MAP-06

Related Technologies for Multiplatform Applications

**Research Project**

Student:

Elina Sardaryan

Udemy tutorial “Complete Android Jetpack Masterclass”.

The link for the course: <https://www.udemy.com/course/android-jetpack-masterclass/>

The course covers

• Android Jetpack suite

• ViewBinding and Animation(Splash Screen)

• MVVM (Model View ViewModel)

• Permissions

• Glide

• ROOM Database

• LiveData, Lifecycles and ViewModels

• Navigation Component, Navigation Graph, Safe Args

*Project #1* **1\_FavDish**

1. Open Android Studio and create a new project using **Bottom Navigation Activity.**
2. Name the application as **FavDish.**
3. Select language as **Kotlin** and click the finish button.
4. Now you are done with the project creation and as you can see there are many auto-added files by Android studio already. By default we have one main UI with three fragments(dashboard, home, notification) with their activities already included in the project. We will work with them later. In this current project we are going to set up the **color themes of the application.**
5. Go to res -> values -> themes and you will notice **themes.xml** with day and night combination. Here we will set the same color combination.
6. To choose a color combination you can go to the link <https://material.io/resources/color/#!/?view.left=0&view.right=0> and select the color combination that you want.
7. For this project we will use the primary color as **Green 700** with all the combination shades. Add them to our application.
8. For secondary color we will use **Green 300** and add all the combinations to the **colors.xml** file. (Step 1)

<color name="primary\_color">#388E3C</color>  
<color name="primary\_dark\_color">#00600F</color>  
<color name="primary\_light\_color">#6ABF69</color>  
  
<color name="secondary\_color">#81C784</color>  
<color name="secondary\_dark\_color">#519657</color>  
<color name="secondary\_light\_color">#B2FAB4</color>

1. In **themes.xml** (day) we will change primary and secondary colors, as well as status bar color. (Step 2)

<item name="colorPrimary">@color/primary\_color</item>  
<item name="colorPrimaryVariant">@color/primary\_dark\_color</item>

<item name="colorSecondary">@color/secondary\_color</item>  
<item name="colorSecondaryVariant">@color/secondary\_dark\_color</item>

<item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>

1. In **themes.xml** (night) we will change primary and secondary colors, as well as status bar color. (Step 3)

<item name="colorPrimary">@color/primary\_light\_color</item>  
<item name="colorPrimaryVariant">@color/primary\_color</item>

<item name="colorSecondary">@color/secondary\_light\_color</item>  
<item name="colorSecondaryVariant">@color/secondary\_color</item>

<item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>

*Project # 2*  **2\_ActivityLifecycle**

As a user navigates through, out of, and back to your app, the Activity instances in your app transition through different states in their lifecycle. The Activity class provides several callbacks that allow the activity to know that a state has changed: that the system is creating, stopping, or resuming an activity, or destroying the process in which the activity resides.

To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks: onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy(). The system invokes each of these callbacks as an activity enters a new state.



1. Open Android Studio and create a new project using **Empty Activity**,

2. Name the application as **ActivityLifecyle**.

3. Select language as **Kotlin** and click the finish button.

4. Override all the lifecycle methods and print the log in it.

override fun onStart() {  
 super.onStart()  
 Log.e("onStart method", "is called...")  
}  
  
override fun onResume() {  
 super.onResume()  
 Log.e("onResume method", "is called...")  
}  
  
override fun onPause() {  
 super.onPause()  
 Log.e("onPause method", "is called...")  
}  
  
override fun onStop() {  
 super.onStop()  
 Log.e("onStop method", "is called...")  
}  
  
override fun onRestart() {  
 super.onRestart()  
 Log.e("onRestart method", "is called...")  
}  
  
override fun onDestroy() {  
 super.onDestroy()  
 Log.e("onDestroy method", "is called...")  
}

5. Run and see which log is printed at what time.

6. To see the Logs, go to Logcat and choose Error as we used Log.e

*Project # 3* **3\_PassingDataToAnotherActivityWithPutExtra**

1. Continue working on **ActivityLifecyle** project.
2. Add a Button in activity\_main.xml.
3. Add id for the TextView and for Button, and make the design adjustments
4. Access the button and add click event to it in Main Activity

val btnSubmit = findViewById<Button>(R.id.*btn\_submit*)  
btnSubmit.setOnClickListener **{}**

