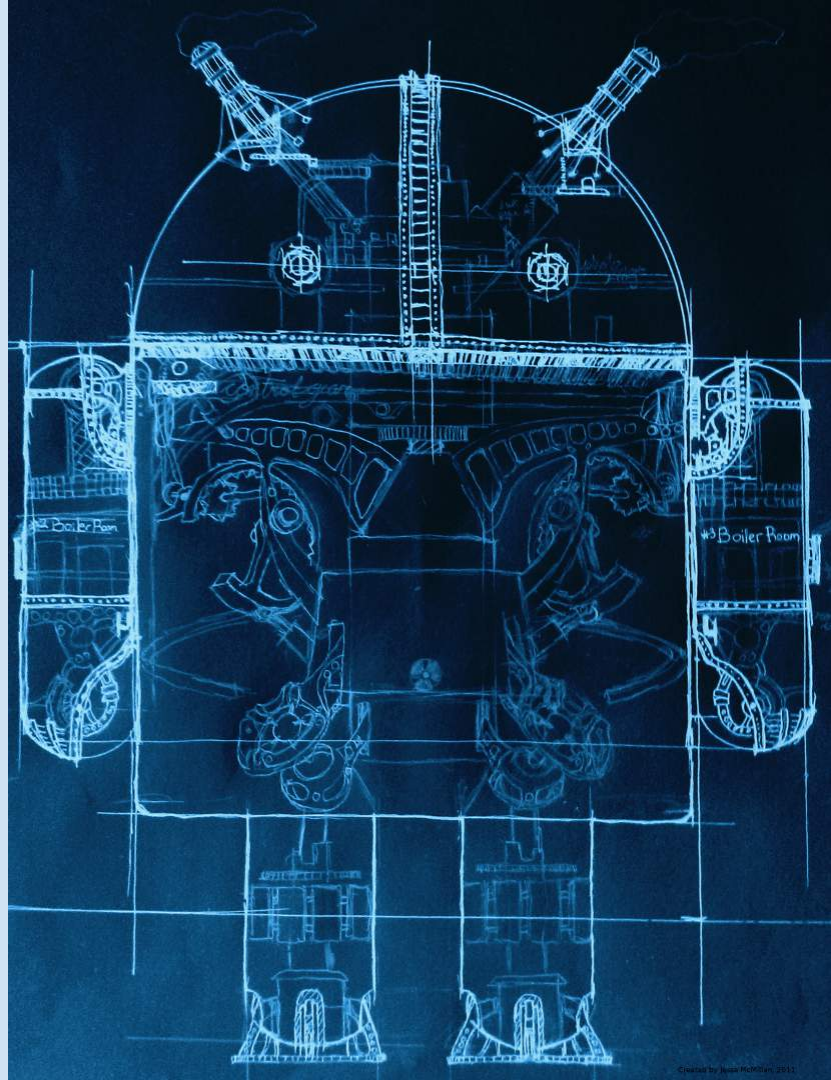



# Android Layouts



- APPS
- GAMES
- MOVIES & TV
- MUSIC
- BOOKS
- NEWSSTAND


New + Updated Games

MORE




FIFA 15  
Ultimate

★★★★★ FREE



Just Dance  
Now

★★★★★ FREE



The Battle  
Cats

★★★★★ FREE

Popular Apps + Games

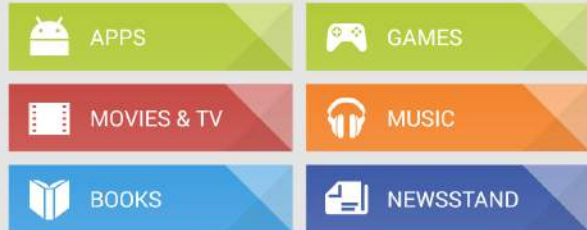
Discover this week's popular titles

MORE









New + Updated Games

MORE



FIFA 15  
Ultimate  
★★★★★ FREE



Just Dance  
Now  
★★★★★ FREE



The Battle  
Cats  
★★★★★ FREE

Popular Apps + Games

Discover this week's popular titles

MORE



1 Sports

# Night to remember: Jeter caps his Yankee Stadium finale in classic style

Yahoo Sports, Associated Press + 1 more



2 US News

# Ferguson unrest: Riots flair after police chief apologizes, marches with protesters


Associated Press, Yahoo News + 4 more






- APPS
- GAMES
- MOVIES & TV
- MUSIC
- BOOKS
- NEWSSTAND

New + Updated Games




FIFA 15 Ultimate

★★★★★ FREE



Just Dance Now

★★★★★ FREE



The Battle Cats

★★★★★ FREE

Popular Apps + Games









1 Sports

Night to remember: Jeter caps his Yankee Stadium finale in classic style

Yahoo Sports, Associated Press + 1 more



2 US News

Ferguson unrest: Riots flair after police chief apologizes, marches with protesters

Associated Press, Yahoo News + 4 more



Monday 11 am

3 km/h

77°

0.0 mm

83°



# The View Class

Layout components all inherit from the `View` class.

Examples:

- `ImageView` **extends** `View`.
- `TextView` **extends** `View`.
- `EditText` **extends** `TextView`.
- `Button` **extends** `TextView`.

Views have attributes that control positioning and appearance.

# The ViewGroup Class

View groups can contain other views (“children”).

Examples:

- `LinearLayout` - aligns children horizontally or vertically.

# The ViewGroup Class

View groups can contain other views (“children”).

Examples:

- `LinearLayout` - aligns children horizontally or vertically.
- `RelativeLayout` - positions children relative to siblings/parent.

# The ViewGroup Class

View groups can contain other views (“children”).

Examples:

- `LinearLayout` - aligns children horizontally or vertically.
- `RelativeLayout` - positions children relative to siblings/parent.
- `ListView` - for long lists of similar items.



# The ViewGroup Class

View groups can contain other views (“children”).

Examples:

- `LinearLayout` - aligns children horizontally or vertically.
- `RelativeLayout` - positions children relative to siblings/parent.
- `ListView` - for long lists of similar items.
- `GridView` - for grids of similar items.

# The ViewGroup Class

View groups can contain other views (“children”).

Examples:

- `LinearLayout` - aligns children horizontally or vertically.
- `RelativeLayout` - positions children relative to siblings/parent.
- `ListView` - for long lists of similar items.
- `GridView` - for grids of similar items.
- `ScrollView` - **wrapper** for long content.

# The ViewGroup Class

View groups can contain other views (“children”).

Examples:

- `LinearLayout` - aligns children horizontally or vertically.
- `RelativeLayout` - positions children relative to siblings/parent.
- `ListView` - for long lists of similar items.
- `GridView` - for grids of similar items.
- `ScrollView` - wrapper for long content.
- `FrameLayout` - dumb container for e.g. Fragments.

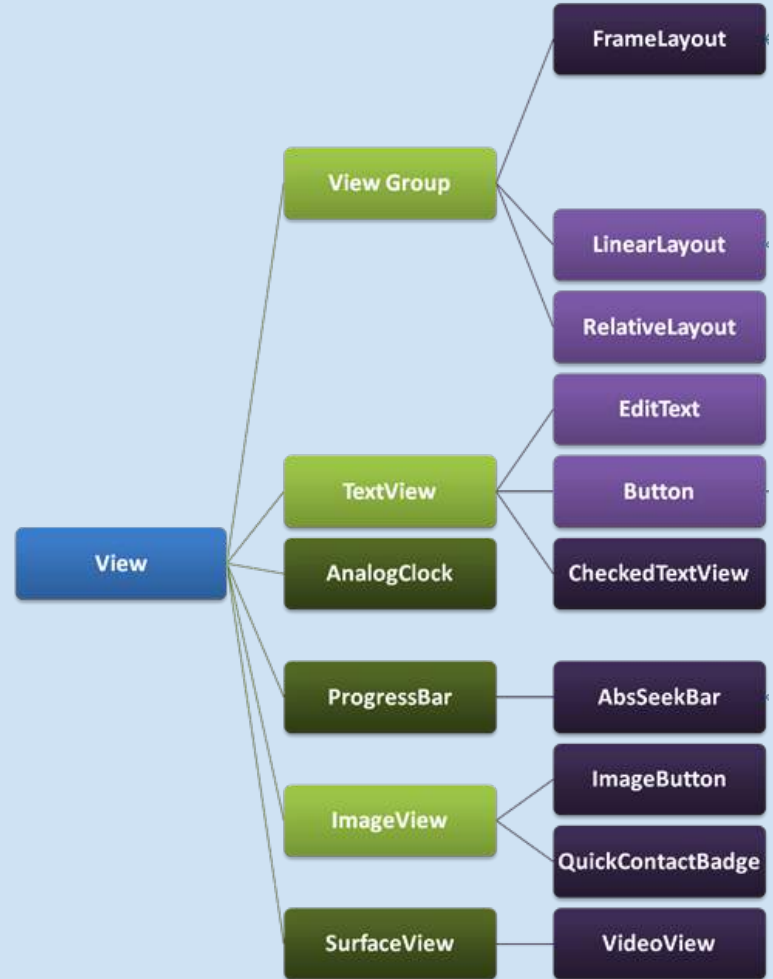
# View Classes cont'd

Views inherit attributes from their parent(s).

TextView has a text attribute

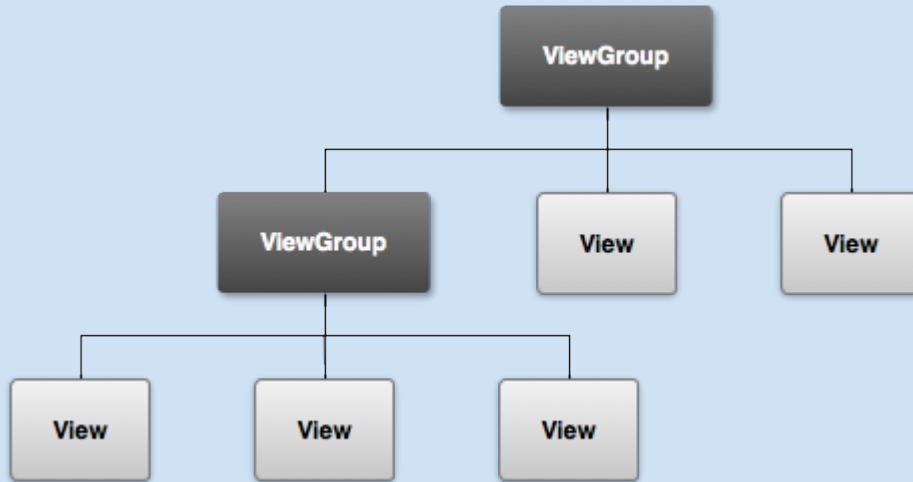


Button has a text attribute,  
EditText has a text attribute,  
etc.



