

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES
**TÉCNICO EN DESARROLLO DE APLICACIONES
MULTIPLATAFORMA**

Programación en Android II: Layouts

04

ÍNDICE

/ 1. Introducción y contextualización práctica	3
/ 2. Introducción a los Layouts	4
/ 3. LinearLayout	4
/ 4. Caso práctico 1: “Anidamiento”	5
/ 5. TableLayout	6
/ 6. FrameLayout	6
/ 7. ConstraintLayout	7
/ 8. Caso práctico 2: “¿Qué tipo de Layout conviene utilizar?”	8
/ 9. Dividiendo la ventana en pestañas	8
/ 10. Resumen y resolución del caso práctico de la unidad	9
/ 11. Bibliografía	10

OBJETIVOS

Crear aplicaciones Android básicas.

Trabajar con interfaces gráficas.

Conocer diferentes elementos gráficos básicos.

Maquetar aplicaciones Android de diferente forma.

/ 1. Introducción y contextualización práctica

En esta unidad, vamos a ver la forma en la que las aplicaciones **Android maquetan sus elementos gráficos** en una interfaz.

Estudiaremos el concepto de **View** y de **Layout**, que son los elementos que nos permitirán distribuir de forma diferente una serie de elementos dentro de una interfaz gráfica.

Dentro de los Layout veremos diferentes tipos, como **LinearLayout**, **TableLayout**, **FrameLayout** y **ConstraintLayout** y profundizaremos en cada uno de ellos.

Por último, aprenderemos cómo dividir nuestra pantalla en diferentes pestañas de una forma sencilla.

Todos estos conceptos los iremos reforzando en los sucesivos temas dedicados a la programación con Android.

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad:

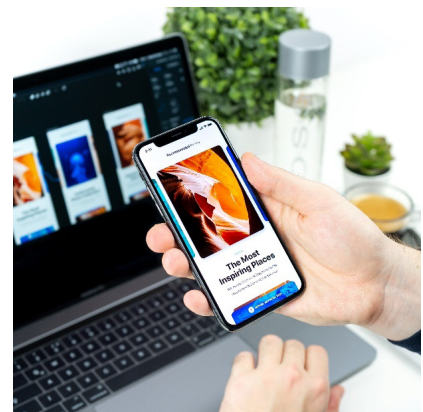


Fig.1. La expresión del bebé



Audio Intro. "Maquetación de elementos"

<https://bit.ly/3gZYYQw>



/ 2. Introducción a los Layouts

Cuando estamos diseñando una interfaz gráfica, tendremos que realizar un maquetado de los elementos gráficos que utilicemos, independientemente del lenguaje de programación que estemos utilizando. Esto significa que tendremos que distribuirlos de cierta forma en la pantalla para que se adapten al diseño que queramos conseguir.

Este proceso ya lo hemos visto en la asignatura de programación, utilizando lo que conocíamos como **Layouts** para la **distribución** o **maquetación** de los **elementos gráficos**.

De forma análoga, en **Android Studio** también vamos a utilizar **Layouts** para la maquetación de las interfaces gráficas de nuestras aplicaciones. Podemos definir los Layouts como elementos no visuales destinados a controlar la distribución, posición y dimensiones de los elementos gráficos que se insertan en su interior. Estos componentes extienden la clase base ViewGroup, como muchos otros componentes contenedores, es decir, capaces de contener a otros controles.

Como es lógico, vamos a tener diferentes tipos de Layouts a nuestra disposición, cada uno de los cuales distribuirá de forma diferente los elementos que coloquemos en su interior. Los Layouts que vamos a utilizar son los siguientes:

- **LinearLayout.**
- **TableLayout y RowLayout.**
- **FrameLayout.**
- **ConstraintLayout.**

Los Layouts los podremos encontrar dentro de la **paleta (Palette)** de elementos de **Android Studio**, en la pestaña llamada Layouts.

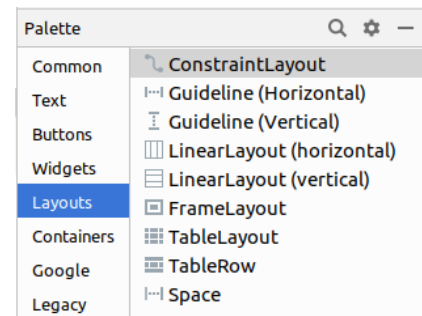


Fig. 2. Pestaña de elementos Layouts.