1. Create an Another Activity to launch it via Intent and to pass the data between two activities.
2. Add the TextView to the Another Activity to just see that it is launched.
3. Launch the Another Activity and pass the data using putExtra. Write this code in the button event listener.

val intent = Intent(this@MainActivity, AnotherActivity::class.*java*).*apply* **{**  
putExtra("key1", "Value1")  
 putExtra("key2", "Value2")  
 // You can add as many params as you want.  
}  
startActivity(intent)

1. Get the data in Another Activity from Main Activity and print it in the log.

val keyValue1 = *intent*.getStringExtra("key1")  
Log.i("value 1", "$keyValue1")  
val keyValue2 = *intent*.getStringExtra("key2")  
Log.i("value 2", "$keyValue2")

*Project # 4* **4\_FavDish – SplashScreen**

1. Use the project **FavDish** we created before.
2. Create a new package name as “activities.”
3. Create a new empty activity as Splash Screen with the name SplashActivity.
4. We are going to implement the **ViewBinding** concept.
5. Add the string value “Fav Dish” to the strings.xml file

<string name="splash\_screen\_title">Fav Dish</string>

1. Design the splash screen layout with one TextView and put string value to text.

android:text="@string/splash\_screen\_title"

1. Enable the ViewBinding in build.gradle(:app)

buildFeatures **{**  
viewBinding true  
}

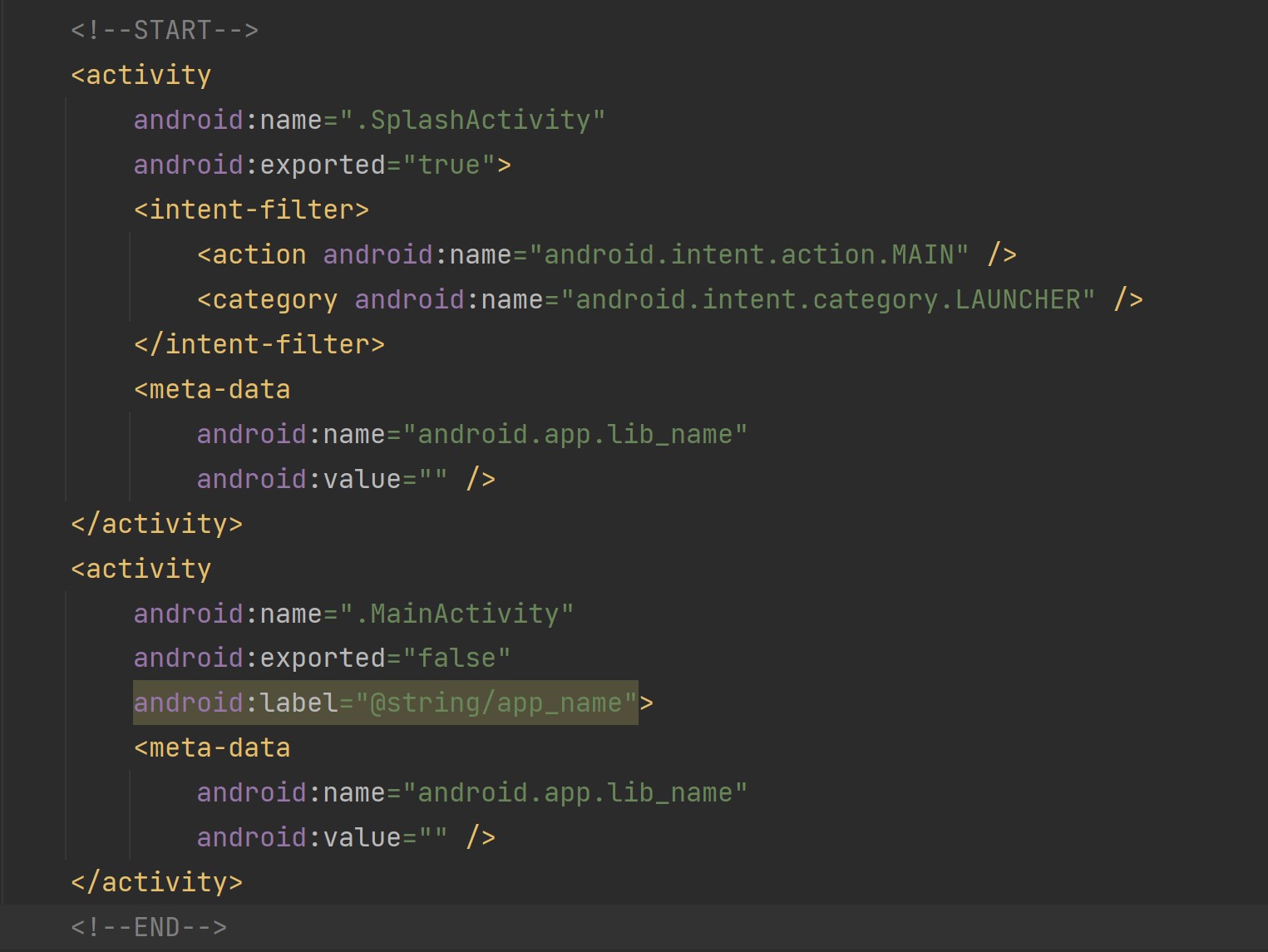
1. Access the XML layout file using the ViewBiding.

