Graphical User Interfaces

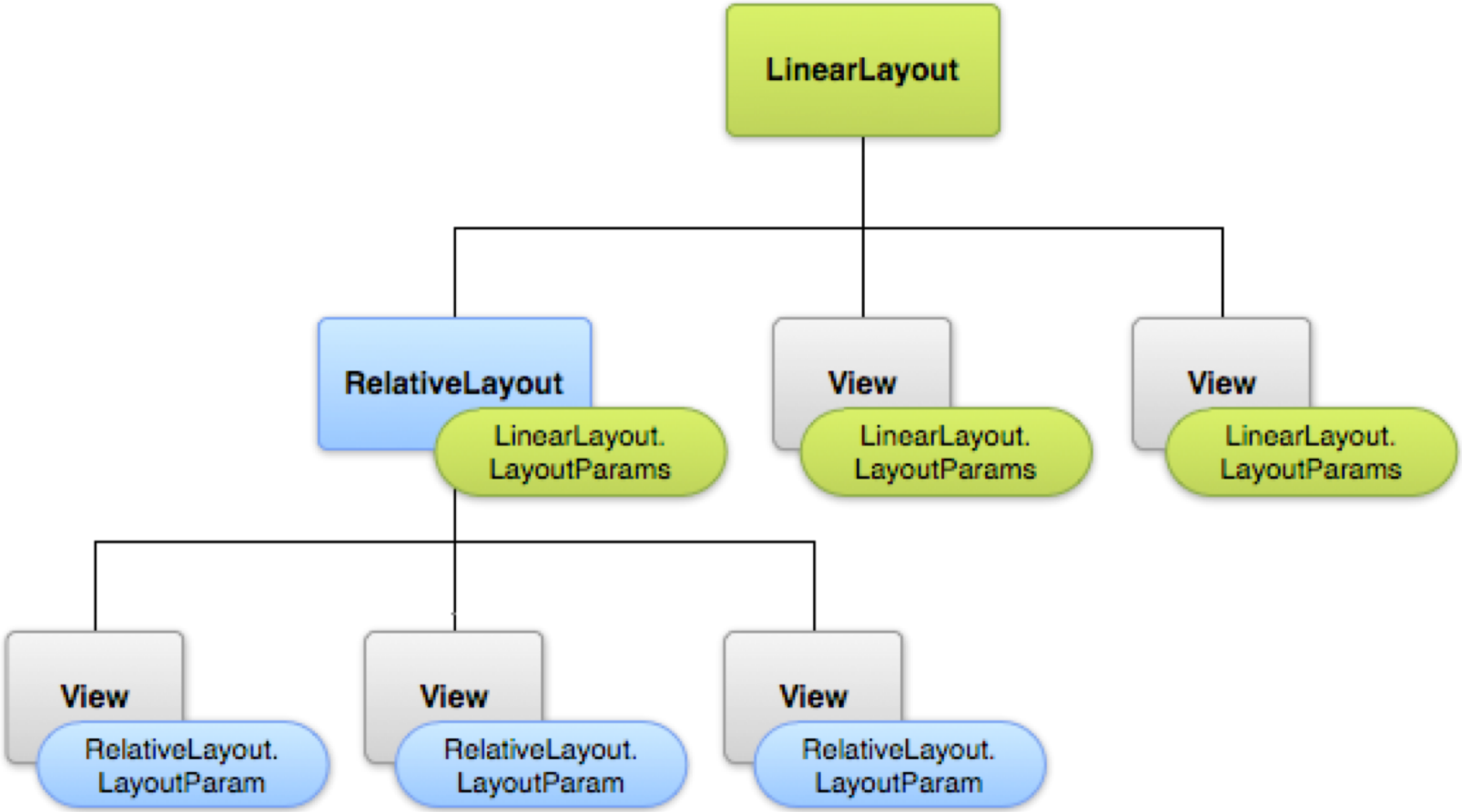
Some suggestions

- Avoid displaying too many things
  - Well-known anti-patterns
- Display useful content on your start screen
- Use action bars to provide consistent navigation
- Keep your hierarchies shallow by using horizontal navigation and shortcuts

Layouts

- A layout is a ViewGroup that is responsible for positioning its child Views. It calculates and set the position and size of those Views
  - Measure: Determines the view size
  - Layout: Sets position and size for the view and all its descendants
  - Draw: Draws the view on the screen

Layouts


LinearLayout
RelativeLayout
LinearLayout.
LayoutParams
View
LinearLayout.
LayoutParams
View
LinearLayout.
LayoutParams
View
RelativeLayout.
LayoutParam
View
RelativeLayout.
LayoutParam
View
RelativeLayout.
LayoutParam

Two ways

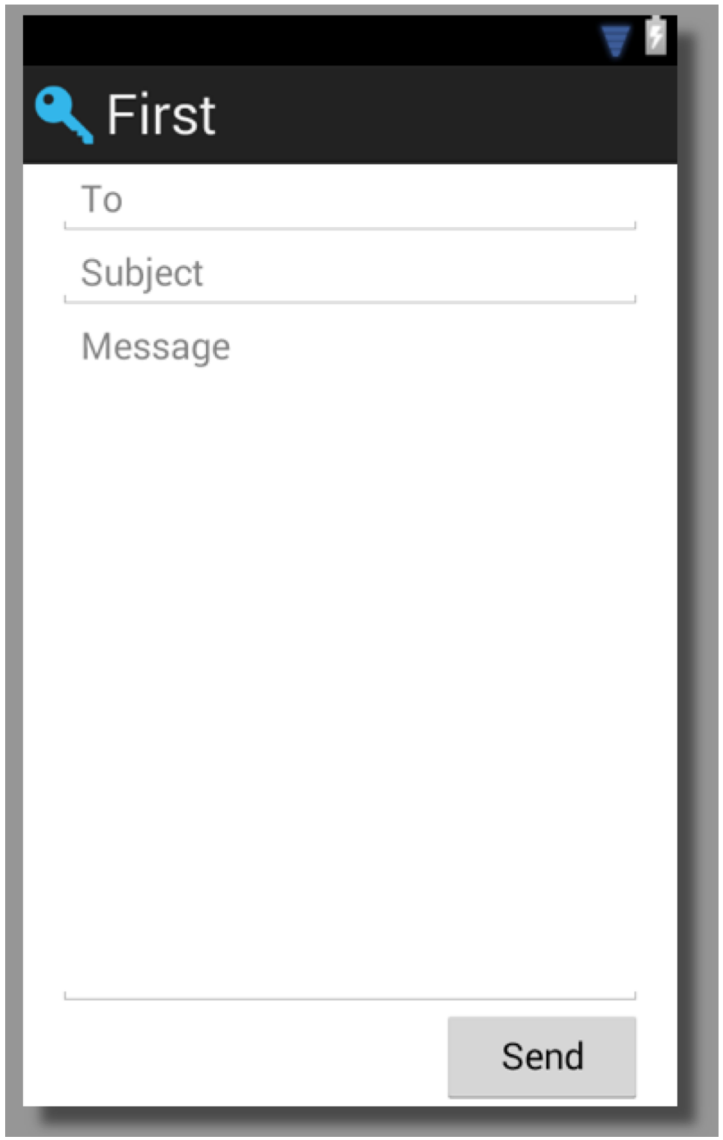
- Declare UI elements in XML
  - Created at build time
- Programmatically
  - Instantiated at runtime
- Together
  - Declare your application's default layouts in XML
  - Add code that modifies the state of the screen objects

