# Building User Interfaces

Dimitar G. Dimitrov

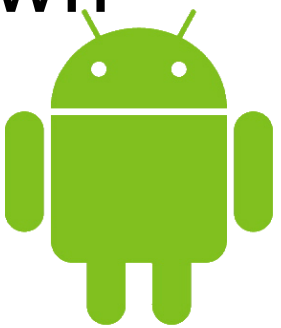
# XML-Based Layouts

- Two ways to build UI: XML vs. Java
- XML:
  - Declarative user interface
  - Intuitive (like HTML)
  - Separation between view and programming logic
  - Easy to use with GUI builders

- XML tag <-> instance of Java class

```
<Button
     android:text="Touch me"
     ...
/>

Button butt = new Button(this);
butt.setText("Touch me");
...
```

- Use Java when the widgets are not known at compile-time



# Getting Started

- New Android Project
- XML file
  - in "res/layout" directory
- Activity
  - in onCreate: setContentView
- AndroidManifest.xml

# Class android.view.View

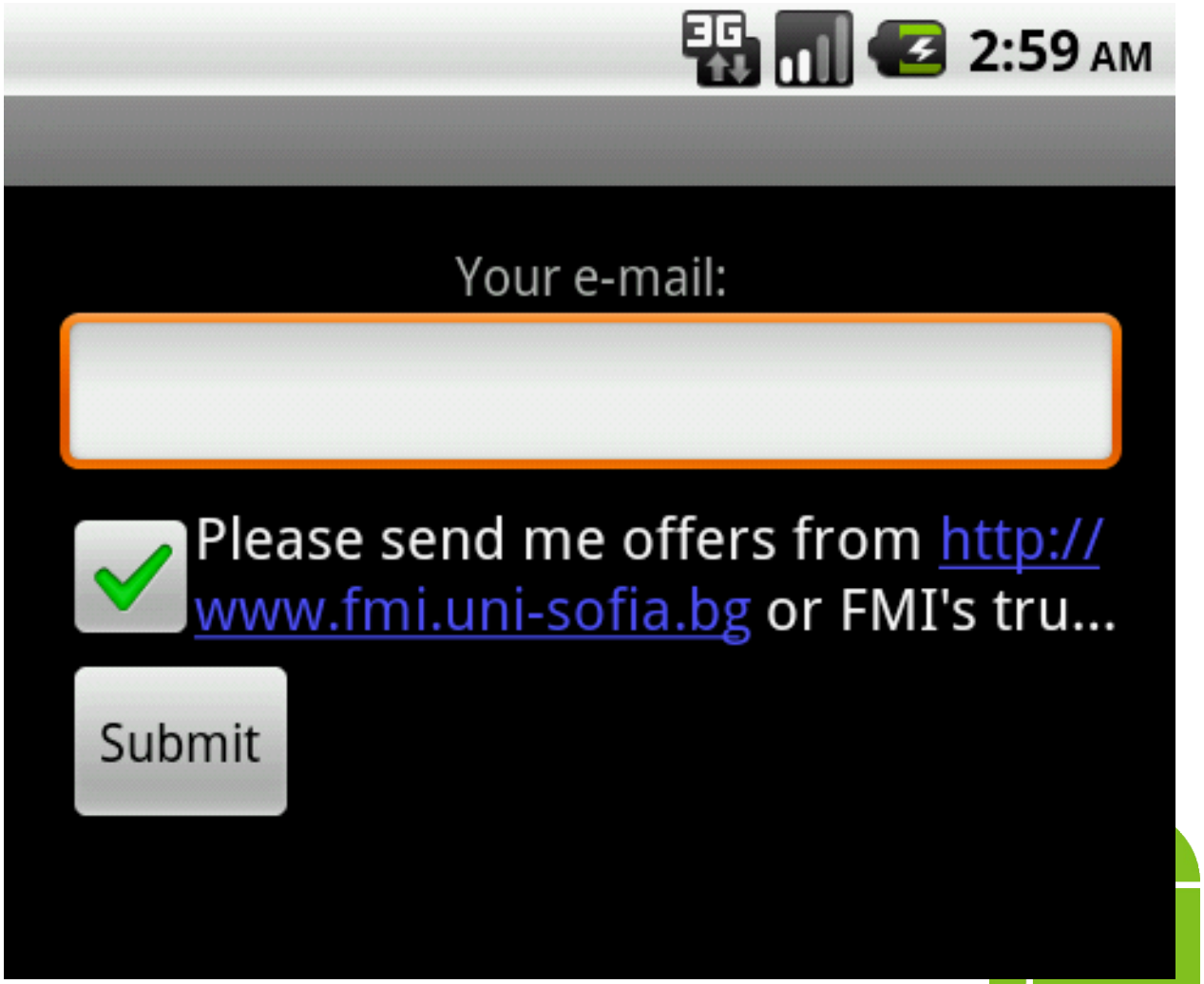
- The basic building block for UI components
- All UI components are direct or indirect subclasses of View (e.g. TextView, Button, LinearLayout, etc.)
- A View occupies a rectangular area on the screen and is responsible for drawing and event handling

