Android UI Basics

Sang Shin
Michèle Garoche
www.javapassion.com
“Learn with Passion!”

Disclaimer

- Portions of this presentation are modifications based on work created and shared by the Android Open Source Project
  - http://code.google.com/policies.html
- They are used according to terms described in the Creative Commons 2.5 Attribution License
  - http://creativecommons.org/licenses/by/2.5/

Topics

- Views
  - View IDs
  - Drawing of Views
  - Focus handling
- Styles and themes
  - Inheritance of styles
  - Style properties
  - Theme
  - Built-in styles and themes

Views

View Tree

- All of the views in a window are arranged in a single View tree.
- You can add views either from code or by specifying a tree of views in one or more XML layout files.
- There are many specialized subclasses of views that are capable of displaying text, images, or other content.

Things You Can do with Views

- Set properties
  - The available properties and the methods that set them will vary among the different subclasses of views
  - Properties that are known at build time can be set in the XML layout files.
- Set focus
- Set up listeners
- Set visibility

Framework Responsibility

- The Android framework is responsible for measuring, laying out and drawing views.
- You should not call methods that perform these actions on views yourself unless you are actually creating custom *ViewGroup* class

View IDs

IDs

- Views may have an integer id associated with them.
- These ids are typically assigned in the layout XML files, and are used to find specific views within the view tree.

Example: IDs

- Define a Button in the layout file and assign it a unique ID

```
<Button id="@+id/my_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/my_button_text"/>
```

- From the *onCreate* method of an Activity, find the Button

```
Button myButton =
    (Button) findViewById(R.id.my_button);
```

Drawing of Views

Drawing

- Drawing is handled by walking the tree and rendering each view
- Because the tree is traversed in-order, this means that parents will draw before their children, with siblings drawn in the order they appear in the tree.
- If you set a background drawable for a View, then the View will draw it for you before calling back to its onDraw() method.

Focus Handling

Focus Handling by the Framework

- Android framramework handles routine focus movement in response to user input
  - Changing the focus occurs as views are removed or hidden, or as new views become available.
- Focus movement is based on a built-in algorithm which finds the nearest neighbor in a given direction.

Application Controlled Focus Handling

- In rare cases, the default algorithm may not match the intended behavior of the developer. In these situations, you can provide explicit overrides by using these XML attributes in the layout file
  - *nextFocusDown, nextFocusLeft, nextFocusRight, nextFocusUp*

Styles and Themes

What is a Style?

- A style is a collection of properties that specify the look and format for a View
- A style can specify properties such as
  - height, padding, font color, font size, background color, and much more.
- A style is defined in an XML resource that is separate from the XML that specifies the layout.

Separation of Design from Content

- Styles in Android share a similar philosophy to cascading stylesheets in web design—they allow you to separate the design from the content.
- Reusability also improves
- For example, by creating a new style resource file, you can take this layout XML

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

- And turn it into

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

Defining Style Resource

- To create a set of styles, save an XML file in the res/values/ directory of your project.

```
<?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:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Styles and Themes: Inheritance of Styles

Inheritance of styles

- The *parent* attribute in the <*style*> element is optional and specifies the resource ID of another style from which this style should inherit properties.
- You can then override the inherited style properties if you want to.

```
<?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:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Inheritance of styles defined in the same application

- If you want to inherit from styles that you've defined yourself, you do not have to use the *parent* attribute. Instead, just prefix the name of the style you want to inherit to the name of your new style, separated by a period.
- For example, to create a new style, *Red*, that inherits the *CodeFont* style defined above, but make the color red, you can author the new style like this:

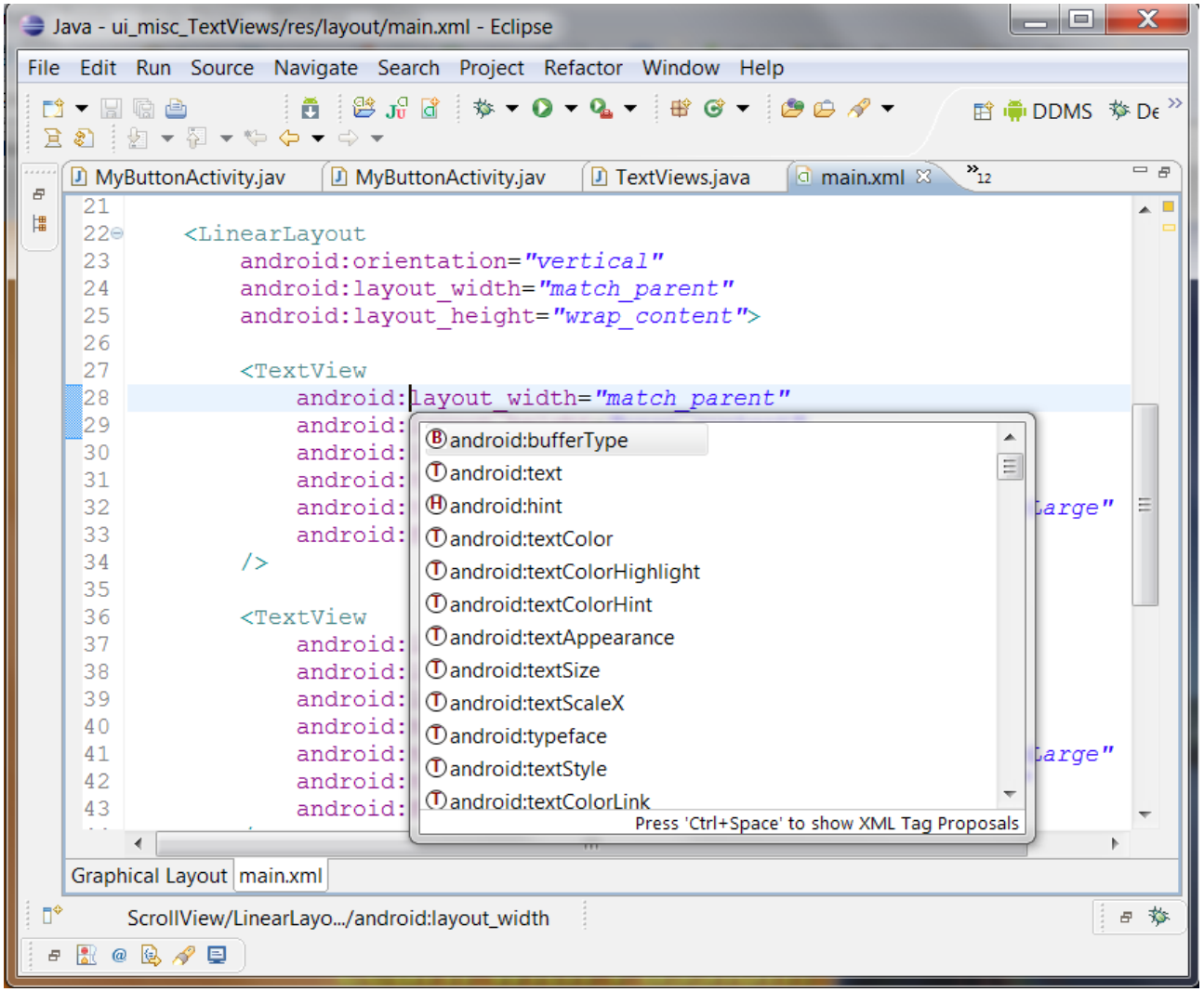
```
<style name="CodeFont.Red">
    <item name="android:textColor">#FF0000</item>
</style>
```

Styles and Themes:
Style Properties

Style Properties

- The best place to find properties that apply to a specific View is the corresponding class reference, which lists all of the supported XML attributes.
  - For example, all of the attributes listed in the table of TextView XML attributes can be used in a style definition for a TextView element (or one of its subclasses)
- You can also use code-completion feature of Eclipse IDE

Code-completion of Attributes



Example: Creating a style

- You might normally place the *android:inputType* attribute in an <EditText>

```
<EditText
    android:inputType="number"
    ... />
```

- You can instead create a style for the EditText element that includes this property:

```
<style name="Numbers">
  <item name="android:inputType">number</item>
  ...
</style>
```

- with your XML for the layout

```
<EditText
    style="@style/Numbers"
    ... />
```

Two Ways to Set a Style

- Option#1 - To an individual View, by adding the *style* attribute to a View element in the XML for your layout.
- Option #2 - To an entire Activity or application, by adding the *android:theme* attribute to <*activity*> or <*application*> element in the Android manifest.

Option #1 - Applying a Style to a View

- When you apply a style to a single View in the layout, the properties defined by the style are applied only to that View.

