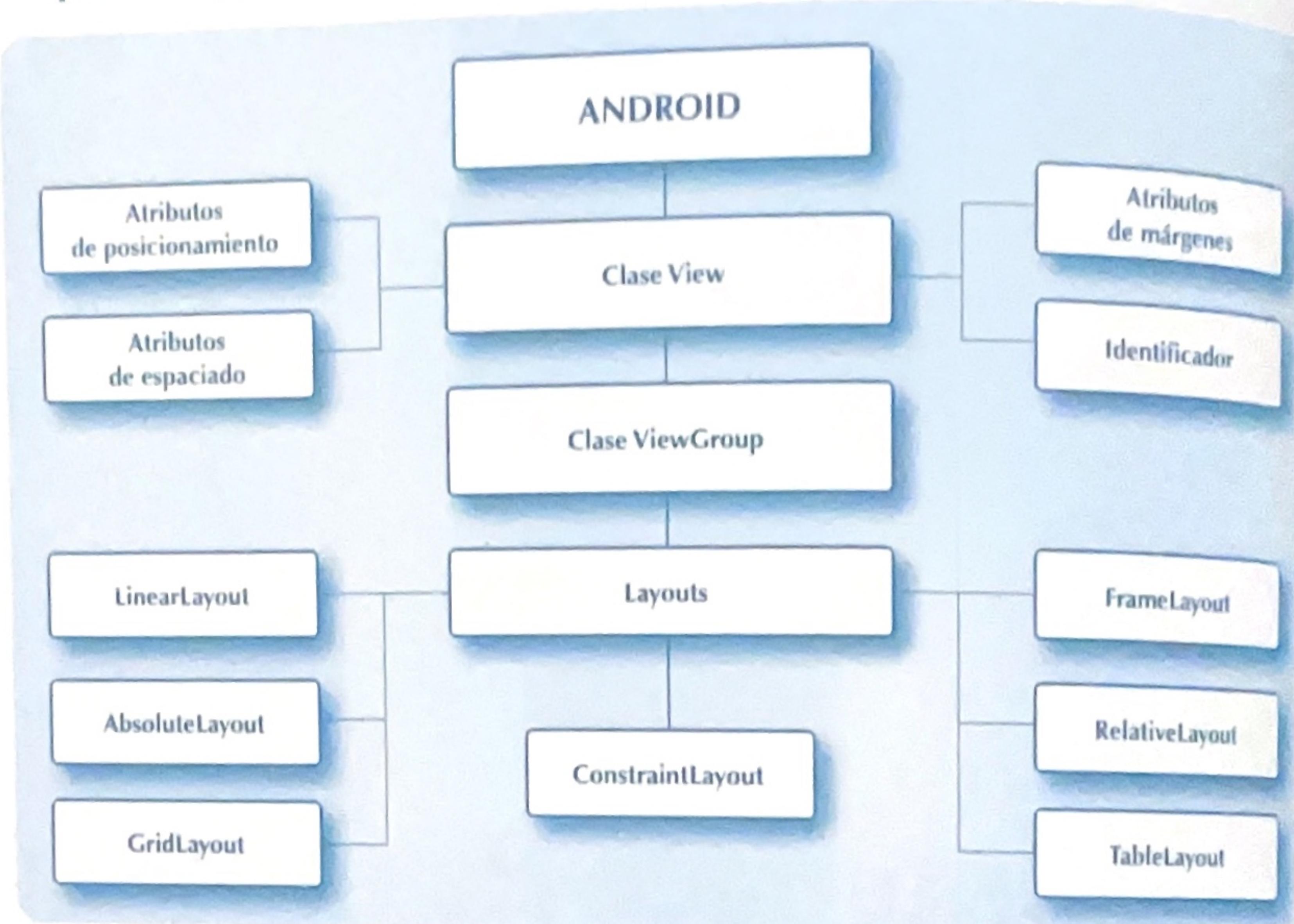


Interface de usuario: los layouts

Objetivos

- ✓ Identificar la diferencia entre la clase View y la ViewGroup.
- ✓ Familiarizarse con los atributos comunes a estas clases.
- ✓ Manejar con soltura los atributos de posicionamiento.
- ✓ Saber aplicar cada tipo de layout según las necesidades de la aplicación.
- ✓ Trabajar sin dificultad con el anidamiento de layout.
- ✓ Descubrir la variedad de uso del TableLayout y del GridLayout.
- ✓ Posicionar sin dificultad los objetos en un RelativeLayout.
- ✓ Conocer la relación entre diseño y código en los ConstraintLayouts.

Mapa conceptual



Glosario

Atributo. Un atributo define la propiedad de un objeto, elemento o archivo. También puede establecer el valor para una instancia determinada de los mismos.

Interfaz. Conexión funcional entre dos componentes, dispositivos o sistemas, facilitando la comunicación entre ellos y permitiendo el intercambio de información.

match_parent. Es un parámetro que fuerza al objeto a expandirse para ocupar todo el espacio disponible en el elemento de diseño que lo contiene.

wrap_content. Establece el tamaño de un contenedor forzándolo a expandirse solo lo suficiente como para contener los objetos que almacena.

4.1. Introducción

Hasta ahora se han visto los fundamentos y componentes de una aplicación en Android, también su ciclo de vida y los elementos que forman la estructura de un proyecto, además de conocer cómo se estructuran los recursos que pueden ser necesarios a la hora de desarrollar

una aplicación. También se han dado los primeros pasos en programación creando una pequeña aplicación, con una actividad simple e incluso haciendo que esta llame a una segunda actividad.

