ITSMAP F13 Lesson 4 Part 1

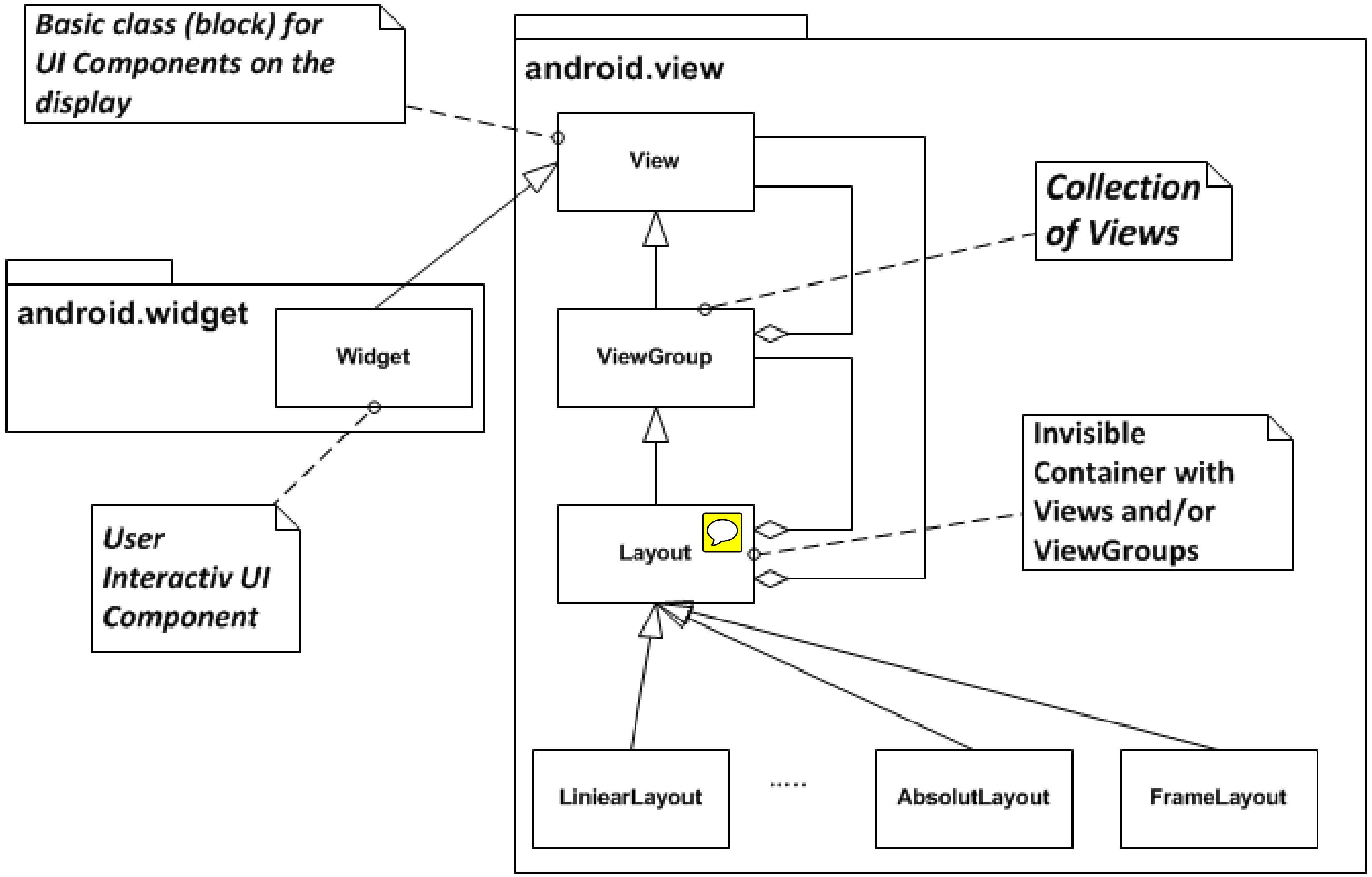
Androids User Interface: Views

Jesper Rosholm Tørresø

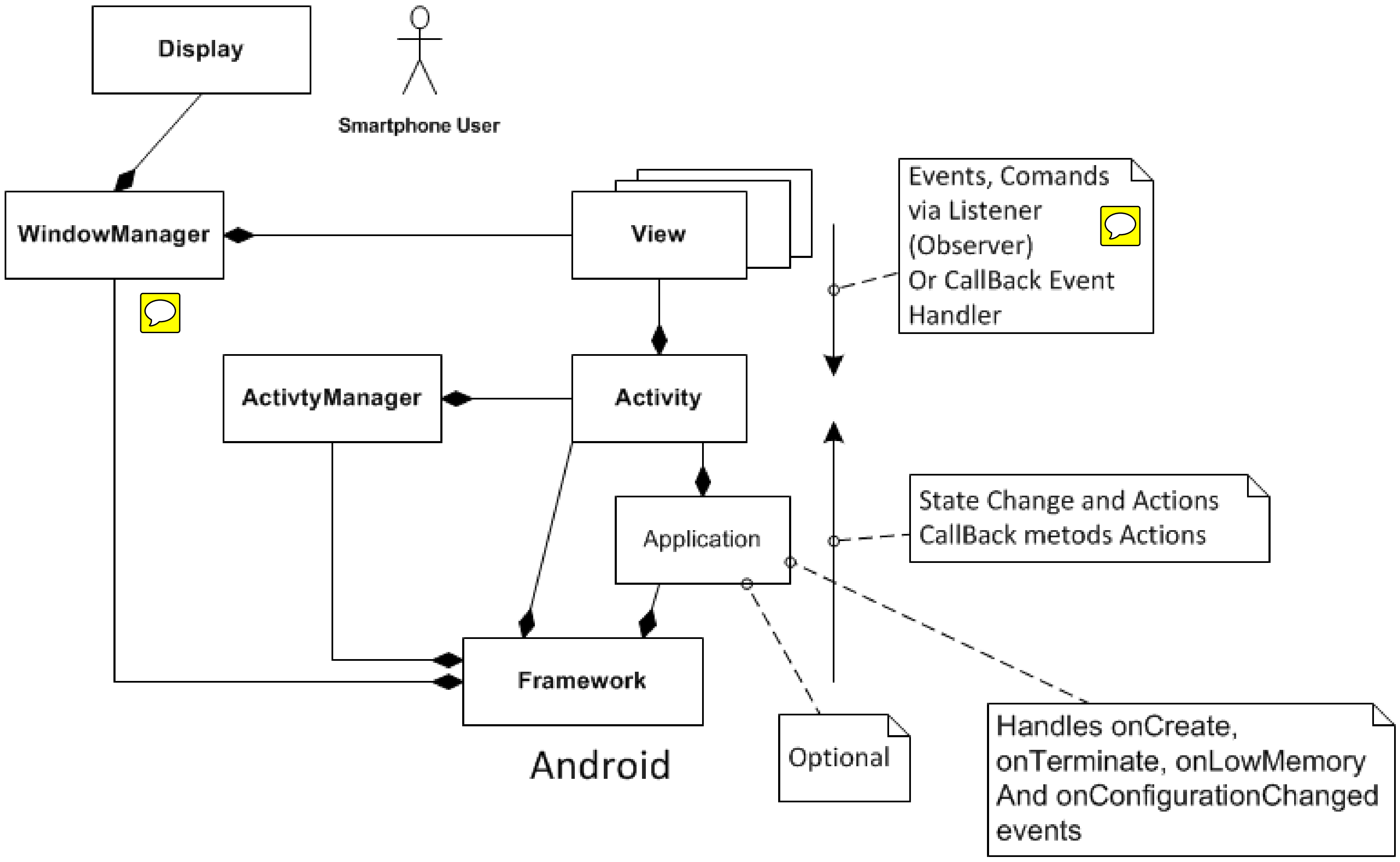
Subjects
Part 1

1. Android Applications and User Interface
2. Overall architecture
3. View organization/Hierarchy
4. View Layout
5. A little more about resources and views
6. Event handling
7. Exercise 1
8. In part 2
   1. Views with Fragments
   2. Adpaters

