NAME: SAKSHI VINAYAK KITTURKAR
CSU ID: 2860273
HOMEWORK 15

The improved pedometer app with the custom activity detection algorithm has a design issue: the step data are pulled from the SQLite database only at the start of the app. So, if you have run the app for a few minutes with multiple steps, the display the main activity won't change at all. To see the data collected, you would have to close the app and restart it.
To address this issue, modify the app so that whenever a new batch of steps are recorded, the display on the main activity would be updated. Alternatively, you may add a refresh button on the main activity layout to retrieve the data and display it.
The app has another issue: the step count is not as accurate as the step counter. Do some experiments (you will need to carry your phone and walk for say, 100 steps, and compare what is recorded in your app with respect to the actual steps you made) and tune the 3 thresholds used for step counting so that they would work for you – meaning the step count is almost as accurately as the actual number of steps.

= In this homework I have improved homework 14 that was a pedometer to count the steps.

In layout: StepsDBHelper this code defines an Android SQLite database helper class called StepsDBHelper for managing step count data. It includes methods to create a database table for storing daily step counts, insert or update step count entries for the current date, and retrieve a list of all date-step count pairs. The code utilizes a DateStepsModel class to represent date and step count information. It's important to note that the code could be enhanced for better resource management and exception handling.

Database Schema:
The class defines a SQLite database with a table named "StepsSummary."
The table has three columns:
ID: An auto-incrementing primary key.
CREATION_DATE: A text column representing the date in the format "MM/DD/YYYY."
STEPS_COUNT: An integer column representing the number of steps for a specific date.

```
private static final String CREATE_TABLE_STEPS_SUMMARY = "CREATE TABLE " +
TABLE_STEPS_SUMMARY + "(" +
        ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +
        CREATION_DATE + " TEXT," +
        STEPS_COUNT + " INTEGER" + ")";
```

The constructor initializes the database helper with the specified database name ("StepsDatabase"), version, and a null cursor factory.

```
public StepsDBHelper(@Nullable Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

Database Creation:
The onCreate method is overridden to execute the SQL query and create the "StepsSummary" table when the database is first created.

```
@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL(CREATE_TABLE_STEPS_SUMMARY);
}
```

Inserting or Updating Step Count Entry:
The createStepsEntry method checks if an entry already exists for the current date in the "StepsSummary" table.
If an entry exists, it increments the step count for that date. If not, it inserts a new entry with a step count of 1.

```
public boolean createStepsEntry() {
   // ...
}
```

Reading Step Count Entries:
The readStepsEntries method retrieves all step count entries from the "StepsSummary" table and returns them as an ArrayList of DateStepsModel objects.

```
public ArrayList<DateStepsModel> readStepsEntries() {
   // ...
}
```

DateStepsModel Class:
Although not provided in the code snippet, it's mentioned that there is a DateStepsModel class, likely used to represent date and step count information.

```
public class DateStepsModel {
   String mDate;
   int mStepCount;
   // ... (Assumed attributes and methods)
}
```

Handling Database Upgrades:
The onUpgrade method is not implemented in the code snippet. It should be used to handle database schema upgrades when the version number is incremented.

```
@Override
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
```

```
    // Handle database upgrades (if needed)
}
```

In layout: StepsHistoryActivity This code defines an Android activity (StepsHistoryActivity) that displays a list of historical step count entries. It uses a custom ListAdapter to populate a ListView with data retrieved from a SQLite database using a StepsDBHelper class. The activity also includes a refresh button (FloatingActionButton) to update and refresh the displayed step count entries.

StepsHistoryActivity is an Android activity extending AppCompatActivity.
It initializes UI elements (ListView, FloatingActionButton).
Retrieves historical step count data from the SQLite database using getDataForList method.
Creates a custom ListAdapter to populate the ListView.
Starts a background service (StepsService) using an Intent.
Implements a refresh button (floatingActionButtonRefresh) that updates the displayed step count entries when clicked.