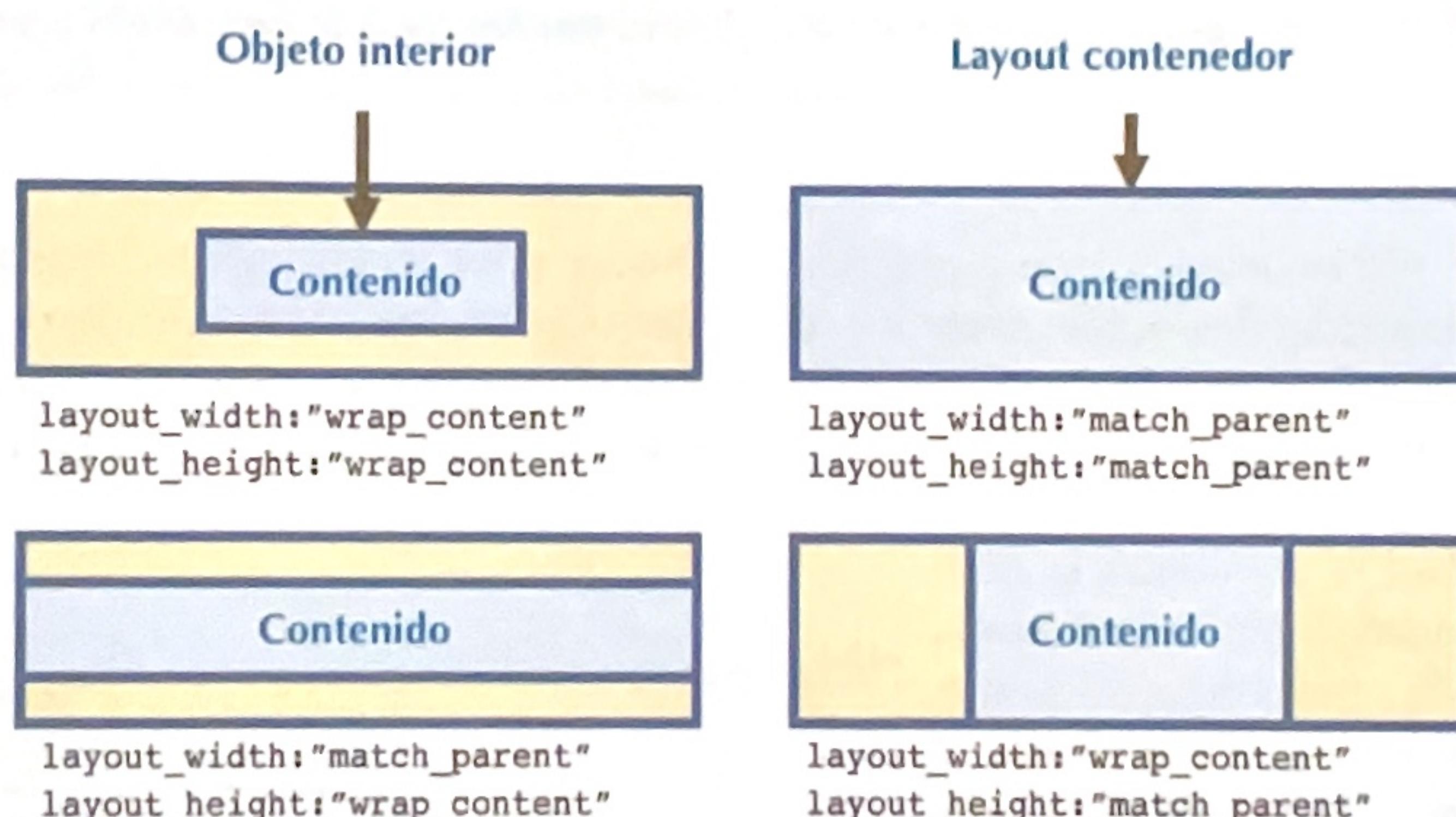
A partir de este capítulo se irán siguiendo pasos de complejidad creciente hasta adquirir soltura en el desarrollo de aplicaciones nativas. Es sumamente necesario que se enfoque cada uno de los apartados desde un punto de vista práctico, realizando las actividades propuestas, tanto de ejercicios de seguimiento como de ampliación.

Se iniciará este aprendizaje trabajando los elementos de interfaz, abordando primeramente para ello los layouts como objetos contenedores de las vistas y de necesario conocimiento para progresar en los siguientes capítulos de este libro.

4.2. Interfaces de usuario. Clases asociadas

El interfaz de una aplicación Android es aquello que aparece en pantalla, que el usuario puede ver y, por tanto, interactuar con él. Android nos ofrece un gran número de componentes implementados y preparados para su uso en la interfaz, en los cuales la clase View sirve siempre como clase base.

La librería android.view nos proporciona los elementos de la interfaz para construir las vistas, lo que se puede hacer usando Java, en principio más complejo y, por tanto, menos eficiente, o bien definiéndolo en el archivo XML ubicado en el directorio res/layout.



La clase View como clase principal en la jerarquía de vistas tiene una serie de atributos que serán heredados por todos los elementos que de ella provienen. A cualquiera de ellos le puedes asignar valores medidos (entre comillas) en pulgadas (*in*), milímetros (*mm*), puntos (*pt*), píxeles (*px*), píxeles independientes de la densidad (*dp*) y píxeles independientes de la escala (*sp*). Más adelante (apartado pantalla) se verá el significado y utilidad de cada uno; por el momento, es recomendable utilizar como unidad de medida los *píxeles independientes de la densidad (dp)*.

CUADRO 4.1
Atributos

Atributos de posicionamiento	layout_width	ancho
	layout_height	alto
Atributos para los márgenes	layout_margin	cuatro márgenes
	layout_marginBottom	márgen inferior
	layout_marginLeft	márgen izquierdo
	layout_marginRight	márgen derecho
	layout_marginTop	márgen superior
Atributos para el espaciado	android:padding	espaciado a los cuatro lados
	android:paddingTop	espaciado superior
	android:paddingBottom	espaciado inferior
	android:paddingLeft	espaciado izquierdo
	android:paddingRight	espaciado derecho

**TOMA NOTA**

Para posicionar los objetos que heredan de la clase View además de poder tomar valores absolutos (en cualquiera de las unidades citadas), también pueden asignársele los valores relativos *wrap_content*, cuando se desea ajustar el tamaño al contenido del objeto o *match_parent* (antes *fill_parent*) para tomar el máximo tamaño que le permite el contenedor.

Si se desea centrar o justificar la vista, se utilizará el atributo *layout_gravity*, y para distribuir el espacio disponible entre los diferentes objetos, *layout_weight*. Por último, para poder identificar el objeto debe utilizarse *@id/+nombre* y para acceder a este identificador, *@+id/nombre*. También pueden utilizarse identificadores definidos por el sistema *@+android:id/nombre*.

Dentro de esta gran clase se encuentra la clase android.view.ViewGroup. Ella proporciona, como su propio nombre indica, los objetos ViewGroup, cuya función es contener y controlar colecciones de View o de otros ViewGroup.

Los objetos Views han de ser hijos de objetos ViewGroup, por lo tanto, se ha de comenzar el diseño de la interfaz con un objeto ViewGroup, a

