ATLS 4120/5120: Mobile Application Development
Week 10: Android Layouts

Layouts

A layout defines the visual structure for a user interface and acts as a container for your UI widgets.
https://developer.android.com/guide/topics/ui/declaring-layout.html

Relative layout

Relative layouts are the default when creating a project in Android Studio. They enable you to specify the location of widgets relative to each other or to the parent.

- In relative layouts you can position views relative to a parent
  - layout_alignParent (Top, Bottom, Start, End)
    - Start is left for a left to right language like English, and End is right
  - center (Horizontal, Vertical, or combined with InParent)
- You can also position views relative to other views using their IDs
  - layout_above or below another view
  - layout_toStartOf or toEndOf to line up left or right edges
  - layout_toLeftOf or toRightOf to place next to another widget on the left or right
  - layout_align (Top, Bottom, Start, End, Baseline) aligns the edge to another widget
    - layout_alignBaseLine is great to align TextView and EditText widgets

Attributes

- android:layout_width and android:layout_height are required attributes for almost all widgets
  - match_parent tells your widget to become as big as its parent view group will allow, which in this case is the screen of the device, so it will use the whole width of the screen(not advisable for height)
    - (Fill_parent is deprecated, equivalent to match_parent)
  - wrap_content tells your widget to size itself to the dimensions required by its content.
- android: padding
  - expands the size of the widget to allow for padding from the inside edge of the widget to the content in the widget
  - paddingLeft, paddingRight, paddingTop, paddingBottom
  - Padding can be used with any layout or view
- Margins add room between widgets
  - layout_margin (Top, Bottom, Right, Left)

Linear Layout

Linear layouts organize its widgets into a single horizontal or vertical row.

- It creates a scrollbar if the length of the window exceeds the length of the screen.
- Linear layouts can be manipulated using different attributes on its components
- android:orientation determines which direction you want to arrange views in (required)
  - vertical: views are displayed in a single column
  - horizontal: views are displayed in a single row

    - When the orientation is horizontal you usually won't want the layout_height to be match_parent or it will use up the whole view. Use wrap_content so it only uses the space needed
    - The opposite is true for vertical orientation
- android:layout:weight helps allocate space across multiple widgets all with the match_parent fill value
    - value is a number that is then use to proportionally divide the space
    - Ex: Two widgets assign one android:layout:weight=1 and the other android:layout:weight=7. The first would get 1/8 the space, the second 7/8 of the space
    - default is 0 which gives just enough room for its contents
    - If only 1 view has a weight of 1 it will get all the extra space
    - The layout will also ensure there is room for all the widgets
- The default for linear layouts is to align all its widgets from the top for vertical, and from the left for horizontal(English). Use the android:layout_gravity attribute to change that.
    - Vertical values: left, center_horizontal, right
    - For horizontal the default is to align with respect to the base of the text of your widget. Use center_vertical to use the widget's center instead
    - Linear layouts use gravity instead of align so you can't use the android:layout_align attributes
- The android:gravity attribute lets you specify how you want to position the contents of a view inside the view https://developer.android.com/reference/android/widget/LinearLayout.LayoutParams.html
    - top, bottom, left, right, and others
- android:layout_gravity deals with the placement of the view itself, whereas android:gravity deals with how to display the view contents.

Table Layout
https://developer.android.com/guide/topics/ui/layout/grid.html
Table layouts organize its views into rows and columns just like (old) HTML tables. You cannot control the number of columns a table has, that is implicitly determined by the row with the most cells.

- TableRow objects define a single row or a table and are the child views of a TableLayout
- By default, Android fills widgets into columns starting with the first column, which starts at 0
- You can specify the column for a widget using android: layout_column
- Cells can span multiple columns using android:layout_span
- Cells cannot span multiple rows
- Column width is determined by the largest widget in that column, similar to wrap_content. You can guide it using android:stretchColumns, android:shrinkColumns, or android:CollapseColumns (covered in the book)
- You can intersperse widgets in your table that aren't in rows, they will be treated as if they were in a linear layout with vertical spacing

Grid Layout
https://developer.android.com/guide/topics/ui/layout/gridview.html
https://developer.android.com/reference/android/widget/GridLayout.html
Grid layouts place its views into a grid of cells.

- Layout parameters rowspec and columnSpec can control the number of rows and columns
- android:layout_gravity determines how a component is positioned in a group of cells
    - Used in place of android:layout_weight

You can also build layouts dynamically using adapters. This is useful when your layout is not pre-determined and you want to populate it at run time. We'll get into adapters next semester.

Constraint layout is new in Android Sutdio 2.2 and we will look at that separately.

