Android App Development

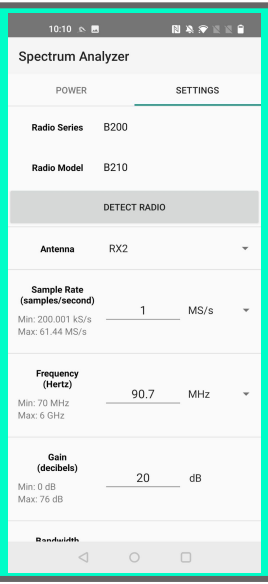
AGENDA

- Uses for mobile app development within CSD-TA
- Getting started with Android dev
  - Languages, tools/packages, Android studio
- How is an application structured?
  - Project file structure
  - Basic components of app development
    - GUI, Java, Native C++, XML layouts
- Demo: let’s make a simple calculator application
- Example application: Flux

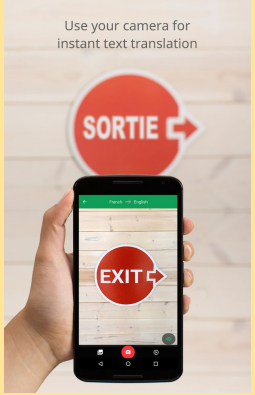
How is mobile app development relevant to CSD-TA as a whole?

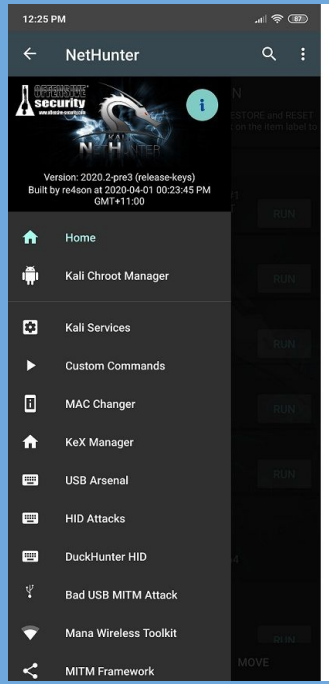
- Any tool or program that could benefit from a handheld GUI (such as for operational use by ECTs or others) could likely be made into an Android application
- Tools developed on Linux can be directly ported to run on an Android phone, provided some device setup
  - Example: a rooted Android phone with a Kali Linux container
  - If it can run on Kali, it can run on this type of phone
  - Run commands/programs directly in a terminal, or create a GUI to control your tool

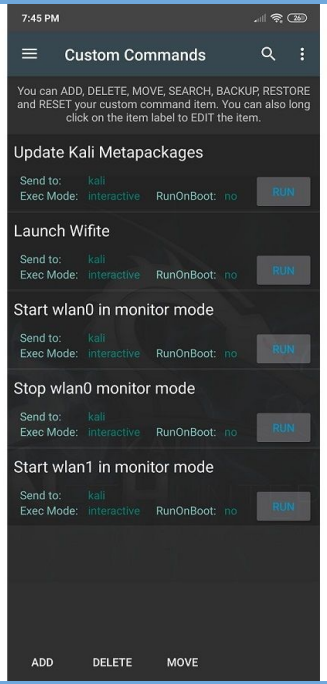
RF

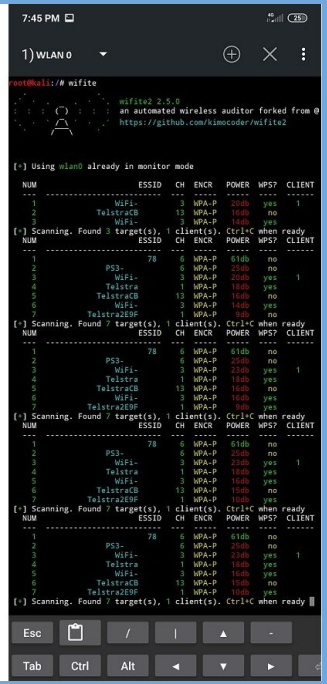


Data Science & Machine Learning









Pentesting

And many other uses…

Getting Started with Android Development

- Android Studio provides an easy to use IDE for application development, although not required
- Languages
  - Most common: Java, Kotlin, C/C++ (through Android Native Development Kit)
  - Others: C#, Python (not native, but Python apps can be converted into an Android package), JavaScript/CSS/HTML for hybrid apps (run on iOS too)
  - Resources and GUI layouts are in XML
- Android Debug Bridge (ADB) allows debugging over USB or Wifi, and you can easily open a shell on your device
  - https://developer.android.com/studio/command-line/adb

