# Topics

- Declaring Layout
- Layout File
- Attributes
  - ID
  - Layout parameters
- Types of Layout
  - LinearLayout
  - RelativeLayout
  - TableLayout
  - FrameLayout
  - Tab layout

# Declaring Layout

# What is a Layout?

- Your layout is the architecture for the user interface in an Activity.
- It defines the layout structure and holds all the elements that appear to the user.

# How to declare a Layout? Two Options

- Option #1: Declare UI elements in XML (most common and preferred)
  - Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for UI controls called widgets (TextView, Button, etc.) and layouts.
- Option #2: Instantiate layout elements at runtime (in Java code)
  - Your application can create View and ViewGroup objects (and manipulate their properties) programmatically (in Java code).

# Example of using both options

- You can use either or both of these options for declaring and managing your application's UI
- Example usage scenario
  - You could declare your application's default layouts in XML, including the screen elements that will appear in them and their properties.
  - You could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time.

# Advantages of Option #1: Declaring UI in XML

- Separation of the presentation from the code that controls its behavior
  - You can modify UI without having to modify your source code and recompile
  - For example, you can create XML layouts for different screen orientations, different device screen sizes, and different languages
- Easier to visualize the structure of your UI (without writing any code)
  - Easier to design/debug UI
  - Visualizer tool (like the one in Eclipse IDE)

# Layout File

# Layout File Structure

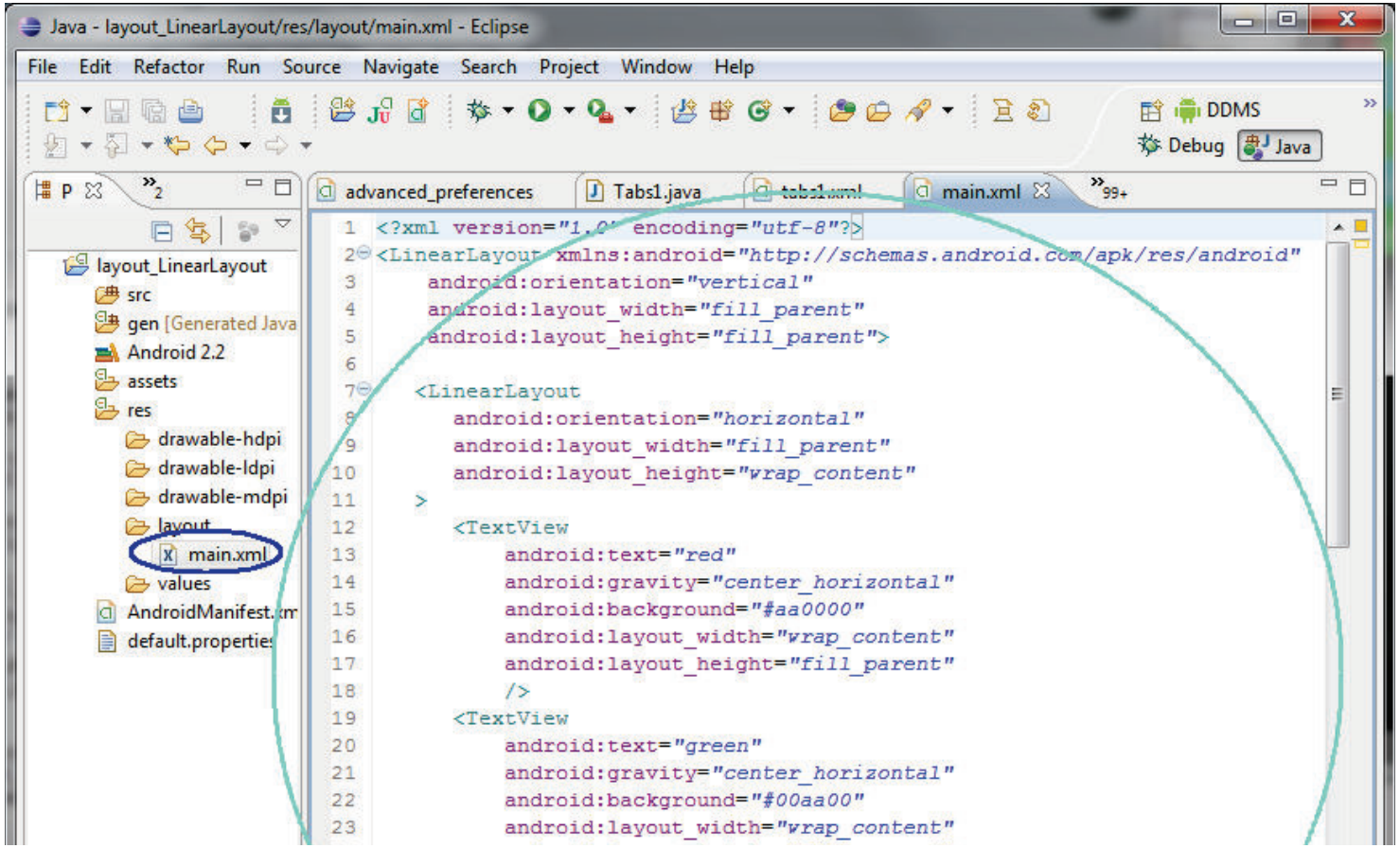
- Each layout file must contain exactly one root element, which must be a View (Button, for example) or ViewGroup object (LinearLayout, for example).
- Once you've defined the root element, you can add additional layout objects or widgets (View) as child elements to gradually build a View hierarchy that defines your layout.