# Some Common View Properties

- android:id (int)
  - Mandatory if the view is referenced by others
  - `id="@+id/my_button"`
- android:visibility (int)
  - View.VISIBLE, View.INVISIBLE, View.GONE
- android:focusable (boolean)
- android:background (int)
  - Background image
  - Use `@drawable/xxx` to refer to `/res/drawable/xxx.png` (or xxx.jpg, xxx.9.png...)

# Basic Android UI Components

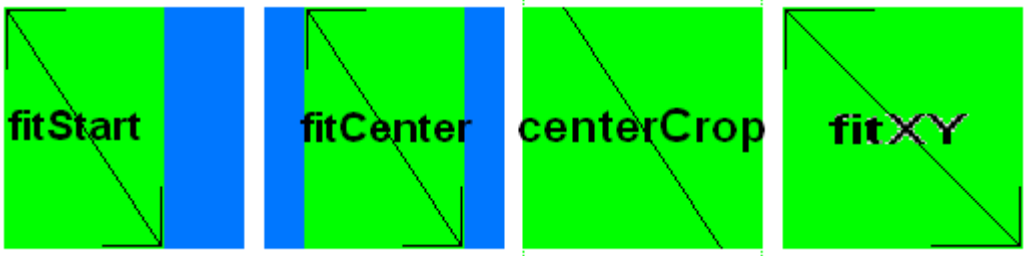
- TextView
- EditText
- CheckBox
- RadioButton
- Button
- ImageView



- android:text, textSize, typeface, textColor
- android:gravity
  - One or more (separated by '|') of the following: `top bottom left right center_vertical center ...`
- android:minLines (also maxLines, minWidth ...)
- android:ellipsize
- android:autoLink
  - `none web email phone map all`
- Rich text support – use class SpannableString

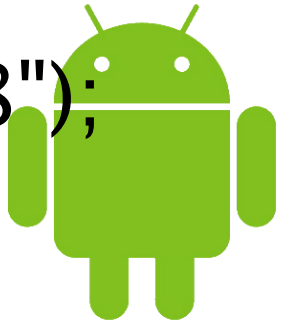
# ImageView

- android:src (drawable id)
- android:scaleType
  - Controls how the image should be resized or moved to match the size of the ImageView
  - fitCenter, fitStart, fitEnd, fitXY, center, centerCrop, centerInside, matrix
- getImageMatrix(), setImageMatrix(Matrix)



# WebView

- Displays web pages
- WebView vs Browser application
- To access Internet, add INTERNET permission in the manifest: `<uses-permission android:name="android.permission.INTERNET" />`
- loadUrl("http://...")
- loadData("<html>...", "text/html", "utf-8");
- goBack(), reload(), ...



# Some More Views

- ProgressBar
- KeyboardView: a virtual keyboard
- SurfaceView: provides a dedicated drawing surface embedded inside of a view hierarchy
- ViewStub: an invisible, zero-sized View that can be used to lazily inflate layout resources at runtime
- DatePicker, TimePicker; AnalogClock
- ToggleButton

# Accessing UI Components From Code

To access the following view:

```
<TextView android:id="@+id/myTextView"
  ... />
```

write the following code:

```
TextView myTextView = (TextView)
  findViewById(R.id.myTextView);
```

# The gen/R.java File

- Contains reference identification numbers for all resources in the res/ directory
- Use these numbers rather than the items on the filesystem
- int constants
- Auto-generated
- Nested classes: id, layout, string, drawable, ...
- No reason to modify it
- Can be ignored from version control (SVN, git...)

