NAME: SAKSHI VINAYAK KITTURKAR
CSU ID: 2860273
HOMEWORK 14

Build the pedometer app using the step detector sensor.

= I have built an application which is pedometer used to detect the steps walked.

In layout: DateStepsModel.java This code defines a Java class named DateStepsModel that represents a simple model for storing information related to a date and a step count. However, the provided code itself does not perform any specific functionality or behavior; it merely defines the structure for storing data.

This line declares the package name for the Java class. The class is defined in the package named com.example.hw14.

```
package com.example.hw14;
```

Here, a class named DateStepsModel is declared with the "public" access modifier. This means that the class can be accessed from outside its package.

```
public class DateStepsModel {
```

Within the class, there are two public member variables:
mDate: It is a public field of type String. This variable is likely intended to store a date value.
mStepCount: It is a public field of type int. This variable is likely intended to store a count of steps.

```
    public String mDate;
    public int mStepCount;
```

In layout: StepsDBHelper the StepsDBHelper class provides functionality for creating, updating, and reading step count entries in an SQLite database in an Android application. The DateStepsModel class is likely used to represent the data model for storing date and step count information.

These constants define the database version, name, table name, and column names.

```
private static final int DATABASE_VERSION = 1;
private static final String DATABASE_NAME = "StepsDatabase";
private static final String TABLE_STEPS_SUMMARY = "StepsSummary";
private static final String ID = "id";
private static final String STEPS_COUNT = "stepscount";
private static final String CREATION_DATE = "creationdate";
```

This SQL query is used to create the StepsSummary table with columns id, creationdate, and stepscount. id is an auto-incrementing primary key.

```
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 with the specified name, version, and a null cursor factory.

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

This method is called when the database is created for the first time. It executes the SQL query to create the StepsSummary table.

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

In layout: StepsHistoryActivity This code, overall, sets up an activity to display a list of historical step count entries using a custom adapter. The data comes from an SQLite database managed by the StepsDBHelper class, and the layout for each row is defined in list_rows.xml.

This method is called when the activity is created.
It sets the content view to the layout defined in activity_steps_history.xml.
Initializes the mSensorListView with the corresponding ListView from the layout.
Calls getDataForList() to populate the mStepCountList.
Creates an instance of ListAdapter and sets it as the adapter for the mSensorListView.
Starts the StepsService using an Intent.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_steps_history);
    mSensorListView = (ListView) findViewById(R.id.steps_list);
    getDataForList();
    mListAdapter = new ListAdapter(mStepCountList, this);
    mSensorListView.setAdapter(mListAdapter);
    Intent stepsIntent = new Intent(getApplicationContext(), StepsService.class);
    startService(stepsIntent);
}
```

This method initializes the mStepsDBHelper and reads historical step count entries from the SQLite database into mStepCountList.

```
public void getDataForList() {
   mStepsDBHelper = new StepsDBHelper(this);
   mStepCountList = mStepsDBHelper.readStepsEntries();
}
```

mDateStepCountText: Represents a TextView for displaying date and step count.
mStepCountList: The data source containing historical step count entries.
mContext: The context of the activity.
mLayoutInflater: Used to inflate the layout for each row in the ListView.

```
TextView mDateStepCountText;
ArrayList<DateStepsModel> mStepCountList;
Context mContext;
LayoutInflater mLayoutInflater;
```

Initializes the member variables with the provided parameters.

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

This method is responsible for creating and returning a View for a specific position in the ListView.
If convertView is null, it inflates the layout for a single row (list_rows.xml).
Sets the text of mDateStepCountText to display date and total steps for the corresponding entry.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
   if (convertView == null) {
      convertView = mLayoutInflater.inflate(R.layout.list_rows, parent, false);
   }
   mDateStepCountText = (TextView) convertView.findViewById(R.id.sensor_name);
   mDateStepCountText.setText(mStepCountList.get(position).mDate + " – Total Steps: " +
String.valueOf(mStepCountList.get(position).mStepCount));
```

```
    return convertView;
}
```

In layout: StepsService This code defines an Android service named StepsService that listens to the step detector sensor and updates a step count entry in an SQLite database.

The onCreate method is called when the service is created.
It initializes the mSensorManager with the system service for sensors.
Checks if the device has a step detector sensor (Sensor.TYPE_STEP_DETECTOR). If available, it registers this service as a listener for the sensor events with normal delay.
Initializes the mStepsDBHelper for database operations.

```
@Override
public void onCreate() {
    super.onCreate();
    mSensorManager = (SensorManager) this.getSystemService(Context.SENSOR_SERVICE);
    if (mSensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR) != null) {
        mStepDetectorSensor                                                    =
mSensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR);
        mSensorManager.registerListener(this,                  mStepDetectorSensor,
SensorManager.SENSOR_DELAY_NORMAL);
        mStepsDBHelper = new StepsDBHelper(this);
    }
}
```

The onStartCommand method is called when the service is started.
It returns Service.START_STICKY, which means that the service will be restarted if it gets terminated.

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    return Service.START_STICKY;
}
```

The onSensorChanged method is called when a sensor value changes, in this case, when a step is detected.
It calls the createStepsEntry method of mStepsDBHelper to update the step count entry in the database.

```
@Override
public void onSensorChanged(SensorEvent event) {
    mStepsDBHelper.createStepsEntry();
}
```

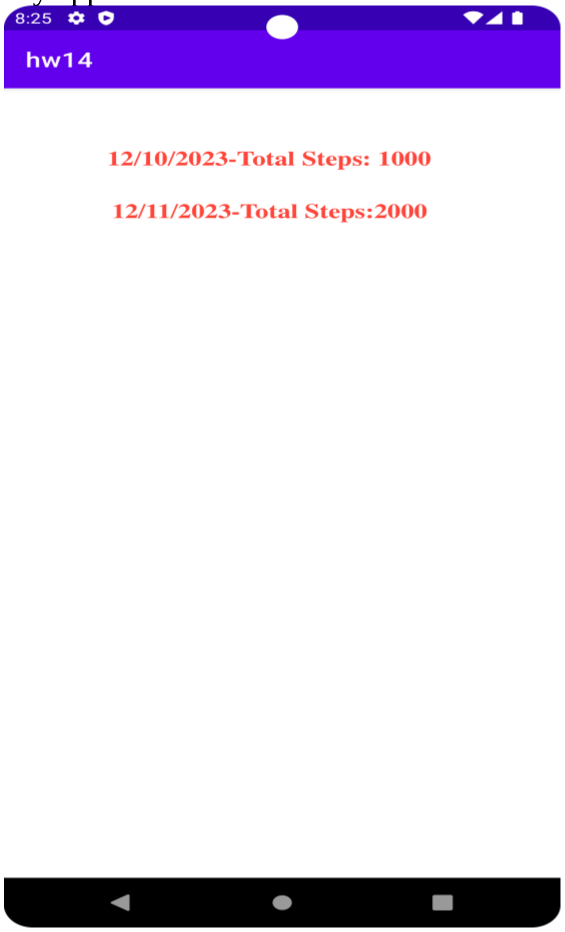
The onAccuracyChanged method is required by the SensorEventListener interface but is not implemented in this code.

```
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}
```

The onBind method is part of the Service lifecycle but is not used in this implementation. It returns null since the service does not provide a binding interface.

```
@Override
public IBinder onBind(Intent arg0) {
    return null;
}
```

My application screen:



```
12/10/2023-Total Steps: 1000
12/11/2023-Total Steps:2000
```