en el centro. Este TextView constará de un menú contextual con dos opciones. Al pulsar estas dos opciones, el texto del TextView cambiará.

PMDM

Controles avanzados.

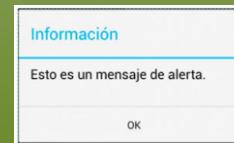
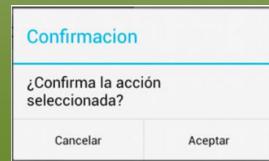


4. Cuadros de diálogo

4.- Cuadros de diálogo

- Los diálogos son ventanas que aparecen en la pantalla de forma espontánea para mostrar información al usuario sobre alguna operación, o preguntarle sobre alguna decisión o configuración que deba tomar para poder proceder con el funcionamiento normal de la aplicación.
- Hay 4 tipos de cuadros de diálogos y todos utilizan la clase Dialog como clase base:

- AlertDialog:

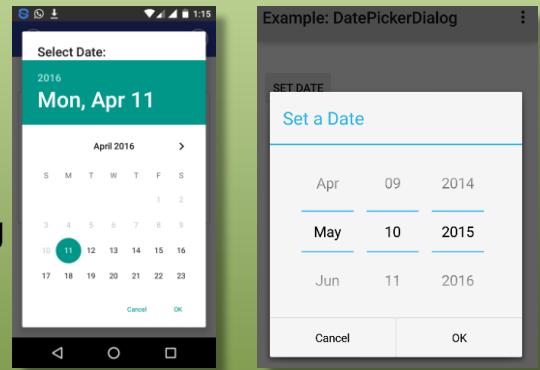


4.- Cuadros de diálogo

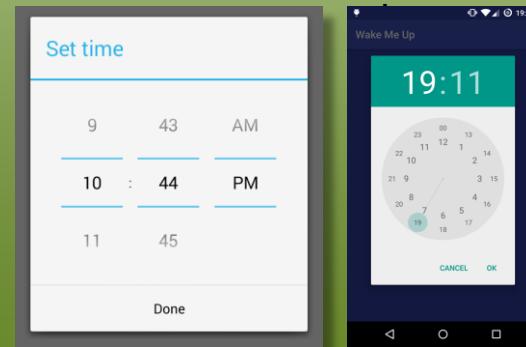
- ProgressDialog:



- DatePickerDialog



- TimePickerDialog



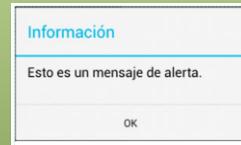
4.- Cuadros de diálogo

- Los **AlertDialog** pueden aparecer con solo texto, con listas de elementos, con RadioButtons, con Checkboxes... e incluso se puede utilizar un fichero XML para definir un Layout y diseñar un diálogo personalizado.
- Para crear un AlertDialog, usaremos *AlertDialog.Builder* y configuraremos propiedades como:
 - `setTitle()`: asigna título
 - `setMessage()`: asigna mensaje
 - `setPositiveButton()`, `setNegativeButton()`, `setNeutralButton()`: botones a los que se les asigna un texto y un Listener a ejecutar cuando el botón es pulsado.

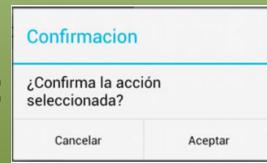
4.- Cuadros de diálogo

- Los diferentes tipos de AlertDialog que podemos crear son:

- De Información:



- De Confirmación:



4.- Cuadros de diálogo

- De Selección:

Selección
Español
Inglés
Francés

- De Selección con RadioButtons:

Selección
Español <input type="radio"/>
Inglés <input checked="" type="radio"/>
Francés <input type="radio"/>

- De Selección con CheckBoxes:

Selección
Español <input checked="" type="checkbox"/>
Inglés <input checked="" type="checkbox"/>
Francés <input type="checkbox"/>

- Personalizados:

Dialogo Linea 1
Dialogo Linea 2

Aceptar

4.- Cuadros de diálogo

- Para crear un diálogo de Alerta de cualquier tipo, lo primero que haremos será crear una objeto de la clase *AlertDialog.Builder*:
- **val builder: AlertDialog.Builder = AlertDialog.Builder(this)**

4.- Cuadros de diálogo

Cuadro de Diálogo de Información

- Este tipo de diálogo se limita a mostrar un mensaje sencillo al usuario, y un único botón de OK para confirmar su lectura.
- Para su implementación basta con crear un objeto de tipo *AlertDialog.Builder* y establecer las propiedades del diálogo mediante sus métodos correspondientes:
 - **Título:** *setTitle("")*.
 - **Mensaje:** *setMessage("")*.
 - **Texto y comportamiento del botón:** *setPositiveButton()*.

