CS 428/429
MBR - Mobile Bus Routing
Documentation

Scott Blessing [blessin2]
Jimmy Guo [jguo14]
Ryan Gisleson [rgisle2]
Varun Goverdhan [goverdh2]
Sirapob Nanthayapirom [nanthay2]
Richard Shen [rnshen2]
Derek Vidovic [vidovic2]

Professor: Tao Xie
TA: Yiming Jiang

Contents

Overview

Mobile Bus Routing (MBR) is an Android App designed to assist with day-to-day scheduling and car-less navigation for residents of Champaign County. MBR makes use of the Champaign-Urbana Mass Transit District (CUMTD) bus system to navigate users to their destinations. MBR allows users to choose events from their device calendar to be scheduled. Once scheduled, MBR will determine when the user should leave to arrive at the event on time. At that time, the device will activate an alarm to notify the user and provide routing information to get to the event. MBR also acts as a general alarm manager and navigation system.

Process

Our team followed the general principles of eXtreme Programming (XP) while working on MBR. We used two-week iterations to measure progress, during which each programming pair was responsible for a specific task. Pairs needed to implement, test, and refactor their assigned task during the iteration. This was done in individual branches, which were then merged together at the end of each iteration. Since our team consisted of seven members, we broke into two pairs and a triplet each iteration. After completing our planning game (which assigned time estimates to tasks), the largest task would be given to the triplet team. We would end iterations with a "merge meeting", in which at least one member from each pair would meet up to merge all their code into the master branch and help resolve any conflicts. We followed XP's principle of writing functional code, making sure it passed tests, and then refactoring. All work done on the project was recorded on our team Wiki and we communicated through Facebook. We used GitHub for version control, and Gradle as a build system. We used Android Studio as our IDE.

We agreed as a team to many program formatting and design paradigms before beginning. For formatting, we decided to follow Oracle's Java formatting standards, such as for casing, bracket placement, and tabbing. We agreed to use JavaDoc for all class descriptions and for methods that aren't self-describing. We also decided on using the try-with-resources style when dealing with I/O and formatted all of our error handlers with multicatch. We followed the ideal of "commit early and often" while coding, and made a best effort to isolate change types between commits (i.e. no tests and refactors in the same commit).

Requirements & Specifications

Use Cases

Actor	Goal
User	Choose Calendars to pull Events from
User	Define a location
User	Choose Events to be scheduled
Calendar	Get a list of upcoming Events
User	Navigate to a location using bus routes
User	Generate a route to any saved location
CUMTD API	Get upcoming bus departures at a stop
CUMTD API, User	Find all nearby bus stops
User	Set an offset for arrival time to an Event
User	Set a maximum walking distance for navigation

Choose Calendars to Pull Events From

Google Calendar allows the user to have multiple calendars to which events are associated. Users should be able to choose which of these calendars they want MBR to pull events from. For instance, a Calendar that consists of people's birthdays or of general holidays would clutter the UI, so a user could choose to remove them. The user should have the option to freely toggle the use of calendars.

Define a Location

The user should be able to associate a Location name with a set of coordinates. For example, a user could associate "Ryan's House" with "123 Main St, Urbana IL". This address association would be usable as a shortcut throughout the app.

Get a List of Upcoming Events

The user should be able to display a list of all events within the next 24 hours. These events should only be those associated with user-allowed calendars (see 'Choose Calendars to Pull Events from'). The events should be marked if they are currently part of the user's schedule.

Choose Events to be Scheduled

The user should be able to choose events they want added to their schedule. By adding an event to the schedule, the user will receive an alert when they should leave for the event. This alert should be received relative to the user's travel time from the event according to CUMTD. The user should also be able to remove an event from their schedule.

Navigate to a Location using Bus Routes

The user can input a starting and destination location and receive a route for navigation. This route should make use of the CUMTD API to find an appropriate bus route for the journey. This route should be drawn on a map, with indicators for boarding and exiting busses.

Generate a Route to any Saved Location

The user has a fast way to generate a route from their current location to any saved address.

Find all Nearby Bus Stops

The user can quickly get a list of all nearby bus stops

Get Upcoming Bus Departures at a Stop

The user can learn when busses will be arriving at a specified bus stop. The bus type, direction, and arrival time should all be listed.

Set an Arrival Offset Time for Events

The user can specify how early they wish to arrive to events. This time should be taken into account when scheduling events (see 'Choose Events to be Scheduled').

Set a Maximum Walking Distance

The user can specify how far they are willing to walk when routes are generated. No route generated should exceed the given distance for non-bus portions of the journey.