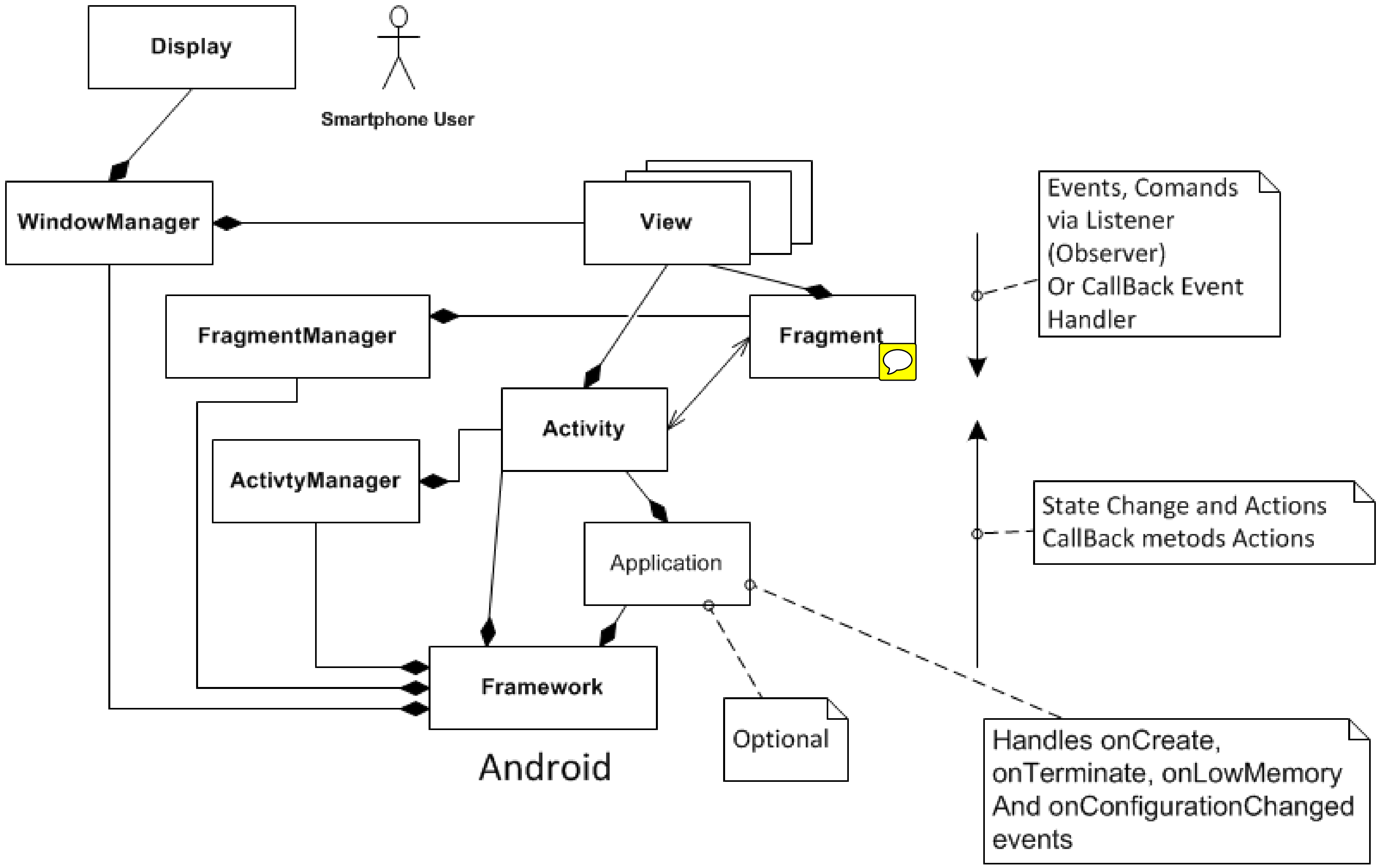
Android Java View Class Hierarchy


Basic class (block) for UI Components on the display
android.view
View
Collection of Views
android.widget
Widget
ViewGroup
Invisible Container with Views and/or ViewGroups
User Interactiv UI Component
Layout
LiniearLayout
.....
AbsolutLayout
FrameLayout

App Overall Architecture:
User Interface by View, no Fragments


Display
Smartphone User
WindowManager
View
Events, Comands via Listener (Observer) Or CallBack Event Handler
ActivtyManager
Activity
Application
State Change and Actions CallBack metods Actions
Framework
Android
Optional
Handles onCreate, onTerminate, onLowMemory And onConfigurationChanged events

App Overall Architecture:
User Interface by View, with Fragments (part 2)

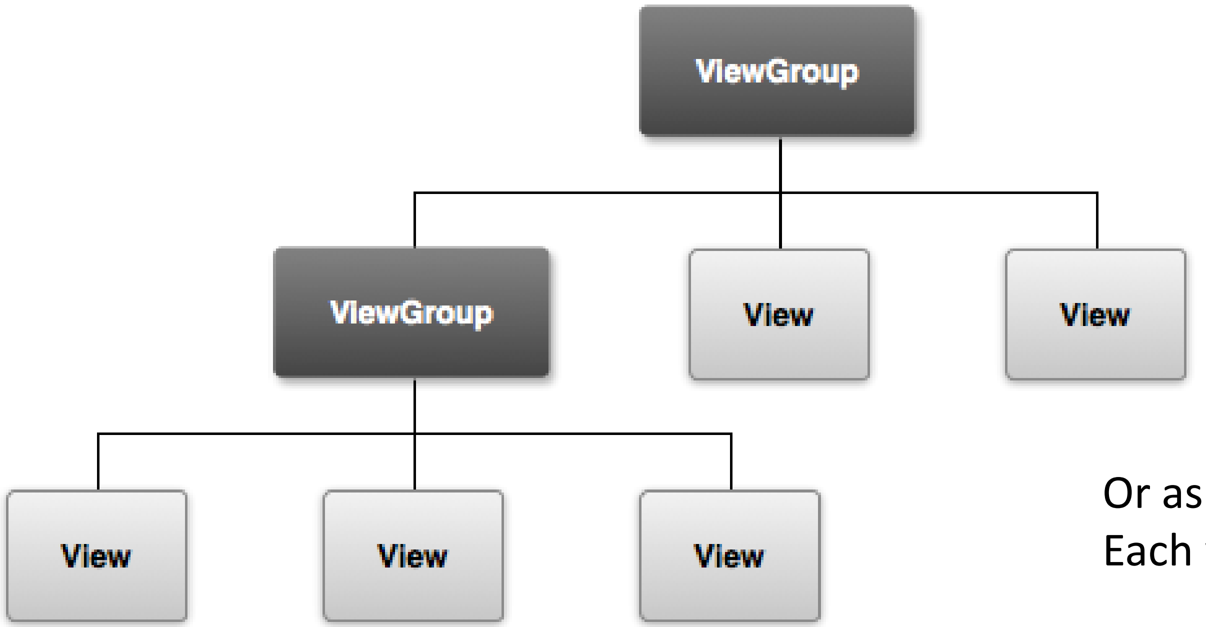

Display
Smartphone User
WindowManager
View
Fragment
FragmentManager
Activity
ActivtyManager
Application
Framework
Android
Optional
Events, Comands via Listener (Observer) Or CallBack Event Handler
State Change and Actions CallBack metods Actions
Handles onCreate, onTerminate, onLowMemory And onConfigurationChanged events

View Hierarchy

The organization of User Layouts

As a Android App programmer

Always look at a “User View “as a **Hierarchy**



Or as a **container of Views**
Each with its own identifier

User Layouts are setup in XML or coded in the Activity

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/andro
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Enter Text Below"
  />
  <EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Text Goes Here!"
  />
</LinearLayout>
```

```
// ** Listing 4-4: Simple LinearLayout in code
LinearLayout ll = new LinearLayout(this);
ll.setOrientation(LinearLayout.VERTICAL);

TextView myTextView = new TextView(this);
EditText myEditText = new EditText(this);

myTextView.setText("Enter Text Below");
myEditText.setText("Text Goes Here!");

int lHeight = LinearLayout.LayoutParams.FILL_PARENT;
int lWidth = LinearLayout.LayoutParams.WRAP_CONTENT;

