Mobile Device Programming
Coursework 2
-
Running Tracking Application
Report

Word Count: 1,044

psylha
14326114

The application is a running tracker. On startup, it loads a RecyclerView of previously tracked runs. The user can click on one of these to view its details, view tracked history of the run and edit its name and description. Having a description available acts as a form of annotating the tracked run. The user can also delete existing tracks. The user can also track a new run, indicated by the button.

The application follows the Model-View-ViewModel (MVVM) design pattern. It stores data by implementing a Room database, which is also made available to others through a content provider following its contract. Through the lifespan of the activity, data from the database is being observed to allow up-to-date modifications. The application requires permission to accessing location and to foreground services.

Database Structure

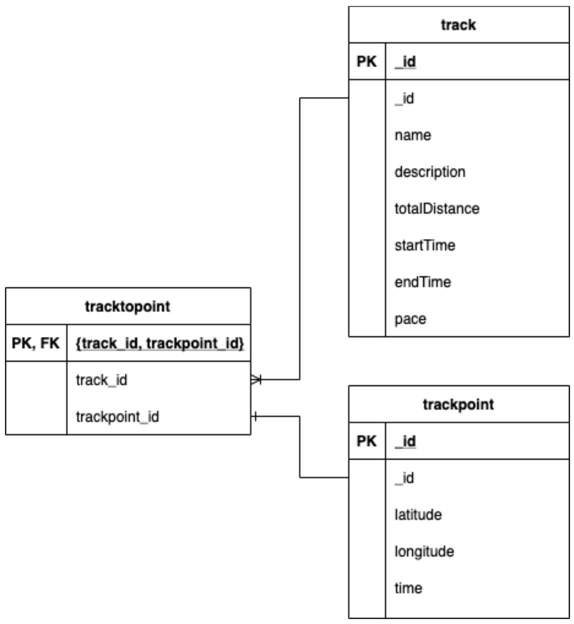


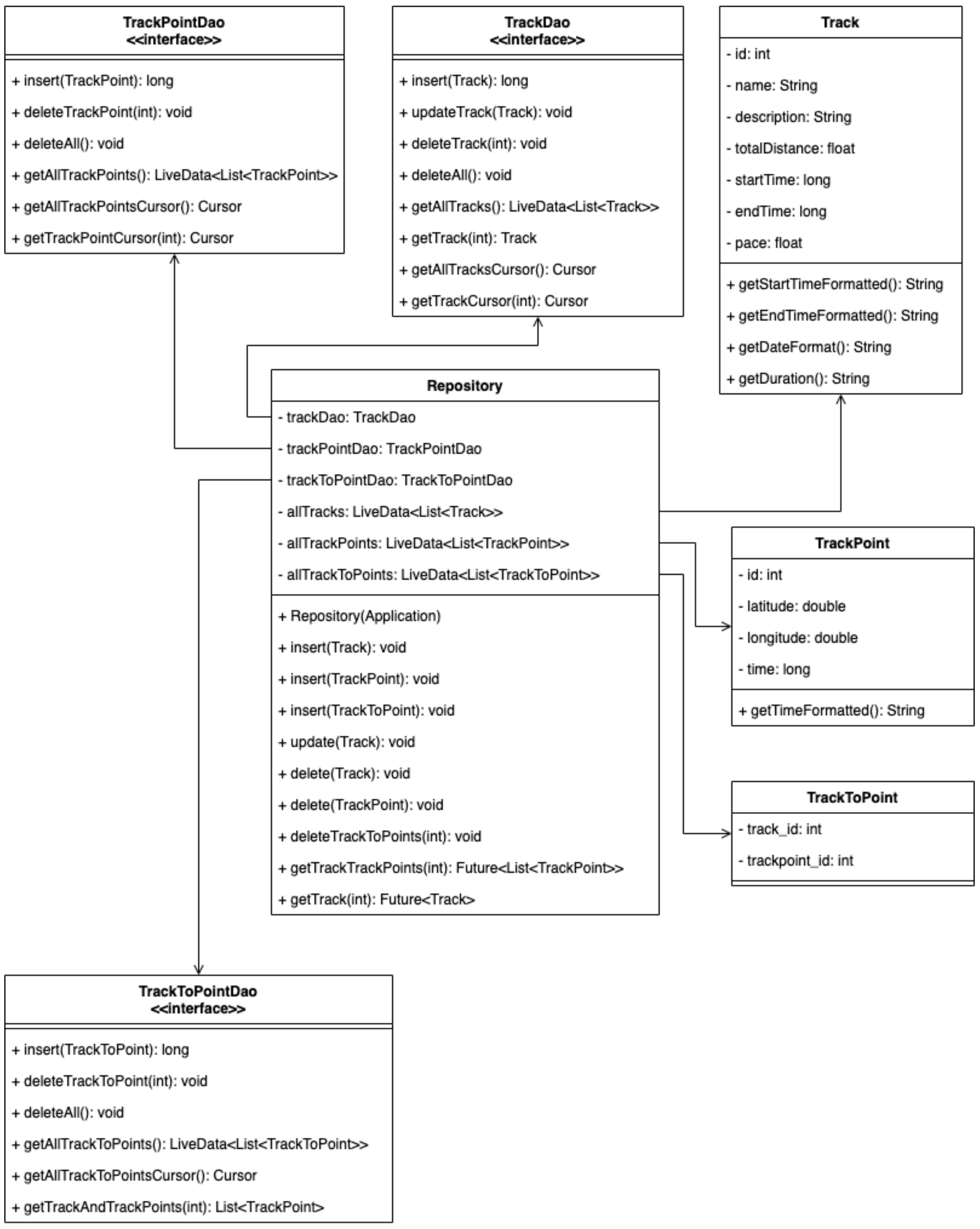
Diagram 1 - Database Structure

There exists three tables. Table ‘track’ has a one to one-or-more relationship to ‘tracktopoint’. Table ‘trackpoint’ has a one-to-one relationship to ‘tracktopoint’. ‘track’ exists to contain general information regarding the tracked run, whereas ‘trackpoint’ contains specific points during a run that was recorded. The LocationListener within the application is set to save a trackpoint with a minimum of every 5 metres, and minimum of every 5 seconds. Using ‘tracktopoint’ as a separate table to relate the two tables further abstracts one from the other, and also gives each table separate purposes.

Content Provider