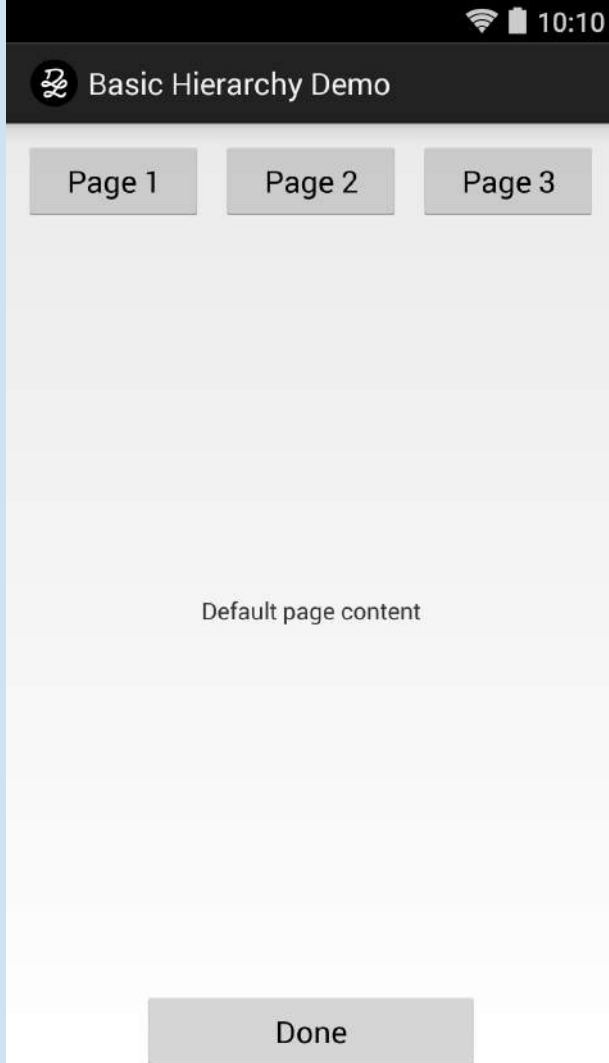
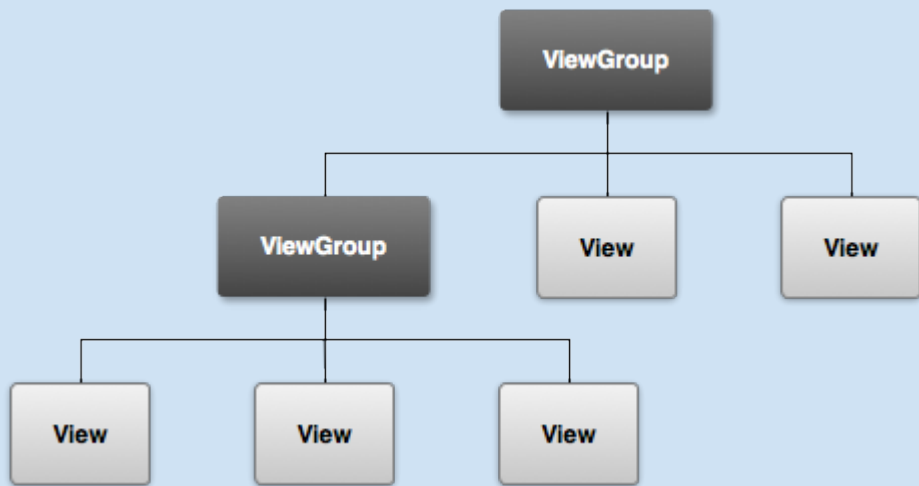
# The 'View Hierarchy'

Describes the structure of a layout.



# The 'View Hierarchy'

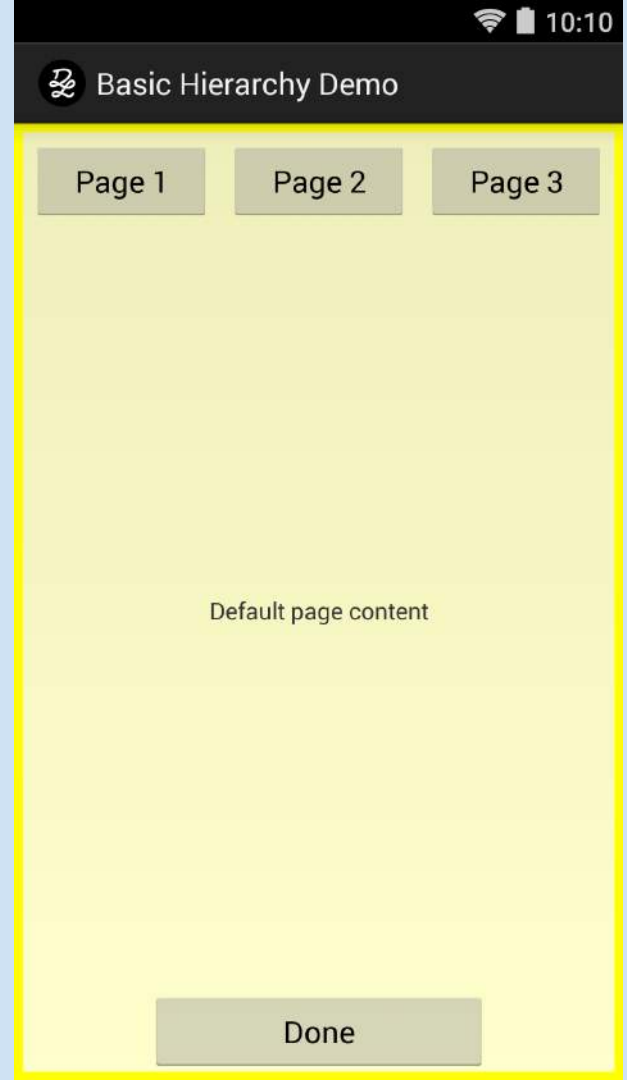
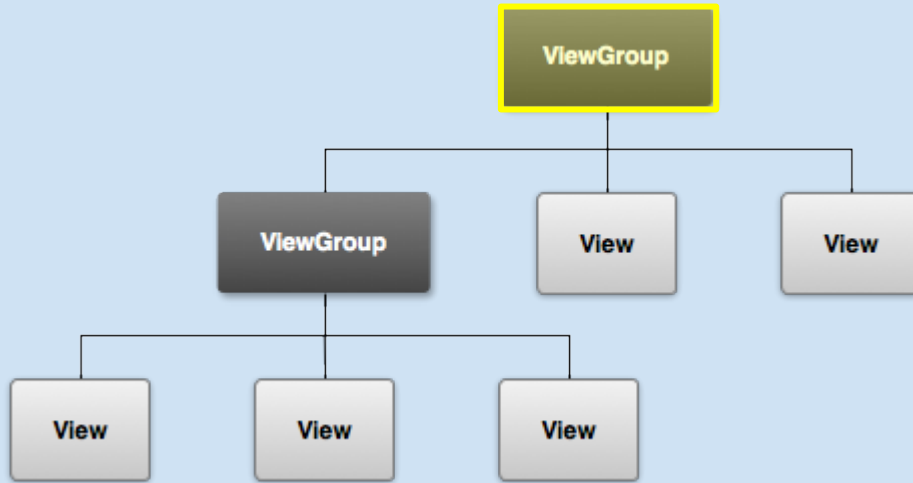
Describes the structure of a layout.





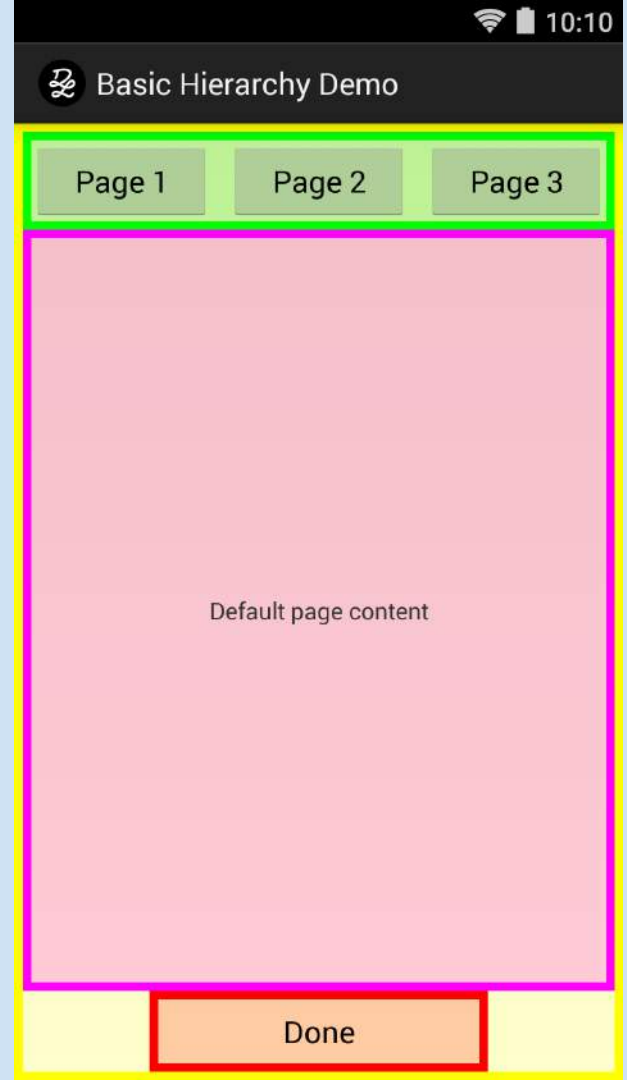
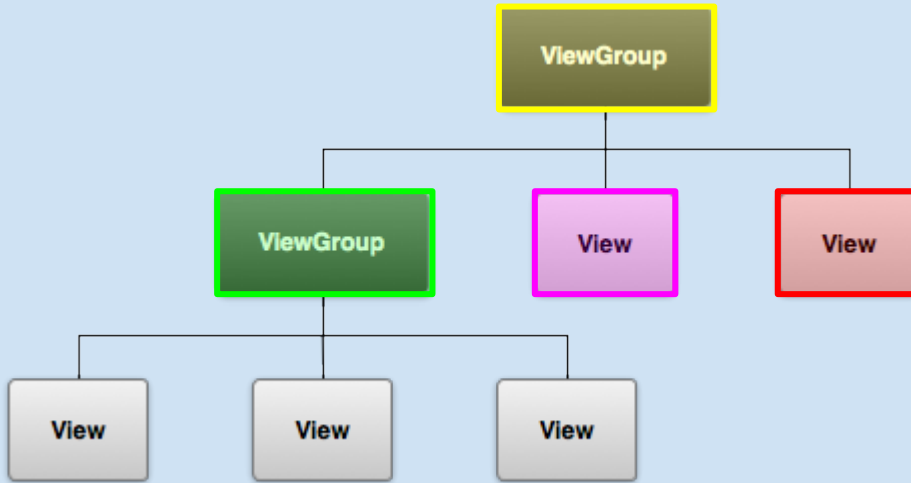
# The 'View Hierarchy'

Describes the structure of a layout.



