Android Development with Kotlin

Style & Design II

Introduction:

Now continuing our interesting lecture, we will be covering more material design oriented components in this lecture.

1) Toolbars
2) Resource Qualifiers
3) Fonts
4) Coordinator Layout
5) Navigation Drawer

Toolbars in Android:

Android Toolbar widget is generally found on the top of the screen. The application title, logo, navigation icon, and the menu bar is displayed inside the toolbar.

The toolbar is the material design replacement for the old and now deprecated ActionBar.

The toolbar is available with the following dependency.

```
implementation 'com.android.support:appcompat-v7:27.1.0'
```

Setting a toolbar in android:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <android.support.v7.widget.Toolbar
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />


</LinearLayout>
```

Resource Qualifiers:

You should place each type of resource in a specific subdirectory of **your project's res/ directory.** For example, here's the file hierarchy for a simple project:

```
MyProject/
    src/
        MyActivity.kt
    res/
        drawable/
            graphic.png
        layout/
            main.xml
```

```
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```

Refer to this table for resource grouping:

Directory	Resource Type
`animator/`	XML files that define property animations.
`anim/`	XML files that define tween animations. (Property animations can also be saved in this directory, but the `animator/` directory is preferred for property animations to distinguish between the two types.)
`color/`	XML files that define a state list of colors.
`drawable/`	Bitmap files (`.png`, `.9.png`, `.jpg`, `.gif`) or XML files that are compiled into the following drawable resource subtypes: Bitmap files Nine-Patches (re-sizable bitmaps) State lists Shapes Animation drawables Other drawables
`mipmap/`	Drawable files for different launcher icon densities. For more information on managing launcher icons with `mipmap/` folders.
`layout/`	XML files that define a user interface layout.
`menu/`	XML files that define app menus, such as an Options Menu, Context Menu, or Sub Menu.

raw/	Arbitrary files to save in their raw form. To open these resources with a raw InputStream, call Resources.openRawResource() with the resource ID, which is R.raw.*filename*. However, if you need access to original file names and file hierarchy, you might consider saving some resources in the assets/ directory (instead of res/raw/). Files in assets/ aren't given a resource ID, so you can read them only using AssetManager.
values/	XML files that contain simple values, such as strings, integers, and colors. Whereas XML resource files in other res/ subdirectories define a single resource based on the XML filename, files in the values/ directory describe multiple resources. For a file in this directory, each child of the <resources> element defines a single resource. For example, a <string> element creates an R.string resource and a <color> element creates an R.color resource. Because each resource is defined with its own XML element, you can name the file whatever you want and place different resource types in one file. However, for clarity, you might want to place unique resource types in different files. For example, here are some filename conventions for resources you can create in this directory: arrays.xml for resource arrays (typed arrays). colors.xml for color values dimens.xml for dimension values. strings.xml for string values. styles.xml for styles.
xml/	Arbitrary XML files that can be read at runtime by calling Resources.getXML(). Various XML configuration files must be saved here.
font/	Font files with extensions such as .ttf, .otf, or .ttc, or XML files that include a <font-family> element. For more information about fonts as resources.

Fonts in Android:

Android 8.0 (API level 26) introduces a new feature, Fonts in XML, which lets you use fonts as resources. You can add the `font` file in the `res/font/` folder to bundle fonts as resources. These fonts are compiled in your `R` file and are automatically available in Android Studio. You can access the font resources with the help of a new resource type, `font`. For example, to access a font resource, use `@font/myfont`, or `R.font.myfont`.

```
<?xml version="1.0" encoding="utf-8"?>
<font-family xmlns:android="http://schemas.android.com/apk/res/android">
    <font
        android:fontStyle="normal"
        android:fontWeight="400"
        android:font="@font/lobster_regular" />
    <font
        android:fontStyle="italic"
        android:fontWeight="400"
        android:font="@font/lobster_italic" />
</font-family>
```

Coordinator Layout:

CoordinatorLayout is a super-powered FrameLayout.