User Stories

- Use CUMTD’s API to display bus route on a map
- Find and display the device’s current location on a map
- Find and display the Android calendars and events on the device
- Choose which Android events are added to the schedule
- Find information about nearby bus stops and their upcoming departures
- Set an alarm
- Get directions from one location to another, and display on a map
- Have alerts generated automatically from the schedule, based on travel time

- Choose which Android calendars events are taken from
- Allow recurring events to be scheduled automatically
- Create, modify, and remove address mappings
- Quickly generate and display a route from current device location to a saved address

Architecture & Design

Overview

Our code is divided into five major sections:

1. **Accessing User Calendars**: Accessing the Android calendar and converting the data into usable data structures. Most if this is behind the scenes and not directly interacted with by the user.
2. **Scheduling Events**: Allowing the user to add events to their schedule. Controls which events are presented, rules for automatic selection, and storing location mappings.
3. **Managing Alarms**: Controls how alarms are created and their effects when triggered. Acts as the "schedule" the user adds events to.
4. **Navigation**: Handles all activities that rely on CUMTD or Google Maps. CUMTD queries are converted into usable data structures, which are used to display routes on a map, list stops and their departures, and allow the user to generate routes.
5. **Settings**: Provides global application settings for the user to customize.

Since this is an Android application, our code follows all Android standards. All activity classes (displays) are named as `<functionality>Activity.java`. In addition, all computation-heavy code, such as calling APIs, is done inside of `AsyncTasks`. Additionally, please note that all Activity classes are part of the "activities" package, as well as dialogs being in the "ui" package.

Accessing User Calendars (mbr.calendar)

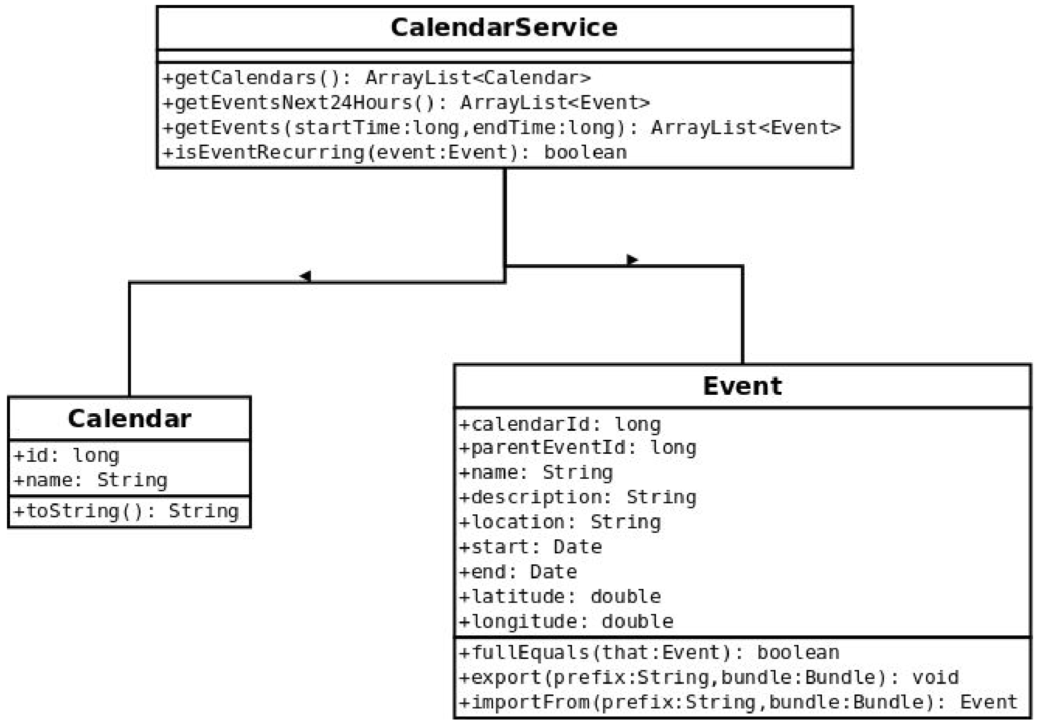
CalendarService
+getCalendars(): ArrayList<Calendar>
+getEventsNext24Hours(): ArrayList<Event>
+getEvents(startTime:long,endTime:long): ArrayList<Event>
+isEventRecurring(event:Event): boolean
Calendar
+id: long
+name: String
+toString(): String
Event
+calendarId: long
+parentEventId: long
+name: String
+description: String
+location: String
+start: Date
+end: Date
+latitude: double
+longitude: double
+fullEquals(that:Event): boolean
+export(prefix:String,bundle:Bundle): void
+importFrom(prefix:String,bundle:Bundle): Event

Fig 1: Diagram of Calendar classes

CalendarService

`CalendarService` is an API for interfacing with the Android calendar system. This class contains methods for retrieving Android calendars, events, and instances from the user's calendar data using the `ContentResolver.query()` method.

