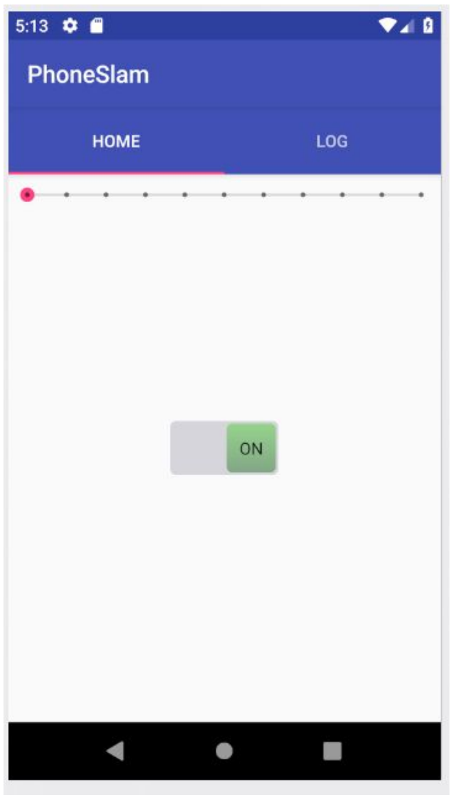
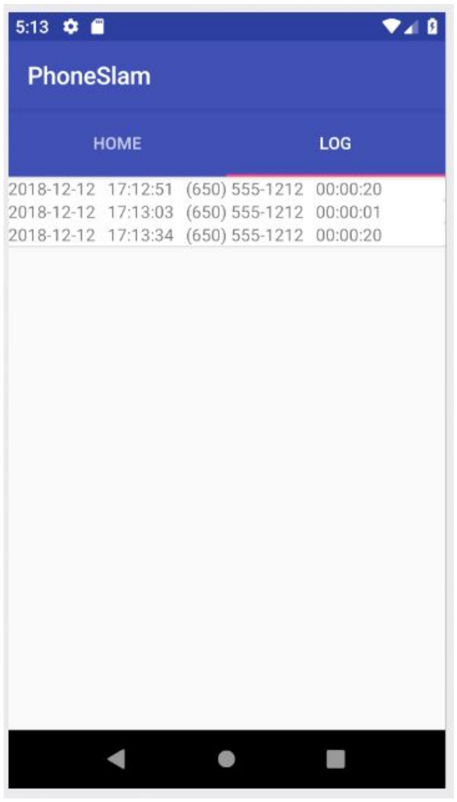
Imagine getting into a heated conversation with someone over the phone, and it gets to the point where you cannot stand hearing the other person's voice. Back when everyone used landline phones, this situation would be resolved by slamming the phone back into the receiver. Since the advent of smartphones, this action has been rendered impossible to achieve. The goal of our app is to recreate these moments. PhoneSlam is an app that hangs up a call when the user slams their phone (or quickly accelerates it).

PhoneSlam uses a minimum SDK version of 21, and its target SDK version is 28. It also uses two additional module dependencies: recyclerview-v7:28.0.0, and cardview-v7:28.0.0 . During testing, a Pixel API 28 virtual device was used. Upon launching the app, the user will be presenting with a simple layout containing a centralized switch, and a seek bar, both on the first page of a tab view. The user may swipe to another tab which contains a RecyclerView that holds updated information on the app as it is used.

Some images of the app in action:





```
2018-12-12  17:12:51  (650) 555-1212  00:00:20
2018-12-12  17:13:03  (650) 555-1212  00:00:01
2018-12-12  17:13:34  (650) 555-1212  00:00:20
```

To use the app, the user simply needs to toggle the switch to the "ON" state. If they so choose, they may also adjust the seek bar to control the sensitivity of the app. While the switch is on, any incoming or outgoing phone call will be subject to being tampered with by the app. If the user accelerates beyond a predetermined threshold (adjusted by the seek bar sensitivity value), the app will activate, and the call will terminate. Data is pulled from the call including the current date, current time, the phone number, and the call duration. All of this information is compiled into a single TextView and placed into the RecyclerView on the Log tab.