The application’s provider allows other applications to query and delete through the database. The insert and update methods are not implemented. By allowing applications to query, it allows making use of the data.

Model - Repository



TrackPointDao
<<interface>>
+ insert(TrackPoint): long
+ deleteTrackPoint(int): void
+ deleteAll(): void
+ getAllTrackPoints(): LiveData<List<TrackPoint>>
+ getAllTrackPointsCursor(): Cursor
+ getTrackPointCursor(int): Cursor
TrackDao
<<interface>>
+ insert(Track): long
+ updateTrack(Track): void
+ deleteTrack(int): void
+ deleteAll(): void
+ getAllTracks(): LiveData<List<Track>>
+ getTrack(int): Track
+ getAllTracksCursor(): Cursor
+ getTrackCursor(int): Cursor
Track
- id: int
- name: String
- description: String
- totalDistance: float
- startTime: long
- endTime: long
- pace: float
+ getStartTimeFormatted(): String
+ getEndTimeFormatted(): String
+ getDateFormat(): String
+ getDuration(): String
Repository
- trackDao: TrackDao
- trackPointDao: TrackPointDao
- trackToPointDao: TrackToPointDao
- allTracks: LiveData<List<Track>>
- allTrackPoints: LiveData<List<TrackPoint>>
- allTrackToPoints: LiveData<List<TrackToPoint>>
+ Repository(Application)
+ insert(Track): void
+ insert(TrackPoint): void
+ insert(TrackToPoint): void
+ update(Track): void
+ delete(Track): void
+ delete(TrackPoint): void
+ deleteTrackToPoints(int): void
+ getTrackTrackPoints(int): Future<List<TrackPoint>>
+ getTrack(int): Future<Track>
TrackPoint
- id: int
- latitude: double
- longitude: double
- time: long
+ getTimeFormatted(): String
TrackToPoint
- track_id: int
- trackpoint_id: int
TrackToPointDao
<<interface>>
+ insert(TrackToPoint): long
+ deleteTrackToPoint(int): void
+ deleteAll(): void
+ getAllTrackToPoints(): LiveData<List<TrackToPoint>>
+ getAllTrackToPointsCursor(): Cursor
+ getTrackAndTrackPoints(int): List<TrackPoint>

