

Android Programming

-Dr. Nimisha Ghosh

Department of Computer Science and Information Technology

Overview of Android Apps

- ▶ Android apps are built as a combination of components that can be invoked individually.
- ▶ For example, an activity is a type of app component that provides a user interface.
- ▶ The "main" activity is what starts when the user taps the app icon.
- ▶ We can also take the user straight into a different activity from other places, such as from a notification or even from a different app.
- ▶ Android allows us to provide different resources for different devices.
- ▶ For example, we can create different layouts for different screen sizes. Then the system determines which layout to use based on the current device's screen size.

Disclaimer: All the slides in this presentation have been prepared from

<https://codelabs.developers.google.com/codelabs/>

Application Fundamentals

- ▶ Android apps can be written using Kotlin, Java, and C++ languages.
- ▶ App components are the essential building blocks of an Android app.
- ▶ There are four different types of app components:
 1. Activities
 2. Services
 3. Broadcast receivers
 4. Content providers

Activities

- ▶ An *activity* is the entry point for interacting with the user.
- ▶ It represents a single screen with a user interface.
- ▶ For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.
- ▶ Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others.
- ▶ As such, a different app can start any one of these activities if the email app allows it. For example, a camera app can start the activity in the email app that composes new mail to allow the user to share a picture.

Activities (contd...)

- ▶ An activity facilitates the following key interactions between system and app:
 1. Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity.
 2. Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around.
 3. Helping the app handle having its process killed so the user can return to activities with their previous state restored.
 4. Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (The most classic example here being share.)

Services

- ▶ A *service* is a general-purpose entry point for keeping an app running in the background for all kinds of reasons.
- ▶ It is a component that runs in the background to perform long-running operations or to perform work for remote processes.
- ▶ A service does not provide a user interface.
- ▶ For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity.
- ▶ Syncing data in the background or playing music also represent two different types of started services that modify how the system handles them.
- ▶ Music playback is something the user is directly aware of, so the app tells the system this by saying it wants to be foreground with a notification to tell the user about it; in this case the system knows that it should try really hard to keep that service's process running, because the user will be unhappy if it goes away.
- ▶ A regular background service is not something the user is directly aware as running, so the system has more freedom in managing its process. It may allow it to be killed (and then restarting the service sometime later) if it needs RAM for things that are of more immediate concern to the user.

