CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES 24/25

CFGS DAM

UD 05

DISEÑO DE LA INTERFAZ DE USUARIO II

CONCEPTOS SOBRE ELEMENTOS DE CREACIÓN DE APLICACIONES

Autor: Mara Vañó

m.vanoalonso@edu.gva.es

Fecha: 2024/2025

Índice

ÍNDICE

1. Objetivos de la unidad

- Conocer los elementos que forman un layout y sus clases.
- Conocer los diferentes tipos de fuentes en Android.
- Conocer la disposición y los atributos de los diferentes elementos.
- Aprender a seleccionar un elemento de todos los elementos disponibles.

2. Introducción

En esta unidad formativa vamos a ver los elementos que conforman un layout completo.

2.1 Fuentes

Por defecto, Android dispone de 3 fuentes instaladas: una tipo sans, una tipo monospace y una tipo serif, (Droid Sans, Droid Sans Mono y Droid Serif).

Para utilizar otro tipo de fuente, esta deberá ser cargada, sin embargo, recuerda que los ficheros de tipo fuentes son pesados, lo que afecta a los dispositivos de **recursos limitados**.

Además, existen fuentes que requieren **licencia** para su uso, por lo que deberemos tener cuidado.

Para importar una fuente:

- Haz clic con el botón derecho en la carpeta res y ve a New > Android resource directory.
- Aparece la ventana New Resource Directory.
- En la lista Resource type, selecciona font y, luego, haz clic en OK.
- Importa el archivo «.ttf».

Enlace como implementar fuentes:

https://developer.android.com/guide/topics/ui/look-and-feel/fonts-in-xml?hl=es-419

https://developer.android.com/reference/android/graphics/Typeface

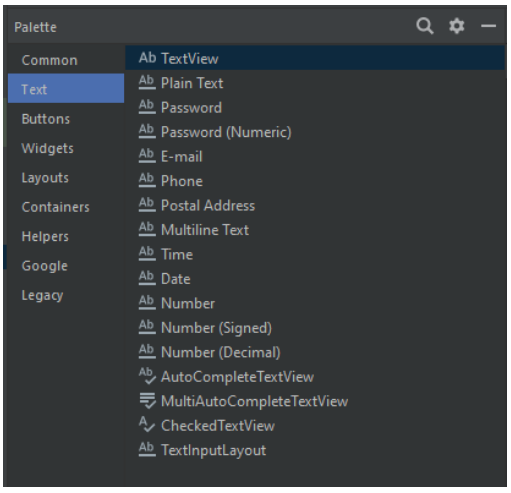
En el xml:

```
<?xml version="1.0" encoding="utf-8"?>
<font-family xmlns:android="http://schemas.android.com/apk/res/android">
  <font
    android:fontStyle="normal"
    android:fontWeight="400"
    android:font="@font/lobster_regular" />
  <font
    android:fontStyle="italic"
    android:fontWeight="400"
    android:font="@font/lobster_italic" />
</font-family>
```

Enlace a recursos de fuentes:
https://fonts.google.com/knowledge

3. Clases de tipo Text

Las clases TextView, TextEdit y Button son las más utilizadas y por ello ha sido necesario utilizarlas desde el principio. Recuerda que hay múltiples propiedades de cada una de las clases que se usan para cambiar el color, el estilo o el tamaño de la fuente.



Importante sobre la clases EditText: es una subclase de *TextView* y tiene las siguientes propiedades (entre otras):

- podemos añadir autotexto
- podemos especificar si queremos sólo una línea
- podemos especificar diferentes tipos de teclado mediante android:inputType Aquí tienes más información sobre EditText: EditText

Enlace como generar un diseño:
https://developer.android.com/codelabs/basic-android-kotlin-training-xml-layouts?hl=es-419#0

4. Clase Imagen e ImagenButton

Son las clases TextView y Button que hemos visto pero incluyendo imágenes. En este caso cada widget tomará un atributo tipo android:src en xlm para especificar qué imagen usará, referenciándola con drawable.