ll.addView(myTextView, new LinearLayout.LayoutParams(lHeight,
lWidth));
ll.addView(myEditText, new LinearLayout.LayoutParams(lHeight,
lWidth));
setContentView(ll);
```

Drawing of a “user view”

The View to an Activity

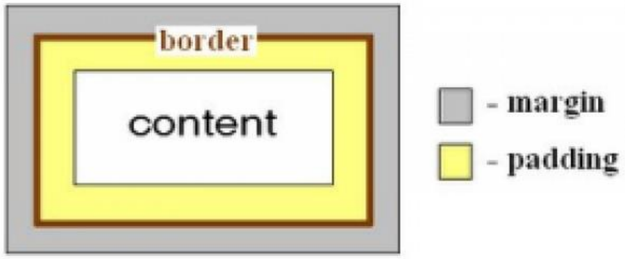
- Views are inflated i.e. layout are drawn, in a hierarchical control structure by the Framework, in the order of the specific view hierarchy.
- The Activity request the Framework to inflate View

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

- The root node, that’s a ViewGroup, draw itself and then calling its children to draw themselves and so on.

http://developer.android.com/guide/topics/ui/declaring-layout.html

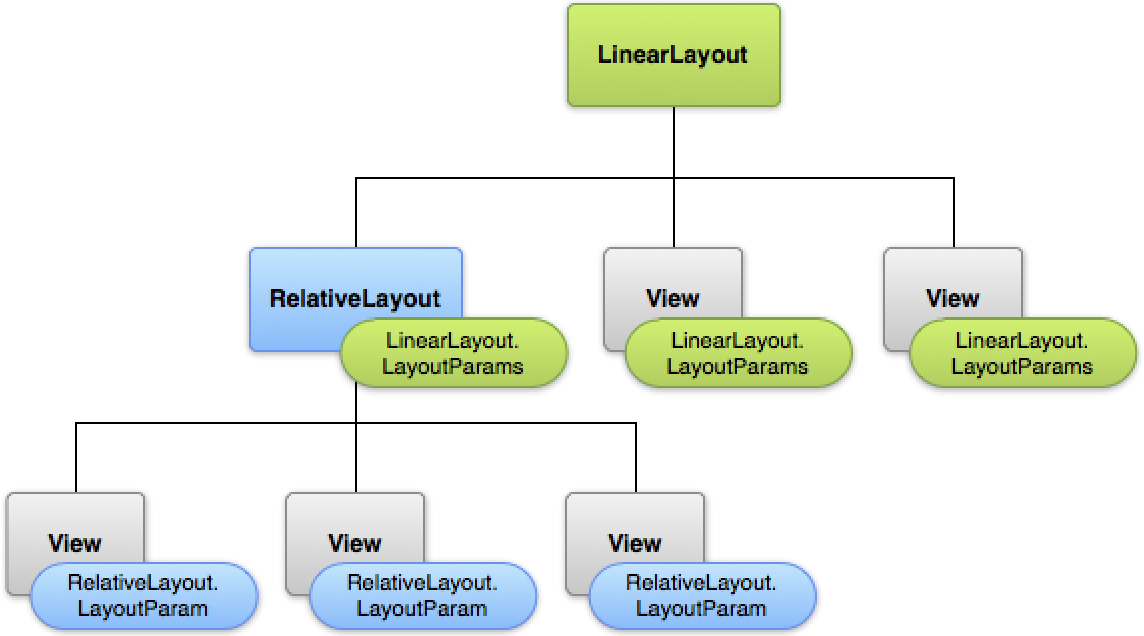
Drawing of a View
Size and Positioning



- **Size: Dimensions** For each dimension in a View, the View can specify
  - an exact number
  - *FILL_PARENT / MATCH_PARENT*, as big as its parent **(minus padding)**
  - *WRAP_CONTENT*, big enough to enclose its content **(plus padding).**
- **Positioning** are handled by ViewGroups, which are specific Layouts
  - Layouts uses own **LayoutParams** subclasses. For example, **RelativeLayout** has its own subclass of LayoutParams, which includes the ability to center child Views horizontally and vertically.

http://developer.android.com/guide/topics/ui/how-android-draws.html

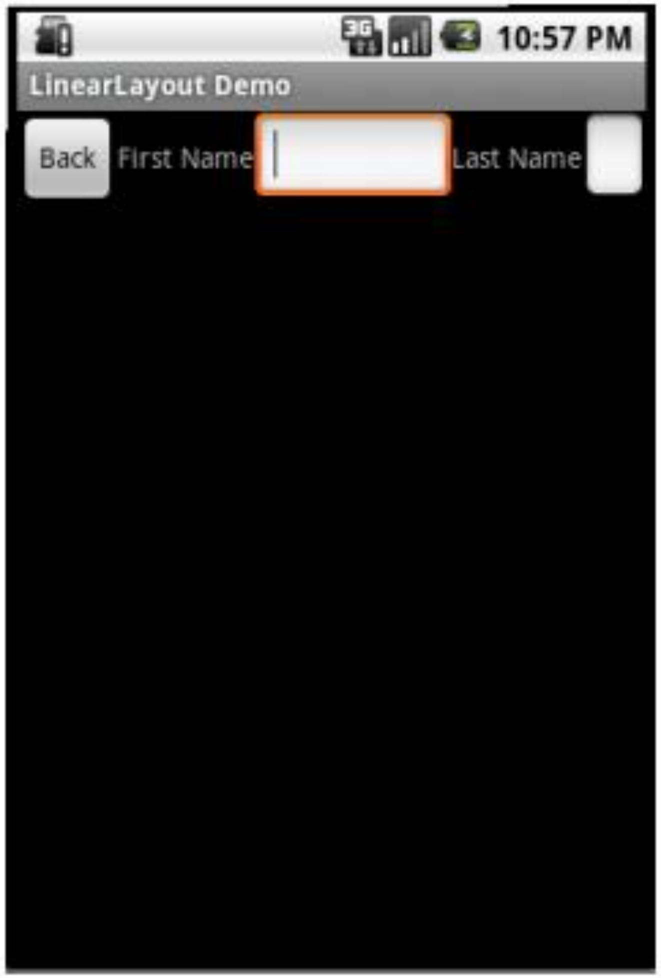
Layout parameters

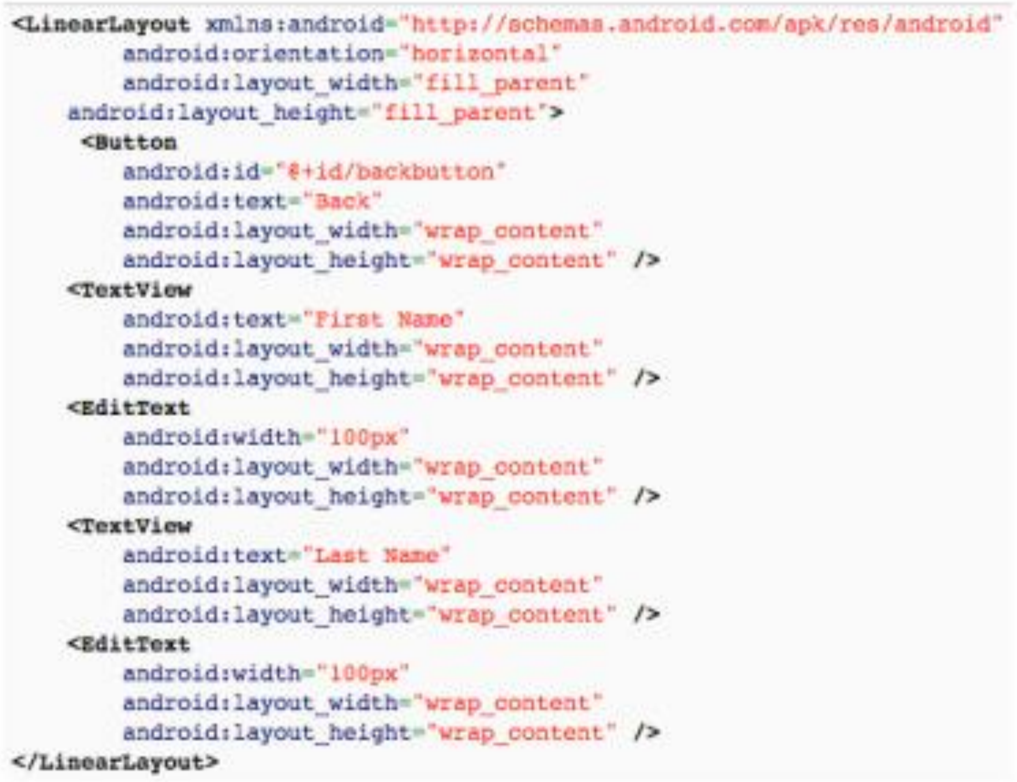

LinearLayout
RelativeLayout
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam

http://developer.android.com/guide/topics/ui/declaring-layout.html

http://developer.android.com/tools/debugging/debugging-ui.html Debug UI

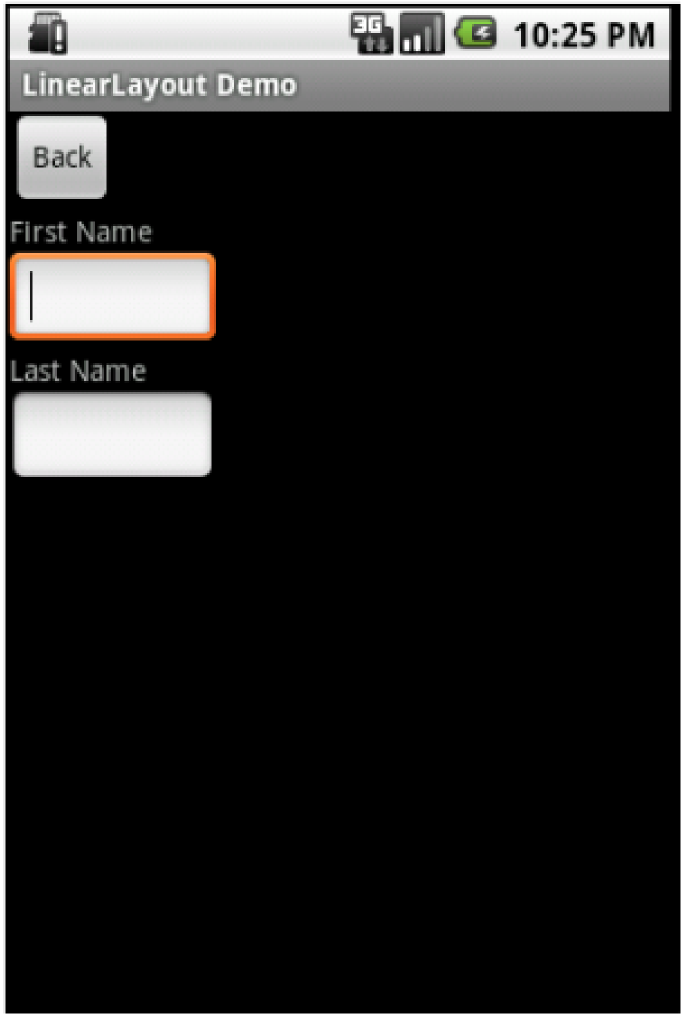
Layouts Linear





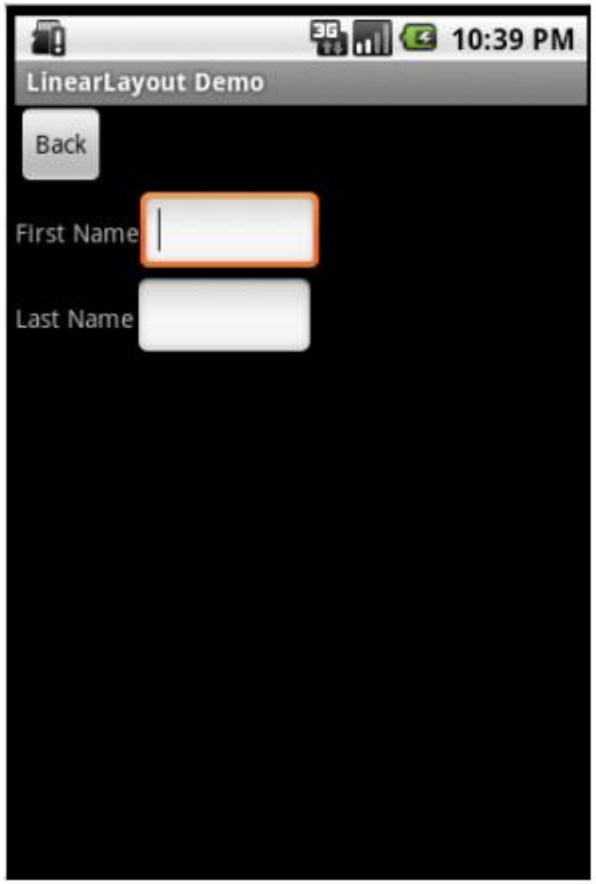
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
    android:layout_height="fill_parent">
     <Button
        android:id="@+id/backbutton"
        android:text="Back"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:text="First Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <EditText
        android:width="100px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:text="Last Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <EditText
        android:width="100px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

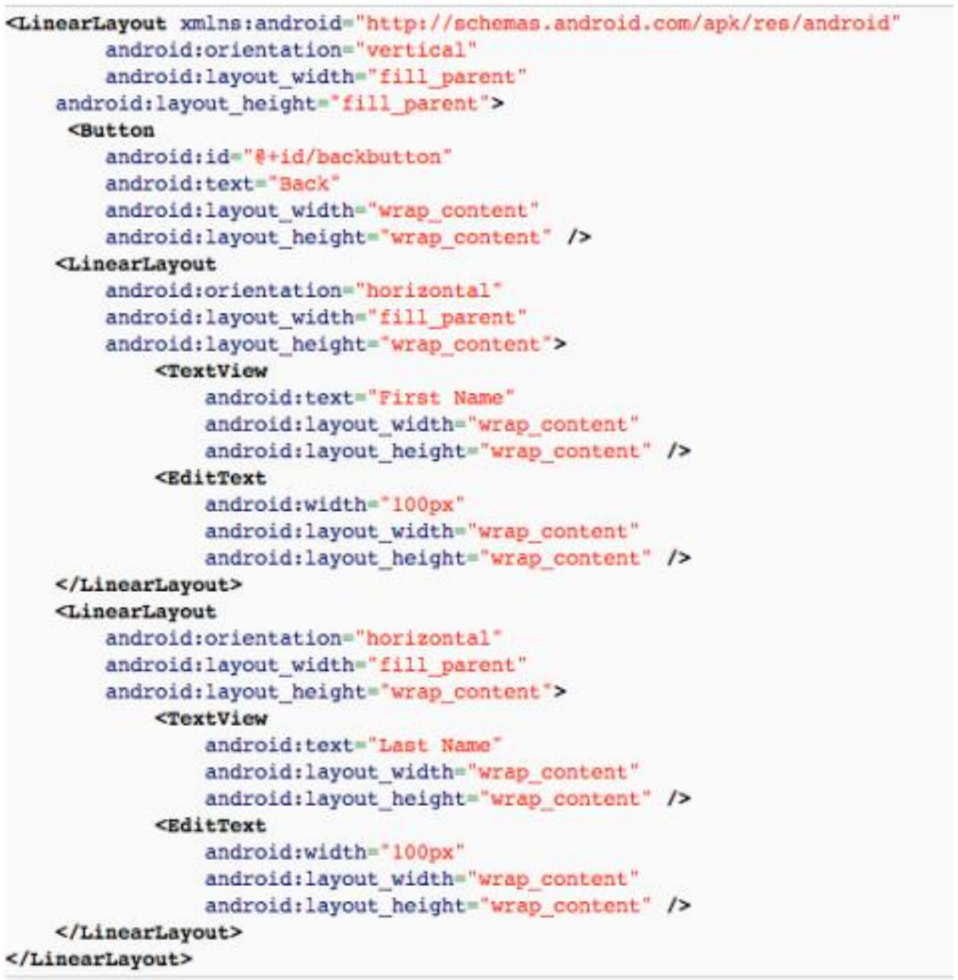
Layouts Linear Vertical



`android:orientation="vertical"`

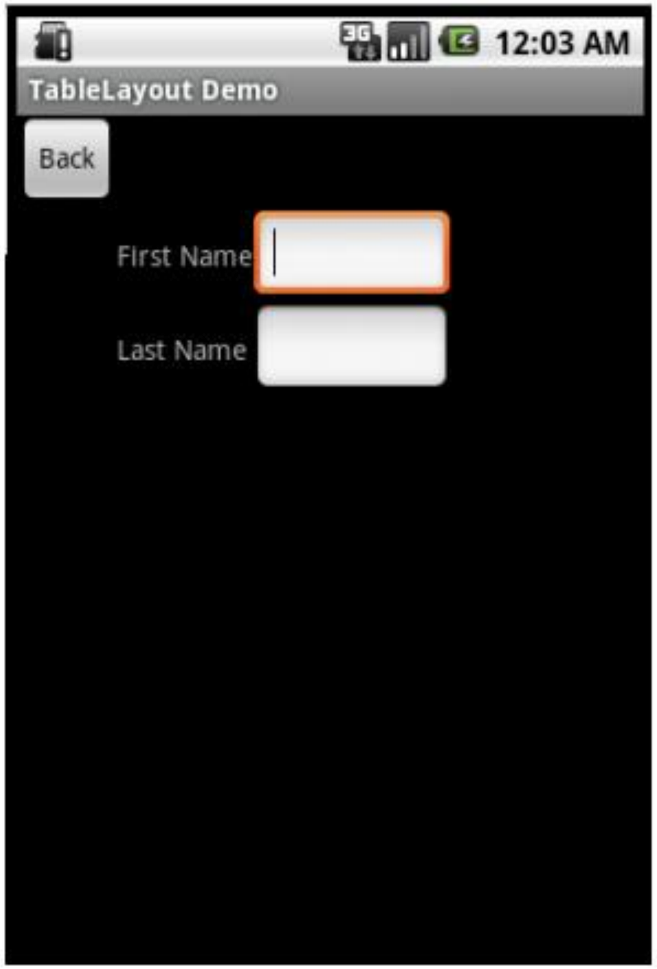
Layouts Nested

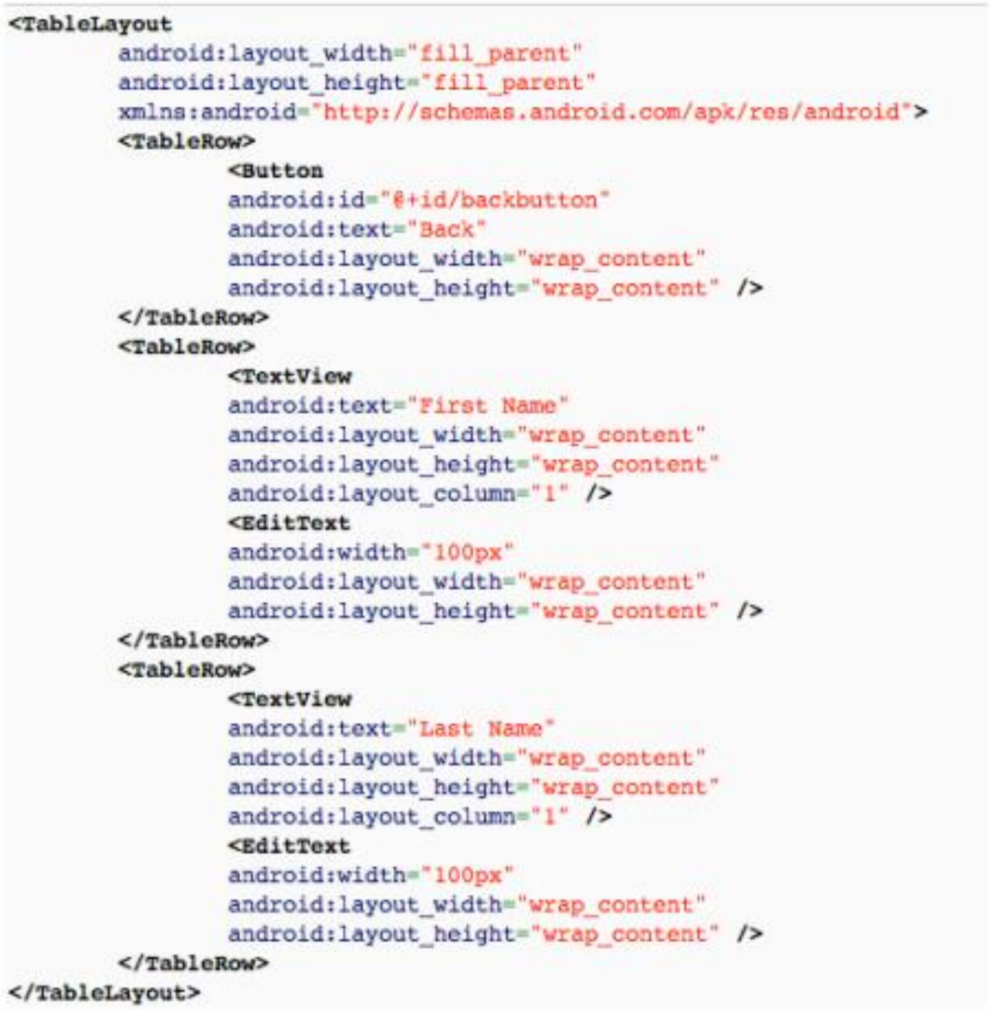




