

The background features several horizontal bars in various colors: a rainbow gradient bar at the top, a blue bar, a red bar, a yellow bar, and a dark blue bar at the bottom. A vertical blue bar is on the left side. A black crosshair is centered on the left side, intersecting the horizontal bars.

ANDROID LAYOUTS AND WIDGETS



Introduction

- Parts of the Android screen
- Sizing widgets and fonts
- Layouts and their characteristics
- Buttons, checkboxes and other widgets
- Ways to program event handlers



Android UI Concepts

- All Android applications have similar user interface elements
 - Some of this has changed from the first phones but that was a long time ago
 - Some of the first devices had a hardware menu button



Android Screen Elements (1)

- The System bar appears along the top of the screen and contains system and notification icons
- Toolbars typically appear across the top of the application and can be rendered differently
 - The App bar (action bar) is a special form of toolbar having the one or many of a navigation icon, title, filter, and action icons and menu icons



Android Screen Elements (2)

- The navigation bar appears along the bottom of the device
- The remainder of the screen is covered by a layout
- There are other popup windows and messages that will be discussed later with multiple activities



Android Font Sizes

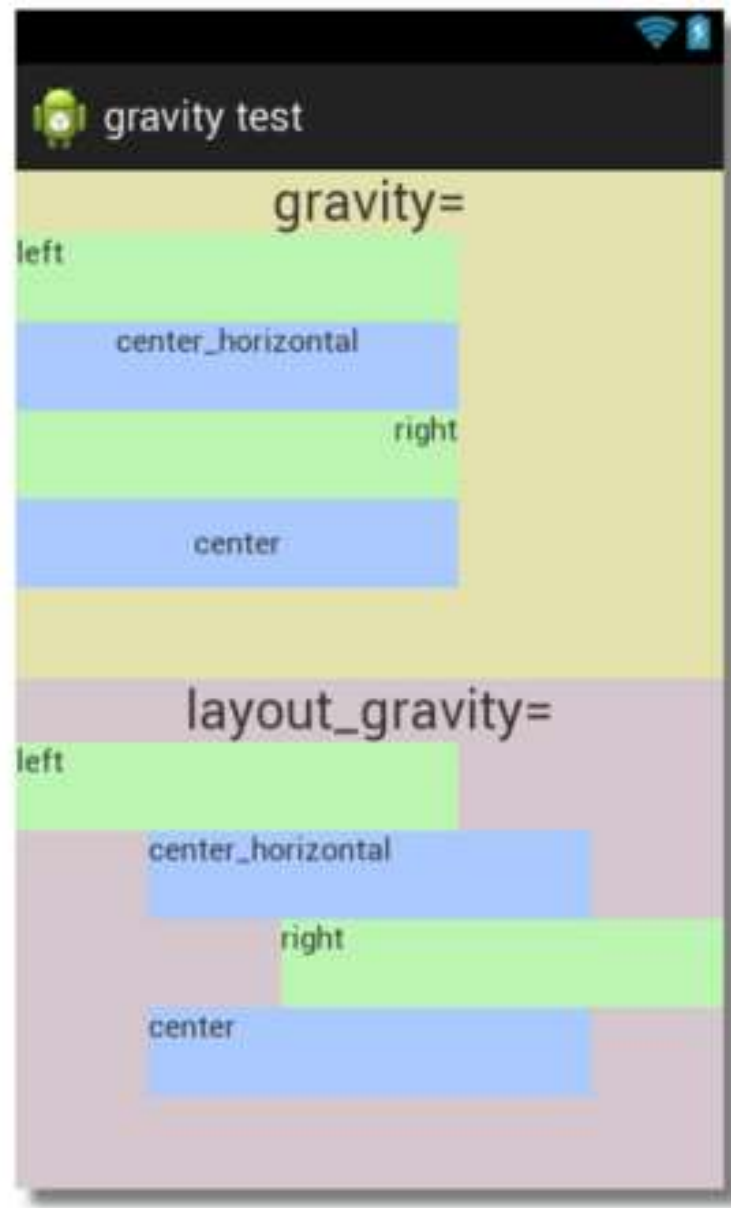
- All views have a size Fonts have a size. We can use the following units of measure
- **px** – screen pixels
- **in** – inches based on the physical screen size
- **mm** – millimeters based on the physical screen size
- **pt** – 1/72 of an inch based on physical screen size
- **dp** or **dip** – device independent unit based relative to a 160 dpi screen
- **sp** – similar to dp but used for font sizes



Gravity

- Gravity is a way to define which side of the parent an object is anchored
- **android:gravity** refers to the gravity of the view's content (text for example)
- **android:layout_gravity** refers to the gravity of the view itself
 - Only works with a linear layout
 - Widgets are anchored along the parent layout
- **center_vertical, center_horiztonal, center**