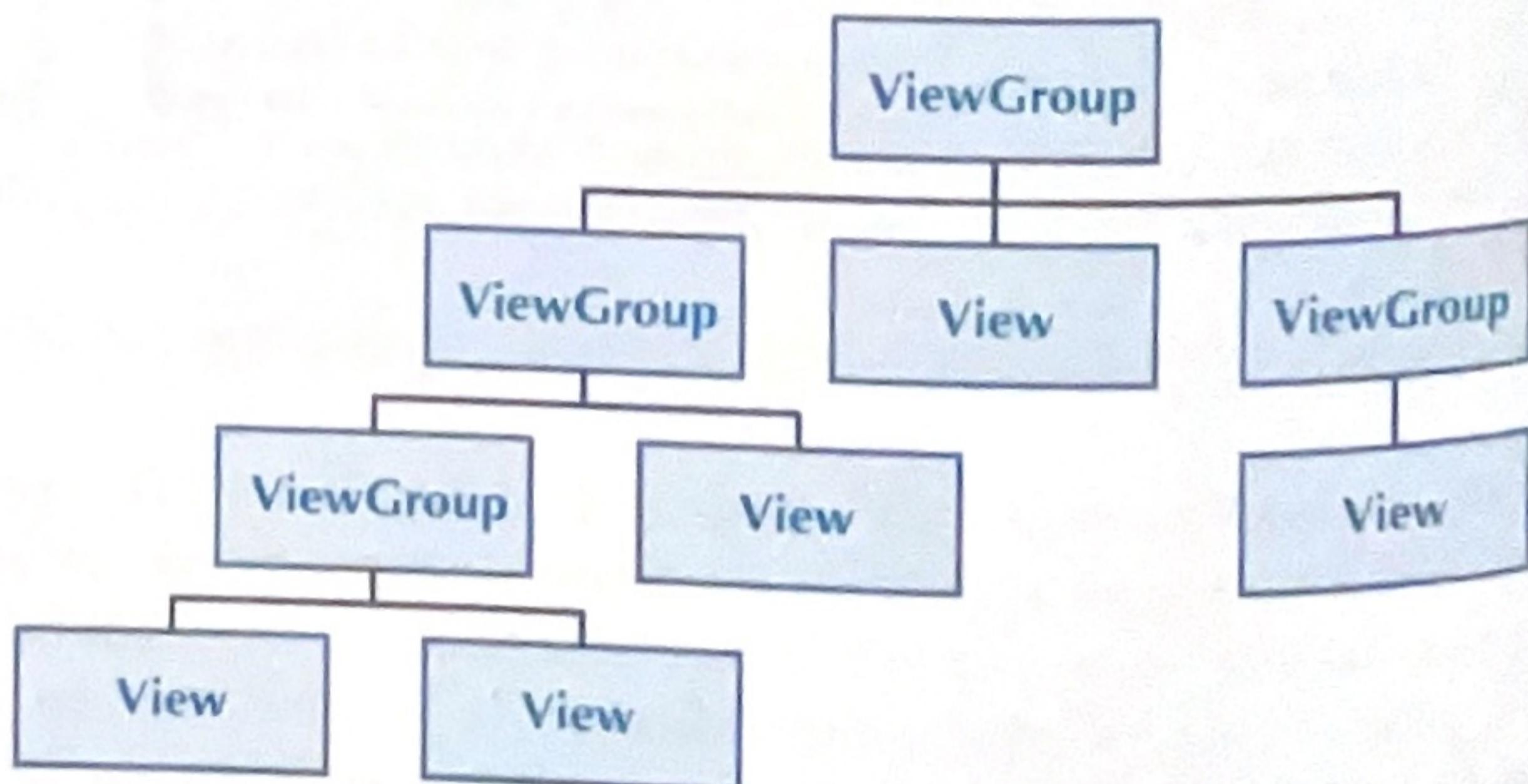


Figura 4.2
Jerarquía de la clase View.

partir del cual se puede realizar un árbol con tantas bifurcaciones como se desee. Utilizaremos la clase ViewGroup como base de estudio de la clase layouts, que son subclases que proveen de los tipos más comunes de layouts de pantalla.

RECUERDA

- ✓ Los atributos de posicionamiento deberán usarse siempre que se coloque un objeto en la vista de la aplicación.
- ✓ Los atributos de espaciado y márgenes tan solo se usarán cuando se deseé mejorar la estética de la aplicación.
- ✓ Siempre es recomendable identificar los objetos en la vista aunque luego no se vaya a acceder a ellos desde el Java de la aplicación.

4.3. Layouts

Los layouts son elementos no visuales destinados a controlar la distribución, la posición y las dimensiones de los controles que se insertan en su interior. Dentro de cada uno pueden ubicarse todos los elementos que sean necesarios en el interfaz de la actividad, incluidos otros layouts, lo que permitirá estructurar la pantalla de la manera deseada. En función de la forma y el posicionamiento en pantalla, existe gran variedad de layouts, que se estudiarán a continuación.

4.4. LinearLayout

Este layout apila uno tras otro todos sus elementos hijos de forma horizontal o vertical, según se establezca su atributo android:orientation.

Si se escribe el siguiente código, tan solo aparecerá una pantalla en rojo.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#ff0000">
</LinearLayout>
```

A continuación, se anidarán, en vertical, tres layout; el central será de triple tamaño que los otros dos y se pondrán en azul los dos más pequeños.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```

    android:orientation="vertical">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:background="#0000ff">
        </LinearLayout>
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="3"
            android:background="#ffffff">
        </LinearLayout>
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:background="#0000ff">
        </LinearLayout>
    </LinearLayout>

```



Figura 4.3
Layouts
en vertical.

Cuando una aplicación tiene un contenido de diseño, en el interfaz, con mayor tamaño que la altura del dispositivo y se necesita hacer el contenido desplazable, se utilizará ScrollView para desplazamiento vertical o HorizontalScrollView para desplazamiento horizontal.

RECUERDA

- ✓ Con `android:background` se puede colorear el fondo de los objetos de la vista.
- ✓ `layout_weight` es un atributo que permite dar a los elementos contenidos en el layout unas dimensiones proporcionales entre ellos.
- ✓ El código `xmlns:android="http://schemas.android.com/apk/res/android"` deberá usarse en el layout principal de la aplicación y sirve para declarar el espacio de nombres donde esta trabajará.

4.5. FrameLayout

Coloca todos sus controles hijos alineados con su esquina superior izquierda, sin distribuirlos, de forma que cada control quedará oculto por el control siguiente.

Este contenedor ofrece la posibilidad de modificar la visibilidad del elemento deseado dentro de su contenido, para lo cual será necesario utilizar la propiedad `android:visibility`, lo que es muchas veces útil para determinados efectos de animación. Para colocar los objetos hijos en la posición deseada, podemos utilizar la propiedad `android:gravity`.

TEN EN CUENTA

- ✓ No debe confundirse `android:gravity` con `android:layout_gravity`: el primero establece la posición del contenido y el segundo posiciona el objeto con respecto a su elemento padre.

```

<FrameLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/framelayout" >
    <FrameLayout
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:background="#ff0000"
        android:layout_gravity="left|bottom">
    </FrameLayout>
    <FrameLayout
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:background="#00ff00"
        android:layout_gravity="right|top">
    </FrameLayout>
</FrameLayout>