val splashBinding: ActivitySplashBinding = ActivitySplashBinding.inflate(*layoutInflater*)

1. Update the content view using the ViewBinding

setContentView(splashBinding.*root*)

1. Create the SplashActivity as the launcher activity instead of MainActivity. So, go to AndroidManifest.xml and take <intent-filter> tag with its content from MainActivity and put it in SplashActivity, and also put exported as true in SplashActivity.



1. Run the application and see the changes.
2. As you can see the launcher screen is changed and it is stuck on the splash screen. We will animate and redirect it to the main screen in the next project.

*Project # 5* **5\_FavDish - AnimatedSplashScreen**

1. Continue with the previous project **FavDish** where we created splash screen for our application.
2. Make the Splash Activity as a full screen view that means hide the Status Bar. So, first in SplashActivity we need to type:

if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*R*) {  
 *window*.*insetsController*?.hide( WindowInsets.Type.statusBars())  
} else {  
 @Suppress("DEPRECATION")  
 *window*.setFlags(  
 WindowManager.LayoutParams.*FLAG\_FULLSCREEN*,  
 WindowManager.LayoutParams.*FLAG\_FULLSCREEN*  
)  
}

Second, go to themes.xml and type:

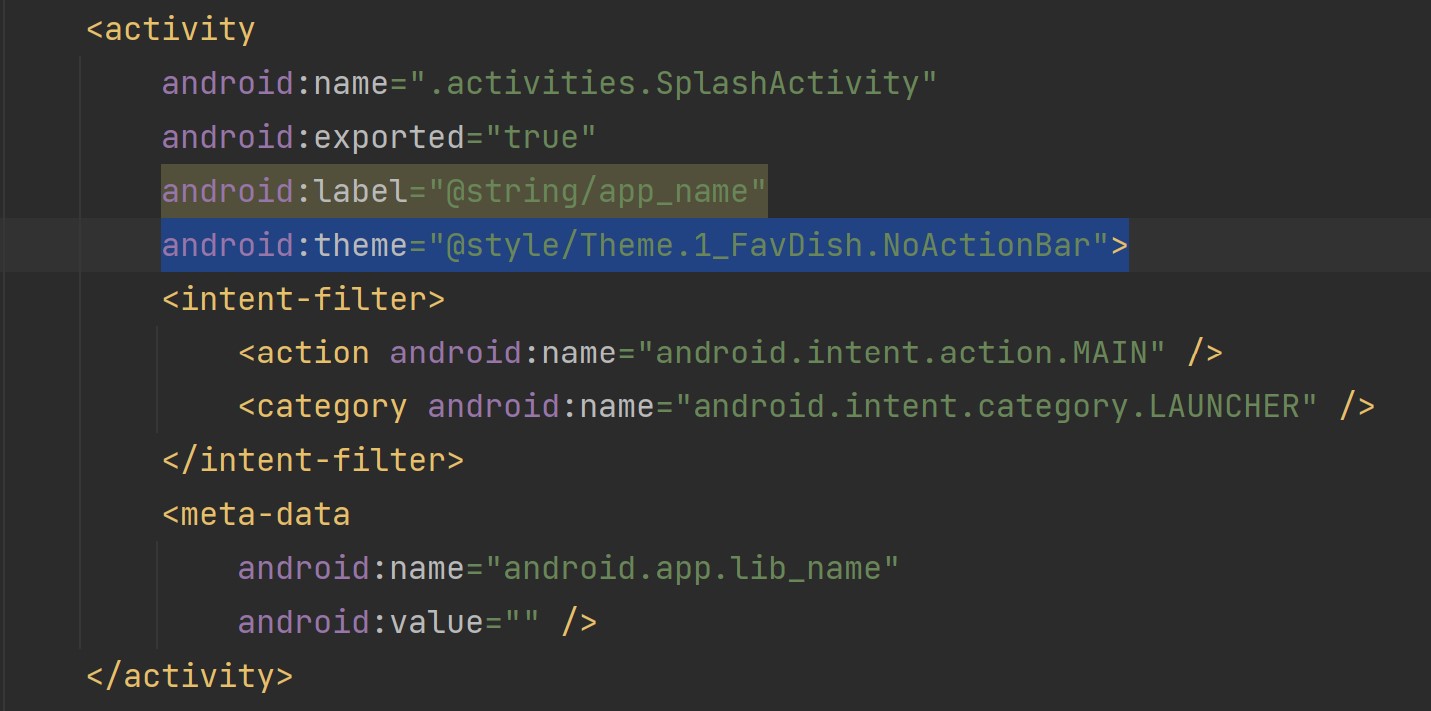
<style name="Theme.FavDish.NoActionBar">  
 <item name="windowActionBar">false</item>  
 <item name="windowNoTitle">true</item>  
</style>

<style name="Theme.FavDish.AppBarOverlay" parent="ThemeOverlay.AppCompat.Dark.ActionBar" />

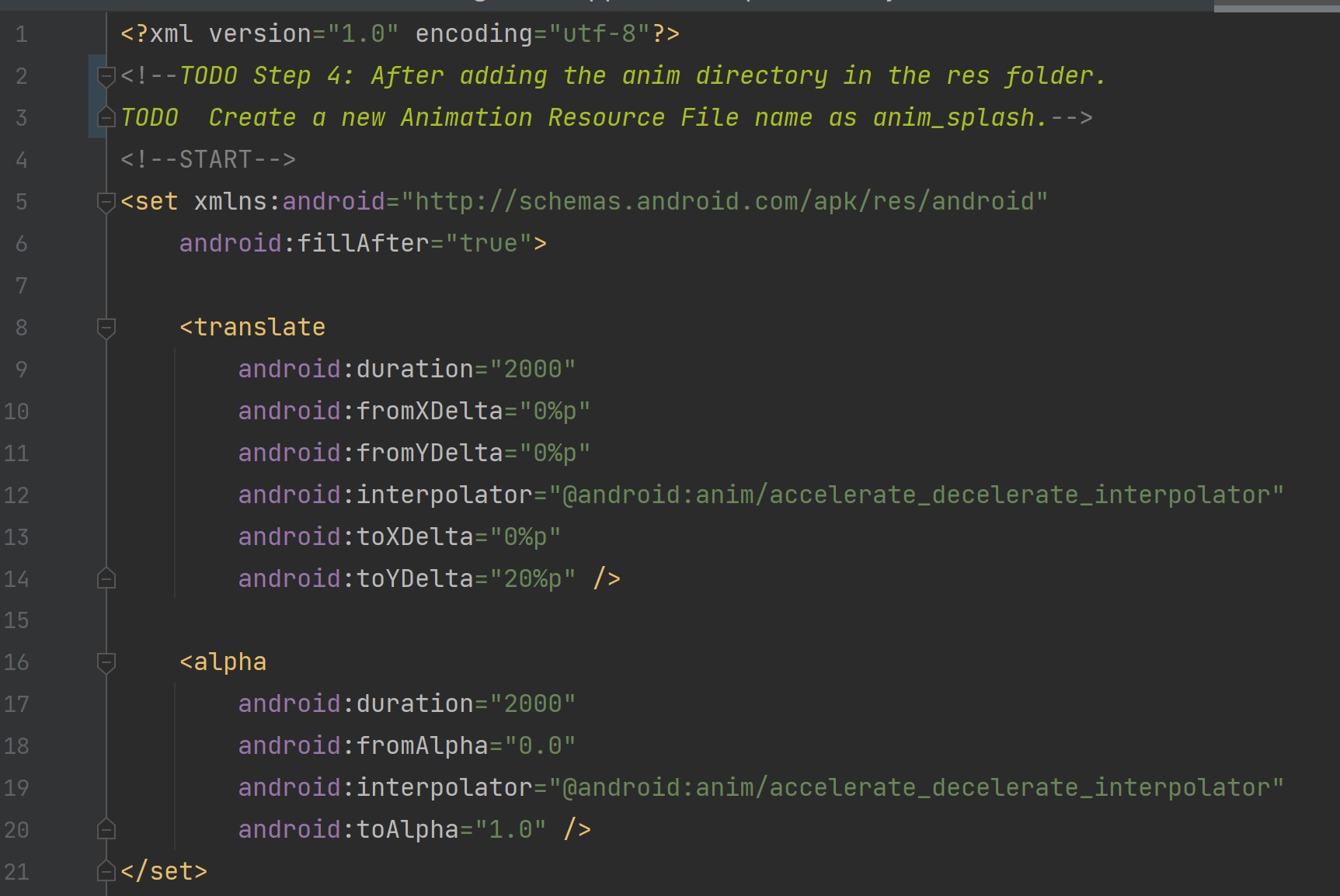
<style name="Theme.FavDish.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />

Third, go to AndroidManifest.xml and update the splash activity theme, so type:

android:theme="@style/Theme.FavDish.NoActionBar"



1. After updating the theme of SplashActivity in the manifest.xml file. Add the anim resource directory in the res folder.
2. After adding the anim directory in the res folder. Create a new Animation Resource File name as anim\_splash.



1. In Splash Activity Create a access variable for Animation as below.

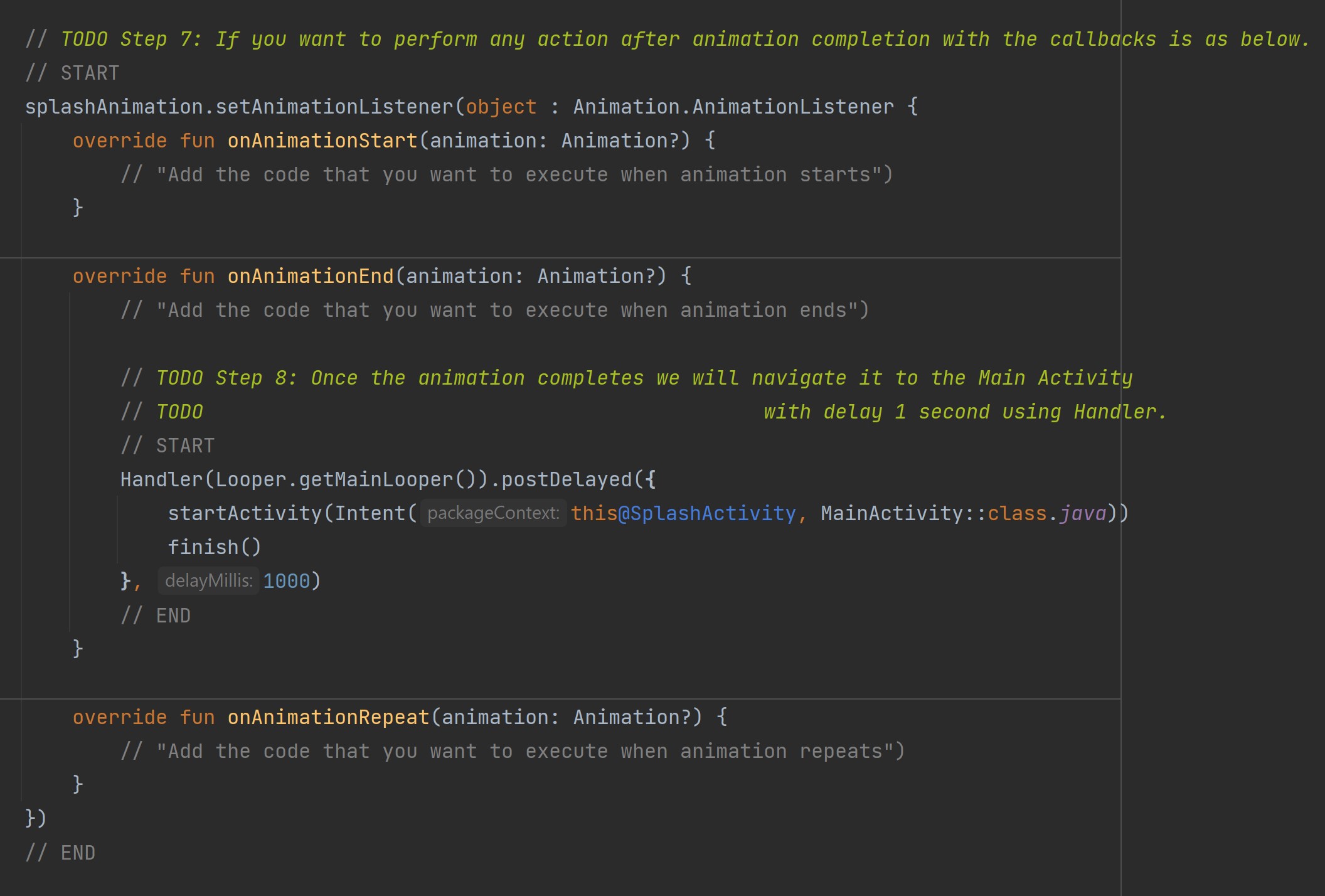
val splashAnimation = AnimationUtils.loadAnimation(this@SplashActivity, R.anim.*anim\_splash*)