```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
    android:layout_height="fill_parent">
     <Button
        android:id="@+id/backbutton"
        android:text="Back"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
            <TextView
                android:text="First Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
            <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
    </LinearLayout>
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
            <TextView
                android:text="Last Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
            <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
    </LinearLayout>
</LinearLayout>
```

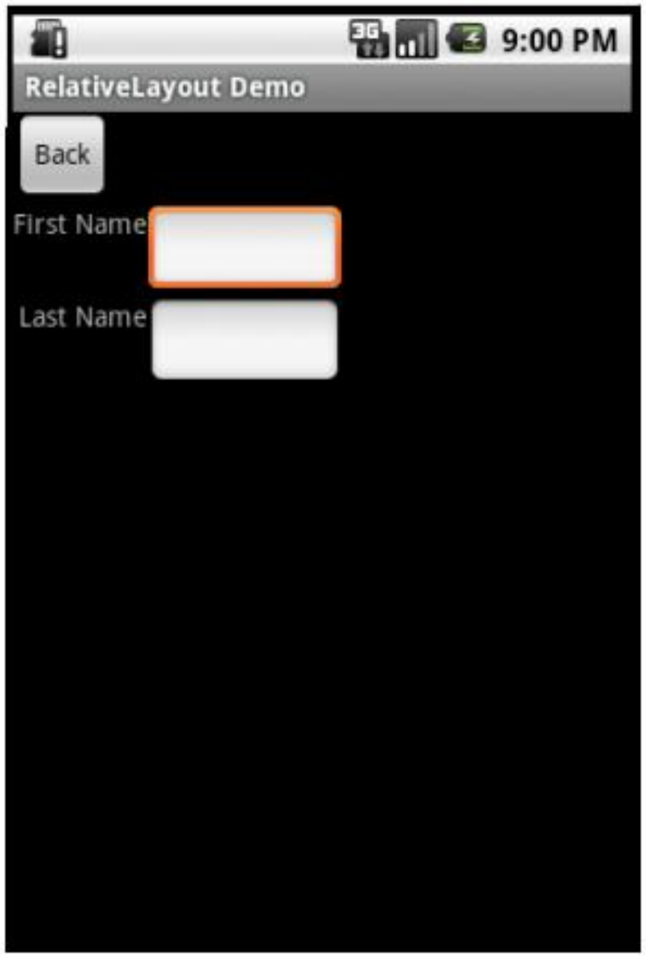
Layouts Table

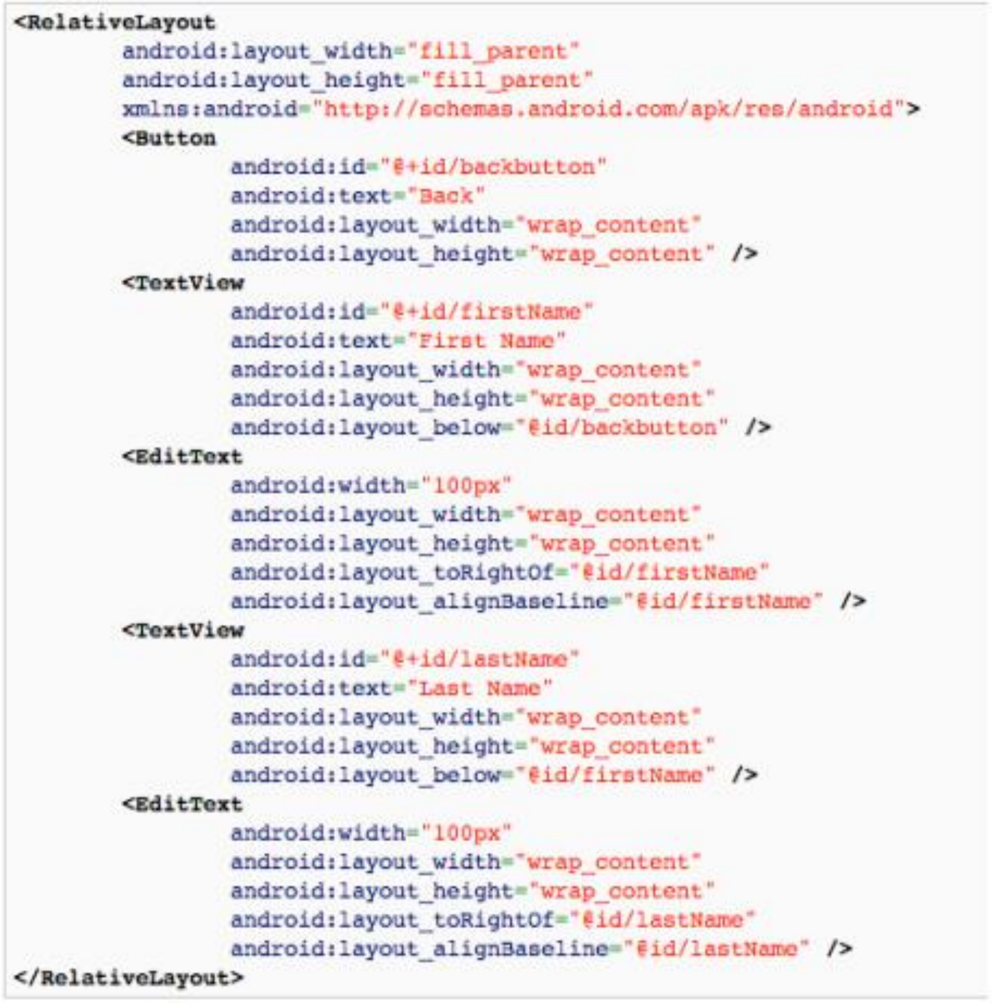




