**Telemeter**

**Assignment Description:**

The goal of this assignment is to demonstrate the use of sensors in android device and plot the sensor values on a line graph at every 5 seconds. This application records data in data structure until user press stop tracing data using selected sensors. These sensor values are then updated at a constant rate of 5 seconds on the graph.

**Files / Pages:**

In this assignment, I have created 2 .java files (Both for same activity but different classes) and 1 .xml file for layout.

XML files involved are:

1. activity\_main.xml

Java files involved are:

1. ActivityMain.java
2. CustomView.java

**activity\_main.xml**

activity\_main.xml is the only file in this android application that contains the UI of MainActivity. This layout consists of 1 Timer, 1 Start/Stop Button, 1 Spinner to select Sensor type, 1 Spinner to select Sensor, 3 different Text View to display live values of X, Y, Z axis from appropriate sensor, 1 Custom View to plot the recorded points on the graph. This layout is using Relative Layout to adjust the elements on the UI relatively to each other. All the spinner values are defined in default android file “values => strings.xml” and are pulled in activity\_main.xml using string arrays. Below is the screenshot of UI containing elements as described above.

A screenshot of a cell phone

Description automatically generated

activity\_main.xml: Image 1

As shown in figure activity\_main.xml: Image 1, Initially the Spinner to select sensor is hidden. User will be able to select sensor once the sensor type has been selected. User will not be able to start tracing unless a proper sensor type and sensor is selected from the spinner.

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

activity\_main.xml: Image 2 activity\_main.xml: Image 3

As shown in activity\_main.xml: Image 2 and Image 3, once an appropriate sensor type is selected, user will be able to select sensor from 2nd spinner.

A close up of a map

Description automatically generated A screenshot of a cell phone

Description automatically generated

activity\_main.xml: Image 4 activity\_main.xml: Image 5

A screenshot of a map

Description automatically generated A screenshot of a cell phone

Description automatically generated

activity\_main.xml: Image 6 activity\_main.xml: Image 7

As shown in above Image 4, Image 5, Image 6 and Image 7, as soon as the tracing has been started, the spinners are disabled. This does not allow user to change sensor type or sensor until trace has been stopped by user. It is also clearly shown that the live values of all the sensors will be updated above CustomView and just below the sensor spinner.

**MainActivity.java**

MainActivity.java is the only activity class in this project responsible to run the entire application and calling CustomView appropriately when required. There are several methods and logics implemented in MainActivity.java. Below are the functions implemented in this Class.

1. onCreate() method – This method is called as soon as an application is started / an activity of MainActivity is created in a device.
2. This method initially has logic to initialize all the required variables and spinners. This handles the visibility of spinner and TextViews that displays the live values of sensors recorded at any time. Initially the ScalarSensor spinner and VectorSensor spinner is set to visibility: INVISIBLE to implement the logic of visibility of spinner as discussed in activity\_main.xml.
3. Next this method has a logic to implement onItemSelectedListner on SensorType spinner. When user selects any one of the given options, onItemSelectedListner is triggered and a functionality to display appropriate 2nd spinner runs and allows user better UI experience. In case user selects scalar sensors, 2nd spinner for options of scalar sensor is set to Visible. In case user selects vector sensors, 2nd spinner for options of vector sensor is set to Visible using logic implemented in these set of code in this method.
4. Next logic is to handle START and STOP button. There is only single button that handles both the functionality. Initially text of button is set to START. On clicking start button, tracing is activated and text of the button changes to STOP. This is because a logic is implemented based on the text of the button. If the text of the button is start, it means trace is not yet started and will start when user clicks on button and if the text on the button is stop, it means trace is in progress and will be stopped when user clicks on the button.
5. A logic to check sensor when user clicks on start button is implemented to validate that trace must not be started without user not having selected appropriate spinner values.
6. There is another logic being implemented below this check to handle appropriate sensor listeners to get activated when correct spinner is selected, and user press start button. In case Sensor Type is 1, user wish to start sensing values from scalar sensor and hence, listener for appropriately selected scalar sensor will be set to active and all the required spinners will be disabled not allowing user to change spinner values when trace is in progress. In case sensor type is 2, user wish to start sensing values from vector sensors and hence, listener for appropriately selected vector sensor will be set to active and all the required spinners will be disabled not allowing user to again change the spinner values.
7. There is a logic to continuously change the timer values every 1 second. There is a delay of 0.5 Second given to update the values as processor can perform other operations in another 0.5 seconds after updating timer values.
8. Finally, there is a logic in this method to handle Stopping the trace. When user clicks on stop button, all the traces will be stopped, all the values will be reset, button text will also be changed to START and all the data structures will be cleared for next run.
9. CheckSpinner() method – This method is used to check if user have selected appropriate spinner values before starting the trace. In case user have not selected appropriate values, this function returns false and a message is toasted to user to select values of spinner before starting trace.
10. FillDataInDataStructure() method – This method is mainly used to fill in all the required data in data structure. All the sensors call this method when a new value needs to be recorded. In case of scalar sensors this method only records values of X. But, in case of vector sensors, this method is designed to record all the X, Y, Z values, respectively. There is a logic to update the Live values in UI from this method after trimming values to 3 digits after decimal places. As soon as this method is called, time and values of X, Y, Z along with sensor id is passed. This method first checks for sensor id and then checks for time to record values for same. In case there are already few values of time already in data structure, it appends a new value for same key. Time being always unique, there is no chance of having a duplicate key and hence, time is used as key in all data structures to records values against. In value for any time is 1st, a new key value pair will be added. This function also has logic to identify and record only last 5 seconds of data in any data structure.