/ 3. LinearLayout

El primer gestor de elementos que vamos a ver es el **LinearLayout**.

Este Layout va a colocar los elementos que contenga de forma consecutiva en pantalla. Este puede, a su vez, contener elementos en dos orientaciones diferentes: vertical y horizontal. La orientación que necesitemos la podremos indicar con la propiedad android:orientation.

Cuando estamos utilizando un LinearLayout también deberemos **ajustar la propiedad android:layout_weight**. Con dicha propiedad podremos otorgar a los elementos del Layout un tamaño proporcionado entre todos ellos. El tema de las proporciones puede parecer algo extraño en un principio, pero una vez lo dominemos, será algo básico y fácil de hacer.

Vamos a imaginar que en un LinearLayout con orientación vertical tenemos dos elementos. Uno de ellos tiene un **layout_weight="1"** y el otro un **layout_weight="2"**. Con esto vamos a conseguir que todo el contenido del Layout quede ocupado por estos dos elementos, además, configuramos que el segundo ocupe el doble de alto que el primero, por la forma en la que están configuradas sus propiedades weight.



Otra de las propiedades con las que podremos jugar con este tipo de Layout es android:gravity, mediante la cual podremos indicar la alineación de los elementos dentro del mismo. Esta propiedad puede tener los siguientes valores:

- **Center:** Centra los elementos dentro del Layout tanto de forma horizontal como vertical.
- **Top:** alinea arriba los elementos dentro del Layout.
- **Bottom:** Alinea abajo los elementos dentro del Layout.
- **Right:** Alinea a la derecha los elementos dentro del Layout.
- **Left:** Alinea a la izquierda los elementos dentro del Layout.
- **Center_horizontal:** Alinea al centro de forma horizontal los elementos dentro del Layout.
- **Center_vertical:** Alinea al centro de forma vertical los elementos dentro del Layout.

Una propiedad bastante interesante que poseen todos los Layouts es que se pueden ocultar automáticamente todos los elementos que contengan. Para ello, deberemos proporcionar un id al Layout y ligarlo a un objeto. De esta forma, y mediante el método setVisibility, podremos aplicarle los siguientes valores:

- **View.GONE:** Oculta el Layout y sus elementos.
- **View.VISIBLE:** Muestra el Layout y sus elementos.



Vídeo 1. "Ejemplo de LinearLayout"
<https://bit.ly/3j5GrUN>



/ 4. Caso práctico 1: "Anidamiento"

Planteamiento: Pilar y José están repasando los conceptos de vista y Layout.

«Esto se parece muchísimo a cuando diseñábamos interfaces gráficas con NetBeans, que recuerdos...», le comenta José a Pilar.

Ella asiente y se pregunta qué ocurrirá cuando tengan que crear una interfaz bastante compleja, ya que las aplicaciones móviles están bastante limitadas.

Nudo: ¿Crees que las limitaciones de los dispositivos móviles van a influir en el diseño y maquetado de los elementos de la interfaz gráfica de una pantalla?

Desenlace: Como ya sabemos, los dispositivos móviles poseen bastantes restricciones a la hora de realizar las aplicaciones. Y una de ellas es la pantalla.

La restricción del tamaño de la pantalla incluirá en la cantidad de elementos que incluyamos en nuestra interfaz, ya que si la pantalla es pequeña se va a llenar rápidamente; no obstante, no va a afectar en absoluto a los Layouts.

Estos nos van a permitir crear las interfaces tan complejas como las necesitemos, incluso podremos anidar unos dentro de otros, pudiendo anidar incluso Layouts diferentes sin ningún problema.

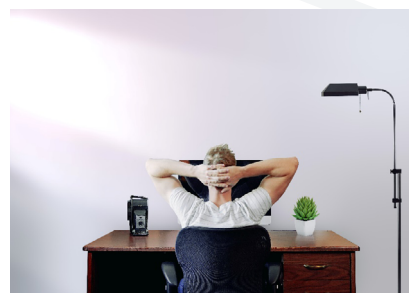


Fig. 3. El diseño de las aplicaciones es algo a hacer, cuidadosamente.

/ 5. TableLayout

Un **TableLayout** nos va a permitir distribuir los elementos que introduzcamos en él en forma de tabla, definiendo las filas y columnas necesarias, así como la posición de cada componente dentro de esta.