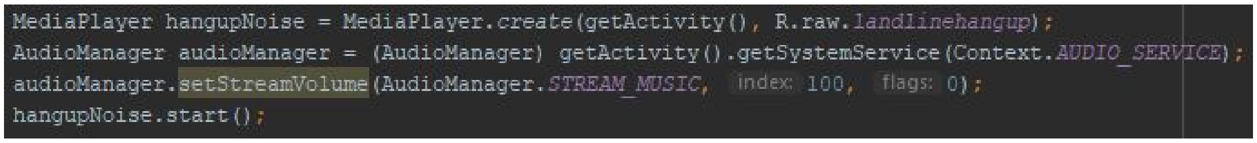
The app's layout is fairly simplistic, but most of its functionalities occur behind the scenes. There is a single activity that uses a Pager Adapter to allow users to swipe back and forth between two fragments: one that holds the switch and seek bar which both control how the app behaves, and another that holds the RecyclerView that displays information on calls that the app has already terminated. As for use cases, the app works on any type of phone call, either sent by the user or received by the user. The app goes through several processes in order to achieve its functionality:

- Upon activating the switch, a custom BroadcastReceiver is turned on. This class listens to the phone's state, specifically whenever the phone is off the hook. If the phone goes off the hook while the BroadcastReceiver is listening for it, an Accelerometer Sensor is registered.
- The activation of the Accelerometer Sensor hides inside a run method running inside a Handler. For the sake of battery preservation, the Sensor will only activate once the BroadcastReceiver has sent a notice to the app that it has detected a call. The Sensor receives events as the phone is accelerated, and once it receives an event that surpasses a predetermined threshold, it calls the TelecomManager to hang up the call.
- The TelecomManager is retrieved as a system service and provides methods that control the current call. Through a simple call of the endCall method, the foreground call on the device is ended.
- Once the call has ended, a sound from an mp3 audio file is played using the MediaPlayer and AudioManager classes. Additionally, the phone number of the other end of the call, and the call duration are retrieving using a Cursor that queries the CallLog.Calls class. This information along with the current date and time are stored into a String and placed into a Singleton ArrayList to store in the backend. The contents of this ArrayList are updated as individual TextViews in the RecyclerView.

Although the app mostly works as intended, there are a few knowns issues and bugs that we have not quite had the time to resolve. Firstly, in order for the app to be able to hang up a call, that calls needs to have been accepted after the switch has been turned on. This is because of how the custom BroadcastReceiver was implemented; it "misses" the call while the switch is turned off and thus doesn't work properly. Another issue that was faced, while not necessarily being a bug, is that the permissions to access the phone and call log need to be set manually in the app settings on the phone/emulator itself. Simply including the permissions in the Manifest was not enough.

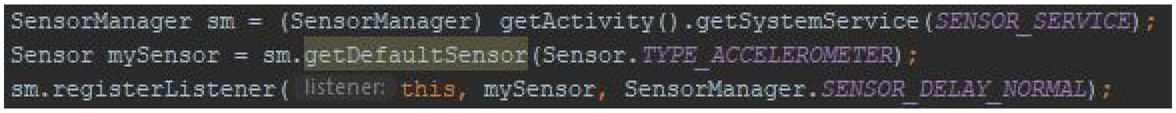
A functionality that were originally going to be included is how the volume of the sound played was going to be adjusted according to the acceleration the phone experienced as it hung up. We were not able to include this feature because any acceleration value that was beyond the threshold would trigger the condition to end the call. As the CallLog needs a bit of time after the call is ended to properly update, the thread is put to sleep for a second. Thus any information after the triggered threshold is effectively disregarded. One bug that was resolved was how the RecyclerView was not properly updating its contents. This was resolved by manually changing the data in the adapter for the RecyclerView then calling notifyDataSetChanged() on it.

In order to get the app to work as desired, several classes and features not covered throughout the course had to be explored. The following section covers such features in further detail.

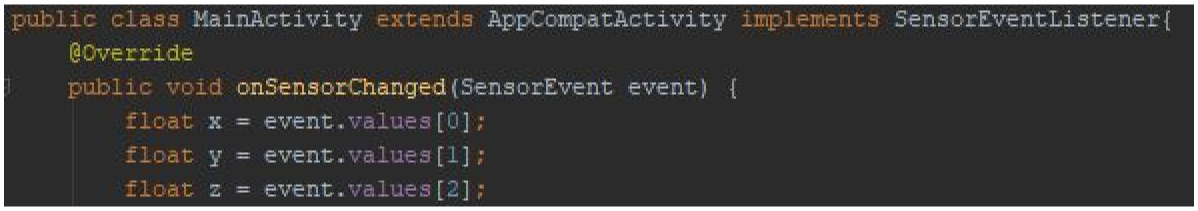


```
MediaPlayer hangupNoise = MediaPlayer.create(getActivity(), R.raw.landlinehangup);
AudioManager audioManager = (AudioManager) getActivity().getSystemService(Context.AUDIO_SERVICE);
audioManager.setStreamVolume(AudioManager.STREAM_MUSIC,  index: 100,  flags: 0);
hangupNoise.start();
```

A noise that sounds like an old phone being hung up is played upon the acceleration threshold being passed by use of a MediaPlayer. The mp3 file of the desired sound is stored in app/res/raw/ and is played with the start() function being called on the MediaPlayer. An AudioManager is used to set the volume of the outputted sound, using setStreamVolume() with the index:100 being the desired volume. Had the implementation of an acceleration-controlled volume been successful, this is the value that would have been adjusted.