Broadcast receivers

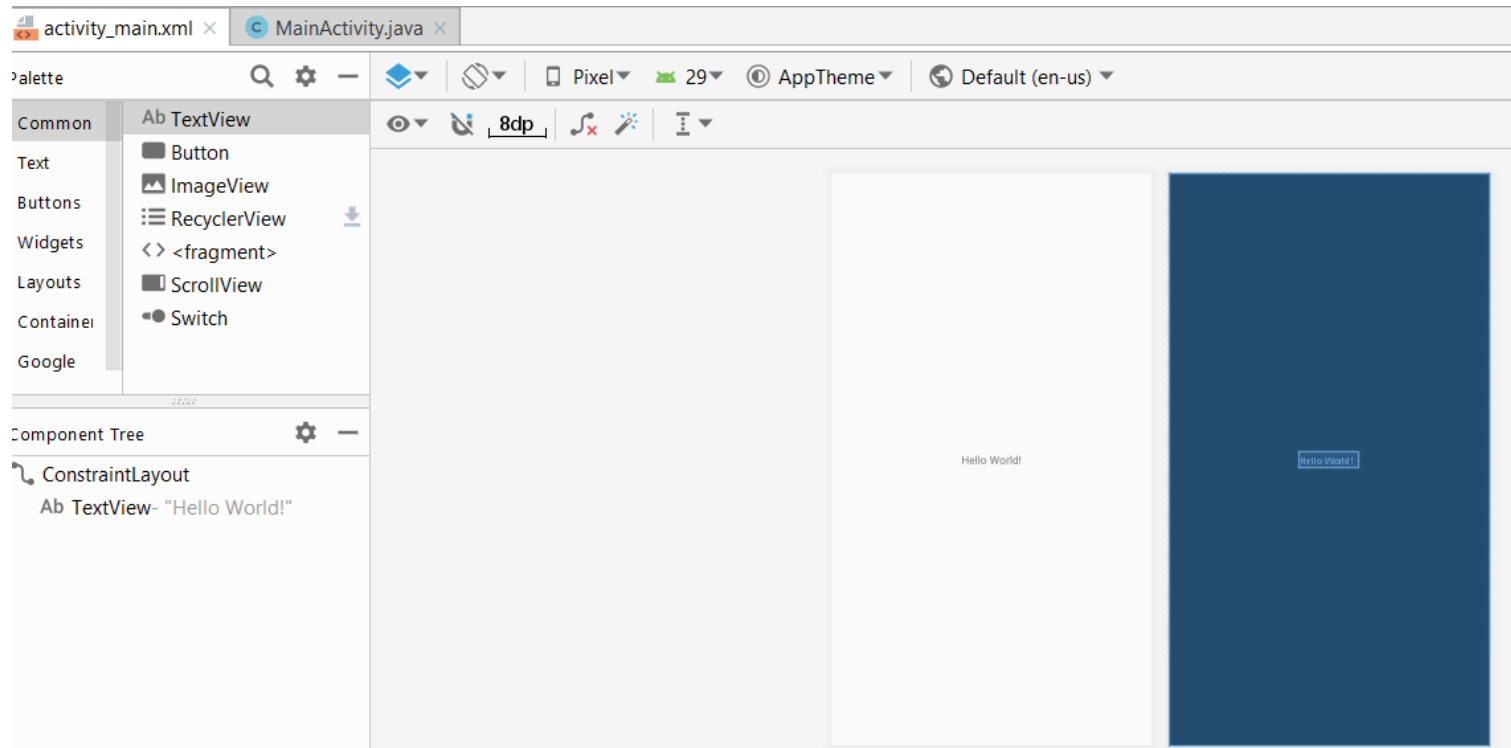
- ▶ A *broadcast receiver* is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements.
- ▶ Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running.
- ▶ So, for example, an app can schedule an alarm to post a notification to tell the user about an upcoming event and by delivering that alarm to a `BroadcastReceiver` of the app, there is no need for the app to remain running until the alarm goes off.
- ▶ Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.
- ▶ Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use.
- ▶ A broadcast receiver is implemented as a subclass of `BroadcastReceiver` and each broadcast is delivered as an *Intent* object.

Content providers


- ▶ A *content provider* manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access.
- ▶ Through the content provider, other apps can query or modify the data if the content provider allows it. For example, the Android system provides a content provider that manages the user's contact information.
- ▶ To the system, a content provider is an entry point into an app for publishing named data items, identified by a URI scheme.
- ▶ Content providers are also useful for reading and writing data that is private to your app and not shared.
- ▶ A content provider is implemented as a subclass of `ContentProvider` and must implement a standard set of APIs that enable other apps to perform transactions.

Create the first app

- Step 1: In the Android Studio editor, follow these steps:
1. Click the *activity_main.xml* tab to see the layout editor.
 2. Click the layout editor Design tab to show a graphical rendition of the layout as shown below.



- Step 3: Click the *MainActivity.java* tab to see the code editor as shown below:

A screenshot of the Android Studio code editor showing the MainActivity.java file. The editor has two tabs at the top: 'activity_main.xml' and 'MainActivity.java'. The code is as follows:

```
1 package com.example.myapplication;
2
3 import ...
4
5
6
7 public class MainActivity extends AppCompatActivity {
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.activity_main);
13    }
14 }
15
```

- Step 4: Create an Android virtual device (AVD)
1. In Android Studio, select **Tools > Android > AVD Manager**.
 2. Click the **+Create Virtual Device**.
 3. Choose a device and click **Next**. The **System Image** screen appears.
 4. Click the **Recommended** tab if it is not already selected, and choose which version of the Android system to run on the virtual device (such as **Oreo**).