La estructura de la tabla creada se define de forma similar a como se hace en el **lenguaje HTML**: tendremos que indicar las filas que compondrán dicha tabla (mediante **TableRow**) y, dentro de cada una de las filas, las columnas necesarias, con la salvedad de que no existe ningún objeto especial para definir una columna, sino que iremos insertando directamente los componentes necesarios dentro del **TableRow**.

Cada componente insertado corresponderá a una columna de la tabla, pudiendo ser un elemento simple u otro Layout. De esta forma, el número final de filas de la tabla se corresponderá con el número de elementos TableRow insertados; y el número total de columnas quedará determinado por el número de componentes de la fila que más componentes contenga.

Una característica que debemos tener bastante en cuenta es la posibilidad de configurar celdas para que ocupen el espacio que les corresponderían a varias de las columnas de la tabla (funcionando igual que el atributo colspan de HTML). Esto lo podremos configurar con la propiedad android:layout_span del elemento que hayamos añadido a nuestro TableLayout.

Como norma general, el ancho de cada una de las columnas va a corresponder al ancho del mayor componente que haya dentro de la columna, aunque también tenemos unas propiedades que nos van a permitir modificar este comportamiento:

- **android:stretchColumns**: Podremos indicar qué columnas se van a poder expandir para ocupar el espacio libre que han dejado las demás columnas a la derecha.
- **android:shrinkColumns**: Nos permitirá indicar qué columnas se van a poder contraer para dejar espacio, y que se puedan salir por la derecha.
- **android:collapseColumns**: Con esta propiedad vamos a poder indicar qué columnas de nuestra tabla queremos ocultar de forma completa.

Celda 2.1	Celda 2.2	Celda 2.3
Celda 3.1	Celda 3.2	

Fig. 4. Ejemplo de TableLayout.

/ 6. FrameLayout

De todos los Layout que vamos a ver en esta unidad, el **FrameLayout es el más simple**. Este alineará automáticamente en la esquina superior izquierda del Layout todos los elementos que contenga, de forma que todos los elementos se irán superponiendo encima del anterior. Este Layout suele utilizarse mucho para mostrar un único elemento gráfico. Dentro de todos los elementos gráficos, incluidos Layouts, vamos a tener **dos propiedades básicas para poder gestionar su tamaño**:

- **android:layout_width**: Esta propiedad va a definir el tamaño del alto del elemento.
- **android:layout_height**: Definirá el tamaño del ancho del elemento.



En estas propiedades podrán configurarse **los siguientes valores**:

- **match_parent**: Con este valor indicamos que el elemento debe medir lo mismo que su contenedor.
- **wrap_content**: Con este valor indicamos que el elemento debe medir lo justo y necesario respecto a su contenido.

Es decir, si tenemos en un Layout un elemento con las propiedades `layout_width` y `layout_height` con valor `match_parent`, ese elemento ocupará todo el tamaño de su contenedor.

El `FrameLayout` se encuentra dentro de la paleta de elementos gráficos, en la pestaña `Layouts`.

Dentro de un Layout, sea cual sea, podremos introducir otro Layout, para así conseguir una interfaz gráfica lo más compleja posible.

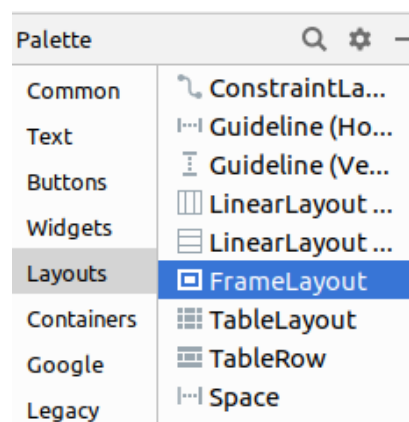


Fig. 5. Localización de `FrameLayout`.



/ 7. ConstraintLayout

Este es el último Layout que se ha añadido a Android Studio y que aparece para ser usado de forma predeterminada cuando creamos una actividad. Este Layout permite crear unos **diseños bastante complejos** sin necesidad de tener que anidar Layouts, ya que la práctica del anidamiento ocasionaba problemas de memoria y eficiencia en dispositivos de poco rendimiento.

Es muy **flexible** y **fácil de usar** desde el editor visual de Android Studio. De hecho, es bastante recomendable crear los Layout de las interfaces gráficas con las herramientas drag-and-drop, en lugar de editar el código del fichero XML. Hay que decir que el resto de Layouts que hemos visto entrañan menos complejidad de creación desde XML que `ConstraintLayout`.