IMPORTANTE: Antes de añadir un ImageView o ImageButton debemos añadir:
src->main->res->drawable donde añadiremos las imágenes que deseemos en **minúsculas** (Habrá tantas carpetas como tamaños de pantalla deseados)

Los ficheros aceptados son:
- png: este es el recomendado
- jpg
- bmp Cada imagen tendrá un identificador que será accesible mediante R.drawable.identificador

Podemos hacerlo desde la parte visual en el xml o si preferimos hacerlo con código:

```
<ImageButton
    android:id="@+id/imageButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.451"
    app:layout_constraintStart_toStartOf="parent"
    app:srcCompat="@drawable/ic_launcher_foreground"  />
```

Enlace como agregar imágenes a tu app para Android

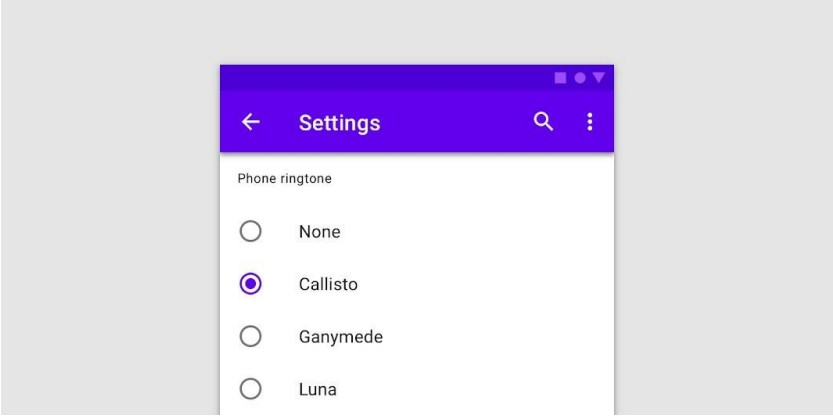
https://developer.android.com/codelabs/basic-android-kotlin-compose-add-images?hl=es-419#0

5. Clase RadioGroup y RadioButton

Son las clases que se utilizan para permitir al usuario hacer selecciones de elementos. La clase RadioGroup es en realidad un contenedor (un conjunto de RadioButton), y por tanto se ubicaran en lugares diferentes.

Para crear cada opción del botón de selección, crea un RadioButton en tu diseño. Sin embargo, debido a que los botones de selección son mutuamente excluyentes, debes agruparlos dentro de un RadioGroup. Cuando los agrupas, el sistema garantiza que solo se pueda seleccionar un botón a la vez.

RadioButton:



En el XML:

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:orientation="vertical">
   <RadioButton android:id="@+id/radio_pirates"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/pirates"
      android:onClick="onRadioButtonClicked"/>
   <RadioButton android:id="@+id/radio_ninjas"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/ninjas"
      android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

En el Activity:

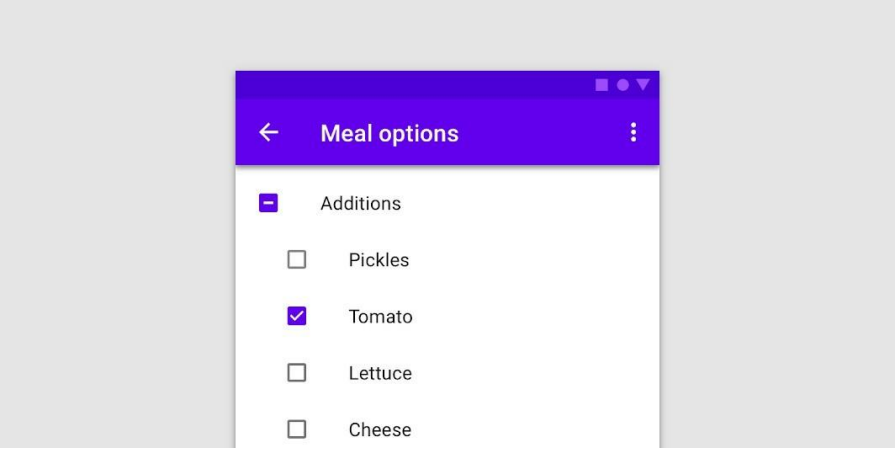
```
fun onRadioButtonClicked(view: View) {
    if (view is RadioButton) {
        // Is the button now checked?
        val checked = view.isChecked
        // Check which radio button was clicked
        when (view.getId()) {
            R.id.radio_pirates ->
                if (checked) {
                    // Pirates are the best
                }
            R.id.radio_ninjas ->
                if (checked) {
                    // Ninjas rule
                }
        }
    }
}
```

Enlace botones de selección:

https://developer.android.com/guide/topics/ui/controls/radiobutton?hl=es-419

6. Clase Checkbox

Las casillas de verificación permiten que el usuario seleccione una o más opciones de un conjunto. Por lo general, debes presentar cada opción de casilla de verificación en una lista vertical.



