```java
/********************************************************************************************************************/
/*    BME 363 Biomedical Instrumentation Laboratory, Biomedical Engineering Program, University of Rhode Island      */
/*            Licensed under the Apache License, Version 2.0 <http://www.apache.org/licenses/LICENSE-2.0>            */
/*                Initiated by Cody Goldberg in January 2016; Modified by Ying Sun in December 2016                  */
/********************************************************************************************************************/

package edu.uri.egr.bme363lab.ui;
import android.Manifest;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ProgressBar;
import android.widget.TextView;
import com.tbruyelle.rxpermissions.RxPermissions;
import java.io.IOException;
import butterknife.Bind;
import butterknife.ButterKnife;
import butterknife.OnClick;
import edu.uri.egr.bme363lab.R;
import edu.uri.egr.bme363lab.RxBluetooth;
import edu.uri.egr.bme363lab.ui.dialog.DeviceListDialog;
import edu.uri.egr.hermes.manipulators.FileLog;
import timber.log.Timber;

public class MainActivity extends AppCompatActivity {
    /* Using the library ButterKnife (https://github.com/JakeWharton/butterknife) we can "Bind" our views directly
    into fields within our MainActivity. This is pretty much short hand. Without ButterKnife, we cannot do
    @Bind, or @OnClick.  We would have to use findViewById() - and that is a pain.                              */
    @Bind(R.id.toolbar)
    Toolbar mToolbar;
    @Bind(R.id.fab)
    FloatingActionButton mFab;
    @Bind(R.id.line_chart_original)
    ReplacingLineChartView mChartOriginal;
    @Bind(R.id.line_chart_transformed)
    ReplacingLineChartView mChartTransformed;
    @Bind(R.id.progressBar)
    ProgressBar progressBar;
    @Bind(R.id.progressText)
    TextView progressText;

    /* Define the fields we are going to use globally within the MainActivity. */
    private BluetoothSocket mSocket; // BluetoothSocket that contains bluetooth info to our PIC.
    private int currentFunction, topFunction = 10; // function code from PIC
    private FileLog originalLog;
    private FileLog transformedLog;
    private volatile boolean logWritable;
    private long logStartTime;
    private boolean logPermissionGranted;
    private GraphTransformController graphController;
    private boolean graphFrozen;

    /******** onCreate - runs when the activity is initiated, screen rotated, or a restore from being deleted ********/
    @Override       // overrides a method of its superclass
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); // Call super, otherwise Android can't perform back-end tasks to onCreate.
        setContentView(R.layout.activity_main); // Set our layout information.
        ButterKnife.bind(this); // Tell ButterKnife we want to bind the views we just made from setContentView.
        setSupportActionBar(mToolbar); // Set our toolbar to the one provided in our layout resource.
        mChartOriginal.setMaximumX(1024); // Prevent both charts from going beyond a 1024 point size in the X direction.
        mChartTransformed.setMaximumX(1024);
        // Add a graph controller to handle the viewport syncing between our two charts.
        graphController = new GraphTransformController(mChartOriginal, mChartTransformed);
        RxPermissions.getInstance(this)
        .request(Manifest.permission.WRITE_EXTERNAL_STORAGE)
        .subscribe(granted -> {
            logPermissionGranted = granted;
            if (granted) {
                originalLog = new FileLog("original.csv");
                transformedLog = new FileLog("transformed.csv");
            } else snackMessage("Unable to write logs without permission!");
        });
    }

    /** OnFabClicked - runs when FAB (R.id.fab) is clicked (via ButterKnife), opens DeviceListDialog to connect PIC **/
    @OnClick(R.id.fab)
    public void onFabClicked() {
        if (mSocket != null) {          // If already connected, this button becomes the freeze button.
            handleGraphFreezeState();
            return;
        }
        DeviceListDialog dialog = new DeviceListDialog();   // Create a new dialog object.
        dialog.getDevice().subscribe(this::deviceSelected); // "Subscribe" to the output of whatever we select.
        dialog.show(getFragmentManager(), "deviceList");    // Finally, show the dialog.