Las posiciones de las diferentes vistas dentro de este Layout se definen usando `constraint` o restricción. Podemos definir una restricción en relación al elemento contenedor, a otra vista, o respecto a una línea de guía (elemento `GuideLine`). Para cada una de las vistas deberemos definir al menos un `constraint` horizontal y uno vertical, aunque también podemos definir más de un `constraint` en el mismo eje.

No es aconsejable usar este tipo de Layout cuando no se tiene un dominio considerable de la materia, ya que puede ser bastante lioso.

Como cada Layout, contemplará una serie de nuestras propiedades a los elementos que le agreguemos.

Un ejemplo de estas nuevas propiedades son las distancias a los lados que tendrán los elementos con respecto a los demás.

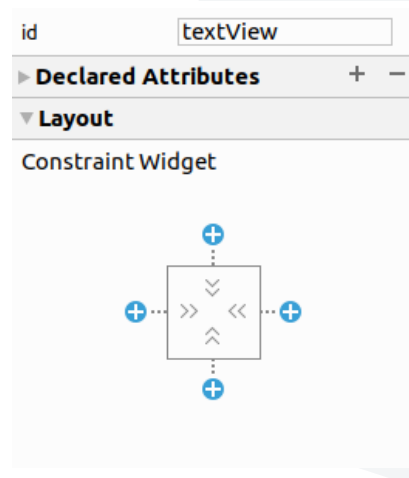


Fig. 6. Propiedades de un elemento en `ConstraintLayout`.

/ 8. Caso práctico 2: “¿Qué tipo de Layout conviene utilizar?”

Planteamiento: Una vez Pilar y José se han adentrado en el mundo de los Layouts, están listos para crear las aplicaciones Android que necesiten de una forma correcta. Nuestros amigos ya conocen que existen varios tipos de Layouts, cada uno con sus respectivas propiedades y distribución de elementos. La pregunta que se hacen ahora es, ¿cuándo se deberá utilizar un Layout u otro? ¿Se puede utilizar un único Layout y adaptarlo a nuestras necesidades?

Nudo: ¿Qué piensas al respecto? ¿Crees que dominando un único Layout se puede adaptar para crear cualquier tipo de distribución en una interfaz gráfica?

Desenlace: Esto es algo bastante normal que suele ocurrir cuando estamos empezando a realizar interfaces gráficas, no solo en Android, sino en cualquier lenguaje.

Cada uno de los Layouts o maquetadores que hemos visto tiene sus propias peculiaridades: unos distribuyen los elementos en forma de matriz, otros de forma horizontal o vertical, etc.

Es posible que utilizando un determinado Layout podamos adaptarlo a la gran mayoría de diseños que se nos puedan ocurrir, pero esto conllevará un arduo trabajo de adaptación de ese Layout al resultado que busquemos.

Por este motivo, lo más fácil, rápido y eficiente será aprender a utilizar cada uno de los diferentes Layouts. Aunque pueda llevar algo más de tiempo al principio, el resultado será que podremos desarrollar una interfaz gráfica de forma sencilla, versátil y aprovechando todas las opciones que nos brindan cada Layout. De esta manera, el desarrollo de las aplicaciones será mucho más llevadero para el propio programador.

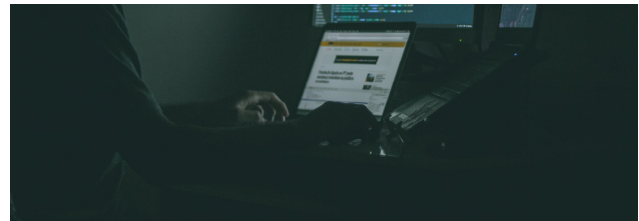


Fig. 7. Desarrollando aplicaciones.

/ 9. Dividiendo la ventana en pestañas

Después de ver cómo funcionan los distintos Layouts con la redistribución de elementos, veremos que es posible dividir una pantalla en varias, por medio de pestañas.

Disponemos de esta opción gracias a las pestañas (llamadas Tab) y a los Fragments.

- **El trabajo con pestañas consistirá en** definir una única actividad, y vamos a establecer varias pestañas, de forma que cada una de ellas tendrá un contenido diferente a las demás.
- **Los Fragments** nos van a **permitir dividir** la interfaz de una pantalla de forma proporcional.

Para integrar todo esto, Android Studio nos facilita el control TabLayout y los TabItem. El primero es el Layout que deberemos utilizar para usar pestañas; el segundo contiene las pestañas en sí mismas.