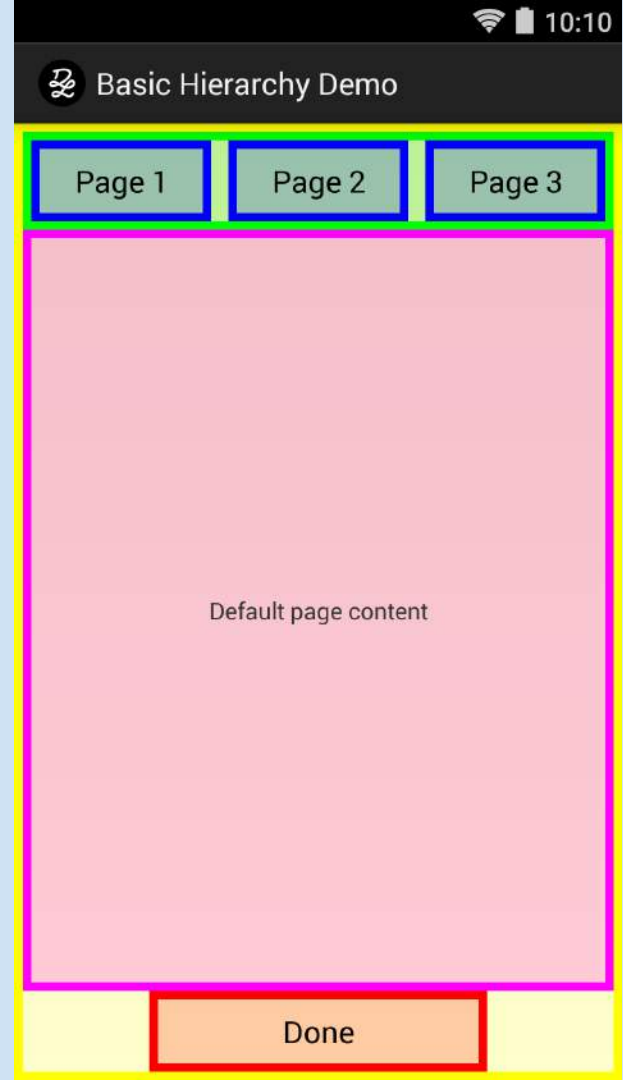
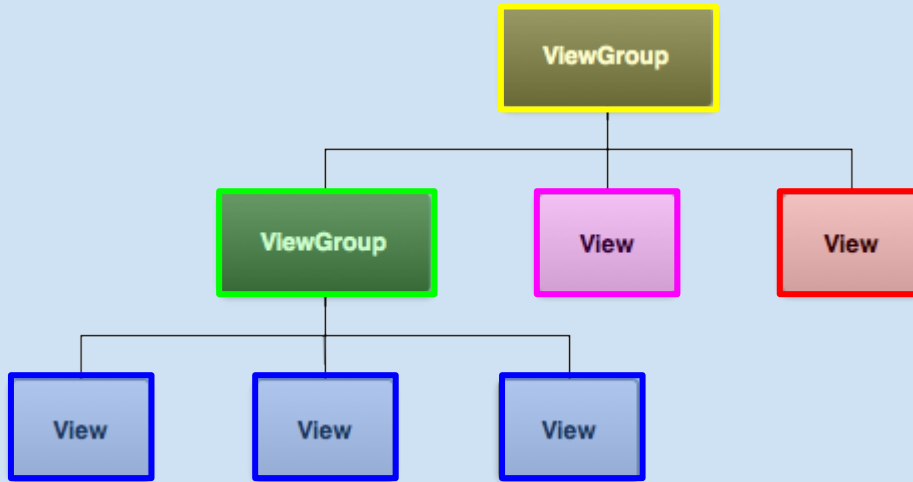
# The 'View Hierarchy'

Describes the structure of a layout.

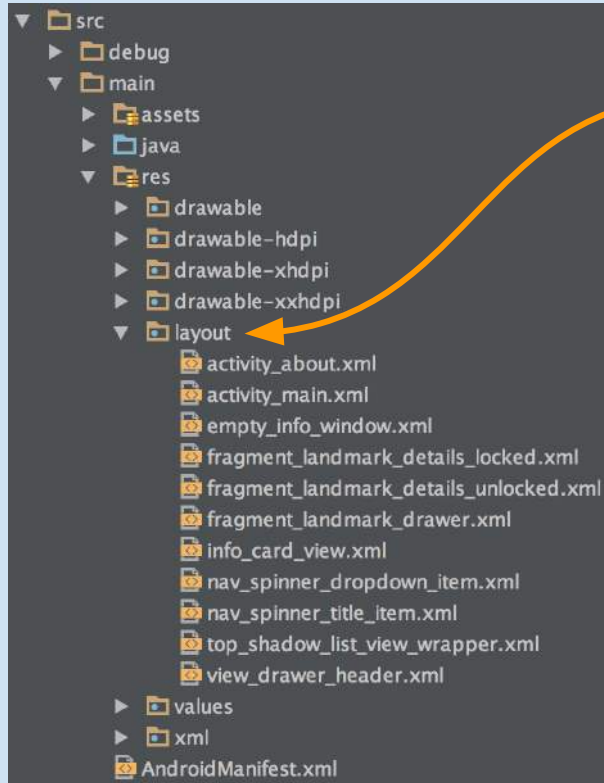


# The 'View Hierarchy'

Describes the structure of a layout.



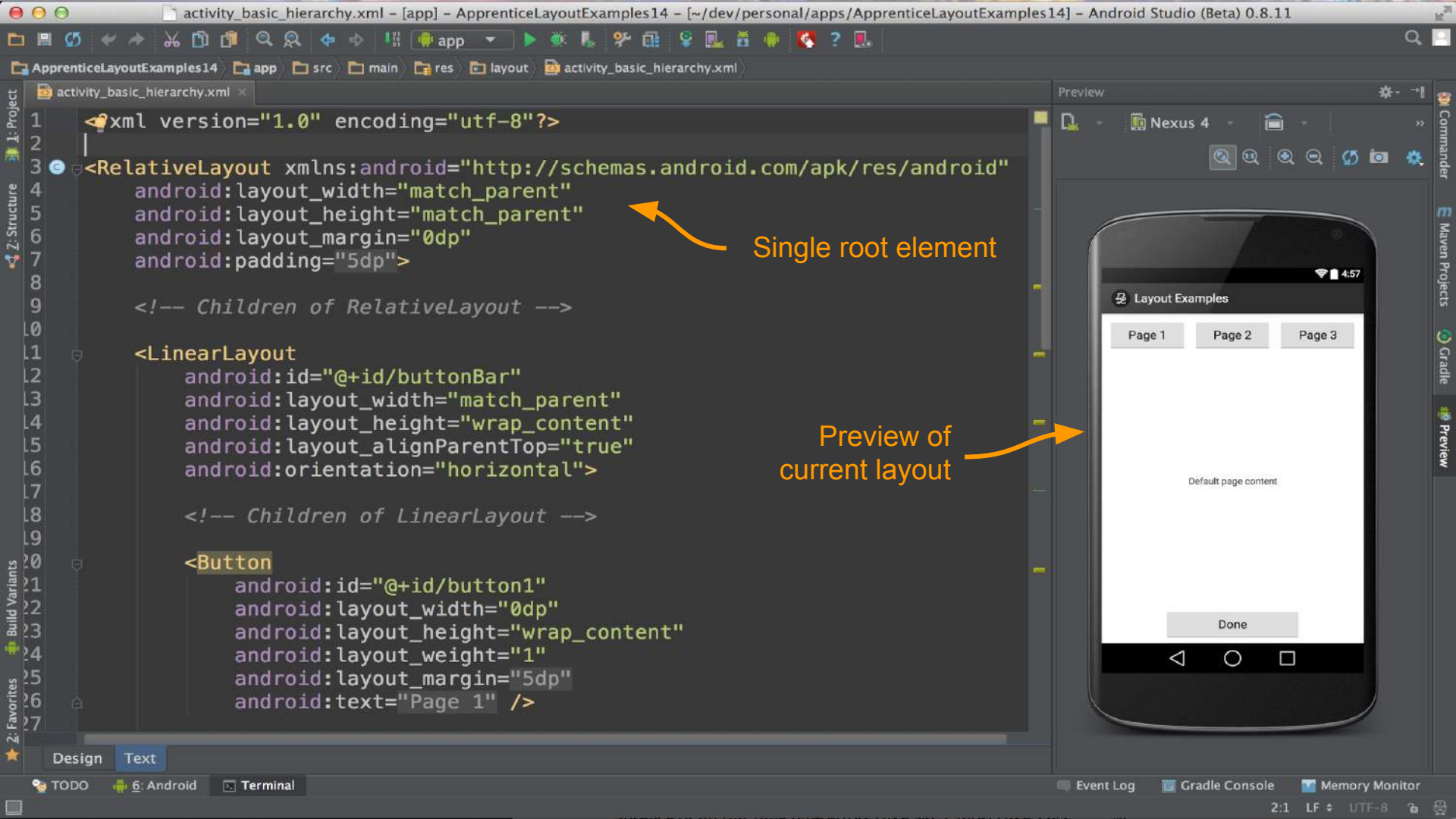
# Describing layouts in xml



All layout xml files live in the 'layout' resource directory.

Separates UI from app logic.

xml layout files can be reused (e.g. list row).



Single root element

Preview of current layout

# Why write xml?

Helps learn available attributes more quickly.

Know attribute names → can guess method names.

Stronger understanding of view hierarchy.



# Describing a View in xml

```
<View  
    android:layout_width="wrap_content"  
    android:layout_height="match_content" />
```

Set view attributes using the key=value format.

