CS 522—Spring 2016
Mobile Systems and Applications
Assignment Nine—Location Awareness

In this assignment, you will add location awareness to the Web-based chat service of Assignment Seven or Assignment Eight (your choice). Whenever a client communicates with the chat service, it always includes its current latitude and longitude coordinates as HTTP request headers. In turn, the service returns the current latitude and longitude of each peer with which the client is in communication (through the Web service). Furthermore, each message should be tagged with the latitude and longitude of the sender when they posted the message (note, not necessarily when it was eventually uploaded to the service), and this location information is included as metadata in the messages that are downloaded from the server.

Your main challenge will be to keep track of this location information in your chat client. Use the Fused Location API discussed in class. There is another API underneath this, an Android location provider API that manages the use of sensors such as GPS and WIFI, but you will find it easier to work with the Google location service:

`http://developer.android.com/training/location/index.html`

This API requires that you use the Google Play Services SDK, which mainly involves adding the Google Play Services library to your application. Technically your application should check that it is running the current version of the library, and update it otherwise, using the API of the Google Play Services library. However you can ignore this for the assignment, as long as you are running the current version of Google Play Services library:

`http://developer.android.com/google/play-services/setup.html`

To use the Google location service, you will need to use callbacks to manage the connection with this service. These callbacks are specified by the `GooglePlayServicesClient.ConnectionCallbacks` and `GooglePlayServicesClient.OnConnectionFailedListener` interfaces. In `onCreate()`, create a Google API client. In `onStart()`, connect this client to the Google Play service, and disconnect it in `onStop()`. To get a "B" grade on this assignment, you can simply use the `getLastLocation()` method in the Fused Location API to obtain the last estimate of the current location.

However this simplicity comes at the cost of potential lack of accuracy in the location estimate it returns, since it simply gives the location at the last time that the service checked. For more accurate location estimates, you should ask the location service to deliver regular location updates, at a frequency that you specify, with a measure of how accurate you want these estimates to be. This API is closer to the "raw" location API that Android provides, so you should follow this approach to get full credit for the assignment. You need to provide callbacks for the `LocationListener` class, and request

location updates using the `requestLocationUpdates()` method of the Fused Location API (when you are connected to the service). Be sure to remove location updates (using the `removeLocationUpdates()` method of the Fused Location API) when you are disconnected from the service. Maintain an internal record of the current location, as returned by these location updates, and use this information to include location information with your Web service calls.

To display location information in your app, show the latitude and longitude information for each message when you display that message. For the list of clients with whom you are in communication, you should also display the geocoded address for each peer. Call the `Geocoder.getFromLocation()` operation to translate from GPS coordinates to a list of addresses, and pick the first address in the resulting list. Since this geocoding is a long-lived operation, the best place to do this is in the background thread that manages Web service calls. Specifically, when the Web service returns a list of peers and their last-known locations, translate each of these locations to an address, and store this address (with the GPS coordinates) in the content provider where you are storing information about other peers. *Do not do this geocoding using* `AsyncTask`.

Remember that you will need to specify the appropriate permission for receiving location information in your application manifest. Use the fine grain location permission (`ACCESS_FINE_LOCATION`) for this assignment.

Web Service Stack

All requests in the Web service stack are defined as instances of subclasses of the following class:

```
public abstract class Request implements Parcelable {
   public long clientID;
   public UUID registrationID;  // sanity check
   public double latitude;
   public double longitude;
   // App-specific HTTP request headers.
   public Map<String,String> getRequestHeaders() {
      Map<String,String> headers = new HashMap<String,String>();
      headers.put("X-latitude", Double.toString(latitude));
      headers.put("X-longitude", Double.toString(longitude));
      return headers;
   }
   // Chat service URI with parameters e.g. query string parameters.
   public abstract Uri getRequestUri();
   // JSON body (if not null) for request data not passed in headers.
   public String getRequestEntity() throws IOException;
   // Define your own Response class, including HTTP response code.
   public Response getResponse(HttpURLConnection connection,
                               JsonReader rd /* Null for streaming */);
}
```

The `getRequestHeaders()` method should return any application-specific HTTP request headers. The two headers used for this assignment are the current latitude and longitude at the time that the request is invoked, which are added as fields to the request object. Your `RestMethod` class should install these app-specific headers in the HTTP request headers (using the `setRequestProperty()` method of `HttpURLConnection`). This header information will be expected by the Web service. It would probably be better if the registration API was included as a request header rather than a query parameter; however we will remain compatible with the original API by not changing this.