```

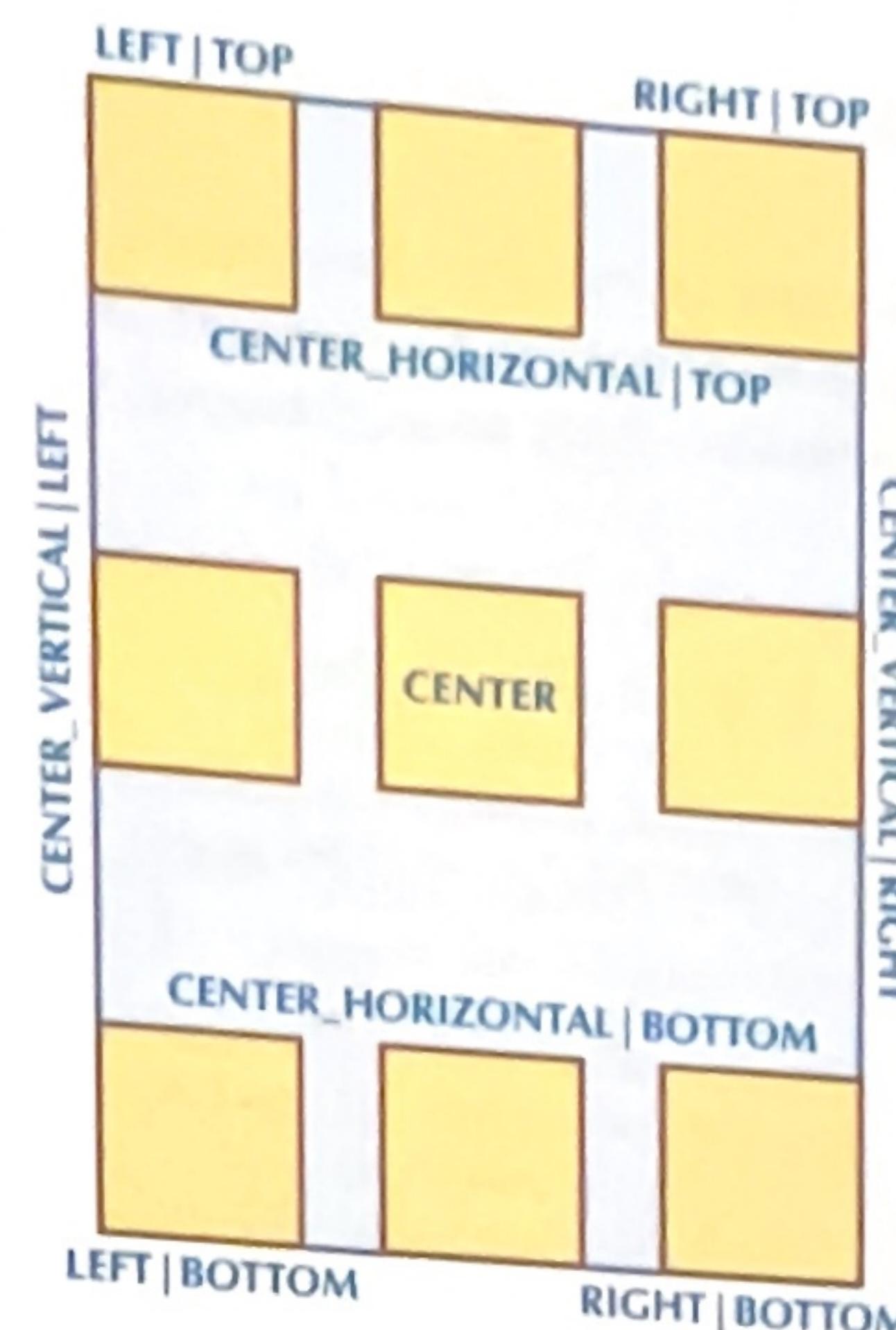
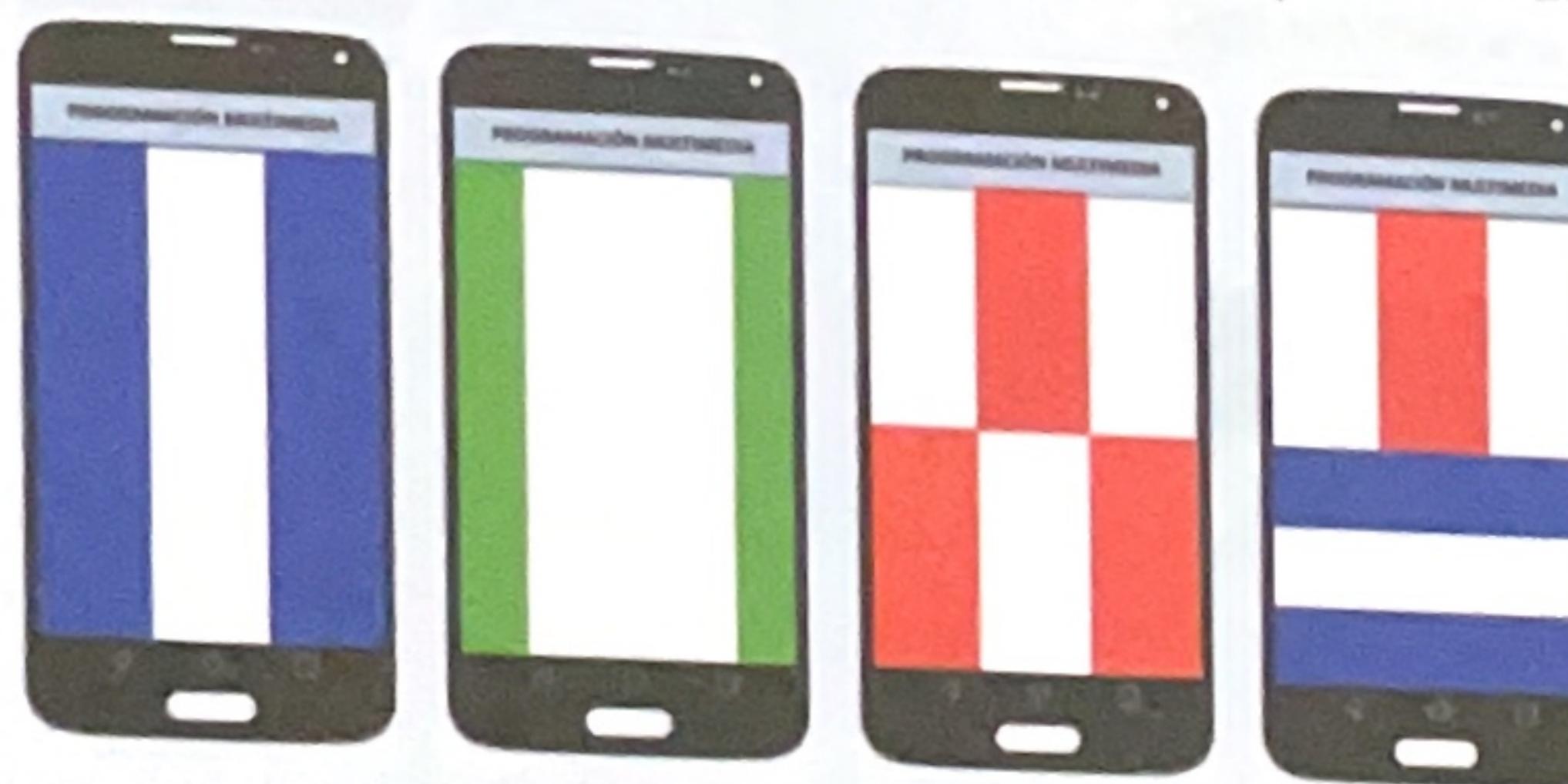


Figura 4.4
Ubicación en FrameLayouts.



Actividades propuestas

- 4.1.** Realiza con LinearLayout la siguiente estructura de layouts anidados.



