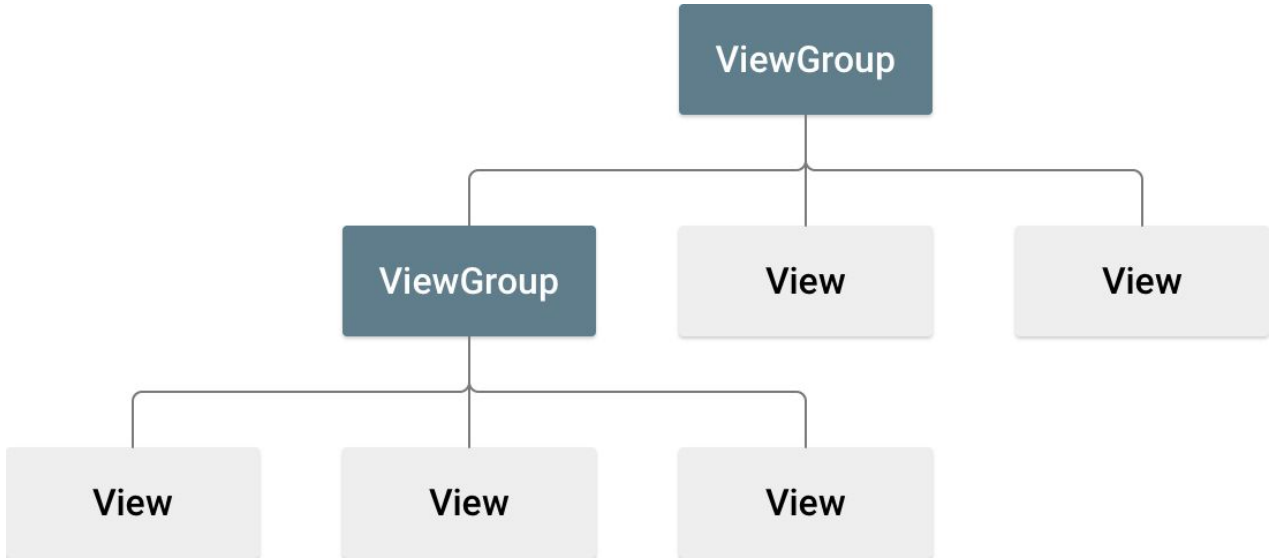
Android Views

UI Design - Views and Layouts

- A layout (`ViewGroup`) defines the structure of the UI.
  - Containers that group one or more widgets (`View`).
    - A button, a text box.
- Many pre-defined types of layouts (`LinearLayout`, `Constraint Layout`).
- UI elements can be declared in XML or in code.

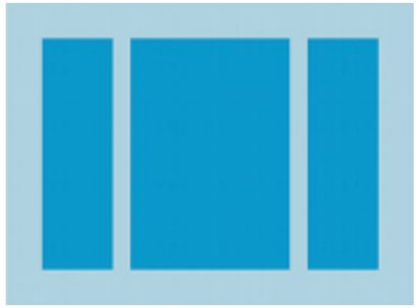


UI Design - Layouts

- Views are rectangles with left and top coordinates.
  - Can get location with `getLeft()` and `getTop()`
  - Defined relative to the parent.
- Size is defined in width and height.
  - Measured width/height are how big the view *wants* to be.
  - Drawing width/height are the actual size of the view on screen, after layout constraints.
  - These can differ.

UI Design - Common Layouts

Linear Layout



A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

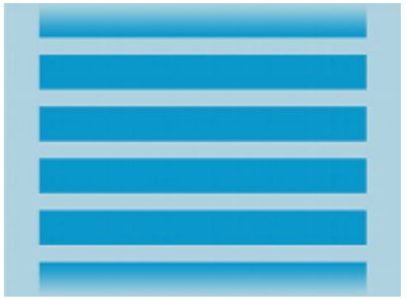
Relative Layout

Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Web View

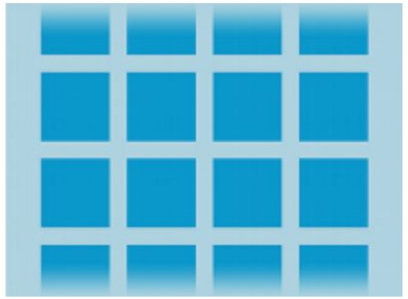
Displays web pages.

List View



Displays a scrolling single column list.

Grid View



Displays a scrolling grid of columns and rows.