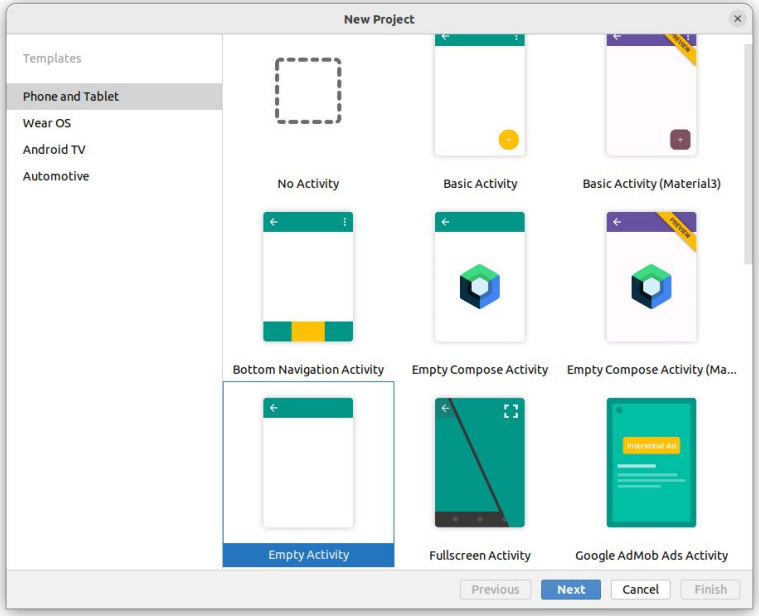
How is an Android application structured?

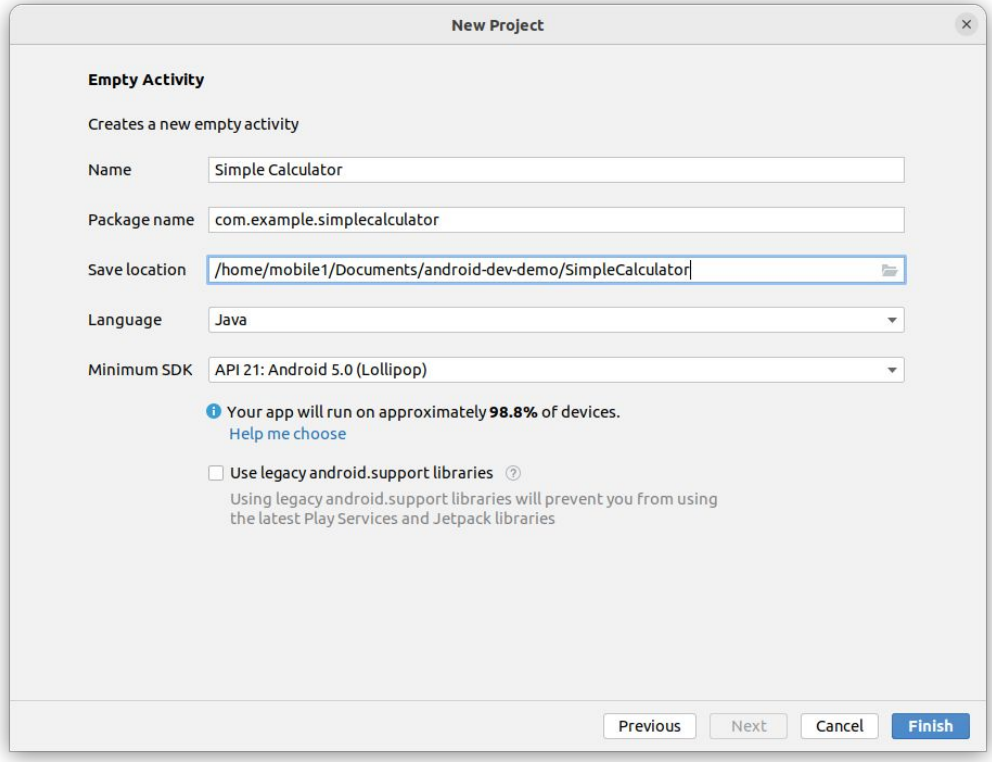
- Project file structure
- Basic components of app development
  - GUI, Java, Native C++, XML layouts

Demo Walkthrough

- Let's make a simple calculator application!
- I made the code available here:
  - https://github.com/elizabethmdrumm/simple-calculator-app
  - XML layout:
    - https://github.com/elizabethmdrumm/simple-calculator-app/blob/main/app/src/main/res/layout/activity_main.xml
  - Main Activity Java code:
    - https://github.com/elizabethmdrumm/simple-calculator-app/blob/main/app/src/main/java/com/example/simplecalculator/MainActivity.java

Create a project with an empty activity. Android Studio will auto-create a template for you.