Diagram 2 - Model-Repository UML

Diagram 2 shows the Entity and Dao classes that are used for the Room database. These make up the ‘Model’ aspect of MVVM. The ‘Repository’ class makes use of these and acts as a way to interact with the database, both reading and writing. Therefore, it has associations with all Entity and Dao classes. The Dao classes has methods that return a Cursor as well as LiveData, to interact with the content provider.

View - ViewModel

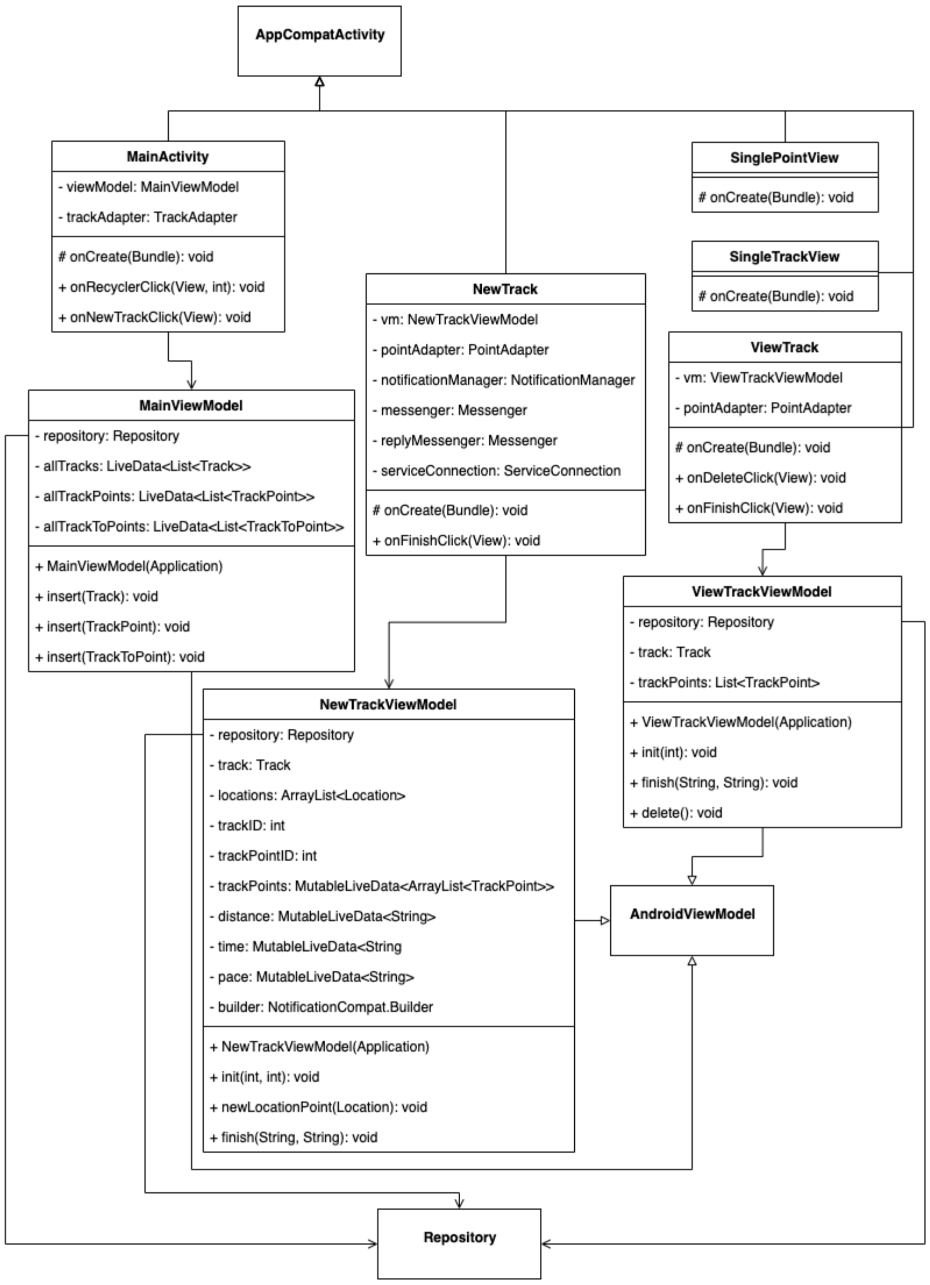

AppCompatActivity
MainActivity
- viewModel: MainViewModel
- trackAdapter: TrackAdapter
# onCreate(Bundle): void
+ onRecyclerClick(View, int): void
+ onNewTrackClick(View): void
SinglePointView
# onCreate(Bundle): void
SingleTrackView
# onCreate(Bundle): void
NewTrack
- vm: NewTrackViewModel
- pointAdapter: PointAdapter
- notificationManager: NotificationManager
- messenger: Messenger
- replyMessenger: Messenger
- serviceConnection: ServiceConnection
# onCreate(Bundle): void
+ onFinishClick(View): void
ViewTrack
- vm: ViewTrackViewModel
- pointAdapter: PointAdapter
# onCreate(Bundle): void
+ onDeleteClick(View): void
+ onFinishClick(View): void
MainViewModel
- repository: Repository
- allTracks: LiveData<List<Track>>
- allTrackPoints: LiveData<List<TrackPoint>>
- allTrackToPoints: LiveData<List<TrackToPoint>>
+ MainViewModel(Application)
+ insert(Track): void
+ insert(TrackPoint): void
+ insert(TrackToPoint): void
ViewTrackViewModel
- repository: Repository
- track: Track
- trackPoints: List<TrackPoint>
+ ViewTrackViewModel(Application)
+ init(int): void
+ finish(String, String): void
+ delete(): void
NewTrackViewModel
- repository: Repository
- track: Track
- locations: ArrayList<Location>
- trackID: int
- trackPointID: int
- trackPoints: MutableLiveData<ArrayList<TrackPoint>>
- distance: MutableLiveData<String>
- time: MutableLiveData<String
- pace: MutableLiveData<String>