Internamente vamos a utilizar la clase **ViewPager**, que nos permitirá cambiar de forma fluida de pestañas.

Para crear una aplicación con pestañas, tendremos que indicar que deseamos este tipo de aplicación cuando creamos el proyecto en Android Studio. Por lo tanto, deberemos seleccionar Tabbed Activity como plantilla, en lugar de Empty Activity, tal como estábamos haciendo hasta ahora.

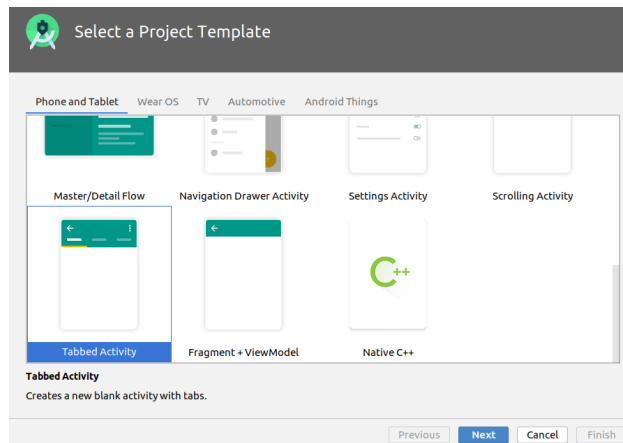




Fig. 8. Selección de Tabbed Activity.

Una vez hecho esto, el propio **Android Studio** nos creará una aplicación con la pantalla principal con varias pestañas.



Vídeo 2. "Creación de un proyecto con pestañas"

<https://bit.ly/3j7lgEX>



/ 10. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad, hemos visto las **diferentes formas** en las que podremos **distribuir los elementos** en una **interfaz gráfica** de nuestra aplicación Android.

Hemos estudiado **conceptos genéricos** como las **vistas (View)** y los **Layouts**, los cuales se encargan de distribuir los elementos en la interfaz gráfica.

Dentro de los Layouts, hemos profundizado en los siguientes: **LinearLayout**, **TableLayout**, **FrameLayout** y **ConstraintLayout**. Hemos visto, a su vez, cómo cada uno de ellos distribuye los elementos.

Por último, hemos comprobado que podemos dividir cualquier pantalla de nuestra aplicación Android en diferentes pestañas, pudiendo incorporar diferentes elementos gráficos en cada una de ellas y, por supuesto, pudiendo organizar cada pestaña con diferentes Layouts.

Resolución del caso práctico de la unidad

Con la experiencia que ya tenemos en el mundo de la programación aplicaciones, sabemos que cuando realizamos interfaces gráficas hay que distribuir los elementos de cierta forma para que se adapten a lo que queremos diseñar.

Las aplicaciones Android no son una excepción y, si queremos maquetar una pantalla con un orden determinado, tendremos que distribuir los elementos, y, por tanto, tener un mecanismo de distribución de los mismos para que se adapten a lo que queremos diseñar. Tal y como hemos visto en el tema, **deberemos utilizar Layouts** para maquetar nuestras pantallas.

Esto es algo bastante básico, ya que, como pasaba en las aplicaciones de escritorio, si no maquetamos o distribuimos de forma correcta los elementos, el resultado será desastroso.

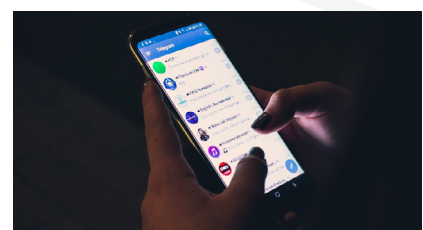


Fig. 9. Aplicación maquetada en forma de lista.



/ 11. Bibliografía

Máster en Desarrollo de Aplicaciones Android - Uso de ConstraintLayout. (s. f.). Recuperado el 23 de junio de 2020 de <http://www.androidcurso.com/index.php/881>

Revelo, J. (29 de enero de 2020). TabLayout: ¿Cómo Añadir Pestañas En Android? Hermosa Programación: +50 Tutoriales Desarrollo Android. Recuperado de: <http://www.hermosaprogramacion.com/2015/06/tablayout-como-anadir-pestanas-en-android/>

Sgoliver. (17 de agosto de 2010). [Entrada de Blog]. Interfaz de usuario en Android: Layouts. Recuperado de: <https://www.sgoliver.net/blog/interfaz-de-usuario-en-android-layouts/>

WUOLAC