CoordinatorLayout is intended for two primary use cases:

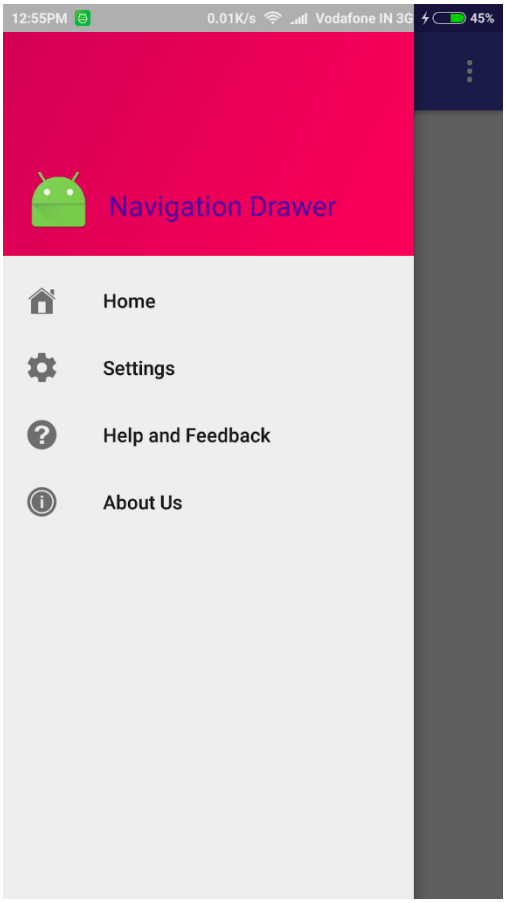
1. As a top-level application decor or chrome layout
2. As a container for a specific interaction with one or more child views

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.coordinatorlayout.widget.CoordinatorLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout_width="match_parent"

android:layout_height="match_parent"

tools:context=".MainActivity">

<com.google.android.material.appbar.AppBarLayout

android:layout_height="wrap_content"

android:layout_width="match_parent"

android:theme="@style/AppTheme.AppBarOverlay">


<androidx.appcompat.widget.Toolbar
```

```
android:id="@+id/toolbar"
android:layout_width="match_parent"
android:layout_height="?attr/actionBarSize"
android:background="?attr/colorPrimary"
app:popupTheme="@style/AppTheme.PopupOverlay"/>
</com.google.android.material.appbar.AppBarLayout>
<include layout="@layout/content_main"/>
<com.google.android.material.floatingactionbutton.FloatingActionButton
android:id="@+id/fab"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="bottom|end"
android:layout_margin="@dimen/fab_margin"
app:srcCompat="@android:drawable/ic_dialog_email"/>

</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Navigation Drawer:

The navigation drawer is a panel that displays the app's main navigation options on the left edge of the screen. It is hidden most of the time but is revealed when the user swipes a finger from the left edge of the screen or, while at the top level of the app, the user touches the app icon in the action bar.



Code in activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout
xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:app="http://schemas.android.com/apk/res-auto"
 xmlns:tools="http://schemas.android.com/tools"
 android:id="@+id/drawer_layout"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 tools:context=".MainActivity"><FrameLayout
 android:id="@+id/content_frame"
 android:layout_width="match_parent"
```