4.- Cuadros de diálogo

- Al método *setPositiveButton* le tendremos que pasar dos argumentos:
 - Un String, que aparecerá en el cuadro de diálogo.
 - Una expresión lambda que implementa la interfaz *DialogInterface.OnClickListener*. Aquí haremos las acciones oportunas del evento *onClick* del botón, como por ejemplo *dialog.cancel()* para que se cierre el cuadro de diálogo.
- Por último llamaremos al método *show()* desde nuestro builder.

4.- Cuadros de diálogo

```
private fun crearCuadroDialogo() {  
    // Creamos un constructor de AlertDialog  
    val builder: AlertDialog.Builder = AlertDialog.Builder(context: this)  
  
    // Configuramos el título y el mensaje del AlertDialog  
    builder.setTitle("¡Información!")  
    builder.setMessage("Mi primer cuadro de diálogo")  
  
    // Configura el botón "Aceptar"  
    builder.setPositiveButton(text: "CERRAR INFO") { dialog, which ->  
        Toast.makeText(applicationContext, text: "Cerrando Información", Toast.LENGTH_SHORT).show()  
        dialog.cancel() // Cierra el AlertDialog  
    }  
  
    // Mostramos el AlertDialog  
    builder.show()  
}
```

4.- Cuadros de diálogo

Cuadro de Diálogo de Confirmación

- La implementación de estos diálogos será prácticamente igual a la ya comentada para el cuadro de Información, salvo que en esta ocasión añadiremos dos botones, uno de ellos para la respuesta afirmativa (`setPositiveButton()`), y el segundo para la respuesta negativa (`setNegativeButton()`).

4.- Cuadros de diálogo

- Tanto al método `setPositiveButton` como el `setNegativeButton` les tendremos que pasar dos argumentos:
 - Un String, que aparecerá en el cuadro de diálogo.
 - Una expresión lambda que implementa la interfaz `DialogInterface.OnClickListener`. Aquí haremos las acciones oportunas del evento `onClick` de los botones. En este caso mostraremos un Toast y estableceremos un texto en un TextView.
- Por último llamaremos al método `show()` desde nuestro builder.

4.- Cuadros de diálogo

```
private fun crearCuadroDialogoConfirmacion() {
    val builder: AlertDialog.Builder = AlertDialog.Builder(context: this)

    builder.setTitle("Confirmación")
    builder.setMessage("¿Eres mayor de edad?")

    builder.setPositiveButton( text: "SI") { dialog, which ->
        Toast.makeText(applicationContext, text: "¡¡¡SI!!!", Toast.LENGTH_SHORT).show()
        textViewRespuesta.setText("Has dicho que sí")
    }

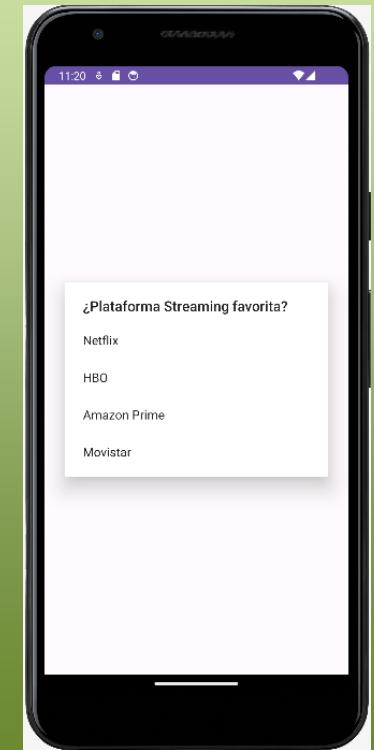
    builder.setNegativeButton( text: "NO") { dialog, which ->
        Toast.makeText(applicationContext, text: "¡¡¡NO!!!", Toast.LENGTH_SHORT).show()
        textViewRespuesta.setText("Has dicho que no")
    }

    builder.show()
}
```