Para crear cada opción, crea un objeto CheckBox en tu diseño. Dado que un conjunto de opciones de casilla de verificación le permite al usuario seleccionar varios elementos, se administra cada casilla por separado y debes registrar un objeto de escucha de clics para cada una.

Cuando el usuario selecciona una casilla de verificación, el objeto CheckBox recibe un evento de clic. A fin de definir el controlador de eventos de clic para una casilla de verificación, agrega el atributo android:onClick al elemento <CheckBox> en tu diseño XML. El valor de este atributo debe ser el nombre del método al que deseas llamar en respuesta a un evento de clic. La Activity que aloja el diseño debe implementar el método correspondiente.

Por ejemplo, estos son un par de objetos CheckBox en una lista:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat"
        android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkbox_cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
        android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

En la Activity que aloja este diseño, el siguiente método procesa el evento de clic para ambas casillas de verificación:

```
fun onCheckboxClicked(view: View) {
    if (view is CheckBox) {
        val checked: Boolean = view.isChecked
        when (view.id) {
            R.id.checkbox_meat -> {
                if (checked) {
                    // Put some meat on the sandwich
                } else {
                    // Remove the meat
                }
            }
            R.id.checkbox_cheese -> {
                if (checked) {
                    // Cheese me
                } else {
                    // I'm lactose intolerant
                }
            }
            // TODO: Veggie sandwich
        }
    }
}
```

Enlace para como implementar casillas de verificación:

https://developer.android.com/guide/topics/ui/controls/checkbox?hl=es-419

7. Clase ToggleButton y Switch

Un botón de activación permite al usuario cambiar un ajuste entre dos estados.

Puedes agregar un botón de activación básico a tu diseño con el objeto ToggleButton.

Android 4.0 (nivel de API 14) presenta otro tipo de botón de activación, llamado interruptor. Este proporciona un control deslizante que puedes agregar con un objeto Switch. SwitchCompat es una versión del widget Interruptor que se ejecuta en dispositivos que usan hasta el nivel de API 7.

Botones de activación

Interruptores (en Android 4.0 y versiones posteriores)

Para detectar cuándo el usuario activa un botón o un interruptor, crea un objeto **CompoundButton.OnCheckedChangeListener** y asígnalo al botón llamando a **setOnCheckedChangeListener()**.

Por ejemplo:

```
val toggle: ToggleButton = findViewById(R.id.togglebutton)
toggle.setOnCheckedChangeListener { _, isChecked ->
    if (isChecked) {
        // The toggle is enabled
    } else {
        // The toggle is disabled
    }
}
```

Enlace como implementar botones de actividad:

https://developer.android.com/guide/topics/ui/controls/togglebutton?hl=es-419

8. Clase AutoCompleteTextView

Una AutoCompleteTextView o vista de texto editable muestra sugerencias de finalización automáticamente mientras el usuario escribe. La lista de sugerencias se muestra en un menú desplegable en el que el usuario puede elegir un elemento para reemplazar el contenido del cuadro de edición.

El menú desplegable se puede descartar en cualquier momento presionando la tecla Atrás o, si no se selecciona ningún elemento en el menú desplegable, presionando la tecla central enter/dpad.

La lista de sugerencias se obtiene de un adaptador de datos y aparece solo después de un número determinado de caracteres definido por el umbral.