Running the Server App

To test the Web service with curl, use the `-H` parameter to include the latitude and longitude request headers. For example, the registration command will have the form:

```
curl -X POST -D headers \
      -H 'X-latitude: 40.7439905' \
      -H 'X-longitude: -74.0323626' \
      'http://localhost:8080/chat?username=joe&regid=...'
```

This sends a POST request to the specified (registration) URI, and places the response headers in a file called `headers`. The HTTP request headers include the latitude and longitude of the device at the time that the Web service is invoked. The query parameters include the desired user name and the UUID for the new registration. The contents of the header file will be of the form:

```
HTTP/1.1 201 Created
Content-Encoding: UTF-8
Content-Location: http://localhost:8080/chat/1
Content-Type: application/json
Date: Fri, 28 Feb 2014 14:57:25 GMT
Content-Length: 8
```

The output will be a JSON object `{"id":1}`. The identifier is used by the client to identify itself in subsequent requests. This identifier will be part of the URI that the client uses to specify the service. Provide the UUID as well in subsequent requests, as a query string parameter, as a sanity check. The value provided for the regid query string parameter should always be a well-formed UUID (e.g., `f47ac10b-58cc-4372-a567-0e02b2c3d479`).

The following command will synchronize messages with the server:

```
curl -X POST -H "Content-Type: application/json" \
      -H 'X-latitude: 40.7439905' \
      -H 'X-longitude: -74.0323626' \
      -d @messages.json -D headers \
      'http://localhost:8080/chat/1?regid=...&seqnum=0'
```

The query string parameters include the registration id (UUID) and the sequence number of the last message received by the client (The first message has a sequence number of 1). The file `messages.json` should contain messages to be uploaded, in JSON format. Each message should include, in addition to a timestamp, the latitude and longitude of the device at the time that the message was posted by the user (not when it was uploaded to the server). For example:

```
[
   {
      "chatroom" : "_default",
      "timestamp" : 12345678,
      "latitude":40.7143528,
      "longitude":-74.0059731,
      "text" : "hello"
   },
   {
      "chatroom" : "_default",
      "timestamp" : 12346678,
      "latitude":40.7143528,
      "longitude":-74.0059731,
      "text" : "is there anybody out there?"
   }
]
```

This will produce a JSON output of the form:

```
{"clients":[
      {"sender":"joe",
       "latitude":40.7439905,
       "longitude":-74.0323626}],
 "messages":[
      {"chatroom":"_default",
       "timestamp":12345678,
       "latitude":40.7143528,
       "longitude":-74.0059731,
       "seqnum":1,
       "sender":"joe",
       "text":"hello"},
      {"chatroom":"_default",
       "timestamp":12346678,
       "latitude":40.7439905,
       "longitude":-74.0323626,
       "seqnum":2,
       "sender":"joe",
       "text":"is there anybody out there?"}]}
```

The response includes a list of all clients registered with the service (with their last known locations), and a list of messages uploaded to the service since the last time this client synchronized (including messages just uploaded by the client itself). For

unregistering, use the DELETE HTTP command on the URI for the client service (specifying the last known position of the client before they unregistered):

```
curl -X DELETE -D headers \
      -H 'X-latitude: 40.7439905' \
      -H 'X-longitude: -74.0323626' \
      'http://localhost:8080/chat/1?regid=...'
```

Submitting Your Assignment

Once you have your code working, please follow these instructions for submitting your assignment:

1. Create a zip archive file, named after you, containing a directory with your name. E.g. if your name is Humphrey Bogart, then name the directory Humphrey_Bogart.
2. In that directory you should provide the Eclipse project for your Android app.
3. Also include in the directory a report of your submission. This report should be in PDF format. Do not provide a Word document.

In addition, record short flash, mpeg, avi or Quicktime videos of a demonstration of your assignment working. Make sure that your name appears at the beginning of the video. For example, put your name in the title of the app. *Do not provide private information such as your email or cwid in the video*.

Your solution should be uploaded via the Canvas classroom. Your solution should consist of a zip archive with one folder, identified by your name. Within that folder, you should have a single Eclipse project, for the app you have built. You should also provide a report in the root folder, called README.pdf, that contains a report on your solution, as well as videos demonstrating the working of your assignment.