Android User Interface

part1

In this chapter

1. Draw the screen using *views* and*ViewGroups.*

2. Find out how to create user interfaces (UIs) in Android and how users interact with the UIs.
3. Discover how to handle changes in screen orientation on your Android devices

Creating and laying out views

- Graphically within Android Studio
- XML Files
- Programmatically

- An activity contains views and ViewGroups.
- A view is a widget that has an appearance on screen.
  - Ex) views are buttons, labels, and text boxes.
  - A view derives from the base class android.view.View.
- One or more views can be grouped into a ViewGroup.
  - Ex) ViewGroups include RadioGroup and ScrollView.
  - A ViewGroup derives from the base class android.view.ViewGroup.
  - Another type of ViewGroup is a Layout.
- Layout is used to group and arrange views visually on the screen.

Common Layout Classes

- The Layouts available to you in Android are as follows:

1. LinearLayout - horizontal or vertical row
2. RelativeLayout - child views relative to each other
3. TableLayout - rows and columns
4. FrameLayout - shows one child of a stack of children
5. GridView - 2D scrollable grid

FrameLayout

- The FrameLayout is the most basic of the Android layouts.
- FrameLayouts are built to hold one view.
- As with all things related to development, there is no hard rule that FrameLayouts can't be used to hold multiple views.

LinearLayout (Horizontal) and LinearLayout (Vertical)

- The LinearLayout arranges views in a single column or a single row.
- Child views can be arranged either horizontally or vertically

```
<?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>
```

- Each view and ViewGroup has a set of common attributes

ATTRIBUTE	DESCRIPTION
layout_width	Specifies the width of the view or ViewGroup
layout_height	Specifies the height of the view or ViewGroup
layout_marginTop	Specifies extra space on the top side of the view or ViewGroup
layout_marginBottom	Specifies extra space on the bottom side of the view or ViewGroup
layout_marginLeft	Specifies extra space on the left side of the view or ViewGroup
layout_marginRight	Specifies extra space on the right side of the view or ViewGroup
layout_gravity	Specifies how child views are positioned
layout_weight	Specifies how much of the extra space in the layout should be allocated to the view
layout_x	Specifies the x-coordinate of the view or ViewGroup
layout_y	Specifies the y-coordinate of the view or ViewGroup


9:42
My Application
Hello World!
CLICK ME


9:46
My Application
Hello World!
CLICK ME

- width of both the TextView and Button views to an absolute value. the width for the TextView is set to 100 density-independent pixels wide, and the Button to 160 density-independent pixels wide
- it is important to understand how Android recognizes screens of varying sizes and densities.

- two (AVDs) with different shows an AVD with 1080 × 1920 resolution and LCD density of 480.
- shows another AVD with 768 × 1280 resolution and LCD density of 320.

- Orientation layout is horizontal, so if you omit the android:orientation attribute

```
<?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="wrap_content"
     android:layout_height="wrap_content"
     android:text="Text 1"
     android:id="@+id/textView" />

<TextView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Text 2"
     android:id="@+id/textView2" />

<TextView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Text 3"
     android:id="@+id/textView3" />

<LinearLayout
     android:orientation="horizontal"
     android:layout_width="match_parent"
     android:layout_height="wrap_content">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button 1"
            android:id="@+id/button" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button 2"
            android:id="@+id/button2" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button 3"
            android:id="@+id/button3" />
    </LinearLayout>
</LinearLayout>
```

TableLayout

- The TableLayout Layout groups views into rows and columns.