Calendar

`Calendar` is a data class for storing data about Android calendars. `CalendarService` converts Android calendar data into `Calendar` objects.

Event

`Event` is a data class for storing data about Android instances (not Android events). Instances are being used because Android events do not repeat for each occurrence, whereas instances do. Each `Event` object stores a reference to its parent event's id. `CalendarService` converts Android instance data to `Event` objects.

Scheduling Events (mbr.event_selection)

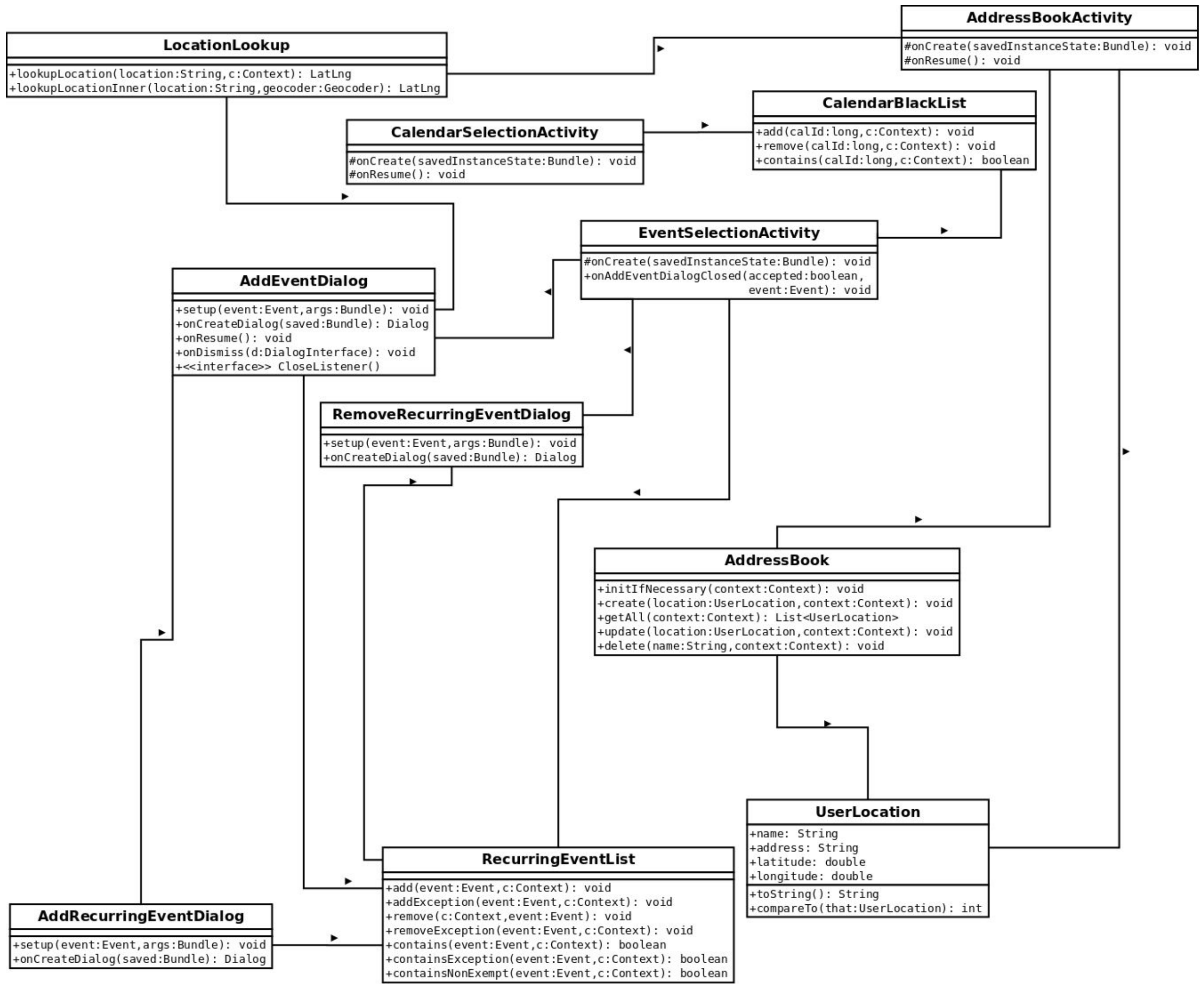
AddressBookActivity
#onCreate(savedInstanceState:Bundle): void
#onResume(): void
LocationLookup
+lookupLocation(location:String,c:Context): LatLng
+lookupLocationInner(location:String,geocoder:Geocoder): LatLng
CalendarBlackList
+add(calId:long,c:Context): void
+remove(calId:long,c:Context): void
+contains(calId:long,c:Context): boolean
CalendarSelectionActivity
#onCreate(savedInstanceState:Bundle): void
#onResume(): void
EventSelectionActivity
#onCreate(savedInstanceState:Bundle): void
+onAddEventDialogClosed(accepted:boolean, event:Event): void
AddEventDialog
+setup(event:Event,args:Bundle): void
+onCreateDialog(saved:Bundle): Dialog
+onResume(): void
+onDismiss(d:DialogInterface): void
+<<interface>> CloseListener()
RemoveRecurringEventDialog
+setup(event:Event,args:Bundle): void
+onCreateDialog(saved:Bundle): Dialog
AddressBook
+initIfNecessary(context:Context): void
+create(location:UserLocation,context:Context): void
+getAll(context:Context): List<UserLocation>
+update(location:UserLocation,context:Context): void
+delete(name:String,context:Context): void
UserLocation
+name: String
+address: String
+latitude: double
+longitude: double
+toString(): String
+compareTo(that:UserLocation): int
RecurringEventList
+add(event:Event,c:Context): void
+addException(event:Event,c:Context): void
+remove(c:Context,event:Event): void
+removeException(event:Event,c:Context): void
+contains(event:Event,c:Context): boolean
+containsException(event:Event,c:Context): boolean
+containsNonExempt(event:Event,c:Context): boolean
AddRecurringEventDialog
+setup(event:Event,args:Bundle): void
+onCreateDialog(saved:Bundle): Dialog