    }
```

```java
/********** handleGraphFreezeState - changes the FAB to show the proper action icon for graph freezing ***********/
private void handleGraphFreezeState() {
    if (!graphFrozen) mFab.setImageResource(R.drawable.ic_play_arrow_black_24dp);
    else mFab.setImageResource(R.drawable.ic_pause_black_24dp);
    graphFrozen = !graphFrozen;
}

/************************* switchFunction - displays the function received from the PIC *************************/
private void switchFunction(int function) {
    Timber.d("Switching to function %d.", function);    // Log for debugging.
    /* Android requires any manipulation of the views to be done on the UI Thread (also known as the Main Thread)
    Because this function is being called from the Bluetooth callbacks, we need to hop over to the UI thread by
    running runOnUiThread. */
    runOnUiThread(() -> {
            switch (function) {
            case 0:     setTitle("0: Binary Counter");         break;      // PIC function = 0
            case 1:     setTitle("1: ECG Simulation");         break;      // PIC function = 1
            case 2:     setTitle("2: Echo (A/D - D/A)");       break;      // PIC function = 2
            case 3:     setTitle("3: Variable Sampling");      break;      // PIC function = 3
            case 4:     setTitle("4: Derivative");             break;      // PIC function = 4
            case 5:     setTitle("5: Low-pass Filter");        break;      // PIC function = 5
            case 6:     setTitle("6: Hi-Freq Enhance");        break;      // PIC function = 6
            case 7:     setTitle("7: 60hz Notch Filter");      break;      // PIC function = 7
            case 8:     setTitle("8: Median Filter");          break;      // PIC function = 8
            case 9:     setTitle("9: Detect QRS: MOBD");       break;      // PIC function = 9
            case 10:    setTitle("10: Photoplethysmogram");     break;      // PIC function = 10
            default:    setTitle("Unknown Function");       break;
            }
        }
    );
    // updateLogs();       // Update the logs to match our current set function. Disable to avoid CVS files.
}

/********* updateLogs - updates the original and transformed logs to match the current function and time *********/
private void updateLogs() {
    if (!logPermissionGranted) return;
    logWritable = false; // Donn't write if we receive a byte stream faster than we can run this code.
    originalLog = new FileLog(String.format("%d (original) - %d.csv", currentFunction, System.currentTimeMillis()));
    transformedLog = new FileLog(String.format("%d (transformed) - %d.csv", currentFunction, System.currentTimeMillis()));
    originalLog.setHeaders("Time (ms)", "Value");       // Re-create new log files and set headers.
    transformedLog.setHeaders("Time (ms)", "Value");
    logStartTime = System.currentTimeMillis();
    logWritable = true;
}

/************************* writeValue - writes @param val of @param chart to log *********************************/
private void writeValue(int val, ReplacingLineChartView chart) {
    if (!logWritable) return;
    if (!logPermissionGranted) return;
    if (chart.equals(mChartOriginal)) originalLog.write(originalLog.msTimeFrom(logStartTime), val);
    else transformedLog.write(originalLog.msTimeFrom(logStartTime), val);
}

/************************* graphValue - plots @param val to @param chart *****************************************/
private void graphValue(int val, ReplacingLineChartView chart) {
    if (!graphFrozen) runOnUiThread(() -> chart.addEntry(val));     // Plot to chart under UI tread if not frozen
    // writeValue(val, chart);    // Write the value to disk. Disabled to avoid CVS files
}

/**************** onBytesReceived - a callback runs whenever byte[] data received from the PIC *****************/
private void onBytesReceived(byte[] data) {
    int i = 0;
    while (i < data.length) {                            // length should be 3, but uncertain
        int val = data[i] & 0xFF;                        // Keep the lower byte (0-255)
        i++;                                             // Point to next data sample
        int functionCode = 0;                            // This is the flag for a valid "function" from PIC
        if (val >= 240 && val <= 240+topFunction) {      // PIC functionBT: starting 0xF0
            int val0 = val - 240;                        // possible PIC function code
            if (val0 == currentFunction+1 || val0 == currentFunction-1) functionCode = 1;   // Allow change by 1
            if (currentFunction == 0 && val0 == topFunction) functionCode = 1;              // Hande wrap-around
            if (currentFunction == topFunction && val0 == 0) functionCode = 1;              // Hande wrap-around
            if (functionCode == 1) {
                switchFunction(val0);                    // Call switchFunction to update the Android display
                currentFunction = val0;                  // Update currentFunction
            }
            else if (i < data.length-1) {                // If there are 2 more points left,
                int val1 = data[i] & 0xFF;               // read the first channel
                i++;
                int val2 = data[i] & 0xFF;               // read the second channel
                i++;
                graphValue(val1, mChartOriginal);        // Display the first channel
                graphValue(val2, mChartTransformed);     // Display the second channel
            }
        }
        if (data.length > 300) i = data.length;         // If too many data points queued up, ignore them
    }
}

/*** deviceSelected - runs when a device is selected from DeviceListDialog, handles connecting to that device ****/
private void deviceSelected(BluetoothDevice device) {
    showRefresh("Connecting to the PIC.");  // Let the user know we're connecting by showing a refresh.