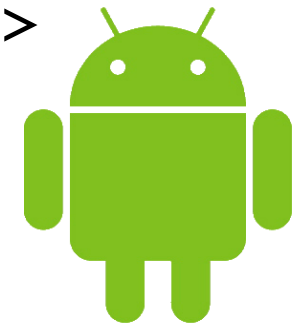
# Handling User Interactions

- MVC
- Event Listeners:
  - onClick(View v)
  - onLongClick(View v)
  - onFocusChange(View v, boolean hasFocus)
  - onKey(View v, int keyCode, KeyEvent event)
  - onTouch(View v, MotionEvent event)

# Handling User Interactions – 2

- In Android 1.6 or later:

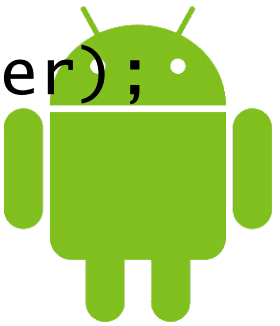
```
class MyActivity extends Activity {
    public void myClickHandler(View target) {
        // Do stuff
    }
}

<Button android:onClick="myClickHandler" />
```



# Handling User Interactions – 3

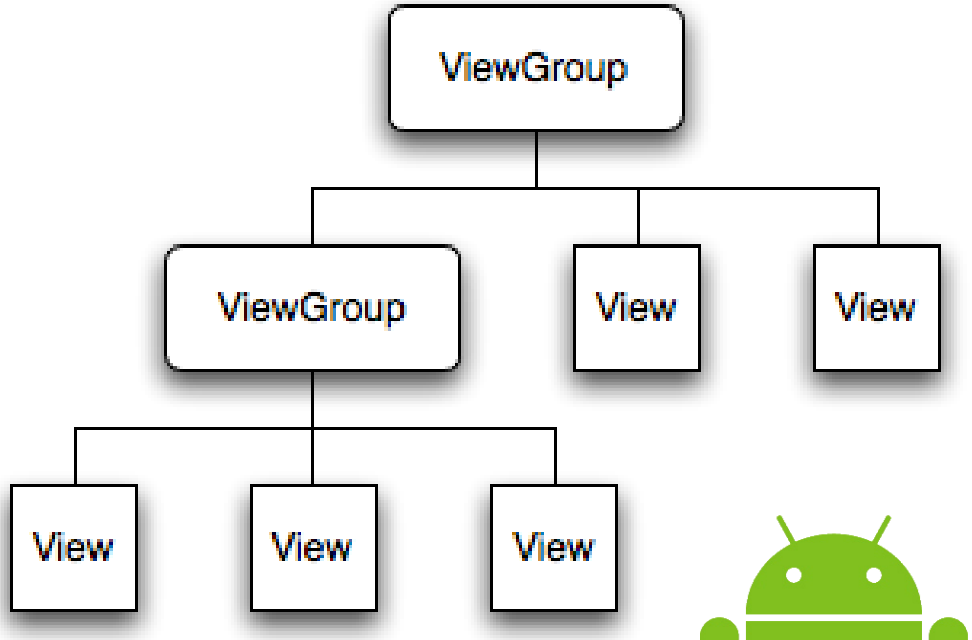
- In all Android versions:

```
private OnClickListener mMyButtonListener = new
  OnClickListener() {
     public void onClick(View v) {
       // do something when the button is clicked
     }
};
protected void onCreate(Bundle savedValues) {
     ...
     Button button =
        (Button)findViewById(R.id.myButton);
     button.setOnClickListener(mMyButtonListener);
     ...
}
```



# View Groups

- Can contain other views (called children)
- Views are leaves and ViewGroups are internal nodes
- Composite design pattern

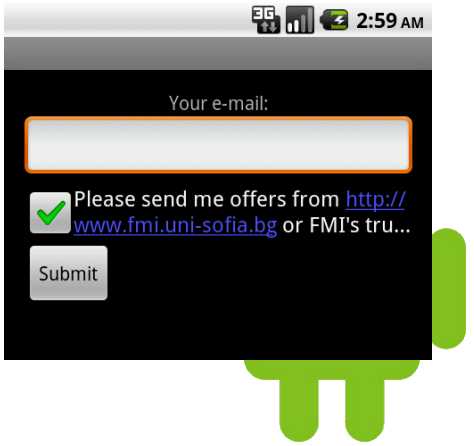



# Layout Parameters

- Used by views to tell their parents how they want to be laid out
- android:layout_width, android:layout_height
  - Describe how big the view wants to be
  - Mandatory for any view inside of a containing layout manager!
  - Allowed values:
    - wrap_content – just large enough to fit the view's internal content, taking its own padding into account
    - fill_parent (match_parent in API Level 8) – as big as the parent, minus the parent's padding
    - <exact number>

