CSCI 5448, Fall 2015

Hike Tracker

Final Report

Ryan Milvenan & Diana Southard

Hike Tracker

Table of Contents

Project Summary

This project is an Android application that tracks which of Colorado's famous "14ers" a user has hiked, how long each hike took, and overall average hiking length over all recorded hikes. The application will also present all mountain peaks locations along with brief summaries when any individual peak is selected which will include the mountain's actual height and which mountain range it is a part of.

Project Requirements

Our initial requirements are listed below, along with a column indicating whether they were finally implemented by the end of the course.

Business Requirements		
ID	**Requirement**	**Implemented?**
BR-001	Only Application Admin can change a mountain peak's name/elevation	*Yes*
BR-002	Application will accurately display all 53 14ers mountain peaks	*Yes*
User Requirements		
ID	**Requirement**	**Implemented?**
UR-001	User will see personal hiking summary on loading page	*Yes*
UR-002	User can click on "14ers" and see mapped location of all 14ers mountain peaks	*Yes*
UR-003	User can click on "History" and see list of saved hiked	*Yes*
UR-004	User can record a new hike.	*Yes*
UR-005	User can manually enter in a new hike.	*Yes*
UR-006	User can edit previously saved hike data.	*Yes*
UR-007	User can update displayed user name.	NO
Functional Requirements		
ID	**Requirement**	**Implemented?**
FR-001	Application will display saved user name on landing page	*Yes*
FR-002	Application will be able to maintain timer throughout any-duration hike	*Yes*

FR-003	When saving new hike, “Peak Name” field will be populated by selected peak’s name.	*Yes*
FR-004	When saving new hike, “Hike Date” field will be populated by current date.	*Yes*
FR-005	After correctly saving hike data, user will be redirected to application landing page.	*Yes*
FR-005	Application will correctly write data to private SQL database stored on phone’s internal memory	*Yes*
Non-Functional Requirements		
ID	**Requirement**	**Implemented?**
NFR-001	Application will load within 5 seconds.	*Yes*
NFR-002	During transition between activities, application will load new activity within 5 seconds	*Yes*
NFR-003	Application will have same behavior on different Android platforms	*Yes*
NFR-004	Application can be used by any level of user expertise	*Yes*

Updating the stored username was a feature that was not implemented by the time the course came to an end. All other features were fully implemented, though the class diagram has noticeably changed.