Simple Calculator – AndroidManifest.xml [Simple_Calculator.app.main]

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help

SimpleCalculator › app › src › main › AndroidManifest.xml

app | OnePlus KB2005

Android

- app
  - manifests
    - AndroidManifest.xml
  - java
    - com.example.simplecalculator
      - MainActivity
        - onCreate(Bundle):void
    - com.example.simplecalculator (androidTest)
    - com.example.simplecalculator (test)
  - res
    - drawable
    - layout
      - activity_main.xml
    - mipmap
    - values
      - colors.xml
      - strings.xml
      - themes (2)
        - themes.xml
        - themes.xml (night)
    - xml
- Gradle Scripts

MainActivity.java | AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.simplecalculator">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Simple Calculator"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.SimpleCalculator"
        tools:targetApi="31">


        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

manifest

Text | Merged Manifest

The App Manifest

Here we can specify attributes such as:

- what Android API we are targeting. API level 31 is Android 12.
- what icon your app will have, as well as what style/theme it will use
- what permissions your app requires (not shown here)

Version Control | TODO | Problems | Terminal | Build | Logcat | Profiler | App Inspection | Event Log | Layout Inspector

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.simplecalculator">

```

- Every Android app has a unique application ID that looks like a Java package name, such as com.example.simplecalculator
- This ID uniquely identifies your app on the device, and in the Google Play Store if you intend to publish the app

What is an Activity?

```
17        <activity
18            android:name=".MainActivity"
19            android:exported="true">
20            <intent-filter>
21
22                <!--  This activity is the entry point for the app. -->
23                <action android:name="android.intent.action.MAIN" />
24
25                <!--  This activity will be launched when the app opens/starts up. -->
26                <category android:name="android.intent.category.LAUNCHER" />
27            </intent-filter>
28        </activity>
```

An activity is a single, focused thing that a user can do.

- Almost all Activities interact with the user. The Activity class allows you to create a window for your UI (with setContentView(View)). We will look at this in more detail later.

What is an Intent?

```
17      <activity
18          android:name=".MainActivity"
19          android:exported="true">
20          <intent-filter>
21
22              <!--  This activity is the entry point for the app. -->
23              <action android:name="android.intent.action.MAIN" />
24
25              <!--  This activity will be launched when the app opens/starts up. -->
26              <category android:name="android.intent.category.LAUNCHER" />
27          </intent-filter>
28      </activity>
```

- An intent is an abstract description of an operation to be performed
- Example: it can be used with startActivity to launch an Activity

- The primary pieces of an intent are:
  - **action** -- The general action to be performed, such as ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc.
  - **category** -- optional
  - **data** -- The data to operate on, such as a person record in the contacts database, expressed as a Uri.
    - Although for this activity, we do not need data with our intent
- These elements combine to specify the type of intent to which your activity can respond.