4.- Cuadros de diálogo

Cuadro de Diálogo de Selección

- Utilizando la clase *AlertDialog.Builder* le indicaremos la lista de opciones a mostrar (mediante el método *setItems()*) y proporcionaremos la expresión lambda que implementa la interfaz *DialogInterface.OnClickListener*. Aquí realizaremos las acciones oportunas del evento *onClick()* según la opción elegida de la lista de selección.
- La expresión lambda recibe en *which* el índice de la opción seleccionada.
- La lista de opciones la definiremos como un *array de Strings*.



```
private fun crearCuadroDialogoSeleccion() {
    //LISTADO
    val arrayPlataformas: Array<String> = arrayOf("Netflix", "HBO", "Amazon Prime", "Movistar")

    //ALERTDIALOG
    val builder: AlertDialog.Builder = AlertDialog.Builder(context: this)
    builder.setTitle("¿Plataforma Streaming favorita?")
    builder.setItems(arrayPlataformas) { dialog, which ->
        val opcSeleccionada: String = arrayPlataformas[which]
        Toast.makeText(applicationContext, text: "Has elegido: $opcSeleccionada", Toast.LENGTH_SHORT).show()
    }
    builder.show()
}
```

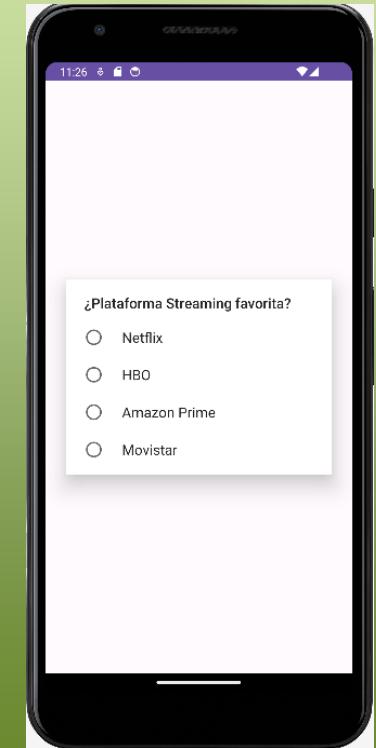
4.- Cuadros de diálogo

Cuadro de Diálogo de Selección con RadioButtons

- Lo único que cambia respecto al anterior es que utilizaremos el método *setSingleChoiceItems()* en vez de *setItems()*. Este método tan sólo se diferencia de *setItems()* en que recibe un segundo parámetro adicional, que indica el índice de la opción marcada por defecto. Si no queremos tener ninguna de ellas marcadas inicialmente pasaremos el valor **-1**.

4.- Cuadros de diálogo

```
private fun crearCuadroDialogSelecRadio() {  
    //LISTADO  
    val arrayPlataformas: Array<String> = arrayOf("Netflix", "HBO", "Amazon Prime", "Movistar")  
  
    //ALERTDIALOG  
    val builder: AlertDialog.Builder = AlertDialog.Builder(context: this)  
    builder.setTitle("¿Plataforma Streaming favorita?")  
    builder.setSingleChoiceItems(arrayPlataformas, checkedItem: -1) { dialog, which ->  
        val opcSeleccionada: String = arrayPlataformas[which]  
        Toast.makeText(applicationContext, text: "Has elegido: $opcSeleccionada", Toast.LENGTH_SHORT).show()  
    }  
    builder.show()  
}
```



4.- Cuadros de diálogo

Cuadro de Diálogo de Selección con Check Boxes

- Lo único que cambia, respecto a los anteriores, es que utilizaremos el método ***setMultipleChoiceItems()*** en vez de ***setItems()***. Este método tan sólo se diferencia de ***setItems()*** en que recibe un segundo parámetro adicional, un ***array de booleanos (BooleanArray)***, para establecer las opciones marcadas por defecto. En caso de no querer ninguna opción seleccionada por defecto pasaremos el valor null.
- También tendremos que implementar un listener del tipo ***DialogInterface.OnMultiChoiceClickListener***. En este caso, en la expresión lambda mediante la cual implementamos el evento ***onClick***, recibiremos tanto la opción seleccionada (***which***) como el estado en el que ha quedado (***isChecked***).