```
<TableLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        xmlns:android="http://schemas.android.com/apk/res/android">
        <TableRow>
                <Button
                android:id="@+id/backbutton"
                android:text="Back"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
        </TableRow>
        <TableRow>
                <TextView
                android:text="First Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="1" />
                <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
        </TableRow>
        <TableRow>
                <TextView
                android:text="Last Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="1" />
                <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
        </TableRow>
</TableLayout>
```

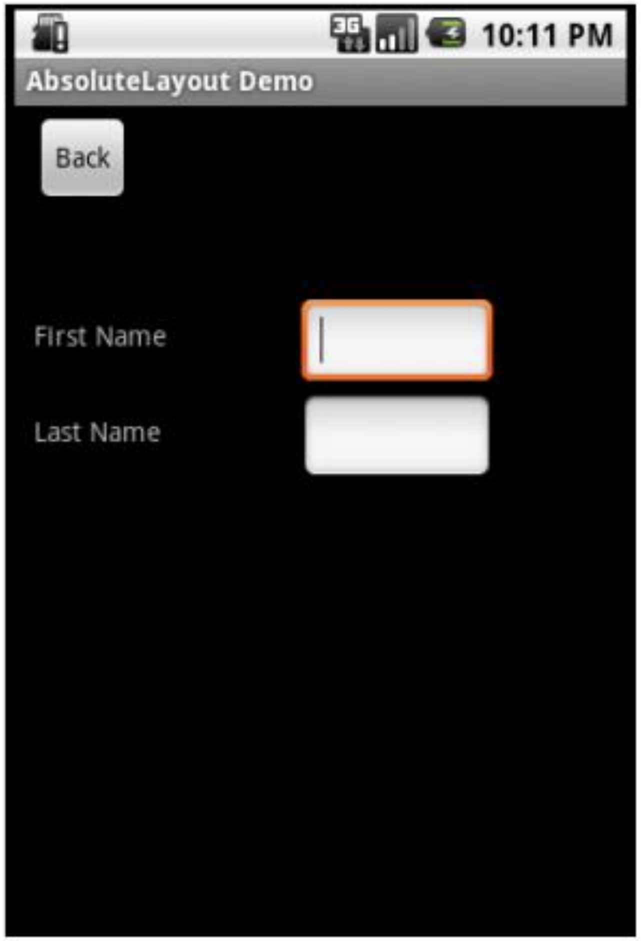
Layouts Relative

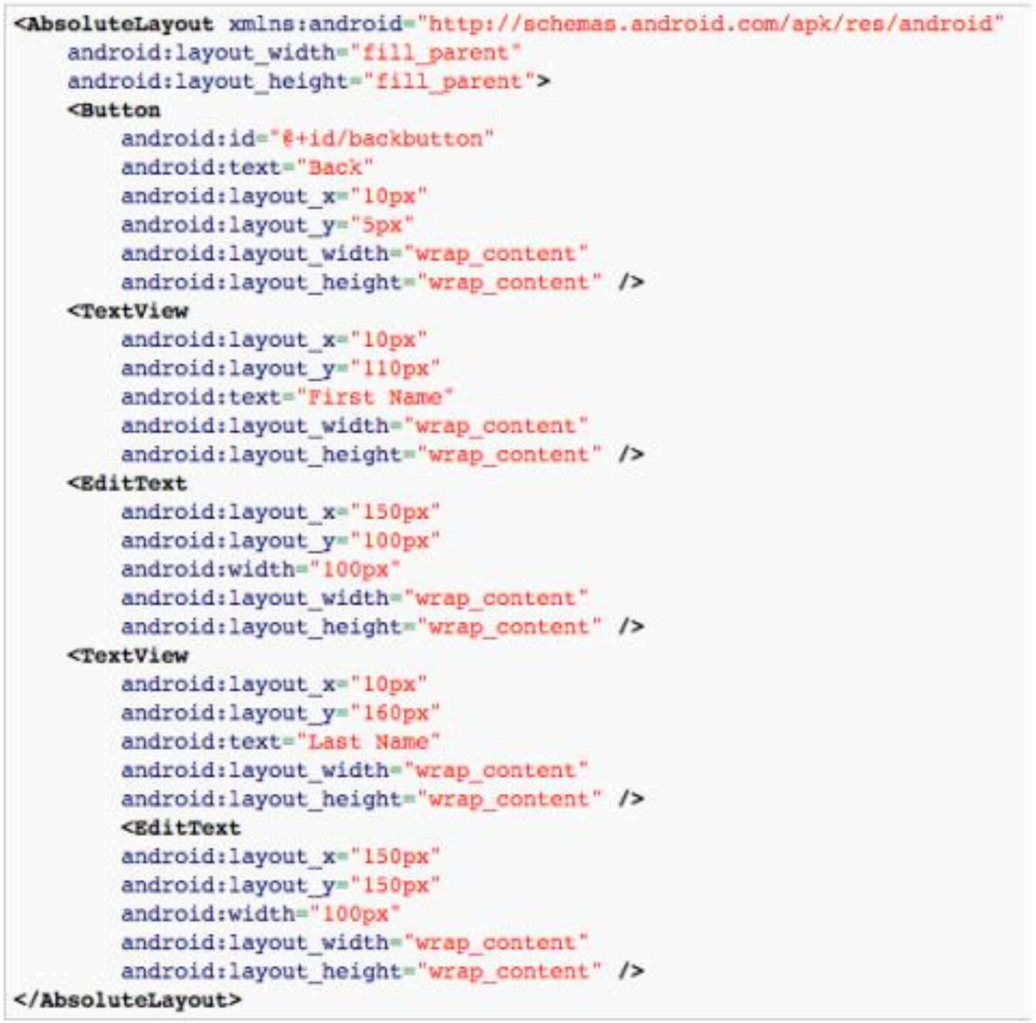




