Applying Styles and Themes

http://javat.in

Applying Styles and Themes

- A **style** is a collection of properties that specify the look and format for a View or window.
  - 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.
- A **theme** is a style applied to an entire Activity or application, rather than an individual View.
  - When a style is applied as a theme, every View in the Activity or application will apply each style property that it supports.

Applying Styles and Themes

For example, by using a style, 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 this:

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

Defining Styles

- To create a set of styles, save an XML file in the res/values/ directory of your project.
- The name of the XML file is arbitrary, but it must use the .xml extension and be saved in the res/values/ folder.
- The root node of the XML file must be <resources>.
  - For each style you want to create, add a <style> element to the file with a name that uniquely identifies the style.
  - Then add an <item> element for each property of that style, with a name that declares the style property and a value to go with it .

Defining Styles - Example

```
 <?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>
```

- Each child of the <resources> element is converted into an application resource object at compile-time, which can be referenced by the value in the <style> element's name attribute.
- In this example style can be referenced from an XML layout as @style/CodeFont

Creating 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.

Inheritance

- you can inherit the Android platform's default text appearance and then modify it:

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

Inheritance

- 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.

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

- You can even inherit the above mentioned style

```
 <style name="CodeFont.Red.Big">
      <item name="android:textSize">30sp</item>
   </style>
```

- You can't inherit Android built-in styles this way.

Style Properties

- if you apply a style to a View that does not support all of the style properties, the View will apply only those properties that are supported and simply ignore the others.
- Some style properties, however, are not supported by any View element and can only be applied as a theme.
- These style properties apply to the entire window and not to any type of View.
  - For example, style properties for a theme can hide the application title, hide the status bar, or change the window's background.
  - windowNoTitle , windowBackground

Applying Styles and Themes to the UI

- There are two ways to set a style:
  - To an individual View, by adding the style attribute to a View element in the XML for your layout.
  - To an entire Activity or application, by adding the android:theme attribute to the <activity> or <application> element in the Android manifest.
- If a style is applied to a ViewGroup the child View elements will **not** inherit the style properties.
- You *can* apply a style so that it applies to all View elements—by applying the style as a theme.
- To apply a style definition as a theme, you must apply the style to an Activity or application in the Android manifest.

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

Android inbuilt themes

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

- you can modify the traditional dialog theme to use your own background image like this:

```
<style name="CustomDialogTheme"
parent="@android:style/Theme.Dialog">
    <item name="android:windowBackground">
@drawable/custom_dialog_background
</item>
</style>
```

Now use CustomDialogTheme instead of Theme.Dialog inside the Android Manifest:

```
<activity android:theme="@style/CustomDialogTheme">
```

Using Platform 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. To use the styles listed here, replace all underscores in the style name with a period.
- For example, you can apply the Theme_NoTitleBar theme with "@android:style/Theme.NoTitleBar".