# Layouts

- Subclasses of ViewGroup
- FrameLayout
  - Children are drawn in a stack, pegged to the top left of the screen
  - ScrollView is a FrameLayout
- LinearLayout
  - Arranges its children in a single column or a single row
  - RadioGroup is a LinearLayout

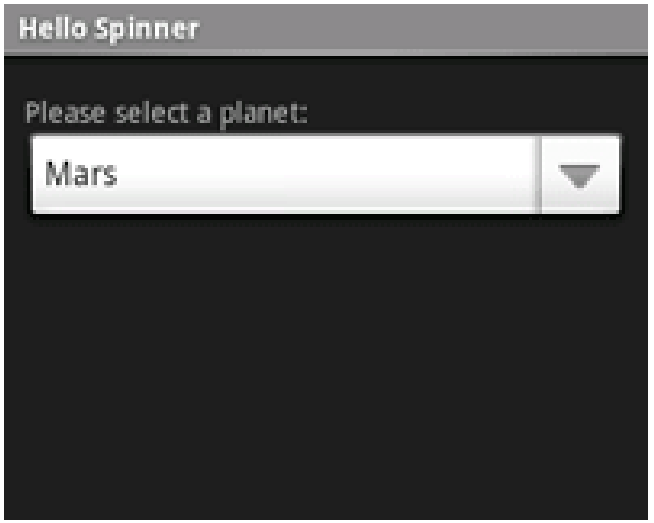



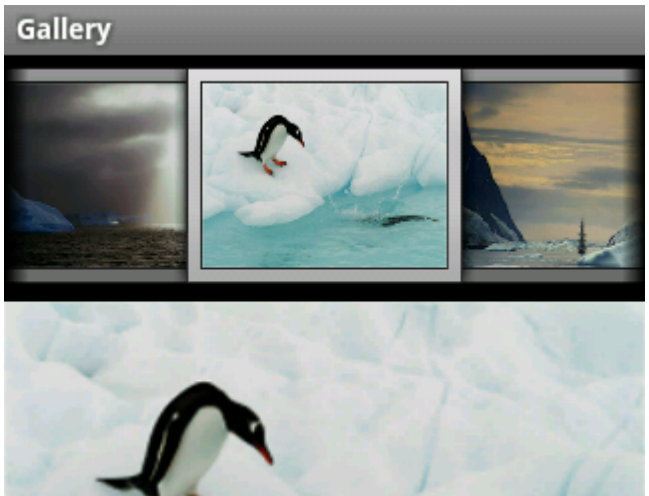
# Layouts – 2

- RelativeLayout
  - The positions of the children can be described in relation to each other or to the parent
  - In child views: android:layout_toRightOf, etc.
- TableLayout
  - Arranges its children into rows and columns
  - Consists of a number of TableRow objects
- AbsoluteLayout – deprecated

# Adapter Views

- Subclasses of ViewGroup
- ListView – list of scrollable items
- GridView – shows items in 2D scrolling grid
- Spinner – displays one child at a time and lets the user pick among them
- Gallery – shows items in a center-locked, horizontally scrolling list



# Adapters

- A bridge between an AdapterView and data
- Provides access to the data items
- Creates a View for each item in the data set
- `myView.setAdapter(adapterInstance);`

- ArrayAdapter<T>
  - By default: T[] -> TextViews using toString()
  - Override getView to create more complex views
- SimpleCursorAdapter
  - Maps columns from a cursor to TextViews or ImageViews defined in an XML file

# Custom Adapters

- `int getCount();`
- `Object getItem(int position);`
- `View getView(int position, View convertView, ViewGroup parent);`
  - convertView: it can be either null or some old view that eventually can be reused