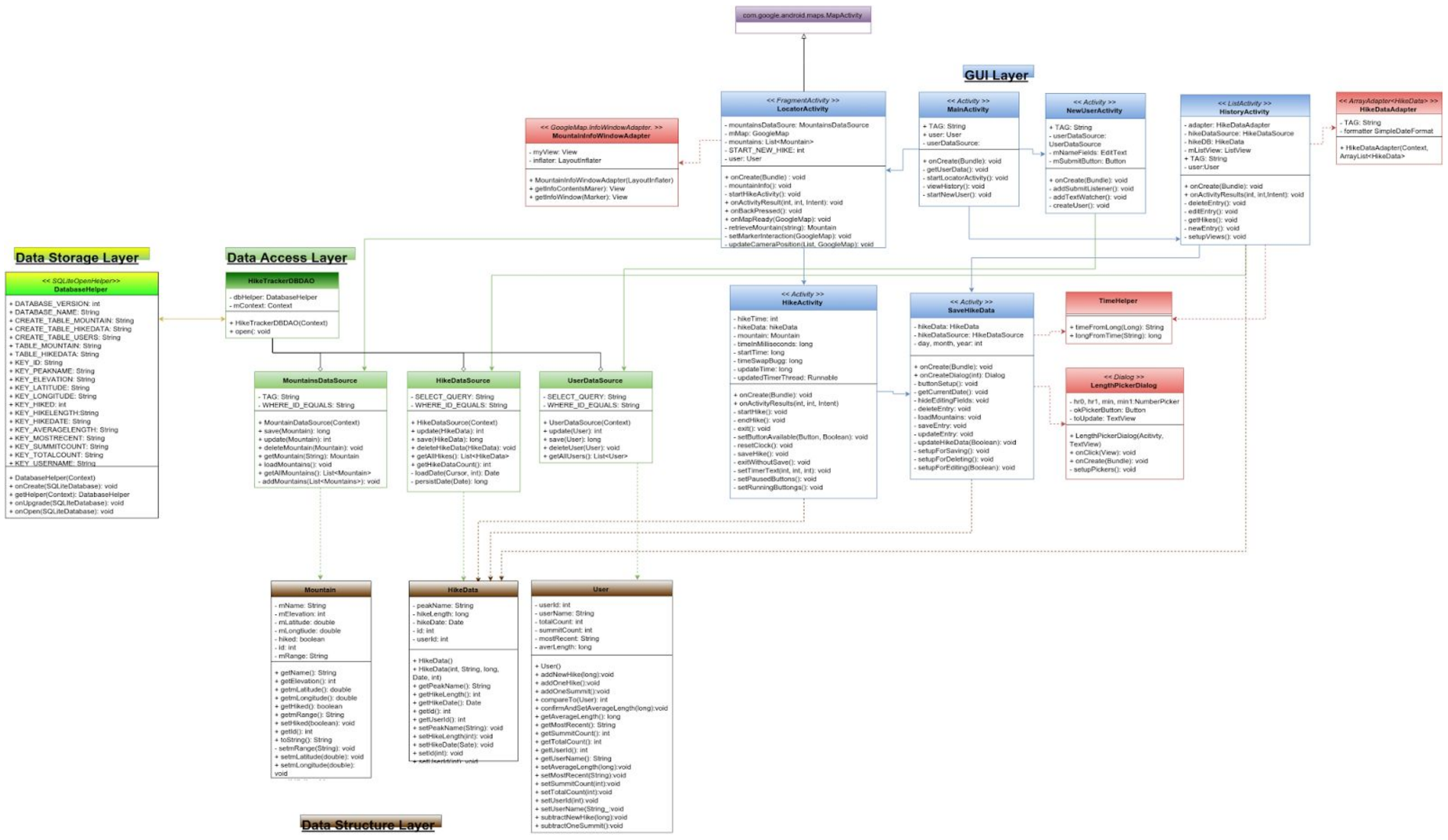
Design Patterns

We decided on using a Layers Architecture in order to allow the application the potential to be upgraded later on in terms of changing the data storage or updating the GUI. This actually proved useful right away. We had originally planned on saving the application data to a simple text file on the phone, but then realized that it would be simple to take advantage of Android’s SQLite database feature. To implement that desired change, we only had to adjust the data storage layer of the application, not adjusting anything at all in the other higher layers.

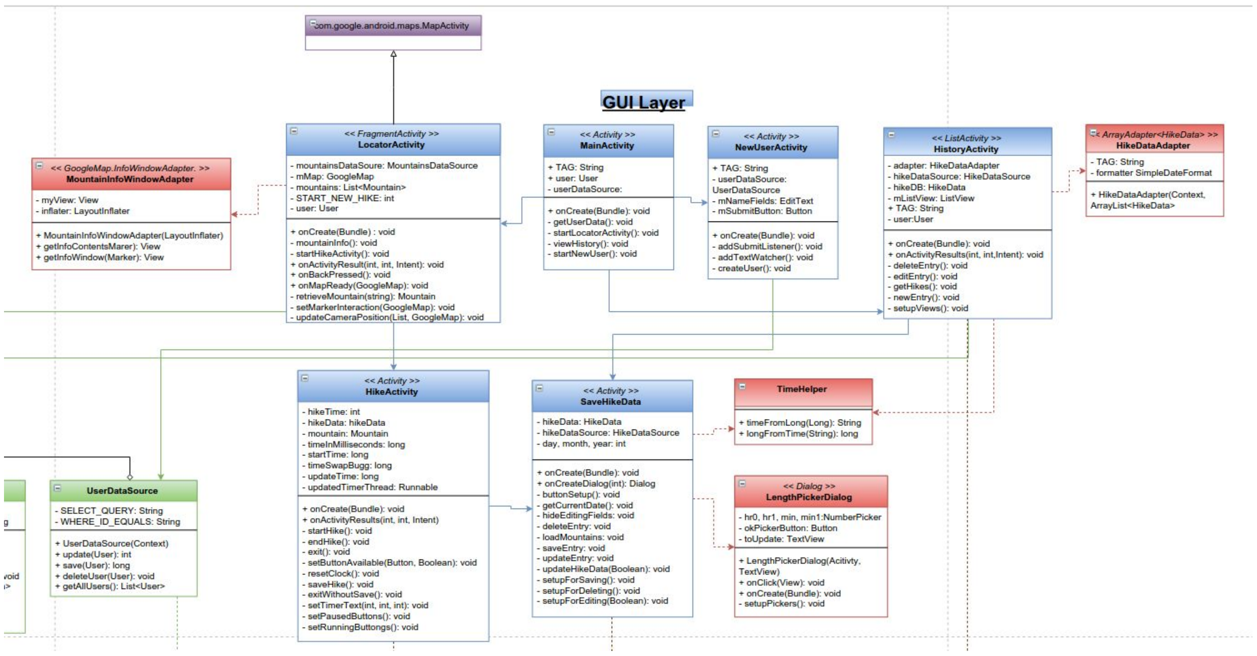
In the future, if this application were to become commercialized on a larger user population, it would be similarly easy to adjust the data storage layer such that data was sent to a cloud storage server instead of taking up space on the user’s phone.