- Step 5: Run the app on the virtual device.



► Step 6: Turn on USB debugging

1. To let Android Studio communicate with your device, you must turn on USB Debugging on your Android device. This is enabled in the Developer options settings of your device.
2. On Android 4.2 and higher, the Developer options screen is hidden by default. To show developer options and enable USB Debugging:
3. On your device, open Settings, search for About phone, click on About phone, and tap Build number seven times.
4. Return to the previous screen (Settings / System). Developer options appears in the list. Tap Developer options.
5. Choose USB Debugging.

► Step 7: Run your app on a device

1. Connect your device to your development machine with a USB cable.
2. Click the **Run** button. The **Select Deployment Target** window opens with the list of available emulators and connected devices.
3. Select your device, and click **OK**.

Interactive UI

- ▶ In this app, we will try to have some idea about TextView, Buttons and Toast.
- ▶ Step 1: Add a Button to the layout:

```
<Button
    android:id="@+id/button_toast"
    style="buttonstyle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimary"
    android:text="Toast"
    android:textColor="@android:color/white"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:layout_editor_absoluteY="49dp"
    android:onClick="showToast"/>
```

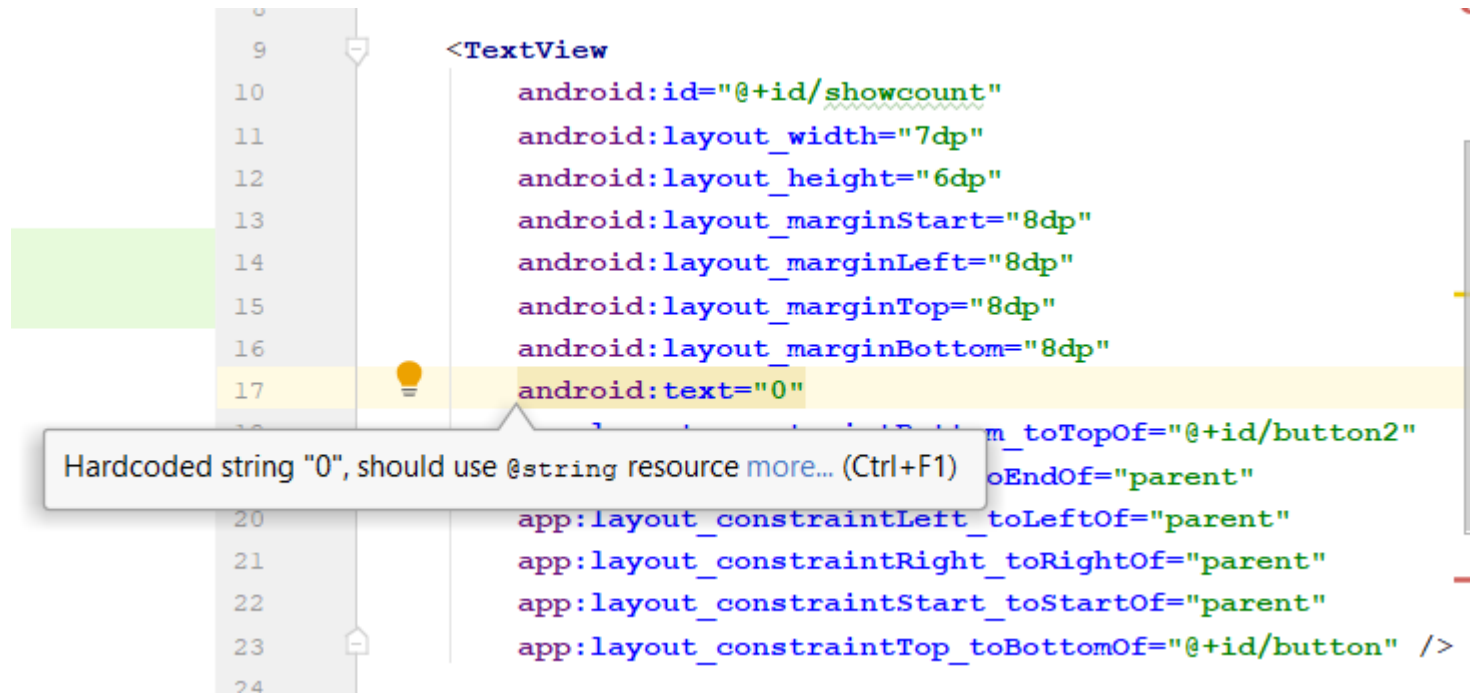
► Step 2: Add the second button

```
<Button
    android:id="@+id/button2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Button"
    tools:layout_editor_absoluteX="143dp"
    tools:layout_editor_absoluteY="548dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    android:onClick="countup"/>
```

