Layout and Control in UI

• The user interface (UI) is the graphical interface user can see and interact with your app comprising UI controls like textbox, button, checkbox, radio button, date, picker, and many more, and is presented in a variety of layouts.

• Every Android's user interface is composed of a hierarchical collection of View and ViewGroup objects.

• The View class is the base class of all UI controls or Widgets. Every visible UI control occupies an area on your screen and provides the point of interaction between you and the app through events like clicking, touching, and keying.

• The ViewGroup class, on the other hand, is the invisible organizer(container) that provides a variety of UI layouts to position the various View objects on your screen. For ex., you may arrange the UI controls one above another in a linear fashion by LinearLayout layout, or relative to each other by RelativeLayout layout.

• ViewGroup class is also derived from View class, you can have a ViewGroup object (child) nested in another ViewGroup object (parent), and each can take on different layouts.

Refer to: [http://www.codeproject.com/Articles/807524/Android-UI-Layouts-and-Controls](http://www.codeproject.com/Articles/807524/Android-UI-Layouts-and-Controls)
Linear Layout Orientation
Relative layout
Table Layout
Frame Layout

- A `FrameLayout` is a `ViewGroup` that divides the screen into blocks of area each of which is supposed to holds a single child View
Buttons and Icon
TextView and EditText
Check Box and radio Button
ToggleButton

- A ToggleButton is an on/off switch. It can be used in turning on and off features like vibration mode, silent mode, WiFi connection.
A Spinner is just a fanciful name for the all-too-familiar dropdown list.
Progress Bar
DatePicker and TimePicker
ListView

- A ListView is a ViewGroup that displays a list of scrollable items from a data source via an Adapter.
- An Adapter acts as the middle man to pull rows of data from a data source such as an array or database, and to convert each row of data into a View which is then inserted into the ListView.
<ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/listView"
    android:padding="20dp" />

Item 1
Sub item 1

Item 2
Sub item 2

Item 3
Sub item 3

Item 4
Sub item 4

Item 5
Sub item 5

Item 6
Sub item 6

Item 7
Sub item 7

Item 8

Aries
Taurus
Gemini
Cancer
Leo
Virgo
Libra
Scorpio
Sagittarius
Capricorn
UI layouts

- LinearLayout
- RelativeLayout
- TableLayout
- FrameLayout
- RadioGroup
- ListView
- GridView

UI Controls

- Button
- TextView
- EditText
- CheckBox
- RadioButton
- ToggleButton
- Spinner
- AutoCompleteTextView
- Progress Bar
- Pickers
- ImageView
Android Interfaces: Code or XML

Two ways to create UI of your App:

• Write code using SDK with classes like LinearLayout, TextView, ....

• Create XML files in res/Layout (i.e. main.xml) where Android XML view tags like <LinearLayout> <TextView>, etc.

View: drawable object, controls
UI contains a hierarchy of Views
View is a class, subclassed by the drawable objects in the UI

References:
http://developer.android.com/guide/topics/ui/overview.html
package demo;

import android.app.Activity;
import android.widget.TextView;
import android.os.Bundle;

public class AndroidDemo extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Activity is a subclass of context, so the TextView takes this as a parameter
        TextView tv = new TextView(this);
        tv.setText("Hello");
        setContentView(tv);
    }
}

Manual Declarative UI

main.xml Layout File:

```xml
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/textview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="@string/hello"/>
```

strings.xml resource file:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello Again</string>
    <string name="app_name">AndroidDemo</string>
</resources>
```
Layout xml

- `<TextView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="@string/hello"/>

- `xmlns:android` XML namespace declaration that tells the Android tools that you are going to refer to common attributes defined in the Android namespace. The outermost tag in every Android layout file must have this attribute.

- `android:layout_width` This attribute defines how much of the available width on the screen this View should consume. As it's the only View so you want it to take up the entire screen, which is what a value of "fill_parent" means.

- `android:layout_height` This is just like `android:layout_width`, except that it refers to available screen height.

- `android:text` This sets the text that the TextView should display. In this example, you use a string resource instead of a hard-coded string value. The hello string is defined in the `res/values/strings.xml` file.
XML interface

- Layouts defined with XML located in res/layout
What’s R?

/* AUTO-GENERATED FILE. DO NOT MODIFY. This class was automatically generated by the * aapt tool from the resource data it found. It should not be modified by hand. */

package demo;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int textview=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
Android Event Handlers

From the code file for the activity:

```java
Button ok = (Button) findViewById(R.id.button1);
ok.setOnClickListener(new View.OnClickListener()
{
    public void onClick(View v)
    {
        CharSequence s = et.getText();
        tv.setText("Welcome, \" + s);
    }
});
```
Visually Creating XML interface

- res/layout/main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <EditText android:text="@string/hello"
        android:id="@+id/editText1"
        android:inputType="textMultiLine"
        android:layout_width="169dp"
        android:layout_height="115dp"
        android:layout_x="11dp"
        android:layout_y="20dp"></EditText>

    <Button android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        android:layout_x="27dp"
        android:layout_y="146dp"></Button>
</AbsoluteLayout>
```
## Layout Tags

- **LinearLayout**
  A Layout that arranges its children in a single column or a single row. The direction of the row can be set by calling `setOrientation()`. You can also specify gravity, which specifies the alignment of all the child elements by calling `setGravity()` or specify that specific children grow to fill up any remaining space in the layout by setting the weight member of `LinearLayout.LayoutParams`. The default orientation is horizontal.