Class Diagram

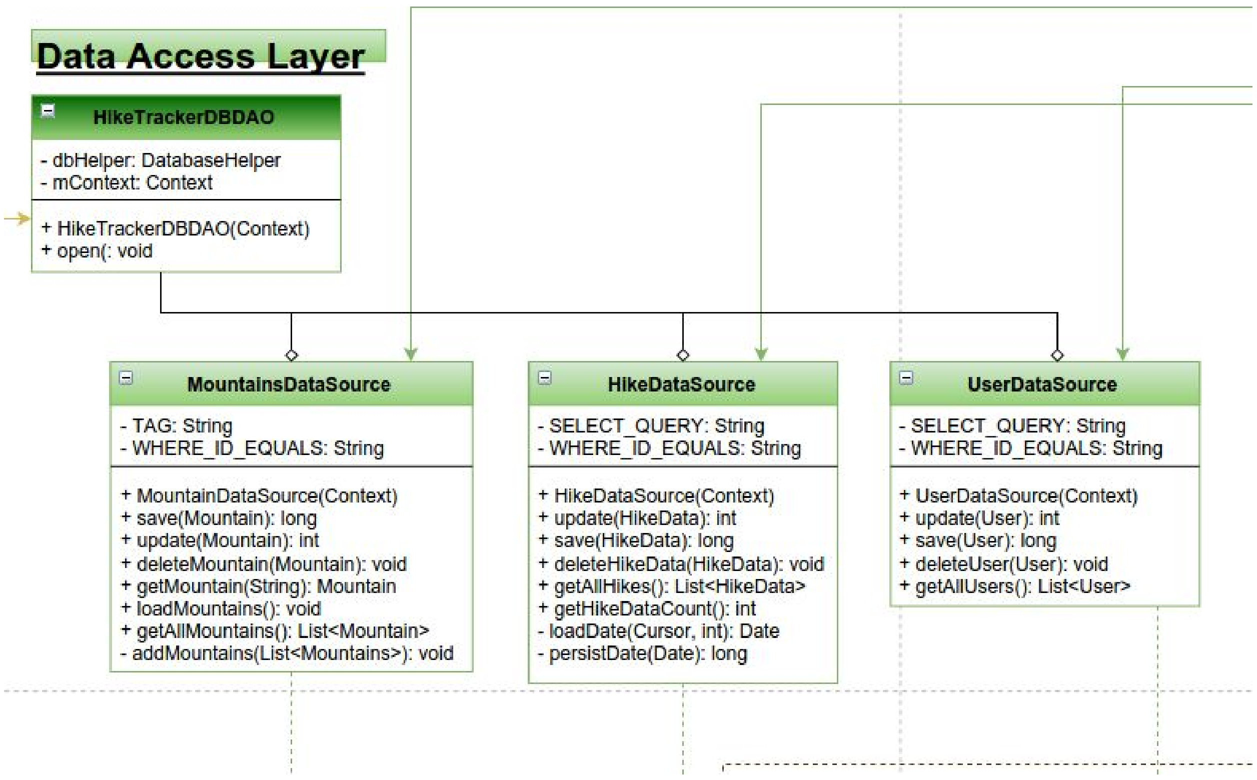
Below is an overview of the entire class diagram. Because the diagram is hard to see all at once, there are several screenshots of the individual layers following the overall image.