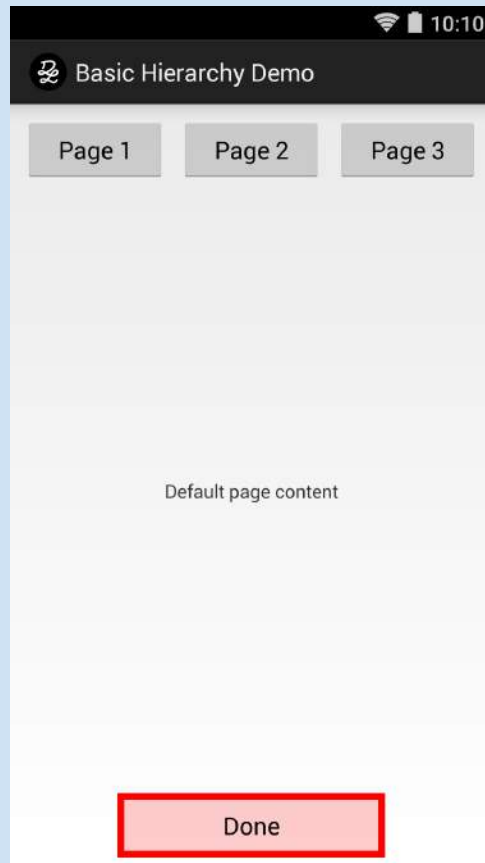
layout\_width and layout\_height are *required* attributes.

/> is a self-closing tag (no children).

# Example: View in xml

```
<Button
    android:id="@+id/bottomButton"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:padding="@dimen/default_padding"
    android:text="@string/done"
    android:background="@drawable/red_highlight" />
```

@ symbol refers to resources (ids, drawables, dimensions, colors, strings, etc.)



# Describing a ViewGroup in xml

```
<ViewGroup
    android:layout_width="match_content"
    android:layout_height="match_content">

    <!-- ViewGroup children described here -->

</ViewGroup>
```

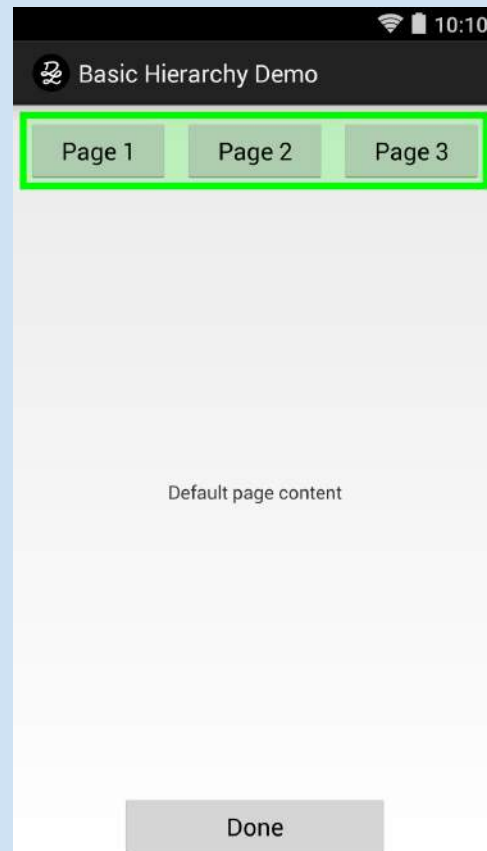
Use a separate closing tag `</ . . . >` to include children.

# Example: ViewGroup in xml

```
<LinearLayout
    android:id="@+id/buttonBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:orientation="horizontal"
    android:background="@drawable/green_highlight">

    <!-- Buttons described here -->

</LinearLayout>
```



# View attributes: overview

Common `View` attributes:

- `layout_width` **and** `layout_height` (required)
- `id`
- `padding`
- `layout_margin`
- `gravity`
- `layout_gravity`

Other useful `View` attributes: `background`, `visibility`

# View attributes: layout\_width

```
android:layout_width="match_parent"  
android:layout_width="wrap_content"  
android:layout_width="150dp"  
android:layout_width="@dimen/my_custom_width_dimension"
```

Defines width of `View`. `layout_height` is analogous.

Can be relative (to parent) or absolute.

dp = density independent pixels. Use them (except for text)!



# View attributes: id

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

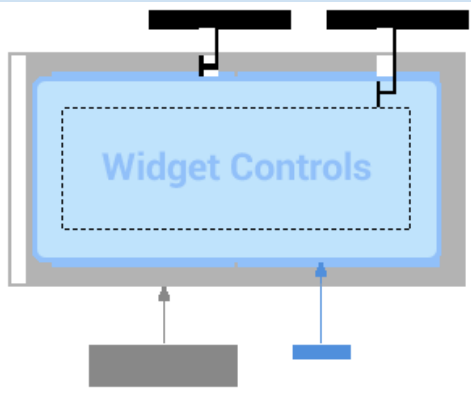
Needed to refer to a view (in layout or application code).

Unique within an xml layout file.

In xml: use `@+id` to define a new id; use `@id` to refer to it.

# View attributes: padding

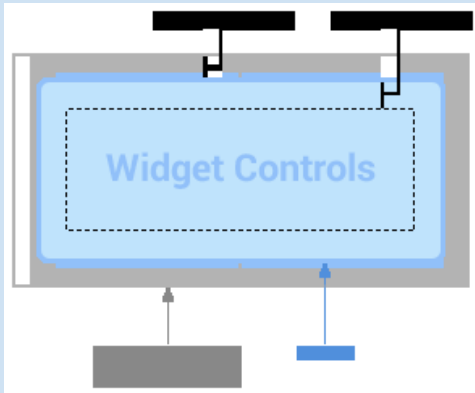
```
android:padding="15dp"  
android:paddingTop="5dp"  
android:paddingBottom="10dp"  
android:paddingLeft="@dimen/dimen_defined_in_resources"  
android:paddingRight="@dimen/some_other_dimen"
```



Control distance between view content and edges.

# View attributes: `layout_margin`

```
android:layout_margin="15dp"  
android:layout_marginTop="5dp"  
android:layout_marginBottom="10dp"  
android:layout_marginLeft="@dimen/dimen_defined_in_resources"  
android:layout_marginRight="@dimen/some_other_dimen"
```



Control distance from any other view.

- Background colors padding.
- Background does not color margins.

# View attributes: gravity

```
android:gravity="top"  
android:gravity="left"  
android:gravity="top|left"  
android:gravity="center"  
android:gravity="center_horizontal"
```

Position content **within a view**.

Content examples:

- text inside a `TextView`
- child views inside a view group

# View attributes: layout\_gravity

```
android:layout_gravity="top"  
android:layout_gravity="left"  
android:layout_gravity="top|left"  
android:layout_gravity="center"  
android:layout_gravity="center_horizontal"
```

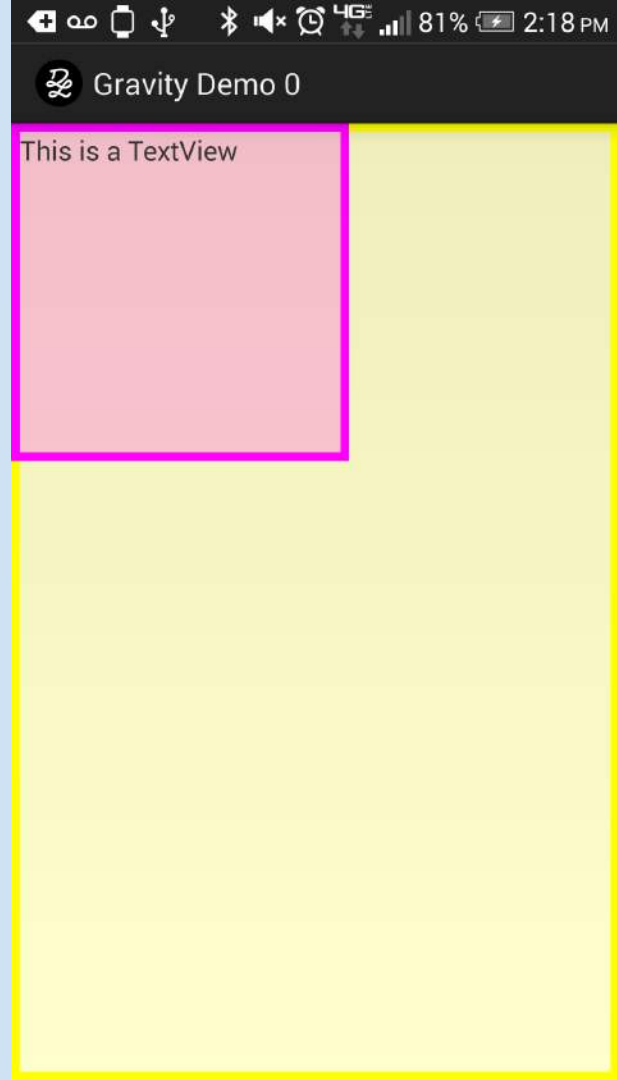
Position a child view [within its parent view group](#).

# Gravity example 0

`TextView` has fixed width and height.

No `gravity` set.

No `layout_gravity` set.



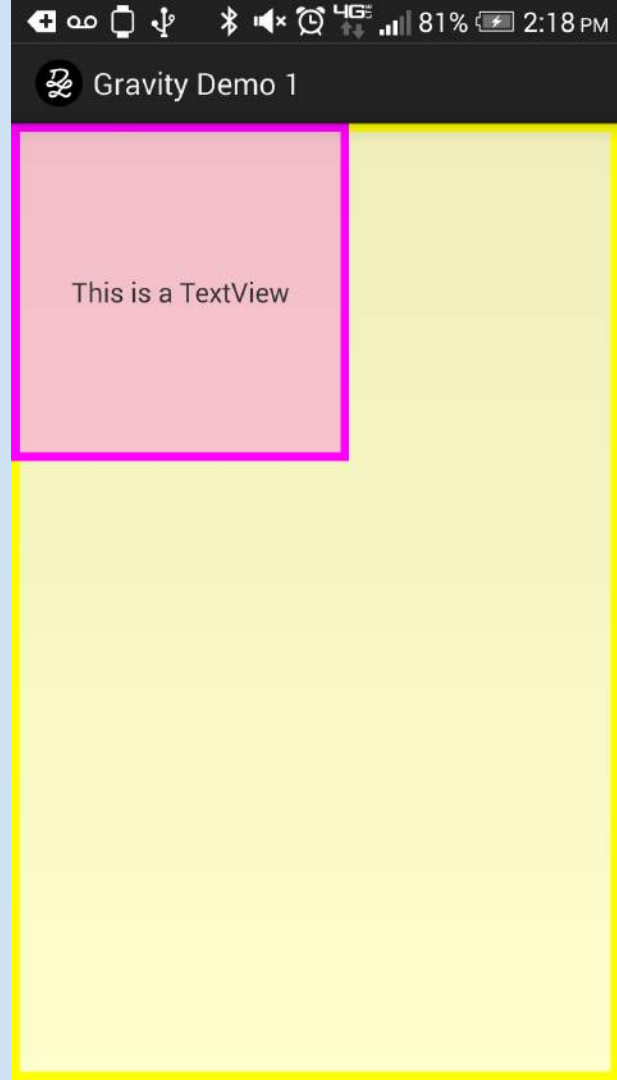


# Gravity example 1

`TextView` has fixed width and height.