Fig 2: Diagram of Event Selection Classes

CalendarSelectionActivity

`CalendarSelectionActivity` shows the user a list of all the calendars on their device, and allows them to toggle whether each calendar's events should be used for scheduling. Calendar data is retrieved from `CalendarService`, and the user's selections are saved by `CalendarBlacklist`.

CalendarBlacklist

`CalendarBlacklist` contains a set of static methods for saving and loading the set of calendar ids the user has chosen not to use for scheduling. The user can modify the blacklist from the `CalendarSelectionActivity`.

EventSelectionActivity

`EventSelectionActivity` displays a list of `Events` that will occur within the next 24 hours to the user. Any `Event` associated with a blacklisted calendar will not be displayed (See `CalendarBlacklist`). If an `Event` has been marked as recurring, it will automatically be added to the schedule when the activity is opened (See `RecurringEventList`). `Events` are marked if they have already been added to the schedule. The user can tap on `Events` to toggle them on/off in the schedule. If the `Event` is being added, the `AddEventDialog` is triggered. Otherwise, the `Event` is removed from the schedule, and the `RemoveRecurringEventDialog` is triggered (if applicable - See `RecurringEventList`).

AddEventDialog

`AddEventDialog` is a pop-up that facilitates adding a selected `Event` to the schedule. If the `Event` does not have a location, `AddEventDialog` will display an input field to the user asking for an address. If the user cancels the dialog, the event is not scheduled. Once an address has been provided, the `Event` is scheduled. If the `Event` already had a location, the input window is skipped. `AddEventDialog` will also save the `Event`'s associated address to the `AddressBook`. If the `Event` is recurring, the `AddRecurringEventDialog` will be triggered once the `Event` has been added.

AddRecurringEventDialog

`AddRecurringEventDialog` prompts the user to set a recurring `Event` to be automatically selected. If the user chooses ‘yes,’ any future instances of the recurring `Event` will be automatically added to the schedule (See `RecurringEventList`).

RemoveRecurringEventDialog

`RemoveRecurringEventDialog` prompts the user to stop the automatic selection of a recurring Event. If the user chooses ‘yes,’ the `Event`’s recurrence rule is removed from the `RecurringEventList`. If the user chooses ‘no,’ the specific `Event` is marked as an exception in the `RecurringEventList`, but the recurrence rule is not removed.

RecurringEventList

`RecurringEventList` is a set of static methods for saving and loading recurrence rules and their exceptions to memory. If an `Event` has a recurrence rule, then `EventSelectionActivity` will automatically add it to the schedule. However, if the specific `Event` also has an exception, this will not occur.