# Example: Layout File

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="fill_parent"
            android:layout_height="fill_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>
```

# Where to create Layout file?

- Save the file with the .xml extension, in your Android project's *res/layout/* directory



# Load the Layout XML Resource

- When you compile your application, each XML layout file is compiled into a View resource.
- You should load the layout resource from your application code, in your *Activity.onCreate()* callback implementation.
  - By calling setContentView(), passing it the reference to your layout resource in the form of: *R.layout.layout_file_name*

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

# Attributes

# Attributes

- Every View and ViewGroup object supports their own variety of attributes.
  - Some attributes are specific to a View object (for example, TextView supports the textSize attribute), but these attributes are also inherited by any View objects that may extend this class.
  - Some are common to all View objects, because they are inherited from the root View class (like the id attribute).
  - Other attributes are considered "layout parameters," which are attributes that describe certain layout orientations of the View object, as defined by that object's parent ViewGroup object.
  - These attributes are typically in XML form

# Attributes: ID

# ID Attribute

- Any View object may have an integer ID associated with it, to uniquely identify the View within the tree.
- When the application is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the id attribute.
- Syntax
  - *android:id="@+id/my_button"*

# ID Attribute - Android Resource ID

- There are a number of other ID resources that are offered by the Android framework.
- When referencing an Android resource ID, you do not need the plus-symbol, but must add the android package namespace
  - *android:id="@android:id/empty"*
- With the android package namespace in place, we're now referencing an ID from the *android.R* resources class, rather than the local resources class.

# How to reference views in Java code?

- Assuming a view/widget is defined in the layout file with a unique ID

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

- Then you can make a reference to the view object via *findViewById(R.id.<string-id>)*.

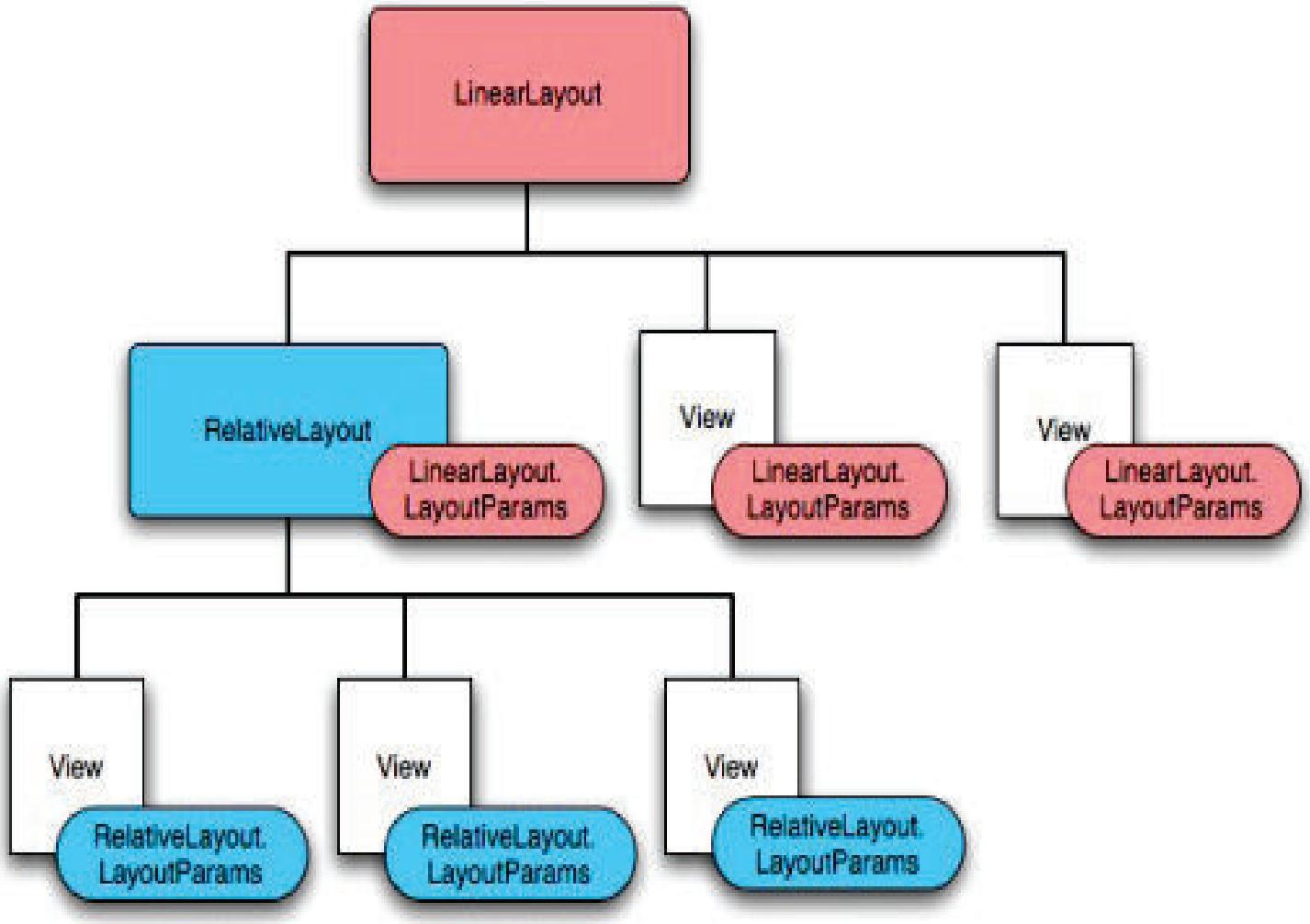
```
Button myButton = (Button) findViewById(R.id.my_button);
```

