G53MDP
Coursework 2 - Report
BurnBoy, Michael Carter

For coursework 2, my activity tracker begins logging when you hit start inside the FAB, a library on github that i made use of.
For this to work we make use of a LogService.java. Overriding Service methods, on create sets up the location manager. We get an instance of that system service so we can then set it up to, depending on permissions and activation status, allow locations to be received using either GPS or the network. Implementing a locationListener, through here both GPS and network are requested for their location data and each location is stored in a MovementMarker model. When first made, the constructor allows us to get the latest location from each corresponding provider (GPS, network) instantiating with the name as a parameter into new Location(); When the user moves, onLocationChanged is triggered and we get a new Marker. When the service stops, onDestroy is called last and to save what we have logged, fetching all markers from each locationListener.

LogContentHelper is a clean wrapper for contentresolver where we then create a new movementLog session in the database, and with that, in a separate table, have their corresponding markers saved. The table saves all the markers data with a foreign key to refer back to the MovementLog table so we know what each is referring to.

Maps activity has two functions. First, you can press the action bar button and the starting point of each Log is shown as a marker (though not fully realised, i got stuck fully getting the contentResolver query to get what I wanted). Secondly, from the history expandableList, the user can select to view the map and all points corresponding to that individual MovementLog will appear. Calculating the time the logging took and the distance covered are little useful additions.

When the user hits the start logging button, we check for permissions, the status of the service (running or not, which was a bit unreliable, so I temporarily simplified) and then run startService(); By starting the service this way there is a background process made so that now we can freely bind to it and from it irrespective of whether the app itself has closed or not. This thread is controlled by the binder itself who is responsible to linking the higher-level component with the service. We immediately bind to the service using the custom local binder class in the service class and are on our way. Service connection from the main activity is a crucial component of service process as this is where we discern between a managable and in use service, and a disconnected one. For it is here that we establish the binding variable and its link to the activity.

The notification works reasonably well, though difficulty arose when the user force closed the app (square then swipe close). The Notification tended to hand, even when pending intents to the broadcastReceiver were made.

A broadcast receiver was used in relation to the notification. I considered the use of it for tracking markers though the way implemented seemed more simple. When unBind is called, the notification is cancelled and the NotificationBroadcast receiver catches and performs the function. From the main activity, when one hits the stop logging button an intent with the corresponding cancel action is called, and a broadcast is made. Using the universal notification id (which is referenced by the service class when its created, and destroys the notification) we point the system service NotificationManager to this and cancel. The user can also press the action button on the notification. The broadcast is received through completing the cancel action somehow gets lost.

For back end, the contract is key in defining the location of each item we desire. Making heavy reference to the constants in this class, the ContentProvider can query the database and return the relevant cursors, insert or delete attributes. Contentprovider is one steb above dbHelper where it makes use of the methods made in there. To clean things up and make the content provider easier to use, a ContentHelper was made, that nicely takes in a context and returns certain pre-made queries, to save rewriting common queries.

I initially had considerable difficulties trying to understand how services work which I failed to understand for the musicplayer piece of coursework. Understanding how contentProvider works and how to use it, as well as constructing a framework took considerable time too. I was comfortable with DBhelper, and initially made it fully integrated into that system, then changing it to fully integrate contentProvider also took time. Whilst well prepared and planning well in advance, these two key aspects stalled progress. i believe I’ve got a good understanding of both facets of an Android application, though the overall quality of the tracker itself suffered. I planned on adding in graphical functionality (for statistics) and route pathfinding (so maps arent just a collection of markers). Whilst making the service i found I’d spend good chunks of time authoring, and assuming itll work at the end. Its one trap i get myself in at times and an aspect of my development behaviour i look to improve. I started testing more frequently at a point, after spending days on one aspect, and tended to move forward more efficiently. It makes sense, though tunnel vision effects me there.