Layouts/Widgets in Android

GUI widgets and their placement in Activity,

View Hierarchy

Slides By Alireza Kazemi, Content From: https://developer.android.com/guide/topics/ui/declaring-layout.html & https://developer.android.com/guide/topics/ui/notifiers/toasts.html

Layouts

- A layout defines the visual structure for a user interface

- You can declare a layout in two ways
  - Declaring UI elements in XML layout file
  - Instantiate layout elements at runtime (in Java code)
    - And adding them to proper ViewGroups (layouts)

Layout in 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>
```

Loading the XML Layout

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

Layout/View attributes: ID

- Any view object may have an integer ID
  - Which we can use to access it during program execution
- Defining a new ID
  - android:id="@+id/my_button"
- Referencing a resource by ID:
  - android:text="@id/hello_text" (without namespace)
  - android:text="@android:id/empty" (using a namespaces)
- Define a view/widget & assign an ID:

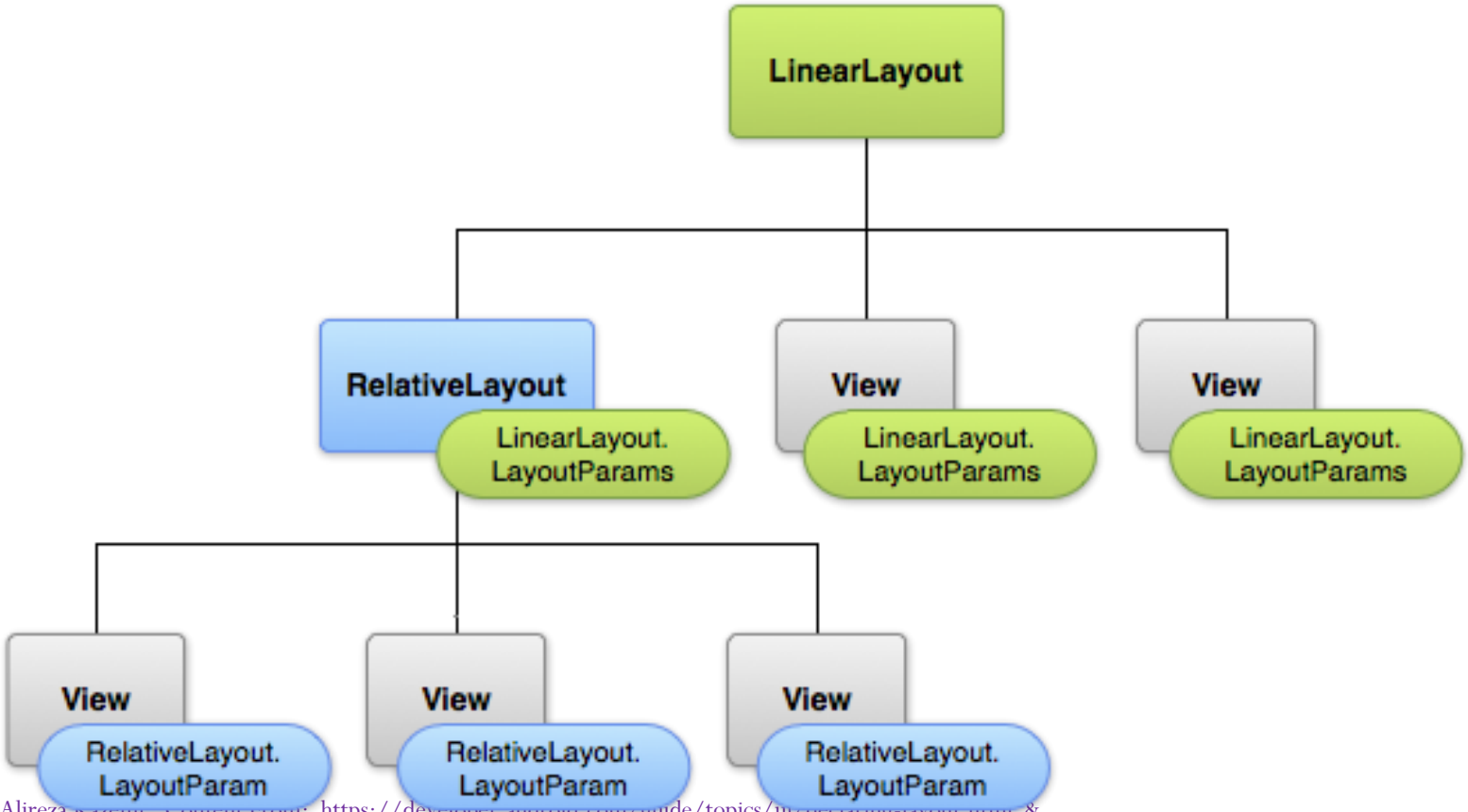
```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