► Step 3: Add TextView

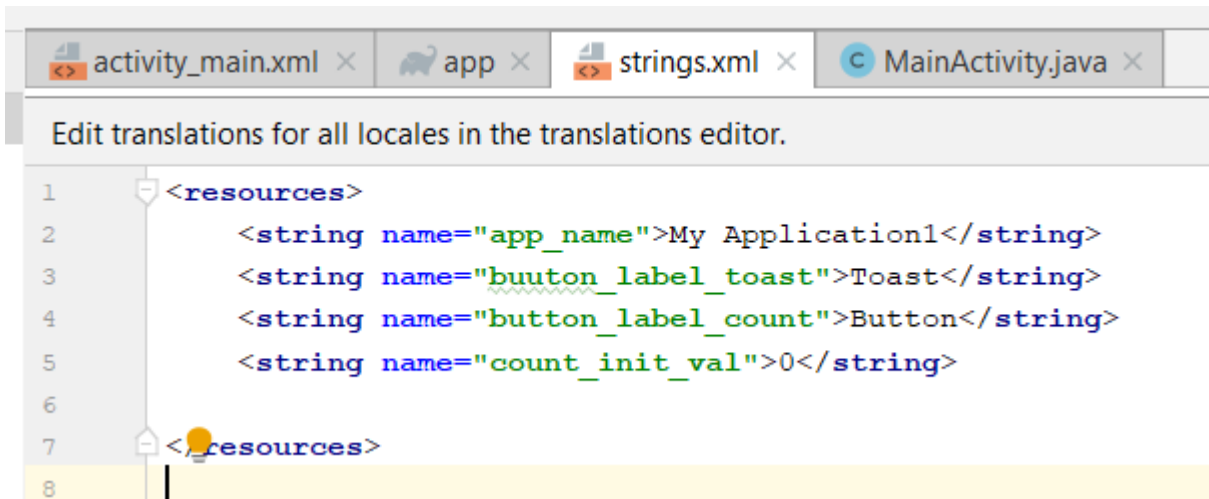
```
<TextView
    android:id="@+id/showcount"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:text="0"
    android:textSize="160sp"
    app:layout_constraintBottom_toTopOf="@+id/button2"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="1.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button_toast"
    app:layout_constraintVertical_bias="0.553" />
```

- ▶ After you are done with the aforementioned parts, you can see that an exclamation point appears next to each UI element in the Component Tree.
- ▶ Instead of hard-coding strings, it is a best practice to use string resources, which represent the strings.



- ▶ Step 4: Press **Alt-Enter** in Windows and choose **Extract string resource** from the popup menu.

- Step 5: Open strings.xml



The screenshot shows an IDE window with four tabs: activity_main.xml, app, strings.xml, and MainActivity.java. The strings.xml tab is active, displaying the XML content for editing translations. The text "Edit translations for all locales in the translations editor." is visible at the top of the editor. The XML code is as follows:

```
1 <resources>
2     <string name="app_name">My Application1</string>
3     <string name="buutton_label_toast">Toast</string>
4     <string name="button_label_count">Button</string>
5     <string name="count_init_val">0</string>
6
7 </resources>
8
```

- Step 6: `<string name = "toast_msg">Your toast message</string>`
- Step 7: Make changes in MainActivity.java

- The final app would look like this:

