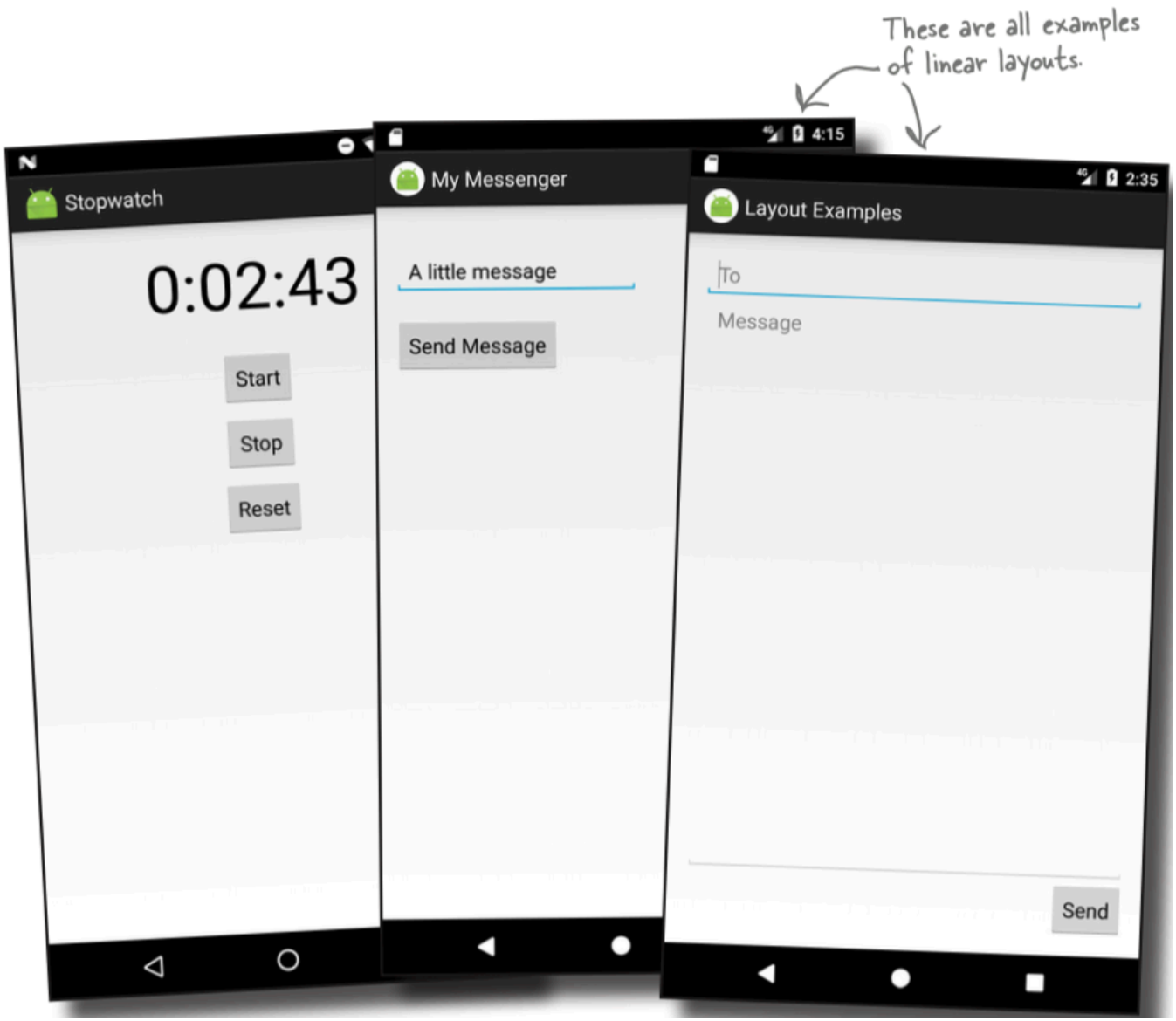
5
La interfaz de usuario

La interfaz de usuario está hecha de layouts y componentes



Los principales layouts

- Relative layout
- Linear layout
- Grid layout
- Frame layout

Un RelativeLayout presenta las vistas en posiciones relativas

This tells Android you're using a relative layout.

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

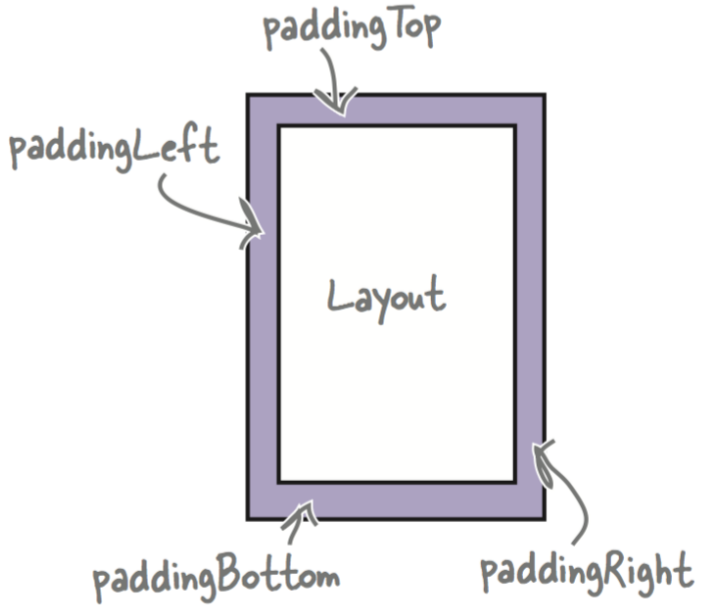
The layout_width and layout_height specify what size you want the layout to be.

There may be other attributes too.

Agregando espacio entre componentes (padding)

```
<RelativeLayout ...
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp">
    ...
</RelativeLayout>
```

Add padding of 16dp.



Agregando espacio entre componentes (padding)

```
<RelativeLayout ...
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin">
```

The paddingLeft and paddingRight attributes are set to @dimen/activity_horizontal_margin.

The paddingTop and paddingBottom attributes are set to @dimen/activity_vertical_margin.

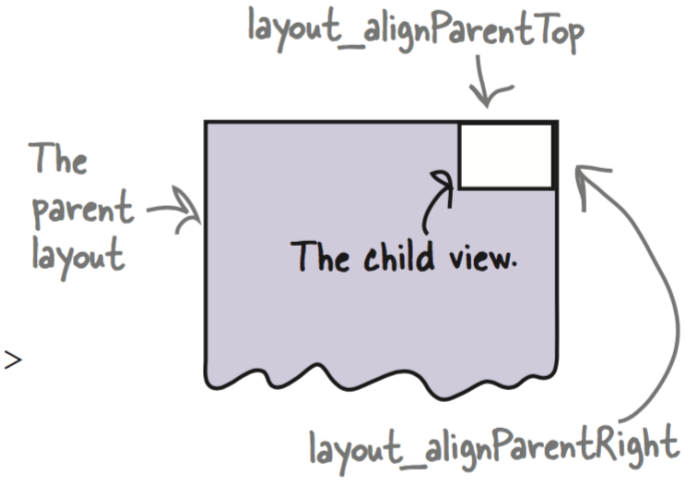
```
<resources>
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```

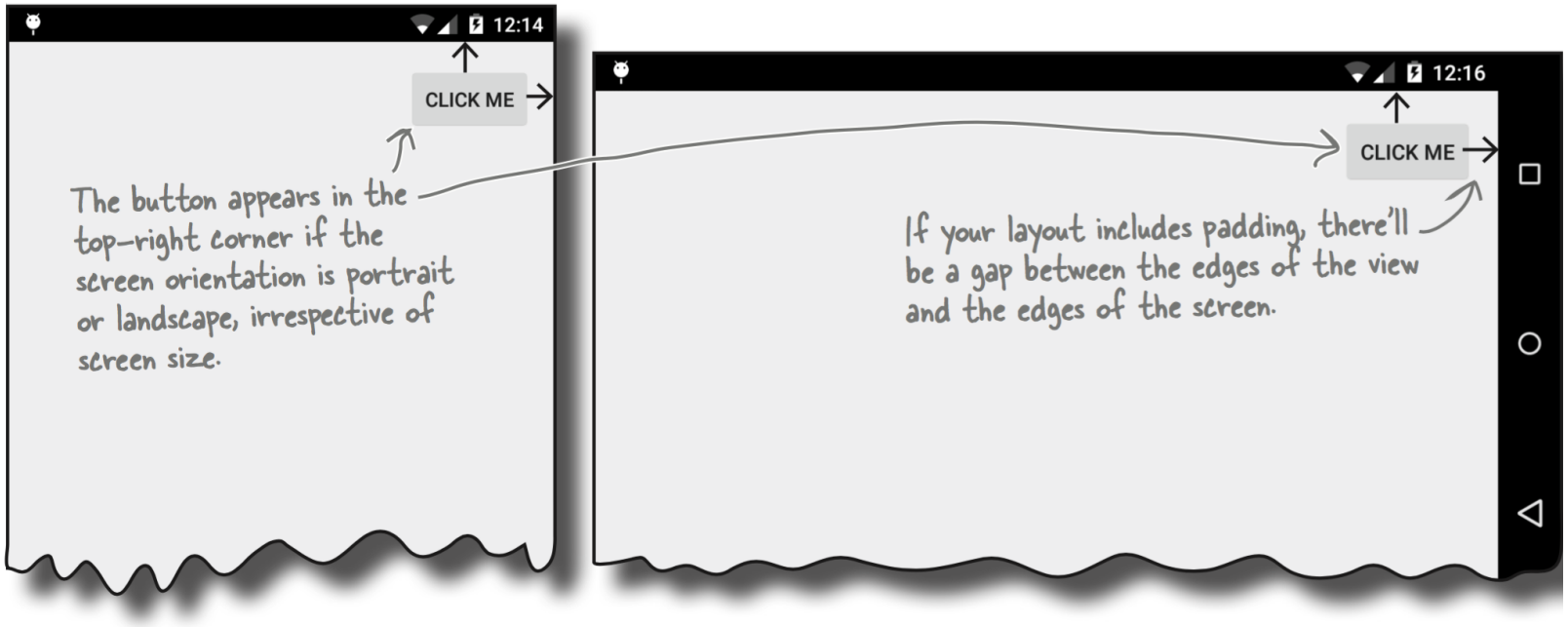
The layout looks up the padding values from these dimen resources.

Colocando las vistas relativas al layout padre

```
<RelativeLayout ... >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/click_me"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
</RelativeLayout>
```

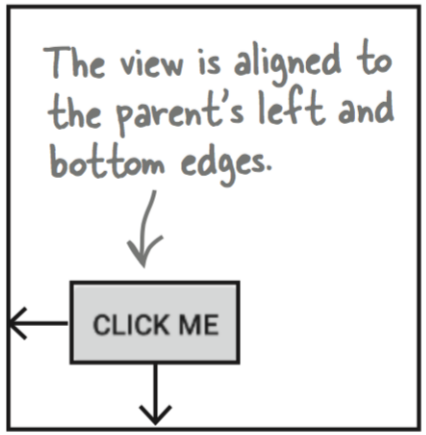
The layout contains the button, so the layout is the button's parent.

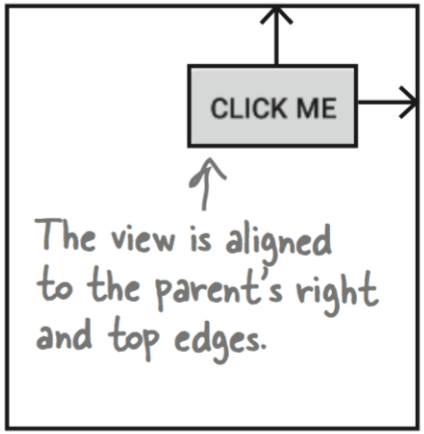




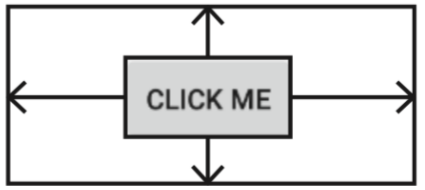
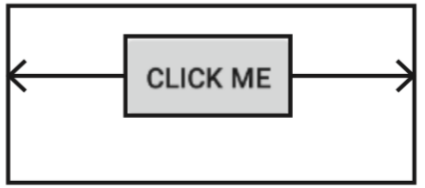
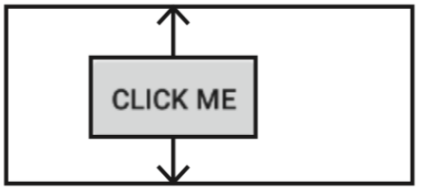
Atributos para colocar vistas relativas al layout padre

android: layout_alignParentBottom	Aligns the bottom edge of the view to the bottom edge of the parent.
android: layout_alignParentLeft	Aligns the left edge of the view to the left edge of the parent.
android: layout_alignParentRight	Aligns the right edge of the view to the right edge of the parent.
android: layout_alignParentTop	Aligns the top edge of the view to the top edge of the parent.





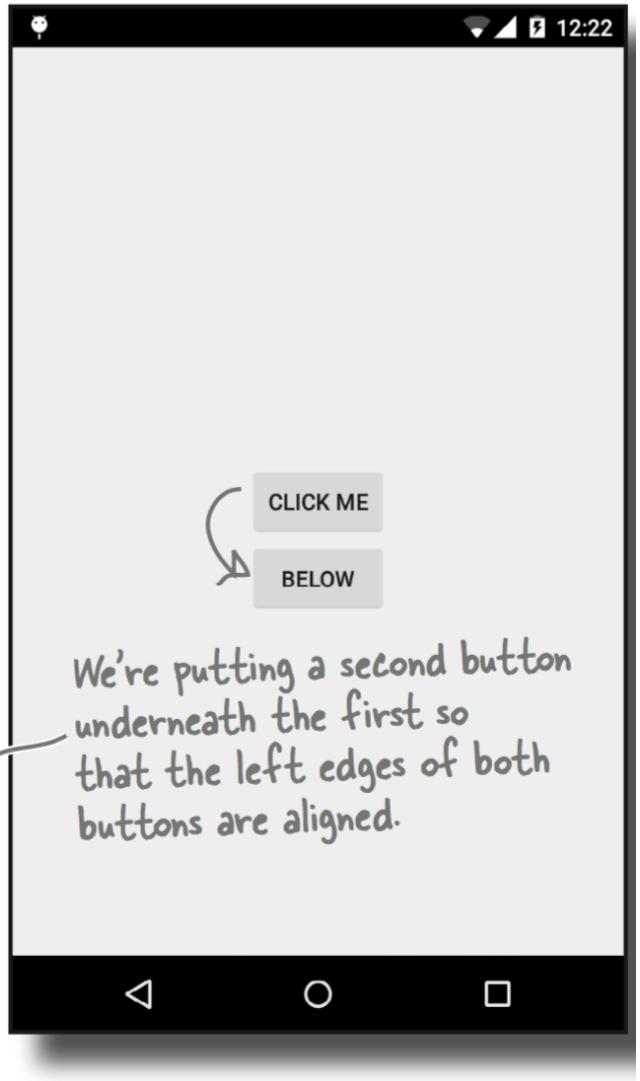
Atributos para colocar vistas relativas al layout padre

android: layout_centerInParent	Centers the view horizontally and vertically in the parent.	
android: layout_centerHorizontal	Centers the view horizontally in the parent.	
android: layout_centerVertical	Centers the view vertically in the parent.	

Colocando las vistas relativas a otros componentes

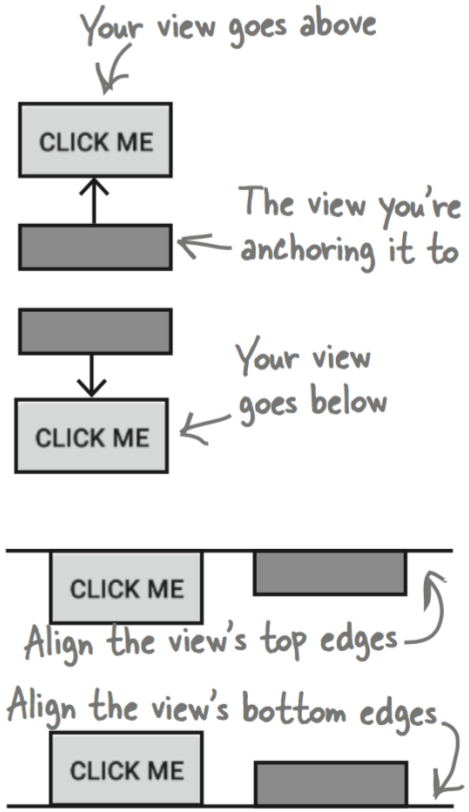
```
<RelativeLayout ... >
    <Button
        android:id="@+id/button_click_me"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="@string/click_me" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button_click_me"
        android:layout_below="@+id/button_click_me"
        android:text="@string/new_button_text" />
</RelativeLayout>
```

We're using this button as an anchor for the second one, so it needs an ID.



Atributos para colocar vistas relativas a otras vistas

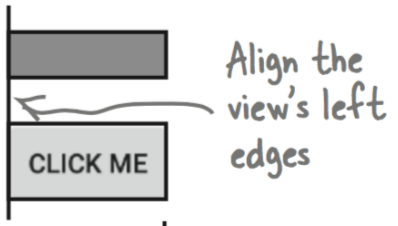
android:layout_above	Put the view above the view you're anchoring it to.
android:layout_below	Puts the view below the view you're anchoring it to.
android:layout_alignTop	Aligns the top edge of the view to the top edge of the view you're anchoring it to.
android:layout_alignBottom	Aligns the bottom edge of the view to the bottom edge of the view you're anchoring it to.



Atributos para colocar vistas relativas a otras vistas

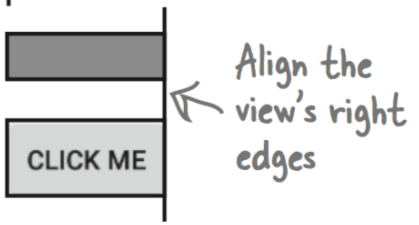
android:layout_alignLeft

Aligns the left edge of the view to the left edge of the view you're anchoring it to.



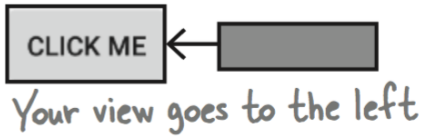
android:layout_alignRight

Aligns the right edge of the view to the right edge of the view you're anchoring it to.



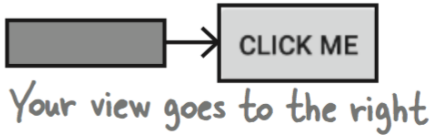
android:layout_toLeftOf

Puts the right edge of the view to the left of the view you're anchoring it to.



android:layout_toRightOf

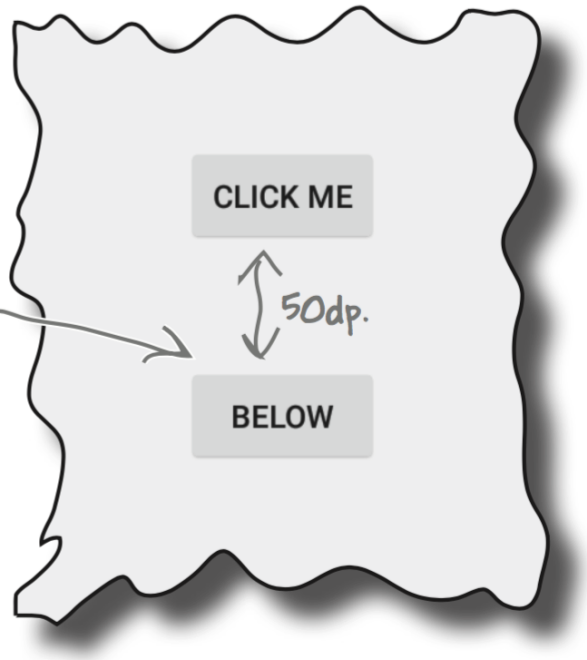
Puts the left edge of the view to the right of the view you're anchoring it to.



Utilizando márgenes para agregar distancia entre vistas

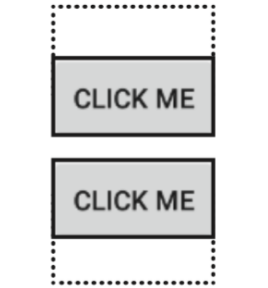
```
<RelativeLayout ... >
    <Button
        android:id="@+id/button_click_me"
        ... />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button_click_me"
        android:layout_below="@+id/button_click_me"
        android:layout_marginTop="50dp"
        android:text="@string/button_below" />
</RelativeLayout>
```

Adding a margin to the top of the bottom button adds extra space between the two views.



Atributos para agregar márgenes entre vistas

android:layout_marginTop	Adds extra space to the top of the view.
android:layout_marginBottom	Adds extra space to the bottom of the view.
android:layout_marginLeft	Adds extra space to the left of the view.
android:layout_marginRight	Adds extra space to the right of the view.



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

You use <LinearLayout> to define a linear layout.

These are the same attributes we used for our relative layout.

Display views vertically.

Un LinearLayout presenta las vistas en un sólo renglón o columna

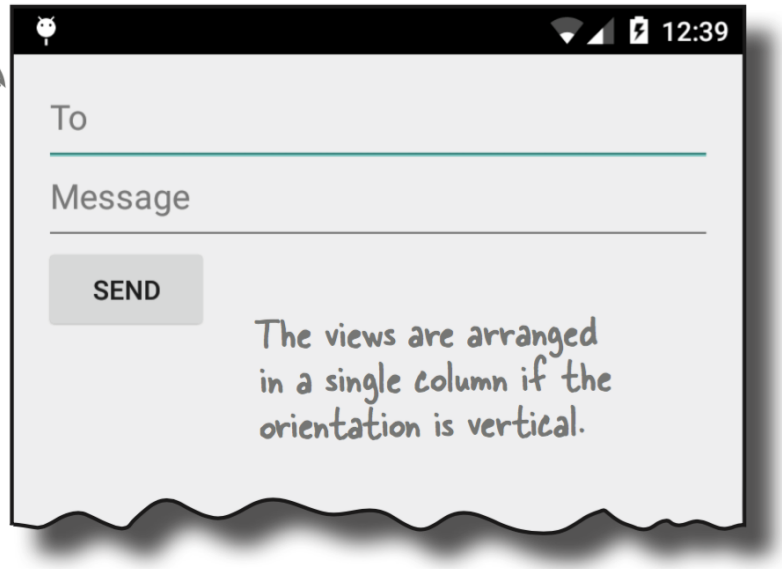
You arrange views vertically using:

```
android:orientation="vertical"
```

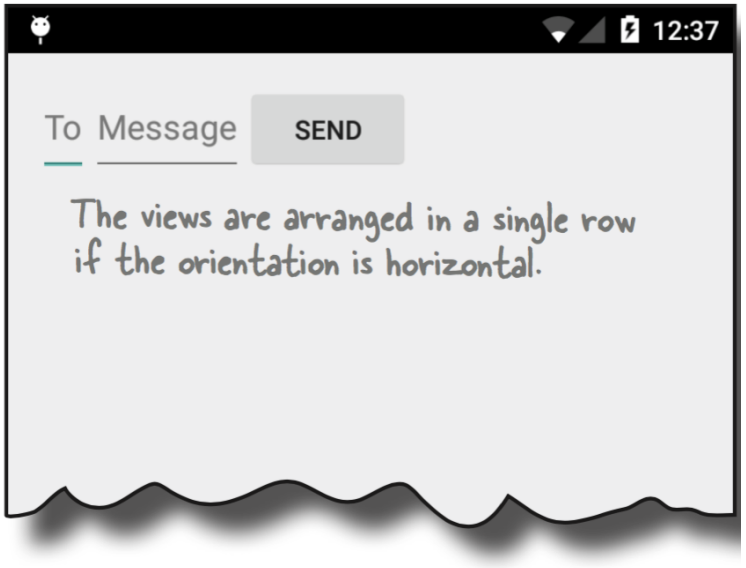
You arrange views horizontally using:

```
android:orientation="horizontal"
```

A linear layout with a vertical orientation.



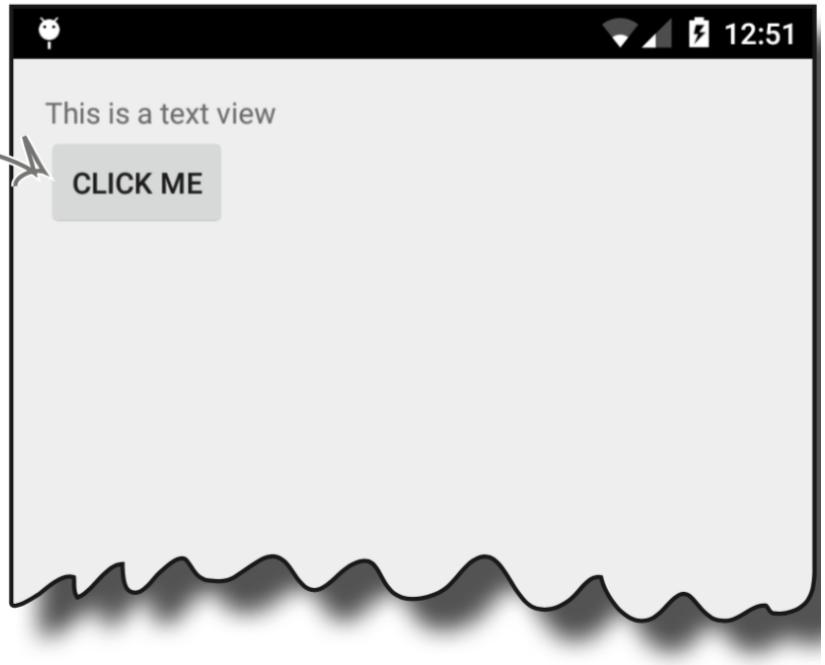
A linear layout with a horizontal orientation.



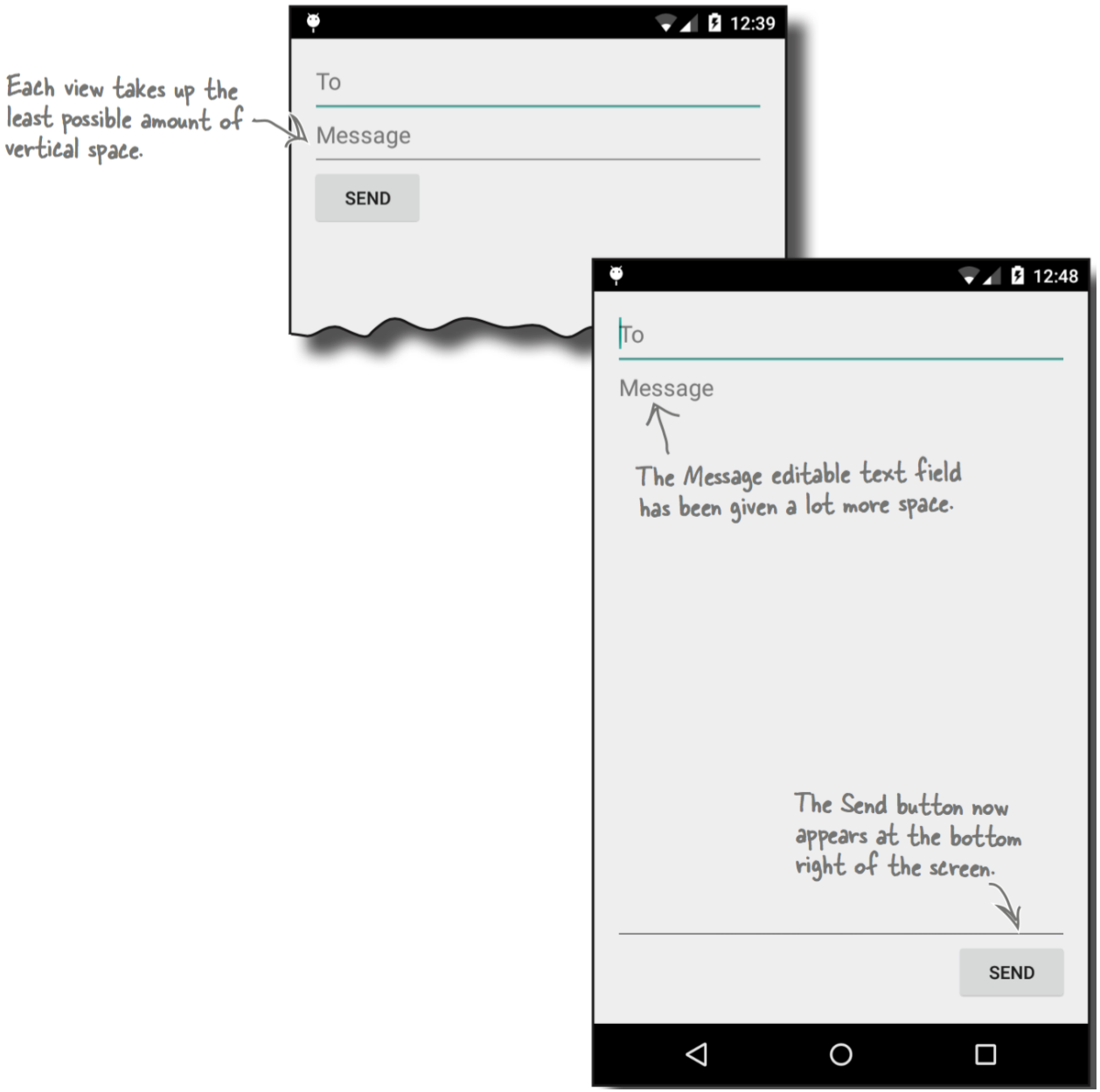
Un LinearLayout presenta las vistas en el orden en que aparecen en el código XML

```
<LinearLayout ... >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/textView1" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/click_me" />
</LinearLayout>
```

If you define the text view above the button in the XML, the text view will appear above the button when displayed.



Modificando un LinearLayout básico



El código XML original

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:orientation="vertical"
    tools:context=".MainActivity" >
```

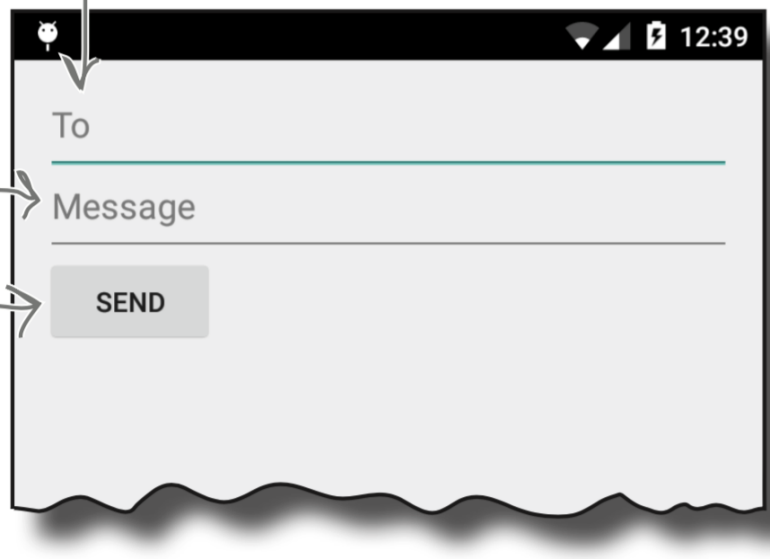
El código XML original

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/to" />

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/message" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/send" />
</LinearLayout>
```

The editable text fields are as wide as the parent layout.

android:hint displays a hint to the user as to what they should type in the editable text field.

The values of these strings are defined in Strings.xml as usual.



