Assignment 3

Anas Djebbari

2852680

The data structure which was used for this app was an array list which would be needed later on to do calculations. GPSActivityDetails class was created, it contained the variables, speed, langt, longt, distance, time, now these are the variables which will be needed for the plotting point onto the graph.

The initial start of this app was to get the GPS sensor working to collect the data that is required for the app to draw up the graph based on. Since the locationListener in the java library for **onLocationChanged** method wasn't suitable it had to be overridden. The listener then is able to receive a new location, this is done by returning a longitude and latitude. These coordinates are now sent and saved in a class we create called **GPSActivityDetails**. Moreover we also get a the speed for the user, by calling the location.getSpeed method. Once all of this data is collected a straightforward calculation is done in order to give us the averag e speed for the user, in method **getAvgSpeed**(). This calculation is done by all of the speeds collected from the array and dividing it by the array size.

Next step, we now need to calculate the distance between two locations, this is done by using the coordinates by using the **distanceTo** method. The time is stored for every 1000 metres reached. Once it hit the 1000 meters it's converted to kilometres. Now we need to somehow get the time, to this we use the chronometer, this is a simple timer which can be display the timer, this timer gives us the time in milliseconds yet since we need it to be in seconds a simple conversion is done and is then sent to **GPSActivityDetails**. Now that we have collected all of the data we can display information on the app, we display the time, the average speed, distance and the time in the main activity. This is how the data is being displayed in the main activity, refer to diagram below.

```
long duration_milisec = SystemClock.elapsedRealtime() - cm.getBase();
long duration_sec = duration_milisec / 1000;
user.setTime(duration_sec);

spd.setText("Speed " + (int) Math.round(s) + " km/hour");
avgSpeed.setText("Avg Speed " + (int) Math.round(temp_speed) + " km/hour");
dis.setText("Distance : " + displayed_distance);
```

Now that we have all our data displaying, we can start adding some functionality for the user to interact with the app. Three buttons were added, start, reset and finish. Start button would start the timer. Finish button would send you to the graph and the reset button would clear the view, reset the chronometer and sets the locations to null. One of the issues which was that once started the user can still click the start button again, and before the app ran the user can tap the finish button and they would be redirected to the graph page even though no data was collected, this was solved by the the clickable method. An example of this can be seen in the diagram below.

```
strtBtn.setClickable(false);
rstrtBtn.setClickable(true);
stpBtn.setClickable(true);
```

Also when the app is started and the timer starts running and it starts running, it would be a nice touch for the overall user experience to have some sort of an alert to ket them know that the application is running, to do this we just change the entire background colour. Also the layout of the buttons are spread and are big, so that they user can access the buttons easily without worry about hitting another button.

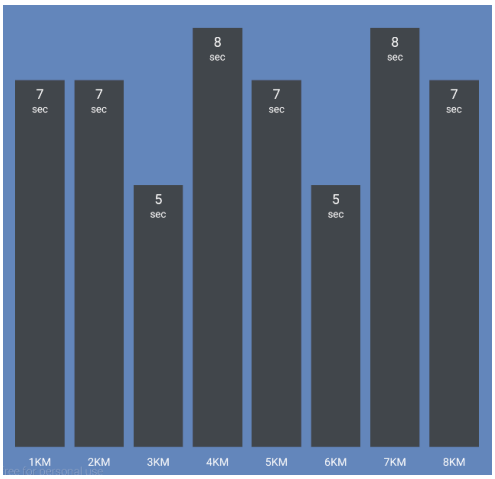
To this point the main activity is pretty much sent and does what its needed, kilometres are being collected alongside there time, the average speed is being displayed and the current speed is also shown. Now to plotting this data onto a graph, we start this by getting the screens width, reason being we need the graph to be a square so only one side counts, and we use the width because a square with sides of the length would be too large. Then we minus a bit off the square so that it fits into the layout with some space left over, we call it graphWidthHeight. To centre this square we divide the screen width by the graphWidthHeight and let it equal the left and right, because since they are both square with exact dimensions we only need to do it on one side the left and top, their opposites will automatically match.

```
int screenWidthHeight = getWidth();

float scale = 1.0f;

float graphWidthHeight = screenWidthHeight - 70;
float sqr = (screenWidthHeight - graphWidthHeight)/2;
float left = sqr;
float top = sqr;
```

Next we draw up our labels, to do this we use an iterator which will go through the hashmap in order to identify how far the bar should go. We calculate the sizes for each bar and how wide each bar is. We also have a second iterator which will be drawing the bars onto the graphs which will depend on the results from the previous calculation. Now we have a graph displaying we would like to have some labels also attached for more information. The labels will look as follows:

```
paint.setColor(Color.parseColor("#ffffff"));
paint.setTextSize(40*scale);
canvas.drawText(String.format("%.0f",value), left+margin+px, barTop + offset, paint);
paint.setTextSize(30*scale);
canvas.drawText(String.format("sec"), left+margin+px, barTop + offset+40, paint);
paint.setTextSize(32*scale);
canvas.drawText(String.format(String.valueOf(km) + "KM"), left+margin+px, barBottom + offset, paint);
```

To get the kilometres to show, a counter was printed out which was set as 1 before the loop, so as it goes in we just print out the counter for each time, it would actually print each kilometre in order with their corresponding times.



To get the data we collected from the previous view to appear in the DrawView (where the graph is being drawn), we had to send the data into an insert method which would be called from within the **onDraw** method, which would take in a float. Our data was saved in an array, it was collected from the **GPSActivityDetails** class. We than had a loop which would loop through the array and after every iteration this insert method was called to put our km and time into the data LinkedHashMap.

In the DrawView, we override the onMeasure method, this will force the custom view to be a square the whole time. RelativeLayout was used to achieve a custom view, this helped us push the linear layout to the bottom of the page. Then LinearLayout was used for the textviews.

In the **GPSActivityDetails**, here we have a custom view with three text view. We use getIntent().getExtras() in order to display the distance travelled, user average speed and the total time taken for their journey. The integer array list was taken in, looped and each integer was converted into a float and then added into an array list of floats which would then be passed to the DrawView class in the set_temp method

```
Bundle data = getIntent().getExtras();
int speed = data.getInt("avg_speed");
String dist = data.getString("total_distance");
String time = data.getString("total_time");
collected_info = data.getIntegerArrayList("graph_array");
```