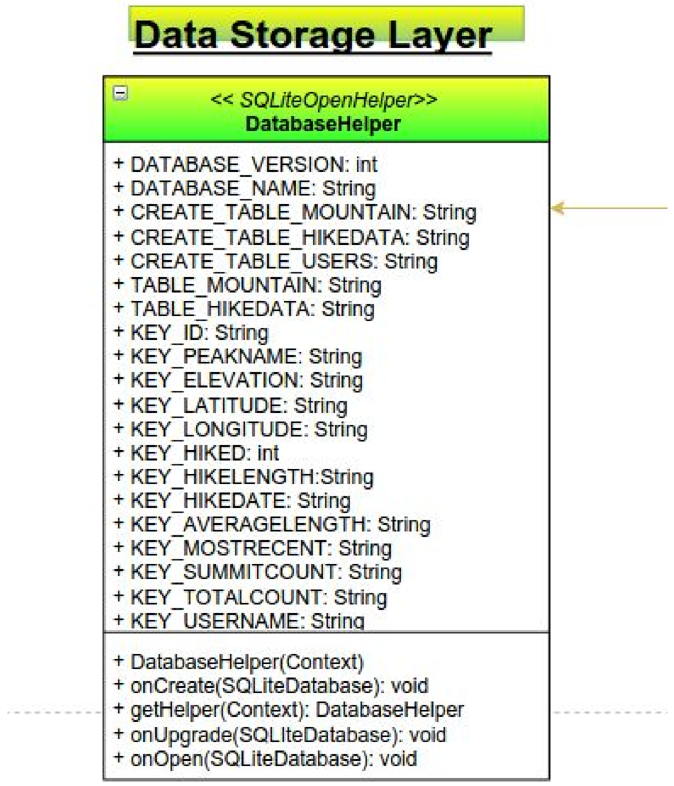
GUI Layer


com.google.android.maps.MapActivity
GUI Layer
<< GoogleMap.InfoWindowAdapter. >>
MountainInfoWindowAdapter
- myView: View
- inflater: LayoutInflater
+ MountainInfoWindowAdapter(LayoutInflater)
+ getInfoContentsMarer): View
+ getInfoWindow(Marker): View
<< FragmentActivity >>
LocatorActivity
- mountainsDataSoure: MountainsDataSource
- mMap: GoogleMap
- mountains: List<Mountain>
- START_NEW_HIKE: int
- user: User
+ onCreate(Bundle) : void
- mountainInfo(): void
- startHikeActivity(): void
+ onActivityResult(int, int, Intent): void
+ onBackPressed(): void
+ onMapReady(GoogleMap): void
- retrieveMountain(string): Mountain
- setMarkerInteraction(GoogleMap): void
- updateCameraPosition(List, GoogleMap): void
<< Activity >>
MainActivity
+ TAG: String
+ user: User
- userDataSource:
+ onCreate(Bundle): void
- getUserData(): void
- startLocatorActivity(): void
- viewHistory(): void
- startNewUser(): void
<< Activity >>
NewUserActivity
+ TAG: String
- userDataSource: UserDataSource
- mNameFields: EditText
- mSubmitButton: Button
+ onCreate(Bundle): void
- addSubmitListener(): void
- addTextWatcher(): void
- createUser(): void
<< ListActivity >>
HistoryActivity
- adapter: HikeDataAdapter
- hikeDataSource: HikeDataSource
- hikeDB: HikeData
- mListView: ListView
+ TAG: String
- user:User
+ onCreate(Bundle): void
+ onActivityResults(int, int,Intent): void
- deleteEntry(): void
- editEntry(): void
- getHikes(): void
- newEntry(): void
- setupViews(): void
<< ArrayAdapter<HikeData> >>
HikeDataAdapter
- TAG: String
- formatter SimpleDateFormat
+ HikeDataAdapter(Context, ArrayList<HikeData>
<< Activity >>
HikeActivity
- hikeTime: int
- hikeData: hikeData
- mountain: Mountain
- timeInMilliseconds: long
- startTime: long
- timeSwapBugg: long
- updateTime: long
- updatedTimerThread: Runnable
+ onCreate(Bundle): void
+ onActivityResults(int, int, Intent)
- startHike(): void
- endHike(): void
- exit(): void
- setButtonAvailable(Button, Boolean): void
- resetClock(): void
- saveHike(): void
- exitWithoutSave(): void
- setTimerText(int, int, int): void
- setPausedButtons(): void
- setRunningButtongs(): void
<< Activity >>
SaveHikeData
- hikeData: HikeData
- hikeDataSource: HikeDataSource
- day, month, year: int
+ onCreate(Bundle): void
+ onCreateDialog(int): Dialog
- buttonSetup(): void
- getCurrentDate(): void
- hideEditingFields: void
- deleteEntry: void
- loadMountains: void
- saveEntry: void
- updateEntry: void
- updateHikeData(Boolean): void
- setupForSaving(): void
- setupForDeleting(): void
- setupForEditing(Boolean): void
TimeHelper
+ timeFromLong(Long): String
+ longFromTime(String): long
<< Dialog >>
LengthPickerDialog
- hr0, hr1, min, min1:NumberPicker
- okPickerButton: Button
- toUpdate: TextView
+ LengthPickerDialog(Acitivty, TextView)
+ onClick(View): void
+ onCreate(Bundle): void
- setupPickers(): void
UserDataSource
- SELECT_QUERY: String
- WHERE_ID_EQUALS: String
+ UserDataSource(Context)
+ update(User): int
+ save(User): long
+ deleteUser(User): void
+ getAllUsers(): List<User>