```
<LinearLayout ... >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:hint="@string/message" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/send" />
</LinearLayout>
```

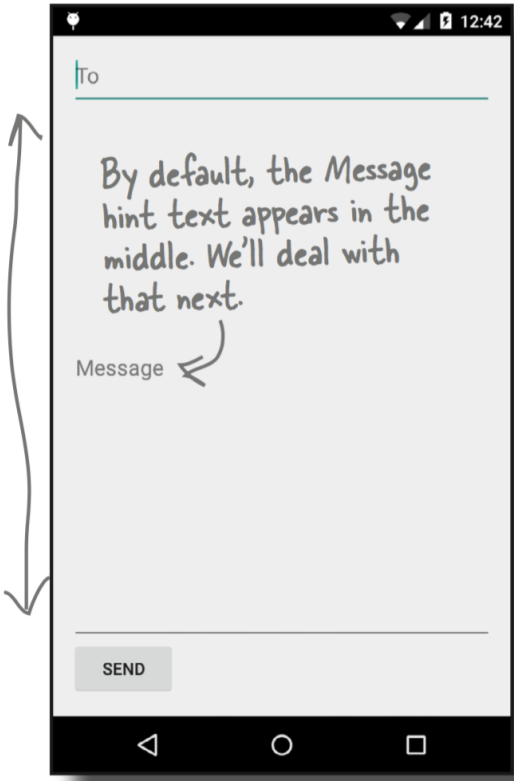
This <EditText> and the <Button> have no layout_weight attribute set. They'll take up as much room as their content needs, but no more.

This view is the only one with any weight. It will expand to fill the space that's not needed by any of the other views.

The height of the view will be determined by the linear layout based on the layout_weight. Setting the layout_height to 0dp is more efficient than setting it to "wrap_content".

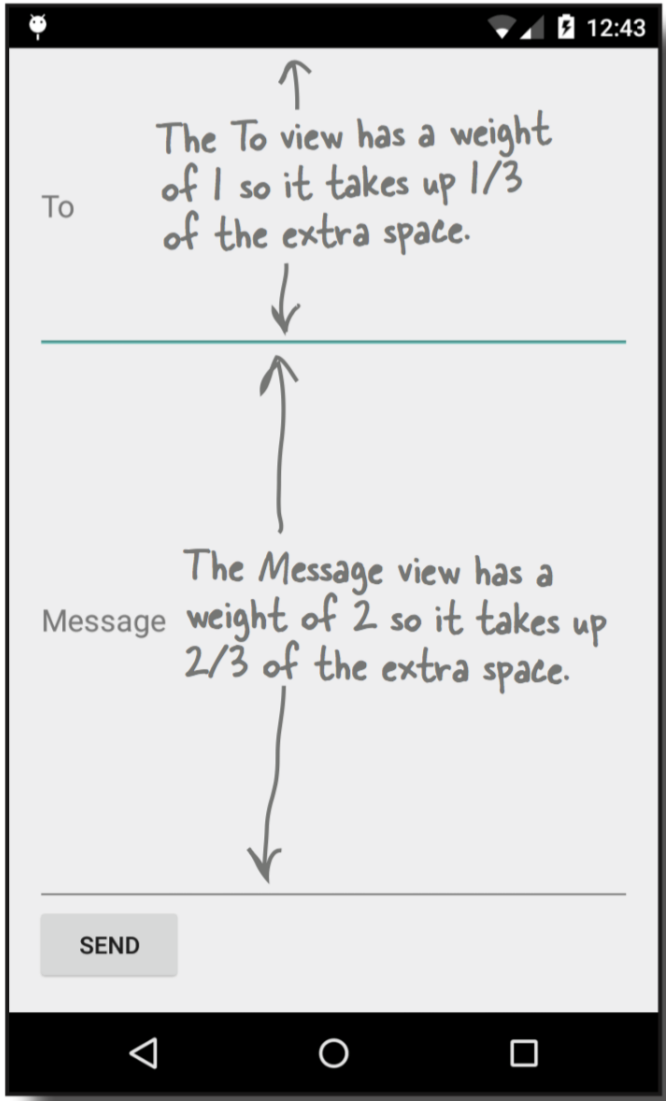
Alargando una vista al agregar peso

The Message view has a weight of 1. As it's the only view with its weight attribute set, it expands to take up any extra vertical space in the layout.

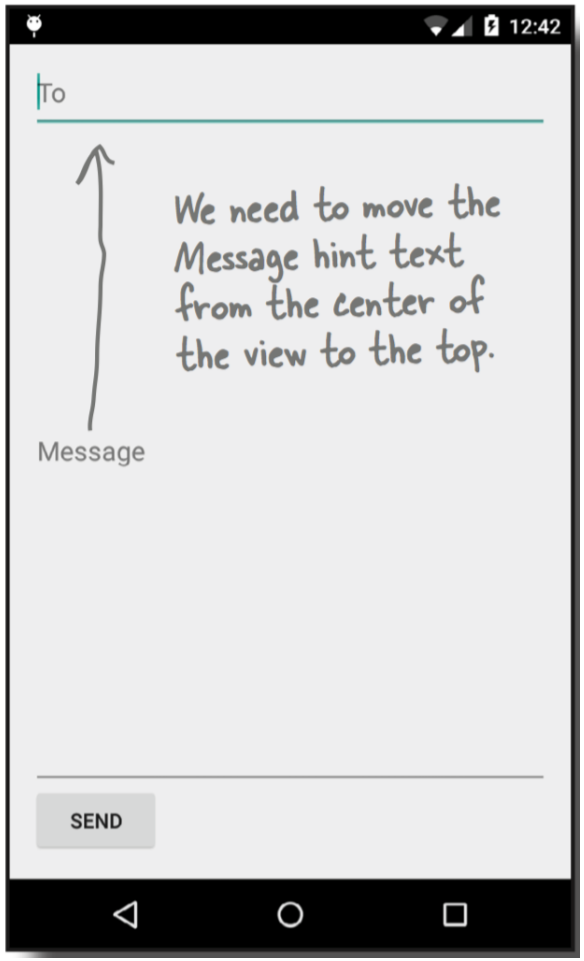


Agregando peso a múltiples vistas

```
<LinearLayout ... >
    ...
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:hint="@string/to" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:hint="@string/message" />
    ...
</LinearLayout>
```



Usando gravedad para especificar donde debe aparecer el texto




```
<LinearLayout ... >
    ...
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    ...
</LinearLayout>
```

Display the text inside the text field at the top of the text field.

Prueba

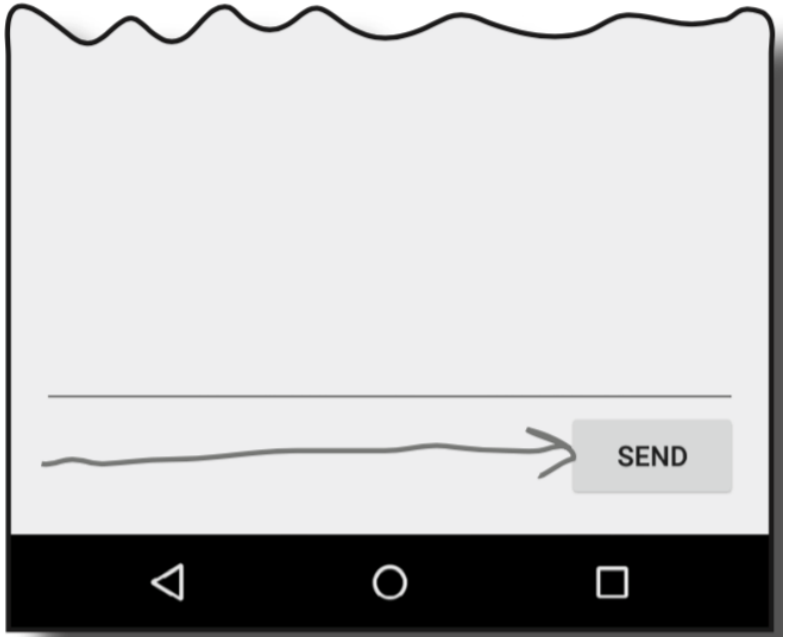

12:46
To
Message
The Message hint text now appears at the top of the view.
SEND

Valores para el atributo de gravedad (gravity)

top	Puts the view's contents at the top of the view.
bottom	Puts the view's contents at the bottom of the view.
left	Puts the view's contents at the left of the view.
right	Puts the view's contents at the right of the view.
center_vertical	Centers the view's contents vertically.
center_horizontal	Centers the view's contents horizontally.
center	Centers the view's contents vertically and horizontally.
fill_vertical	Make the view's contents fill the view vertically.
fill_horizontal	Make the view's contents fill the view horizontally.
fill	Make the view's contents fill the view.

Moviendo el botón a la derecha

```
android:layout_gravity="right"
```




```
android:layout_gravity="end"
```

Valores para el atributo de gravedad (layout_gravity)

top, bottom, left, right	Puts the view at the top, bottom, left, or right of its container.
start, end	Puts the view at the start or end of its container.
center_vertical, center_horizontal	Centers the view vertically or horizontally in its container.
center	Centers the view vertically and horizontally in its container.
fill_vertical, fill_horizontal	Grow the view so that it fills its container in a vertical or horizontal direction.
fill	Grow the view so that it fills its container in a vertical and horizontal direction.

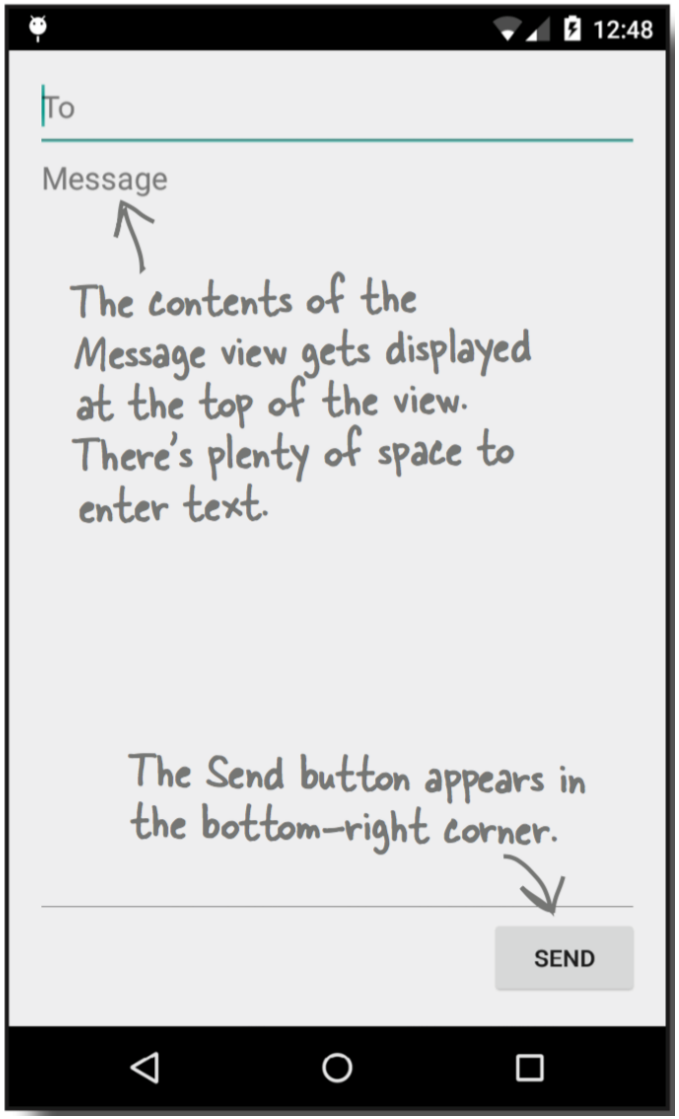
El código completo para el LinearLayout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
```

El código completo para el LinearLayout

```
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />

</LinearLayout>
```

android:gravity is different to android:layout_gravity. android:gravity relates to the contents of the view, android:layout_gravity relates to the view itself.



Un GridLayout presenta las vistas en una rejilla

Watch it!

GridLayout requires API level 14 or above.

If you plan on using a grid layout, make sure your app uses a minimum SDK of API 14.

You use <GridLayout> here.

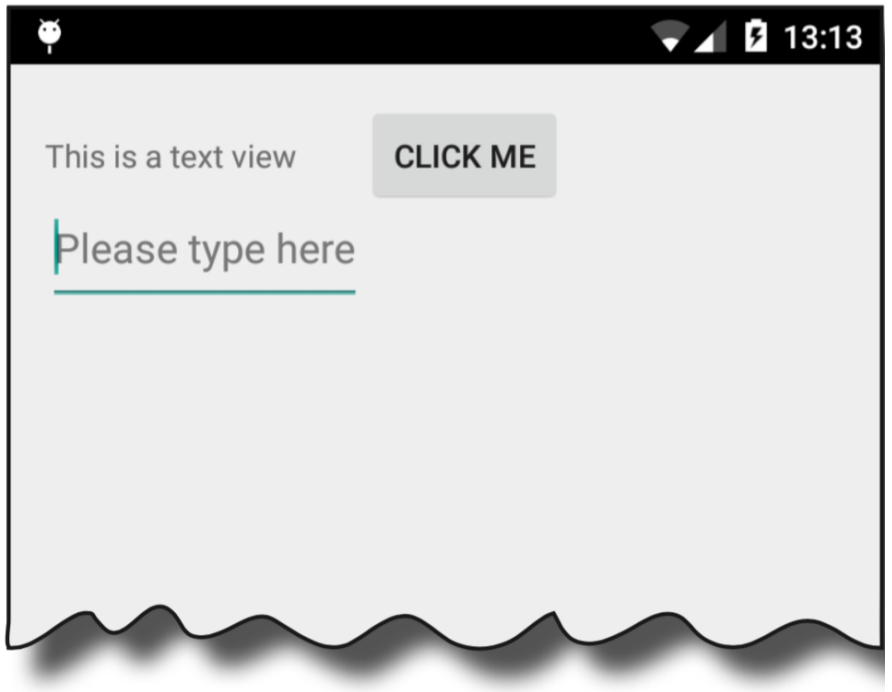
```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    ... >
    ...
</GridLayout>
```

These are the same attributes we used for our other layouts.

How many columns you want your layout to have (in this case, 2).

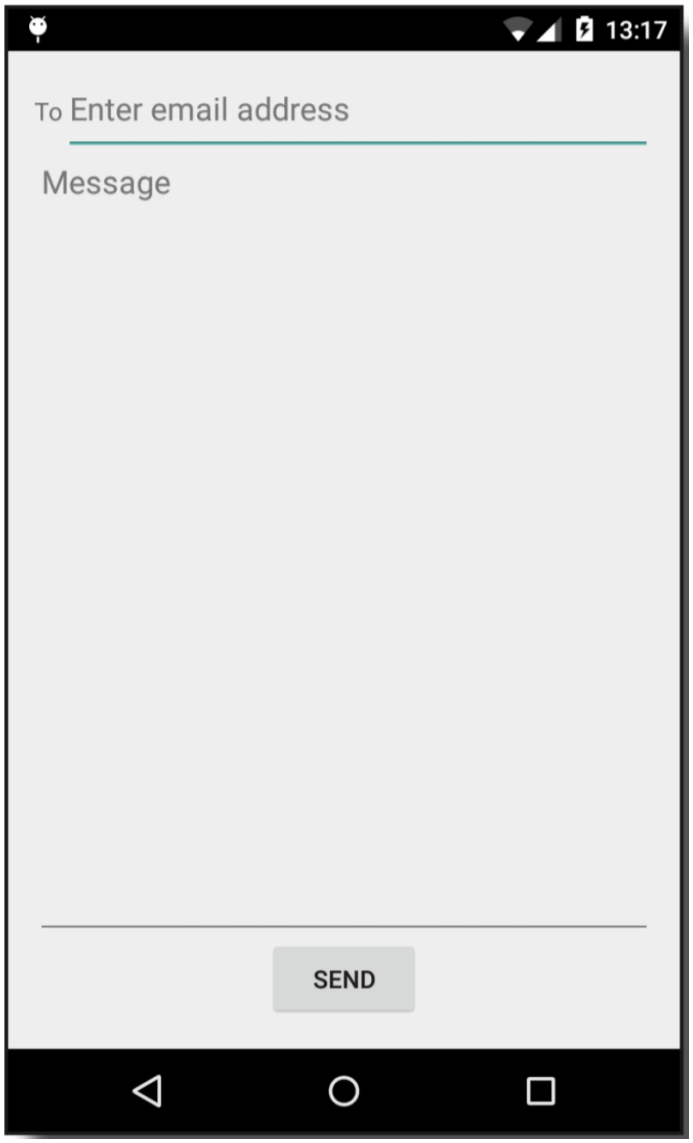
Agregando vistas a un GridLayout

```
<GridLayout ... >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/textview" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/click_me" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit" />

</GridLayout>
```



Creando un nuevo GridLayout

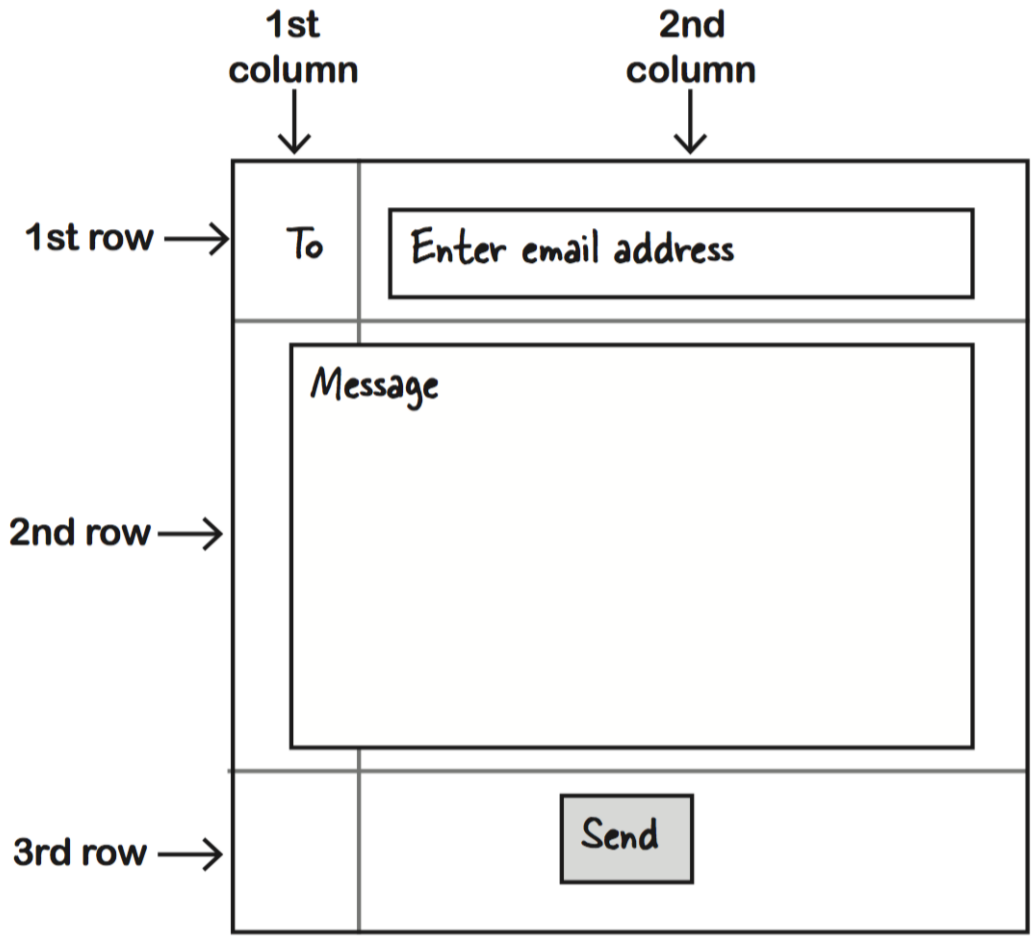
This is similar to the example we used with the linear layout, except that there's now a To text field at the top, and the Send button is centered horizontally at the bottom.