**gravity=center.**

No `layout_gravity` set.

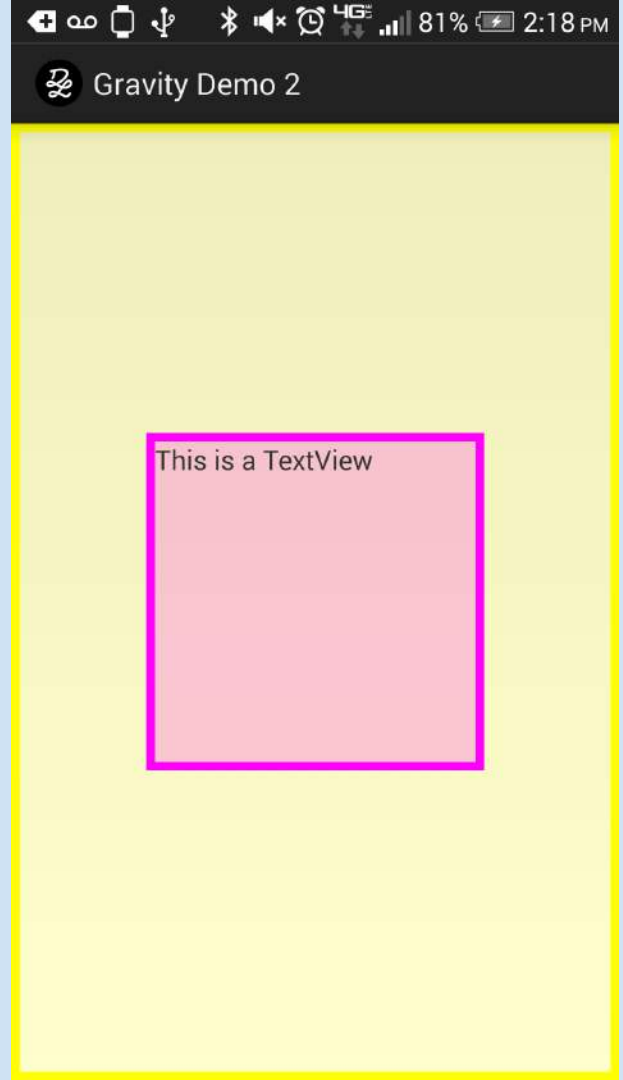


# Gravity example 2

`TextView` has fixed width and height.

No gravity set.

`layout_gravity=center`.

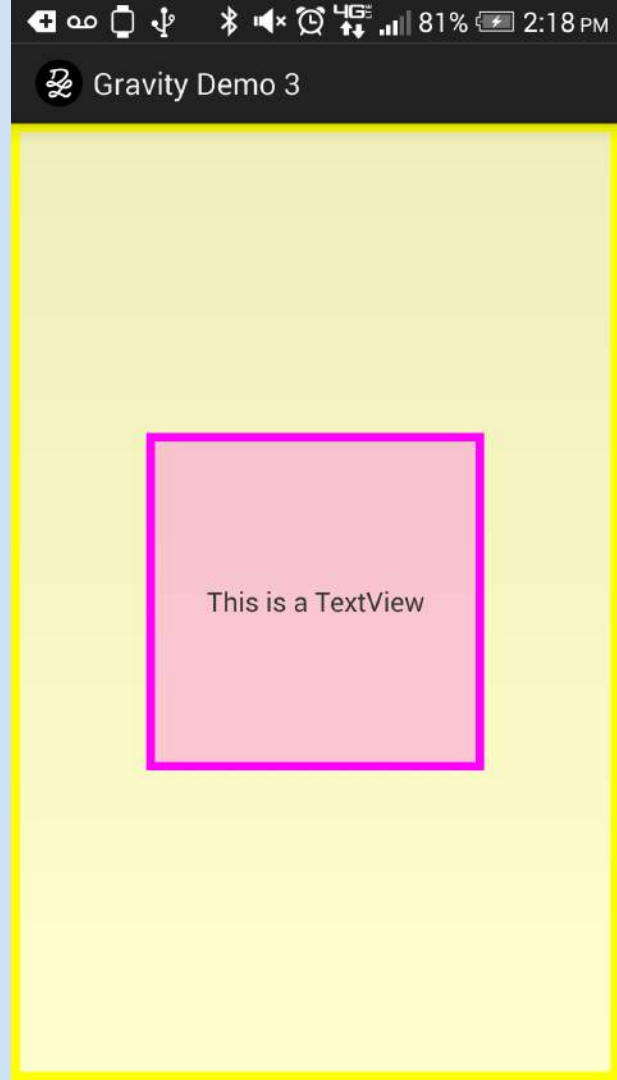


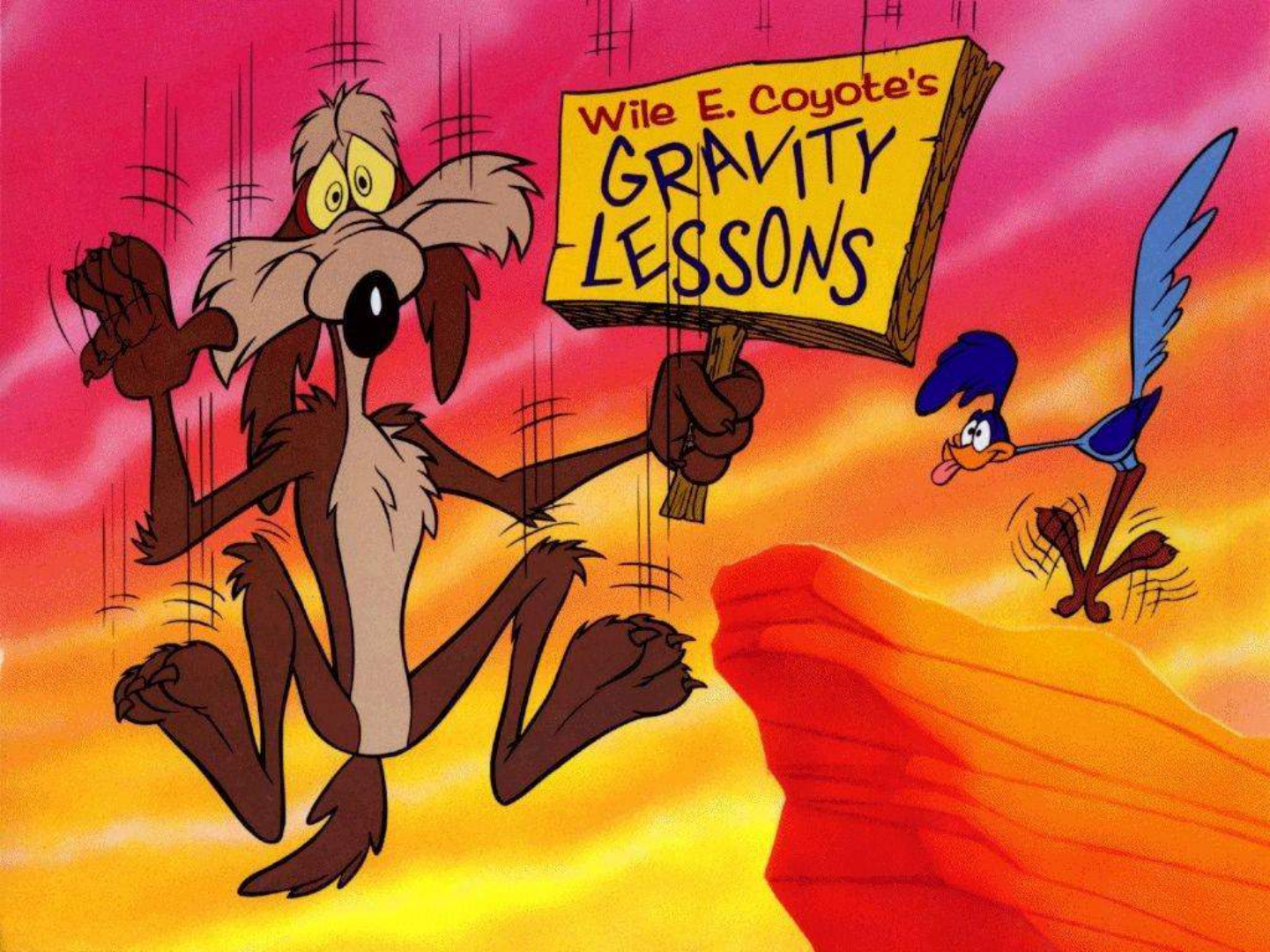
# Gravity example 3

`TextView` has fixed width and height.

`gravity=center.`

`layout_gravity=center.`





# gravity vs layout\_gravity

Use gravity for:

- positioning content inside a non-ViewGroup (e.g. text inside a TextView).
- applying the same gravity to all children of a ViewGroup.

Use layout\_gravity for:

- positioning children of a ViewGroup independently.



# gravity vs layout\_gravity

Use `gravity` for:

- positioning content inside a non-`ViewGroup` (e.g. text inside a `TextView`).
- applying the same gravity to all children of a `ViewGroup`.

Use `layout_gravity` for:

- positioning children of a `ViewGroup` independently.



# Common views

View	Optional Attributes
TextView	<code>text</code> , <code>textColor</code> , <code>textSize</code> (in sp, not dp), <code>textStyle</code> , <code>drawableLeft</code> , <code>drawableRight</code>
EditText	<code>hint</code> (and TextView attributes)
Button	<code>enabled</code> (and TextView attributes)
ImageView	<code>src</code> , <code>scaleType</code>

# Describing a ViewGroup in xml

Same structure as for regular views:

- `layout_width` is required.
- `layout_height` is required.
- Optional attributes vary by view group.



# Positioning in a `LinearLayout`

`LinearLayout` attributes:

```
android:orientation="horizontal"  
android:orientation="vertical"  
android:weightsum="1"
```

Children are positioned sequentially (left to right or top to bottom).

Children specify their 'weight' within the `LinearLayout`.