- Access to a view object reference in Java Code:

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Layout Parameters

- XML layout attributes that have names like **layout_xxx**
- View hierarchy with layout parameters:



Layout Geometry

- Position: each view corresponds to a rectangular area
  - getLeft(), getTop()
  - getWidth(), getHeight()
  - getRight() => getLeft() + getWidth()
  - getBottom() => getTop() + getHeight()
- Size
  - Size = Width x Height,
- Two kind of sizes:
  - **Measured Size**
    - **getMesuredWidth(), getMeasuredHeight()**
    - The size that a View object **wants to be**
  - **Drawing Size**
    - **getWidth(), getHeight()**
    - The actual size of View object on screen **after doing the layout process**

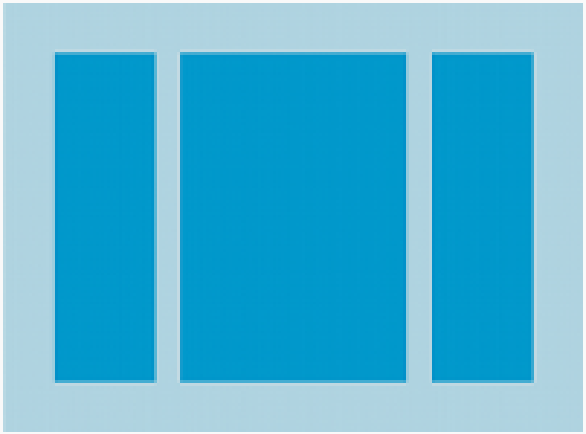
Layout Geometry

- Padding & Margin
  - Views just have **Padding**
  - ViewGroups have **Margin and Padding**

- View Padding:
  - setPadding(int, int, int, int)
  - getPaddingLeft()
  - getPaddingTop()
  - getPaddingRight()
  - getPaddingBottom()