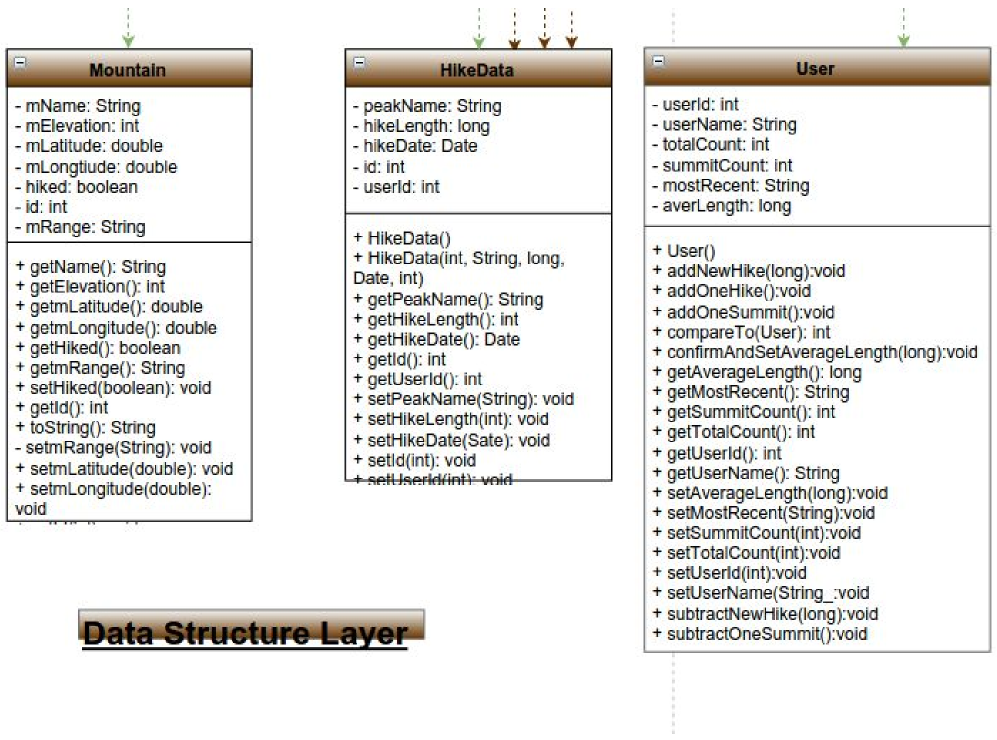
Data Access Layer


Data Access Layer
HikeTrackerDBDAO
- dbHelper: DatabaseHelper
- mContext: Context
+ HikeTrackerDBDAO(Context)
+ open(: void
MountainsDataSource
- TAG: String
- WHERE_ID_EQUALS: String
+ MountainDataSource(Context)
+ save(Mountain): long
+ update(Mountain): int
+ deleteMountain(Mountain): void
+ getMountain(String): Mountain
+ loadMountains(): void
+ getAllMountains(): List<Mountain>
- addMountains(List<Mountains>): void
HikeDataSource
- SELECT_QUERY: String
- WHERE_ID_EQUALS: String