    RxBluetooth.connectAsClient(device)     // Use the provided RxBluetooth library to connect to the device.
```

```
        .subscribe(socket -> {
            Timber.d("Connected.");
            snackMessage("Connected");
            mSocket = socket;           // Save the socket for when we need to clean up when we're done.
            runOnUiThread(() -> {       // Set the FAB to become a graph freeze function.
                mFab.setImageResource(R.drawable.ic_pause_black_24dp);
                hideRefresh();          // Stop our refresh.
            });     // Use RxBluetooth to listen to the InputStream of data.  Use onBytesReceived as a callback.
            RxBluetooth.readInputStream(socket).subscribe(this::onBytesReceived, this::onError);
        }, this::onError);
    }

    /************* showRefresh - shows the refresh progress bar and text, while setting a text message **************/
    private void showRefresh(String text) {
        progressText.setText(text);
        Animation.show(progressBar);
        Animation.show(progressText);
    }

    /************* hideRefresh - hides the progress views ************************************************************/
    private void hideRefresh() {
        Animation.hide(progressBar);
        Animation.hide(progressText);
    }

    /************* onError - called anytime a Throwable is used to signify an error **********************************/
    private void onError(Throwable e) {
        Timber.e(e, "Input Stream Error");
        if (e.getMessage().contains("bt socket closed")) snackMessage("Disconnected");
        else snackMessage(e.getMessage());
        tryToCloseSocket();                 // Try to close our socket, if its not null and it exists.
        runOnUiThread(this::hideRefresh);   // Hide any refresh we've got going on!
    }

    /************* tryToCloseSocket - tries to close the BluetoothSocket cleanly *************************************/
    private void tryToCloseSocket() {
        if (mSocket != null) {
            try {
                mSocket.close();
            } catch (Exception e) {
            }
            mSocket = null;
            mFab.setImageResource(R.drawable.ic_bluetooth_connected_24dp);
            graphFrozen = false;
        }
    }

    /************* snackMessage - a little helper function to display a Snackbar with a text provided ***************/
    private void snackMessage(String text) {
        Snackbar.make(mToolbar, text, Snackbar.LENGTH_LONG).show();
    }

    /**** onDestroy - called when Activity is about to go down for the count, clean up by closing BluetoothSocket ****/
    @Override       // overrides a method of its superclass
    protected void onDestroy() {
        super.onDestroy();
        ButterKnife.unbind(this); // We need to unbind from ButterKnife, otherwise we leak memory.
        tryToCloseSocket(); // Cleanup our bluetooth stuff.
    }

    /************* onOptionsItemSelected - Called when button is clicked on in the Toolbar **************************/
    @Override       // overrides a method of its superclass
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {             // Check to see what the ID of this item is.
        case R.id.action_health_guess:      // If it's the Heart icon, switch to the new activity.
            Intent intent = new Intent(this, HealthGuessActivity.class);
            startActivity(intent);
            return true;                    // Return true to say we've consumed this call.
        case R.id.action_location:          // If it's the Location icon, switch to the new activity.
            Intent locationIntent = new Intent(this, LocationActivity.class);
            startActivity(locationIntent);
            return true;
        }
        return false;
    }

    /******* onCreateOptionsMenu - called when the Activity is ready to have the menu inflated from a resource *******/
    @Override       // overrides a method of its superclass
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
}
```