1. **Sketch the user interface, and split it into rows and columns.**
This will make it easier for us to see how we should construct our layout.

2. **Build up the layout row by row.**

Iniciando con un bosquejo



The first row has a text view in the first column with text of "To", and an editable text field in the 2nd column with a hint of "Enter email address".

The second row has an editable text field with text of "Message". It starts in the first column and spans across the second. It needs to fill the available space.

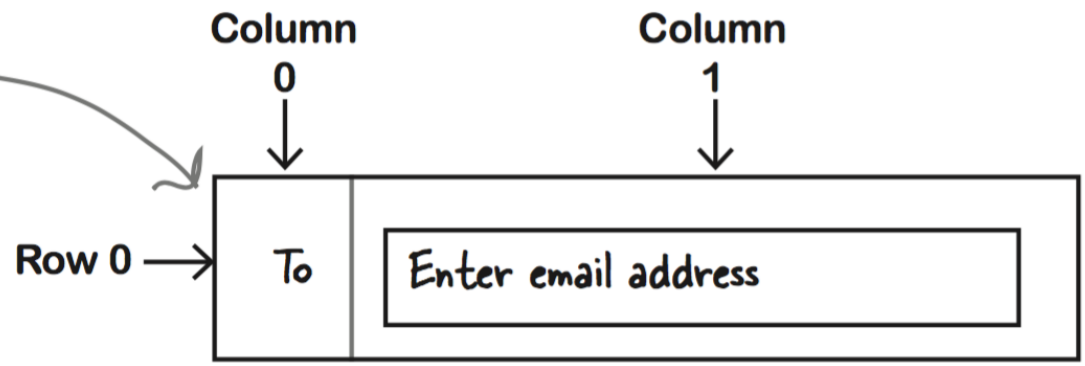
The third row has a button with text of "Send". It's centered horizontally across both columns, which means it needs to span the two columns.

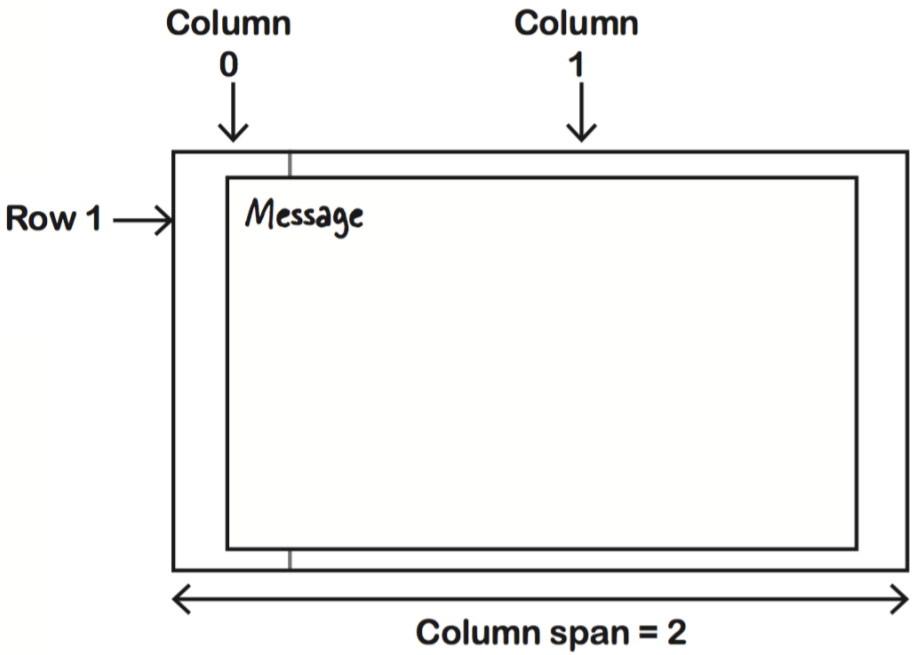
El GridLayout necesita 2 columnas

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:columnCount="2"
    tools:context=".MainActivity" >
</GridLayout>
```

Agregando vistas en el renglón 0

```
<GridLayout...>
    <TextView
        ...
        android:layout_row="0"
        android:layout_column="0"
        android:text="@string/to" />

    <EditText
        ...
        android:layout_row="0"
        android:layout_column="1"
        android:hint="@string/to_hint" />
</GridLayout>
```





Agregando vistas en el renglón 1

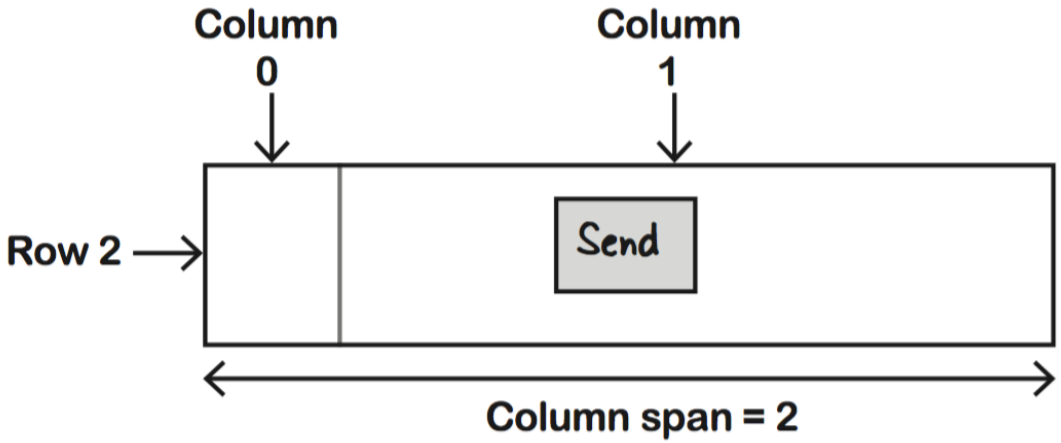
```
<GridLayout...>
    <TextView... />
    <EditText.../>
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill"
        android:gravity="top"
        android:layout_row="1"
        android:layout_column="0"
        android:layout_columnSpan="2"
        android:hint="@string/message" />
</GridLayout>
```

These are the views we added on the last page for row 0.

We want the view to fill the available space, and for the text to appear at the top.

The view starts in column 0, and spans 2 columns.

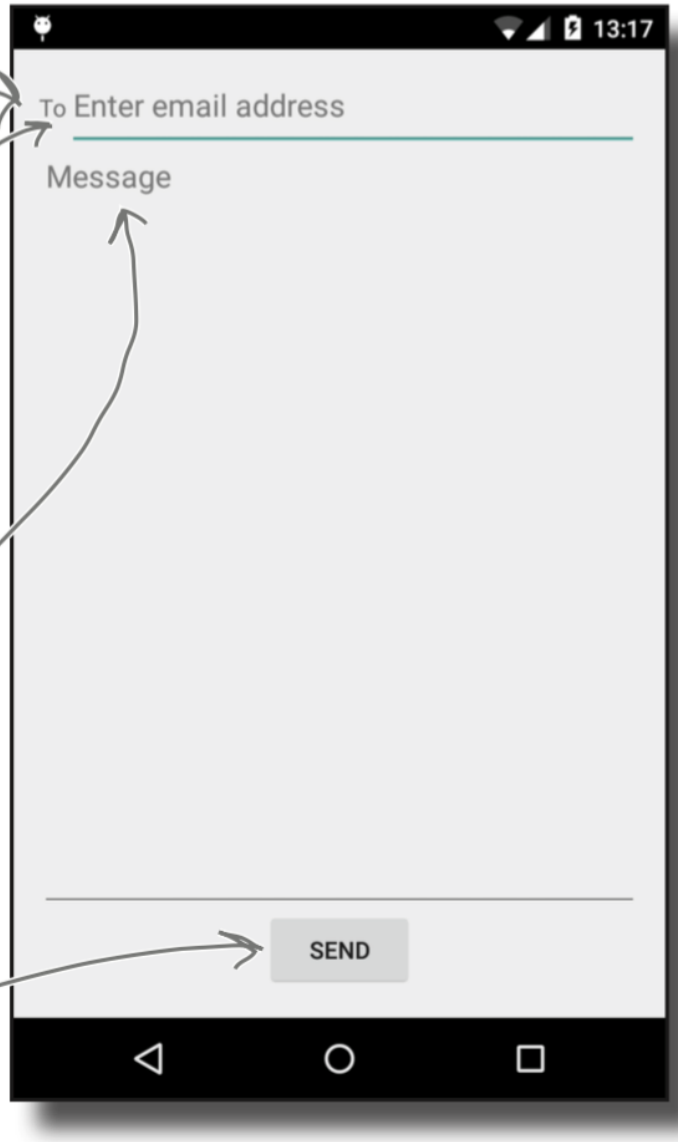
Agregando vistas en el renglón 2



El código completo para el GridLayout

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:columnCount="2"
    tools:context=".MainActivity" >
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_row="0"
    android:layout_column="0"
    android:text="@string/to" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="fill_horizontal"
    android:layout_row="0"
    android:layout_column="1"
    android:hint="@string/to_hint" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="fill"
    android:gravity="top"
    android:layout_row="1"
    android:layout_column="0"
    android:layout_columnSpan="2"
    android:hint="@string/message" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_row="2"
    android:layout_column="0"
    android:layout_gravity="center_horizontal"
    android:layout_columnSpan="2"
    android:text="@string/send" />
</GridLayout>
```

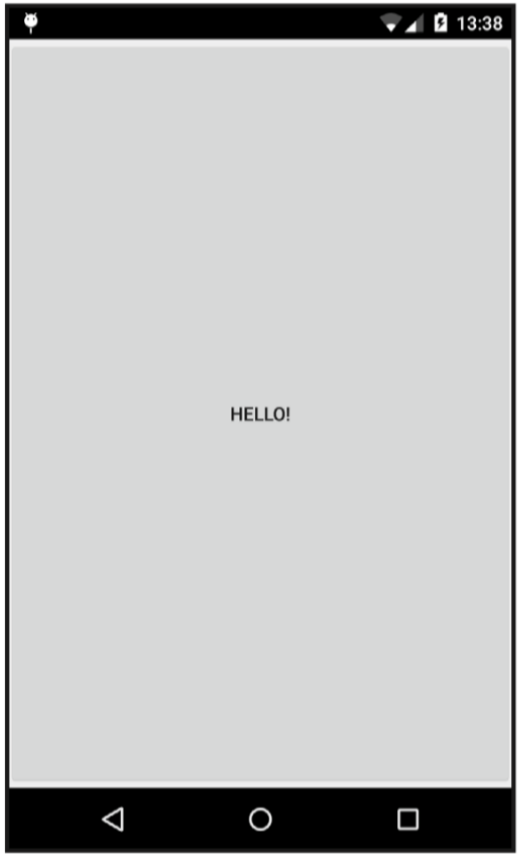


The button spans two columns, starting from row 2 column 1. It's centered horizontally.

El código completo para el GridLayout

Ejercicio

1



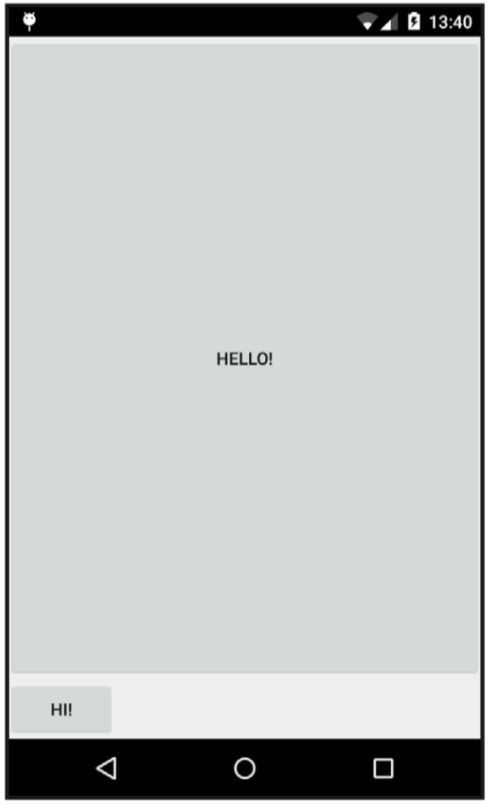
A

```
<GridLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="3"
    tools:context=".MainActivity" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill"
        android:layout_columnSpan="3"
        android:text="@string/hello" />
</GridLayout>
```

This has one button that fills the screen.

Ejercicio

3



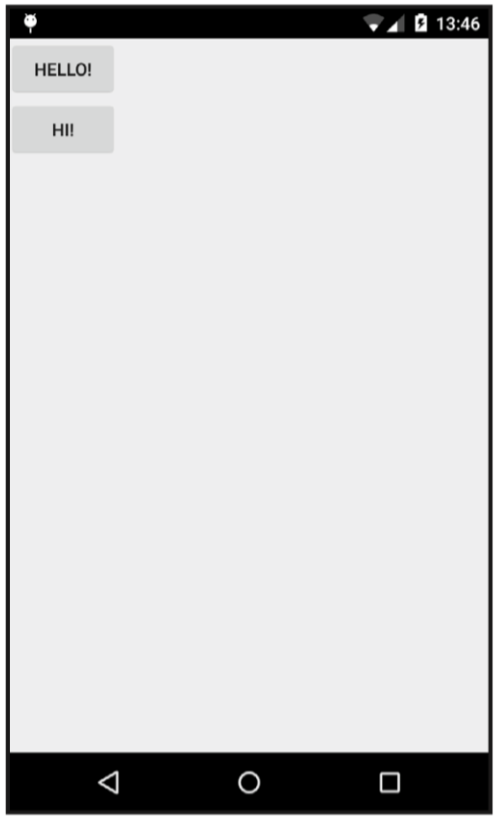
B

```
<GridLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    tools:context=".MainActivity" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill"
        android:layout_columnSpan="2"
        android:text="@string/hello" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hi" />
</GridLayout>
```

This button fills the screen, leaving space for another one underneath it.

Ejercicio

4



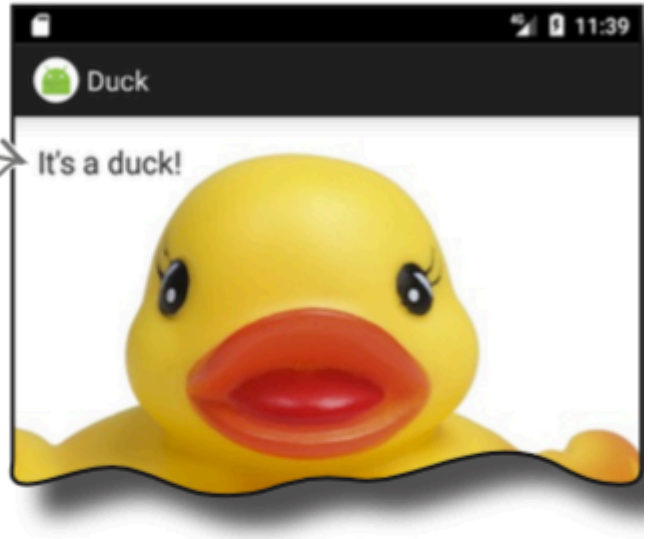
C

```
<GridLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    tools:context=".MainActivity" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="0"
        android:layout_column="0"
        android:layout_columnSpan="2"
        android:text="@string/hello" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="1"
        android:layout_column="0"
        android:text="@string/hi" />
</GridLayout>
```

Even though the button spans two columns, we didn't tell it to fill the screen horizontally.