- 4.2.** Realiza una aplicación con una actividad cuyo FrameLayout principal tenga nueve FrameLayouts interiores y muestren la disposición de la figura 4.4.

4.6. AbsoluteLayout

Permite indicar las coordenadas absolutas de pantalla (x-y) donde se quiere visualizar cada elemento:

android:layout_x="posición horizontal"

android:layout_y="posición vertical"

Los fabricantes crean dispositivos de muy variada resolución, por lo que Android nos permite un rango de resoluciones de pantalla muy variado, lo que hace que el contenedor distribuya los elementos que acoge según sus propias reglas, y que el resultado pueda no ser el deseado. Por ello, no es recomendable utilizar este tipo de contenedor, que hoy día ha sido marcado como obsoleto.

4.7. RelativeLayout

En este layout los elementos van colocados en una posición relativa unos a otros, especificando la posición de cada elemento de forma relativa a su elemento padre o a cualquier otro elemento incluido en el propio layout. Las opciones de posicionamiento disponibles son las siguientes:

Posición relativa al control

- android:layout_above
- android:layout_below
- android:layout_toLeftOf
- android:layout_toRightOf

Alineación con respecto al control

- android:layout_alignLeft
- android:layout_alignRight
- android:layout_alignTop
- android:layout_alignBottom
- android:layout_alignBaseline

Posición relativa al layout padre

- android:layout_alignParentLeft
- android:layout_alignParentRight
- android:layout_alignParentTop
- android:layout_alignParentBottom
- android:layout_centerHorizontal
- android:layout_centerVertical
- android:layout_centerInParent

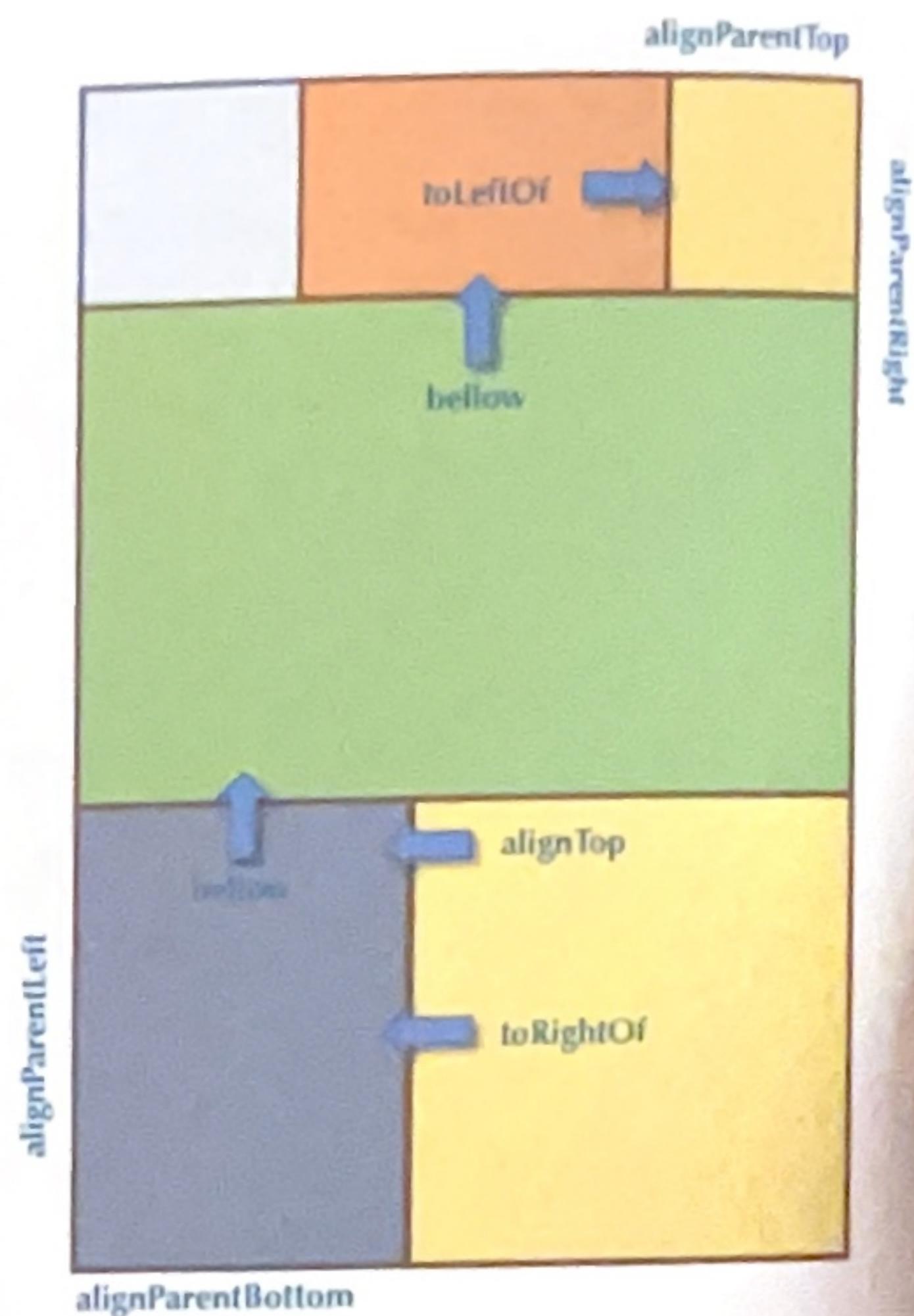


Figura 4.5
Relaciones
en un
RelativeLayout.

Actividad propuesta 4.3



El código para posicionar los dos primeros layouts (fila superior) de la figura 4.5 es el que se expone a continuación. Realiza un proyecto cuya vista, además de estos, tenga todos los mostrados en dicha figura.

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/padre">
    <RelativeLayout
        android:id="@+id/hijo1"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:background="#ff0000">
    </RelativeLayout>
    <RelativeLayout
        android:id="@+id/hijo2"
        android:layout_width="200dp"
        android:layout_height="100dp"
        android:layout_alignParentTop="true"
        android:layout_toLeftOf="@+id/hijo1"
        android:background="#0000ff">
    </RelativeLayout>
</RelativeLayout>
```

4.8. TableLayout

Este layout permite distribuir sus elementos hijos de forma tabular, definiendo las filas y columnas necesarias, y la posición de cada componente dentro de la tabla. Realmente, esta modalidad es un tipo especial de `LinearLayout`, con una funcionalidad especial al añadirle `TableRow`, lo que se asemejaría a un `LinearLayout` horizontal dentro de un `LinearLayout` vertical.

Se podría simular un `TableLayout` mediante varios `LinearLayout`, pero la diferencia entre uno y otro se encuentra en el tratamiento global que se le da a las vistas que se incluyen en todos los `TableRow`.

Este tipo de layout no solo tendría las propiedades del `LinearLayout` (aunque, en este caso, `android:orientation` carecería de sentido), sino que también tendría algunas propiedades exclusivas, como:

- `android:shrinkColumns`: permite indicar las columnas que se pueden contraer para dejar espacio al resto de columnas, adaptándose al tamaño del contenedor. Si se desea permitir que se contraigan todas las columnas de la tabla, se utilizará el valor “*”. Para igualar el aspecto de la tabla al contraer columnas, pueden utilizarse los atributos `android:lines`, `android:scrollHorizontally` o `android:ellipsize`.
- `android:stretchColumns`: esta propiedad nos indica las columnas que se pueden expandir para absorber el espacio libre dejado por las demás columnas incluidas en el contenedor. Los parámetros que podremos utilizar serán los mismos que los utilizados en el caso anterior.