1. Apply the animation to TextView

splashBinding.tvAppName.*animation* = splashAnimation

Note\*\*\* tvAppName is id of TextView in activity\_splash.xml layout

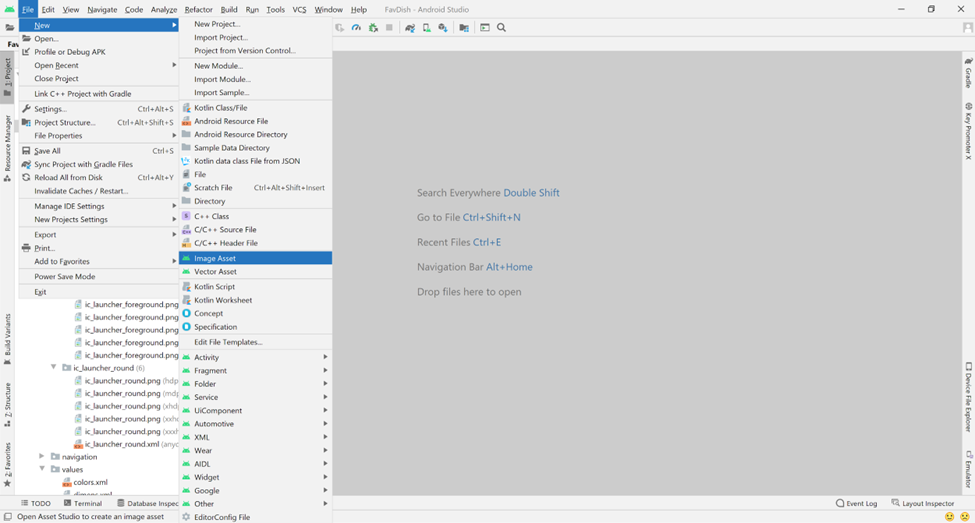
1. Once the animation completes we will navigate it to the Main Activity with delay 1 second using Handler. We will use setAnimationListener as below:

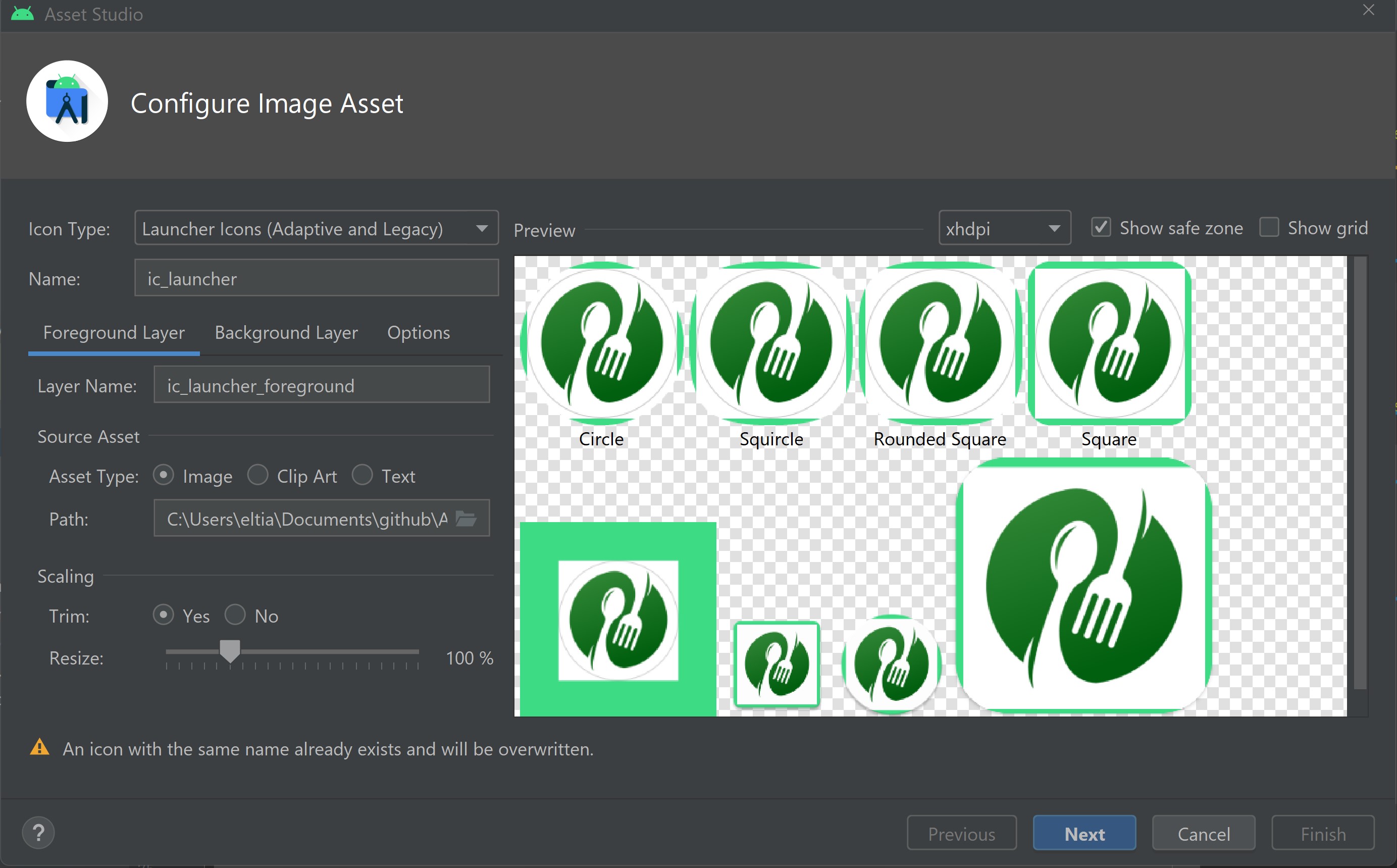


1. Run the application and see the changes.
2. You can get the animations online that you want or create your own as per your requirement. A reference link <https://www.raywenderlich.com/2785491-android-animation-tutorial-with-kotlin#toc-anchor-001>.
3. Next, we are going to create app icon using Android Studio.

First, download an icon you would like to use as your app icon.

Second, click anywhere in the Project and then go to **File -> New -> Image Asset**.





Third, write the path and select the image you want to use.

Forth, you can choose to trim it or no. In this project we chose trim.

Fifth, go to Background Layer and choose color white “FFFFFF”. You can also use another image as background, but in this project we will choose only color white.

Sixth, go to Next and then Finish.

1. You will have the below app icon.