Un frame layout coloca en una pila sus vistas

Frame layouts let your views overlap one another. This is useful for displaying text on top of images, for example.



You use <FrameLayout> to define a frame layout.

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

    ...

</FrameLayout>
```

These are the same attributes we used for our linear layout.

This is where you add any views you wish to stack in the frame layout.

Crear un nuevo proyecto llamado “Duck”, con la actividad principal llamada “MainActivity”, el layout llamado “activity_main” y desactivando la opción AppCompat.

Agregar una imagen al proyecto llamada “duck.jpg” en la carpeta *drawable*.

Modificando el diseño para utilizar un frame layout

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context="com.hfad.duck.MainActivity">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:scaleType="centerCrop"
        android:src="@drawable/duck" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:textSize="20sp"
        android:text="It's a duck!" />
</FrameLayout>
```

We're using a frame layout.

This adds an image to the frame layout. You'll find out more about image views later in the chapter.

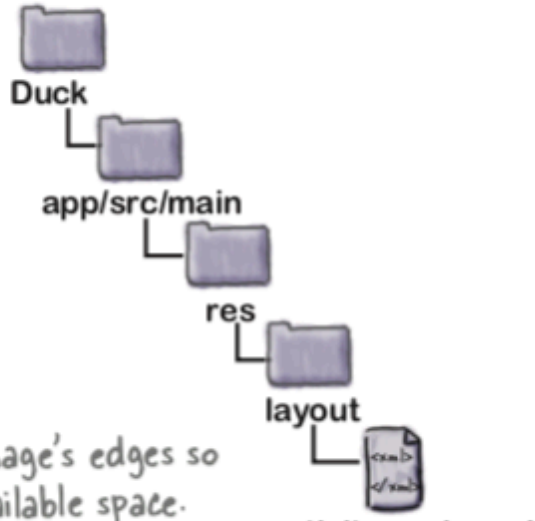
This crops the image's edges so it fits in the available space.

This line tells Android to use the image named "duck" located in the drawable folder.

This adds a text view to the frame layout.

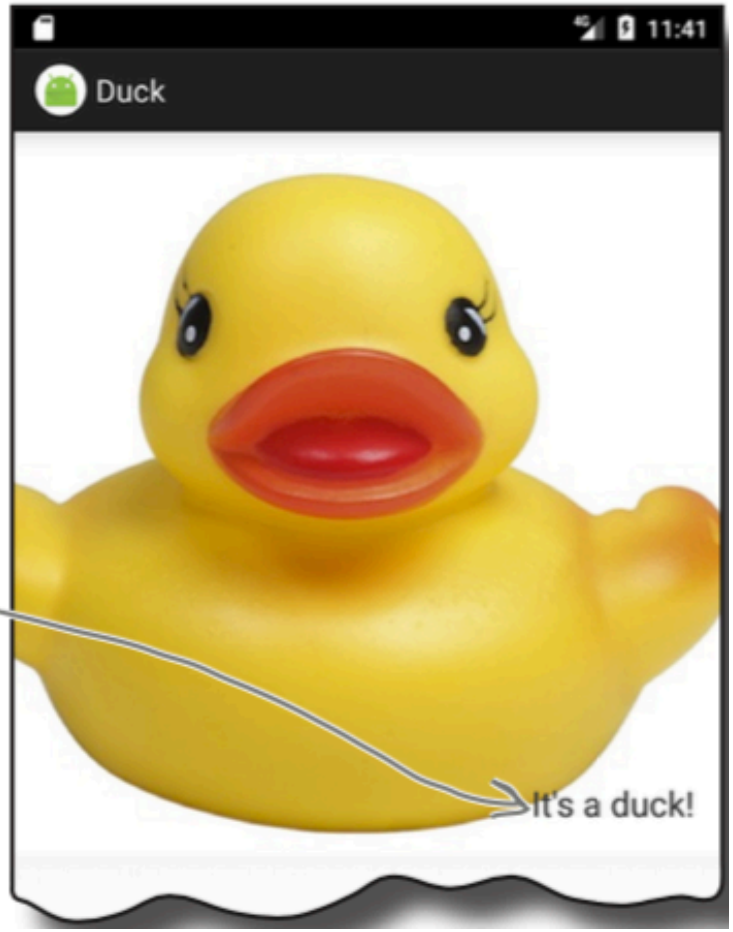
We've increased the size of the text

In a real-world duck app, you'd want to add this text as a String resource.

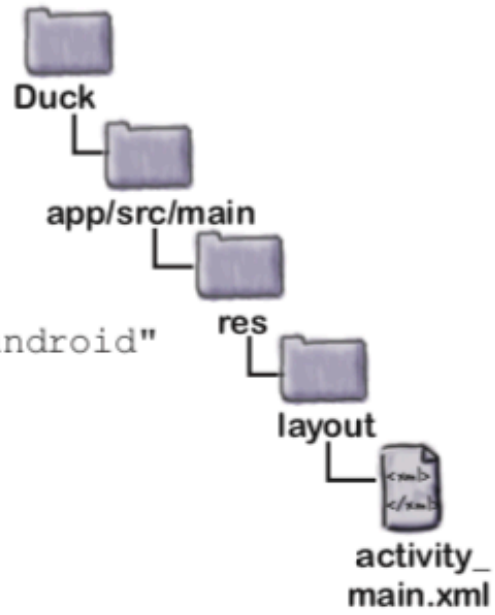


Un frame layout apila las vistas en el orden en que aparecen. Para modificar el orden se puede utilizar layout_gravity

```
...
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:layout_gravity="bottom|end"
        android:textSize="20sp"
        android:text="It's a duck!" />
</FrameLayout>
```



Anidando un layout dentro de un frame layout




```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context="com.hfad.duck.MainActivity">

    <ImageView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:scaleType="centerCrop"
    android:src="@drawable/duck"/>
```

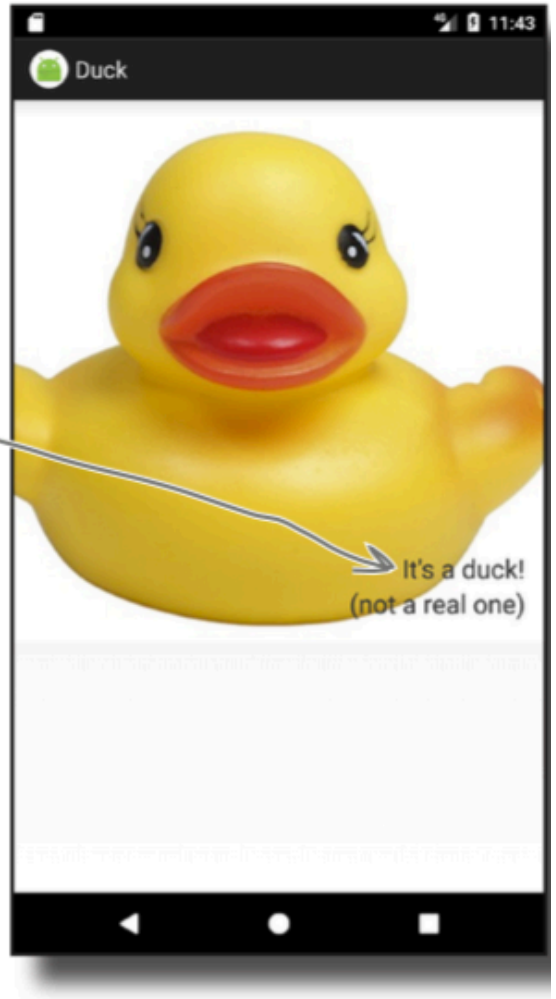
Anidando un layout dentro de un frame layout

We're adding a linear layout that's just big enough to contain its text views.

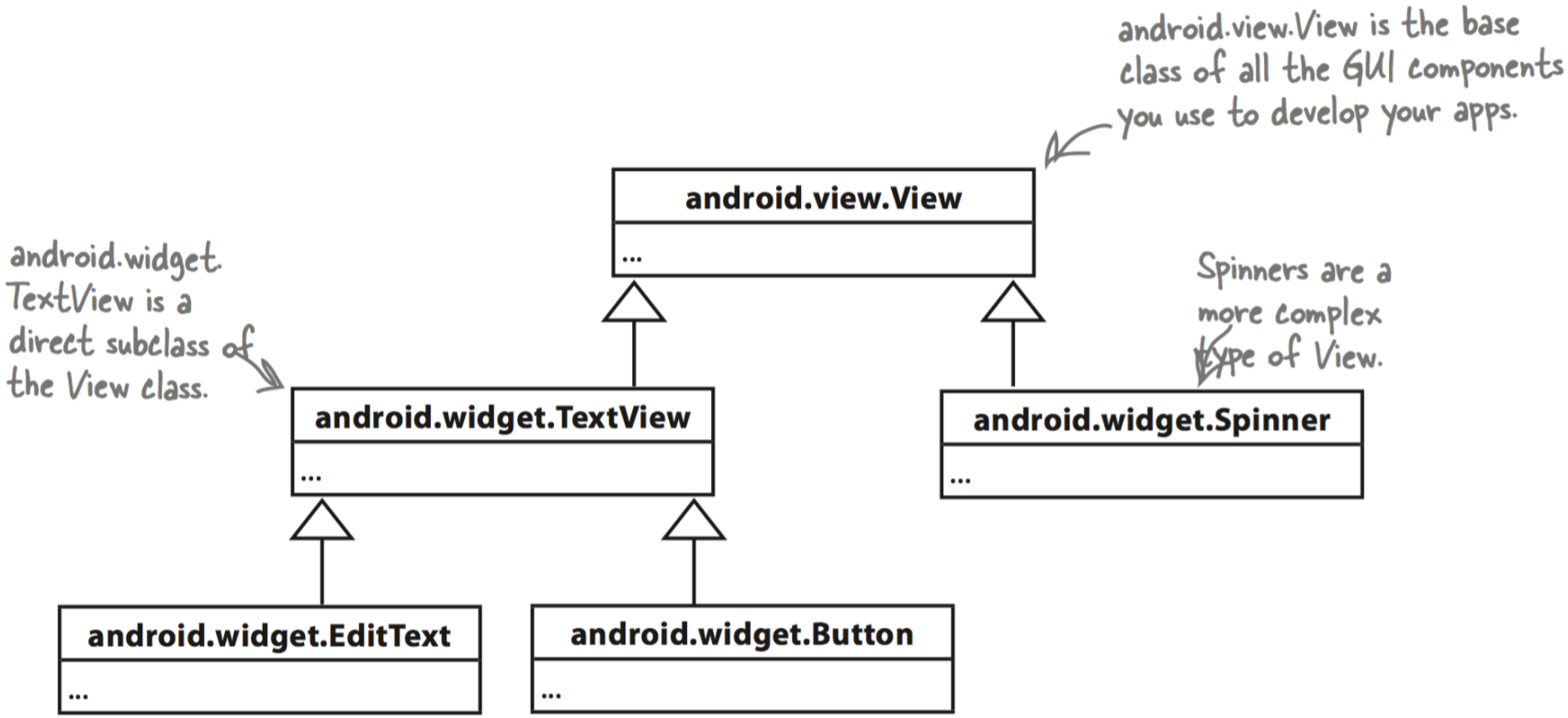
Move each text view in the linear layout to the end of its available space.

This line sinks the linear layout to the bottom-end corner of the frame layout.

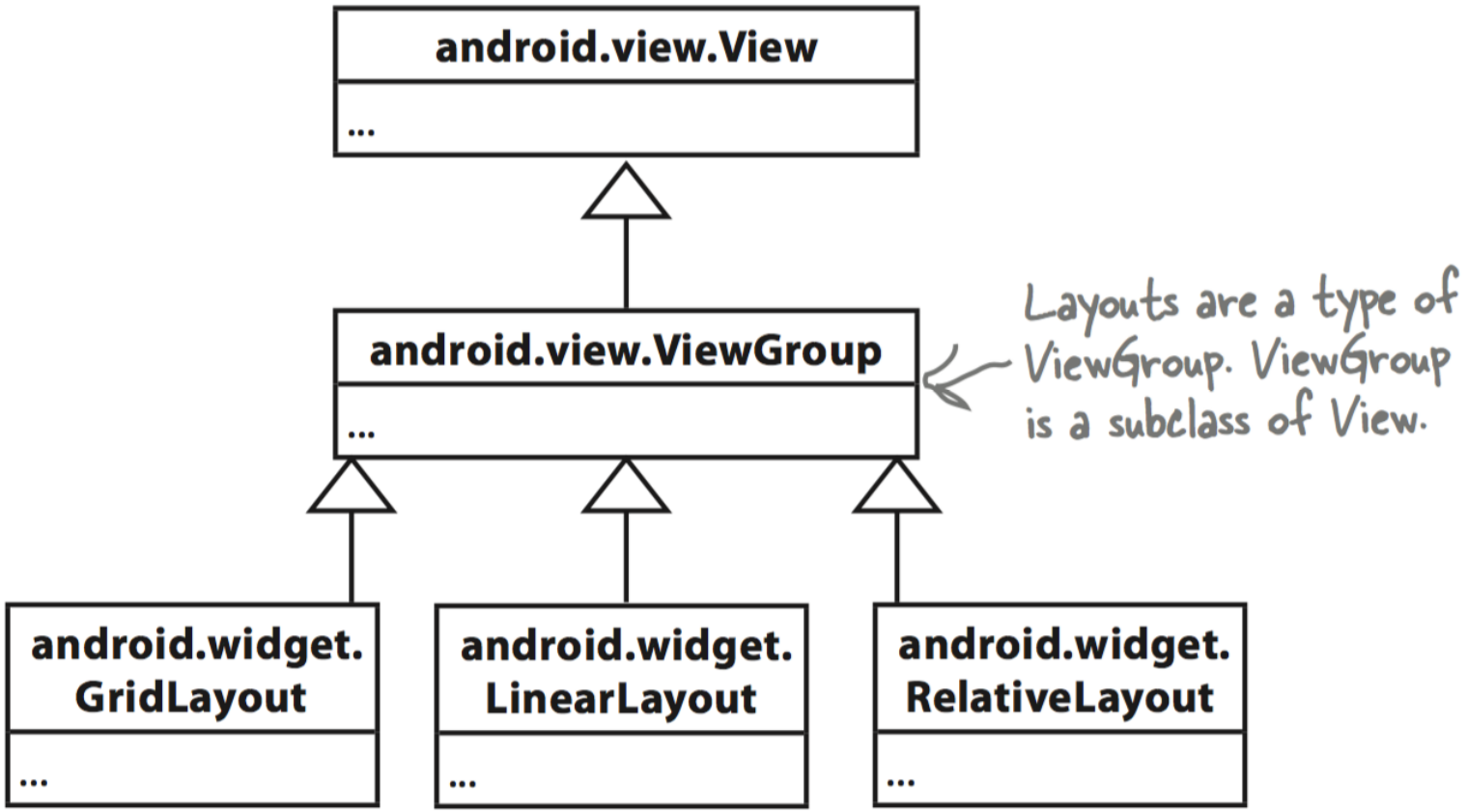
```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_gravity="bottom|end"
    android:gravity="end"
    android:padding="16dp" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:text="It's a duck!"  />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:text="(not a real one)" />
</LinearLayout>
</FrameLayout>
```