```
17      <activity
18          android:name=".MainActivity"
19          android:exported="true">
20          <intent-filter>
21
22              <!--  This activity is the entry point for the app. -->
23              <action android:name="android.intent.action.MAIN" />
24
25              <!--  This activity will be launched when the app opens/starts up. -->
26              <category android:name="android.intent.category.LAUNCHER" />
27          </intent-filter>
28      </activity>
```

- <activity> tags make the named activity callable within the application with the Context.startActivity() function.
- Two tags here work together to designate MainActivity as our default entry point into the app:
  - “android.intent.action.MAIN”
  - “android.intent.category.LAUNCHER”
  - Meaning, this activity will be launched on app startup.

Simple Calculator – activity_main.xml [Simple_Calculator.app.main]

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help

SimpleCalculator › app › src › main › res › layout › activity_main.xml

app | OnePlus KB2005

Android

- app
  - manifests
  - java
    - com.example.simplecalculator
      - MainActivity
        - onCreate(Bundle):void
    - com.example.simplecalculator (and
    - com.example.simplecalculator (test
  - res
    - drawable
    - layout
      - activity_main.xml
    - mipmap
    - values
      - colors.xml
      - strings.xml
      - themes (2)
    - xml
  - Gradle Scripts

activity_main.xml | MainActivity.java

Code | Split | Design

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns: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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_main.xml | Pixel | 33 | SimpleCalculator | Default (en-us)

0dp

Hello World!

Hello World!

The UI

- This is the XML layout that will be loaded for the Main Activity. Right now, it contains one TextView which simply displays a string.
- You can work on either the raw XML in the Code tab, or you can drag and drop widgets and set different parameters in the Design tab. Or both.

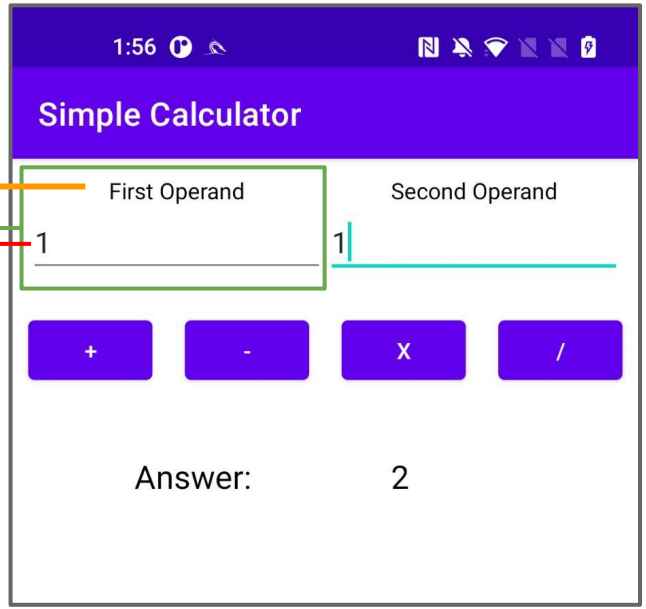
Version Control | TODO | Problems | Terminal | Build | Logcat | Profiler | App Inspection | Event Log | Layout Inspector

Gradle sync finished in 1 m 0 s 223 ms (3 minutes ago)

20:1 LF UTF-8 4 spaces

UI components can be nested

```
<!--  Contains the two numbers to be used in calculations  -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:orientation="horizontal"
        android:weightSum="100">

<!--    Number A    -->
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical"
            android:layout_weight="50">

            <TextView
                android:id="@+id/label_number_a"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:gravity="center_horizontal"
                android:textColor="@color/black"
                android:text="@string/label_number_a" />

            <!--     We will only allow signed integer input for this example      -->
            <EditText
                android:id="@+id/editText_number_a"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:ems="10"
                android:inputType="numberSigned"
                />
        </LinearLayout>

<!--    Number B    -->
        <LinearLayout...>

    </LinearLayout>
```



We are going to use some basic Buttons, TextViews, and EditTexts in this demo.

UI components can be customized

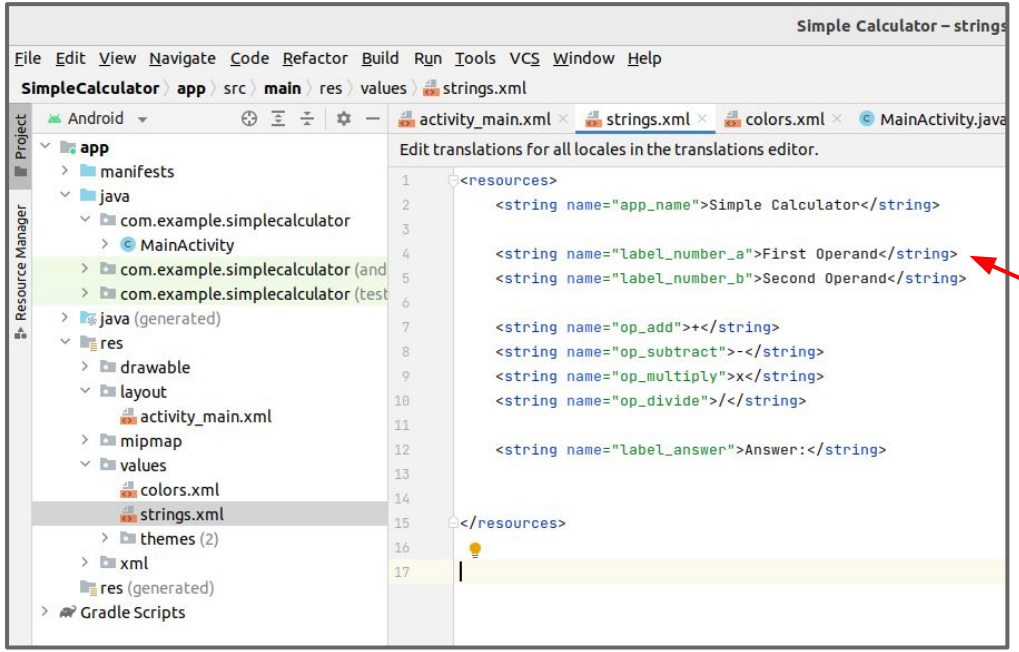
```
<TextView
    android:id="@+id/label_number_a"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:textColor="@color/black"
    android:text="@string/label_number_a" />
```

A TextView is used to show strings within the UI.

Widgets such as this TextView can be customized by setting their individual attributes.

In this example, we set the text color to black, and centered the text within the space allotted for this TextView.

Resources such as strings, colors, and themes are stored as XML files under the res folder of the app, and can be referenced throughout the application.