# Attributes:
# Layout Parameters

# What Are Layout Parameters?

- XML layout attributes named *layout_something* define layout parameters for the View that are appropriate for the ViewGroup in which it resides
- Every ViewGroup class implements a nested class that extends *ViewGroup.LayoutParams*.
  - This subclass contains property types that define the size and position for each child view, as appropriate for the view group.

Parent view group defines layout parameters for each child view (including the child view group)



# layout_width & layout_height

- wrap_content
  - tells your view to size itself to the dimensions required by its content
- fill_parent
  - tells your view to become as big as its parent view group will allow.
- match_parent
  - Same as fill_parent
  - Introduced in API Level 8

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

# layout_weight

- Is used in LinearLayouts to assign "importance" to Views within the layout.
- All Views have a default layout_weight of zero
- Assigning a value higher than zero will split up the rest of the available space in the parent View, according to the value of each View's layout_weight and its ratio to the overall layout_weight specified in the current layout for this and other View elements.

# Layout Types

# Layout Types

- All layout types are subclass of ViewGroup class
- Layout types
  - LinearLayout
  - RelativeLayout
  - TableLayout
  - FrameLayout
  - Tab layout

# LinearLayout

- Aligns all children in a single direction — vertically or horizontally, depending on how you define the orientation attribute.
- All children are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are
- A LinearLayout respects
  - margins between children
  - gravity (right, center, or left alignment) of each child.
  - weight to each child

# RelativeLayout

- RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID)
  - You can align two elements by right border, or make one below another, centered in the screen, centered left, and so on
- Elements are rendered in the order given, so if the first element is centered in the screen, other elements aligning themselves to that element will be aligned relative to screen center.

# RelativeLayout Example

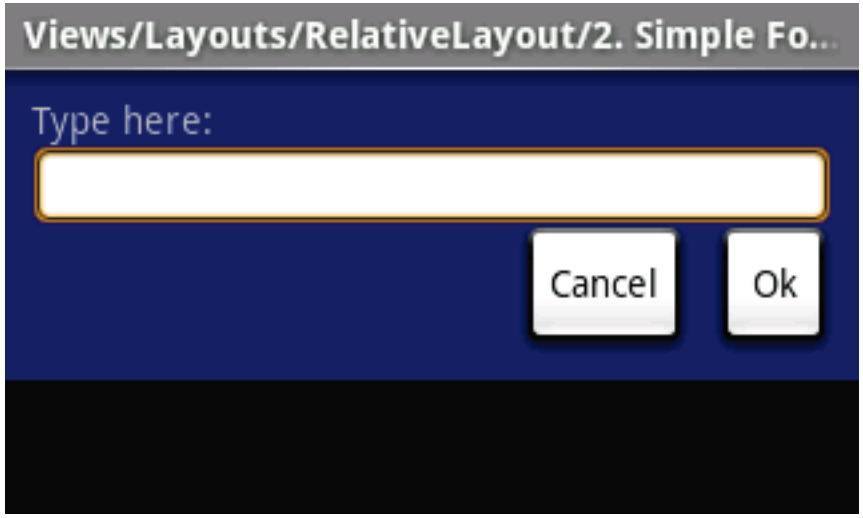
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:background="@drawable/blue"
            android:padding="10px" >

    <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="10px"
            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>
```