Feelings
Let's play around with the layout in our Feelings app.
I'm going to make a copy of it in the Finder so I keep my original version.
Feelings layout.
All of our changes are going to be in activity_main.xml in the text view.
Remember that you can move around the xml, the order doesn't matter.

1. Right now the name edittext takes up a lot of room and doesn't really need it so let's put the edittext and togglebutton on the same line using a linear layout.
When you add a LinearLayout tag in the xml you'll notice it requires the layout_width and layout_height attributes that all views require. We want it to use the full width of the screen but not the full height. So layout_width will use match_parent and layout_height will use wrap_content.
We also need to give it an orientation.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
```

Move the edittext and togglebutton in this tag and put the </LinearLayout> tag after.

Now you'll notice you have some errors. That's because we had some attributes on the editext and togglebutton for the relative layout but they're not valid for a linear layout. Delete the highlighted attributes:

```
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true"
android:layout_below="@+id/spinner"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
```

These should now be on the same line.
If you want to give the edittext more room add

```
android:layout_weight="1"
```

You’ll still have other errors, those are because the relative layout used the edittext and togglebutton as a way to place the other elements but now those are not children of the relative layout so they can’t be used. We’ll fix each of these in turn.

2. We could use another linear layout for the radio group and its textview but instead let’s try a grid layout. (In a real app you want to mix as few layouts as possible, but we’re just playing here)
We need to tell it we want it below the first linear layout so let’s give the first linear layout an id.

```
android:id="@+id/linear1"
```

```
<GridLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_below="@id/linear1"
   android:id="@+id/grid_radio">
```

Put the textview, radiogroup and all the radiobutton tags in the gridlayout and then the </GridLayout> tag.

Again delete all the yellow highlighted attributes.

```
android:layout_below="@id/energyToggleButton"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
android:layout_alignTop="@+id/textView3"
android:layout_toRightOf="@+id/energyToggleButton"
android:layout_toEndOf="@+id/energyToggleButton"
```

This moves them under the linear layout but they’re a little squished. You can add a margin using marginStart (or marginLeft for APIs before 17) to space them out a bit.

```
android:layout_marginStart="20dp"
```

The grid layout is a little close to the edit text and toggle button above it so I added a margin to the gridlayout

```
android:layout_marginTop="20dp"
```

Note that this is the margin to the linearlayout since that’s what layout_below refers to.

3. We could use two linear layouts to put the checkboxes on two rows but instead let’s try a table layout.

```
<TableLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_below="@id/grid_radio"
android:id="@+id/table_check" >
```

Then add a row <TableRow> put two check boxes in it then </TableRow>
Repeat for the other two checkboxes and then close the table </TableLayout>

Again delete all the yellow highlighted attributes.

```
android:layout_below="@+id/textView3"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
android:layout_alignTop="@+id/checkBox1"
android:layout_alignParentRight="true"
android:layout_alignParentEnd="true"
android:layout_above="@+id/button"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
android:layout_above="@+id/button"
android:layout_alignLeft="@+id/checkBox2"
android:layout_alignStart="@+id/checkBox2"
```

Also take out any margin attributes as we want the table rows to line up.
You can again add a margin to the left

```
android:layout_marginStart="20dp"
```

To move the tablelayout down a bit I added a top margin

```
android:layout_marginTop="20dp"
```

4. Let's put the two remaining widgets, the spinner and switch in a linear layout. I like the spinner here because when the user taps it and the list expands down, it doesn't cover any other widget.

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:id="@+id/linear2"
   android:layout_below="@id/table_check"
   android:orientation="horizontal" >
```

Put the spinner and switch in the linear layout and then close it </LinearLayout>

Again delete all the yellow highlighted attributes.

```
android:layout_alignBaseline="@+id/energyToggleButton"
android:layout_alignBottom="@+id/energyToggleButton"
android:layout_alignParentRight="true"
android:layout_alignParentEnd="true"
```

```
android:layout_below="@+id/nameEditText"
android:layout_centerHorizontal="true"
```

Also remove any margins as we want the linear layout to handle it.

And again I added a top margin to the layout to space it from the table above it.

```
    android:layout_marginTop="20dp">
```

If the spinner is very stretched out make sure it has layout_width="wrap_content" so it only takes the space it needs.
You can move the switch over by adding a margin

```
android:layout_marginStart="50dp"
```

5. We should just have our button and textview left in our original relative layout. Update them as needed.

```
<Button
    android:text="@string/mood_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button"
    android:onClick="findMood"
    android:layout_below="@id/linear2"
    android:layout_marginTop="25dp"
    android:layout_centerHorizontal="true" />

<TextView
    android:text="@string/feeling"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/feelingTextView"
    android:layout_below="@id/button"
    android:layout_marginTop="25dp"
    android:textSize="20sp"
    android:layout_centerHorizontal="true" />
```