```
<!--  within activity_main.xml:    -->

<TextView
    android:id="@+id/label_number_a"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:textColor="@color/black"
    android:text="@string/label_number_a" />
```

Simple Calculator – MainActivity.java [Simp

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help

SimpleCalculator › app › src › main › java › com › example › simplecalculator › MainActivity

MainActivity.java

```java
/**
 * The package name here reflects the path from your app/src/main/java/ folder down to this file.
 */
package com.example.simplecalculator;

/**
 * AppCompatActivity is a base class for activities that wish to use some of the newer platform
 * features on older Android devices. This allows the app to be used on a wider variety of devices.
 */
import ...

/**
 * We are extending the functionality of the base Activity class.
 */
public class MainActivity extends AppCompatActivity {

    // This gets the class name as a string, and will be used for Log tags.
    private String TAG = getClass().getSimpleName();

    /**
     * A Context is the current state of the application/object, and is used to get information
     * about the application environment.
     *
     * Uses of Context: Load Resource Values, Start a Service, Bind to a Service, Send a Broadcast,
     * Register BroadcastReceiver.
     */
    private Context context = null;

    /** onCreate is where you initialize your activity. This method is called upon the activity's ...*/

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // This loads the XML layout that we created.
        // res/layout/activity_main.xml is pulled.
        setContentView(R.layout.activity_main);

        context = getApplicationContext();