# TableLayout

- TableLayout positions its children into rows and columns
- TableRow objects are the child views of a TableLayout (each TableRow defines a single row in the table).
- Each row has zero or more cells, each of which is defined by any kind of other View.
- Columns can be hidden, marked to stretch and fill the available screen space, or can be marked as shrinkable to force the column to shrink until the table fits the screen.

# TableLayout Example

```
<?xml version="1.0" encoding="utf-8"?>
<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:text="@string/table_layout_4_open"
         android:padding="3dip" />
      <TextView
         android:text="@string/table_layout_4_open_shortcut"
         android:gravity="right"
         android:padding="3dip" />
   </TableRow>

   <TableRow>
      <TextView
         android:text="@string/table_layout_4_save"
         android:padding="3dip" />
      <TextView
         android:text="@string/table_layout_4_save_shortcut"
         android:gravity="right"
         android:padding="3dip" />
   </TableRow>
</TableLayout>
```

# FrameLayout

- FrameLayout is the simplest type of layout object. It's basically a blank space on your screen that you can later fill with a single object
  - For example, a picture that you'll swap in and out.
- All child elements of the FrameLayout are pinned to the top left corner of the screen; you cannot specify a different location for a child view.
  - Subsequent child views will simply be drawn over previous ones, partially or totally obscuring them (unless the newer object is transparent).

# FrameLayout Example

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

    <TextView
       android:text="yellowyellowyellow"
       android:gravity="bottom"
       android:background="#aaaa00"
       android:layout_width="wrap_content"
       android:layout_height="120dip"/>
    <TextView
       android:text="greengreengreen"
       android:gravity="bottom"
       android:background="#00aa00"
       android:layout_width="wrap_content"
       android:layout_height="90dip" />
   <TextView
       android:text="blueblueblue"
       android:gravity="bottom"
       android:background="#0000aa"
       android:layout_width="wrap_content"
       android:layout_height="60dip" />
    <TextView
       android:text="redredred"
       android:gravity="bottom"
       android:background="#aa0000"
       android:layout_width="wrap_content"
       android:layout_height="30dip"/>
</FrameLayout>
```