```
public class StepsHistoryActivity extends AppCompatActivity {
    // Class variables
    private ListView mSensorListView;
    private ListAdapter mListAdapter;
    private StepsDBHelper mStepsDBHelper;
    private ArrayList<DateStepsModel> mStepCountList;
    private FloatingActionButton floatingActionButtonRefresh;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_steps_history);

        // Initialize UI elements
        floatingActionButtonRefresh = findViewById(R.id.floatingActionButtonRefresh);
        mSensorListView = findViewById(R.id.steps_list);

        // Retrieve and display step count data
        getDataForList();
        mListAdapter = new ListAdapter(mStepCountList, this);
        mSensorListView.setAdapter(mListAdapter);

        // Start a background service for steps (StepsService)
        Intent stepsIntent = new Intent(getApplicationContext(), StepsService.class);
        startService(stepsIntent);
```

```
        // Set a click listener for the refresh button
        floatingActionButtonRefresh.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Refresh data and update the ListView
                getDataForList();
                mListAdapter = new ListAdapter(mStepCountList, StepsHistoryActivity.this);
                mSensorListView.setAdapter(mListAdapter);
            }
        });
    }

    // Method to retrieve step count data from the database
    public void getDataForList() {
        mStepsDBHelper = new StepsDBHelper(this);
        mStepCountList = mStepsDBHelper.readStepsEntries();
    }
}
```

ListAdapter is a custom adapter extending BaseAdapter used to populate the ListView.
It holds an ArrayList<DateStepsModel> for step count data.
Inflates the layout (list_rows.xml) for each list item.
Overrides getView method to set the text of a TextView with date and total steps for each entry.

```
class ListAdapter extends BaseAdapter {
    // Class variables
    TextView mDateStepCountText;
    ArrayList<DateStepsModel> mStepCountList;
    Context mContext;
    LayoutInflater mLayoutInflater;

    // Constructor
    public ListAdapter(ArrayList<DateStepsModel> mStepCountList, Context mContext) {
        this.mStepCountList = mStepCountList;
        this.mContext = mContext;
        this.mLayoutInflater = (LayoutInflater)
this.mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }
```

```
    // Override methods for BaseAdapter
    // ...

    // Method to create and return the View for each list item
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            convertView = mLayoutInflater.inflate(R.layout.list_rows, parent, false);
        }
        mDateStepCountText = convertView.findViewById(R.id.sensor_name);
        mDateStepCountText.setText(mStepCountList.get(position).mDate + " – Total Steps: " +
                String.valueOf(mStepCountList.get(position).mStepCount));
        return convertView;
    }
}
```

list_rows.xml is the layout file for each row in the ListView.
It contains a TextView (sensor_name) to display date and total steps.

```
<!-- The layout file for each row in the ListView -->
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/sensor_name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="10dp"/>
```

In layout: StepsService This code defines an Android service, StepsService, which monitors the step detector sensor and updates a SQLite database with step count entries.

onCreate: Method called when the service is initially created.
mSensorManager: Manages access to device sensors.
Check if the device has a step detector sensor.
If available, get the step detector sensor, register the service as a listener for sensor events, and initialize the database helper (StepsDBHelper).

```
@Override
public void onCreate() {
    super.onCreate();

    // Initialize the SensorManager
```

```
    mSensorManager = (SensorManager) this.getSystemService(Context.SENSOR_SERVICE);

    // Check if the device has a step detector sensor
    if (mSensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR) != null) {
        // If available, get the step detector sensor
        mStepDetectorSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR);

        // Register the sensor listener (this service) to receive sensor events
        mSensorManager.registerListener(this, mStepDetectorSensor,
SensorManager.SENSOR_DELAY_NORMAL);

        // Initialize the database helper
        mStepsDBHelper = new StepsDBHelper(this);
    }
}
```

onStartCommand: Method called when the service is started.
Returns Service.START_STICKY: Indicates that the service should be restarted if it gets terminated.

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // Return START_STICKY to make the service restart if it gets terminated
    return Service.START_STICKY;
}
```