4.- Cuadros de diálogo

```
private fun crearCuadroDialogSelcCheckBox() {
    //LISTADO
    val arrayPlataformas: Array<String> = arrayOf("Netflix", "HBO", "Amazon Prime", "Movistar")

    //ALERTDIALOG
    val builder: AlertDialog.Builder = AlertDialog.Builder( context: this)
    builder.setTitle("¿Plataforma Streaming favorita?")

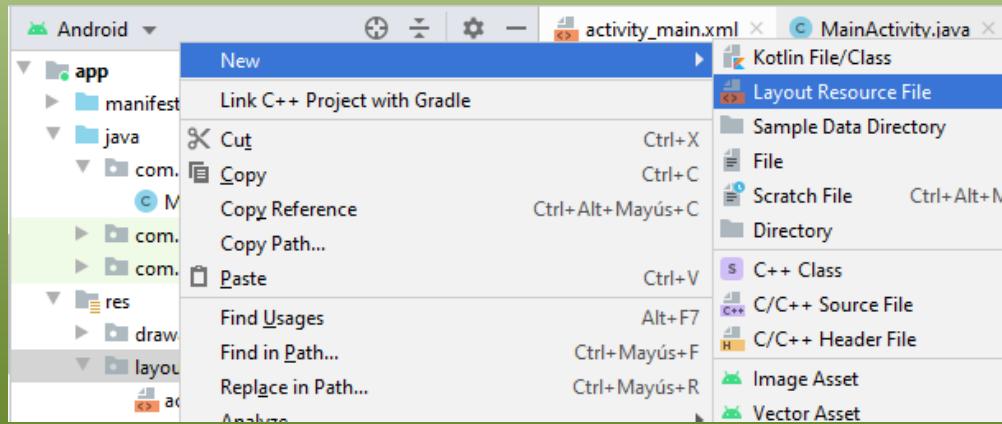
    //LISTADO DE OPCIONES MARCADAS POR DEFECTO
    val arrayMarcadas: BooleanArray = booleanArrayOf(false, true, true, false)

    builder.setMultiChoiceItems(arrayPlataformas, arrayMarcadas) { dialog, which, isChecked ->
        if(isChecked){
            arrayMarcadas[which] = true
            Toast.makeText(applicationContext, text: "Has seleccionado: ${arrayPlataformas[which]}",
                Toast.LENGTH_SHORT).show()
        } else {
            arrayMarcadas[which] = false
            Toast.makeText(applicationContext, text: "Has deseleccionado: ${arrayPlataformas[which]}",
                Toast.LENGTH_SHORT).show()
        }
    }
    builder.show()
}
```

4.- Cuadros de diálogo

Cuadro de Diálogo Personalizado

- Lo primero que tendremos que hacer es crear un nuevo layout con los elementos a mostrar en el diálogo.



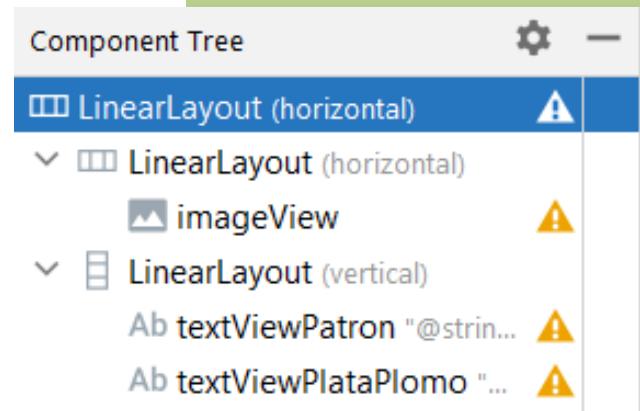
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="30">

