Phil Dutson

Android™ Development Patterns

Best Practices for Professional Developers

Android™ Development Patterns

of `0dp` for an equal vertical orientation or you can set each view to `android:layout_width` with a value of `0dp` for an equal horizontal orientation.

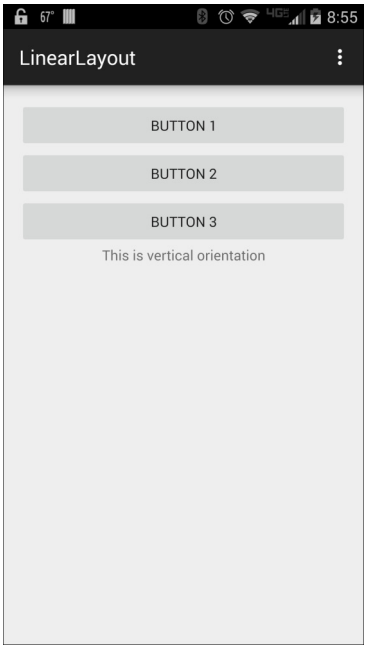

Figure 6.3 The buttons and text are positioned starting at the top of the layout and continuing down vertically.

To change the orientation to horizontal, you should change the value of the `android:orientation` property to horizontal, and then each child element will need to have the `android:layout_height` and `android:layout_width` properties adjusted.

Figure 6.4 demonstrates the same layout adjusted to be displayed horizontally.

Relative Layout

The relative layout is used when you have a complex user interface that requires specific sizing and relies on knowing where a view or layout element will be. It thus is named because elements are placed based on the relative proximity or position of other elements in the layout as well as to the containing layout.

The relative layout provides a somewhat flexible and adaptable interface. Elements can be told to position based off of the center, left, right, top, or bottom of the parent view. This can also be combined with placement values off of other child views that are already positioned.

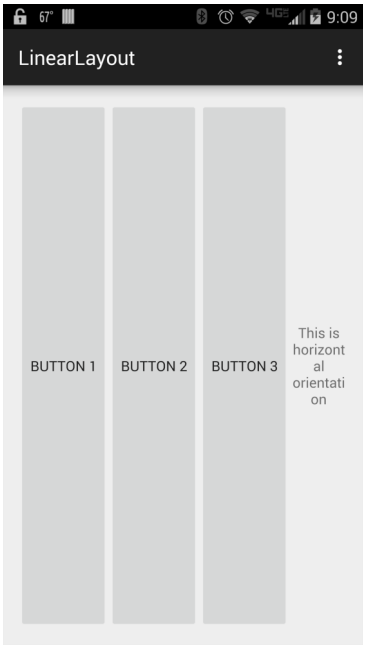

Figure 6.4 Text becomes almost impossible to read when there are too many elements being forced into a cramped area.

The following shows a layout XML file that uses a relative layout to position text and four buttons:

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

  <TextView
    android:text="Relative layouts allow flexible positioning"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />
```

```
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button 1"
    android:id="@+id/button"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button 2"
    android:id="@+id/button2"
    android:layout_below="@id/button"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button 3"
    android:id="@+id/button3"
    android:layout_alignTop="@id/button2"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button 4"
    android:id="@+id/button4"
    android:layout_below="@id/button2"
    android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Rather than use `gravity` to adjust how text is displayed, the `TextView` is placed at the top of the page in the center by using `android:layout_alignParentTop="true"` and `android:layout_centerHorizontal="true"`. Buttons 2, 3, and 4 are positioned based off of Button 1, which is placed directly in the center of the layout. Some extra properties are also used to align Buttons 2 and 3 to the left and right sides of the layout. Figure 6.5 demonstrates what the layout appears like when viewed on an Android device.

Another reason you should consider using a relative layout is that rather than create complex layouts by nesting multiple linear layouts, you can create the same type of interface without complicating the layout. By avoiding nesting layouts, you are able to keep the layout "flat." This decreases the amount of processing needed to display your layout and speeds up the screen rendering of your layout.

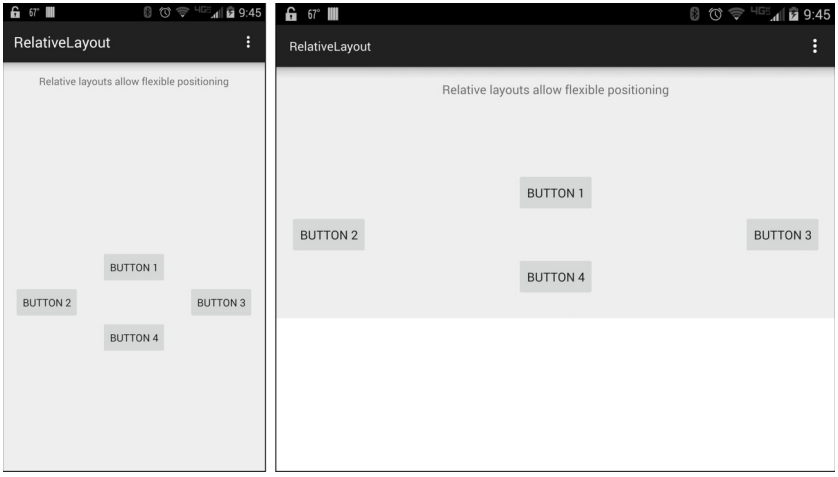

Figure 6.5 Regardless of device orientation, Button 1 is always at the center of the screen.