+ HikeDataSource(Context)
+ update(HikeData): int
+ save(HikeData): long
+ deleteHikeData(HikeData): void
+ getAllHikes(): List<HikeData>
+ getHikeDataCount(): int
- loadDate(Cursor, int): Date
- persistDate(Date): long
UserDataSource
- SELECT_QUERY: String
- WHERE_ID_EQUALS: String
+ UserDataSource(Context)
+ update(User): int
+ save(User): long
+ deleteUser(User): void
+ getAllUsers(): List<User>

Data Storage Layer

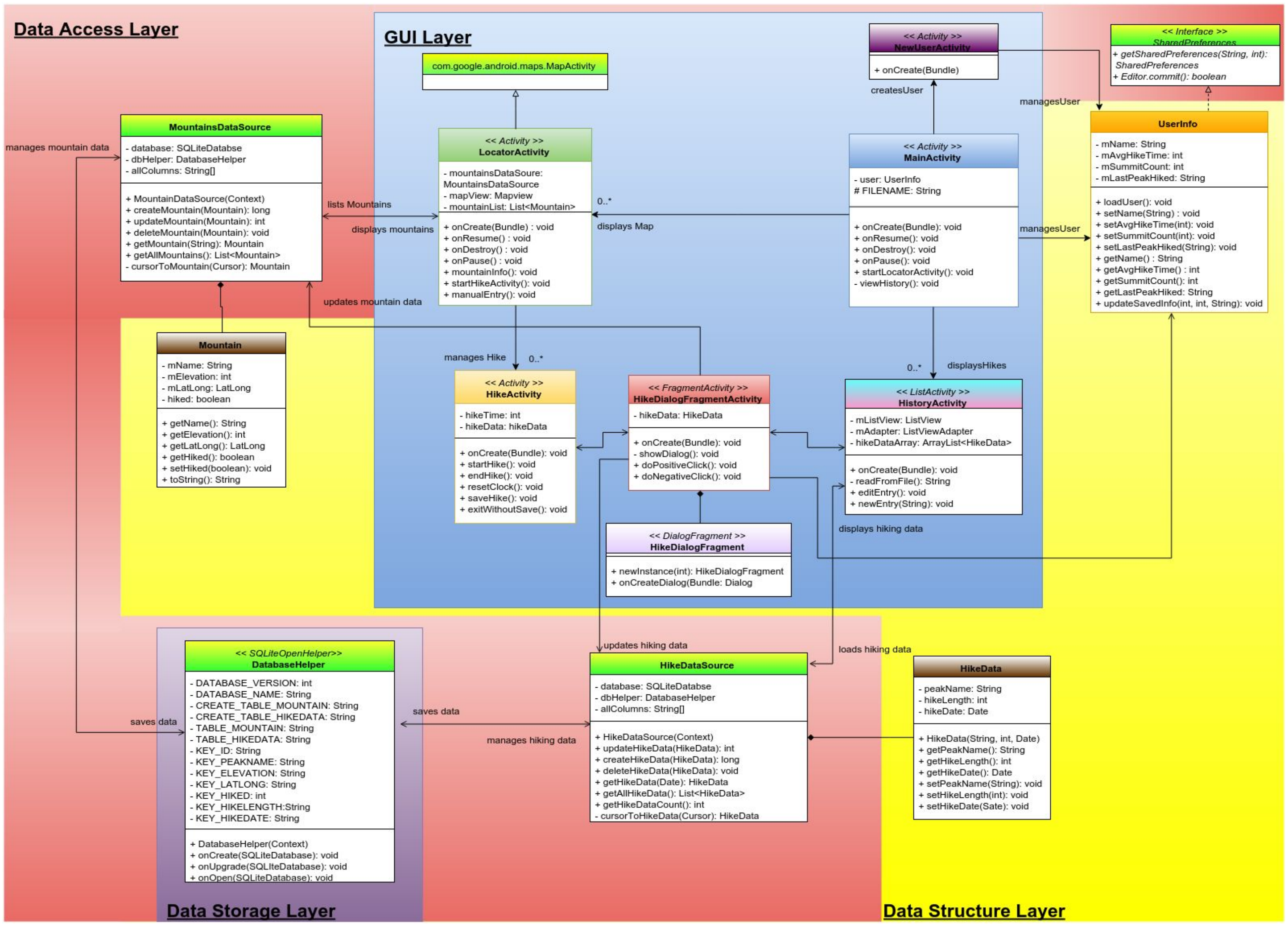


Data Structure Layer



Changes in Class Diagram

Below is the previously submitted class diagram:


Data Access Layer
GUI Layer
Data Storage Layer
Data Structure Layer
com.google.android.maps.MapActivity
<< Activity >> NewUserActivity
+ onCreate(Bundle)
<< Interface >> SharedPreferences
+ getSharedPreferences(String, int): SharedPreferences
+ Editor.commit(): boolean
MountainsDataSource
- database: SQLiteDatabse
- dbHelper: DatabaseHelper
- allColumns: String[]
+ MountainDataSource(Context)
+ createMountain(Mountain): long
+ updateMountain(Mountain): int
+ deleteMountain(Mountain): void
+ getMountain(String): Mountain
+ getAllMountains(): List<Mountain>
- cursorToMountain(Cursor): Mountain
<< Activity >> LocatorActivity
- mountainsDataSoure: MountainsDataSource
- mapView: Mapview
- mountainList: List<Mountain>
+ onCreate(Bundle) : void
+ onResume() : void
+ onDestroy() : void
+ onPause() : void
+ mountainInfo(): void
+ startHikeActivity(): void
+ manualEntry(): void
<< Activity >> MainActivity
- user: UserInfo
# FILENAME: String
+ onCreate(Bundle): void
+ onResume(): void
+ onDestroy(): void
+ onPause(): void
+ startLocatorActivity(): void
- viewHistory(): void
UserInfo
- mName: String
- mAvgHikeTime: int
- mSummitCount: int
- mLastPeakHiked: String
+ loadUser(): void
+ setName(String) : void
+ setAvgHikeTime(int): void
+ setSummitCount(int): void
+ setLastPeakHiked(String): void
+ getName() : String
+ getAvgHikeTime() : int
+ getSummitCount(): int
+ getLastPeakHiked: String
+ updateSavedInfo(int, int, String): void
manages mountain data
lists Mountains
displays mountains
displays Map
0..*
createsUser
managesUser
managesUser
updates mountain data
manages Hike
0..*
0..*
displaysHikes
Mountain
- mName: String
- mElevation: int
- mLatLong: LatLong
- hiked: boolean
+ getName(): String
+ getElevation(): int
+ getLatLong(): LatLong
+ getHiked(): boolean
+ setHiked(boolean): void
+ toString(): String
<< Activity >> HikeActivity
- hikeTime: int
- hikeData: hikeData
+ onCreate(Bundle): void
+ startHike(): void
+ endHike(): void
+ resetClock(): void
+ saveHike(): void
+ exitWithoutSave(): void
<< FragmentActivity >> HikeDialogFragmentActivity
- hikeData: HikeData
+ onCreate(Bundle): void
- showDialog(): void
+ doPositiveClick(): void
+ doNegativeClick(): void
<< ListActivity >> HistoryActivity
- mListView: ListView
- mAdapter: ListViewAdapter
- hikeDataArray: ArrayList<HikeData>
+ onCreate(Bundle): void
- readFromFile(): String
+ editEntry(): void
+ newEntry(String): void
<< DialogFragment >> HikeDialogFragment
+ newInstance(int): HikeDialogFragment
+ onCreateDialog(Bundle: Dialog
displays hiking data
updates hiking data
loads hiking data
<< SQLiteOpenHelper>> DatabaseHelper
- DATABASE_VERSION: int
- DATABASE_NAME: String
- CREATE_TABLE_MOUNTAIN: String
- CREATE_TABLE_HIKEDATA: String
- TABLE_MOUNTAIN: String
- TABLE_HIKEDATA: String
- KEY_ID: String
- KEY_PEAKNAME: String
- KEY_ELEVATION: String
- KEY_LATLONG: String
- KEY_HIKED: int
- KEY_HIKELENGTH:String
- KEY_HIKEDATE: String
+ DatabaseHelper(Context)
+ onCreate(SQLiteDatabase): void
+ onUpgrade(SQLIteDatabase): void
+ onOpen(SQLiteDatabase): void
saves data
saves data
manages hiking data
HikeDataSource
- database: SQLiteDatabse
- dbHelper: DatabaseHelper
- allColumns: String[]
+ HikeDataSource(Context)
+ updateHikeData(HikeData): int
+ createHikeData(HikeData): long
+ deleteHikeData(HikeData): void
+ getHikeData(Date): HikeData
+ getAllHikeData(): List<HikeData>
+ getHikeDataCount(): int
- cursorToHikeData(Cursor): HikeData
HikeData
- peakName: String
- hikeLength: int
- hikeDate: Date
+ HikeData(String, int, Date)
+ getPeakName(): String
+ getHikeLength(): int
+ getHikeDate(): Date
+ setPeakName(String): void
+ setHikeLength(int): void
+ setHikeDate(Sate): void