LocationLookup

`LocationLookup` contains a static method `lookupLocation(String address, Context c)` that accesses the Google Geocoder API to determine if the given address is valid. If the address is valid, its `LatLng` is returned, otherwise `null` is returned.

UserLocation

`UserLocation` is a data class that stores a mapping from a location name to an address (and its associated latitude and longitude).

AddressBook

`AddressBook` is a set of static methods for saving and loading location name to address mappings in device memory.

AddressBookActivity

`AddressBookActivity` displays all of the entries in `AddressBook` and allows the user to modify or delete them. Additionally, the user can directly add new `UserLocations` to the `AddressBook`.

Managing Alarms (mbr.alarm)

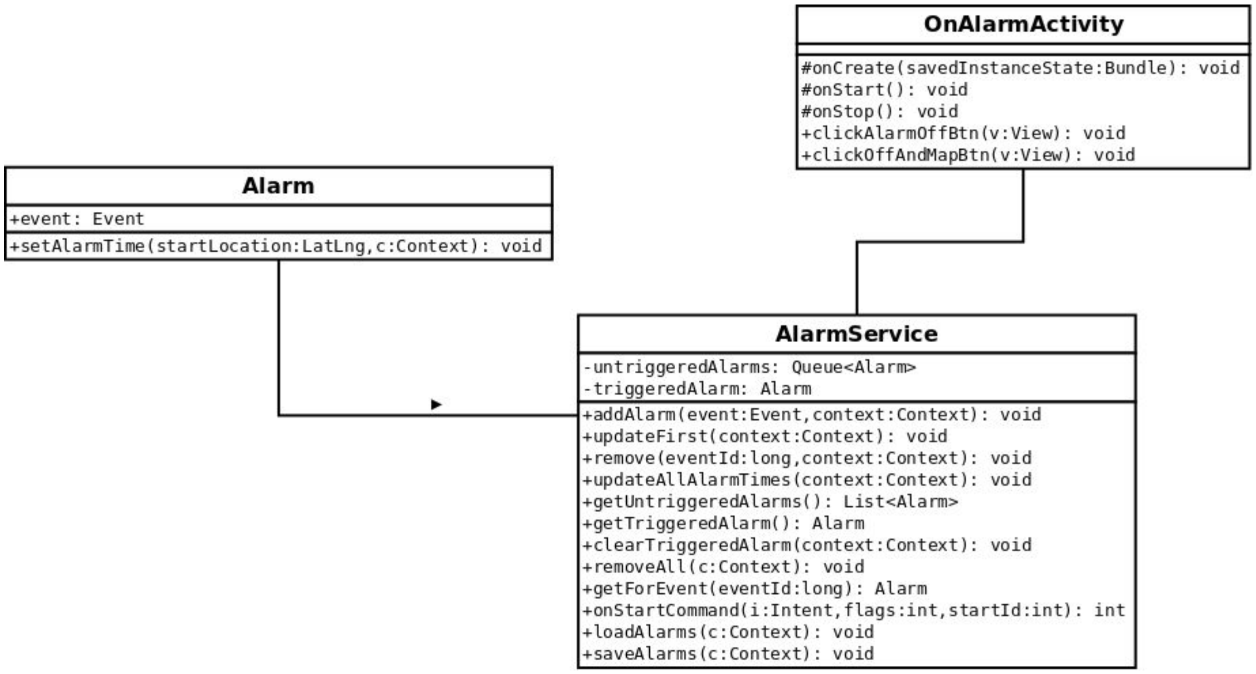
OnAlarmActivity
#onCreate(savedInstanceState:Bundle): void
#onStart(): void
#onStop(): void
+clickAlarmOffBtn(v:View): void
+clickOffAndMapBtn(v:View): void
Alarm
+event: Event
+setAlarmTime(startLocation:LatLng,c:Context): void
AlarmService
-untriggeredAlarms: Queue<Alarm>
-triggeredAlarm: Alarm
+addAlarm(event:Event,context:Context): void
+updateFirst(context:Context): void
+remove(eventId:long,context:Context): void
+updateAllAlarmTimes(context:Context): void
+getUntriggeredAlarms(): List<Alarm>
+getTriggeredAlarm(): Alarm
+clearTriggeredAlarm(context:Context): void
+removeAll(c:Context): void
+getForEvent(eventId:long): Alarm
+onStartCommand(i:Intent,flags:int,startId:int): int
+loadAlarms(c:Context): void
+saveAlarms(c:Context): void

Fig 3: Diagram of Alarm classes

Alarm

`Alarm` calculates and stores when a system alarm associated with an `Event` should occur. By calling the `setAlarmTime(LatLng startLoc, Context c)` method, the `Alarm` will calculate the travel time from `startLoc` to the `Event`'s location and use that to set an appropriate alarm time. This travel time will take into account the user's max walking distance and arrival time offset (see `Settings`).