Built from data using an Adapter

1	Linear Layout LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	Relative Layout RelativeLayout is a view group that displays child views in relative positions.
3	Table Layout TableLayout is a view that groups views into rows and columns.
4	Absolute Layout AbsoluteLayout enables you to specify the exact location of its children.
5	Frame Layout The FrameLayout is a placeholder on screen that you can use to display a single view.
6	List View ListView is a view group that displays a list of scrollable items.
7	Grid View GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Layout Managers

A view that controls how the sub-views are arranged

- **LinearLayout**
  - components arranged horizontally or vertically
  - Similar to Java BoxLayout
- **TableLayout**
  - Specify rows and columns
  - Similar to laying out HTML tables
- **RelativeLayout:** positioning relative to other components
  - layout_centerInParent, alignParentTop, alignParentBottom
  - layout_above, layout_toRightOf, layout_toLeftOf, layout_below
- **FrameLayout**
  - Multiple components in the same space
  - Similar to Java OverlayLayout and CardLayout

UI Design - XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Relative Layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <TextView
      android:id="@+id/label"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Type here:"/>
   <EditText
      android:id="@+id/entry"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:background="@android:drawable/editbox_background"
      android:layout_below="@id/label"/>
   <Button
      android:id="@+id/ok"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_below="@id/entry"
      android:layout_alignParentRight="true"
      android:layout_marginLeft="10dip"
      android:text="OK" />
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_toLeftOf="@id/ok"
      android:layout_alignTop="@id/ok"
      android:text="Cancel" />
</RelativeLayout>
```

RelativeLayout – how it works

Parameters in XML *(or can map to method calls in Java RelativeLayout class)*

- Position relative to Parent

android:layout_alignParentTop, **android:layout_alignParentBottom, android:layout_alignParentLeft, android:layout_alignParentRight**

VALUE = 'true' ---If "true", moves to that edge of Parent

android:layout_centerVertical

VALUE= "true" -- If "true", centers this child vertically within its parent.

- Position relative to another widget

android:layout_below, **android:layout_above**, **android:layout_toLeftOf, android:layout_toRightOf**

VALUE="resource ID of other widget" -- Positions the top edge of this view below/aboveof the view specified with a resource ID.

OR Positions the left edge of this view to the left/right of the view specified with a resource ID.

RelativeLayout – how it works

Example

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="16dp"
   android:paddingRight="16dp" >
   <EditText
      android:id="@+id/name"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/reminder" />
   <Spinner
      android:id="@+id/dates"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_below="@id/name"
      android:layout_alignParentLeft="true"
      android:layout_toLeftOf="@+id/times"/>
   <Spinner
      android:id="@id/times"
      android:layout_width="96dp"
      android:layout_height="wrap_content"
      android:layout_below="@id/name"
      android:layout_alignParentRight="true"/>
   <Button
      android:layout_width="96dp"
      android:layout_height="wrap_content"
      android:layout_below="@id/times"
      android:layout_alignParentRight="true"
      android:text="@string/done" />
</RelativeLayout>
```

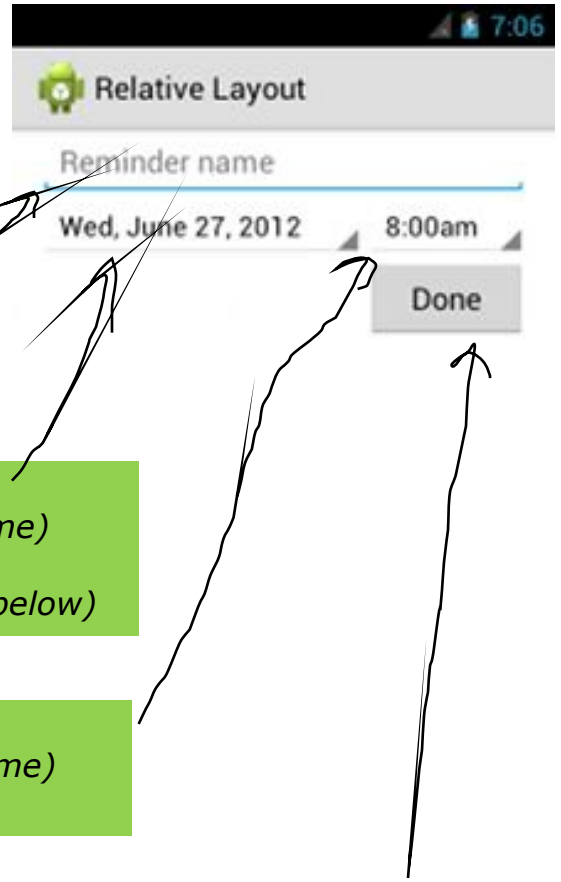
Says we have RelativeLayout that width and height match parent (which is the entire app screen)