For the reader who is not completely familiar with Android, each Activity is similar to a new screen the phone opens up to, depending on the user's actions. Initially, we were going to use a custom DialogFragment to display the dialog for saving/deleting/editing a hike. It was instead easier to just create a new Activity for that purpose. From the old class, the HikeDialogFragmentActivity and HikeDialogFragment classes were both condensed into SaveHikeDataActivity shown in the new class.

It also made sense to store the user data in another table in our SQLite database rather than relying on a different method to persist data (using the SharedPreferences class). This also allows for the user data to be easily transferred to a cloud server since the data is already neatly packaged up.

Aside from those two large changes, the main design on the class diagram remains the same. We added multiple new helper functions in our classes as we went along to reduce repeated code, and we also had to add a few helper classes due to the nature of Android design. For instance, we wanted to display a custom information window when the user clicked on a map marker during the LocatorActivity. That required us to create a custom GoogleMap.InfoWindowAdapter class. Also, when we wanted to display a list of previously saved hikedata, we had to create a custom ArrayListAdapter class to conform with Android's method of displaying lists. Neither of us are experts at Android development by any means, so we learned a lot on the go and added onto our initial design. In the future, we will be better prepared to design for Android specific needs for this type of project.

In regards to the class methods that were added after the original class diagram, it came from not thinking through what each class would be responsible for and what sort of functions that class would repeatedly call. For instance, in the Activities which displayed buttons, all the displayed buttons needed to first be initialized and set up. Our original class diagrams did not take this into account. The buttons could be set up in the same method as the onCreate method, which is used to create the Activity the first time it is opened, but that would result in an unnecessarily large function where it would be messy trying to find the lines of code dealing specifically with the button setup. Instead of doing that, we relegated the button setup to dedicated functions with the result of having cleaner, easier-to-understand code that would also be easier to maintain if we wanted to update the button setup later on.

We now have more experience with some of the details that should be better thought through and planned out than we did before. This should result in better initial design during our future software developments.

Lessons Learned

One of the major lessons learned was the reward that comes from doing good software design prior to coding, particularly when working with a group. Both of the partners in this group have stressed time schedules and any time spent working on the project needed to have valuable results if we were to make the project deadlines.

By having a well-thought-out design prior to starting to implement anything, we were able to use all of our time very efficiently. There was rarely any moment of "What to do next?" because the project had been effectively mapped out ahead of time. Most of the methods had been planned out in terms of input/output, leaving the implementation almost simple to do.

The class interactions were well understood resulting in little to no trouble in integrating them when the time came. And it was very easy to divide up the work or to follow along with what the other partner had been doing because there were no coding surprises.

Another thing that we learned was to spend more time on creating that well-crafted design. We had to add methods to our classes that, in hindsight, should have been originally part of the class diagram. For instance, the application tracks the user’s average time to hike each peak. Originally, we only had a set method to set the user’s average time to some new time. When we started to implement the code, we realized that we would rather have a method that would take in a new time and find/set the user’s new average time on its own. That would be the method that would be repeatedly called each time the user added a new hiking time. Instead of thinking about what methods we would actually need, we had just used the regular getter and setter methods. For future designs, we will be a lot more aware of how the data in our classes truly needs to be manipulated. Additionally, as previously mentioned, with the experience we gained in making Android apps, thorough future planning seems much more attainable.