AlarmService

`AlarmService` manages the schedule by saving, loading, updating, and removing `Alarms`. Additionally, `AlarmService` sets up Android system alarms using Android's `AlarmManager`. When a system alarm occurs, `AlarmService` checks if an `Alarm` exists with the current time. If so, it responds by activating the `OnAlarmActivity` and providing it with the triggered `Alarm`.

OnAlarmActivity

`OnAlarmActivity` displays the full list of upcoming `Alarms`. Additionally, if opened when `AlarmService` has detected an activated `Alarm`, the activity will override system sleep and play a sound effect. The activity will then display the name of the `Event` associated with the `Alarm` and provide the user options to disable the alarm noise and to begin navigation to the `Event`. Choosing to navigate will launch `MapActivity` with the user's current location and the `Event`'s location as parameters.

Navigation (mbr.directions)

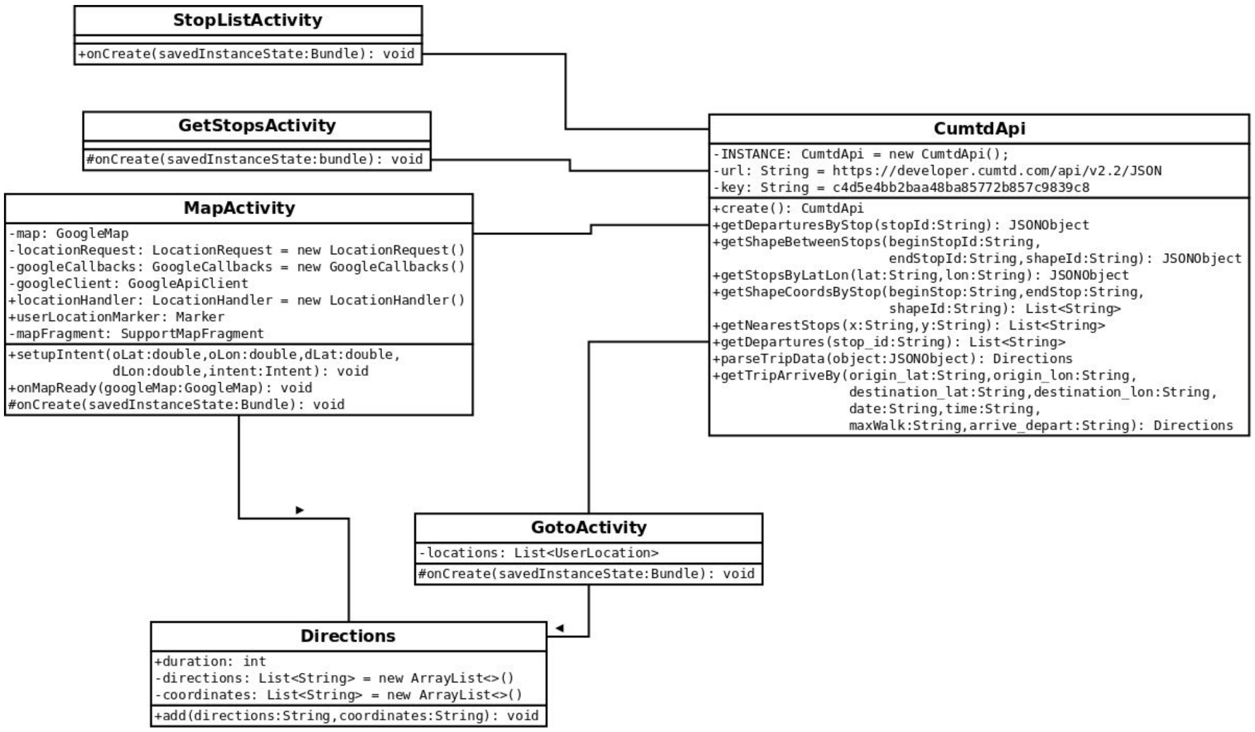
StopListActivity
+onCreate(savedInstanceState:Bundle): void
GetStopsActivity
#onCreate(savedInstanceState:bundle): void
MapActivity
-map: GoogleMap
-locationRequest: LocationRequest = new LocationRequest()
-googleCallbacks: GoogleCallbacks = new GoogleCallbacks()
-googleClient: GoogleApiClient
+locationHandler: LocationHandler = new LocationHandler()
+userLocationMarker: Marker
-mapFragment: SupportMapFragment
+setupIntent(oLat:double,oLon:double,dLat:double, dLon:double,intent:Intent): void
+onMapReady(googleMap:GoogleMap): void
#onCreate(savedInstanceState:Bundle): void
CumtdApi
-INSTANCE: CumtdApi = new CumtdApi();
-url: String = https://developer.cumtd.com/api/v2.2/JSON
-key: String = c4d5e4bb2baa48ba85772b857c9839c8
+create(): CumtdApi
+getDeparturesByStop(stopId:String): JSONObject
+getShapeBetweenStops(beginStopId:String, endStopId:String,shapeId:String): JSONObject
+getStopsByLatLon(lat:String,lon:String): JSONObject
+getShapeCoordsByStop(beginStop:String,endStop:String, shapeId:String): List<String>
+getNearestStops(x:String,y:String): List<String>
+getDepartures(stop_id:String): List<String>
+parseTripData(object:JSONObject): Directions
+getTripArriveBy(origin_lat:String,origin_lon:String, destination_lat:String,destination_lon:String, date:String,time:String, maxWalk:String,arrive_depart:String): Directions
GotoActivity
-locations: List<UserLocation>
#onCreate(savedInstanceState:Bundle): void
Directions
+duration: int
-directions: List<String> = new ArrayList<>()
-coordinates: List<String> = new ArrayList<>()
+add(directions:String,coordinates:String): void