LinearLayout

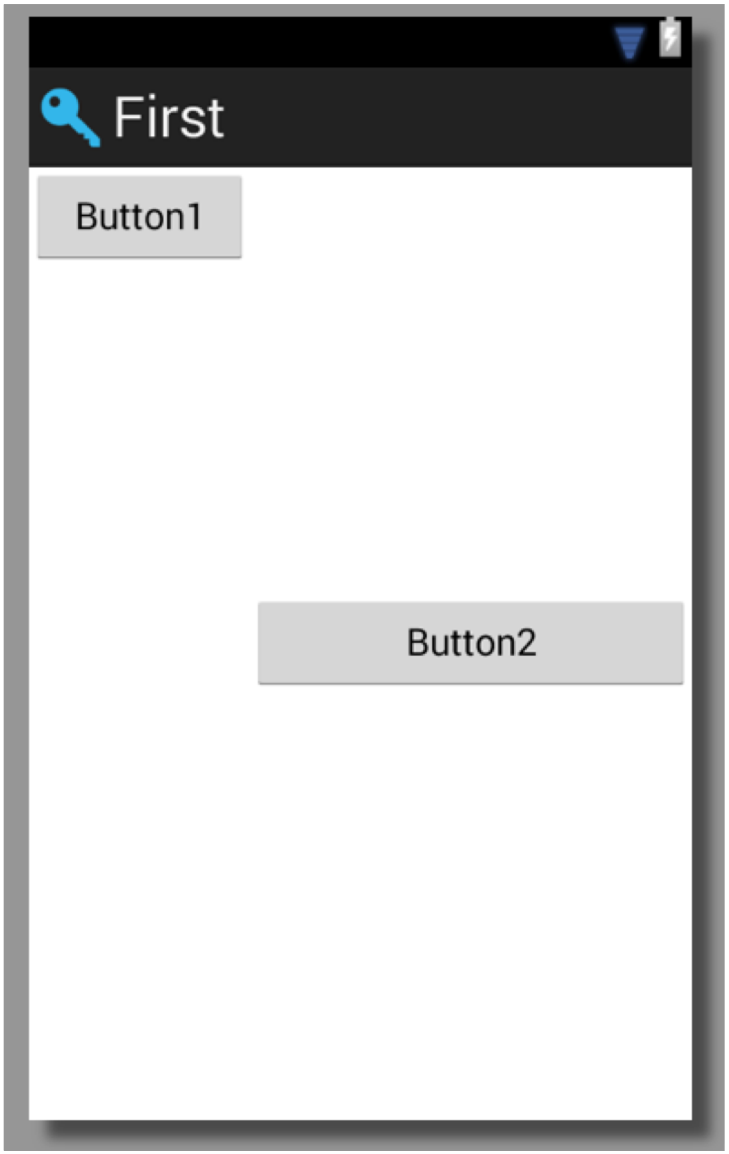
- Views presented on a single row/column
  - Defined by layout_orientation
  - Or setOrientation(int orientation)
    - Where orientation is either HORIZONTAL or VERTICAL
- Two further attributes
  - Gravity specifies how to position the child view wrt the parent
  - Weight indicates how much of the extra space in the LinearLayout will be allocated to the view

Example

```
<LinearLayout xmlns:android= …
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```



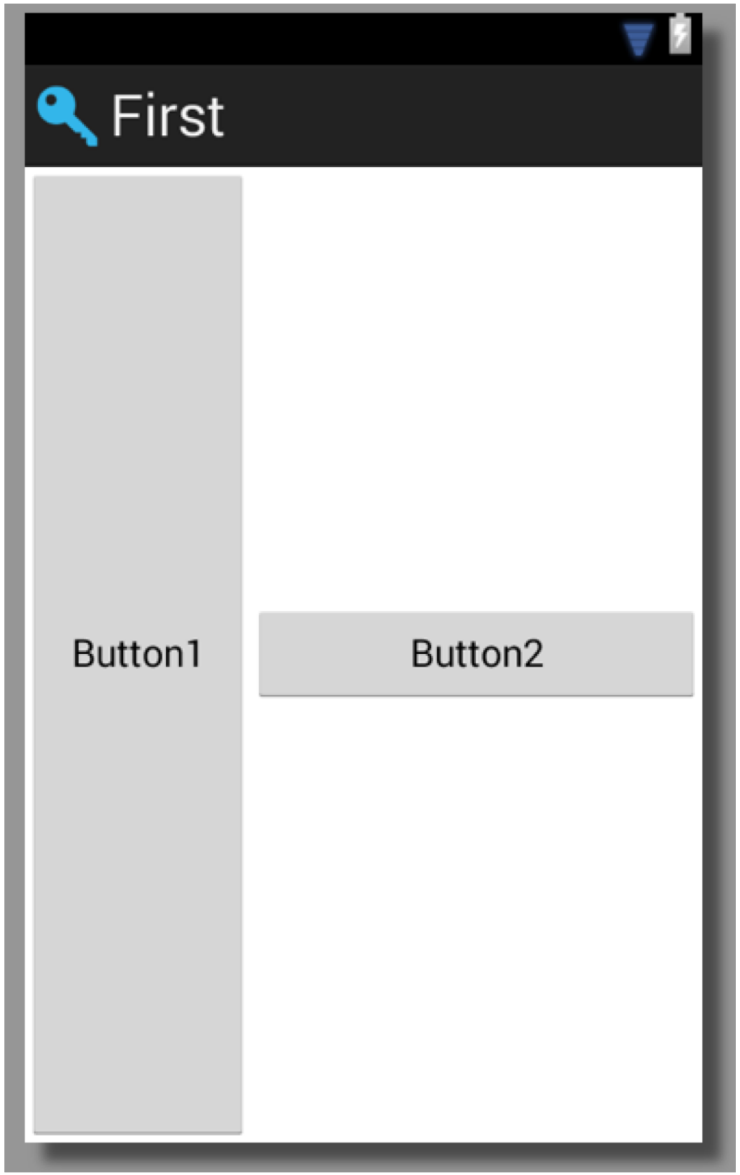
Example 2




```
<LinearLayout xmlns:android= …
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >
    <Button
        android:id="@+id/button1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="@string/b1"
        android:layout_weight="1" />
    <Button
        android:id="@+id/button2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_weight="2"
        android:gravity="center_vertical|center_horizontal"
        android:text="@string/b2" />
</LinearLayout>
```

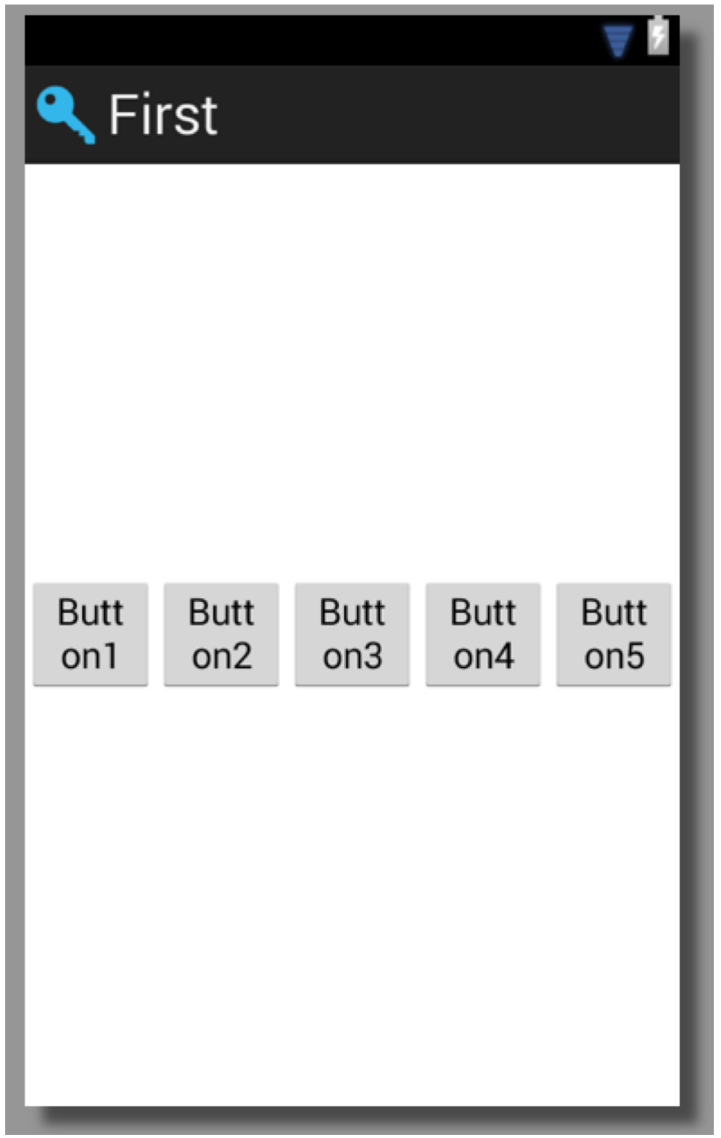
Example 3