```
<RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        xmlns:android="http://schemas.android.com/apk/res/android">
        <Button
                android:id="@+id/backbutton"
                android:text="Back"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
        <TextView
                android:id="@+id/firstName"
                android:text="First Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@id/backbutton" />
        <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_toRightOf="@id/firstName"
                android:layout_alignBaseline="@id/firstName" />
        <TextView
                android:id="@+id/lastName"
                android:text="Last Name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@id/firstName" />
        <EditText
                android:width="100px"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_toRightOf="@id/lastName"
                android:layout_alignBaseline="@id/lastName" />
</RelativeLayout>
```

Layouts, some are deprecated





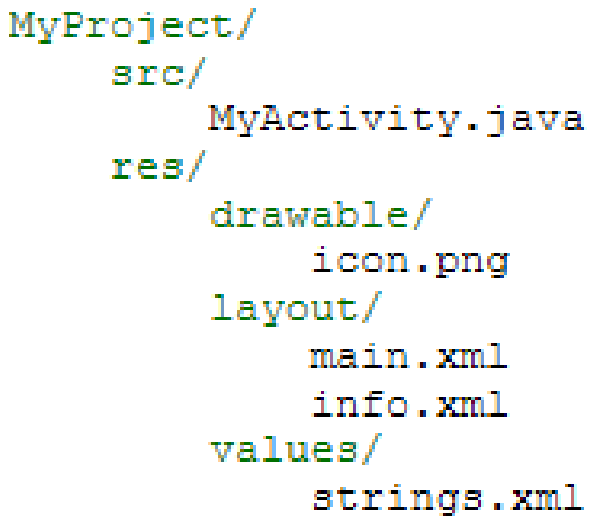
```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/backbutton"
        android:text="Back"
        android:layout_x="10px"
        android:layout_y="5px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:layout_x="10px"
        android:layout_y="110px"
        android:text="First Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <EditText
        android:layout_x="150px"
        android:layout_y="100px"
        android:width="100px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:layout_x="10px"
        android:layout_y="160px"
        android:text="Last Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
        <EditText
        android:layout_x="150px"
        android:layout_y="150px"
        android:width="100px"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</AbsoluteLayout>
```

What to take care of in the Graphical UI?

- View http://developer.android.com/reference/android/view/View.html
- Layout http://developer.android.com/guide/topics/ui/layout-objects.html comming from
  - ViewGroup http://developer.android.com/reference/android/view/ViewGroup.html
- Widget Package http://developer.android.com/reference/android/widget/package-summary.html
- Menu http://developer.android.com/guide/topics/ui/menus.html (Like Menu button)
- UI Input Events
  - Define an event listener and register it with the View
  - Override an existing callback method for the View (Custom Views)
  - Menu Events
- And of course “The View’ appearance”

  http://developer.android.com/guide/topics/ui/index.html

Providing resource



```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            icon.png
        layout/
            main.xml
            info.xml
        values/
            strings.xml
```

/animator
/anim
/color
/drawable
/layout
/menu
/raw
/values
/xml

http://developer.android.com/guide/topics/resources/providing-resources.html

Layout folders

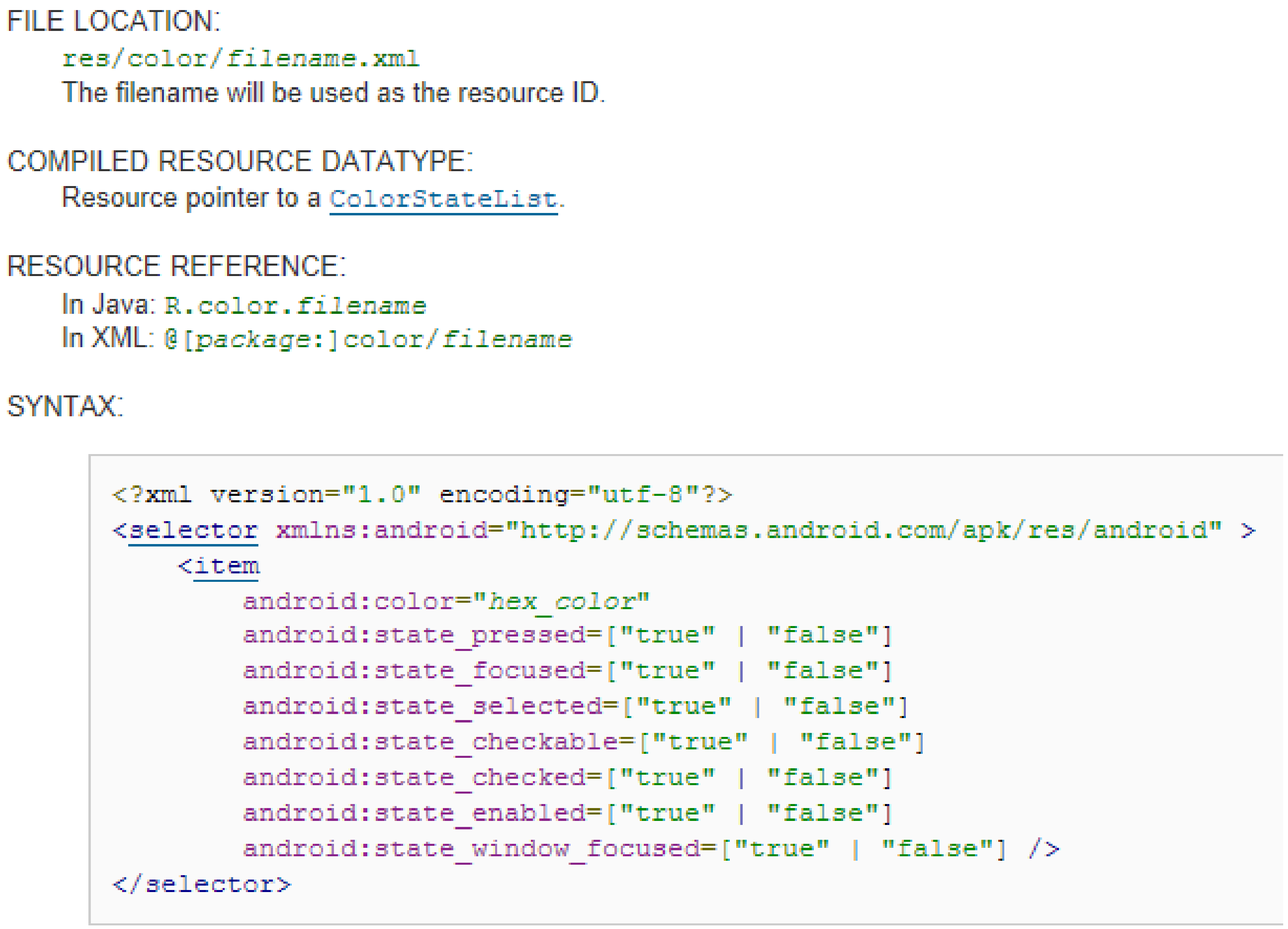
- **res/layout**/my_layout.xml // layout for normal screen size ("default")
- **res/layout-small**/my_layout.xml // layout for small screen size
- **res/layout-large**/my_layout.xml // layout for large screen size
- **res/layout-xlarge**/my_layout.xml // layout for extra large screen size
- **res/layout-xlarge-land**/my_layout.xml // layout for extra large in landscape orientation
- res/drawable-mdpi/my_icon.png // bitmap for medium density
- res/drawable-hdpi/my_icon.png // bitmap for high density
- res/drawable-xhdpi/my_icon.png // bitmap for extra high density

http://developer.android.com/guide/practices/screens_support.html

External Resources

- Remember that you write XML code instead of Java code.
- And by referring to a certain XML file the file name is the resources ID

Example 1 (selector … ex. to a button)



FILE LOCATION:

`res/color/filename.xml`
The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

Resource pointer to a `ColorStateList`.

RESOURCE REFERENCE:

In Java: `R.color.filename`
In XML: `@[package:]color/filename`

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:color="hex_color"
        android:state_pressed=["true" | "false"]
        android:state_focused=["true" | "false"]
        android:state_selected=["true" | "false"]
        android:state_checkable=["true" | "false"]
        android:state_checked=["true" | "false"]
        android:state_enabled=["true" | "false"]
        android:state_window_focused=["true" | "false"] />
</selector>
```

Resource in use (button animation)

With use of ***recommended folder!***

res/color/ffdrinkbtn. xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res/android">

 <item android:drawable="@drawable/drinklilleinv"
                              android:state_pressed="true" />
 <item android:drawable="@drawable/drinklillesh"
                              android:state_focused="true" />
 <item android:drawable="@drawable/drinklille1" />

</selector>
```

From a view

```
<Button      android:id="@+id/drawnbeerbuttoncalc"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:onClick="drawnBeerCount"
             android:text="Button"
             android:background="@color/ffdrinkbtn"
             />
</selector>
```

Resource in use (button animation)

With use of ***non recommended folder!!!***

res/drawable/**ffdrinkbtn.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res/android">

 <item android:drawable="@drawable/drinklilleinv"
                            android:state_pressed="true" />
 <item android:drawable="@drawable/drinklillesh"
                            android:state_focused="true" />
 <item android:drawable="@drawable/drinklille1" />

</selector>
```

From a view

```
<Button      android:id="@+id/drawnbeerbuttoncalc"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:onClick="drawnBeerCount"
             android:text="Button"
             android:background="@drawable/ffdrinkbtn"
             />
</selector>
```

Resource Types

(click on links for Android Dev Guide)

- Animation
- Color State List (Used in previous example)
- Drawable
- Layout
- Menu
- String
- Style
- More Types
  - Bool
  - Color
  - Dimension
  - ID
  - Integer
  - Integer Array
  - Typed Array (which you can use for an array of drawables).

Themes
in Android styles

EXAMPLE:

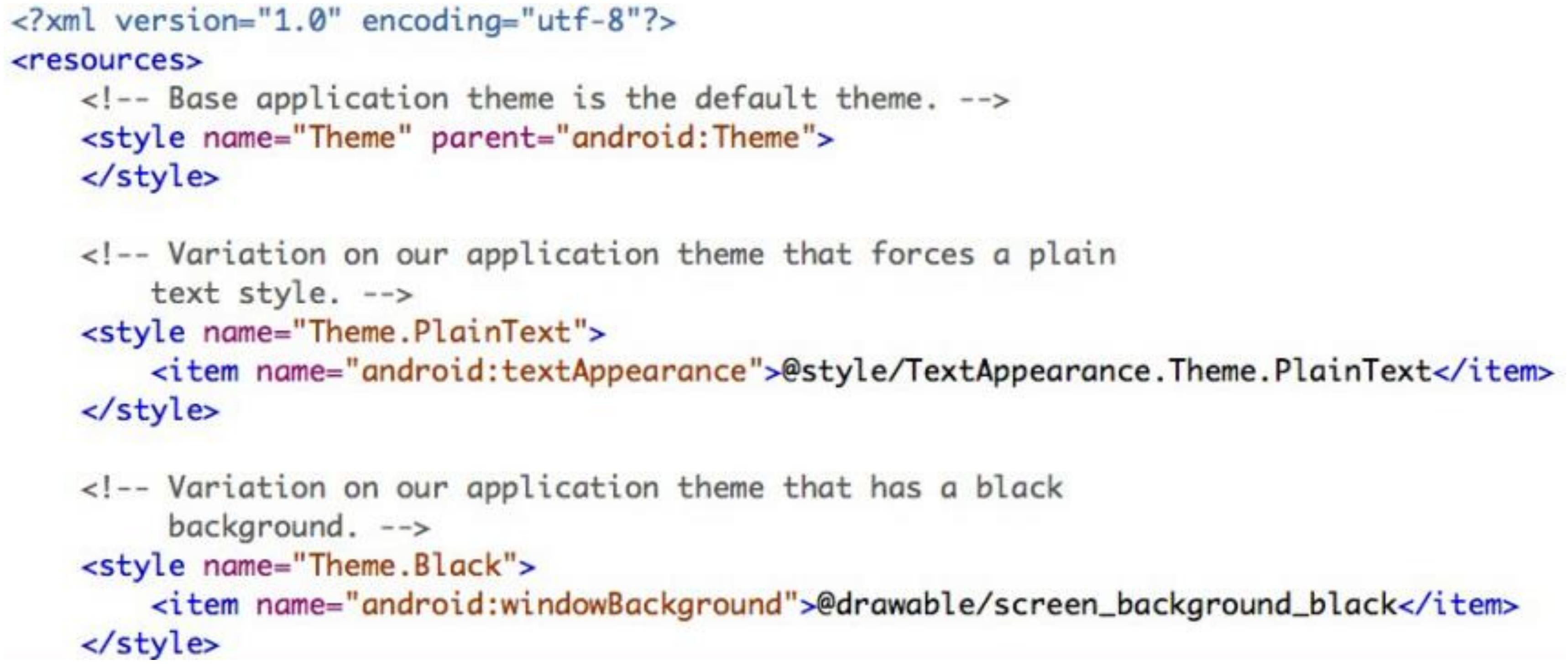
XML file for the style (saved in `res/values/`):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CustomText" parent="@style/Text">
        <item name="android:textSize">20sp</item>
        <item name="android:textColor">#008</item>
    </style>
</resources>
```

XML file that applies the style to a `TextView` (saved in `res/layout/`):

```
<?xml version="1.0" encoding="utf-8"?>
<EditText
    style="@style/CustomText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello, World!" />
```

Themes

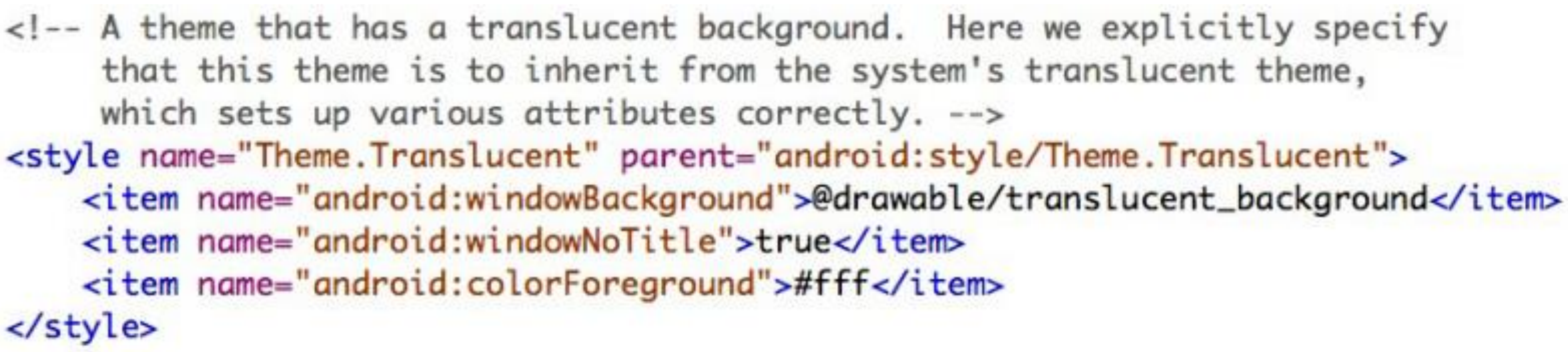


```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- Base application theme is the default theme. -->
    <style name="Theme" parent="android:Theme">
    </style>

    <!-- Variation on our application theme that forces a plain
        text style. -->
    <style name="Theme.PlainText">
        <item name="android:textAppearance">@style/TextAppearance.Theme.PlainText</item>
    </style>

    <!-- Variation on our application theme that has a black
         background. -->
    <style name="Theme.Black">
        <item name="android:windowBackground">@drawable/screen_background_black</item>
    </style>
```

Themes



