Android Development with Kotlin

Style & Design I

Need for a better style and design in android?

Android users expect your app to look and behave in a way that's consistent with the platform. Not only should you follow material design guidelines for visual and navigation patterns, but you should also follow **quality guidelines for compatibility, performance, security, and more.**

Interface design and things to keep in mind:

When designing an interface, it needs to be consistent and logical. We need to create a clear design with appropriate affordances to give the best user experience (UX). Or put simply, the interface needs to make sense.

When we design an interface, we aren't designing pages, or panels, we are creating a design system of elements and components that will be used in various combinations. If we keep this in mind, we can create a truly fluid and flexible interface system.

We can use style guides to abstract and breakdown our designs into modular scalable elements and components, that can work together to form a congruous system.

Material Design:

Material Design defines the qualities that can be expressed by UI regions, surfaces, and components. Design and strategize how your app is built using foundations that address design from both a broad and detailed perspective.

There are multiple components to the material design in android which can be read about in detail using the official link:

https://material.io/develop/android

Styles and themes in android:

Styles and themes on Android allow you to separate the details of your app design from the UI structure and behaviour, similar to stylesheets in web design.

A style is a collection of attributes that specify the appearance for a single View. A style can specify attributes such as font color, font size, background color, and much more.

A theme is a type of style that's applied to an entire app, activity, or view hierarchy—not just an individual view. When you apply your style as a theme, every view in the app or activity applies each style attribute that it supports. Themes can also apply styles to non-view elements, such as the status bar and window background.

Styles and themes are declared in a **style resource file in res/values/, usually named styles.xml.**

Creating and applying a style in android:

To create a new style or theme, open your project's res/values/styles.xml file. For each style you want to create, follow these steps:

1. Add a <style> element with a name that uniquely identifies the style.
2. Add an <item> element for each style attribute you want to define.

The name in each item specifies an attribute you would otherwise use as an XML attribute in your layout. The value in the <item> element is the value for that attribute.

For example, if you define the following style:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="GreenText" parent="TextAppearance.AppCompat">
      <item name="android:textColor">#00FF00</item>
   </style>
</resources>
```

You can apply the style to a view as follows:

```
<TextView
   style="@style/GreenText"
   ... />
```

Each attribute specified in the style is applied to that view if the view accepts it. The view simply ignores any attributes that it does not accept.

Applying a theme:

You can create a theme the same way you create styles. The difference is how you apply it: instead of applying a style with the style attribute on a view, you apply a theme with the android:theme attribute on either the <application> tag or an <activity> tag in the AndroidManifest.xml file.

For example, here's how to apply the Android Support Library's material design "dark" theme to the whole app:

```
<manifest ... >
    <application android:theme="@style/Theme.AppCompat" ... >
    </application>
</manifest>
```

And here's how to apply the "light" theme to just one activity:

```
<manifest ... >
    <application ... >
        <activity android:theme="@style/Theme.AppCompat.Light" ... >
        </activity>
    </application>
</manifest>
```

ActionBar in android:

A primary toolbar within the activity that may display the activity title, application-level navigation affordances, and other interactive items.

Beginning with Android 3.0 (API level 11), **the action bar appears at the top of an activity's window when the activity uses the system's Holo theme (or one of its descendant themes),** which is the default. You may otherwise add the action bar by calling requestFeature(FEATURE_ACTION_BAR) or by declaring it in a custom theme with the windowActionBar property.

Beginning with Android L (API level 21), the action bar may be represented by any Toolbar widget within the application layout. The application may signal to the Activity which Toolbar should be treated as the Activity's action bar**. Activities that use this feature should use one of the supplied .NoActionBar themes, set the windowActionBar attribute to false or otherwise not request the window feature.**

```
var actionBar = getSupportActionBar(); // or getActionBar();
getSupportActionBar().setTitle("My new title"); // set the top title
var title = actionBar.getTitle().toString(); // get the title
actionBar.hide(); // or even hide the actionbar
```

Adaptive App icons in android:

Android 8.0 (API level 26) introduces adaptive launcher icons, which can display a variety of shapes across different device models. **For example, an adaptive launcher icon can display a circular shape on one OEM device, and display a squircle on another device. Each device OEM provides a mask, which the system then uses to render all adaptive icons with the same shape. Adaptive launcher icons are also used in shortcuts, the Settings app, sharing dialogs, and the overview screen.**

To add an adaptive icon to an app using XML, begin by updating the android:icon attribute in your app manifest to specify a drawable resource. You can also define an icon drawable resource using the android:roundIcon attribute. You must only use the android:roundIcon attribute if you require a different icon asset for circular masks, if for example the branding of your logo relies on a circular shape. The following code snippet illustrates both of these attributes:

```
<application
    …
    android:icon="@mipmap/ic_launcher"
    android:roundIcon="@mipmap/ic_launcher_round"
    …>
</application>

```

Next you must create alternative drawable resources in your app for use with Android 8.0 (API level 26) in res/mipmap-anydpi-v26/ic_launcher.xml. You can then use the <adaptive-icon> element to define the foreground and background layer drawables for your icons. The <foreground> and <background> inner elements both support the android:drawable attribute.

```
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon
xmlns:android="http://schemas.android.com/apk/res/android">
    <background android:drawable="@drawable/ic_launcher_background" />
    <foreground android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>
```

We have covered all the concepts related to style and design in the lectures. Feel free to explore more using these links:

Design Concepts: https://developer.android.com/design

All about ActionBar: https://developer.android.com/reference/android/app/ActionBar

Adaptive Icons:

https://developer.android.com/guide/practices/ui_guidelines/icon_design_adaptive