El siguiente fragmento de código muestra cómo crear una vista de texto que sugiere varios nombres de países mientras el usuario escribe:

```
public class CountriesActivity extends Activity {
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.countries);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_dropdown_item_1line, COUNTRIES);
        AutoCompleteTextView textView = (AutoCompleteTextView)
                findViewById(R.id.countries_list);
        textView.setAdapter(adapter);
    }
    private static final String[] COUNTRIES = new String[] {
        "Belgium", "France", "Italy", "Germany", "Spain"
    };
}
```

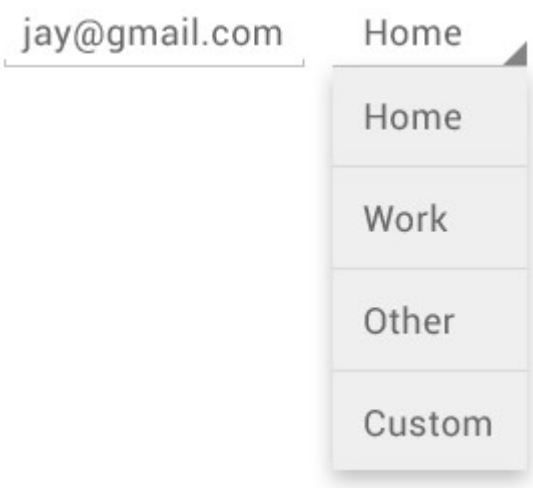
Enlace como implementar AutoCompleteTextView:

https://developer.android.com/reference/android/widget/AutoCompleteTextView

9. Spinner

Sería el equivalente a “ComboBox” en otros lenguajes y su misión es la de mostrar una lista desplegable para seleccionar un único elemento.

Los Spinners proporcionan una forma rápida de seleccionar un valor de un conjunto. En el estado predeterminado, una rueda muestra su valor actualmente seleccionado. Al tocar la rueda de información, se muestra un menú desplegable con todos los demás valores disponibles, desde el cual el usuario puede seleccionar uno nuevo.



Puedes agregar una spinners a tu diseño con el Spinnerobjeto. Por lo general, debe hacerlo en su diseño XML con un <Spinner>elemento. Por ejemplo:

```
<Spinner
   android:id="@+id/planets_spinner"
   android:layout_width="match_parent"
   android:layout_height="wrap_content" />
```

Las opciones deben estar disponibles de manera predeterminada, y se pueden proporcionar mediante una matriz de cadenas definida en un **archivo de recursos de cadena** :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string-array name="planets_array">
      <item>Mercury</item>
      <item>Venus</item>
      <item>Earth</item>
      <item>Mars</item>
      <item>Jupiter</item>
      <item>Saturn</item>
      <item>Uranus</item>
      <item>Neptune</item>
   </string-array>
</resources>
```

En Kotlin:

```
val spinner: Spinner = findViewById(R.id.spinner)
// Create an ArrayAdapter using the string array and a default spinner layout
ArrayAdapter.createFromResource(
        this,
        R.array.planets_array,
        android.R.layout.simple_spinner_item
).also { adapter ->
    // Specify the layout to use when the list of choices appears
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
    // Apply the adapter to the spinner
    spinner.adapter = adapter
}