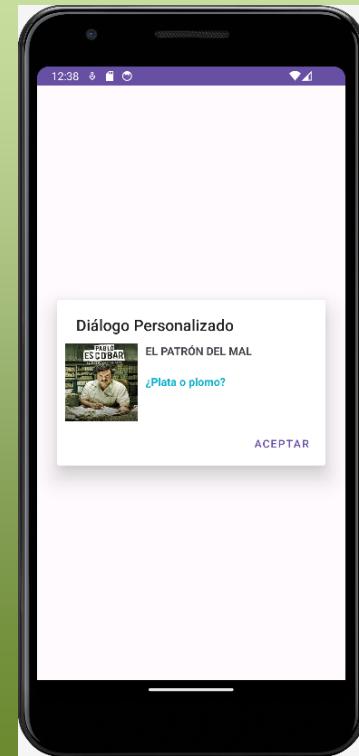
        <ImageView
            android:id="@+id/imageView"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:scaleType="fitXY"
            android:layout_marginTop="10dp"
            android:layout_marginLeft="10dp"
            android:src="@drawable/escobar" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="70"
        android:orientation="vertical">
```



4.- Cuadros de diálogo

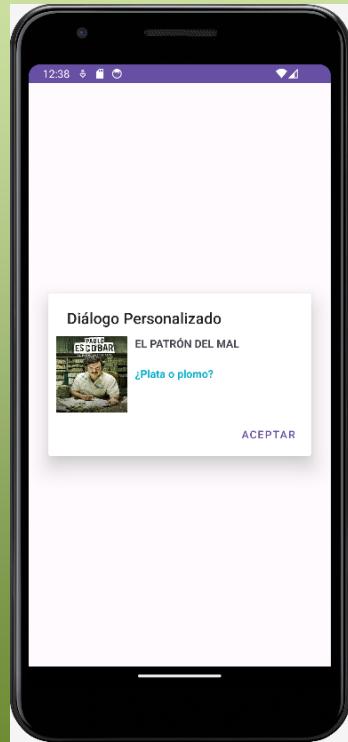


4.- Cuadros de diálogo

- A continuación, procederemos a inflar el cuadro de diálogo desde el código:
 - **val inflater: LayoutInflator = layoutInflater**
- Y luego utilizaremos el método **setView**:
 - **builder.setView(inflater.inflate(R.layout.dialogo_personalizado, null))**
- Finalmente podremos incluir botones tal como vimos para los diálogos de selección y confirmación. En este ejemplo sólo incluiremos un botón de **Aceptar**:

4.- Cuadros de diálogo

```
private fun mostrarDialogPersonalizado() {  
    val builder: AlertDialog.Builder = AlertDialog.Builder( context: this)  
  
    builder.setTitle("Diálogo Personalizado")  
  
    val inflater: LayoutInflator = layoutInflater  
    builder.setView(inflater.inflate(R.layout.alert_dialog_layout, root: null))  
  
    builder.setPositiveButton( text: "ACEPTAR") { dialog, which ->  
        dialog.cancel()  
    }  
  