        /**
```

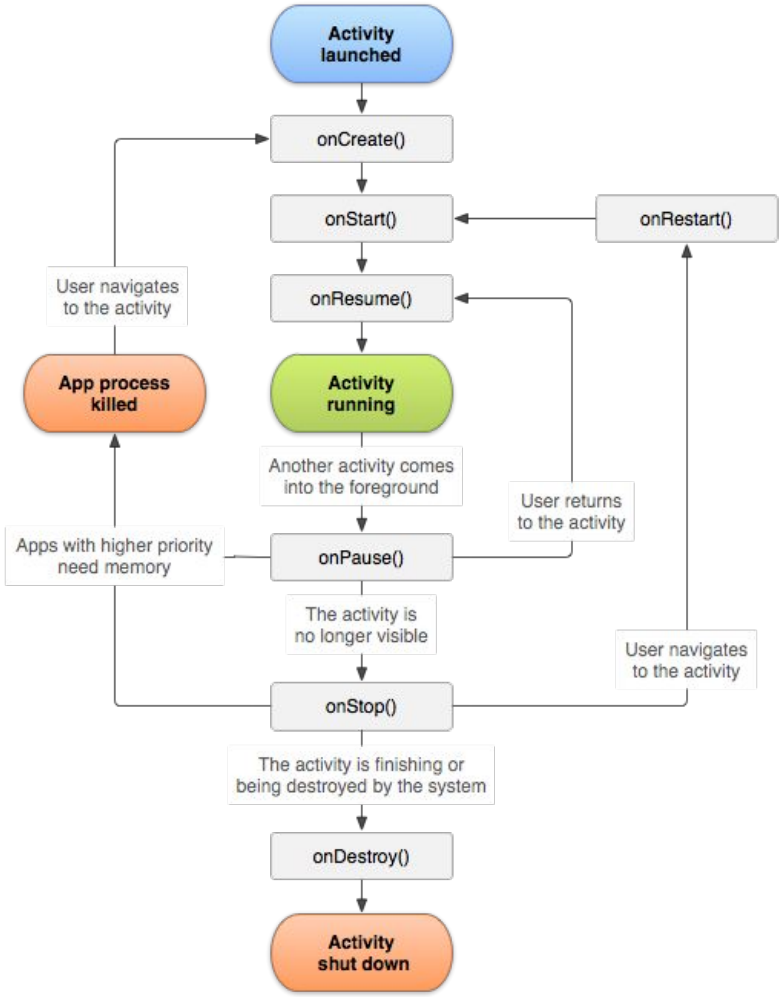
Version Control ▶ Run TODO Problems Terminal Build Logcat Profiler App Inspection

MainActivity Class

- This defines how our Activity will behave. We are using Java here, although Kotlin is also an option.
- An activity enters several different states in its lifecycle, and you use callbacks like onCreate to handle state transitions.
- <u>You must override and implement the onCreate function. onCreate runs automatically upon activity creation.</u> You need to call setContentView() to define what layout will be loaded for this activity.

Activity Lifecycle: Callbacks

- The Activity class provides six callbacks to navigate state transitions:
  - onCreate()
  - onStart()
  - onResume()
  - onPause()
  - onStop()
  - onDestroy()
- The system invokes each of these callbacks as an activity enters a new state.
- Good implementation of these callbacks is necessary to help avoid:
  - Crashing if the user switches to another app.
  - Consuming system resources when the user is not actively using it.
  - Losing the user's progress if they leave your app and return to it at a later time.



Simple Calculator MainActivity

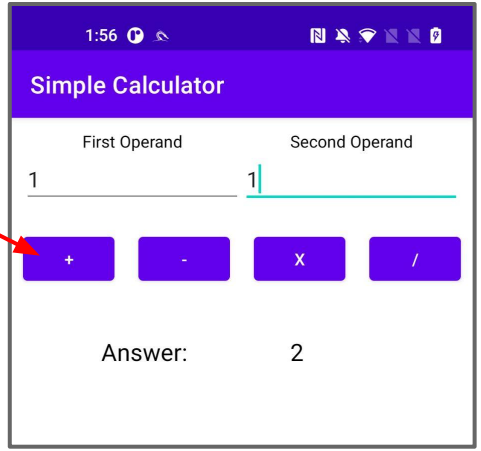
```
47      @Override
48      protected void onCreate(Bundle savedInstanceState) {
49          super.onCreate(savedInstanceState);
50
51          // This loads the XML layout that we created.
52          // res/layout/activity_main.xml is pulled.
53          setContentView(R.layout.activity_main);
54
55          context = getApplicationContext();
56
```

1. Set what layout we are using, in onCreate

Simple Calculator MainActivity

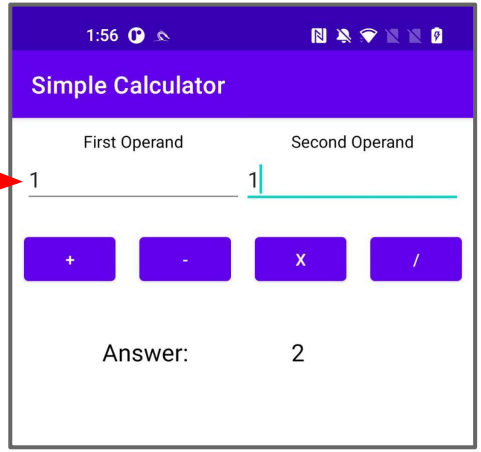
```java
/**
 * Set up button behavior.
 * Each time an operator button is pressed, it should pull the two operands from their fields,
 * perform the calculation, and place the result in its corresponding TextView.
 */

// Addition button.

// Find its corresponding UI widget in the XML file.
// Its name will be whatever ID we set for it in the XML attribute such as: android:id="@+id/button_add"
Button button_add = findViewById(R.id.button_add);
if (null == button_add)
{
    // Error.
    return;
}

// Set the behavior for the button click.
button_add.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view)
    {
        // Call add function when the button is clicked.
        // This will handle calculation, and updating the UI.
        add();
    }
});

// Do the same for the rest of the buttons...
```

2. Set up button behavior



Simple Calculator MainActivity