```
 android:layout_height=”match_parent”><androidx.appcompat.widget.Toolbar
 android:id=”@+id/toolbar”
 android:layout_width=”match_parent”
 android:layout_height=”?attr/actionBarSize”
 android:background=”?attr/colorPrimary”
 app:popupTheme=”@style/ThemeOverlay.AppCompat.Light” /></FrameLayout>
<com.google.android.material.navigation.NavigationView
 android:id=”@+id/navigationView”
 android:layout_width=”wrap_content”
 android:layout_height=”match_parent”
 android:layout_gravity=”start”
 app:headerLayout=”@layout/nav_header”
 app:menu=”@menu/drawer_view”
/></androidx.drawerlayout.widget.DrawerLayout>
```

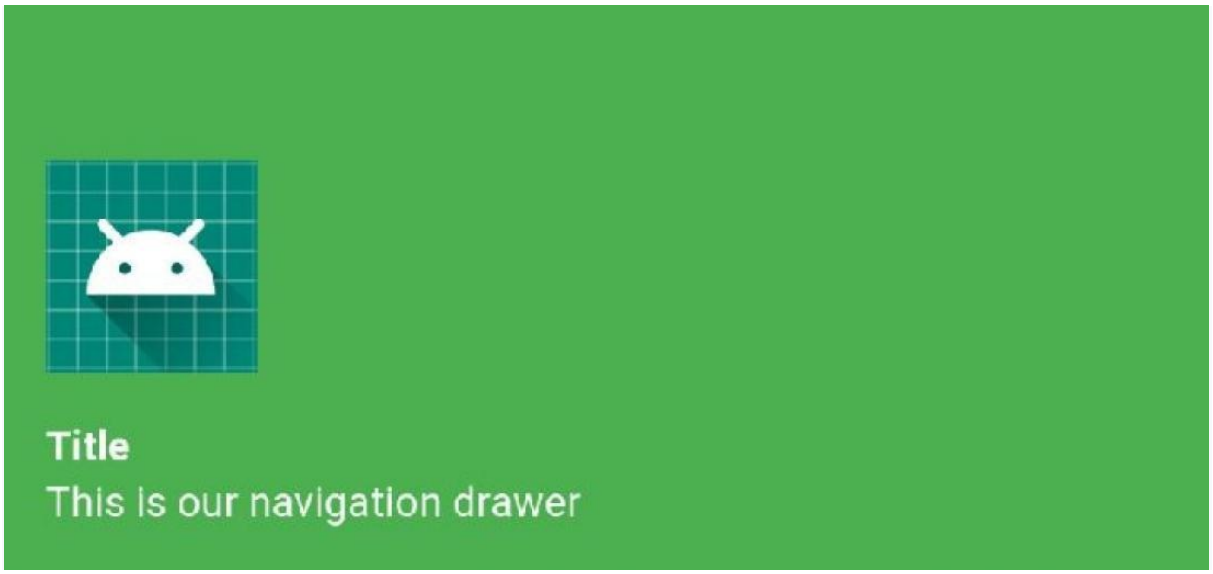
As you can see, the layouts that will contain our navigation drawer must have a DrawerLayout as a parent view. our other ui design codes will be inside this. Here include tag is used to grab all the views from another layout resource file.

Navigation Drawer in nav_header.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:background="#4CAF50"
        android:orientation="vertical"
        android:gravity="bottom"
        android:paddingLeft="15dp"
        android:paddingBottom="15dp">

    <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:srcCompat="@mipmap/ic_launcher"
            android:id="@+id/imageView"
            android:layout_marginBottom="15dp"/>

    <TextView
            android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:text="Title"
            android:textColor="#FFFFFF"
            android:textStyle="bold"/>

    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="This is our navigation drawer"
            android:textColor="#FFFFFF"/>

</LinearLayout>
```

This will create our drawer header design like:



Now we'll modify the menu resource file "nav_menu.xml" which will be the options displayed in our navigation drawer.

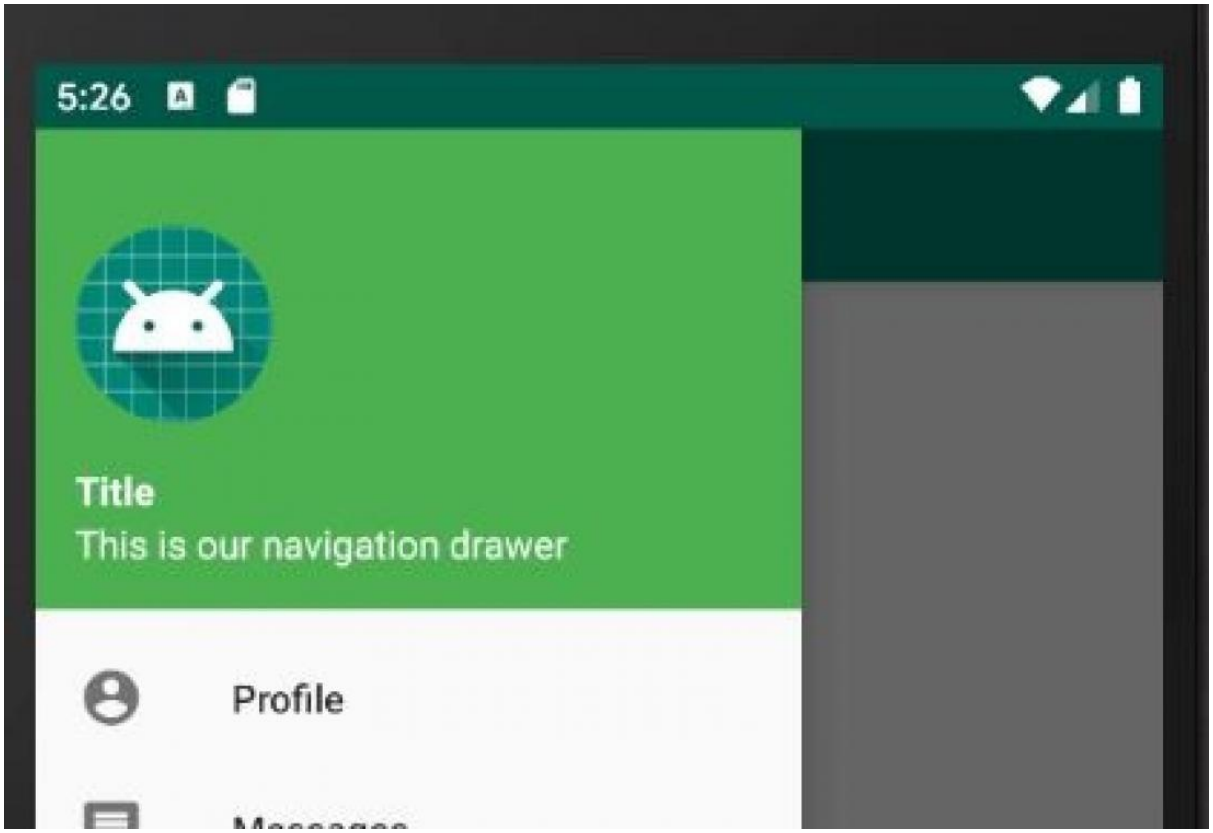
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <group>
        <item
                android:id="@+id/nav_profile"
                android:icon="@drawable/ic_profile"
                android:title="Profile"/>
        <item
                android:id="@+id/nav_messages"
                android:icon="@drawable/ic_messages"
                android:title="Messages"/>
```