Gravity

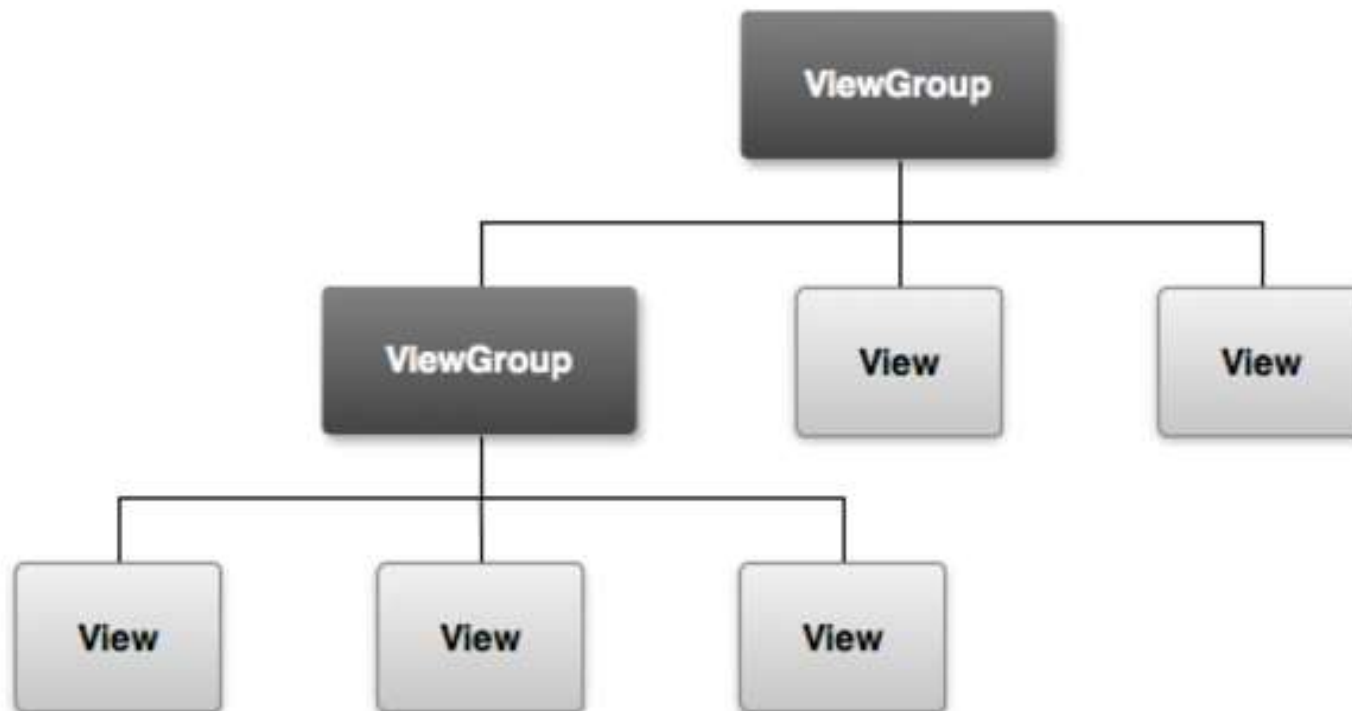




Android Layouts (Introduction)

- Visual elements are built from
 - **ViewGroup**: These are the different layouts which conning other ViewGroups and other widgets (**LinearLayout**, **RelativeLayout**)
 - **Views** are the widgets with which the user interacts (**Button**, etc).
- **ViewGroup** and **View** objects are organized into a hierarchy

Android Layouts (Illustration)



Application Components (Layout)

- There are different types of layouts for a screen (View)
 - Using a **LinearLayout**, child widgets are aligned horizontally or vertically
 - Using a **RelativeLayout** child widgets are aligned relative to each other or the parent window
- There are other layouts not discussed in this section

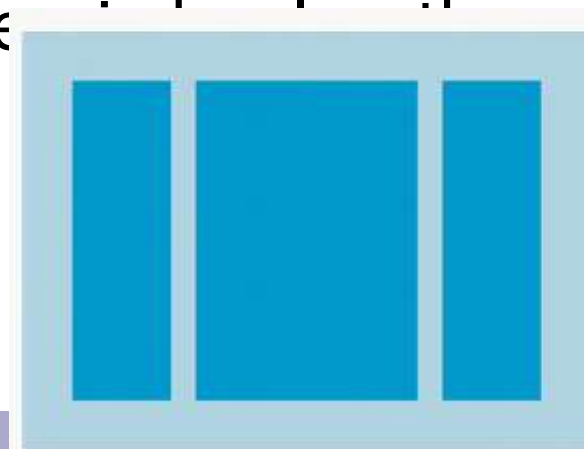
Application Components (Layout) (1)

- A layout describes the visual structure for a UI element, such as the UI for an activity
- It's an XML document, so you need some familiarity with XML
- Android provides an XML vocabulary that corresponds to the **View** classes and subclasses, such as those for widgets and layouts

Application Components (`LinearLayout`)

- `LinearLayout` aligns child objects vertically or horizontally
 - Use the `android:orientation` attribute to specify the layout direction (`vertical` / `horizontal`)
 - Scrollbars appear if the content exceeds the screen length

```
<LinearLayout 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"
    android:orientation="vertical" >
```