    builder.show()  
}
```



4.- Cuadros de diálogo

- **Ejercicio 09:** Crea un proyecto en Android Studio que utilice un diálogo personalizado como el del ejemplo anterior.

PMDM

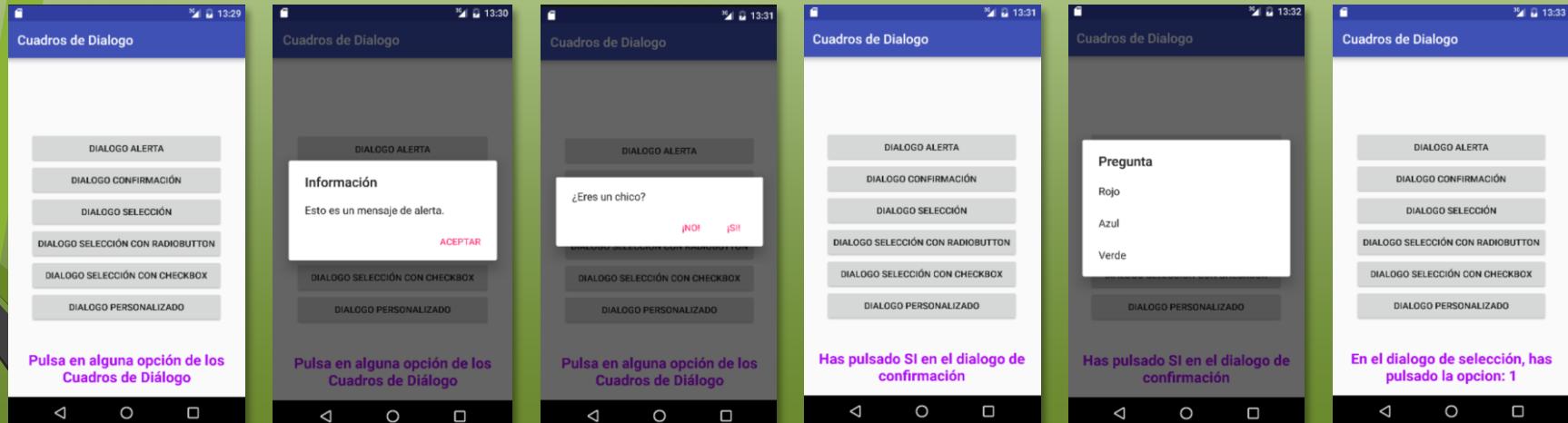
Controles avanzados.

5. Ejercicios de consolidación

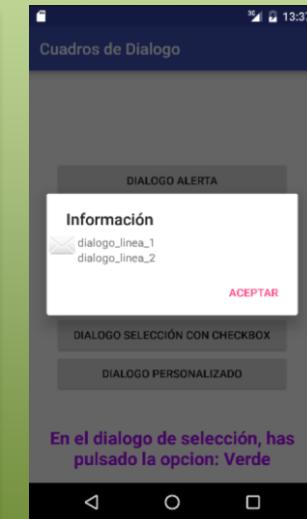
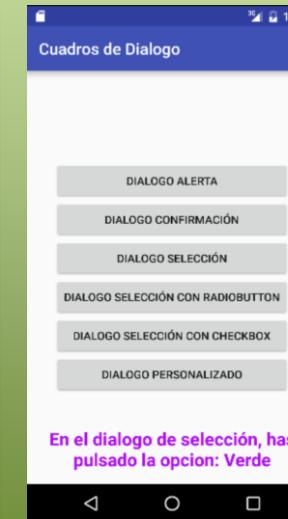
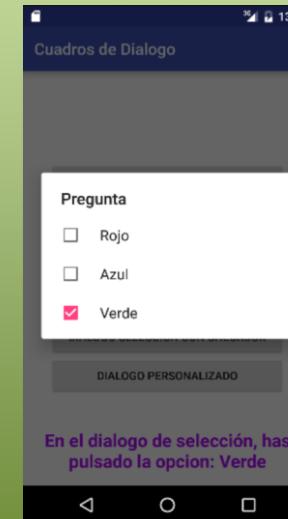
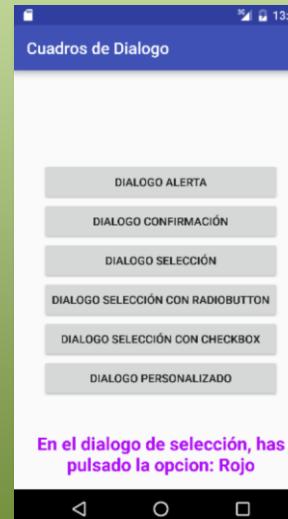
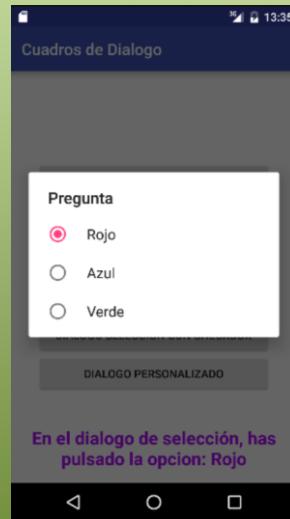


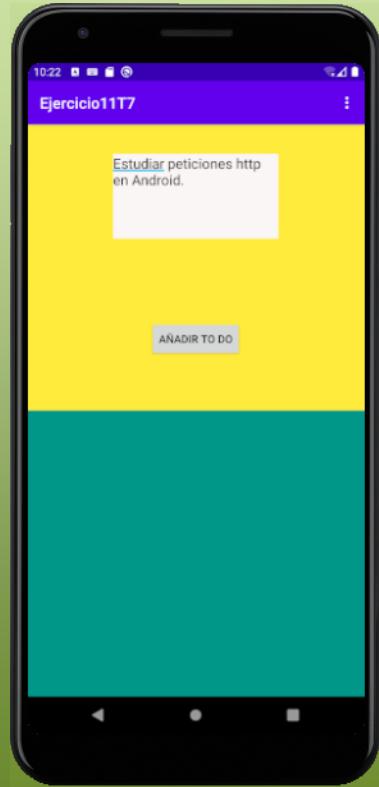
EJERCICIOS

- **Ejercicio 10:** Crea una actividad que contenga 6 botones. Cada botón llamará a un cuadro de diálogo de alerta distinto de los 6 tipos existentes:

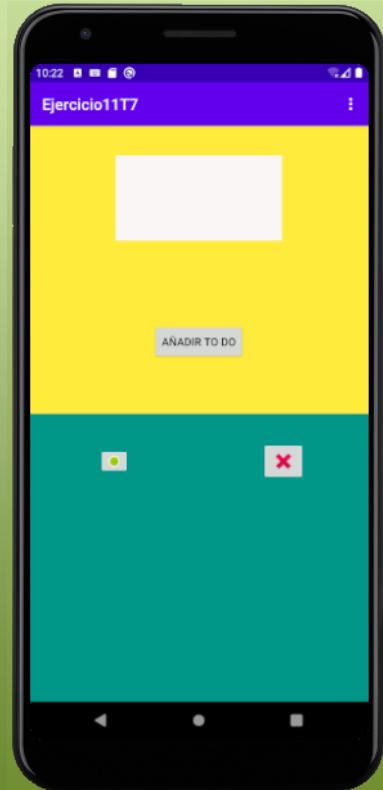


EJERCICIOS



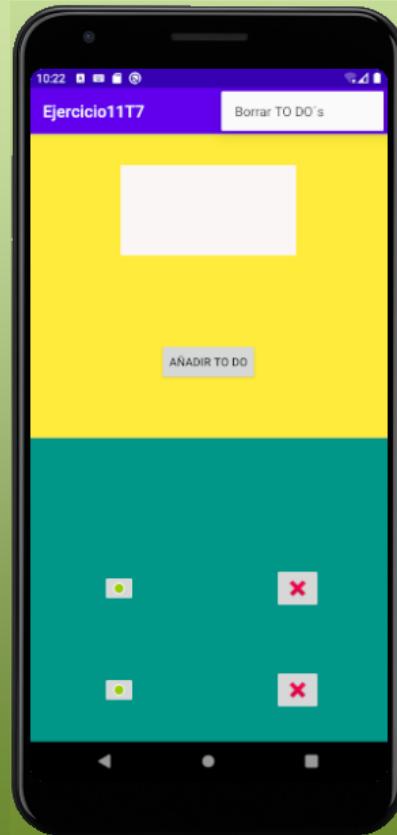


- **Ejercicio 11:** Crea una app con capacidad para gestionar hasta 3 “TODO’s” (Tareas para hacer). Se almacenarán en un array de Strings.
- La app estará dividida en un layout superior y en otro inferior, ambos de igual tamaño.
- El EditText superior, donde el usuario debe escribir el texto de cada To Do, será de tipo “*Multiline text*” con su propiedad *line* a 5.



- Cuando el usuario pulse el botón “Añadir To Do”, se mostrará un Toast (“To Do guardado correctamente”) y, en el layout inferior, se mostrarán dos botones asociados a ese To Do.

- Los dos botones asociados a cada *To Do* son:
 - **Botón mostrar:** Al pulsar en el botón de la izquierda, se mostrará un cuadro de diálogo con el texto del To Do.
 - **Botón eliminar:** Al pulsar el botón, se eliminará el To Do de nuestra aplicación y, automáticamente, desaparecerán los dos botones asociados al To Do eliminado.
- El To Do se insertará en el primer hueco libre que haya en el array.



- La aplicación también dispondrá de un menú con una única opción: *Borrar To Do's*.
- Al pulsar esta opción, se borrarán todos los To Do's que tengamos almacenados en ese momento y se ocultarán los botones asociados a todos los To Do's..