**CustomView.java**

CustomView.java consist of few predefined default constructors of custom view and four important methods:

1. onMeasure() method – This method takes two inputs one is CustomViewWidth and CustomViewHeight this methods are called when the view is displayed on the screen and above two default parameters are passed in the method. I am using this method to force the custom view layout to be square on the UI. In this method, I am comparing viewHeight and viewWidth and whichever is smaller I am forcing my custom view dimension to be of that size.

1. CheckSensor() method – This method is used at all times in my custom view to identify sensor selected by the user.
2. If user have selected sensor type equals to scalar sensor and scalar sensor equals to light sensor, this method returns 1.
3. If user have selected sensor type equals to scalar sensor and scalar sensor equals to proximity sensor, this method returns 2.
4. If user have selected sensor type equals to vector sensor and vector sensor equals to gyroscope, this method returns 3.
5. If user have selected sensor type equals to vector sensor and vector sensor equals to linear accelerometer, this method returns 4.
6. If in case there are no sensor selected by the user this method returns 0.
7. FindMaxValue() method – This method is mainly used to identify the maximum and minimum value of y-axis on the graph for the respective sensor. This method checks for sensor id and:
8. If selected sensor is a scalar sensor, it compares for values of x recorded in last 5 second of the trace. It fast finds 5 different maximum values from 5 list stored in HashMap for each second. It then returns the maximum of the five values.
9. If selected sensor is vector sensor, it compares in the similar way for maximum values in all the three i.e. in HashMap of x, y and z values respectively and stores the maximum value in a list. Here there is an additional logic of finding the minimum values from all the list in the HashMap, this is to handle the negative values recorded in vector sensors. The found minimum value is stored in a list of different minimum values. Now, the maximum values in the list of max values is compared with the negative multiplication of minimum value from min values list and whichever is maximum is returned from this function.

This is how the maximum value of y-axis is calculated in this method.

1. onDraw() method – This method is highly responsible to plot all the recorded points on the graph using canvas. The canvas width and height is forced to square in onMeasure() method as describe above. Initially all the colour coding is being defined for text x-axis and y-axis and the three lines of graph. Here, the text size and stroke width are also initialized. Now, the graph of set is initialized to canvas width by 14 parts. The canvas width is divided into 14 parts because the x-axis is the time axis and it is to be divided into 5 seconds with each second having space of GraphOffset \* 2 plus the 1 second of offset in negative x-axis to display y-axis values. The other two GraphOffset is to maintain equal spaces from all sides.
2. Next, I am checking sensor id to identify the position of x-axis if the sensor id is 1 or 2 i.e. if anyone of the scalar sensor is selected the x-axis will be at bottom of the graph after leaving two GraphOffsets. If the sensor id is 3 or 4 i.e. if anyone of the vector sensor are selected, the x-axis will be positioned in the centre of the graph. The position of y-axis will be same for all the sensor graphs.
3. Next, I am calling method to return next value recorded in last 5 seconds. Now, based on sensor type and selected sensor the max value is assigned for the first 5 seconds trace. This max value is then used to divided the y-axis into equal parts from 0 to maximum value in scalar sensors and minimum to maximum value in vector sensors.
4. Next, I am initializing the zero position on my graph for different sensor types. The zero position of the graph will be different based on position of x-axis for different sensor types. Now, I am dividing the x-axis into number of points recorded per seconds in my HashMap for different sensors. Now, I am calculating Xvalue2 and Yvalue2 based on this calculation.