```java
146     // Retrieve the first operand from its EditText field.
147 @   private Integer get_num_a()
148     {
149         Integer answer = null;
150         int num_a = -1;
151
152         EditText edittext_num_a = findViewById(R.id.editText_number_a);
153         if (null == edittext_num_a)
154         {...}
158
159         // Grab the contents of the EditText as a string.
160         String num_a_str = edittext_num_a.getText().toString();
161
162         if (null == num_a_str)
163         {...}
167
168         // Convert the string to an integer.
169         try
170         {
171             num_a = Integer.parseInt(num_a_str);
172         }
173         catch (NumberFormatException nfe)
174         {
175             // Error. Was not a valid number.
176             return answer;
177         }
178
179         answer = new Integer(num_a);
180
181         return answer;
182     }
```



3. On button press, get whatever is currently in the EditText fields.

Simple Calculator MainActivity

```java
    // Wraps all action and behavior for addition button.
    private void add()
    {
        // Grab the current contents of the first operand's field.
        final Integer num_a = get_num_a();
        if (null == num_a)
        {
            // Set the answer field of the UI to reflect that an error occurred.
            set_error();
            return;
        }

        // Grab the second operand from its field.
        final Integer num_b = get_num_b();
        if (null == num_b)
        {...}

        int result = -1;

        try
        {
            result = Math.addExact(num_a.intValue(), num_b.intValue());
        }
        catch (Exception e)
        {
            set_error();
            return;
        }

        // Set the result in the answer field of the UI.
        set_result(result);
    }
```

4. Perform the selected calculation.

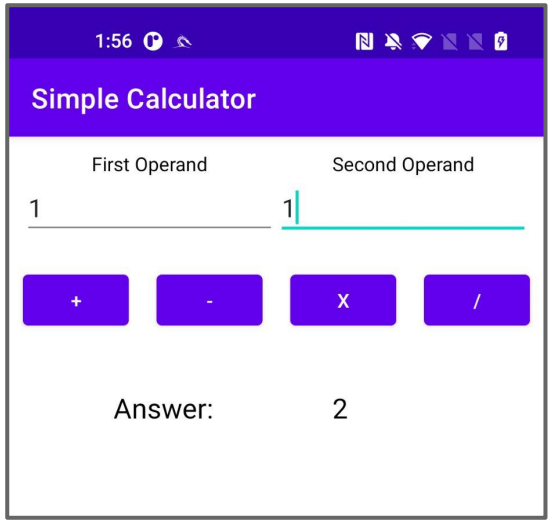
Simple Calculator MainActivity

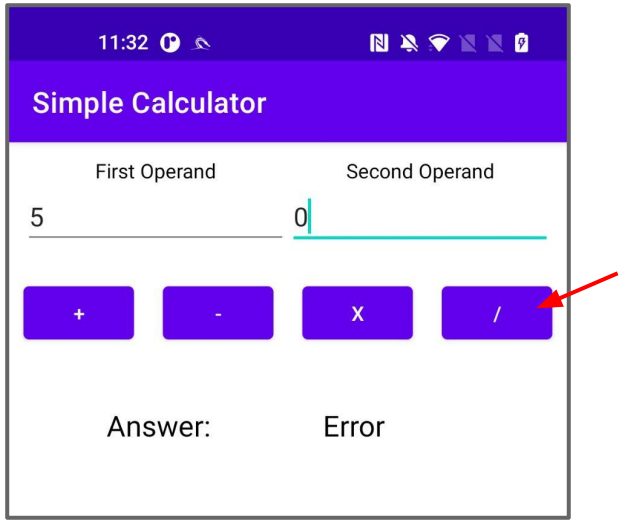
```java
    // Set the TextView to an integer, our answer.
    private void set_result(int answer)
    {
        // Find the result TextView.
        TextView result = findViewById(R.id.text_result);
        if (null == result)
        {
            // Error.
            return;
        }

        // Convert integer to string, and update the TextView.
        result.setText(String.valueOf(answer));

    }
```

```java
    // Updates the UI to reflect that an error in calculation occurred.
    private void set_error()
    {
        // Find the result TextView.
        TextView result = findViewById(R.id.text_result);
        if (null == result)
        {
            // Error.
            return;
        }

        // Convert integer to string, and update the TextView.
        result.setText("Error");