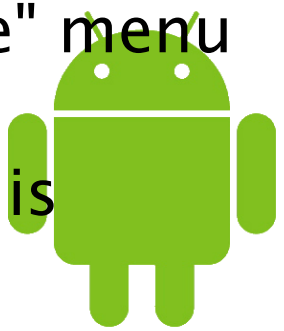
```
<SomeView> result;
if (convertView != null) {
   result = (<SomeView>) convertView; //reuse
} else { /*create new View*/ }
// setting result's properties, listeners, ...
return result;
```

# Menus

- Provide a familiar interface for the user to access application functions and settings
- Useful for additional functionality since menus don't occupy valuable screen space
- Base class: android.view.Menu
  - Not related with the View class hierarchy

# Menu Types

1. **Options Menu**: appears when the user presses the device MENU key
   - Icon Menu
     - maximum of 6 menu items
     - menu items support icons and do not support checkboxes or radio buttons
   - Expanded Menu
     - vertical list of menu items exposed by the "More" menu item in the Icon Menu
     - when the Icon Menu is full, the expanded menu is comprised of the 6th menu item and the rest



# Menu Types – 2

2. **Context Menu**: appears on long press on a View
3. **Submenu**:
   - A floating list of menu items that the user opens by pressing a menu item in the Options Menu or a context menu
   - Items cannot support nested submenus

# Defining Menus

- res/menu/game_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/
  android">
     <item android:id="@+id/new_game"
           android:icon="@drawable/ic_new_game"
           android:title="@string/new_game" />
     <item android:id="@+id/quit"
           android:icon="@drawable/ic_quit"
           android:title="@string/quit" />
</menu>
```

# Defining Menus – 2

- In the activity:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

- When the user opens the Options Menu for the first time, Android calls onCreateOptionsMenu

# Defining Menus – 3

- Handling user operations

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.new_game:
        newGame();
        return true;
    case R.id.quit:
        quit();
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}
```

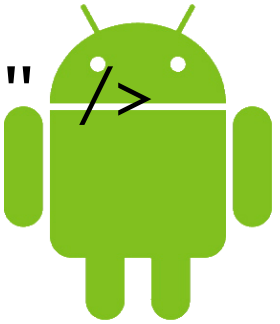
- To change the menu when it opens - onPrepareOptionsMenu()

# Defining Context Menus

- You can create a context menu for any View
- Call Activity.registerForContextMenu() and pass it the View
- Override onCreateContextMenu() and onContextItemSelected()

# Defining Submenus

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/
  android">
     <item android:id="@+id/file"
           android:icon="@drawable/file"
           android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/new"
                  android:title="@string/new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
     </item>
</menu>
```



# Specifying Colors

- Directly: similar to HTML
  - #RRGGBB, #AARRGGBB, #RGB, #ARGB
- Android predefined colors:
  - @android:color/white

# Specifying Colors – 2

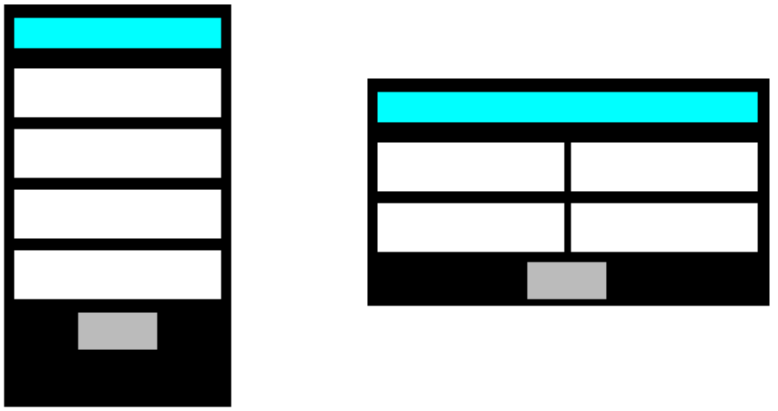
- Custom predefined colors:
  - /res/values/<filename>.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="mywhite">#ffffff</color>
    <color name="myblack">#000000</color>
</resources>
```

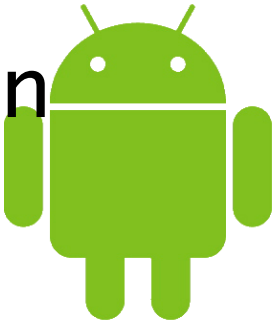
  - Example:

```
android:textColor="@color/mywhite"
```