class SpinnerActivity : Activity(), AdapterView.OnItemSelectedListener {
    override fun onItemSelected(parent: AdapterView<*>, view: View?, pos: Int, id: Long) {
        // An item was selected. You can retrieve the selected item using
        // parent.getItemAtPosition(pos)
    }
    override fun onNothingSelected(parent: AdapterView<*>) {
        // Another interface callback
    }
}
```

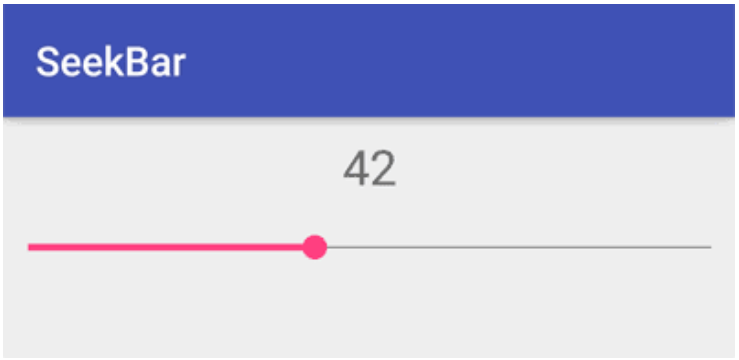
Enlace como implementar Spinners:

https://developer.android.com/develop/ui/views/components/spinner

10. Seekbar

Un SeekBar es una extensión de ProgressBar que agrega un pulgar que se puede arrastrar.

El usuario puede tocar el pulgar y arrastrar hacia la izquierda o hacia la derecha para establecer el nivel de progreso actual o usar las teclas de flecha. Se desaconseja colocar widgets enfocables a la izquierda o derecha de un SeekBar.

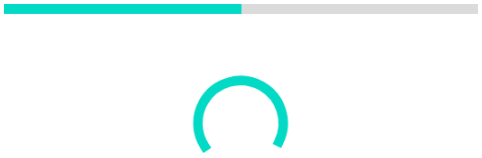


Enlace como implementar Seekbar:

https://developer.android.com/reference/android/widget/SeekBar

11. ProgressBar

Este control suele utilizarse con tareas asíncronas y servirá para informar de la evolución de un proceso, completando el tiempo de espera entre la acción del usuario y la respuesta que nos dé la aplicación.



Podemos hacerlo de 2 formas:

- Circular
- Lineal

Aquí puedes ampliar tu conocimiento sobre las progressbar:

https://developer.android.com/reference/android/widget/ProgressBar

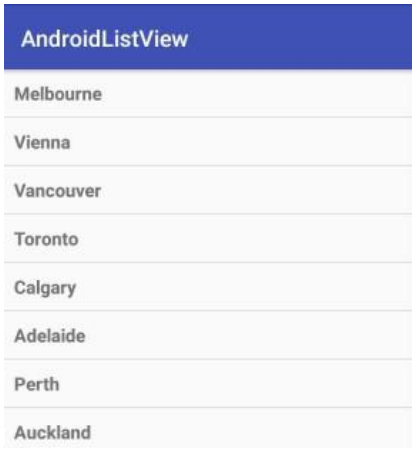
12. Listados: ListView y GridView

Ahora aprenderemos a crear listas en Android y a manejar sus diferentes elementos trabajando con los ViewGroup más frecuentes:

- ListView
- GridView

12.1 ListView

Mediante ListView visualizaremos, una lista deslizable verticalmente de varios elementos y cada uno de ellos podrá ser seleccionados sobre el propio control. Si tuviéramos más elementos de los que permite mostrar la pantalla, aparecerá una barra de scroll que permitirá acceder a ellos.



En nuestro XML se incluirá:

```
<ListView
    android:id="@+id/list_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

En Kotlin:

```
import android.widget.ArrayAdapter
import android.widget.ListView
class MainActivity : AppCompatActivity() {

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

        // use arrayadapter and define an array
        val arrayAdapter: ArrayAdapter<*>
        val users = arrayOf(
            "Virat Kohli", "Rohit Sharma", "Steve Smith",
            "Kane Williamson", "Ross Taylor"
        )

        // access the listView from xml file
        var mListView = findViewById<ListView>(R.id.userlist)
        arrayAdapter = ArrayAdapter(this,
            android.R.layout.simple_list_item_1, users)
        mListView.adapter = arrayAdapter
    }
}
```

12.2 GridView

En este caso mostramos los datos como una rejilla bidimensional, incluyendo un scroll para cuando los datos ocupen más tamaños de las posibilidades de la pantalla El código en xml será: Y para manejarlo en java, deberemos instanciar el objeto y declarar el array: GridView listado = (GridView) findViewById(R.id.nuevoGridView); final String datos = new Strin[]{“El1”,”El2”,}; ArrayAdapter

adaptador = new ArrayAdapter(this,android.R.layout.simple_list_item_1,datos); listado.setAdapter(Adaptador); y para detectar lo qeu elige el usuario lo haremos como en ListView

En el XML:

```
<GridView
    android:id="@+id/idGRV"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:horizontalSpacing="6dp"
    android:numColumns="2"
    android:verticalSpacing="6dp" />
```

Para almacenar los datos:

```
package com.gtappdevelopers.kotlingfgproject


// on below line we are creating a modal class.
data class GridViewModal(

    // we are creating a modal class with 2 member
    // one is course name as string and
    // other course img as int.
    val courseName: String,
    val courseImg: Int
)
```