$\text{weight} / \text{weightSum} = \text{child size as proportion of parent size.}$

weightSum = 2

LinearLayout Demo 0

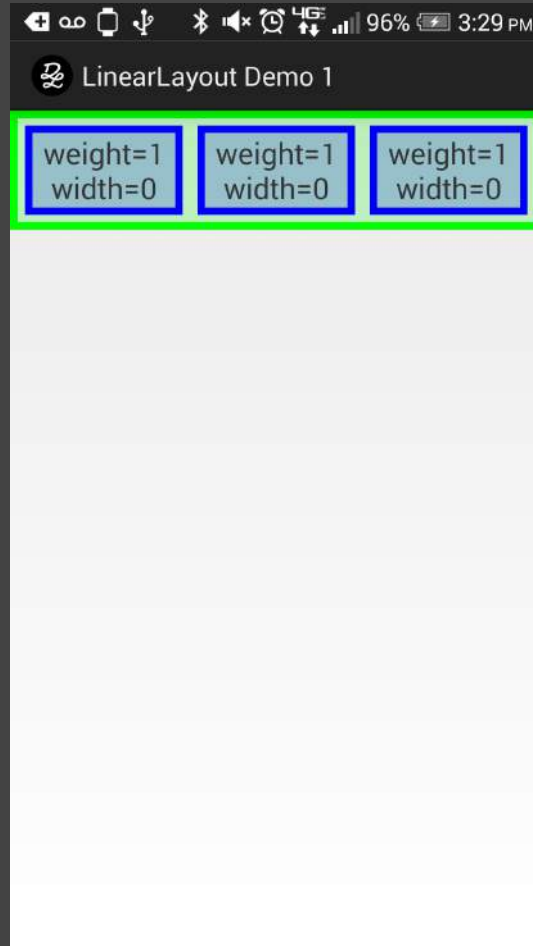
weight=0.5  
height=0

weight=1  
height=0

weight=0.5  
height=0

weightSum = 2

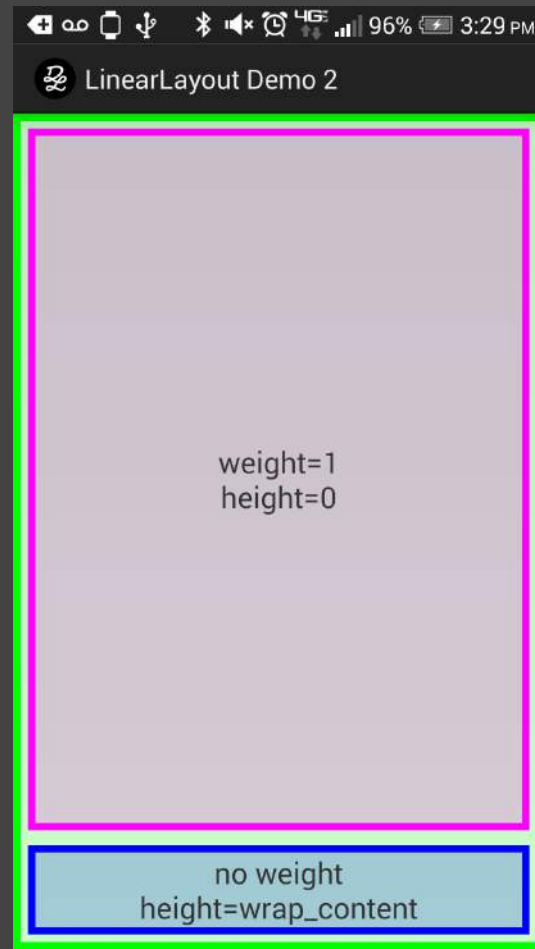
weightSum = 3



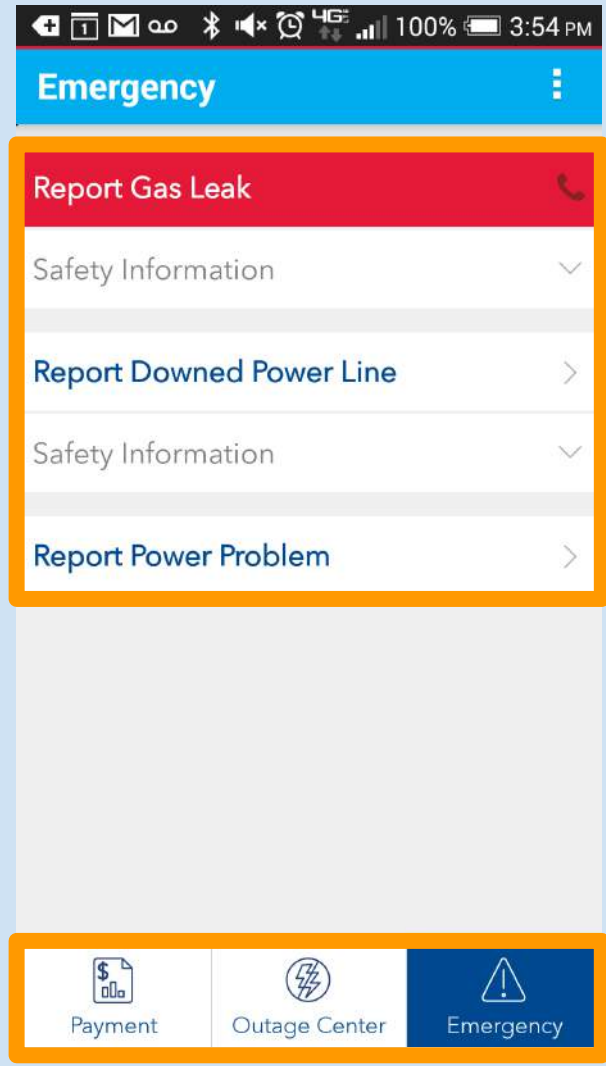
weightSum = 2

weightSum = 3

No weightSum!



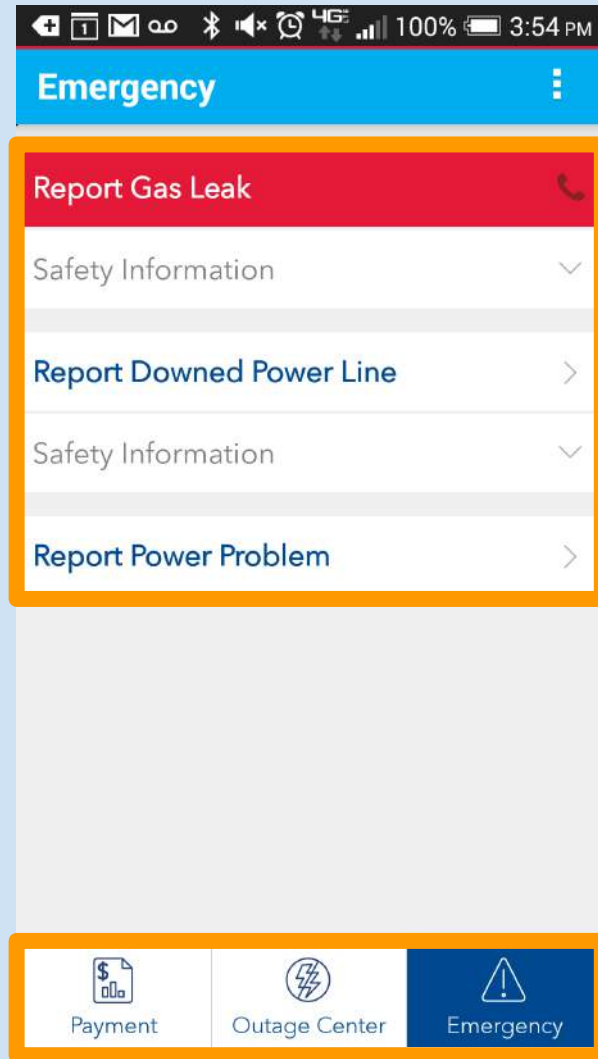
# LinearLayout cont'd



# LinearLayout cont'd

Vertical LinearLayout;  
children have fixed  
heights; no weights used

Horizontal LinearLayout;  
children have equal weights;  
all widths = 0dp



# Positioning in a RelativeLayout

Child View attributes:

```
android:layout_alignParentTop="true"  
android:layout_centerHorizontal="true"  
android:layout_alignRight="@+id/id_of_a_sibling"  
android:layout_toRightOf="@+id/id_of_a_sibling"  
android:layout_alignBaseline="true"
```

Most of the above have top/bottom/left/right variants.

Must be careful not to combine conflicting rules.