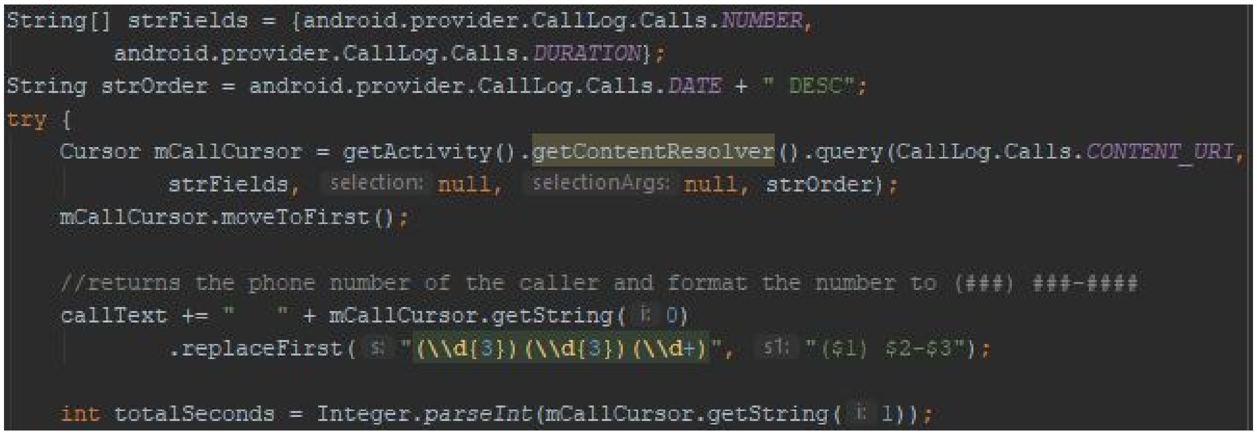


```
SensorManager sm = (SensorManager) getActivity().getSystemService(SENSOR_SERVICE);
Sensor mySensor = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sm.registerListener( listener: this, mySensor, SensorManager.SENSOR_DELAY_NORMAL);
```



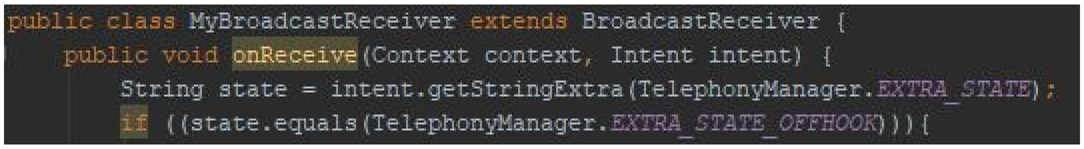
```
public class MainActivity extends AppCompatActivity implements SensorEventListener{
    @Override
    public void onSensorChanged(SensorEvent event) {
        float x = event.values[0];
        float y = event.values[1];
        float z = event.values[2];
```

To measure the acceleration the phone experiences the accelerometer is used. This is accessed by using the SensorManager class then getting a Sensor of type accelerometer. A class that implements SensorEventListener is then registered as the listener of the sensor. Whenever the accelerometer measures a changed acceleration, onSensorChanged(event) is called in the listener. Within the event is an array of floats called values that correspond to the x, y and z directions.



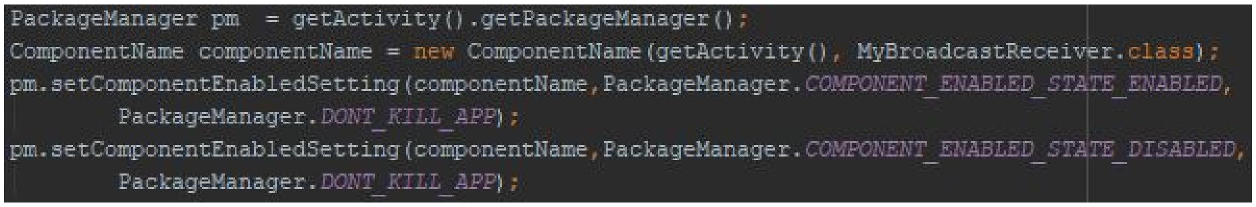
```
String[] strFields = {android.provider.CallLog.Calls.NUMBER,
        android.provider.CallLog.Calls.DURATION};
String strOrder = android.provider.CallLog.Calls.DATE + " DESC";
try {
    Cursor mCallCursor = getActivity().getContentResolver().query(CallLog.Calls.CONTENT_URI,
            strFields, selection: null, selectionArgs: null, strOrder);
    mCallCursor.moveToFirst();

    //returns the phone number of the caller and format the number to (###) ###-####
    callText += "    " + mCallCursor.getString( i: 0)
            .replaceFirst( s: "(\\d{3})(\\d{3})(\\d+)", s1: "($1) $2-$3");

    int totalSeconds = Integer.parseInt(mCallCursor.getString( i: 1));
```