```
<TableLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:background="#cdced4"
    android:useDefaultMargins="true">

    <TableRow>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:background="#0000ff" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:background="#0000ff" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:background="#0000ff" />
    </TableRow>

    <TableRow>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </TableRow>

```

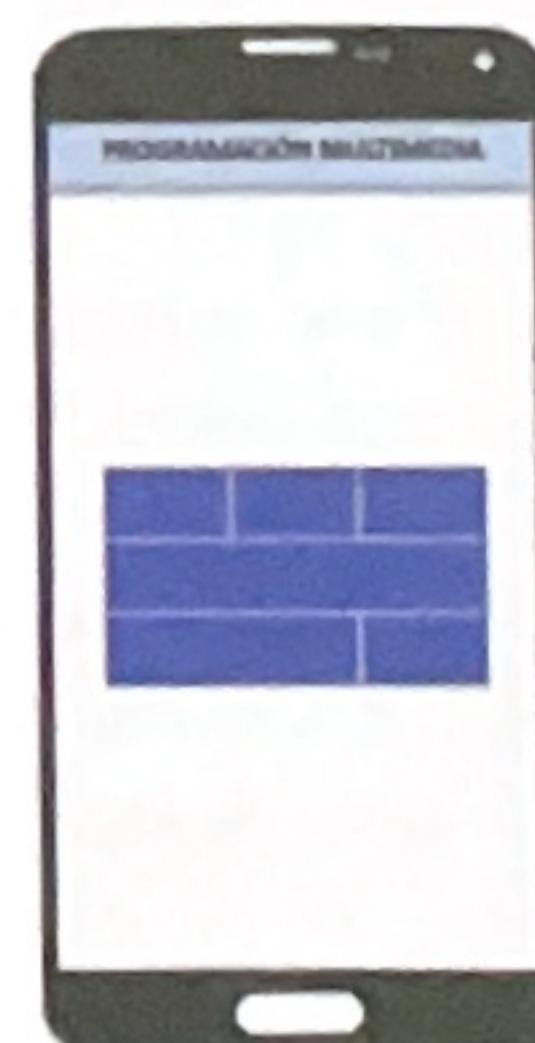


Figura 4.6
`TableRow`.

```

        android:layout_margin="2dp"
        android:layout_span="3"
        android:background="#0000ff" />

    <TableRow>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:layout_span="2"
            android:background="#0000ff" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:background="#0000ff" />
    </TableRow>
</TableLayout>

```

Además de lo anterior, hay otros dos interesantes parámetros aplicables a TableRow:

- *android:layout_span*: posibilita que una celda determinada pueda ocupar el espacio de varias columnas de la tabla.
- *android:layout_column*: mueve la vista a una columna diferente a la que le correspondería, según el orden en el que se ha incluido en el TableRow (las columnas se numeran a partir del 0).

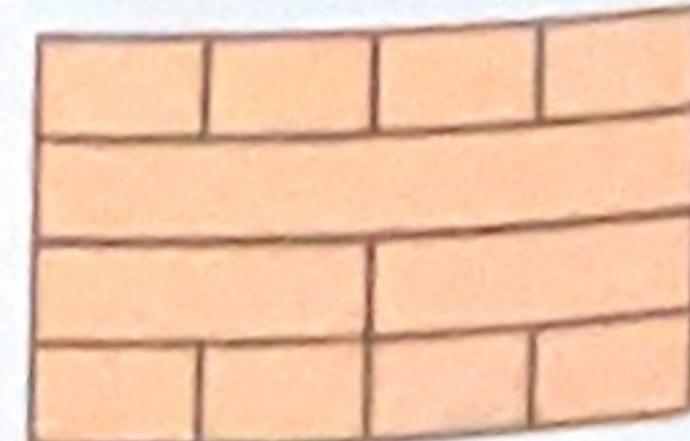
RECUERDA

- ✓ Con *android:layout_margin* se puede poner el mismo margen a los cuatro lados de una celda. Si solo se quiere hacer en uno de ellos, se puede utilizar *layout_marginBottom*, *layout_marginTop*, *layout_marginLeft*, *layout_marginRight*.
- ✓ Es posible modificar el aspecto de las celdas definiendo el *Background* en un fichero XML situado en los recursos (*res/xml*). En él se podrán incluir, además de colores, tipos de líneas o las formas (*shape*) con las que se presentará cada celda.

Actividad propuesta 4.4



Realiza una aplicación con una actividad cuya vista contenga un TableLayout con la estructura mostrada a la derecha. Puedes usar los botones como objetos incluidos en el interior de la tabla (siguiendo el ejemplo anterior).



4.9. GridLayout

Incluido a partir de la API 14 (Android 4.0), similar al TableLayout, utiliza un interfaz de forma tabular, distribuido en filas y columnas. A diferencia que en el TableLayout, se indica el número de filas y columnas como propiedades del layout, mediante android:rowCount y android:columnCount.