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

Option #2 - Applying a Style as a Theme

- You can apply a style so that it applies to all View elements—by applying the style as a theme.
- To apply a style as a theme, you must apply the style to an Activity or application in the Android manifest.
  - Every View within the Activity or application will apply each property that it supports.
- Apply a theme to an application

```
<application android:theme="@style/CustomTheme">
```

- Apply a theme to an activity

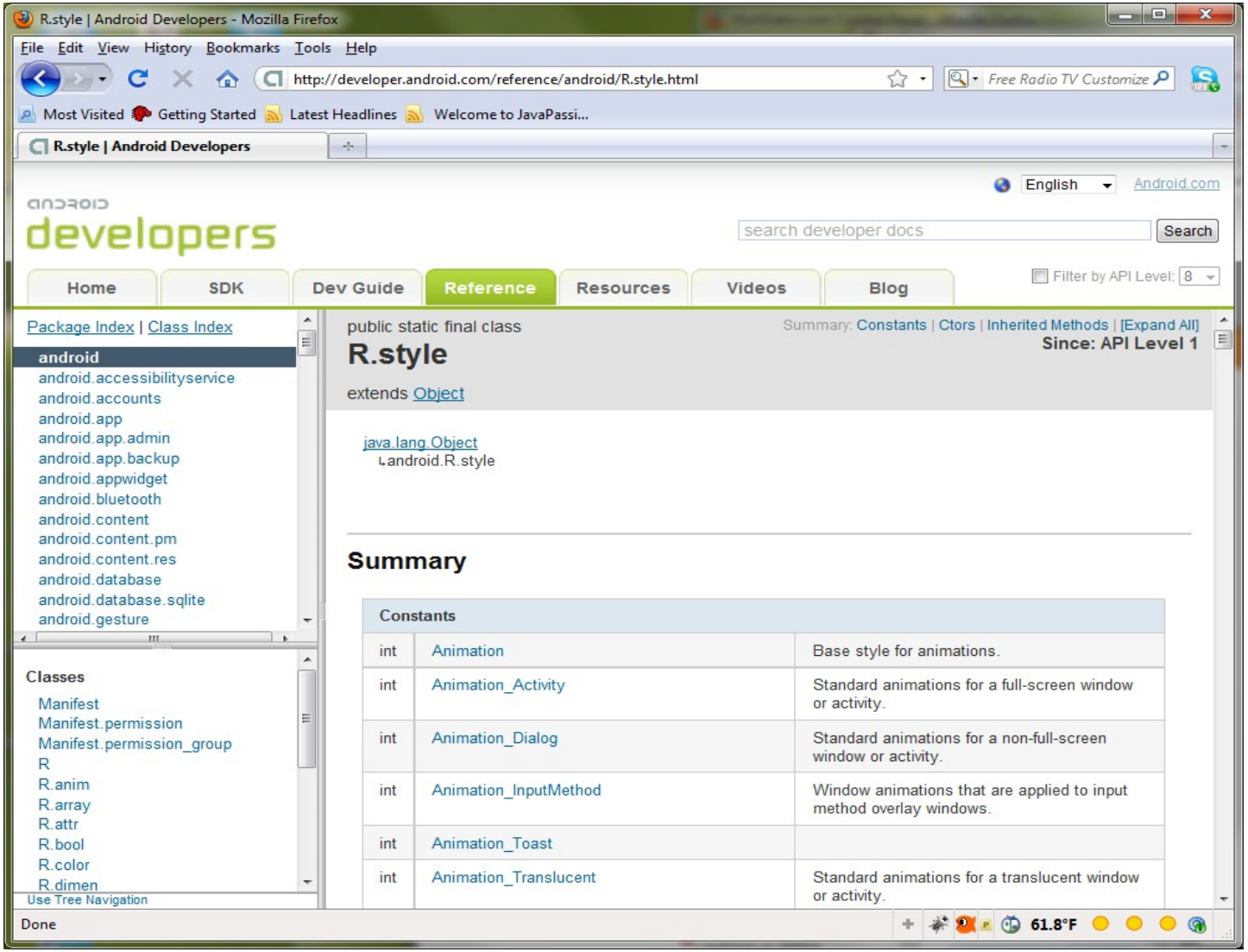
```
<activity android:theme="@android:style/Theme.Translucent">
```

Android's Built-in Styles and Themes

Android's Built-in Styles and Themes

- The Android platform provides a large collection of styles and themes that you can use in your applications.
- You can find a reference of all available styles in the R.style class.
  - http://developer.android.com/reference/android/R.style.html
- To use the styles listed above, replace all underscores in the style name with a period.
  - For example, you can apply the Theme_NoTitleBar theme with "@android:style/Theme.NoTitleBar".

R.style class



Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"