    }
```

5. Update the UI to show the answer, or that an error occurred.

Simple Calculator MainActivity





5. Update the UI to show the answer, or that an error occurred.

Simple Calculator – MainActivity.java [Simple_Calculator.app.main]

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help

SimpleCalculator › app › src › main › java › com › example › simplecalculator › MainActivity

app | OnePlus KB2005

Android

- app
  - manifests
    - AndroidManifest.xml
  - java
    - com.example.simplecalculator
      - MainActivity
        - add():void
        - divide():void
        - get_num_a():Integer
        - get_num_b():Integer
        - multiply():void
        - onCreate(Bundle):void
        - set_error():void
        - set_result(int):void
        - subtract():void
        - context:Context
        - TAG:String
    - com.example.simplecalculator (a
    - com.example.simplecalculator (te
  - java (generated)
  - res
  - res (generated)
- Gradle Scripts

MainActivity.java

```
/**
 * The package name here reflects the path from your app/src/main/java/ folder down to this file.
 */
package com.example.simplecalculator;

/**
 * AppCompatActivity is a base class for activities that wish to use some of the newer platform
 * features on older Android devices. This allows the app to be used on a wider variety of devices.
 */
import ...

/**
 * We are extending the functionality of the base Activity class.
 */
public class MainActivity extends AppCompatActivity {

    // This gets the class name as a string, and will be used for Log tags.
    private String TAG = getClass().getSimpleName();

    /**
     * A Context is the current state of the application/object, and is used to get info
     * about the application environment.
     *
```

Build: Sync | Build Output | Build Analyzer

Build SimpleCalculator: finished At 9/9/22, 11:32 AM — 1 sec, 355 ms

```
> Task :app:writeDebugAppMetadata UP-TO-DATE
> Task :app:writeDebugSigningConfigVersions UP-TO-DATE
> Task :app:processDebugManifestForPackage
> Task :app:processDebugResources
> Task :app:compileDebugJavaWithJavac
> Task :app:dexBuilderDebug
> Task :app:mergeProjectDexDebug
> Task :app:packageDebug
> Task :app:createDebugApkListingFileRedirect UP-TO-DATE
> Task :app:assembleDebug

BUILD SUCCESSFUL in 1s
31 actionable tasks: 8 executed, 23 up-to-date

Build Analyzer results available
```

Android Studio makes it easier to build and deploy your application onto your device. Just hit the play button.

- It builds an Android Package (.apk) file for your application that is then installed onto your device.
- You can interact with your device through a USB connection, or through Wifi.

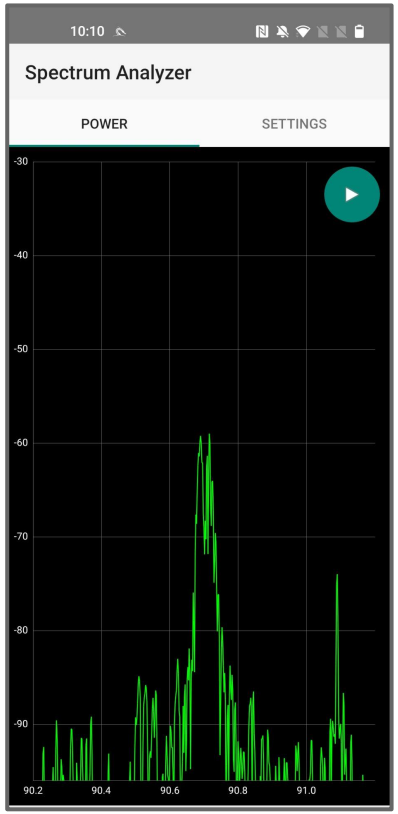
Version Control | Run | TODO | Problems | Terminal | Build | Logcat | Profiler | App Inspection | Event Log | Layout Inspector

Run selected configuration

289:1 LF UTF-8 4 spaces

Other uses for app development: RF

- Flux
  - Is an Android application we are working on, intended to provide a GUI for interfacing with SDRs such as our Ettus Research B200 series radios.
  - Current scope:
    - Spectrum Analyzer
      - Power v. Frequency Plot
      - Waterfall Plot
      - Signal Record
    - Signal Transmit
      - from an IQ file
      - control transmit from custom binary/Python file
    - Inspection of a previously recorded signal
      - scroll through IQ file manually
      - visual-based playback against real time



https://webcast.airdroid.com/

Resources

- Android Development:
  - https://developer.android.com/
- Simple Calculator example:
  - https://github.com/elizabethmdrumm/simple-calculator-app
- Using a Kali container on a rooted phone:
  - https://www.kali.org/docs/nethunter/
  - https://gitlab.com/kalilinux/nethunter/apps/kali-nethunter-app