- builder: NotificationCompat.Builder
+ NewTrackViewModel(Application)
+ init(int, int): void
+ newLocationPoint(Location): void
+ finish(String, String): void
AndroidViewModel
Repository

Diagram 3 - View-ViewModel UML

Diagram 3 outlines the main activities of the application and its ViewModels. Each ViewModel extends AndroidViewModel to make use of the Repository class, which requires an Application object for its constructor. A Repository instance exists in each ViewModel to be able to access the data without limitations. The size is excessive if passing information solely through Bundles in Intents, due to the likely large number of TrackPoints. Further information regarding each activity's purpose will be explored individually.

MainActivity - MainViewModel
The MainActivity acts as the homepage. It is a point of navigation between previously tracked runs and starting a new tracked run. MainActivity contains a TrackAdapter, to inflate SingleTrackView into a RecyclerView. It only has two additional methods besides existing lifecycle-related methods. These deal with creating Intents to start other activities. The MainActivity also observes the LiveData instances, for the data in the RecyclerView to always be up-to-date.

Since it is the first to start, the MainViewModel is the one that initialises the first instance of the data from the database. It provides access to the Repository's methods.

ViewTrack - ViewTrackViewModel
ViewTrack activity allows the user to view existing tracked runs, and also allows the user to edit its name and description. It includes a PointAdapter, inflating SinglePointView onto a RecyclerView. This showcases all TrackPoints associated to the currently viewed tracked run. This activity includes a delete button, which allows the user to delete the track and all its associated trackpoints from the database. If clicked, it will return to MainActivity after having been deleted.