# Different Layouts in Portrait and Landscape Mode



- /res/layout/<your-layout>.xml and /res/layout-land/<your-layout>.xml
- For drawables: /res/drawable and /res/drawable-land
- By default, onCreate is called on rotation
- Left Ctrl + F11 in emulator



# Styles

- A collection of properties that specify the look and format for a View or window
- Similar to CSS
- Separation between the design and the content
- Instead of this:

```
<TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:typeface="monospace"
  android:text="@string/hello" />
```

- you can write this:

```
<TextView style="@style/CodeFont"
  android:text="@string/hello" />
```

# Defining Styles

- res/values/<arbitrary_name>.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style
     name="CodeFont"
     parent="@android:style/TextAppearance.Medium">
     <item
      name="android:layout_width">fill_parent</item>
     <item
    name="android:layout_height">wrap_content</item>
     <item name="android:typeface">monospace</item>
   </style>
</resources>
```

- Several styles can be defined in a single file

# Applying Styles

```
<TextView style="@style/CodeFont"
   android:text="@string/hello" />
```

- The style attribute does not use the android: namespace prefix
- If a style is applied to a ViewGroup, the child View elements will **not** inherit the style properties

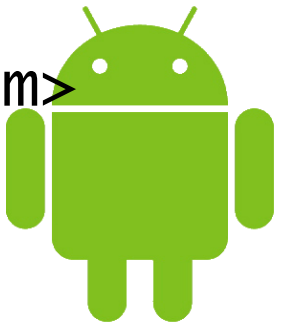
# Style Inheritance

- To inherit from built into Android style:

```
<style name="GreenText"
  parent="@android:style/TextAppearance">
   <item name="android:textColor">#00FF00</item>
</style>
```

- To inherit from styles that you've defined yourself:

```
<style name="CodeFont.White">
   <item name="android:textColor">#FFFFFF</item>
</style>
```

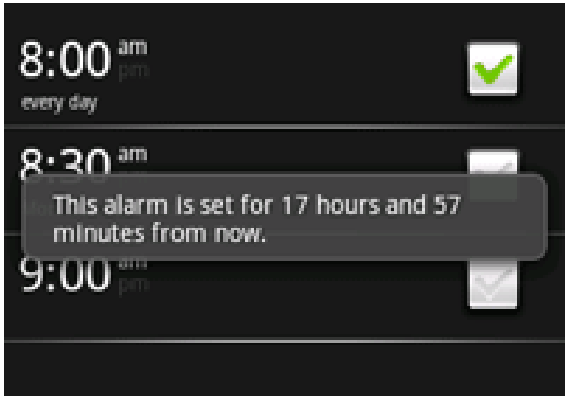


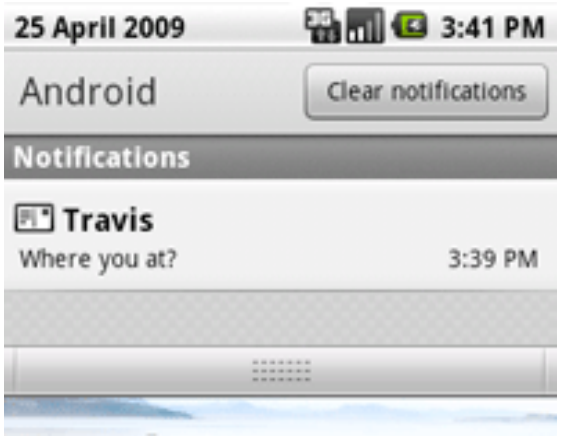
# Themes

- A style applied to an entire Activity or application, rather than an individual View
- Every View will apply each style property that it supports
- Add `android:theme` attribute to the `<activity>` or `<application>` element in the Android manifest

# Notifying the User

- Toast Notification
  - for brief messages that come from the background
- Status Bar Notification
  - for persistent reminders that come from the background and request the user's response
- Dialog Notification
  - for Activity-related notifications




# Localization

- Do not use hardcoded strings
- Place them in res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="welcome">Welcome</string>
  <string name="title">title</string>
</resources>
```

- Use @string/title and getString(R.string.title)
- Put localized strings in res/values-*xx*/strings.xml where *xx* is the ISO 639-1 language code
- Example: values-el/strings.xml for Greek:

```
<resources>
  <string name="welcome">Καλώς ήρθατε</string>
  <string name="title">τίτλος</string>
</resources>
```

ANDROID

# Thank You!