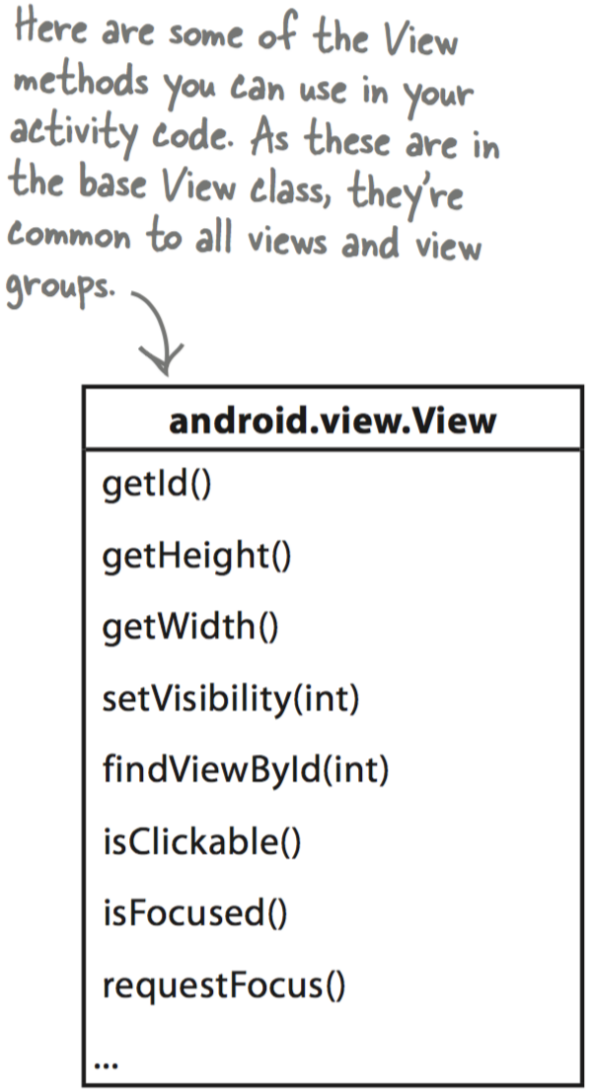
Los componentes de la interfaz de usuario son vistas (views)



Los layouts son un tipo de vista



Métodos de la clase View



Un layout es una jerarquía de vistas

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ... >

    <Button
        android:id="@+id/send"
        ... />

    <EditText
        android:id="@+id/message"
        ... />

</RelativeLayout>
```

We've left out a lot of the XML. The key thing is the views the view group contains.

The relative layout

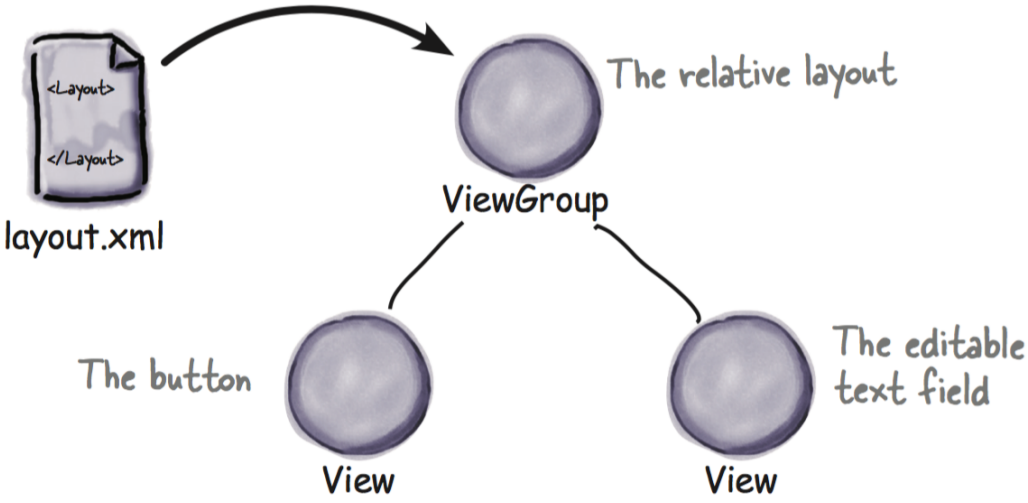
ViewGroup

The button

View

View

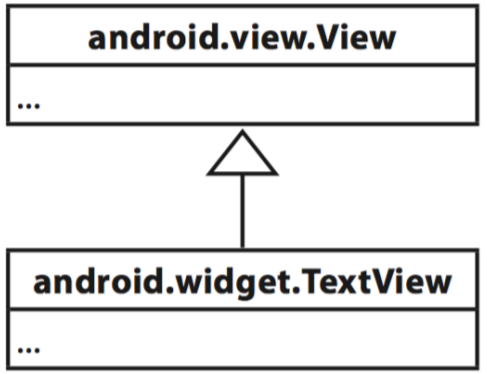
The editable text field



El componente TextView

Text view

Used for displaying text.



Defining it in XML

You define a text view in your layout using the <TextView> element. You use android:text to say what text you want it to display, usually by using a string resource:

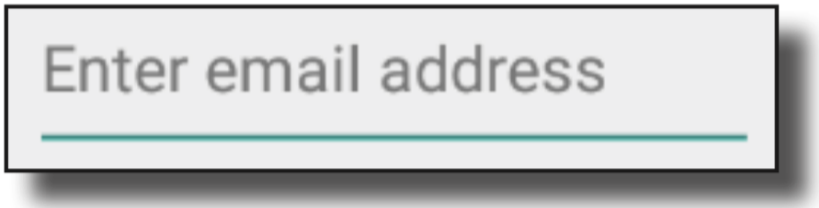
```
<TextView
    android:id="@+id/text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/text" />
```

```
TextView textView = (TextView) findViewById(R.id.text_view);
textView.setText("Some other string");
```

El componente EditText

Edit Text

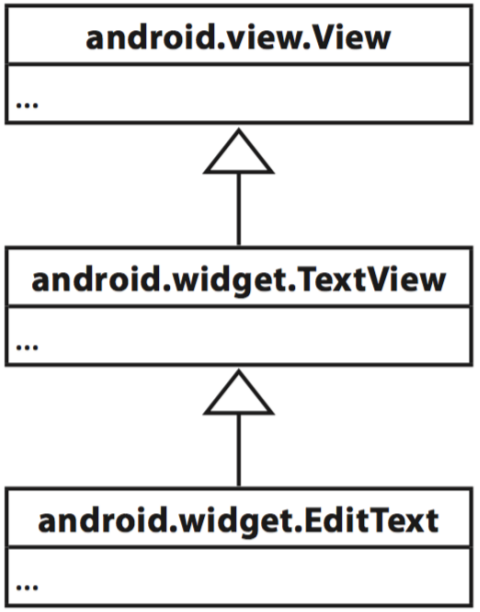
Like a text view, but editable.



Defining it in XML

You define an editable text view in XML using the `<EditText>` element. You use the `android:hint` attribute to give a hint to the user as to how to fill it in.

```
<EditText
    android:id="@+id/edit_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_text" />
```




```
EditText editText = (EditText) findViewById(R.id.edit_text);
String text = editText.getText().toString();
```

Valores para el atributo inputType

You can find the entire list in the online Android developer documentation.

Value	What it does
phone	Provides a phone number keypad.
textPassword	Displays a text entry keypad, and your input is concealed.
textCapSentences	Capitalizes the first word of a sentence.
textAutoCorrect	Automatically corrects the text being input.

```
android:inputType="number"
```

```
android:inputType="textCapSentences|textAutoCorrect"
```

El componente Button

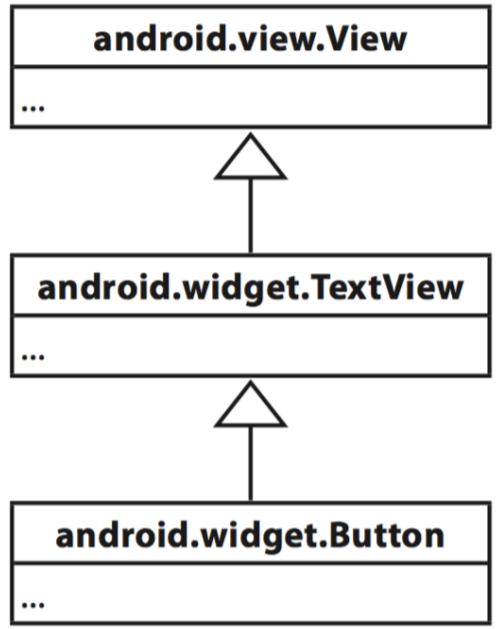
Button

Usually used to make your app do something when the button's clicked.

Defining it in XML

You define a button in XML using the <Button> element. You use the android:text attribute to say what text you want the button to display:

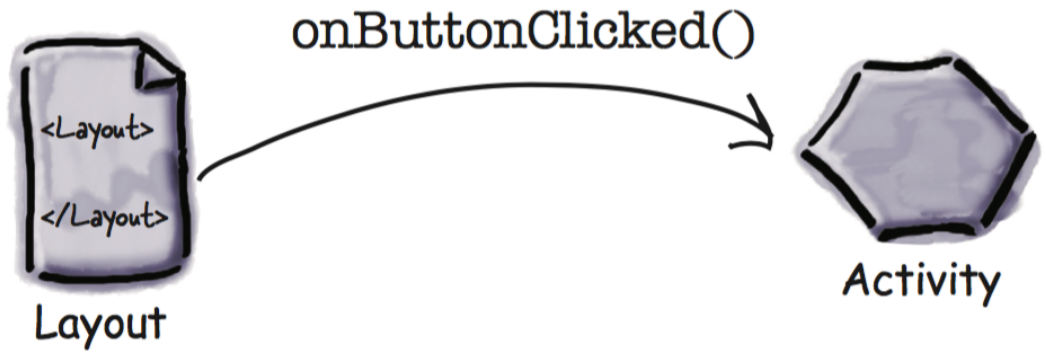
```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text" />
```



El componente Button

```
android:onClick="onButtonClicked"
```

```
/** Called when the button is clicked */
public void onButtonClicked(View view) {
    // Do something in response to button click
}
```



El componente ToggleButton

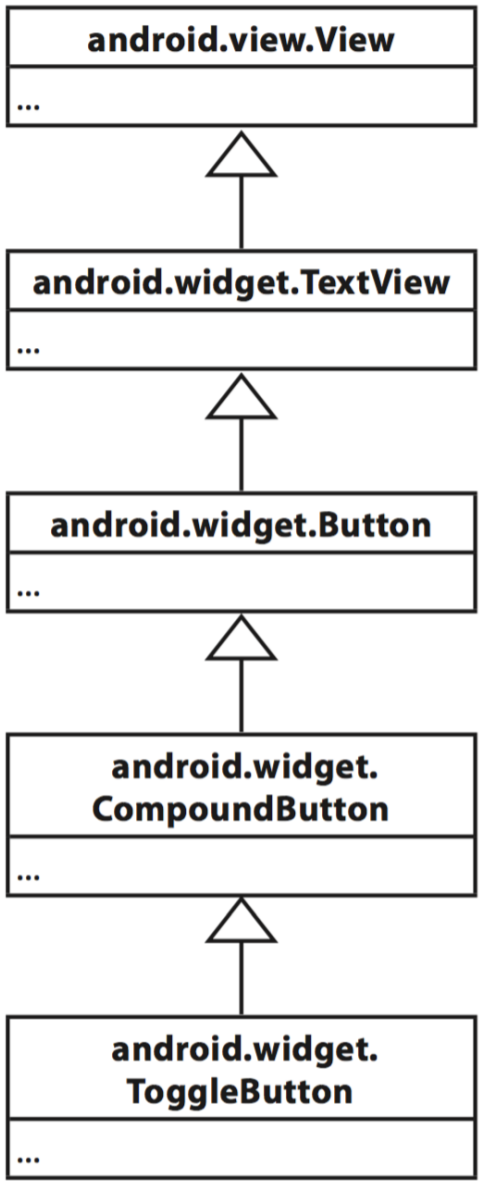
Toggle button

A toggle button allows you to choose between two states by clicking a button.

Defining it in XML

You define a toggle button in XML using the `<ToggleButton>` element. You use the `android:textOn` and `android:textOff` attributes to say what text you want the button to display depending on the state of the button:

```
<ToggleButton
    android:id="@+id/toggle_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="@string/on"
    android:textOff="@string/off" />
```



El componente ToggleButton

```
android:onClick="onToggleButtonClicked"
```

This is exactly the same as calling a method when a normal button gets clicked.

```
/** Called when the toggle button is clicked */
public void onToggleClicked(View view) {
    // Get the state of the toggle button.
    boolean on = ((ToggleButton) view).isChecked();
    if (on) {
        // On
    } else {
        // Off
    }
}
```

This returns true if the toggle button is on, and false if the toggle button is off.

El componente Switch

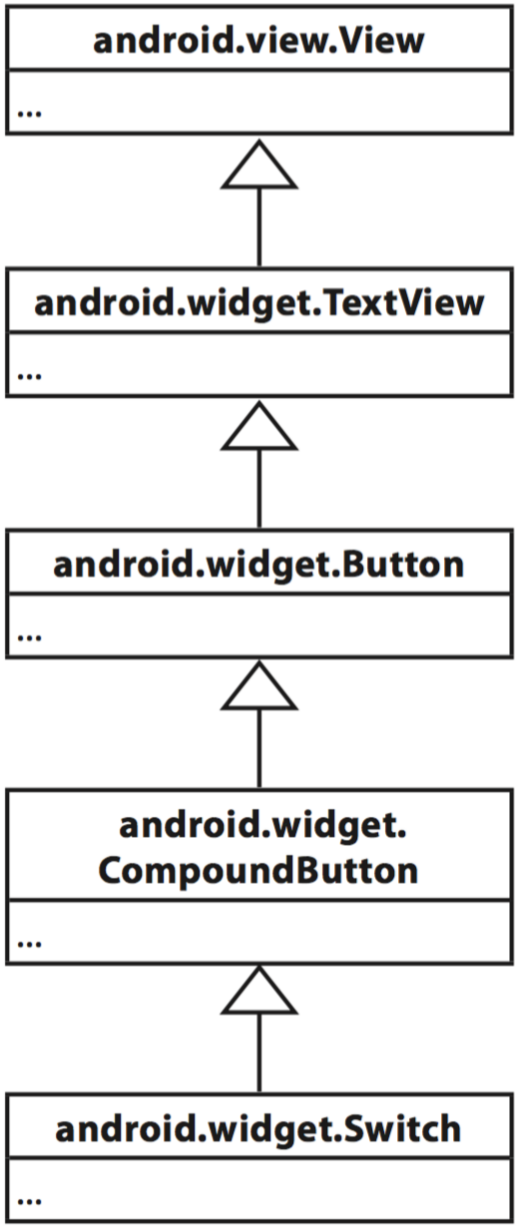
Switch

A switch is a slider control that acts in the same way as a toggle button.

Defining it in XML

You define a toggle button in XML using the <Switch> element. You use the android:textOn and android:textOff attributes to say what text you want the switch to display depending on the state of the switch:

```
<Switch
    android:id="@+id/switch_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="@string/on"
    android:textOff="@string/off" />
```



El componente Switch

```
android:onClick="onSwitchClicked"
```

```
/** Called when the switch is clicked */
public void onSwitchClicked(View view) {
    // Is the switch on?
    boolean on = ((Switch) view).isChecked();

    if (on) {
        // On
    } else {
        // Off
    }
}
```

This is very similar code to that used with the toggle button.

El componente CheckBox

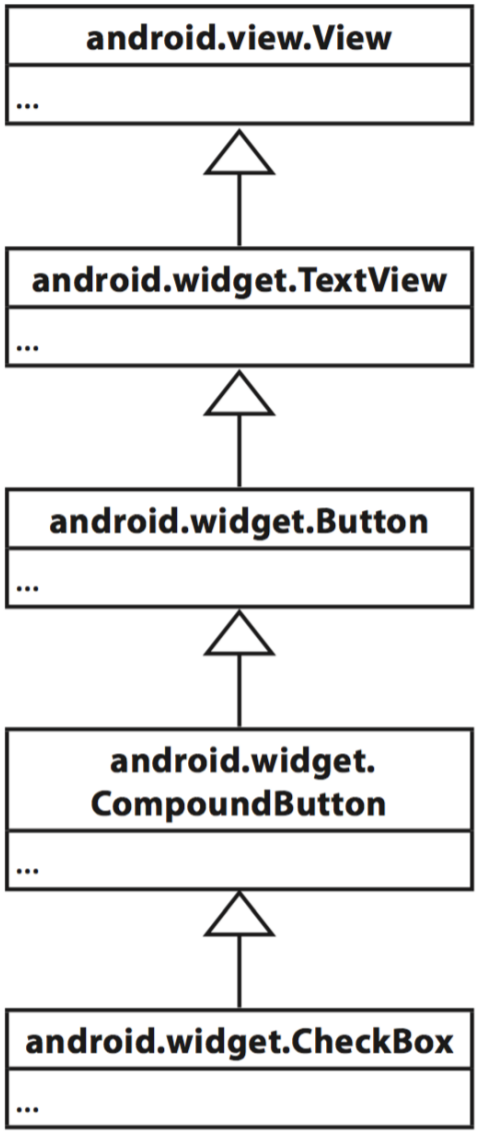
Check boxes

Check boxes let you display multiple options to users. They can then select whichever options they want. Each of the checkboxes can be checked or unchecked independently of any others.

Defining them in XML

You define each checkbox in XML using the `<CheckBox>` element. You use the `android:text` attribute to display text for each option:

```
<CheckBox android:id="@+id/checkbox_milk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/milk" />
```



El componente CheckBox

```
<CheckBox android:id="@+id/checkbox_milk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/milk"
    android:onClick="onCheckboxClicked"/>