- **AbsoluteLayout**
  A layout that lets you specify exact locations (x/y coordinates) of its children. Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.

- **RelativeLayout**
- **FrameLayout**
- **TableLayout**
ListView <ListView .....>

- A view that shows items in a vertically scrolling list.

Attributes
- `android:divider` Drawable or color to draw between list items.
- `android:dividerHeight` Height of the divider.
- `android:entries` Reference to an array resource that will populate the ListView.
- `android:footerDividersEnabled` When set to false, the ListView will not draw the divider before each footer view.
- `android:headerDividersEnabled` When set to false, the ListView will not draw the divider after each header view.
A view that shows items in a center-locked, horizontally scrolling list.

The default values for the Gallery assume you will be using Theme_galleryItemBackground as the background for each View given to the Gallery from the Adapter. If you are not doing this, you may need to adjust some Gallery properties, such as the spacing.

Attributes

- **android:animationDuration** [setAnimationDuration(int)]
  Sets how long a transition animation should run (in milliseconds) when layout has changed.
- **android:gravity** [setGravity(int)]
  Specifies how to place the content of an object, both on the x- and y-axis, within the object itself.
- **android:spacing** [setSpacing(int)]
- **android:unselectedAlpha** [setUnselectedAlpha(float)]
  Sets the alpha on the items that are not selected.
Code—setting up Gallery

- `@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Gallery gallery = (Gallery) findViewById(R.id.gallery);
    gallery.setAdapter(new ImageAdapter(this));

    gallery.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView parent, View v, int position, long id) {
            Toast.makeText(HelloGallery.this, "" + position, Toast.LENGTH_SHORT).show();
        }
    });
}
Views and ViewGroups

- An Activity can contain views and ViewGroups.
- `android.view.View.*` = base class for all Views.
- example sub-classes include: `TextView`, `ImageView`, etc.
- `android.view.ViewGroup` = Layout for views it contains, subclasses include
  - `android.widget.LinearLayout`
  - `android.widget.AbsoluteLayout`
  - `android.widget.TableLayout`
  - `android.widget.RelativeLayout`
  - `android.widget.FrameLayout`
  - `android.widget.ScrollLayout`
LinearLayout (<LinearLayout> or android.widget.LinearLayout)

- arranges by single column or row.
- child views can be arranged vertically or horizontally. ```xml
<?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:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"/>
</LinearLayout>
```
Hello World, HelloWorldAndroid! Lynne is here
Linear Layout Example

```xml
<?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" >

    <Button
        android:id="@+id/btn_webbrowser"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Web Browser"
        android:onClick="onClickWebBrowser" />

    <Button
        android:id="@+id/btn_makecalls"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Make Calls"
        android:onClick="onClickMakeCalls" />

    <Button
        android:id="@+id/btn_showMap"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Show Map"
        android:onClick="onClickShowMap" />

    <Button
        android:id="@+id/btn_launchMyBrowser"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Launch My Browser"
        android:onClick="onClickLaunchMyBrowser" />

</LinearLayout>
```
LinearLayout attributes

• You can set either in XML or with set*() methods by programing.

• i.e. Xml

  android:orientation="vertical"

• code (ll is LinearLayout instance)

  ll.setOrientation(VERTICAL);
Each View or ViewGroup can have its own set of attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layout_width</td>
<td>specifies width of View or ViewGroup</td>
</tr>
<tr>
<td>layout_height</td>
<td>specifies height</td>
</tr>
<tr>
<td>layout_marginTop</td>
<td>extra space on top</td>
</tr>
<tr>
<td>layout_marginBottom</td>
<td>extra space on bottom side</td>
</tr>
<tr>
<td>layout_marginLeft</td>
<td>extra space on left side</td>
</tr>
<tr>
<td>layout_marginRight</td>
<td>extra space on right side</td>
</tr>
<tr>
<td>layout_gravity</td>
<td>how child views are positioned</td>
</tr>
<tr>
<td>layout_weight</td>
<td>how much extra space in layout should be allocated to View (only when in LinearLayout or TableView)</td>
</tr>
<tr>
<td>layout_x</td>
<td>x-coordinate</td>
</tr>
<tr>
<td>layout_y</td>
<td>y-coordinate</td>
</tr>
</tbody>
</table>