1st View object in RelativeLayout will be at the top and is the EditText

2nd View object here is specified to be **below the 1st object** *EditText (id = name)* & **aligned to left of parent(app)** & **Left of** *the Button with id=times (see below)*

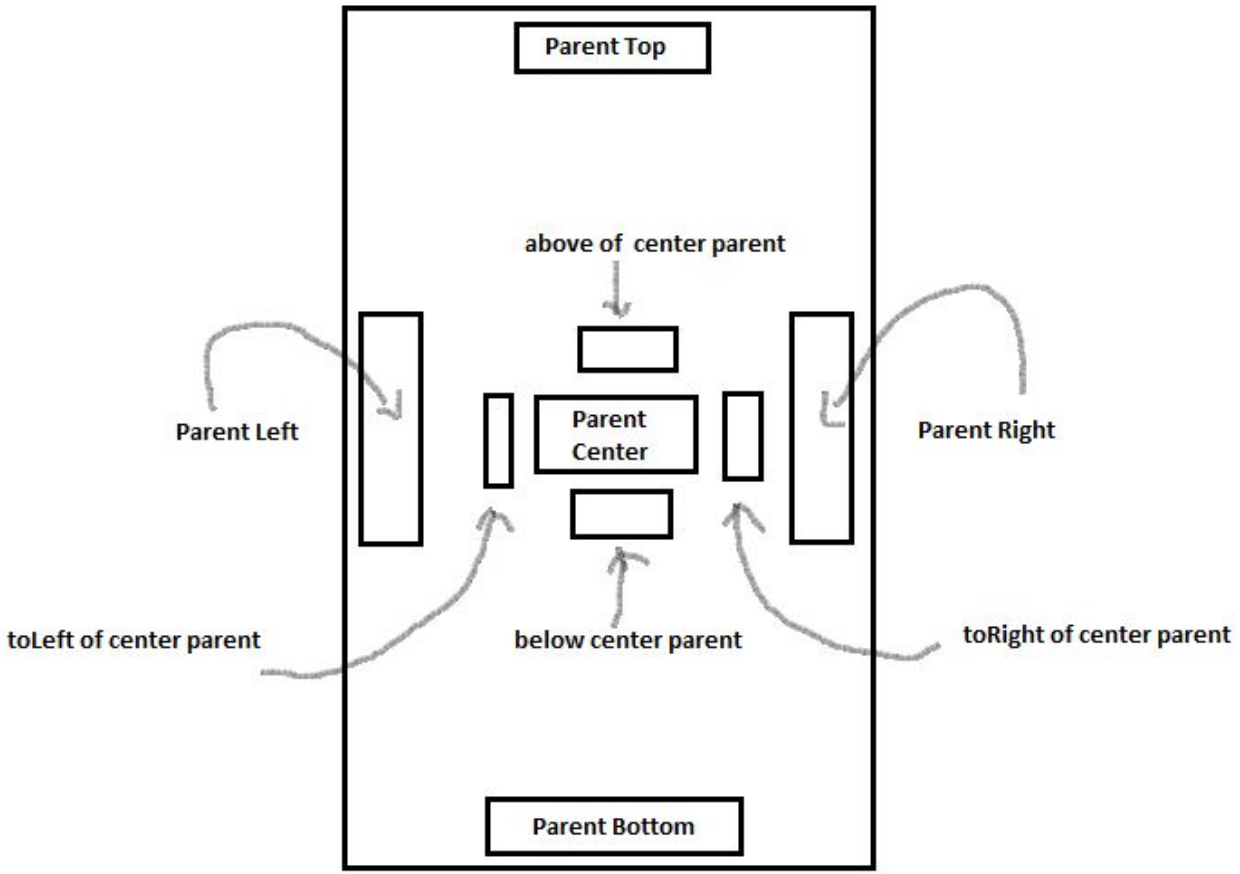
3rd View object here is specified to be **below the 1st object** *EditText (id = name)* & **aligned to left of parent(app)**

4th View object here is specified to be **below the 2nd object** *Spinner (id = times)* & **aligned to right of parent(app)**