Relative Layout Demo 0

1

2

3

8

4

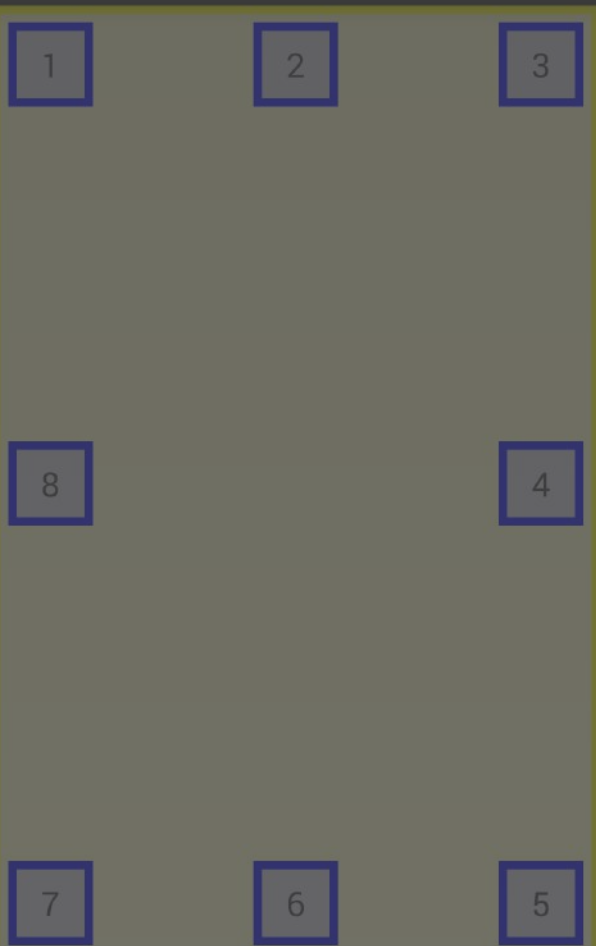
7

6

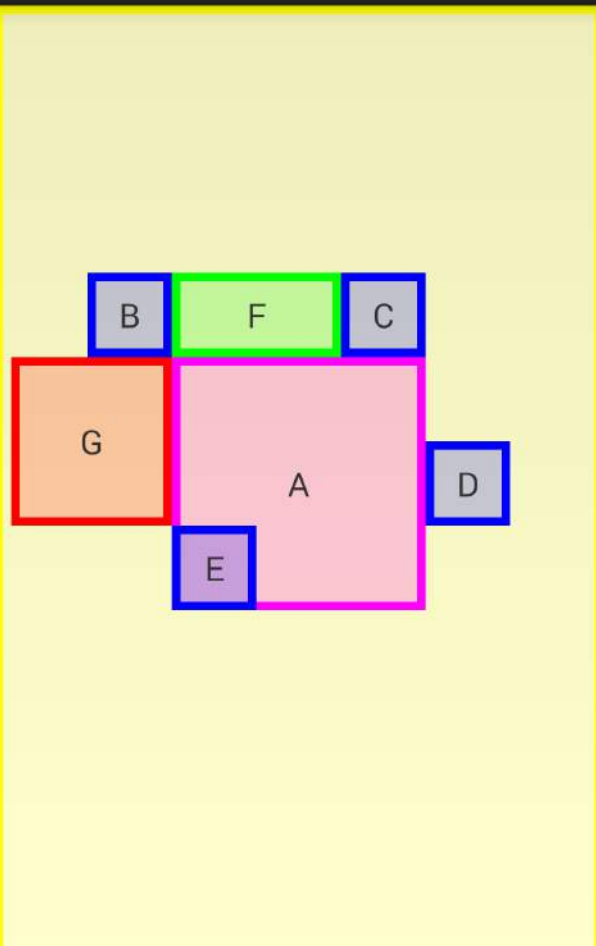
5



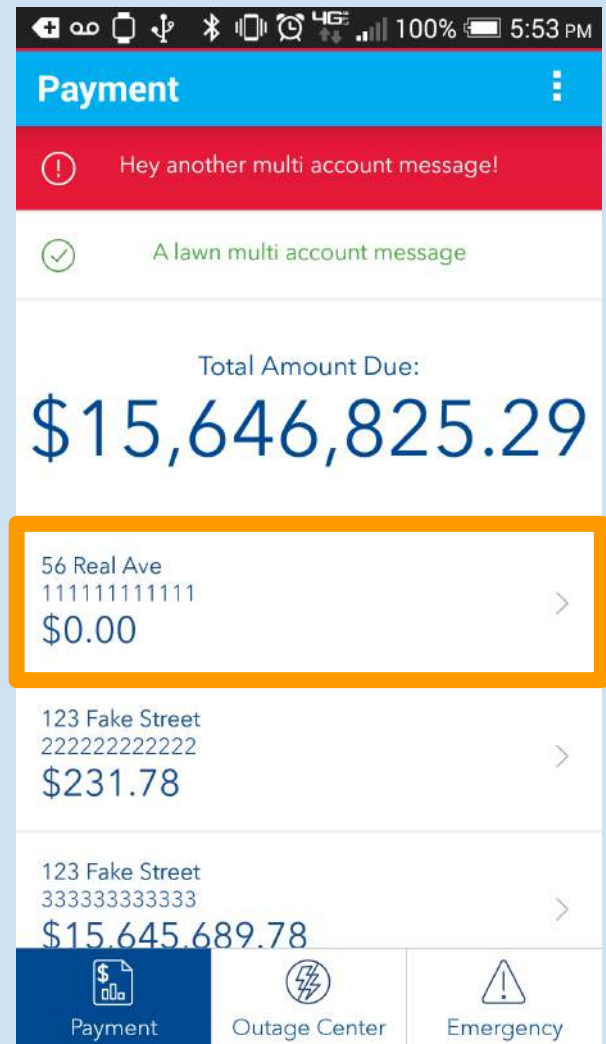
## Relative Layout Demo 0



## Relative Layout Demo 1

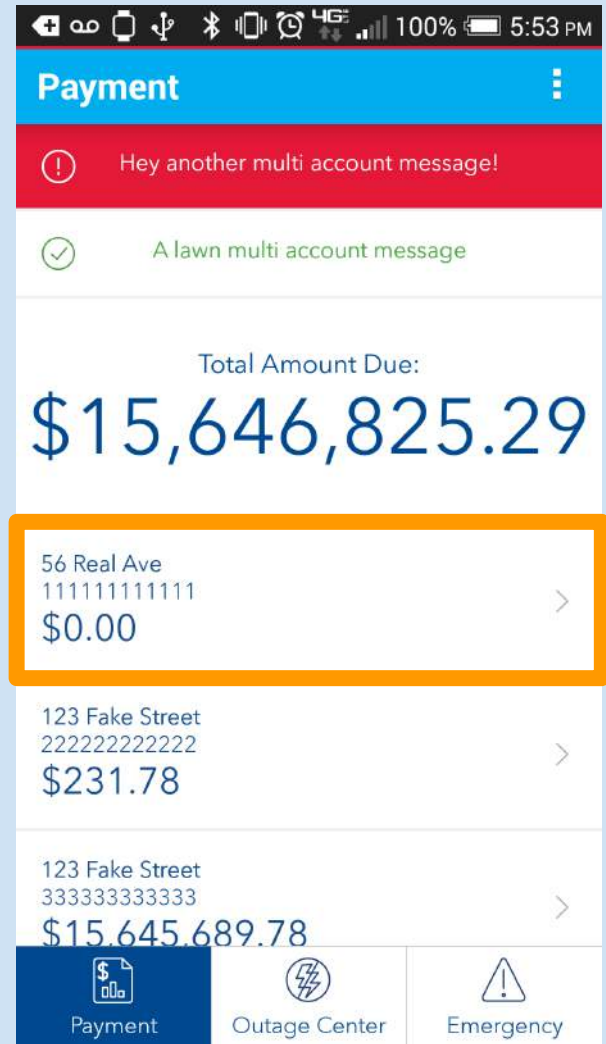
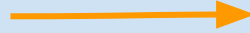


# RelativeLayout cont'd



# RelativeLayout cont'd

Single RelativeLayout;  
no LinearLayout used!



# Layout inflation in activities

```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // set activity layout  
        setContentView(R.layout.activity_main);  
    }  
}
```

# Layout inflation in fragments

```
public class ExampleFragment extends Fragment {  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
        Bundle savedInstanceState) {  
  
        // return inflated fragment layout  
        return inflater.inflate(R.layout.example_fragment, container, false);  
    }  
}
```

# Interacting with a view in code

```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // set activity layout  
        setContentView(R.layout.activity_main);  
  
        Button myButton = (Button) findViewById(R.id.my_button);  
  
        // set button click listener here, for example  
    }  
}
```

# Interacting with a view in code

```
public class MainActivity extends Activity {  
  
    private Button myButton;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // set activity layout  
        setContentView(R.layout.activity_main);  
  
        myButton = (Button) findViewById(R.id.my_button);  
    }  
  
    // other methods can now interact with myButton: set text, set  
    // enabled/disabled, etc.  
}
```

# Interacting with a view in code

Most view attributes can be read and changed using getters & setters.

Getters:

- `String contents = myEditText.getText().toString();`



# Interacting with a view in code

Most view attributes can be read and changed using getters & setters.

Getters:

- `String contents = myEditText.getText().toString();`

Setters:

- `myTextView.setText("Some String here.");`
- `myButton.setEnabled(false);`
- `myImageView.setVisibility(View.GONE);`

# Lists and Grids

Start with a List or array of Objects

# Lists and Grids

Start with a List or array of Objects



Adapter 'converts' each Object into an  
inflated layout

# Lists and Grids

Start with a List or array of Objects



Adapter 'converts' each Object into an  
inflated layout



ListView or GridView displays all those  
layouts

# Lists and Grids

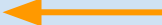
Start with a List or array of Objects



Adapter 'converts' each Object into an inflated layout



ListView or GridView displays all those layouts



Adapter is constructed and attached to ListView in code.

# Lists and Grids

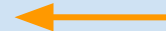
Start with a List or array of Objects



Adapter 'converts' each Object into an  
inflated layout



ListView or GridView displays all those  
layouts



ListView is part of  
Activity or Fragment  
xml layout

# Lists and Grids

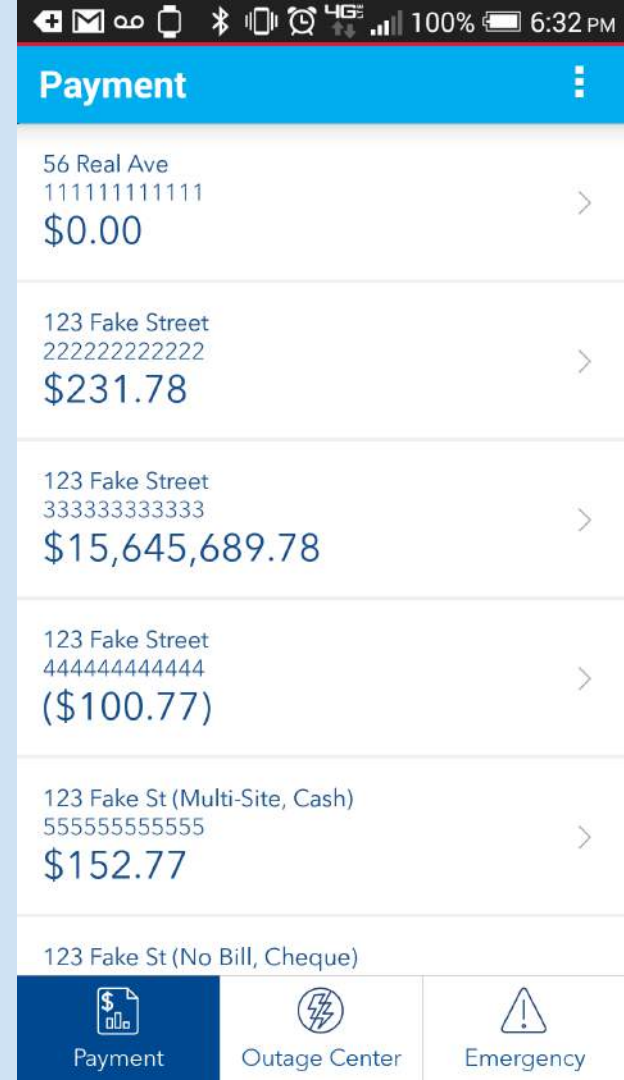
Start with a List of **DTE accounts**



Adapter 'converts' each **DTE account**  
into an inflated layout



ListView displays all those layouts



# **Why Fragment?**



# Why Fragment?

The screenshot shows a mobile application interface for making a payment. At the top, a status bar displays various icons and '100%' battery. Below it, a blue header bar contains a back arrow and the text 'Make a Payment'. The main content area is divided into sections: 'Payment Amount' set to '\$ 0.00', 'Payment Date' set to 'Today, October 9, 2014', and 'Payment Method' set to 'DISCOVER \*\*\*\*5554'. Below this, a list of payment methods is shown, each with a checkbox, logo, card details, and an action button. The 'DISCOVER' method is selected with a blue checkmark. At the bottom right, there is a blue button labeled 'Add Payment Method'.