ViewTrackViewModel provides the necessary data to be shown. As data can not be passed between activities as mentioned, it accesses information by being given the track id. The 'finish' method finalises everything in case of changes, namely editing the name or description.

NewTrack - NewTrackViewModel
NewTrack activity allows user to start tracking their current position by GPS. This begins as soon as this activity starts. It is assumed that the user only starts this activity as they begin running. This is important as it would affect their stored 'pace'. It also includes a PointAdapter, which updates in real-time, as new trackpoints are being notified.

Tracking is done in a foreground service. Therefore, the service starts as soon as the activity begins. It is necessary to use a foreground service as Android limits how frequently an app can retrieve the user's current location while the application is running in the background[1]. As it is a foreground service, a notification will always be present while the Service is running. This ensures the user is aware. The notification updates with the number of trackpoints that is being saved. The notification is set with an up-to-date PendingIntent, such that the user returns to the activity with expected data. The Service stops when the activity is destroyed.

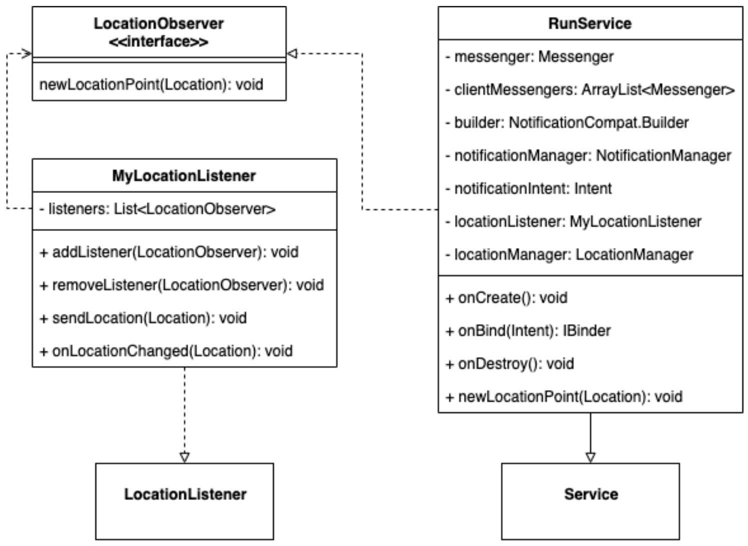

LocationObserver
<<interface>>
newLocationPoint(Location): void
MyLocationListener
- listeners: List<LocationObserver>
+ addListener(LocationObserver): void
+ removeListener(LocationObserver): void
+ sendLocation(Location): void
+ onLocationChanged(Location): void
LocationListener
RunService
- messenger: Messenger
- clientMessengers: ArrayList<Messenger>
- builder: NotificationCompat.Builder
- notificationManager: NotificationManager
- notificationIntent: Intent
- locationListener: MyLocationListener
- locationManager: LocationManager
+ onCreate(): void
+ onBind(Intent): IBinder
+ onDestroy(): void
+ newLocationPoint(Location): void
Service

Diagram 3 - Service UML

Diagram 3 shows the interaction the Service has with the LocationListener. RunService implements the LocationObserver interface, which allows updates from onLocationChanged in MyLocationListener to be passed to the Service as RunService had added itself as a listener to its instance of MyLocationListener. Then, it sends this information in a Message back to NewTrack class, which will update its components for the new information passed.

NewTrack also contains LiveData observed to update the components from the NewTrackViewModel. These variables are updated when a new Location point is given. Therefore,

[1] https://developer.android.com/about/versions/oreo/background-location-limits

a limitation is that the time shown is not real-time. There are gaps between updates. As the user would not be monitoring the time shown, it did not seem necessary to update every second. The ‘finish’ method in NewTrackViewModel would finalise the track and trackpoints by inserting it into all three tables. The view model holds variables ‘trackID’ and ‘trackPointID’ to ensure the correct ids are used when inserting into the ‘tracktopoint’ table. This is necessary as the entities had not been inserted yet, and therefore were not assigned an id (which is specified to be auto-generated in their respective Entity class).