<CheckBox android:id="@+id/checkbox_sugar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/sugar"
    android:onClick="onCheckboxClicked"/>
```

In this case, the onCheckboxClicked() method will get called no matter which checkbox gets clicked. We could have specified a different method for each checkbox if we'd wanted to.

El componente CheckBox

```
public void onCheckboxClicked(View view) {
    // Has the checkbox that was clicked been checked?
    boolean checked = ((CheckBox) view).isChecked();

    // Retrieve which checkbox was clicked
    switch(view.getId()) {
        case R.id.checkbox_milk:
            if (checked)
                // Milky coffee
            else
                // Black as the midnight sky on a moonless night
            break;
        case R.id.checkbox_sugar:
            if (checked)
                // Sweet
            else
                // Keep it bitter
            break;
    }
}
```

El componente CheckBox

```
CheckBox checkbox = (CheckBox) findViewById(R.id.checkbox_milk);
boolean checked = checkbox.isChecked();
if (checked) {
    //do something
}
```

El componente RadioButton

Radio buttons

These let you display multiple options to the user. The user can select a single option.

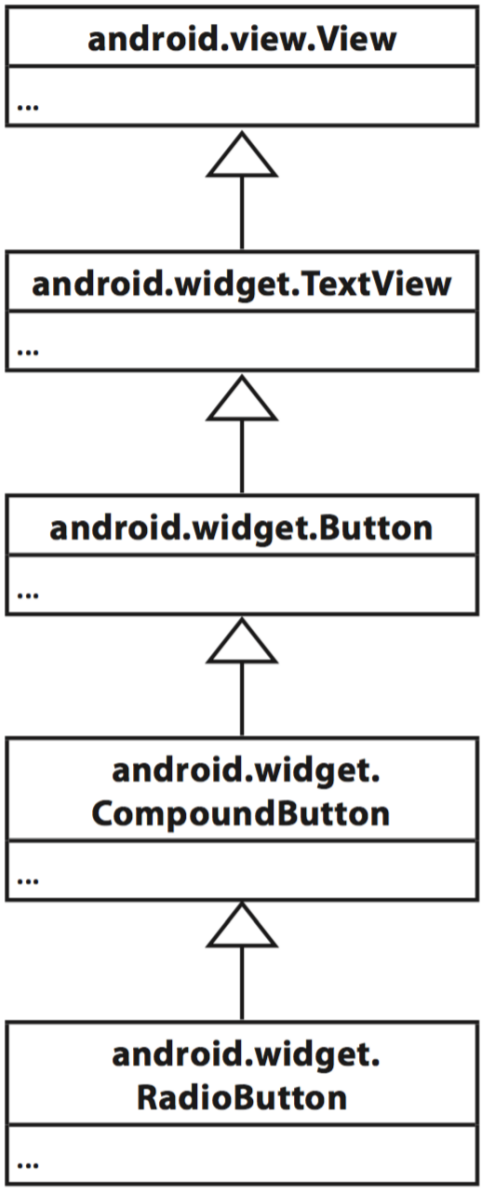
Use radio buttons to restrict the users choice to just one option.

Defining them in XML

You start by defining a radio group, a special type of view group, using the `<RadioGroup>` tag. Within this, you then define individual radio buttons using the `<RadioButton>` tag:

```
<RadioGroup android:id="@+id/radio_group"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <RadioButton android:id="@+id/radio_cavemen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cavemen" />
```

You can choose to display the radio buttons in a horizontal or vertical list.



El componente RadioButton

```
RadioGroup radioGroup = (RadioGroup) findViewById(R.id.radioGroup);
int id = radioGroup.getCheckedRadioButtonId();
if (id == -1){
    //no item selected
}
else{
    RadioButton radioButton = findViewById(id);
}
```

El componente RadioButton

```
<RadioGroup android:id="@+id/radio_group"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <RadioButton android:id="@+id/radio_cavemen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cavemen"
        android:onClick="onRadioButtonClicked" />

    <RadioButton android:id="@+id/radio_astronauts"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/astronauts"
        android:onClick="onRadioButtonClicked" />
</RadioGroup>
```

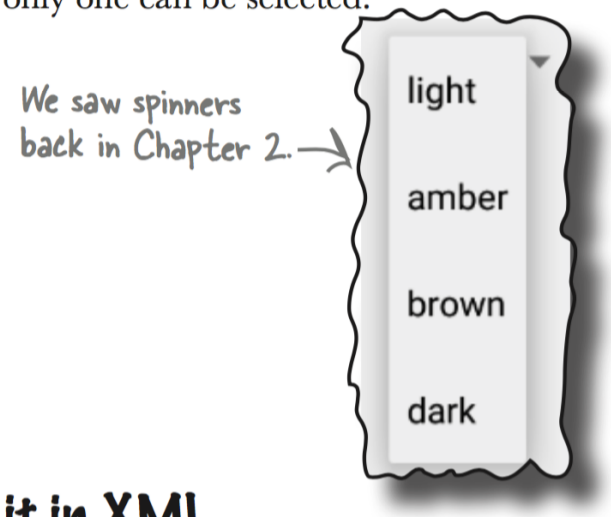
El componente RadioButton

```
public void onRadioButtonClicked(View view) {
    RadioGroup radioGroup = (RadioGroup) findViewById(R.id.radioGroup);
    int id = radioGroup.getCheckedRadioButtonId();
    switch(id) {
        case R.id.radio_cavemen:
            // Cavemen win
            break;
        case R.id.radio_astronauts:
            // Astronauts win
            break;
    }
}
```

El componente Spinner

Spinner

As you've already seen, a spinner gives you a drop-down list of values from which only one can be selected.

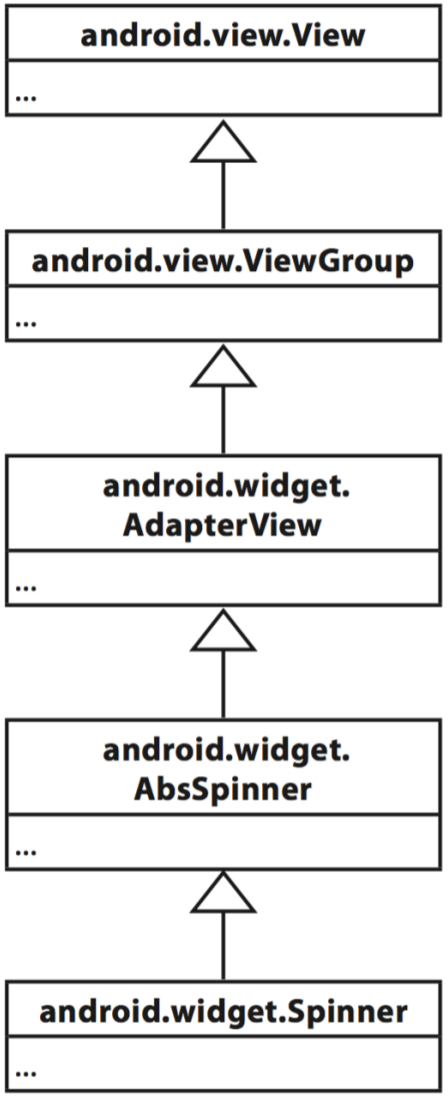


Defining it in XML

You define a spinner in XML using the `<Spinner>` element. You add a static array of entries to the spinner by using the `android:entries` attribute and setting it to an array of strings.

There are other ways of populating the spinner, which you'll see later in the book.

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:entries="@array/spinner_values" />
```



El componente Spinner

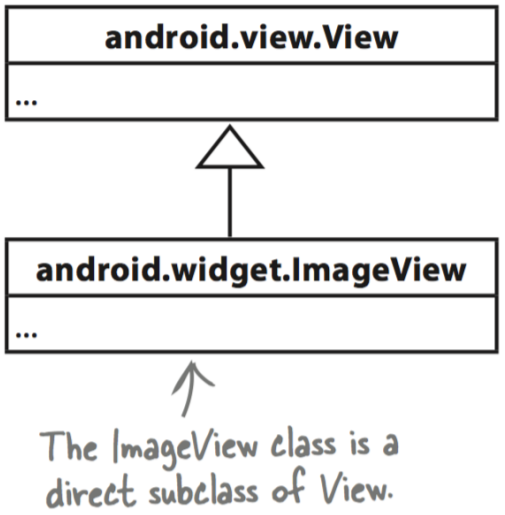
```
<string-array name="spinner_values">
    <item>light</item>
    <item>amber</item>
    <item>brown</item>
    <item>dark</item>
</string-array>
```

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
String string = String.valueOf(spinner.getSelectedItem());
```

El componente ImageView

Image views

You use an image view to display an image:



El componente ImageView

`android-ldpi`	Low-density screens, around 120 dpi.
`android-mdpi`	Medium-density screens, around 160 dpi.
`android-hdpi`	High-density screens, around 240 dpi.
`android-xhdpi`	Extra-high-density screens, around 320 dpi.
`android-xxhdpi`	Extra-extra-high-density screens, around 480 dpi.
`android-xxxhdpi`	Extra-extra-extra high-density screens, around 640 dpi.

Depending on what version of Android Studio you're running, the IDE may create some of these folders for you automatically.

El componente ImageView

```
<ImageView
    android:layout_width="200dp"
    android:layout_height="100dp"
    android:src="@drawable/starbuzz_logo"
    android:contentDescription="@string/starbuzz_logo" />
```

```
ImageView photo = (ImageView)findViewById(R.id.photo);
int image = R.drawable.starbuzz_logo;
String description = "This is the logo";
photo.setImageResource(image);
photo.setContentDescription(description);
```

Imágenes en botones

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableRight="@drawable/android"
    android:text="@string/click_me" />
```

Display the android image resource on the right side of the button.

You can also use drawableStart and drawableEnd to support right-to-left languages.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableLeft="@drawable/android"
    android:text="@string/click_me" />
```

Imágenes en botones

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableBottom="@drawable/android"
    android:text="@string/click_me" />
```


```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableTop="@drawable/android"
    android:text="@string/click_me" />
```

El componente ImageButton

Image button

An image button is just like a button, except it contains an image and no text.

Defining it in XML

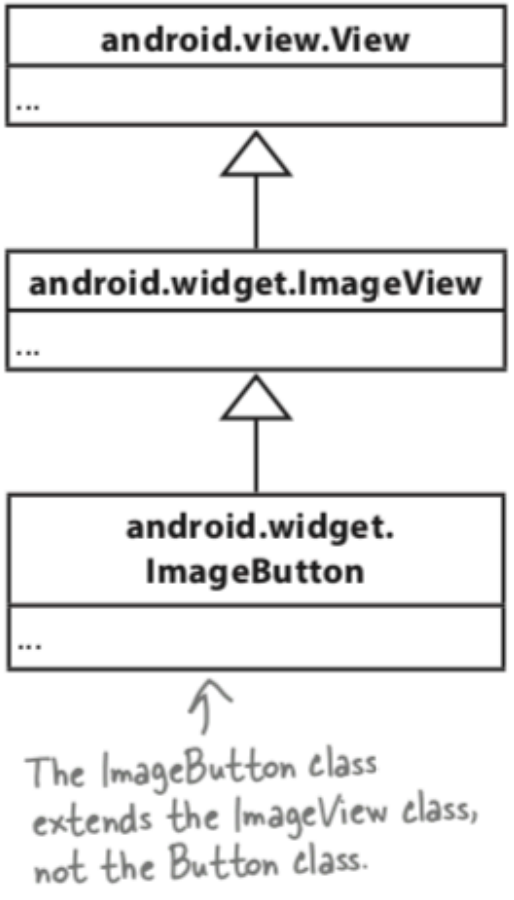
You define an image button in XML using the `<ImageButton>` element. You use the `android:src` attribute to say what image you want the image button to display:

```
<ImageButton
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon" />
```

Using it in your activity code

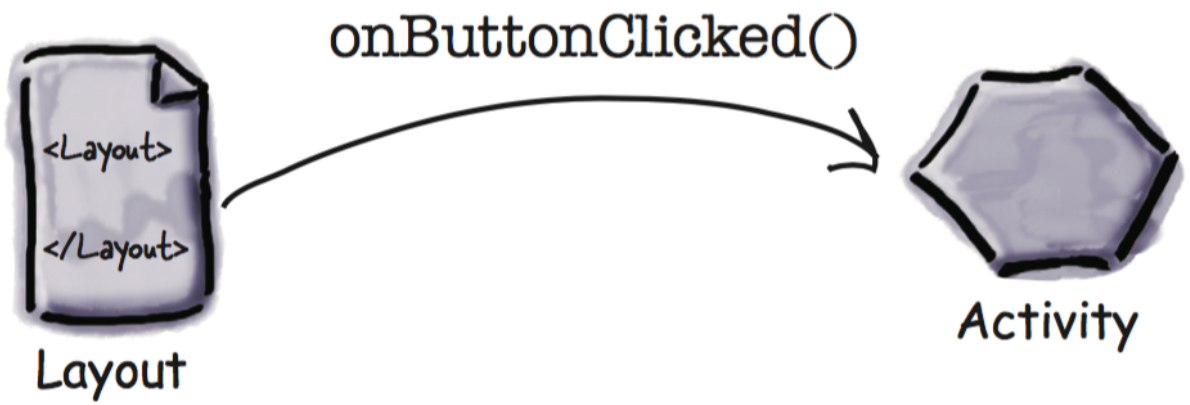
You get the image button to respond to the user clicking it by using the `android:onClick` attribute in the layout XML, and setting it to the name of the method you want to call in your activity code:

```
android:onClick="onButtonClicked"
```



El componente ImageButton

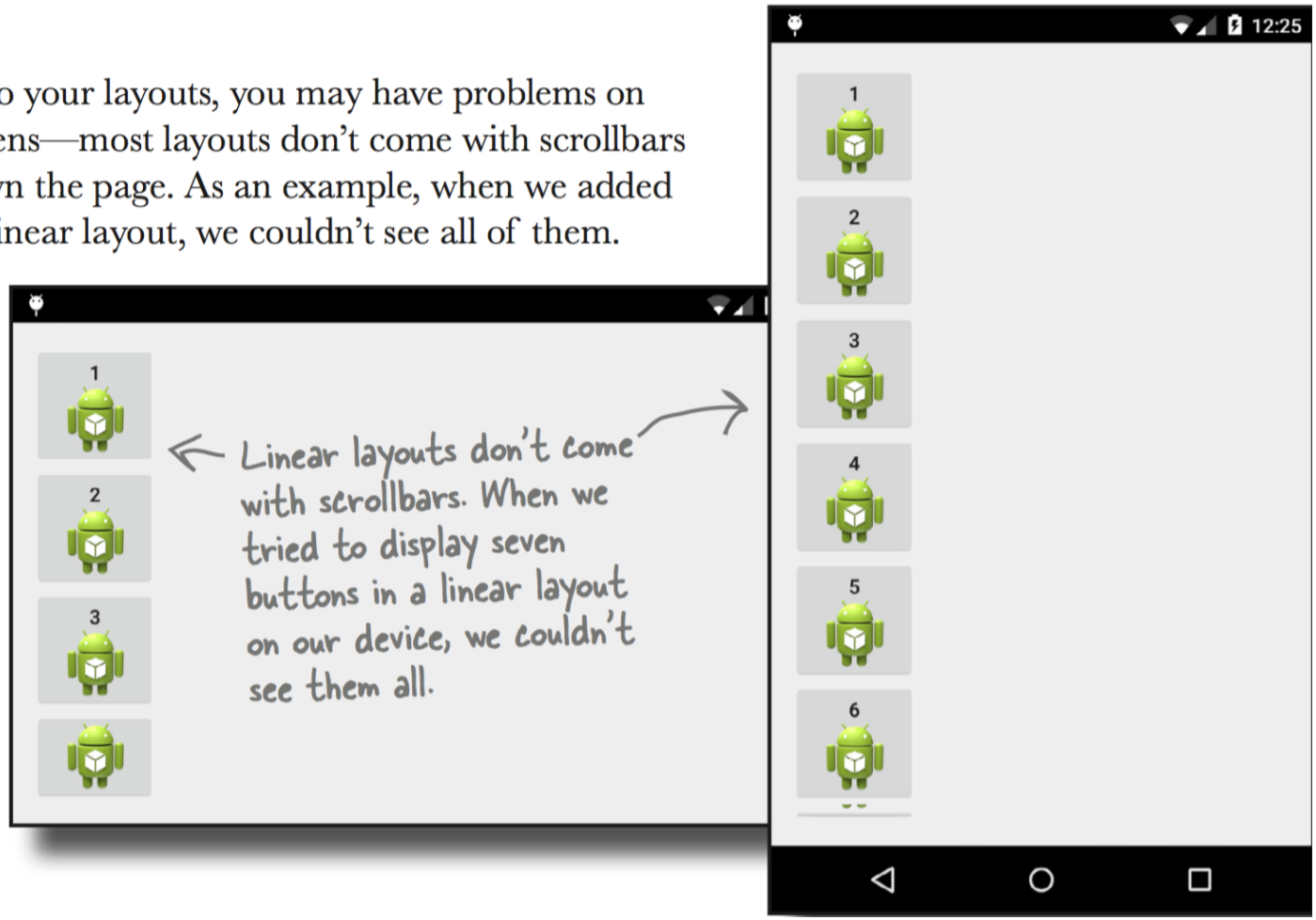
```
/** Called when the image button is clicked */
public void onButtonClicked(View view) {
    // Do something in response to button click
}
```