Enlace como implementar GridView:

https://developer.android.com/reference/kotlin/android/widget/GridView

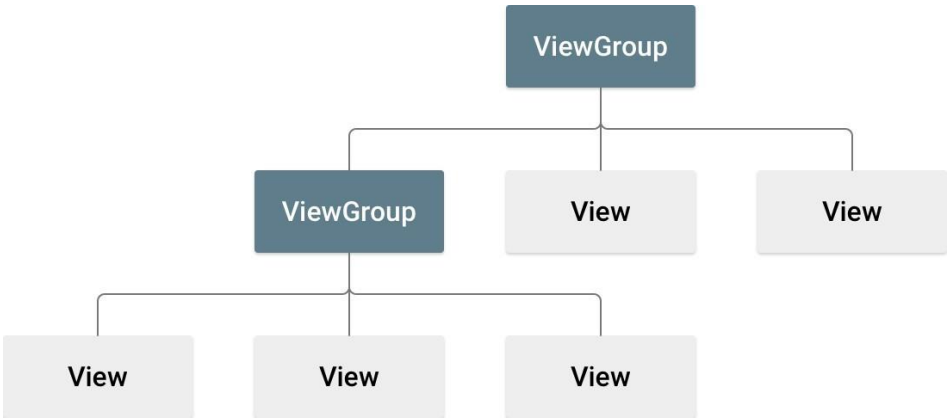
13. Layouts (Diseños)

Una vez hemos visto las diferentes clases, ahora es el momento de ver cómo pueden organizarse en nuestra aplicación, a esto lo llamamos layout y puedes encontrar dentro del layout contenedores, que nos permiten organizar un conjunto de campos. Sabemos del módulo de programación que Java organiza los controles en contenedores y que usa layout managers para poder usarlos.

Es la misma filosofía que utiliza AndroidStudio para mostrar y organizar los widgets en pantalla. Hasta ahora hemos ido declarando los ficheros en .XML, pero podríamos manejar los layouts también instanciando objetos programáticamente en tiempo de ejecución.

Al hacerlo a través del .XML estamos separando la parte gráfica de la lógica de programación, lo

que permite que las modificaciones en diseño no afecten al código y por tanto no haya que compilar cada vez, por ello y por ser la forma más sencilla de hacerlo será la que mantengamos en el curso.



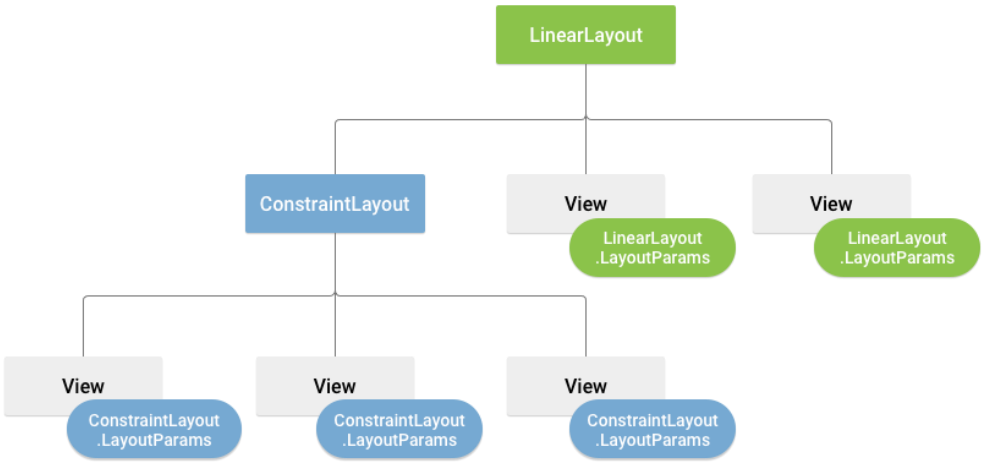
13.1 Propiedades de los layouts

Un diseño define la estructura de una interfaz de usuario en tu aplicación, por ejemplo, en una actividad. Todos los elementos del diseño se crean usando una jerarquía de objetos View y ViewGroup. Una View suele mostrar un elemento que el usuario puede ver y con el que puede interactuar. En cambio, un ViewGroup es un contenedor invisible que define la estructura de diseño de View y otros objetos ViewGroup, como se muestra en la siguiente figura:

13.2 Tipos de layouts

Los principales que utilizaremos (hasta ahora lo hacíamos todo con ConstraintLayout), serán:

- LinearLayout: uno de los que crea por defecto.
- RelativeLayout: modelo basado en reglas
- TableLayout/GridLayout: modelo basado en celdas



13.2.1 Constraint layout

Es el que viene por efecto con Artic y una de las ventajas que tiene es que permite la simplificación del proceso al ser drag and drop. En cuanto a posicionamiento se hace como el relativeLayout, pero su uso es más sencillo y flexible. Es importante revisar las soluciones de las actividades, donde se explican estos conceptos:

https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout

13.2.2 LinearLayout

Es un modelo de cajas, que alinea los controles de forma horizontal o vertical, según lo indiquemos. Este diseño organiza sus elementos secundarios en una sola fila horizontal o vertical. Si la longitud de la ventana supera la de la pantalla, crea una barra de desplazamiento.

13.2.3 RelativeLayout

Te permite especificar la ubicación de los objetos secundarios en función de ellos mismos (el objeto secundario A a la izquierda del objeto secundario B) o en función del elemento primario (alineado con la parte superior del elemento primario).

13.2.4 TableLayout

Este layout crea una distribución basada en filas y columnas, para comenzar definirás el número de filas y columnas con android:rowCount y android:columnCount y luego en cada uno de los elementos que introduzcamos, indicar en qué fila y en qué columna estarán.

https://developer.android.com/guide/topics/ui/layout/grid

Enlace con más información:

https://developer.android.com/develop/ui/views/layout/declaring-layout

14. Recursos de animación

Animación de propiedades:

Se trata de una animación definida en XML que modifica las propiedades del objeto de destino, como el color de fondo o el valor Alfa, durante un período determinado.

Ejemplo de sintaxis:

```
<set android:ordering="sequentially">
  <set>
    <objectAnimator
      android:propertyName="x"
      android:duration="500"
      android:valueTo="400"
```

```
        android:valueType="intType"/>
    <objectAnimator
        android:propertyName="y"
        android:duration="500"
        android:valueTo="300"
        android:valueType="intType"/>
  </set>
  <objectAnimator
    android:propertyName="alpha"
    android:duration="500"
    android:valueTo="1f"/>
</set>
```

Recursos de animación:

https://developer.android.com/guide/topics/resources/animation-resource?hl=es-419

15. Menús y submenús

Los menús son un componente común de la interfaz de usuario en muchos tipos de aplicaciones. Para proporcionar una experiencia de usuario conocida y uniforme, debes usar las API de Menu a fin de presentar al usuario acciones y otras opciones en las actividades.

A partir de Android 3.0 (nivel de API 11), los dispositivos con Android ya no tienen que proporcionar un botón Menú exclusivo. Con este cambio, las apps para Android dejarán de depender de los paneles de menú tradicionales de 6 elementos y, en su lugar, proporcionarán una barra de la app para mostrar las acciones más comunes del usuario.

Aunque haya cambiado el diseño de la experiencia del usuario para algunos elementos de menú, la semántica para definir un conjunto de acciones y opciones sigue basándose en las API de Menu. Esta guía muestra cómo crear tres tipos fundamentales de presentaciones de menús o acciones en todas las versiones de Android.

En Android podemos encontrar 3 tipos diferentes de menús:

- Menús Principales: aparecen en la zona inferior de la pantalla al pulsar el botón 'menu' del teléfono.
- Submenús. Son secundarios, se pueden mostrar al pulsar sobre una opción de un menú principal.
- Menús Contextuales. Útiles en muchas ocasiones, aparecen al realizar una pulsación larga sobre algún elemento de la pantalla.

15.1 Menús principales

Para crear un menú en nuestro fichero xml (los iconos utilizados deberán estar por supuesto en las carpetas «res\drawable-…» de nuestro proyecto).

Para todos los tipos de menús, Android proporciona un formato XML estándar que permite definir los elementos de menú. En lugar de incorporar un menú en el código de la actividad, debes definir un menú y todos los elementos en un recurso de menú XML. Luego, puedes aumentar el recurso de menú (cargarlo como un objeto Menu) en la actividad o el fragmento.

El uso del recurso de menú es una práctica recomendada por algunos motivos:

- Es más fácil visualizar la estructura del menú en XML.
- Separa el contenido del menú del código de comportamiento de la aplicación.
- Te permite crear configuraciones alternativas del menú para diferentes versiones de plataforma, tamaños de pantalla y otras configuraciones aprovechando el framework de recursos de la app.

Para definir el menú, crea un archivo en formato XML dentro del directorio res/menu/ del proyecto y desarrolla el menú con los siguientes elementos:

<menu>
Define un Menu, que es un contenedor para elementos de menú. Un elemento <menu> debe ser el nodo raíz del archivo y puede tener uno o más elementos <item> y <group>.

<item>
Crea un MenuItem, que representa un único elemento en un menú. Este elemento puede contener un elemento <menu> anidado para crear un submenú.

<group>
Es un contenedor opcional e invisible para elementos <item>. Te permite categorizar los elementos de menú para que compartan propiedades, como el estado de la actividad y la visibilidad. Para obtener más información, consulta la sección Cómo crear grupos del menú.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new_game"
        android:title="@string/new_game"
        android:showAsAction="ifRoom"/>
   <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```

Enlace como implementar menús:

https://developer.android.com/guide/topics/ui/menus?hl=es-419

15.2 Submenús

Si queremos crear sus opciones dentro de un menú procederemos igual que en el caso anterior, definiéndolo desde nuestro fichero XML.

https://developer.android.com/reference/android/view/SubMenu

15.3 Menú contextuales

Un menú contextual es un menú flotante que aparece cuando el usuario hace un clic largo en un elemento. Proporciona acciones que afectan el contenido seleccionado o el marco contextual.

En el modo de acción contextual, se muestran los elementos de acción que afectan al contenido seleccionado en una barra en la parte superior de la pantalla y se permite al usuario seleccionar varios elementos.

En un **menú contextual flotante:**

Un menú aparece como una lista flotante de elementos de menú (similar a un cuadro de diálogo) cuando el usuario realiza un clic largo (mantiene presionado) en una vista que declara admitir un menú contextual. Los usuarios pueden realizar una acción contextual de a un elemento por vez.

En el **modo de acción contextual:**

Este modo es una implementación del sistema de ActionMode que muestra una barra de acciones contextuales en la parte superior de la pantalla con elementos de acción que afectan a los elementos seleccionados. Cuando este modo está activo, los usuarios pueden realizar una acción en varios elementos a la vez (si la app lo permite).

Enlace como implementar menús contextuales:

https://developer.android.com/guide/topics/ui/menus?hl=es-419#context-menu

16. Orientación de la pantalla

Una de las principales características de los smartphones actuales es su habilidad para cambiar la orientación de la pantalla. En Android existen 2 orientaciones de pantalla:

- Horizontal
- Vertical

Por defecto cuando cambias la orientación de la pantalla de tu dispositivo Android, la actividad automáticamente se redibuja en la nueva orientación.

Sin embargo, cuando las vistas son redibujadas, pueden haber cambios en las ubicaciones, dependiendo del layout seleccionado, para que todo quede como queremos hay dos técnicas disponibles:

- Anchoring (Anclado)
- Redimensionamiento y reposicionamiento.

https://developer.android.com/guide/topics/resources/runtime-changes

17. BIBLIOGRAFÍA

i. Android. Programación Multimedia y de dispositivos móviles. Garceta.

ii. Programación Multimedia y Dispositivos Móviles. Editorial Síntesis.

iii. Android App Development. For Dummies.

iv. Beginning Android Programming with Android Studio. Wrox.

v. Java Programming for Android Developers. For Dummies.

vi. https://academiaandroid.com/

vii. https://developer.android.com/

viii. https://www.redhat.com

ix. https://desarrolloweb.com

x. https://m2.material.io/develop/android/components/checkboxes

xi. https://fonts.google.com/knowledge

xii. https://www.geeksforgeeks.org/android-gridview-in-kotlin/