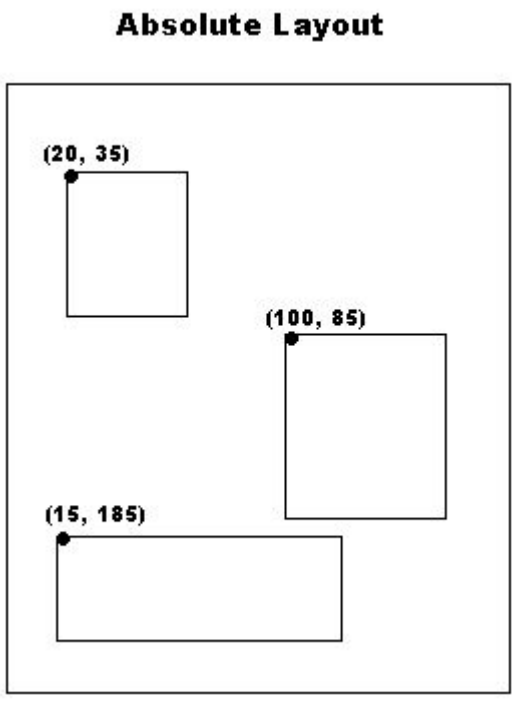
More on RelativeLayout parameters

- Center
  Top
  Bottom
  of
  Parent



Absolute Layout

An Absolute Layout lets you specify exact locations (x/y coordinates) of its children. Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.




```
<AbsoluteLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
   <Button
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:text="OK"
    android:layout_x="50px"
    android:layout_y="361px" />
  <Button
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:text="Cancel"
    android:layout_x="225px"
    android:layout_y="361px" />
</AbsoluteLayout>
```

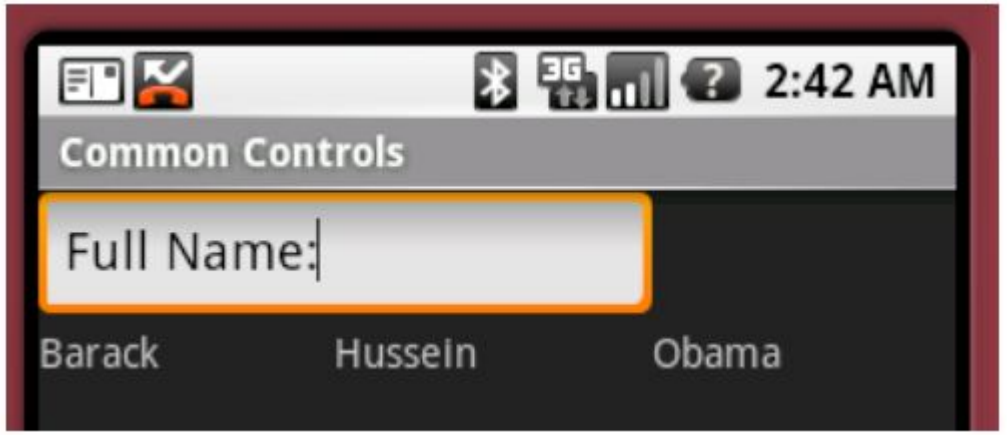
Table Layout

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
      <TextView
        android:layout_column="1"
        android:text="Open..."
        android:padding="3dip" />
      <TextView
        android:text="Ctrl-O"
        android:gravity="right"
        android:padding="3dip" />
    </TableRow>

    <TableRow>
      <TextView
        android:layout_column="1"
        android:text="Save..."
        android:padding="3dip" />
      <TextView
        android:text="Ctrl-S"
        android:gravity="right"
        android:padding="3dip" />
    </TableRow>

    <TableRow>
      <TextView
        android:layout_column="1"
        android:text="Save As..."
        android:padding="3dip" />
      <TextView
        android:text="Ctrl-Shift-S"
        android:gravity="right"
        android:padding="3dip" />
    </TableRow> </TableLayout>
```

TableLayout Example



One row/ three columns
Stretch to fill entire width