Payment Method	Card Number	Expiration Date	Action
<input type="checkbox"/> VISA	7777	Expired 07/14	Edit
<input checked="" type="checkbox"/> DISCOVER	5554	Expires 01/27	Edit
<input type="checkbox"/> MasterCard	7894	Expires 01/18	Delete
<input type="checkbox"/> Checking	8734		Edit

[Add Payment Method](#)

# Why Fragment?

4G 100% 7:38 PM

< **Make a Payment**

Payment Amount \$ 0.00

Payment Date Today, October 9, 2014

Payment Method **DISCOVER** \*\*\*\*5554

☐ 7777 / Expired 07/14 Edit

☒ 5554 / Expires 01/27 Edit

☐ 7894 / Expires 01/18 Delete

☐ Checking 8734 Edit

Add Payment Method

4G 100% 7:39 PM

< **AutoPay**

You are currently enrolled in the AutoPay program

Account No.  
111111111111

☐ 7777 / Expired 07/14 Edit

☒ 5554 / Expires 01/27 Edit

☐ 7894 / Expires 01/18 Delete

☐ Checking 8734 Edit

Add Payment Method

Update

# Why Fragment?

**Make a Payment**

Payment Amount \$ 0.00

Payment Date Today, October 9, 2014

Payment Method **DISCOVER** \*\*\*\*5554

☐ VISA 7777 / Expired 07/14 Edit

☒ DISCOVER 5554 / Expires 01/27 Edit

☐ MasterCard 7894 / Expires 01/18 Delete

☐ Checking 8734 Edit

Add Payment Method

**AutoPay**

You are currently enrolled in the AutoPay program

Account No. 111111111111

☐ VISA 7777 / Expired 07/14 Edit

☒ DISCOVER 5554 / Expires 01/27 Edit

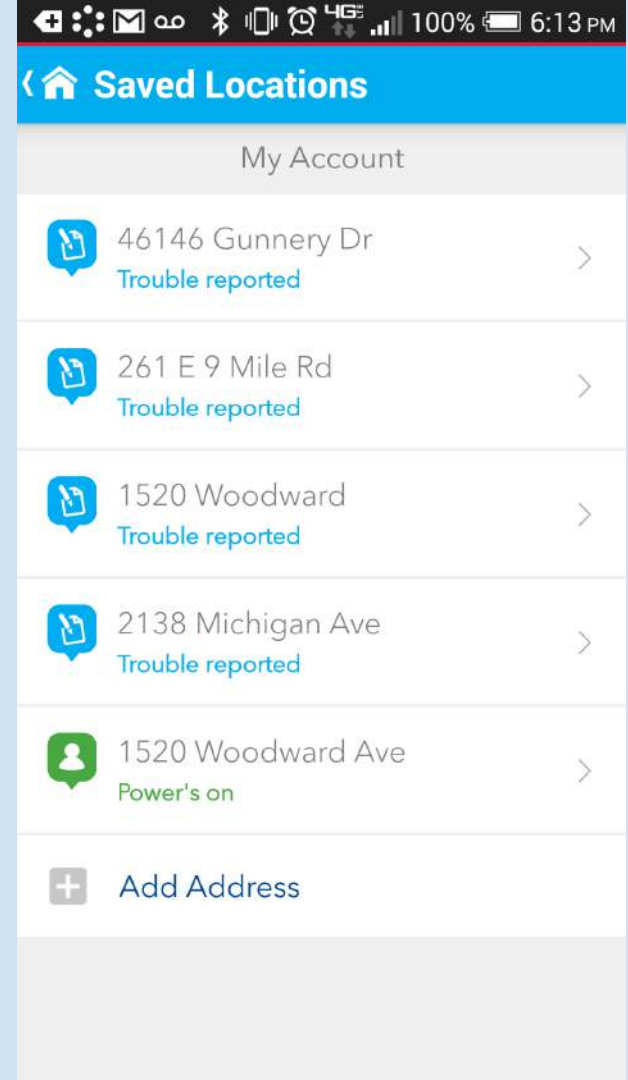
☐ MasterCard 7894 / Expires 01/18 Delete

☐ Checking 8734 Edit

Add Payment Method

Update

# Layout Breakdown 1



# Layout Breakdown 2

Report Problem

1 — 2 — 3 — 4

Please provide a phone number where you can be reached.

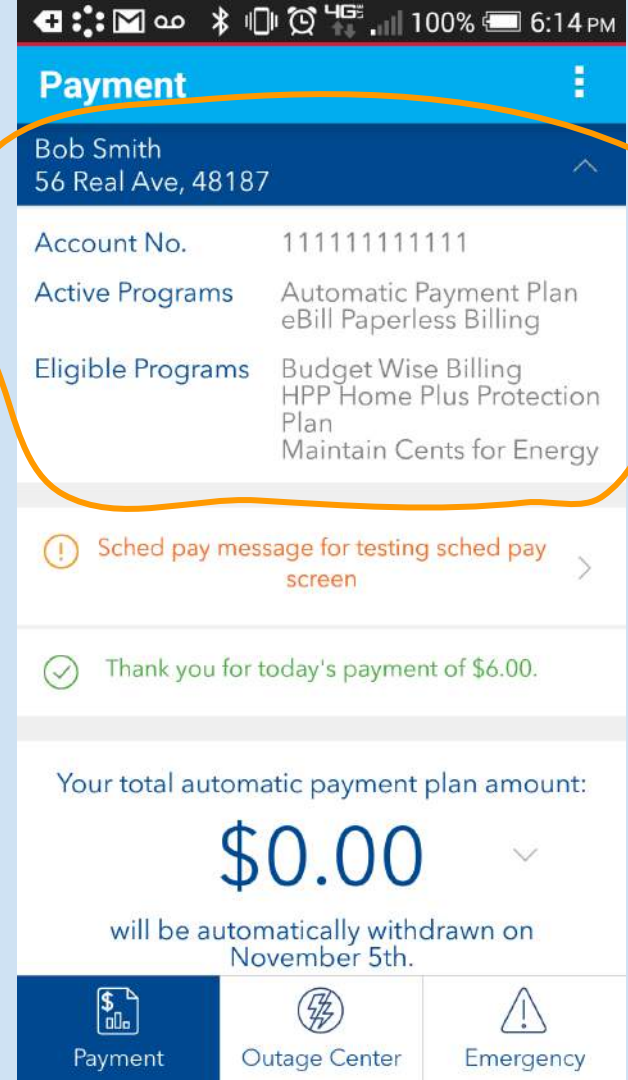
(313) 555-5555

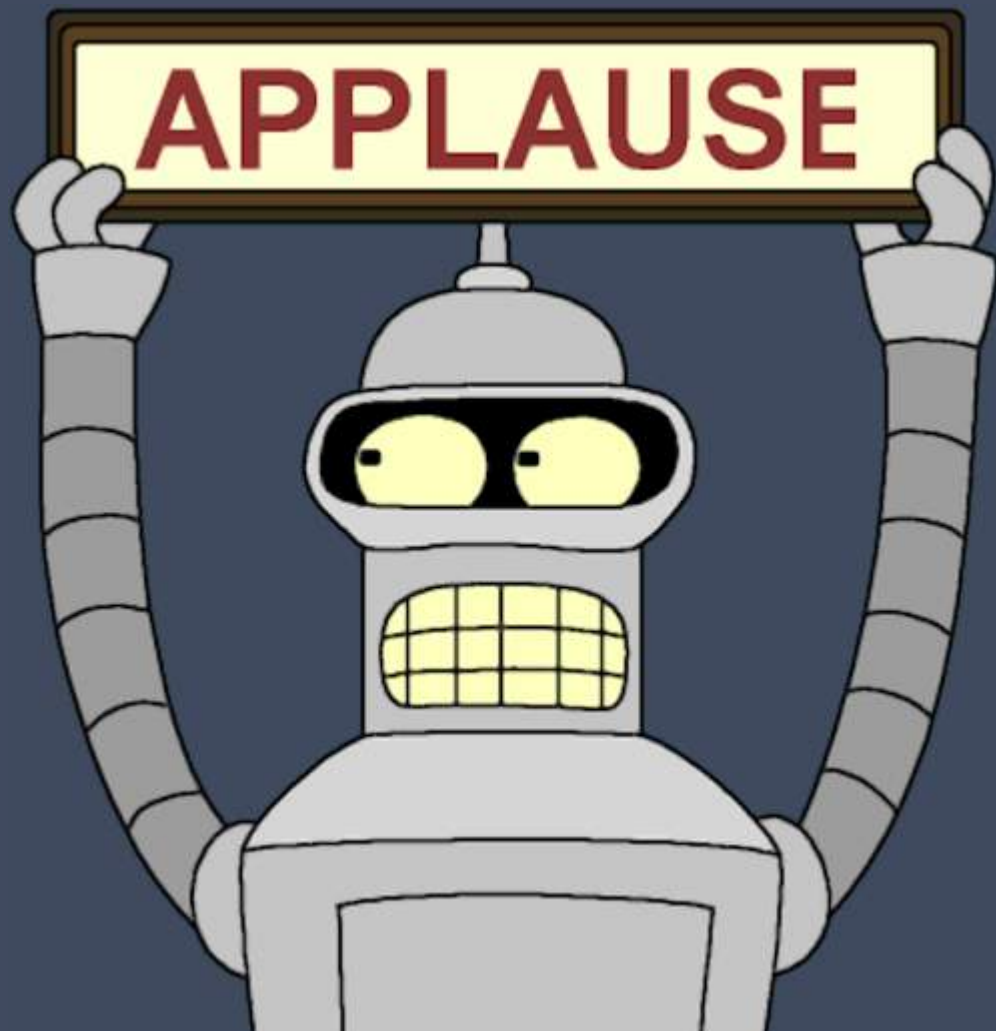
How would you like to be notified of status updates?

Email Text Push **None**

Review

# Layout Breakdown 3





# View attributes: background

```
android:background="#00FF3C"  
android:background="@color/color_name"  
android:background="@drawable/drawable_name"
```

Defines background color for `View`.

Can be static color, or state-based drawable (e.g. `Button`).

Does color view padding; does not color view margins.



# View attributes: visibility

```
android:visibility="visible"  
android:visibility="invisible"  
android:visibility="gone"
```

Sets visibility of `View`. Visible by default.

Invisible - not drawn but still takes up space in layout.

Gone - not drawn and doesn't take up space in layout.