**Below is the mathematical calculation for plotting points on graph:**

I am using cross multiplication formula to identify the pixel to plot the points.

* Pixel height (x) = (CanvasHeight \* scalar sensor data recorded)/ (max value recorded in last 5 seconds)

I am adjusting the Graphaaoffset while calculating the required pixel value. Now the final result is then subtracting from CanvasHeight to get the YValue2.

For XValue2 I am adding XValue1 and (Twice the GraphOffset)/(Number of values recorded per second in scalar or vector value list).

Finally, after plotting I am assigning XValue2 to XValue1 and YValue2 to YValue1 to make the graph continuous. This will be iterated till all the points plotted on the graph. In next refresh after 5 seconds all the process will be repeated.

**Data structure:**

I have made use of HashMap and Lists mostly for recording all the values from the sensor. I am using 5 HashMaps to record the data from sensor:

1. TimeRepresentedBySeconds – This HashMap is storing all the data related to time. I am using seconds as Key and the Time as value to identify which time at any particular second is recording data in other HashMaps. Data in this HashMap is recorded as {(0 = 01:10:10), (1 = 01:10:11), (2 = 01:10:12), (3 = 01:10:13), (4 = 01:10:14)}. This HashMap is designed in a way that it will always contain only 5 values from 0th Second to 4th Second. Once the time exceeds 5 seconds, all the values of 0th Second in this HashMap will be replaced with the 6th Second. Here, 0th second indicates 1st second in timer, 1st second indicates 2nd second in timer and so on till 4th second indicating 5th second in timer.
2. TimeData1 – This HashMap is storing all the data related to scalar sensors only. Scalar sensors have only X values and hence, this HashMap will be used to store all the values of X recorded from scalar sensors. The data stored in this sensor is String, List<Float>. The key in this HashMap is the value of TimeRepresentedBySeconds. This is because, at any time there will be no data of same time as a different Key. There can be multiple values recorded within 1 second and hence, List<Float> is used in value of TimeData1 to store multiple values detected from scalar sensors.
3. TimeData2X – This HashMap is like TimeData1 but the difference is this HashMap is used to store data of Vector sensors only. This HashMap stores all the X values recorded from Vector Sensors.
4. TimeData2Y – This HashMap is like TimeData1 but the difference is this HashMap is used to store data of Vector sensors only. This HashMap stores all the Y values recorded from Vector Sensors.
5. TimeData2Z - This HashMap is like TimeData1 but the difference is this HashMap is used to store data of Vector sensors only. This HashMap stores all the Z values recorded from Vector Sensors.
6. Apart of the above data structures, there are globally defined static variables to identify Sensor type and Sensor. These variables are also shared with CustomView Class to keep in sync of sensor’s activity in between the two classes.

**Design Decisions:**

1. I have tried to keep the UI as easy to understand as possible. I have added Button on the top centre with text Start to indicate user can start trace by clicking on same.
2. I have also added a spinner exactly below the START button to explain user that he is expected to select one of the given options from spinner.
3. Once user selects any one sensor type, another spinner based on selected sensor type will become visible allowing user to select the sensor. This step by step approach of selecting values on spinner makes the UI of this application more user friendly.
4. I have even added few text messages on blank graph indicating that graph will be visible once user start tracing from sensor. This makes user easy to understand why the graph is empty on the main screen.
5. I have selected colours red, black and green to indicate graph lines for Vector sensors and colour red to indicate graph line for scalar sensor. The reason of choosing these colours are that they are dark and are easy to compare values on graph. I have also increased the thickness of graph line for clear visibility and easy to understand points on X-Axis and Y-Axis. I have tried to keep all the colours in this application as dark as possible to keep this UI understandable and distinguishable for the person having poor visibility or any issues with eyesight.
6. Once, trace is started, all the live values of X-axis, Y-axis, Z-axis will be displayed in the Text View just at the top of the graph. This allows the user to understand which value is being recorded at any given instance. These values will be later displayed on the graph.