```
<LinearLayout xmlns:android= …
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >
    <Button
        android:id="@+id/button1"
        android:layout_width="0dp"
        android:layout_height="”fill_parent"
        android:text="@string/b1"
        android:layout_weight="1" />
    <Button
        android:id="@+id/button2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_weight="2"
        android:gravity="center_vertical|center_horizontal"
        android:text="@string/b2" />
</LinearLayout>
```



Be careful

```
<Button
    android:id="@+id/button1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical"
    android:layout_weight="1"
    android:text="@string/b1" />
```



RelativeLayout

- Displays child views in relative positions
- The position of each view can be specified as
  - Relative to sibling elements (such as to the left-of or below another view)
  - Relative to the parent area (such as aligned to the bottom, left or center)
- Useful to align views
- Can eliminate nested view groups and keep your layout hierarchy flat

Some layout properties

- android:layout_alignParentTop (boolean)
- android:layout_centerVertical (boolean)
- android:layout_below (ID)
- android:layout_toRightOf (ID)

- Dependencies against other views in the layout can be declared in any order

```
<RelativeLayout xmlns:android= …
…
      <EditText
          android:id="@+id/name"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:hint="@string/reminder" />
      <Spinner
          android:id="@+id/options"
          android:layout_width="0dp"
          android:layout_height="wrap_content"
          android:layout_below="@id/name"
          android:layout_alignParentLeft="true"
          android:layout_toLeftOf="@+id/ok" />
       <EditText
          android:id="@+id/password”
          android:text="@string/password"
          android:inputType="textPassword"
          android:layout_below="@id/options"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_alignParentRight="true"
          android:layout_toRightOf="@+id/passwordLabel" />
      <TextView
          android:id="@id/passwordLabel"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_alignBaseline="@id/password"
          android:text="@string/password" />
       <Button
          android:id="@id/ok"
          android:layout_width="96dp"
          android:layout_height="wrap_content"
          android:layout_below="@id/name"
          android:layout_alignParentRight="true"
          android:text="@string/done" />
</RelativeLayout>
```

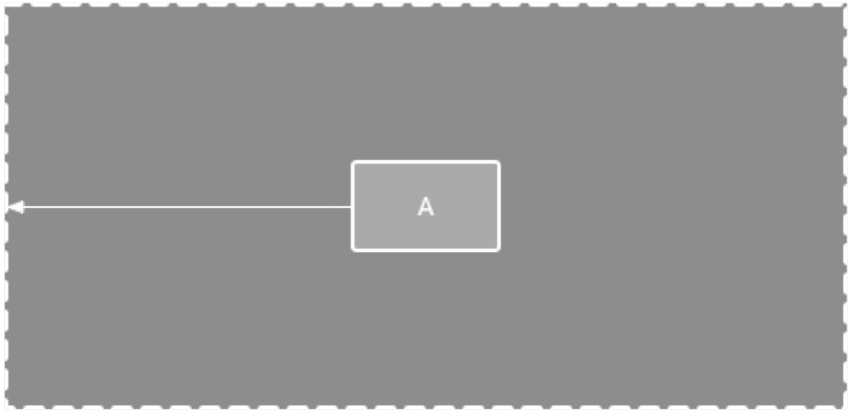
Example

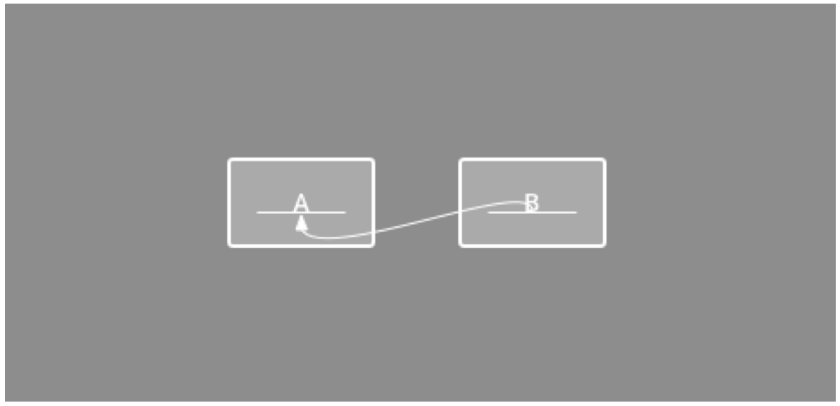
ConstraintLayout

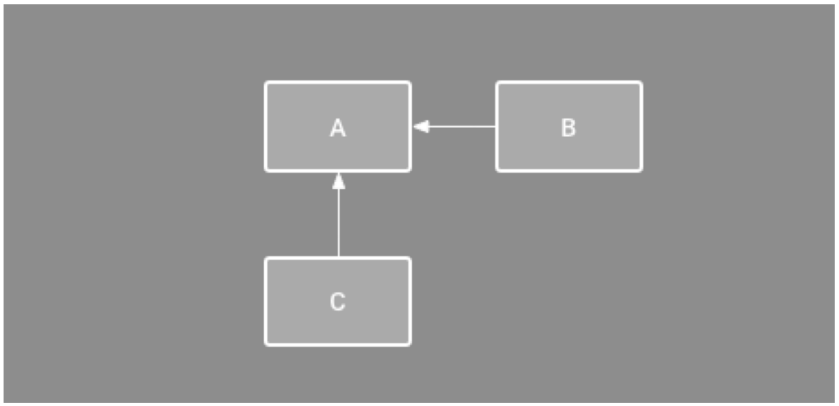
- Now default layout in Android Studio
- Designed to
  - Create efficient UI
  - Be a top-level, flat layout
    - Flat hierarchies display quick and are memory efficient
- Solution to address the performance problems of other layouts
  - Nested weighted Linear Layouts
  - Relative Layouts

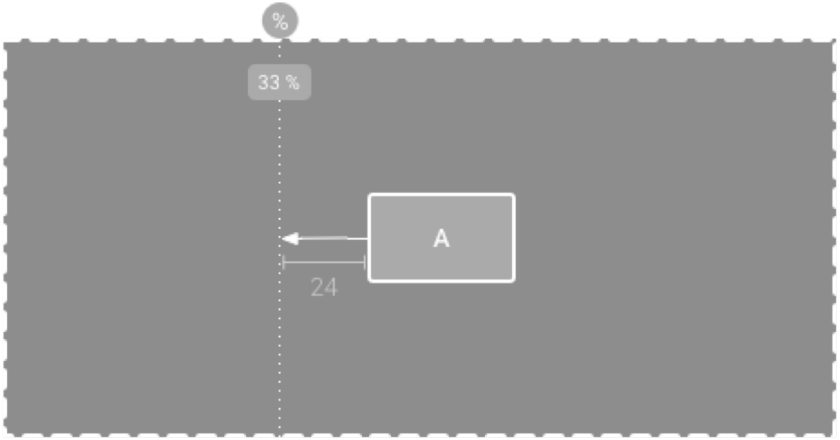
ConstraintLayout

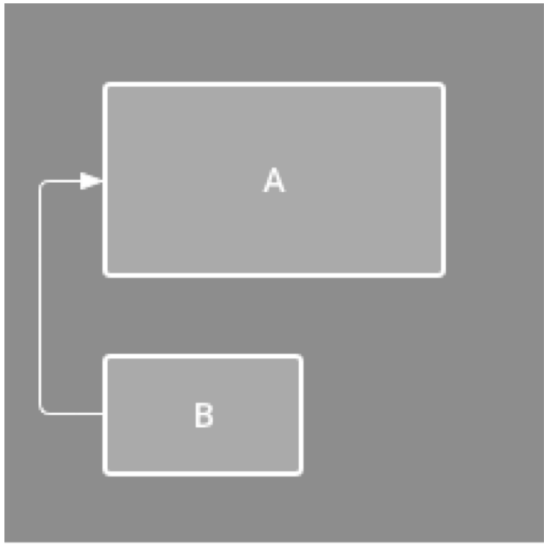
- One horizontal and one vertical constraint for the view
- Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline
- Each constraint defines the view's position along either the vertical or horizontal axis
  - Each view must have a minimum of one constraint for each axis, but often more are necessary.

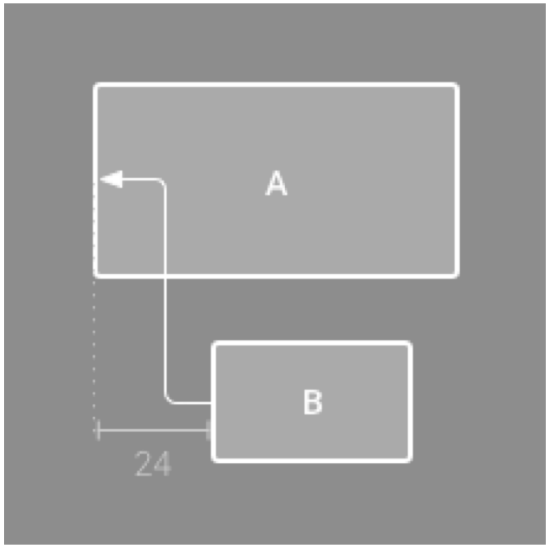

A


A
B

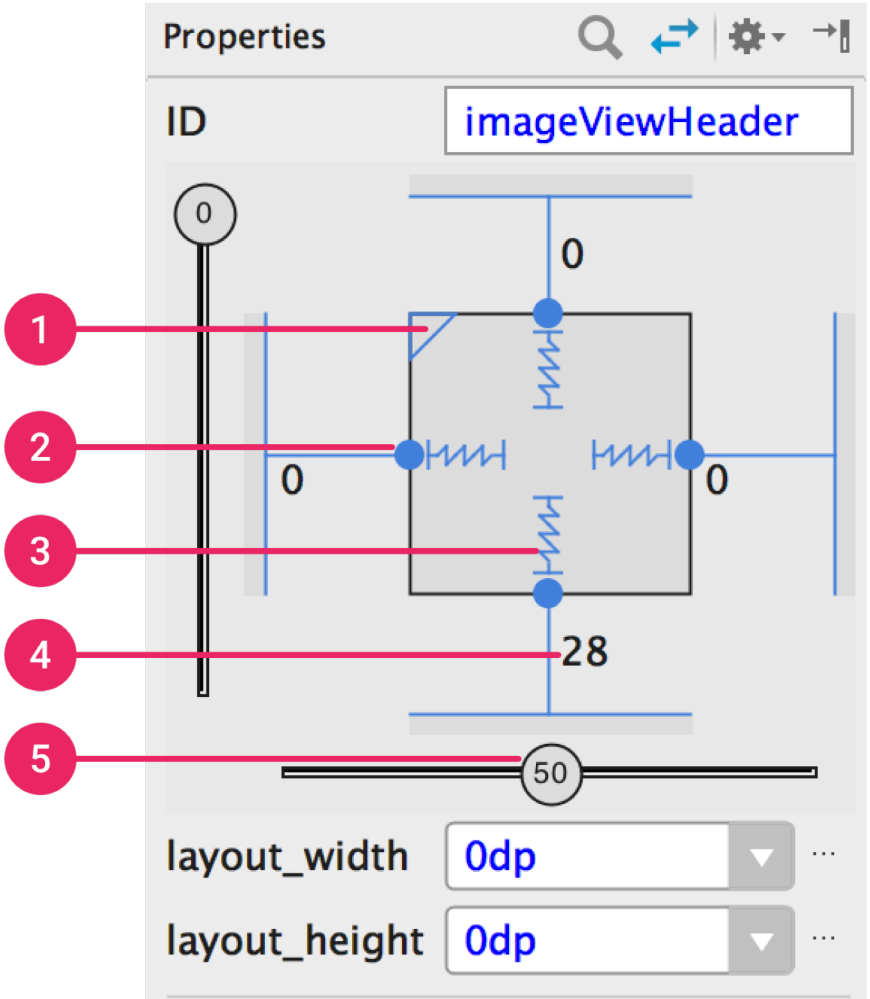

A
B
C

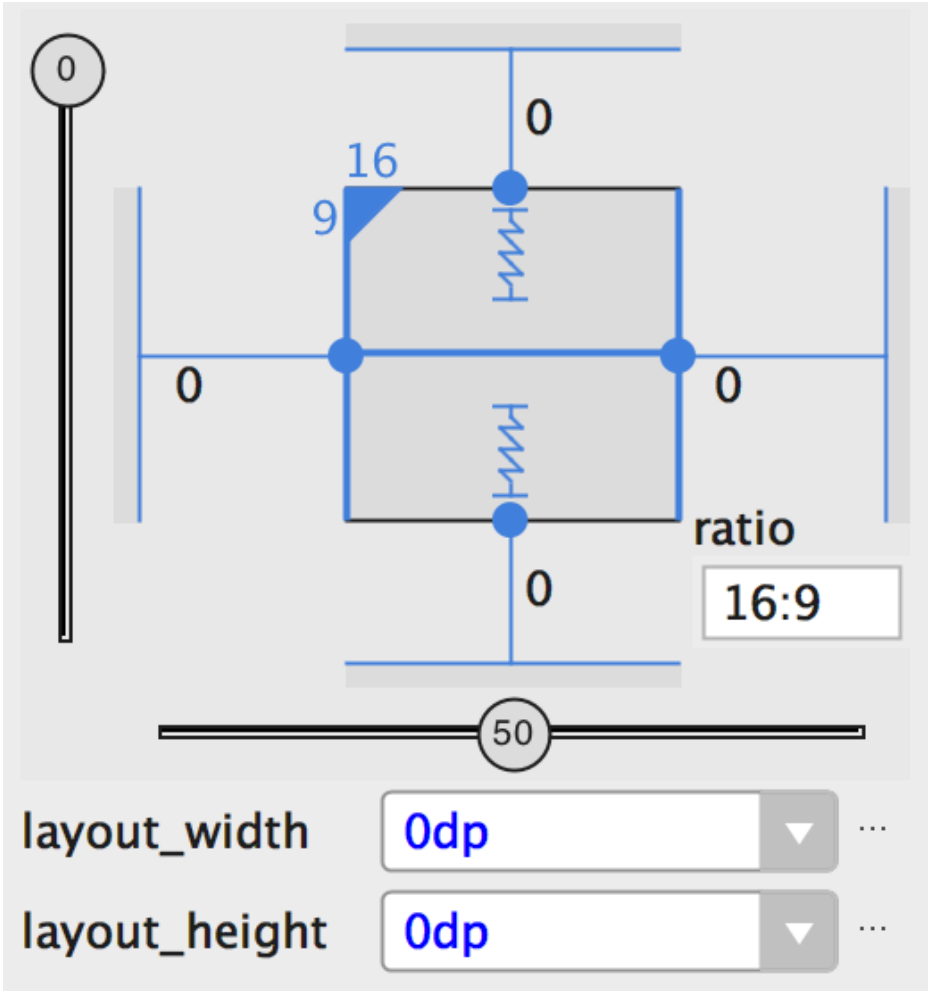

%
33 %
A
24


A
B


A
B
24

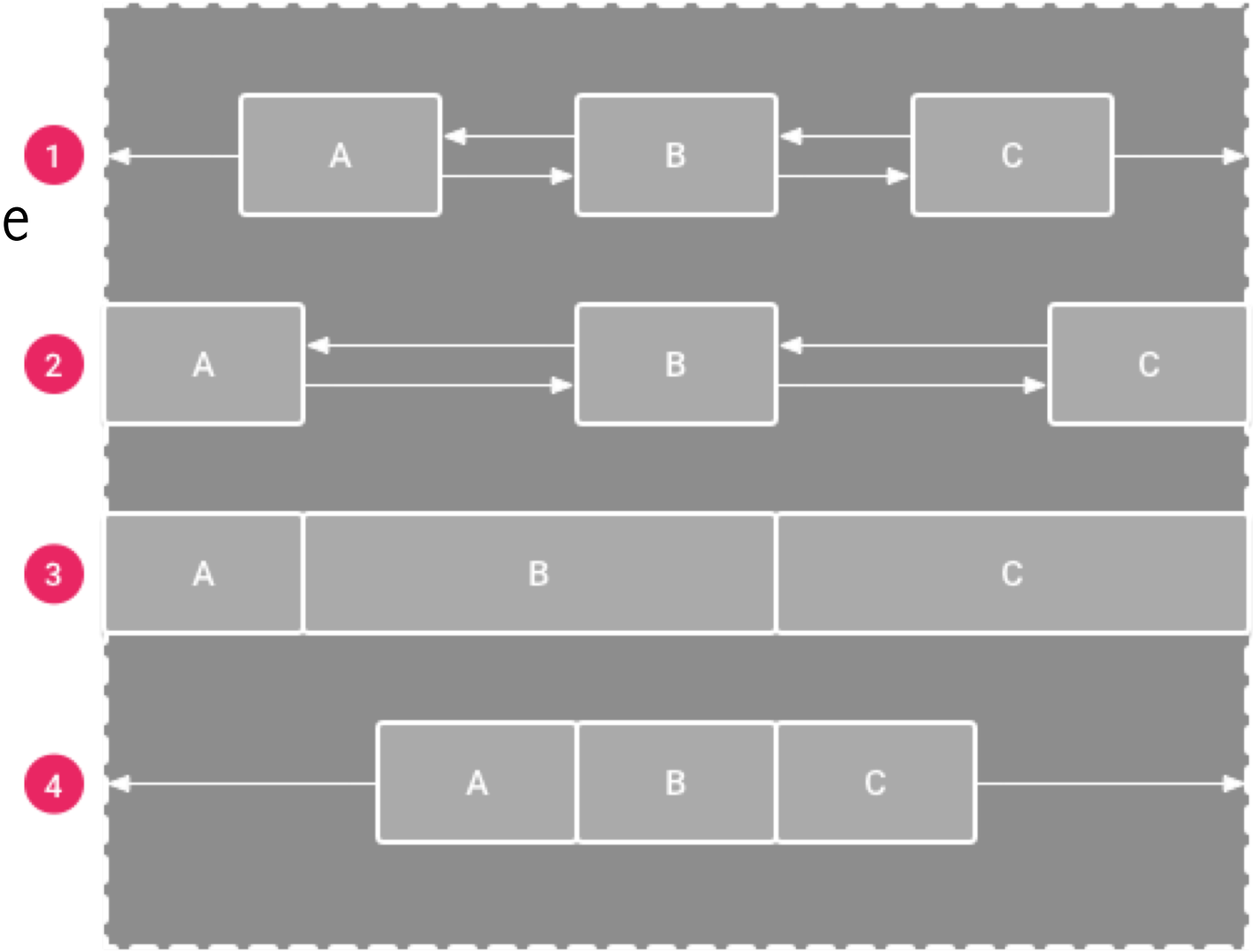
Adnroid Studio





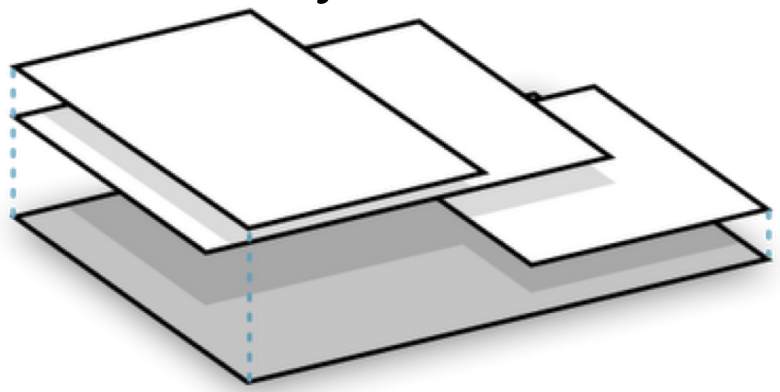
Chains

1. Spread
2. Spread inside
3. Weighted
4. Packed



FrameLayout

- Designed to display a single or multiple UI elements
  - The position of multiple children can be controlled by assigning a gravity to each child
  - Elements that overlap are displayed overlapping
- Child views are drawn in a stack, with the most recently added child on top
- Adds android:visibility to manage the visibility of views

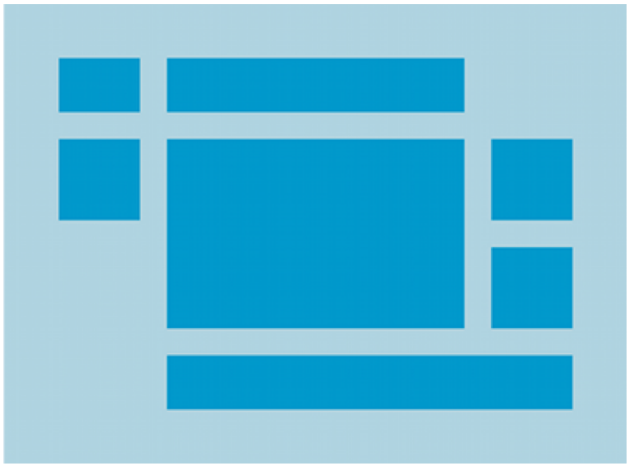


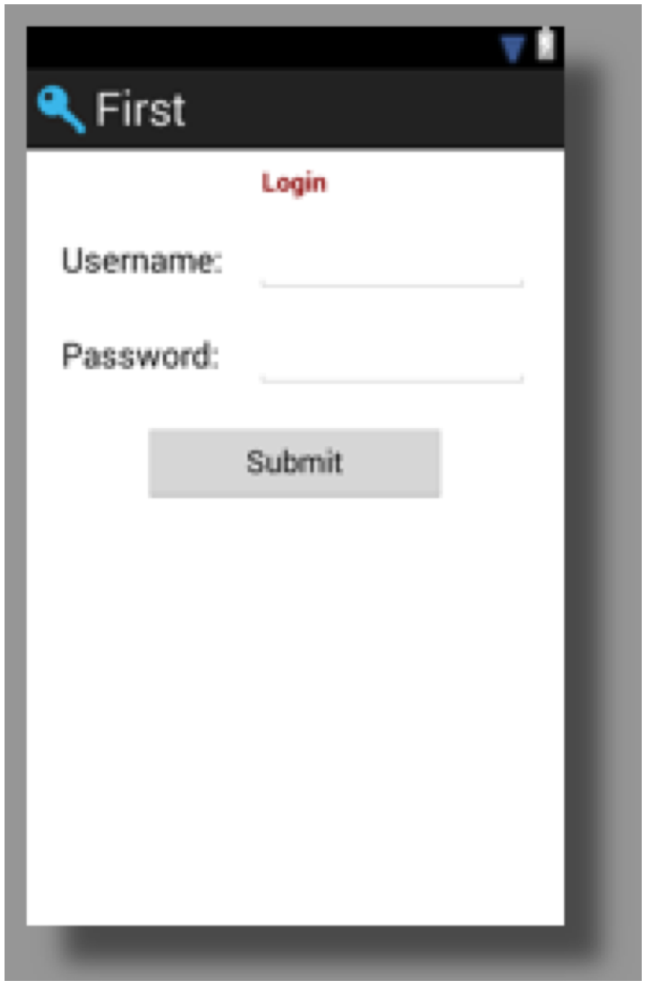
Example

```
<FrameLayout xmlns:android= …
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <ImageView
        android:src="@drawable/ic_launcher"
        android:contentDescription="@string/test"
        android:scaleType="fitCenter"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"/>

    <TextView
        android:text="@string/test"
        android:textSize="30sp"
        android:textStyle="bold"
        android:textColor="#003399"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:gravity="center"/>
</FrameLayout>
```

TableLayout

```
<TableLayout xmlns:...
     <TableRow android:paddingTop="10dp"
                    android:gravity="center">
     <TextView ...
          android:layout_gravity="center"
          android:layout_span="2"/>
     </TableRow>

     <TableRow android:layout_marginTop="20dp" >
     <TextView ... />
         <EditText ... />
     </TableRow>

     <TableRow android:layout_marginTop="20dp" >
     <TextView ... />
         <EditText ... />
     </TableRow>

     <TableRow android:gravity="center”
             android:layout_marginTop="20dip" >
     <Button ...
          android:layout_span="2"/>
     </TableRow>
</TableLayout>
```





WebView

- Helps deliver a web page as part of a client application
- You
  - Must enable JavaScript if needed
  - Can create interfaces between JavaScript code and Android code
  - Can control navigation


```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("http://www.example.com");
```

Lists and grids

- When the content for a Layout is dynamic or not pre-determined, we need a layout that subclasses AdapterView to populate the layout with Views at runtime
- Items are automatically inserted into the Layout by an Adapter
  - It pulls content from a source such as an array or database query
  - Converts each item result into a View that is placed into the Layout
- ListView displays a list of scrollable items
- GridView displays items in a two-dimensional, scrollable grid

Adapter

- Used to visualize data
  - Acts as a bridge between an AdapterView and the underlying data for that view
    - ListView, GridView, Spinner are subclasses of AdapterView
  - Provides access to the data items
- Makes a ViewGroup to interact with data
  - Also responsible for making a View for each item in the data set
- Some methods
  - isEmpty(), getItem(), getCount(), getView()

A couple of Adapters

- ArrayAdapter should be used when the data source is an array
  - It creates a view for each array item by calling toString() on each item and placing the contents in a TextView
- SimpleCursorAdapter should be used when data come from a Cursor
  - We must specify a layout for each row in the Cursor and specify the columns that should be inserted into each view of the layout
- onItemClickListener should be used to respond to click events on each item in an AdapterView

ListView example

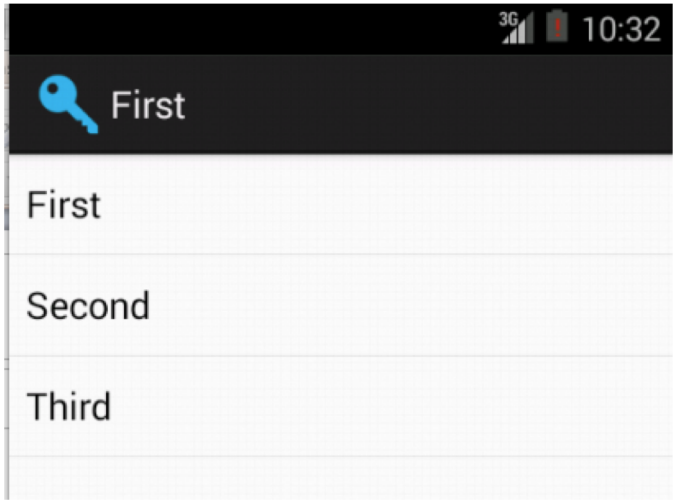
```
<ListView xmlns:android= …
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:id="@+id/list" />
```

onCreate()

```
public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);

     String[] data = {"First", "Second", "Third"};
     ListView lv = (ListView) findViewById(R.id.list);
     lv.setAdapter(new ArrayAdapter<String>(this,
               android.R.layout.simple_list_item_1, data));
}
```

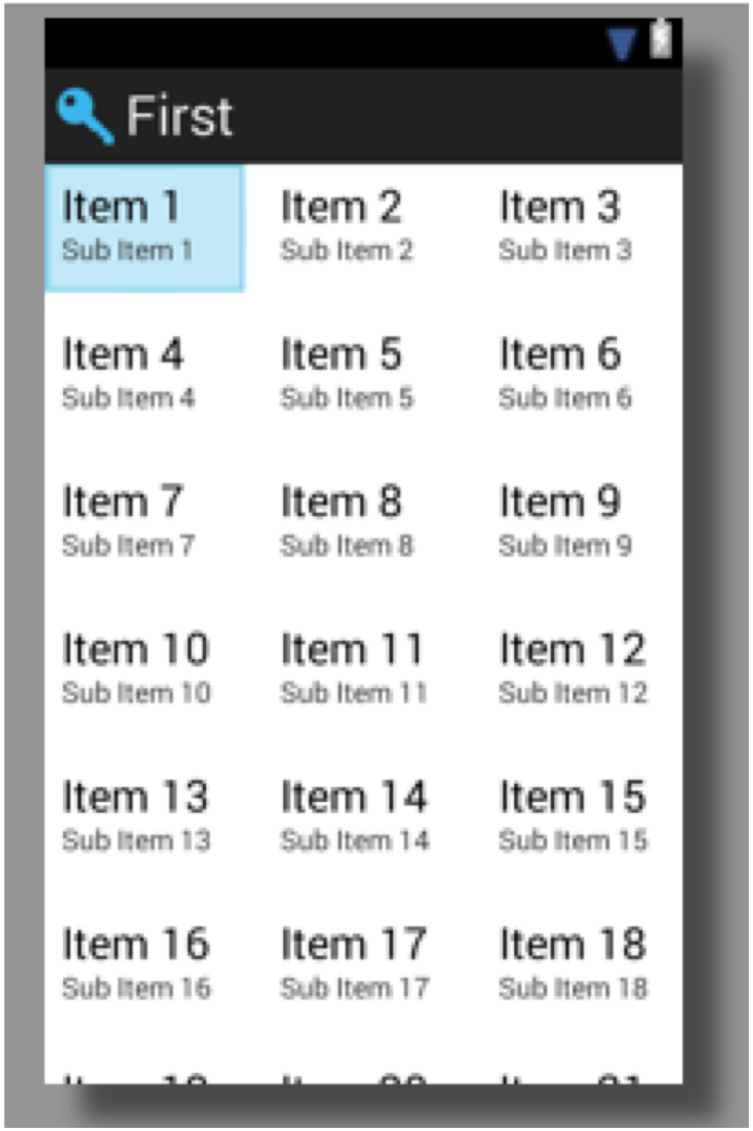
simple_list_item_1

```
<TextView xmlns:android= …
     android:id="@android:id/text1"
     style="?android:attr/listItemFirstLineStyle"
     android:paddingTop="2dip"
     android:paddingBottom="3dip"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content" />
```



GridView example

```
<GridView xmlns:android= …
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```



onCreate()

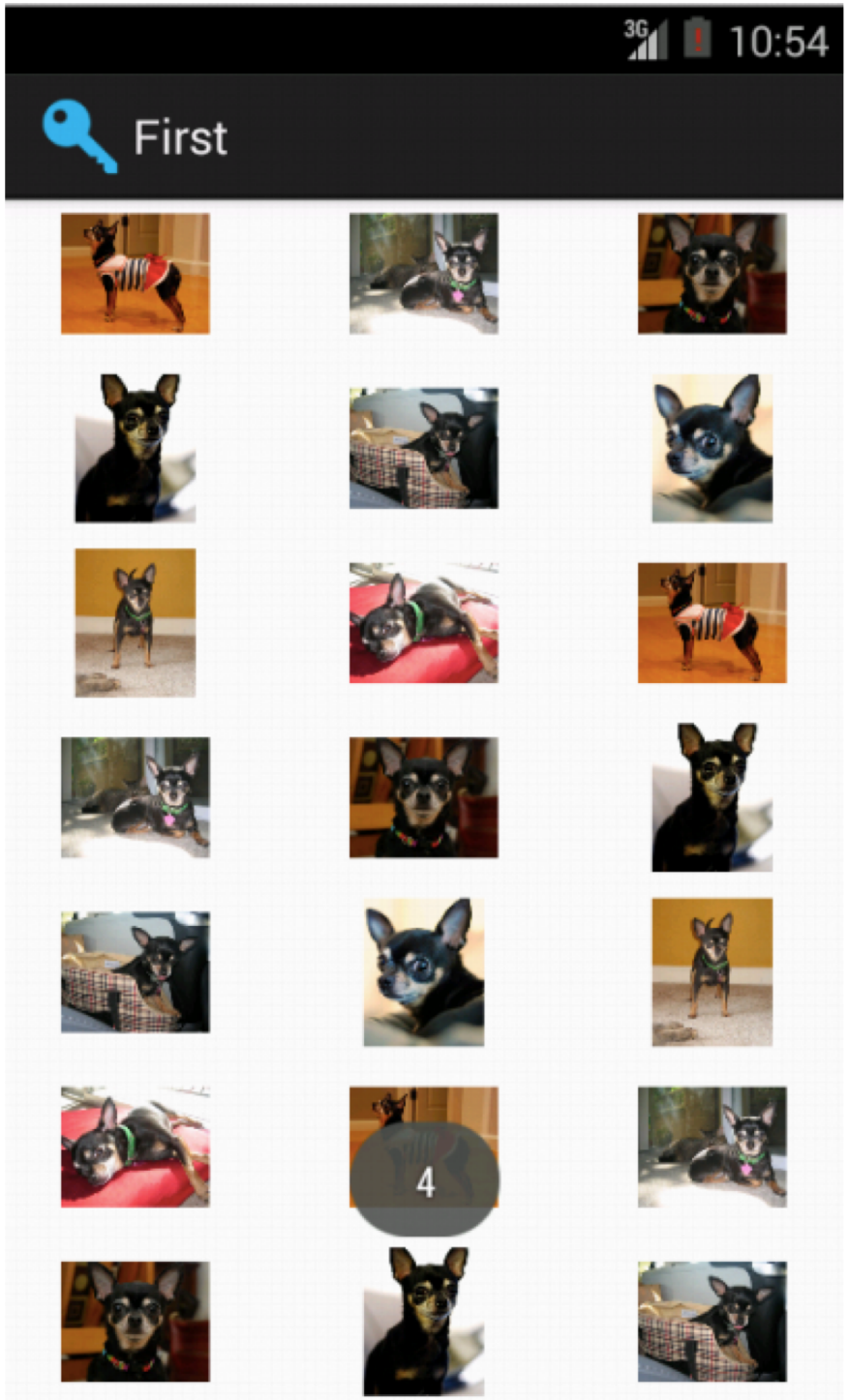
```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        GridView gridview = (GridView) findViewById(R.id.gridview);
        gridview.setAdapter(new ImageAdapter(this));

        gridview.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent,
                                    View v, int position, long id) {
                    Toast.makeText(MainActivity.this, "" + position,
                                    Toast.LENGTH_SHORT).show();
                }
        });
    }
```

ImageAdapter (I)

```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;

 // references to our images
    private Integer[] mThumbIds = {
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7,
            R.drawable.sample_0, R.drawable.sample_1,
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7,
            R.drawable.sample_0, R.drawable.sample_1,
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7
    };
…
```

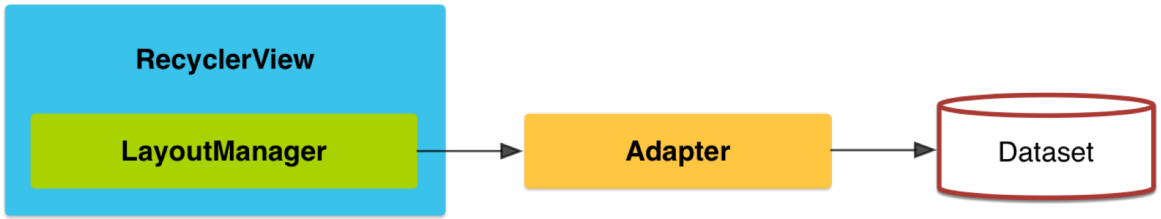
ImageAdapter (II)

```
// create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if (convertView == null) {  // if it's not recycled, initialize some
                                    // attributes
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }
}
```


10:54
First
4

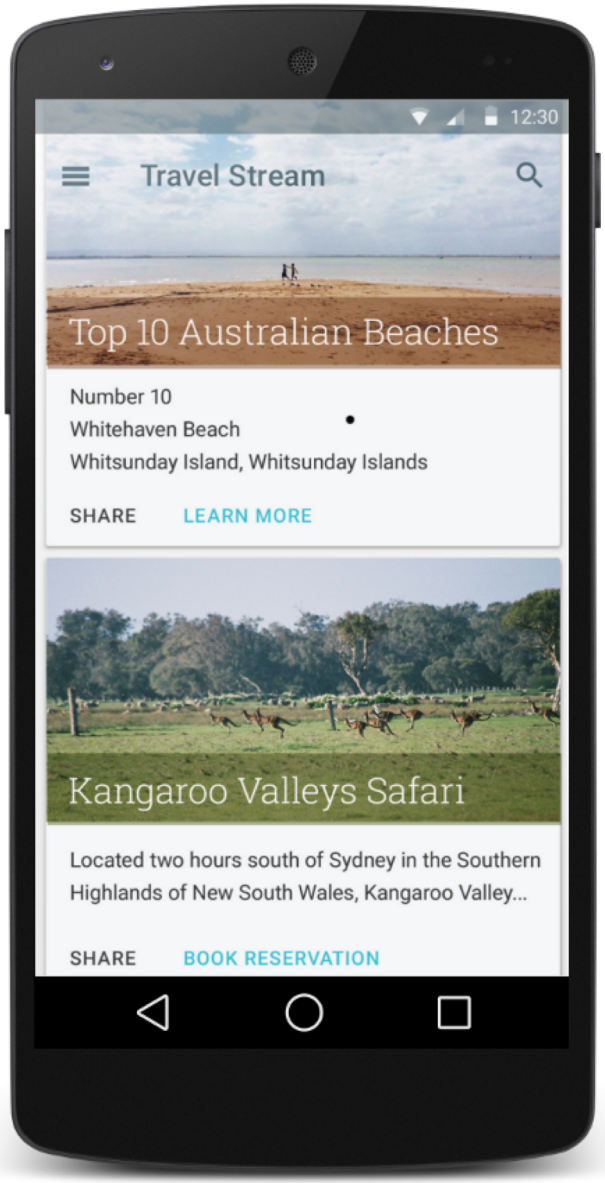
RecyclerView (flexible version of ListView)