El componente ScrollView

Scroll views

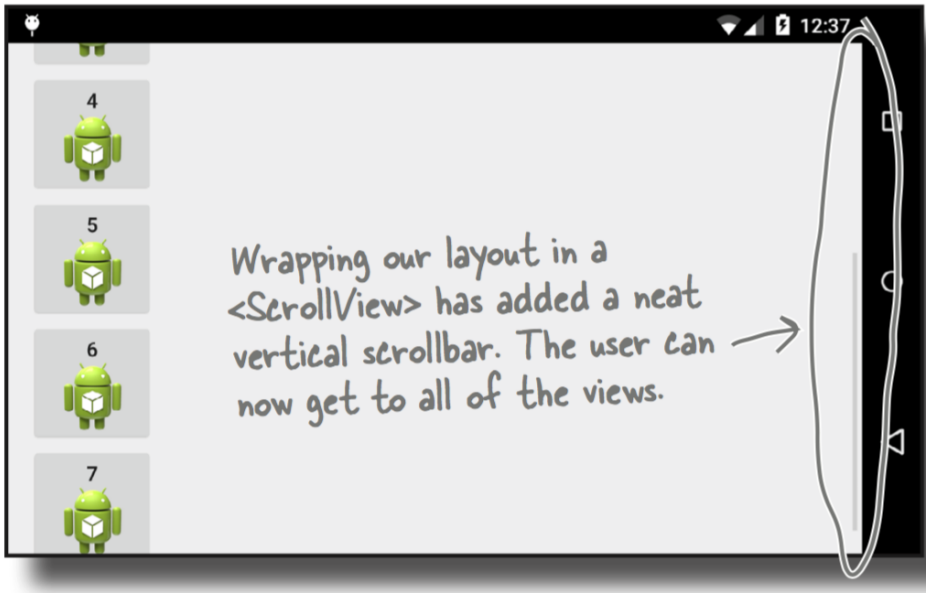
If you add lots of views to your layouts, you may have problems on devices with smaller screens—most layouts don't come with scrollbars to allow you to scroll down the page. As an example, when we added seven large buttons to a linear layout, we couldn't see all of them.



El componente ScrollView

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingBottom="16dp"
        android:paddingLeft="16dp"
        android:paddingRight="16dp"
        android:paddingTop="16dp"
        android:orientation="vertical" >
    ...
    </LinearLayout>
</ScrollView>
```

Move these attributes from the original layout to the <ScrollView> as the <ScrollView> is now the root element.

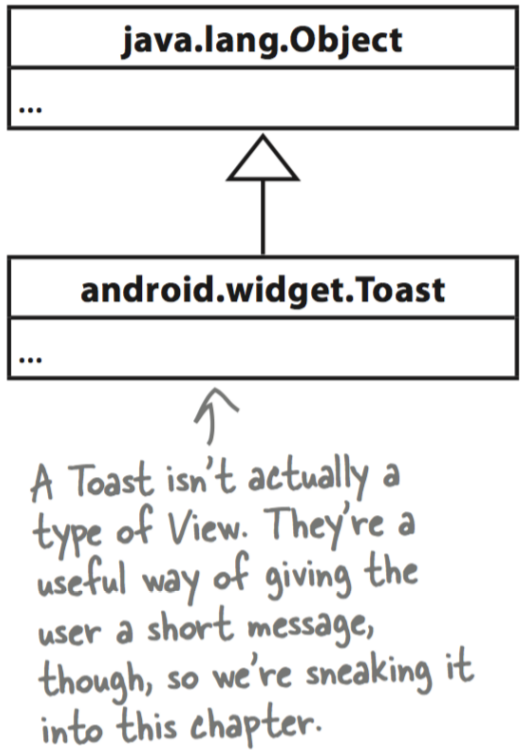


El componente Toast

Toasts

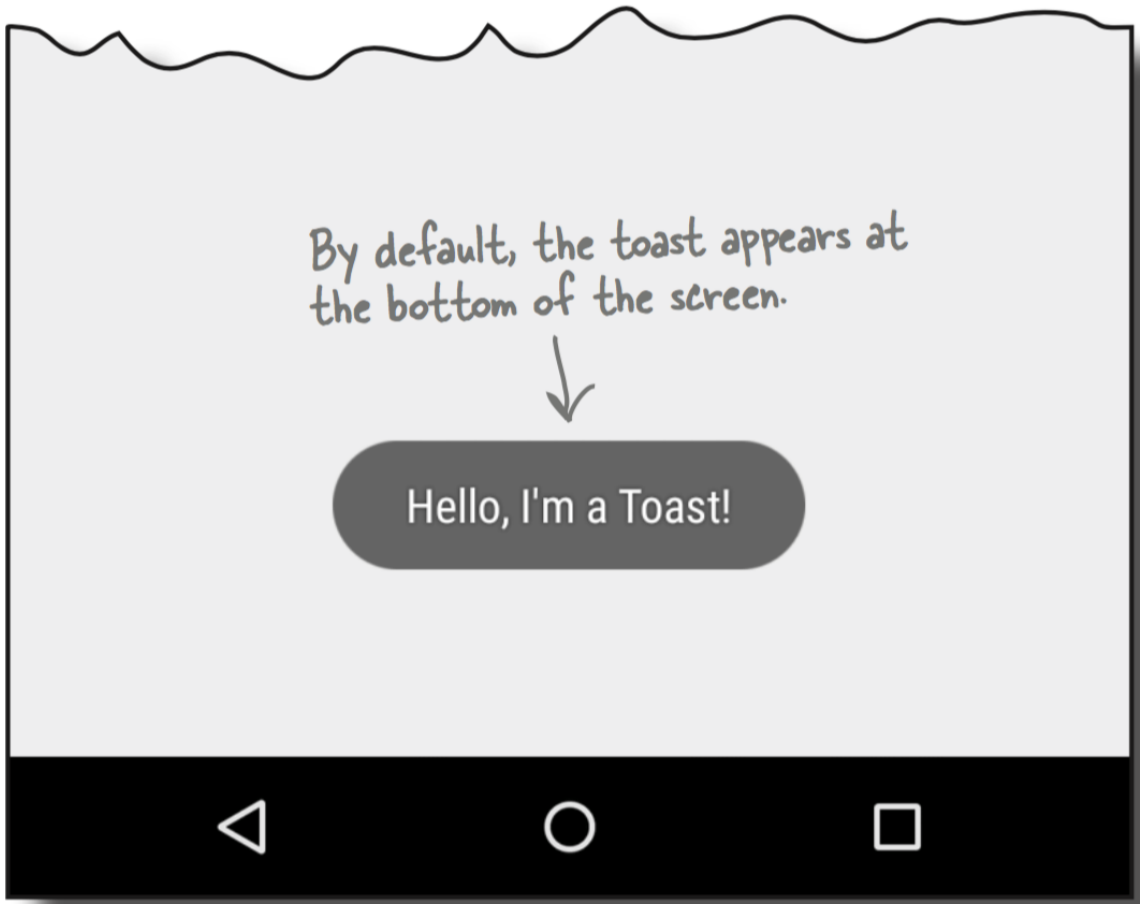
There's one final widget we want to show you in this chapter: a toast. A toast is a simple pop-up message you can display on the screen.

Toasts are purely informative, as the user can't interact with them. While a toast is displayed, the activity stays visible and interactive. The toast automatically disappears when it times out.



El componente Toast

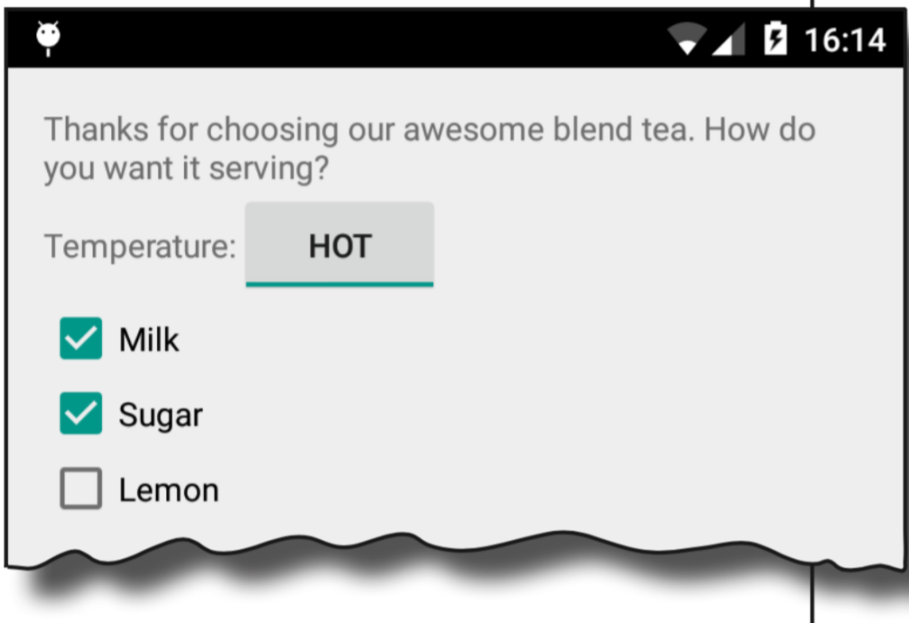
```
CharSequence text = "Hello, I'm a Toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(this, text, duration);
toast.show();
```



Ejercicio 1

It's time for you to try out some of the views we've introduced you to this chapter. Create a layout that will create this screen:

You probably won't want to write the code here, but why not experiment in the IDE?



Solución

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:columnCount="2"
    tools:context=".MainActivity" >
```

We used a grid layout, but you could have used a relative layout instead.

We'll use two columns.

Solución

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_row="0"
    android:layout_column="0"
    android:layout_columnSpan="2"
    android:text="@string/message" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_row="1"
    android:layout_column="0"
    android:text="@string/temp" />
```

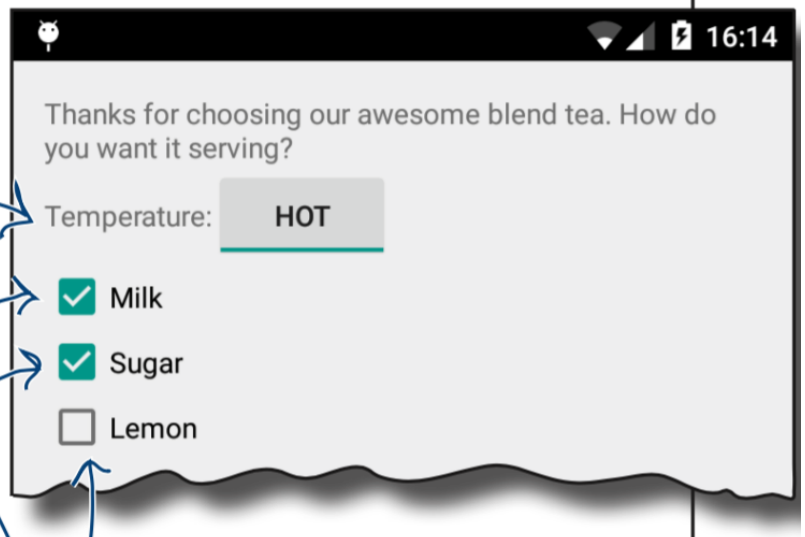
Display a text view at the top that spans both columns.

All of the views need strings to be added to strings.xml.

Add a temperature label to the next row in the first column.

Solución

```
<ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_row="1"
    android:layout_column="1"
    android:textOn="@string/hot"
    android:textOff="@string/cold" />

<CheckBox android:id="@+id/checkbox_milk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_row="2"
    android:layout_column="0"
    android:text="@string/milk" />

<CheckBox android:id="@+id/checkbox_sugar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_row="3"
    android:layout_column="0"
    android:text="@string/sugar" />

<CheckBox android:id="@+id/checkbox_lemon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_row="4"
    android:layout_column="0"
    android:text="@string/lemon" />
```



We used a checkbox for each of the values (Milk, Sugar, and Lemon). We put each one on a separate row.

Ejercicio 2

It's time for you to try out some of the views we've introduced you to this chapter. Create a layout that will create this screen:

You probably won't want to write the code here, but why not experiment in the IDE?

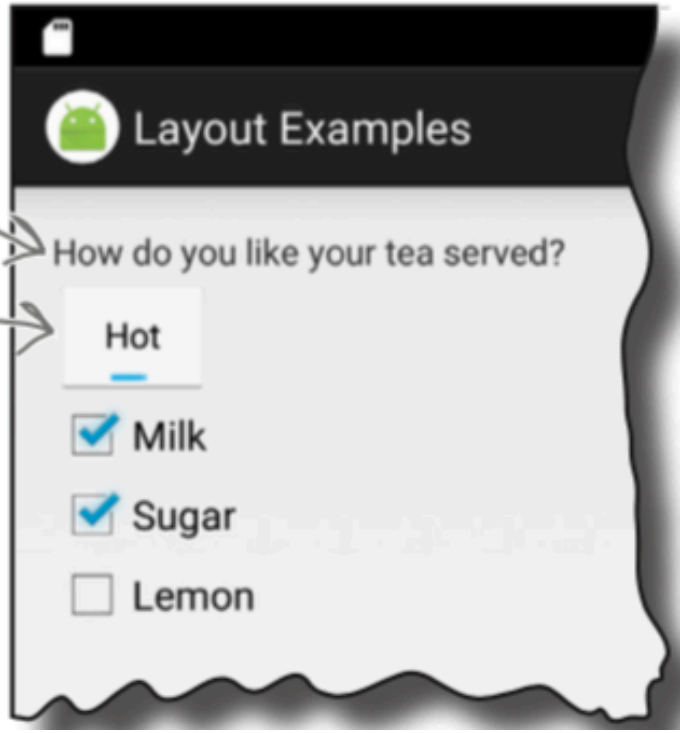
Solución

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.layoutexamples.MainActivity" >
```

Solución

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="How do you like your tea served?" />

<ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Hot"
    android:textOff="Cold" />
```

We used a toggle button to display whether the drink should be served hot or cold.



Solución

```
<CheckBox android:id="@+id/checkbox_milk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Milk" />

<CheckBox android:id="@+id/checkbox_sugar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Sugar" />

<CheckBox android:id="@+id/checkbox_lemon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Lemon" />

</LinearLayout>
```

We used a checkbox for each of the values (Milk, Sugar, and Lemon). We put each one on a separate row.

BULLET POINTS

- GUI components are all types of view. They are all subclasses of the `android.view.View` class.
- All layouts are subclasses of the `android.view.ViewGroup` class. A view group is a type of view that can contain multiple views.
- The layout XML file gets converted to a ViewGroup containing a hierarchical tree of views.
- A relative layout displays child views relative to other views, or relative to the parent layout.
- A linear layout lists views either horizontally or vertically. You specify the direction using the `android:orientation` attribute.
- Use `android:layout_weight` in a linear layout if you want a view to use up extra space in the layout.
- `android:layout_gravity` lets you say where you want views to appear in their available space.
- `android:gravity` lets you say where you want the contents to appear inside the view.
- `<ToggleButton>` defines a toggle button which allows you to choose between two states by clicking a button.
- `<Switch>` defines a switch control that behaves in the same way as a toggle button. It requires API level 14 or above.
- `<CheckBox>` defines a checkbox.

- A frame layout stacks views.

- A grid layout divides the screen into a grid of cells so that you can specify which cell (or cells) each view should occupy. Use `android:columnCount` to say how many columns there should be. Use `android:layout_row` and `android:layout_column` to say which cell you want each view to appear in. Use `android:layout_columnSpan` to say how many columns the view should spread across.

- Use `android:padding*` attributes to specify how much padding you want there to be around a view.

- To define a group of radio buttons, first use `<RadioGroup>` to define the radio group. Then put individual radio buttons in the radio group using `<RadioButton>`.

- Use `<ImageView>` to display an image.

- `<ImageButton>` defines a button with no text, just an image.

- Add scrollbars using `<ScrollView>` or `<HorizontalScrollView>`.

- A `Toast` is a pop-up message.