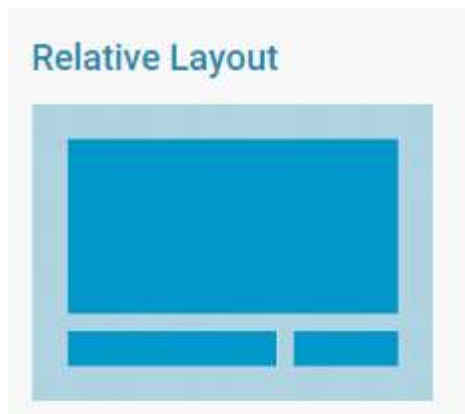
Application Components (`LinearLayout`)

- Create multiple nested `LinearLayout` widgets to nest vertical and horizontal layouts

Application Components (RelativeLayout)

- **RelativeLayout** aligns objects relative to each other (siblings)
- Such as:
 - Child A to the left of child B
 - Or align to the parent

```
<LinearLayout  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:orientation="horizontal" >
```



Android RelativeLayout (Positioning Child Widgets)

- Position children relative to each other or the parent
 - By default, all views are drawn at the top left of the layout
- These properties belong to `RelativeLayout.LayoutParams`
- There are many properties but we really only need a few for most layouts

Android RelativeLayout (Positioning Child Widgets)

- **android:layout_alignParentTop**
 - If **true** makes the top edge of this widget match the top edge of the parent
- **android:layout_alignTop**
 - Positions the top edge of this view to match the one with the specified resource ID

Android RelativeLayout (Positioning Child Widgets)

- `android:layout_below`
 - Positions the top edge of this view below the one with the specified resource id
- `android:layout_toRightOf`
 - Positions the left edge of this view to the right of the one with the specified resource id



Introduction to Input Controls

- **TextView** operates like a VB TextBox
- **Button** operates like a VB button
- **CheckBox** operates like a VB check box
- Etc..
- All are configured as XML in a layout file

TextView Control

- It displays a prompt to the user
 - **labelFor** attribute contains a reference to the **EditText** control in which the user enters input
 - **text** attribute contains the visible text

```
<TextView  
    android:id="@+id/tvLength"  
    android:labelFor="@+id/txtLength"  
    android:layout width="wrap_content"  
    android:layout height="wrap_content"  
    android:text="@string/length_prompt" />
```

EditText Control

- An editable text control similar to a text box
 - `android:inputtype` defines the allowable input values
 - `android:text` contains the text
 - Runtime `getText` and `setText` method retrieve or set the text, respectively

```
<EditText  
    android:id="@id/txtLength"  
    android:layout width="match_parent"  
    android:layout height="wrap_content"  
    android:inputType="numberSigned"  
>
```

EditText Control

- Using `getText` and `setText`

```
final EditText txtDemo;  
txtDemo = (EditText) findViewById(R.id.txtDemo)  
txtDemo.setText("Hello");  
String s = txtDemo.getText.toString();
```

Button Control

- A button has a width and height
- The button's text appears in strings.xml
- As you will see, there are several ways to wire events

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_text"  
    ... />
```

CheckBox Control

- As in most environments, it's a two-state control that is checked or unchecked
 - The `isChecked` accessor indicates whether the box is checked or not
 - The `setChecked` mutator accepts a Boolean argument to set the state of the widget (checked / unchecked)



Handling Events

- There are different ways to handle events in Android
- We will discuss a few of them
 - Link XML document to handler

Application Components

(Handling a Click – Method 1)

- When the user clicks a button, the object receives an **onClick** event which you can handle

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

Application Components

(Handling a Click – Method 2)

- The event handler can also be declared programmatically using an anonymous class
 - The book uses this technique

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

Application Components

(Handling a Click – Method 3)

- Using this technique, you create a class that implements the **OnClickListener** interface
 - When you implement this interface, you must implement the **onClick** method
- Call the **setOnClickListener** for the view passing an instance of the above class

Application Components

(Handling a Click – Method 3)

- X

```
...  
final Button button3 = (Button) findViewById(R.id.btnTechnique3);  
button3.setOnClickListener(new HandleClickTechnique3());
```

```
private class HandleClickTechnique3 implements OnClickListener  
{  
    public void onClick(View v)  
    {  
        TextView tvResult;  
        tvResult = (TextView) findViewById(R.id.tvResult);  
        tvResult.setText("btnTechnique3 Clicked");  
    }  
}
```

Application Components (Activities) (onCreate)

- **@Override** indicates that we are overriding a base class method
 - It's an informative annotation
 - Annotations are used to control compiler behavior
 - Similar to .NET attributes

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_quiz);
}
```

Application Components (Activities) (`onCreate`)

- `super.onCreate` calls the base class method
 - Super is roughly equivalent `MyBase` in VB
 - It typically appears as the first statement in the method

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_quiz);
}
```

Application Components (Activities) (onCreate)

- `setContentView` takes one argument – the resource id corresponding to the activity
- It associates a particular view with the activity
 - The resource is always named R
 - Layout is the layout that you want to use
 - Followed by the resource id of the layout

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_quiz);
}
```