- It is a container for displaying large data sets that can be scrolled very efficiently
  - It simplifies the display and handling of large data sets by providing
    - Layout managers for positioning items
    - Default animations for common item operations, such as removal or addition of items
- A layout manager positions item views inside a RecyclerView and determines when to reuse item views that are no longer visible
  - Recycling views improve performance by avoiding the creation of unnecessary views or performing expensive lookups



CardView (extends FrameLayout)

- Shows information inside cards that have a consistent look across the platform
- A RecyclerView comes with an adapter to inflate a layout for organizing the cards
  - Cards need a dedicated layout
- RecyclerView and CardView widgets are part of the v7 Support Libraries



Styles and themes

- A style is a collection of properties that specify the look and format of a View or window
  - Styles share a similar philosophy to cascading stylesheets
- A theme is a style applied to an entire Activity or application
  - Every View in the Activity or application will apply each style property it supports
- Android provides a large collection of styles and themes

Example

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

ProjectName/res/values/styles.xml

```
<resources>
    <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

This example style can be referenced from an XML layout as @style/CodeFont

Views and events

- The user interacts with the Views and generates events
  - Events for clicks, long clicks, gestures, focus, external events
- Android manages the creation and distribution of these events, but not the reactions to them
  - We must implement them by hand

Two elements

- Event Handlers
  - Views have callback methods to handle specific events
    - E.g., when a Button is touched, method onTouchEvent() is called
  - Special-purpose reactions are obtained by extending the particular View class and by overriding the method
    - Suitable for custom elements
- Event Listeners
  - Are Interfaces that contain a single callback method
  - This method is called by the Android framework when the event is fired on the particular View

Some Listeners

- OnClickListener: onClick()
- OnLongClickListener: onLongClick()
- OnFocusChangeListener: onFocusChange()
- OnKeyListener: onKey()
- OnCheckedChangeListener: onCheckedChanged()
- OnTouchListener: onTouch()
- OnCreateContextMenuListener: onCreateContextMenu()

Option 1

- Implement the callback method
- Define the listener object as an anonymous class
- Pass an instance of the ActionListener implementation to the View through method setOnXXXListener

```
Button btn = (Button) findViewById(R.id.button);
btn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        // Implementation
    }
});
```

Option 2

- Implement the callback method
- Implement the interface in the Activity
- Pass an instance of the listener to the View through method setOnXXXListener

```
public class MyActivity extends Activity implements OnClickListener {

    …
    Button btn = (Button) findViewById(R.id.button);
    btn.setOnClickListener(this);
    …

    public void onClick(View v) {
        // Implementation
    }
}
```

Option 3

- Use onClick in the XML definition of the View (if possible)
- Implement the method in the activity

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

```
public void myOnClick(View view) {
        // Implementation
}
```

How to fire events in the code

- Through perform”something” methods
  - The corresponding listener (if set) is activated
  - performClick() activates the view's OnClickListener, if defined
- Used to produce events
  - As consequences of other actions
  - To test listeners

Composing layouts

- Create a new XML file and define the layout
  - Example, a title bar that should be included in each activity
- Add the <include/> tag inside the new layout
- The <merge/> tag helps eliminate redundant view groups in your view hierarchy
  - When you include this layout in another layout, the system ignores the <merge> element and places the elements directly in the layout

<include/>

```
<FrameLayout …
   android:layout_width="match_parent”
   android:background="@color/titlebar_bg">

   <ImageView android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:src="@drawable/gafricalogo" />
</FrameLayout>
```

```
<LinearLayout …
   android:layout_width="match_parent”
   android:layout_height="match_parent”
   android:orientation="vertical”
   android:background="@color/app_bg”
   android:gravity="center_horizontal">

   <include layout="@layout/titlebar"/>
</LinearLayout>
```

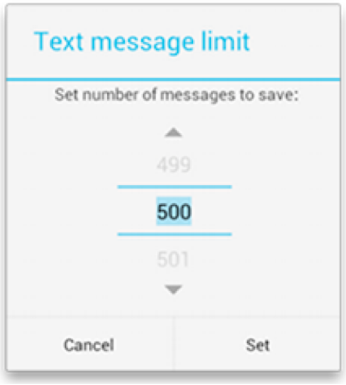
Toasts

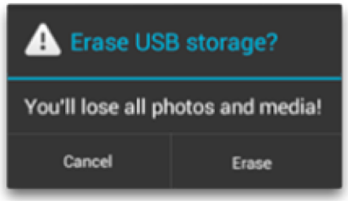
- Tiny messages over the Activity
  - Used to highlight minor problems or provide confirmations
  - Usually centered horizontally (gravity)
  - We can control their duration

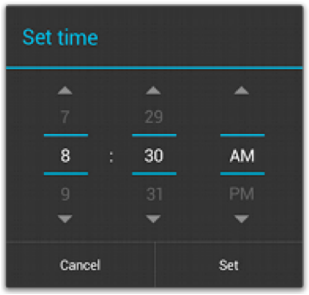
```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

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

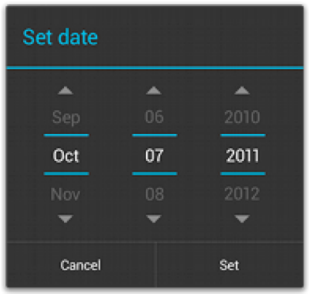
Dialogs

- Used to interact with the user
- Short messages, easy answers
  - AlertDialog
  - ProgressDialog
    - ProgressBar would be better
  - DatePickerDialog
  - TimePickerDialog



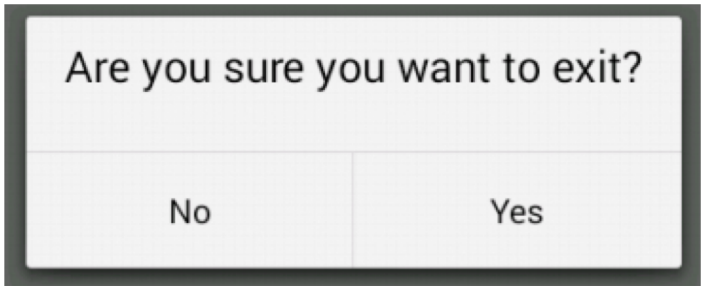






AlertDialog

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?").setCancelable(false);

builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        MainActivity.this.finish();
    }
});

builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        dialog.cancel();
    }
});
AlertDialog alert = builder.create();
alert.show();
```



Animations

- Used to move/shrink/color elements
- Two solutions
  - Subsequent images (frame-by-frame)
  - Initial state, final state, time, transition (tween)
- Android 5.0 (API level 21) includes new APIs to create custom animations in your app
  - Animations are heavy and expensive
  - Too many animations are confusing and may become a problem