Fig 4: Diagram of Navigation classes

CumtdApi

`CumtdApi` contains a set of methods for interfacing with CUMTD's public API. This class will automatically format requests and parse results into appropriate data types, such as `Directions`.

Directions

Directions is a data class that stores two equal-length arrays of Strings. One array consists of written directions, and the other consists of coordinates.

MapActivity

`MapActivity` displays a route and its accompanying directions to the user. Starting and ending `LatLngs` can be passed to the activity through an intent. Any coordinates not provided will be prompted from the user after the page loads.

GoToActivity

`GoToActivity` presents the user with a list of all the addresses stored in the `AddressBook`. If the user selects a location, their device's current location and the chosen address' coordinates are passed to `MapActivity` as starting and ending locations.

GetStopsActivity

`GetStopsActivity` finds and displays the names of the five closest CUMTD stops to the device's current location. Selecting a stop will launch the `StopListActivity` for the chosen stop.

StopListActivity

`StopListActivity` displays a list of CUMTD departures for a given stop provided by `GetStopsActivity`.

Settings

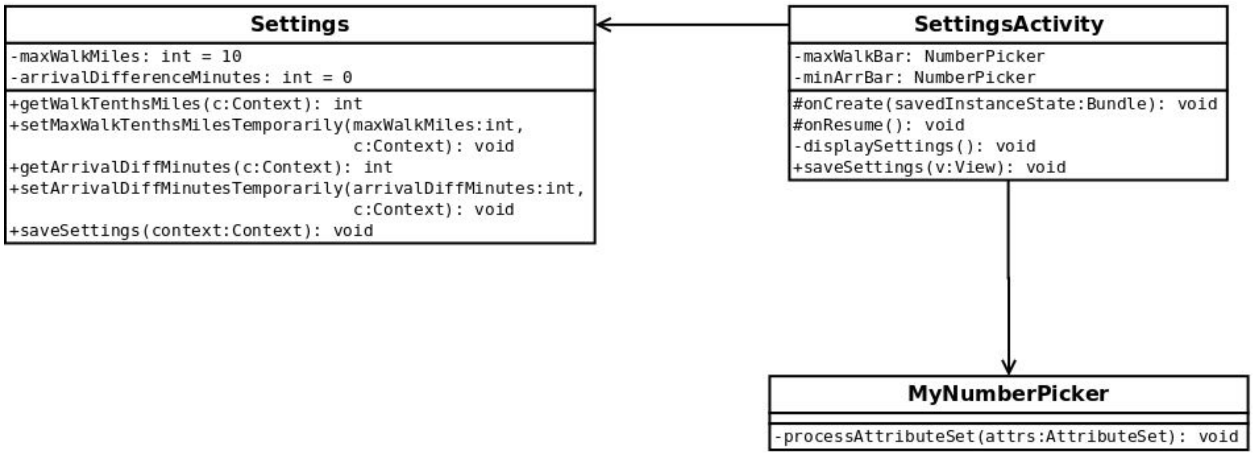
Settings
-maxWalkMiles: int = 10
-arrivalDifferenceMinutes: int = 0
+getWalkTenthsMiles(c:Context): int
+setMaxWalkTenthsMilesTemporarily(maxWalkMiles:int, c:Context): void
+getArrivalDiffMinutes(c:Context): int
+setArrivalDiffMinutesTemporarily(arrivalDiffMinutes:int, c:Context): void
+saveSettings(context:Context): void
SettingsActivity
-maxWalkBar: NumberPicker
-minArrBar: NumberPicker
#onCreate(savedInstanceState:Bundle): void
#onResume(): void
-displaySettings(): void
+saveSettings(v:View): void
MyNumberPicker
-processAttributeSet(attrs:AttributeSet): void