```
        <item
                android:id="@+id/nav_friends"
                android:icon="@drawable/ic_friends"
                android:title="Friends"/>
    </group>


    <item android:title="Account Settings">
        <menu>
            <item
                    android:id="@+id/nav_update"
                    android:icon="@drawable/ic_update"
                    android:title="Update Profile"/>
            <item
                    android:id="@+id/nav_logout"
                    android:icon="@drawable/ic_signout"
                    android:title="Sign Out"/>
        </menu>
    </item>
</menu>
```

Code in MainActivity.kt :

```
package com.codingninjas.navigationdrawer

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.MenuItem
import android.widget.Toast
import androidx.appcompat.app.ActionBarDrawerToggle
import androidx.appcompat.widget.Toolbar
import androidx.core.view.GravityCompat
import androidx.drawerlayout.widget.DrawerLayout
import com.google.android.material.navigation.NavigationView

class MainActivity : AppCompatActivity(),
NavigationView.OnNavigationItemSelectedListener {


    lateinit var toolbar: Toolbar
    lateinit var drawerLayout: DrawerLayout
    lateinit var navView: NavigationView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

```
        toolbar = findViewById(R.id.toolbar)
        setSupportActionBar(toolbar)

        drawerLayout = findViewById(R.id.drawer_layout)
        navView = findViewById(R.id.nav_view)

        val toggle = ActionBarDrawerToggle(
            this, drawerLayout, toolbar, 0, 0
        )
        drawerLayout.addDrawerListener(toggle)
        toggle.syncState()
        navView.setNavigationItemSelectedListener(this)
    }

    override fun onNavigationItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.nav_profile -> {
                Toast.makeText(this, "Profile clicked",
Toast.LENGTH_SHORT).show()
            }
            R.id.nav_messages -> {
                Toast.makeText(this, "Messages clicked",
Toast.LENGTH_SHORT).show()
            }
            R.id.nav_friends -> {
                Toast.makeText(this, "Friends clicked",
Toast.LENGTH_SHORT).show()
            }
            R.id.nav_update -> {
                Toast.makeText(this, "Update clicked",
Toast.LENGTH_SHORT).show()
            }
            R.id.nav_logout -> {
                Toast.makeText(this, "Sign out clicked",
Toast.LENGTH_SHORT).show()
            }
        }
        drawerLayout.closeDrawer(GravityCompat.START)
        return true
    }
}
```



We have covered all the topics in depth in the lecture. Feel free to explore more about the topics using the below links:

Fonts: https://developer.android.com/guide/topics/ui/look-and-feel/fonts-in-xml

Resource Qualifiers:https://developer.android.com/guide/topics/resources/providing-resources

Toolbars: https://developer.android.com/reference/android/widget/Toolbar

Navigation Drawer: https://developer.android.com/guide/navigation/navigation-ui