CallLog.Calls is a class that contains information on recent calls. A String array “strFields” is defined in order to spy on the Android Call Log Content Provider, and it is specified that it is the phone number and call duration that are being looked for. A cursor is used to query the content URI of CallLog.Calls, and is able to return the phone number with getString(0), and the call duration in seconds using getString(1).



```
public class MyBroadcastReceiver extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        String state = intent.getStringExtra(TelephonyManager.EXTRA_STATE);
        if ((state.equals(TelephonyManager.EXTRA_STATE_OFFHOOK))){
```

The phone state is kept track of using a BroadcastReceiver, which receives any intents sent out to the phone. For example, when a phone call is received by your phone, it sends out an intent containing relevant information.   In this case we look in the intent for a string with key related to extra state, which indicates that a new call state has occured. It is then checked if it’s an offhook extra state, which means a new call has taken place.

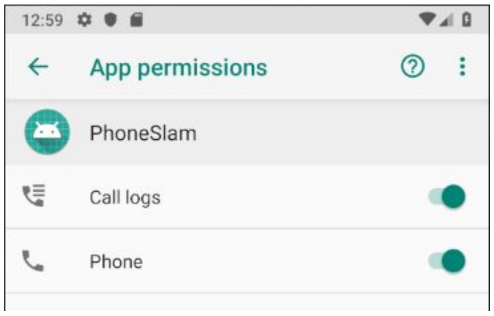


```
PackageManager pm  = getActivity().getPackageManager();
ComponentName componentName = new ComponentName(getActivity(), MyBroadcastReceiver.class);
pm.setComponentEnabledSetting(componentName,PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
        PackageManager.DONT_KILL_APP);
pm.setComponentEnabledSetting(componentName,PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
        PackageManager.DONT_KILL_APP);
```

By default our BroadcastReceiver is off as indicated by the android:enabled=”false” attribute it has in the manifest. The switch on the main page is used to turn the BroadcastReceiver on and off, by way of a PackageManager. The PackageManager gets the ComponentName for the BroadcastReceiver then uses setComponentEnabledSetting() to turn it on and off based upon the switch’s state.

```
TelecomManager tm = (TelecomManager) this.getSystemService(Context.TELECOM_SERVICE);
tm.endCall();
```

The TelecomManager provides access to information on active calls and some functionality to manage said calls. In a simple two lines of code, a reference to the TelecomManager is retrieved, then the endCall() method is called to terminate the active call.


12:59

App permissions

PhoneSlam

Call logs ●

Phone ●

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ANSWER_PHONE_CALLS" />
<uses-permission android:name="android.permission.READ_CALL_LOG" />
```

In order to successfully use the BroadcastReceiver to listen for when the phone is off the hook, the READ_PHONE_STATE permission must be included in the Manifest. For the TelecomManager to hang up the outgoing call, the ANSWER_PHONE_CALLS permission is required. Finally, in order to query CallLog.Calls, the READ_CALL_LOG permission is needed. Permissions to access the Call logs and the Phone must also be manually set in the app settings on the device itself.

References

CallLog.Calls | Android Developers. (n.d.). Retrieved from https://developer.android.com/reference/android/provider/CallLog.Calls

Change Media volume in Android? (n.d.). Retrieved from https://stackoverflow.com/questions/4178989/change-media-volume-in-android

End call in android programmatically. (n.d.). Retrieved from https://stackoverflow.com/questions/18065144/end-call-in-android-programmatically/26162973

Faroque, O. (2015, October 19). Custom Switch (Like IOS) In Android Tutorial. Retrieved from https://androidtutorialmagic.wordpress.com/android-material-design-tutorial/custom-switch-like-ios-in-android-tutorial/

How to get the outgoing call duration in real time? (n.d.). Retrieved from https://stackoverflow.com/questions/16982894/how-to-get-the-outgoing-call-duration-in-real-time

How to update RecyclerView Adapter Data? (n.d.). Retrieved from https://stackoverflow.com/questions/31367599/how-to-update-recyclerview-adapter-data

Programmatically enable/disable Broadcastreceiver. (n.d.). Retrieved from https://stackoverflow.com/questions/24266039/programmatically-enable-disable-broadcastreceiver

TelecomManager | Android Developers. (n.d.). Retrieved from https://developer.android.com/reference/android/telecom/TelecomManager

TelephonyManager | Android Developers. (n.d.). Retrieved from https://developer.android.com/reference/android/telephony/TelephonyManager

Vintage 1976 Wheel Phone, Receiver Down, Hard, Slam[MP3]. (n.d.). Https://www.soundsnap.com: Tom Hutchings.