```
<GridLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:background="#cdced4"
    android:columnCount="3"
    android:orientation="horizontal"
    android:rowCount="4"
    android:useDefaultMargins="true">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#0000ff" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_rowSpan="2"
        android:layout_gravity="fill"
        android:background="#0000ff" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_columnSpan="3"
        android:layout_gravity="fill"
        android:background="#0000ff" />
</GridLayout>
```

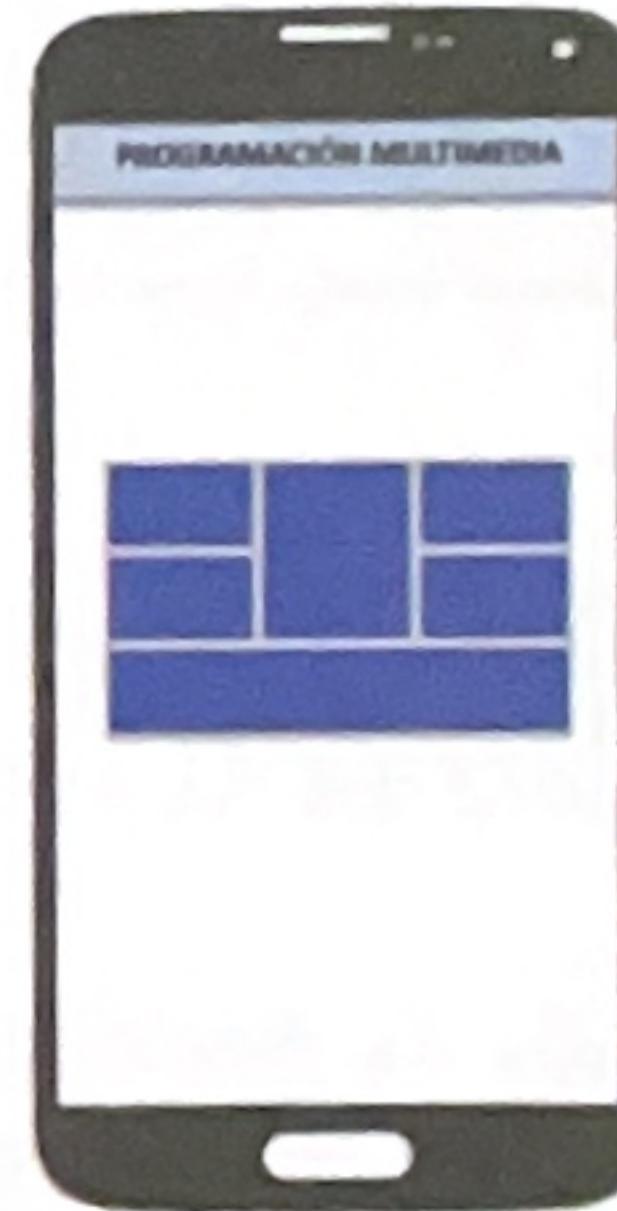
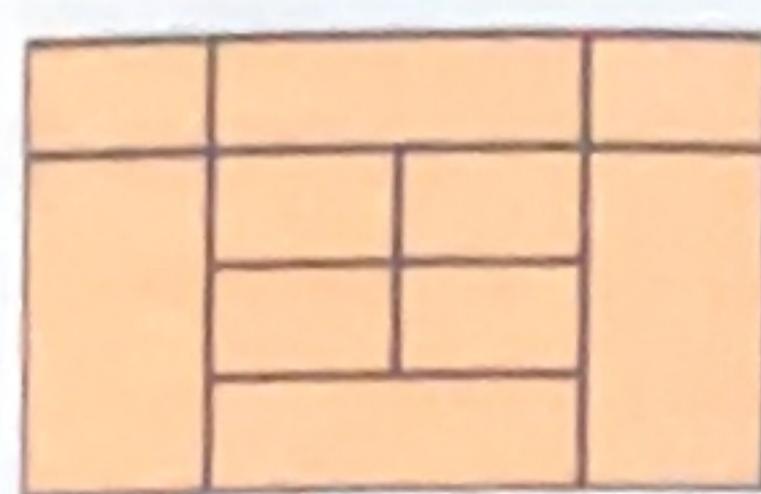


Figura 4.7
GridLayout.

- *android:layout_orientation*: marca la forma de colocar los elementos hijos, por filas o columnas.
- *android:layout_columnSpan*: posibilita que una celda pueda ocupar el espacio de varias columnas.
- *android:layout_rowSpan*: hace que una celda pueda ocupar el espacio de varias filas de la tabla.

Actividad propuesta 4.5

Realiza una aplicación con una actividad cuya vista contenga un GridLayout con la estructura mostrada a la derecha. Puedes usar los botones como objetos incluidos en el interior de la tabla (siguiendo el ejemplo anterior).

**Recurso web**

En este código QR encontrarás un tutorial de Shane Conder & Lauren Darcey para realizar el diseño de un teclado numérico mediante GridLayout.

**4.10. ConstraintLayout**

Con la llegada de Android Studio 2.2, Google presentó este nuevo layout con el que cambia radicalmente la forma de diseñar interfaces gráficas. ConstraintLayout permite simplificar el anidamiento de interfaces con un diseño visual para el que utiliza herramientas drag and drop.

Siguiendo el estilo del RelativeLayout, los objetos se posicionan en relación a objetos hermanos y padres, pero es más flexible y más sencillo de usar mediante la vista diseño de Android Studio. Para poder utilizar este modelo de layout se deberá incorporar su propia librería, que deberá añadirse al proyecto como una dependencia, sin embargo, Android Studio usa ConstraintLayout por defecto, por lo que esta librería ya viene incorporada a los nuevos proyectos.

**Recurso web**

En este código QR puedes ver el vídeo oficial de presentación de ConstraintLayout:



Se diseñan los interfaces trabajando con la vista Blueprint, que permite ver las conexiones (Constraints) entre los objetos. Al igual que en el RelativeLayout, la posición de las vistas será relativa a otro objeto de la vista o al propio layout, así siempre se ajustarán correctamente al tamaño de la pantalla, independientemente del dispositivo.

Además del ajuste de tamaño que se ha visto hasta ahora (wrap_content y match_parent), el ConstraintLayout permite ajustar las dimensiones mediante un aspect ratio.

Los cuadrados de las esquinas posibilitan cambiar el tamaño de la vista. Los círculos (manejador de restricción) permiten establecer las conexiones.

Una vista tiene que tener al menos una restricción vertical y una horizontal. Si se omite la restricción horizontal, la vista se mostrará alineada a la izquierda. Si se omite la restricción vertical, la vista se mostrará en la parte superior, independientemente de la posición de la vista en el Blueprint. Este es un ejemplo de panel de diseño en Android Studio (Design).

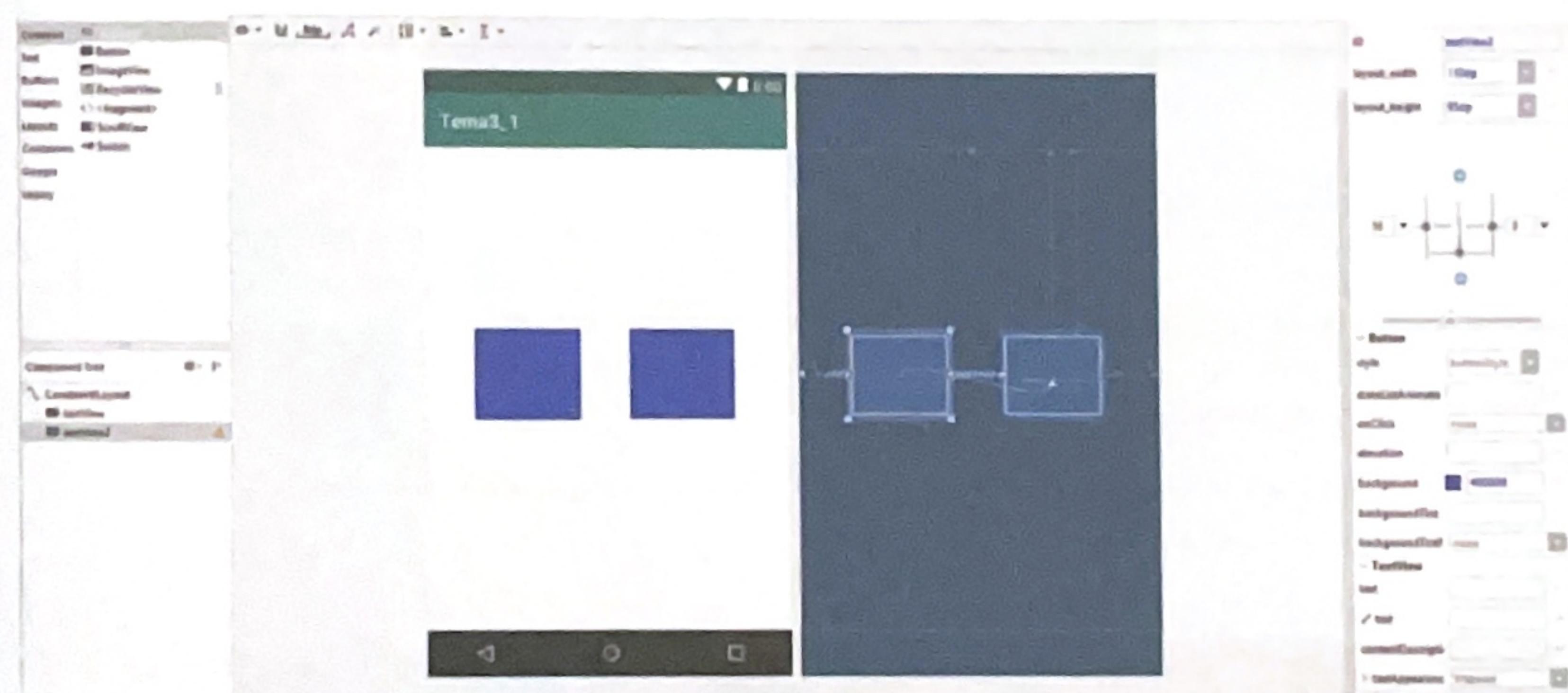


Figura 4.8
Vista diseño de ConstraintLayout en Android Studio.

La pantalla de diseño de la figura 4.8 correspondería con el siguiente código y captura de pantalla.

```
<android.support.constraint.ConstraintLayout
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/bot1"
        android:layout_width="110dp"
        android:layout_height="95dp"
        android:layout_marginLeft="16dp"
        android:layout_marginTop="192dp" />

```

```

        android:layout_marginRight="16dp"
        android:background="#0000ff"
        app:layout_constraintHorizontal_bias="0.825"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    
```

```

<Button
    android:id="@+id/boton2"
    android:layout_width="110dp"
    android:layout_height="95dp"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="8dp"
    android:background="#0000ff"
    app:layout_constraintBaseline_toBaselineOf="@+id/bot1"
    app:layout_constraintHorizontal_bias="0.435"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toLeftOf="@+id/bot1" />

```

```

</android.support.constraint.ConstraintLayout>

```



Figura 4.9
Diseño
ConstraintLayout



PARA SABER MÁS

En el siguiente código QR encontrarás información sobre el uso del editor de diseño para layout que incorpora Android Studio.



Resumen

- Un layout es un elemento destinado a controlar la distribución, posición y dimensiones de los objetos que se ubican en su interior. Este pertenece a la clase ViewGroup que, a su vez, pertenece a la clase View.
- La clase View como clase principal tiene una serie de atributos que serán heredados por todos los elementos que de ella provienen: atributos de posicionamiento, márgenes y espaciado, entre otros.
- En función de la forma y posicionamiento en pantalla, existe gran variedad de layouts que serán utilizados según la necesidad de cada aplicación.
- LinearLayout apila sus objetos interiores de forma horizontal o vertical, según se establezca el atributo de orientación. FrameLayouts coloca todos sus controles hijos alineados con su esquina superior izquierda, pudiendo modificar la posición con la propiedad gravity.
- AbsoluteLayout indica las coordenadas absolutas de pantalla donde queremos que se visualice cada elemento. RelativeLayout coloca los elementos que contiene, especificando la posición de cada elemento de forma relativa al elemento padre o a cualquier otro elemento incluido en el layout.

- TableLayout distribuye sus elementos hijos de forma tabular, definiendo filas, columnas y posición de cada componente dentro de la tabla. GridLayout es similar al anterior, indicándose previamente el número de filas y columnas del layout.
- Un nuevo tipo de contenedor aparece con ConstraintLayout, que permite simplificar el anidamiento mediante un interface visual que utiliza herramientas tipo drag and drop.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cuál de los siguientes términos indica al interfaz que el contenedor debe ajustarse a su tamaño?:
 a) wrap_content.
 b) match_parent.
 c) fill_parent.
 d) Ninguno de los anteriores.
2. ¿Qué atributo distribuye los objetos en el contenedor según el espacio disponible?:
 a) layout_weight.
 b) layout_margin.
 c) layout_width.
 d) layout_gravity.
3. ¿Qué atributo permite colorear el fondo de un objeto ViewGroup?:
 a) android:background.
 b) android:layout_weight.
 c) android:visibility.
 d) android:layout_below.
4. Con respecto a LinearLayout, ¿cuál de las siguientes afirmaciones es falsa?:
 a) Apila a los objetos contenidos de forma horizontal o vertical.
 b) La forma de apilamiento viene marcada por el atributo orientación.
 c) No puede contener objetos de tipo ViewGroup.
 d) Puede hacer el contenido desplazable mediante ScrollView.
5. ¿Cuál de estas afirmaciones es cierta?:
 a) RelativeLayout posiciona a los elementos según coordenadas x-y.
 b) AbsoluteLayout coloca a todos sus objetos en la esquina superior izquierda.
 c) FrameLayout utiliza el atributo gravity para posicionar los objetos.
 d) Es recomendable el uso de AbsoluteLayout para Tablets.
6. En los RelativeLayouts, ¿cuál de estas opciones no se hace en relación al control?:
 a) android:layout_alignTop.
 b) android:layout_centerVertical.
 c) android:layout_toLeftOf.
 d) android:layout_below.

7. En el **TableLayout**, ¿qué atributo indica las **columnas** que se pueden contraer?:

- a) android:ellipsize.
- b) android:shrinkColumns.
- c) android:stretchColumns.
- d) android:layout_span.

8. ¿Qué atributo permite que una celda pueda ocupar el **espacio de varias columnas**?:

- a) android:ellipsize.
- b) android:shrinkColumns.
- c) android:stretchColumns.
- d) android:layout_span.

9. ¿Cuál de los siguientes atributos no le aplicarías a un **GridLayout**?:

- a) android:rowCount.
- b) android:layout_orientation.
- c) android:layout_rowSpan.
- d) Se podrían aplicar los tres.

10. Con respecto a **ConstraintLayout**, ¿cuál de estas afirmaciones es falsa?:

- a) Aparece con Android Studio 1.6.
- b) Utiliza herramientas drag and drop.
- c) Trabaja con vista Blueprint.
- d) Permite ajustar las dimensiones mediante aspect ratio.

SOLUCIONES:

1. a b c d

2. a b c d

3. a b c d

4. a b c d

5. a b c d

6. a b c d

7. a b c d

8. a b c d

9. a b c d

10. a b c d