Fig 5: Diagram of Settings classes

Settings

`Settings` consists of a set of static methods that allow loading, saving, and modifying user settings. `Settings` currently stores two values: A maximum walking distance (in tenths of miles [1 : 10]; corresponding to [0.1 : 1] actual miles) and an arrival time offset (in minutes [0 : 100]).

SettingsActivity

`SettingsActivity` allows the user to directly modify the values stored in `Settings`.

MyNumberPicker

`MyNumberPicker` is an extension of the Android `NumberPicker` widget that loads its default values from XML. `MyNumberPicker` has fields for minimum (`min`), maximum (`max`), and increment step value (`incr`). It is used by the `SettingsActivity` UI to modify the settings values.

Reflections

Scott Blessing
Overall I really enjoyed working with the team this semester. We kept an open communication channel on Facebook that kept everyone informed and up-to-date. Our only major points of conflict were merge issues, but those always got resolved in time for each iteration meeting. Everyone put in a good amount of effort and I'm glad that we completed everything we set out to do in one semester. I learned a lot about working with the Android system, such as pulling calendar information, using AsyncTasks to run computations on the non-ui thread, and how Android handles file I/O.

Jimmy Guo
Throughout the project I was placed on several different teams which consisted of both front end and back end work. By working on both sides of the app, I've been able to gain confidence on creating my own android app. On the testing side, I learned how to set up and write UI testing as well as how annoying it is to do so. I now feel like I have enough experience to create an entire android app on my own, and may do so in the future. Throughout this project, I've been honored with a great team who consistently shows up on time and completes their work.

Ryan Gisleson
I learned a lot of new things about Android development from this project. I learned a large amount about creating persistent services for activities to update our app's status while running in the background. I also learned a lot about using Google API's like Location, Maps, and the Android Calendar API. Additionally I gained experience with new, more advanced testing styles like parameterized testing and user interface testing.

Varun Goverdhan
Initially, I only had experience with Android front-end development. I wanted to work on a project which would help me learn more about back-end development while also allowing me to contribute from my current front-end development knowledge. During this project, I was able to do exactly that. I learned a lot about all the different aspects of android development while also learning how to properly follow a specific development process. Our group was immensely diverse and each member had something unique to contribute. In addition to learning back-end development, I was also able to get a firm grasp on testing techniques and how beneficial they are. All in all, we are all able to learn new things while also being able to work together to create a good finished product.

Sirapob Nanthayapirom

I learned a lot about Android development, such as how activities are setup, what files determines the activities' layout and what files determines the activities' behaviour. I also learned several things about Google Maps api such as, adding either debug or release key to Google Maps api console in order to get Google Maps api to work, how to place markers, as well as how to draw lines on the map using polylines. I also learned how to write multi variable parameterized tests using JUnit Theories. Initially I was hesitant to work on this project because I was much more comfortable with web programming and therefore wanted to choose a web based project. However, with the experiences that I have gained I realized that Android development is not such a difficult task as I initially have thought.

Richard Shen

I gained valuable experience in Android development. In particular, I learned how to save data to Android's memory, create user dialogs, and how to use Google's various APIs (such as Maps and Calendar). Our development process was fairly effective; we rarely ran into any issues with it. However, we faced many challenges with the test-driven-development portion of XP due to the difficulty of writing UI tests. Ultimately we were able to overcome this challenge and write some UI tests with Espresso. I believe our team worked well together and overall, the project was enjoyable.

Derek Vidovic

This project gave me valuable experience jumping into new code and quickly getting up to speed. Over the semester, I got better at working with Git, dealing with merge issues, and minimizing conflicts with other people who were working on the same section of code. I enjoyed the experience of trying eXtreme Programming, which is very different from how I normally write code. I found writing tests more challenging than I was expecting; it seems like there are some types of code that really aren't easily testable by automated methods. However, I'm glad this project gave me a chance to try mock testing and parameterized testing, as this is the first exposure I've had to either.