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

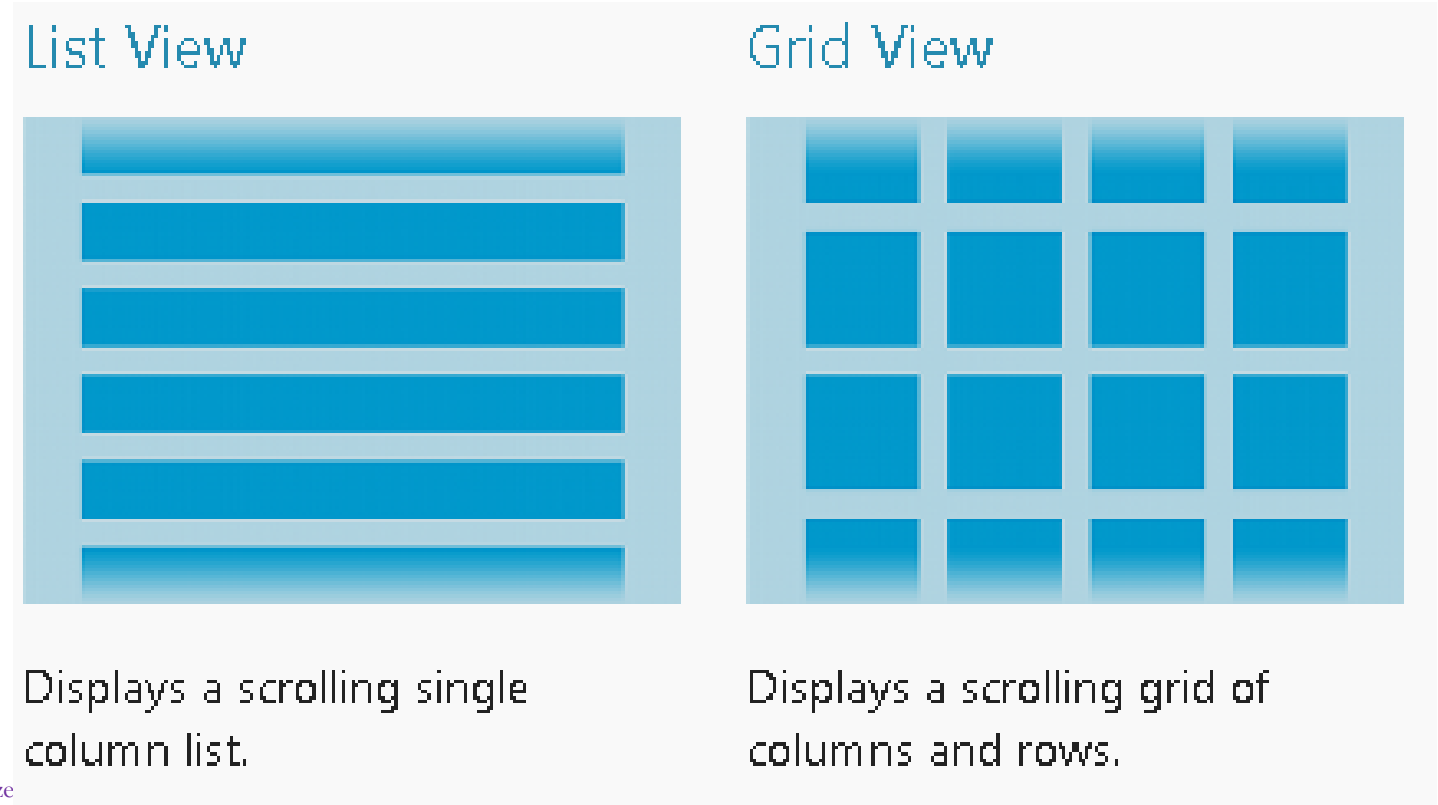
Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Web View

Displays web pages.

Common Layouts (Adapter-based)

- Layouts with a Data Adapter (**Adapter-based Views**)
- You can populated Adapter-based views using a Data Adapter
- These views/layouts inherit from an abstract **AdapterView**



https://developer.android.com/guide/topics/ui/notifiers/toasts.html

Data Adapter

- An object which provides data to a View via some data source
  - It also does the proper data fields projection/conversion prior to feeding the data to the connected view

- Example Data Sources:
  - An existing Array or ArrayList in the program
  - Data obtained from a Database query
    - Result Set
    - Cursor
  - Data obtained from a DataProvider (just in Android)
    - Cursor

Filling a view with ArrayAdapter

- Defining the data adapter

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, myStringArray);
```

The arguments for this constructor are:

- Your app `Context`
- The layout that contains a `TextView` for each string in the array
- The string array

- Using it

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

Filling a view with Data Cursor

- Defining the data adapter and using it

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,
                        ContactsContract.CommonDataKinds.Phone.NUMBER};
int[] toViews = {R.id.display_name, R.id.phone_number};
```

When you instantiate the `SimpleCursorAdapter`, pass the layout to use for each result, the `Cursor` containing the results, and these two arrays:

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
        R.layout.person_name_and_number, cursor, fromColumns, toViews, 0);
ListView listView = getListView();
listView.setAdapter(adapter);
```

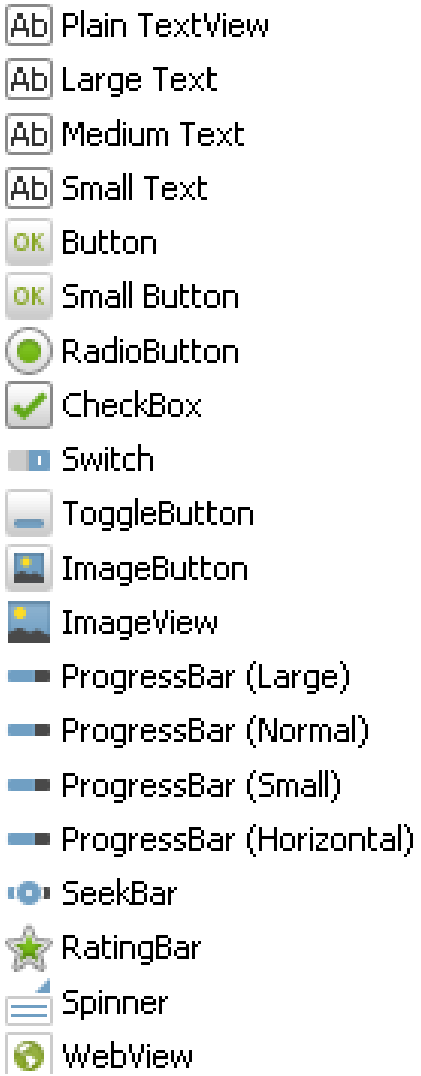
Handling Click Events on Adapter-based Views

```
// Create a message handling object as an anonymous class.
private OnItemClickListener mMessageClickedHandler = new OnItemClickListener() {
    public void onItemClick(AdapterView parent, View v, int position, long id) {
        // Do something in response to the click
    }
};

listView.setOnItemClickListener(mMessageClickedHandler);
```

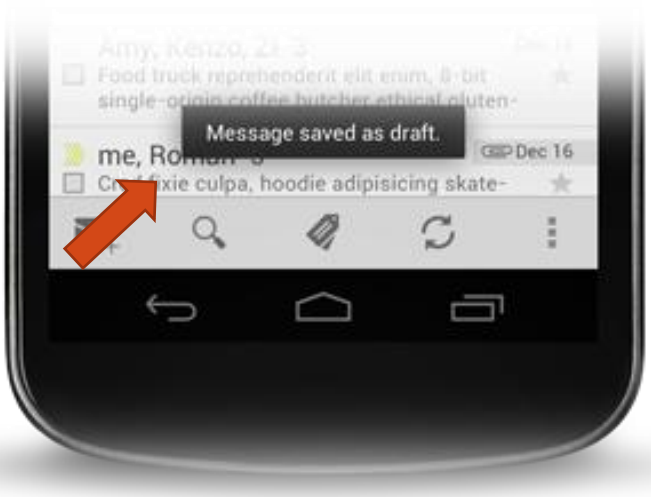
Input/Control/Widget Views

- Used for data entry/show from/to user
  - Button
  - TextEdit
  - Checkbox
  - RadioButton
  - ToggleButton
  - Spinner
  - Picker
  - ProgressBar
  - SeekBar
  - RatingBar



Toasts

- Messages that appear on screen
  - In a small box
  - For a transitional time
    - Short or Long




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

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

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

Toasts with custom layout

- First design a layout:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:id="@+id/toast_layout_root"
              android:orientation="horizontal"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:padding="8dp"
              android:background="#DAAA"
              >
    <ImageView android:src="@drawable/droid"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:layout_marginRight="8dp"
               />
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:textColor="#FFF"
              />
</LinearLayout>
```


https://developer.android.com/guide/topics/ui/notifiers/toasts.html

Toasts with custom layout ...

- Now use it as your Toast custom layout:

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_toast,
                (ViewGroup) findViewById(R.id.toast_layout_root));

TextView text = (TextView) layout.findViewById(R.id.text);
text.setText("This is a custom toast");

Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();
```