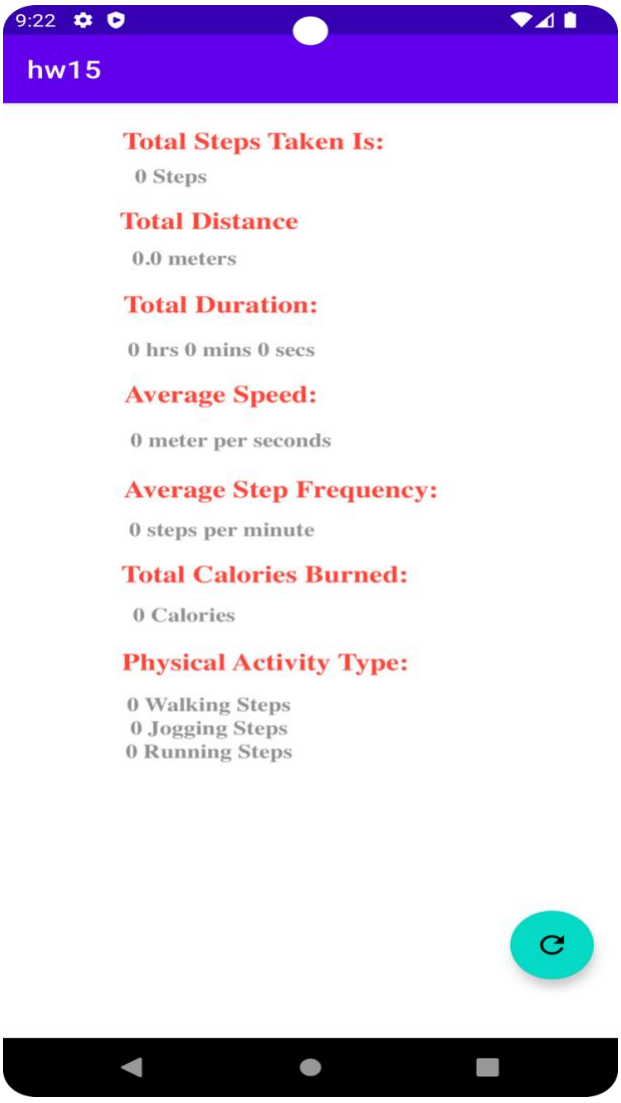
onSensorChanged: Method called when the step detector sensor detects a step.
Invokes the createStepsEntry method of mStepsDBHelper to update the SQLite database with a new step entry.

```
@Override
public void onSensorChanged(SensorEvent event) {
    // Update the SQLite database with a new step entry
    mStepsDBHelper.createStepsEntry();
}
```

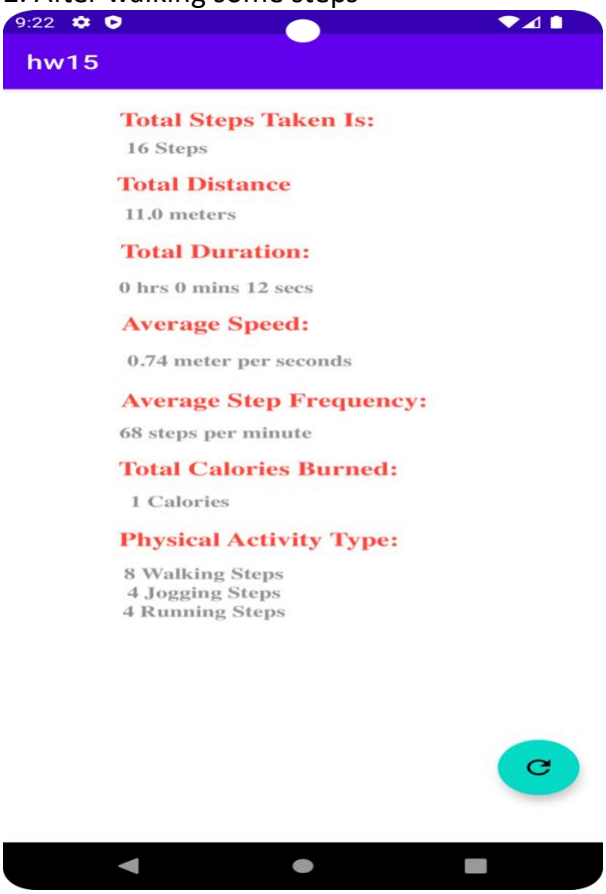
onAccuracyChanged: Method called when the accuracy of the sensor changes.
Not used in this code.

```
public void onAccuracyChanged(Sensor sensor, int accuracy) {
    // This method is not used in this code
}
```

1.When we open and start app



2. After walking some steps



3. After refreshing and start walking again