```xml
<!-- A theme that has a translucent background.  Here we explicitly specify
     that this theme is to inherit from the system's translucent theme,
     which sets up various attributes correctly. -->
<style name="Theme.Translucent" parent="android:style/Theme.Translucent">
    <item name="android:windowBackground">@drawable/translucent_background</item>
    <item name="android:windowNoTitle">true</item>
    <item name="android:colorForeground">#fff</item>
</style>
```

What to do with Views

Once you have created a tree of views, there are typically a few types of common operations you may wish to perform:

- **Set properties:** for example setting the text of a TextView. The available properties and the methods that set them will vary among the different subclasses of views. **Note that properties that are known at build time can be set in the XML layout files.**
- **Set focus:** The framework will handled moving focus in response to user input. To force focus to a specific view, call requestFocus().
- **Set up listeners:** Views allow clients to set listeners that will be notified when something interesting happens to the view. For example, all views will let you set a listener to be notified when the view gains or loses focus. You can register such a listener using setOnFocusChangeListener(android.view.View.OnFocusChangeListener). **Other view subclasses offer more specialized listeners. For example, a Button exposes a listener to notify clients when the button is clicked.**
- **Set visibility:** You can hide or show views using setVisibility(int).

http://developer.android.com/reference/android/view/View.html

Views are clickable

- Gives that every Layout and Widget or other User Controls inherits the capability of being
  - `android:clickable="true" or "false"`
- And can publish the click event to a subscriber i.e. an Activity, this is what we look at now
- But the View itself may capture events from the Android framework for controlling its behavior according to the users input

Inherited View event handling

```
public class MyView extends View {
@Override
public boolean onKeyDown(int keyCode, KeyEvent keyEvent) {
  // Return true if the event was handled.
  return true;
}
@Override
public boolean onKeyUp(int keyCode, KeyEvent keyEvent) {
  // Return true if the event was handled.
  return true;
}
@Override
public boolean onTrackballEvent(MotionEvent event ) {
  // Get the type of action this event represents
  int actionPerformed = event.getAction();
  // Return true if the event was handled.
  return true;
}
@Override
public boolean onTouchEvent(MotionEvent event) {
  // Get the type of action this event represents
  int actionPerformed = event.getAction();
  // Return true if the event was handled.
  return true;
}
```

Event Listeners

/

- Inline Class Implementation
- Use “Implements” method
- Use a variable for a “listener method”

And a FW controlled way for clickable Views

- XML attribute android:onClick=*"click1”*

```
public void click1(View view) {// use this signature
// but1.setClickable(false);
setQuestionAnswer(question + "/1");
but1.setBackgroundColor(Color.GREEN);
this.onClick(view, "Knap 1");
}
```

http://tseng-blog.nge-web.net/blog/2009/02/14/implementing-listeners-in-your-android-java-application

XML Layout 1

```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:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:clickable="false"
        android:text="Please enter your login" />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Login:" />
    <EditText
        android:id="@+id/username"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:singleLine="true" />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Password:" />
```

XML Layout 2

```
    <EditText
        android:id="@+id/password"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        android:singleLine="true" />
    <Button
        android:id="@+id/login_button"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="onLoginClick"
        android:text="Login" />
    <Button
        android:id="@+id/cancel_button"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="onCancelClick"
        android:text="Cancel" />
    <TextView
        android:id="@+id/result"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Inline Class Implementation

```
public class MainActivity extends Activity {
// Declare our Views, so we can access them later
private Button btnCancel;

@Override
public void onCreate(Bundle savedInstanceState) {
……
setContentView(R.layout.eventview);
btnCancel = (Button) findViewById(R.id.cancel_button);
btnCancel.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
// Close the application
finish();
}
}); ……
```

Use “Implements” method

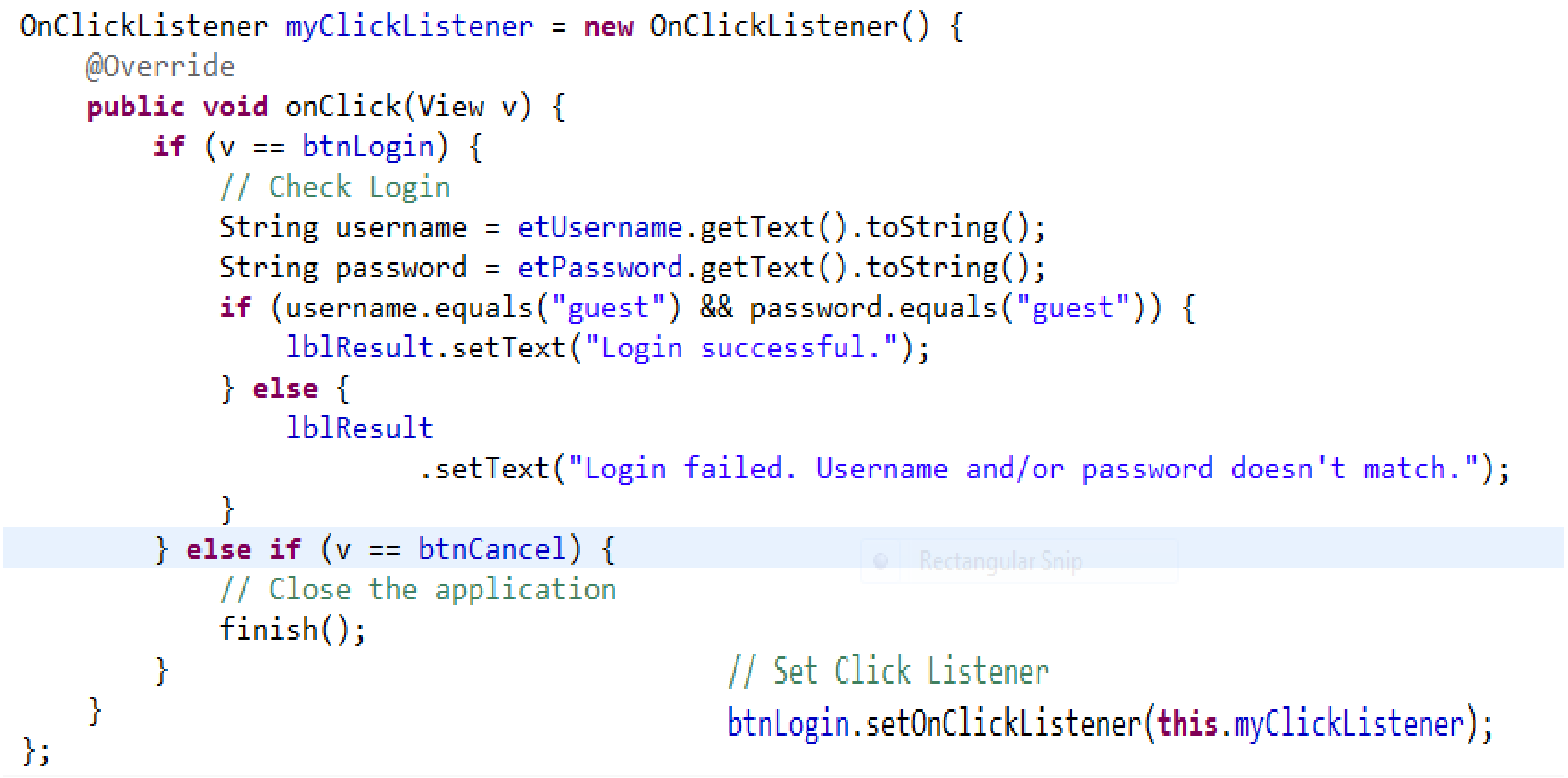
```
public class ImplementActivity extends Activity implements OnClickListener {

private Button btnLogin; //Declare our Views, so we can access them later
private Button btnCancel;
....
public void onCreate(Bundle savedInstanceState) {
setContentView(R.layout.eventview);
btnLogin = (Button) findViewById(R.id.login_button);

btnLogin.setOnClickListener(this); // Set Click Listener ………

…… implement onClick method
@Override
public void onClick(View v) {
if (v == btnLogin) {………
}
}……
```

Use a variable for a “listener method”



```java
OnClickListener myClickListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        if (v == btnLogin) {
            // Check Login
            String username = etUsername.getText().toString();
            String password = etPassword.getText().toString();
            if (username.equals("guest") && password.equals("guest")) {
                lblResult.setText("Login successful.");
            } else {
                lblResult
                        .setText("Login failed. Username and/or password doesn't match.");
            }
        } else if (v == btnCancel) {
            // Close the application
            finish();
        }
    }
};
```

```java
// Set Click Listener
btnLogin.setOnClickListener(this.myClickListener);
```

FW controlled way for clickable Views

```
From layout file
    <Button
        android:id="@+id/cancel_button"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="onCancelClick"
        android:text="Cancel" />


In Activity add this method

public void onCancelClick(View v) {
//remark signature public void [Name](View v)
finish();
};
```

Special activities

- MapActivity
- ListActivity
- ExtendedListActivity
- Fragments
  - Before Fragments: TabActivity

Lesson 4 Exercise 1

1. [Meier] ch 4 tutorial 1 To Do List Example (ch 4 excerpt on Campusnet.)
2. Or Google Notepad tutorial

   Exercise1,2 and 3

   http://developer.android.com/resources/tutorials/notepad/index.html