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent" >
    <TableRow>
        <TextView
            android:text="User Name:"
            android:width ="120dp"
            />
        <EditText
            android:id="@+id/txtUserName"
            android:width="200dp" />
    </TableRow>
    <TableRow>
        <TextView
            android:text="Password:"
            />
        <EditText
            android:id="@+id/txtPassword"
            android:inputType="textPassword"
            />
    </TableRow>
    <TableRow>
        <TextView />
        <CheckBox android:id="@+id/chkRememberPassword"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Remember Password"
            />
    </TableRow>
    <TableRow>
        <Button
            android:id="@+id/buttonSignIn"
            android:text="Log In" />
    </TableRow>
</TableLayout>
```

RelativeLayout

- The RelativeLayout layout enables you to specify how child views are positioned relative to each otherThese attributes are as follows:
- layout_alignParentTop
- layout_alignParentStart
- layout_alignStart
- layout_alignEnd
- layout_below
- layout_centerHorizontal

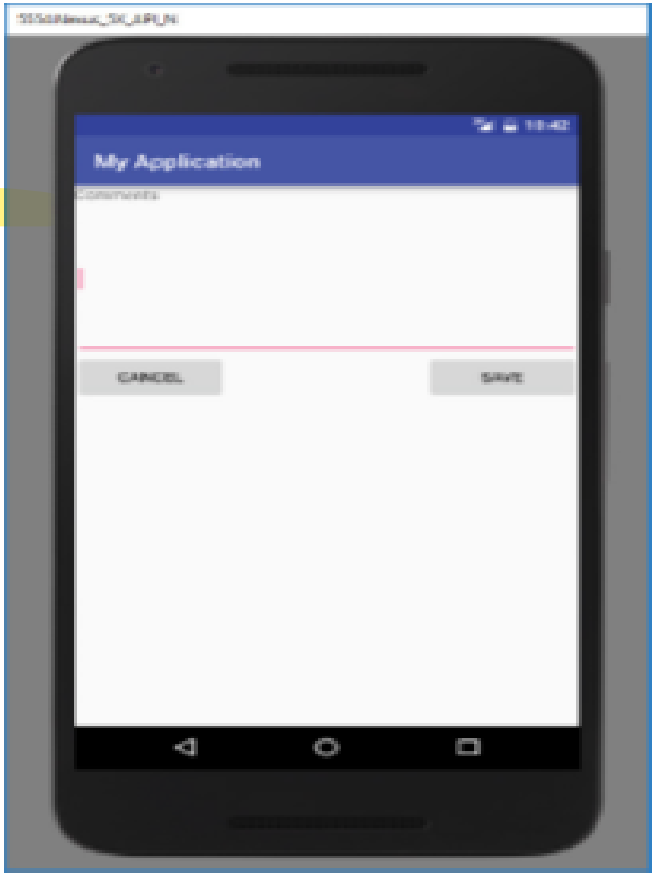
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:id="@+id/RLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android" >

    <TextView
        android:id="@+id/lblComments"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Comments"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true" />


    <EditText
        android:id="@+id/txtComments"
        android:layout_width="fill_parent"
        android:layout_height="170dp"
        android:textSize="18sp"
        android:layout_alignStart="@+id/lblComments"
        android:layout_below="@+id/lblComments"
        android:layout_centerHorizontal="true" />

    <Button
        android:id="@+id/btnSave"
        android:layout_width="125dp"
        android:layout_height="wrap_content"
        android:text="Save"
        android:layout_below="@+id/txtComments"
        android:layout_alignEnd="@+id/txtComments" />

    <Button
        android:id="@+id/btnCancel"
        android:layout_width="124dp"
        android:layout_height="wrap_content"
        android:text="Cancel"
        android:layout_below="@+id/txtComments"
        android:layout_alignStart="@+id/txtComments" />
</RelativeLayout>
```



Adapting to Display Orientation

- One of the key features of modern smartphones is their ability to switch screen orientation, and
- Android is no exception.
- Android supports two screen orientations: *portrait* and *landscape*.
- By default, when you change the display orientation of your Android device, the current activity automatically redraws its content in the new orientation

- In general, you can employ two techniques to handle changes in screen orientation:
- **Anchoring**—The easiest way is to “anchor” your views to the four edges of the screen.
  - When the screen orientation changes, the views can anchor neatly to the edges.
- **Resizing and repositioning**—Whereas anchoring and centralizing are simple techniques to ensure that views can handle changes in screen orientation, the ultimate technique is resizing each and every view according to the current screen orientation.

- **Anchoring Views**
  - Anchoring can be easily achieved by using RelativeLayout.

➤➤ layout_alignParentStart—Aligns the view to the left of the parent view

➤➤ layout_alignParentEnd—Aligns the view to the right of the parent view

➤➤ layout_alignParentTop—Aligns the view to the top of the parent view

➤➤ layout_alignParentBottom—Aligns the view to the bottom of the parent view

➤➤ layout_centerVertical—Centers the view vertically within its parent view

➤➤ layout_centerHorizontal—Centers the view horizontally within its parent view

```
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true" /
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Left"
        android:layout_alignParentStart="true"
        android:layout_alignParentBottom="true
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right"
        android:layout_alignParentEnd="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button5"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Detecting Orientation Changes

- use the getResources() method .

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

    if(getResources().getConfiguration().orientation == Configuration
.ORIENTATION_LANDSCAPE){
        Log.d("StateInfo", "Landscape");
    }else if(getResources().getConfiguration().orientation == Configuration
.ORIENTATION_PORTRAIT){
        Log.d("StateInfo", "Portrait");
    }
}
```

Controlling the Orientation of the Activity

- you might want to ensure that your application is displayed in only a certain orientation.
- For example, you may be writing a game that should be viewed only in landscape mode.

setRequestOrientation() method

```
import android.content.pm.ActivityInfo;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

    //---change to landscape mode---
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
}
```

use the activityInfo.SCREEN_ORIENTATION_PORTRAIT constant

Creating the User Interface Programmatically

- all the UIs you have seen in this chapter are created using XML.
- using XML you can also create the UI using code.
- This approach is useful if your UI needs to be dynamically generated during runtime.

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.LinearLayoutCompat;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayoutCompat.LayoutParams params =
                new LinearLayoutCompat.LayoutParams(
                        LinearLayoutCompat.LayoutParams.WRAP_CONTENT,
                        LinearLayoutCompat.LayoutParams.WRAP_CONTENT);
        //---create a layout---
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        //---create a textview---
        TextView tv = new TextView(this);
        tv.setText("This is a TextView");
        tv.setLayoutParams(params);
        //---create a button---
        Button btn = new Button(this);
        btn.setText("This is a Button");
        btn.setLayoutParams(params);

        //---adds the textview---
        layout.addView(tv);
        //---adds the button---
        layout.addView(btn);
        //---create a layout param for the layout---
        LinearLayoutCompat.LayoutParams layoutParam =
                new LinearLayoutCompat.LayoutParams(
                        LinearLayoutCompat.LayoutParams.WRAP_CONTENT,
                        LinearLayoutCompat.LayoutParams.WRAP_CONTENT );
        this.addContentView(layout, layoutParam);

    }
}
```