```
<TableLayout xmlns:android=http://sche...
   android:layout_width= "match_parent" android:layout_height= "match_parent"
   android:stretchColumns="0,1,2"> <-- indicate which columns to stretch -->
   <EditText  android:text="Full Name:"/>
   <TableRow>
      <TextView android:layout_width="wrap_content"
          android:layout_height="wrap_content" android:text="Barack"/>
      <TextView android:layout_width="wrap_content"
          android:layout_height="wrap_content" android:text="Hussein"/>
      <TextView android:layout_width="wrap_content"
          android:layout_height="wrap_content" android:text="Obama"/>
   </TableRow>
</TableLayout>
```

android:layout_span="2" to span two view columns

UI Design - Responsive Design

- Android defines two characteristics for each screen:
  - Screen Size (physical size)
    - Small, Normal, Large, XLarge
  - Screen Density (density of pixels on screen)
    - MDPI (~160dpi), HDPI (~240dpi), XHDPI (~320dpi), XXHDPI (~480dpi), XXXHDPI (~640dpi)
- Apps are compatible with all screen sizes and densities automatically, but this may not create a good UX.
  - Create specialized layouts, optimize images for density.

Creating a Flexible Layout

- `ConstraintLayout` allows position/size specification based on spatial relationships between views.
  - All views move together as screen size changes.
  - Easiest to create in Android Studio Layout Editor.
- Avoid hard-coded layout sizes.
  - Use wrap_content, match_parent.
  - Automatically adjusts based on size and orientation of screen.

Complex Views

Displays a set of data using a single view

- **ListView:**
  A scrollable list displayed vertically
- **GridView:**
  A scrollable list of items displayed with a fixed number of columns
- **SpinnerView: (combo box / dropdownlist box)**
  A scrollable drop down menu of choices
- **GalleryView: (now deprecated, use ViewPager)**
  Horizontally scrollable list focusing on the center of the list

AdapterViews

Adapter: object that formats the display of a single cell of a complex view

- **Purposes**
  - Connect a complex view to the data
  - Define the display format of a particular cell, which can contain a variety of sub views
  - Reuse the cell layout component for memory, performance efficiency
- **Adapters Required Because:** It is unworkable to display a large lists with thousands of components, each requiring megabytes of memory.
- **Built in Android adapter classes**
  - *ArrayAdapter:* Data items stored in an array
  - *SimpleCursorAdapter:* Data are rows of an SQL table
- ***Custom Adapter:*** Class that overrides various adapter methods

The Need for An Adapter

The Adapter Pattern

"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interface"

- Design Patterns: Elements of Reusable Object Oriented Software

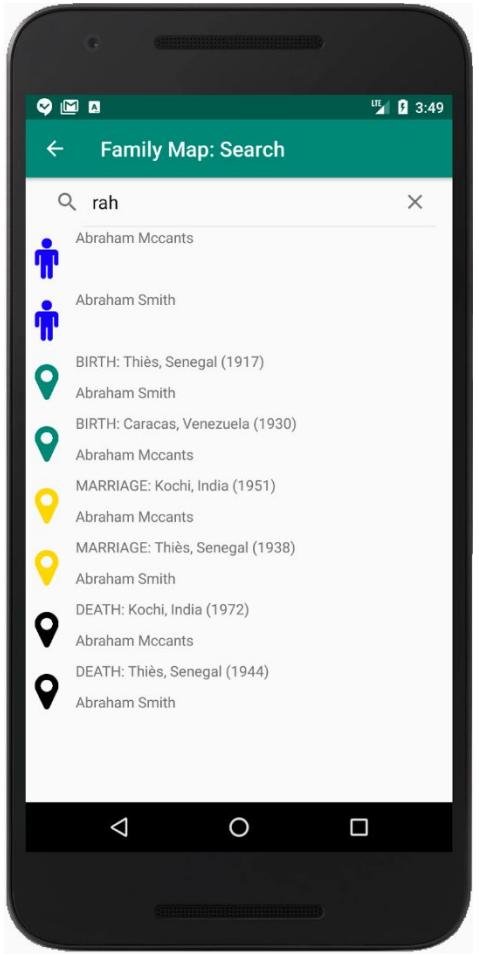
- Client provides an interface for the adaptee to implement
- Adaptee implements the interface, providing methods the client can call
- Client may also provide an abstract class that implements all or part of the interface that the adaptee can extend

Advanced Views

- RecyclerView
  - For displaying an arbitrarily long repeating list of ViewGroups
  - Examples: Family Map SearchActivity
- ExpandableListView
  - For displaying an expandable lists of items, or multiple expandable lists of items by item group
  - Example: Family Map PersonActivity
- MapView
  - For displaying a map

RecyclerView

- We don’t know how many search results will be returned, but we will repeat a set of Views with a specific layout for each result



RecyclerView

- Makes efficient use of the views for each item by recycling (reusing) them as we scroll through the list
  - We may have a very long list with only a subset visible at any moment
- Needs to make specific requests into our model of people and events to respond to scrolling, etc. happening in the UI
- Needs to be able to request a UI layout for a specific item it needs to display
- Requires the ability to adapt the interface provided by our model to the needs of the view

RecyclerView Layouts

1. Provide a layout for the Activity or Fragment that needs to display the RecyclerView
   - Place a RecyclerView in that layout
2. Provide one or more layouts for the items to be displayed in the RecyclerView
   - One layout if all items to be displayed have the same layout
   - One for each type of item to be displayed if different items have different layouts