Table Layout

Table layouts are similar to how HTML table elements work. A table layout aligns child elements into rows and columns. Unlike tables in HTML, cell borders are never displayed and cells may be empty.

The table layout introduces some interesting formatting logic. By using `android:gravity`, you can adjust the text layout for elements. This may be needed as the size of columns is determined by the column that needs the most width or has the largest content.

You may think that you will just adjust the width of each element manually, but each child added to the table layout will default to a width of `match_parent`. The height is changeable, but has a default value of `wrap_content`. The following demonstrates a table layout with two rows containing `TextView`s and `Button`s:

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:stretchColumns="1"
  android:padding="10dp">

  <TableRow>
    <TextView
      android:text="Name:"
      android:padding="5dp"/>
    <TextView
      android:text="Jonathan Generic"
      android:gravity="end"
```

```
      android:padding="5dp"/>
  </TableRow>

  <TableRow>
    <Button
      android:text="Button 1"
      android:id="@+id/button"/>
    <Button
      android:text="Button2"
      android:id="@+id/button2" />
  </TableRow>

</TableLayout>
```

Although you cannot set explicit widths for child elements, you can create columns that are the same width by adding `android:layout_width="0dp"` and `android:layout_weight="1"` to elements in a row to force the table to render the columns with an equal width. The following shows the properties applied to the `<Button>` elements:

```
<TableRow>
  <Button
    android:layout_width="0dp"
    android:layout_weight="1"
    android:text="Button 1"
    android:id="@+id/button"/>
  <Button
    android:layout_width="0dp"
    android:layout_weight="1"
    android:text="Button2"
    android:id="@+id/button2" />
</TableRow>
```

This also requires that the `<TableLayout>` element contains a property of `android:stretchColumns="1"`. Figure 6.6 shows how this fix was used to change the button alignment.

Table layouts are best for displaying tabular data. This is data or visuals that require a grid or specific spacing to allow the user to read and comprehend data quickly without trying to decipher a design to get the data they are looking for.

Frame Layout

The frame layout is used either when you would like to reserve screen space for a single view or when you are creating overlays that have a z-index effect. The spatial effect is achieved due to how the frame layout handles child elements. It positions them into a stack, with the first item added on the bottom and the last item added on the top.

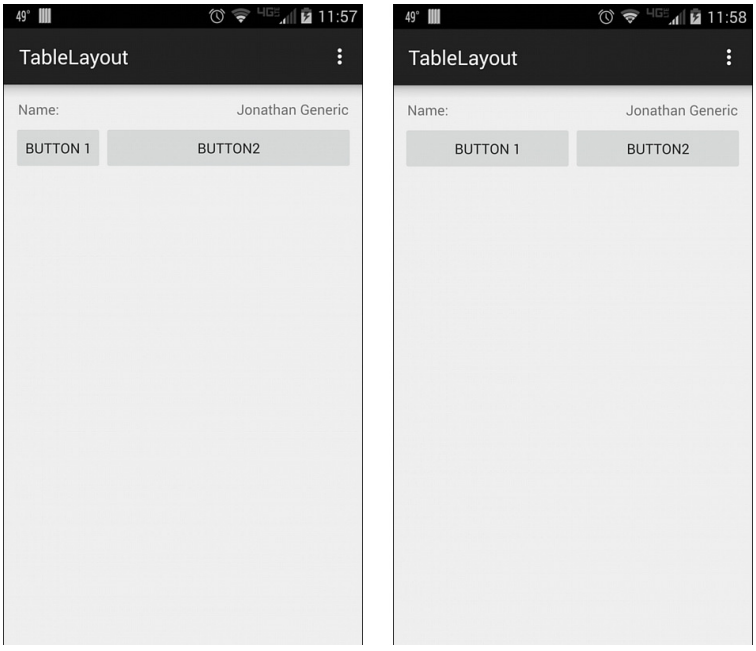

Figure 6.6 The buttons auto-align, leaving one small and the other large (left). With properties set, the buttons take an equal full column width (right).

The following demonstrates a frame layout that contains two `TextView`s and an `ImageView`:

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

  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="This text is under the image in the stack"/>

  <ImageView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:src="@drawable/car"
    android:layout_gravity="center"/>

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```