CS 245 Preassignment 2
Geocode

1 Overview

Location-aware apps are today's most practical and personalized technology. Searching for a coffee shop? A smart search engine will favor local results. Lost your phone or friends? There's likely a Find My — app that can help. Key to making many of these apps meaningful to human users is the process of *geocoding*, which turns an address into a latitude/longitude pair, and its inverse operation *reverse geocoding*, which turns a latitude/longitude pair into a place name or address.

In this preassignment, you will leverage Google's web-based services to reverse geocode latitude/longitude pairs. GPS sensors in our mobile devices are a cheap source of personalized location data, but in order for a mobile device to use Google's services, it must have a reliable network connection. Many apps buffer themselves against communication failures by using a *cache*, a local mirror of remote data. For each latitude/longitude that is reverse geocoded, you will cache the results so that future lookups will not need an Internet connection.

In completing this preassignment, you will learn about hashing, web services, caching, and the use of external libraries.

2 Google's Geocoding API

A general rule of development is that you shouldn't automate what you haven't done manually. Therefore, before you write any code, look at Google's geocoding web service by visiting `http://maps.googleapis.com/maps/api/geocode/json?latlng=%f,%f&sensor=false`, replacing the two `%f`s with a decimal latitude and longitude. (You can get the latitude/longitude in degrees/minutes form from Google Maps by left-clicking on the map when sufficiently zoomed in and inspecting the popup. There are many online converters that will translate degrees/minutes/seconds into decimal form, and it's easy enough to write your own.)

Query 44.79711/-91.49965. Results are returned to you in a format called JavaScript Object Notation (JSON), which defines a portable string representation for arrays, hashes, numbers, booleans, strings, and null. Hashes are notated as {`"key1" : value1, "key2" : value2, ...`} and arrays as `[value1, value2, ...]`. For our purposes, we only care about the value with key `results`, which is an array. Of this array, we only care about the first element, which is a hash. Of this hash, we only care about the string with key `formatted_address`, which is "101 Roosevelt Avenue, Eau Claire, WI 54701, USA."

We could parse this string ourselves, but for this preassignment you will use Jackson (`https://github.com/FasterXML/jackson`), a library that can rip apart JSON. Using an instance of its `ObjectMapper` class, you can fetch a hash from a URL in just one line:

```
Map<String, Object> response = mapper.readValue(new URL(url), Map.class);
```

We can only say that the hash maps `String`s to vanilla `Object`s because each value may be a different type. One value may be an `ArrayList` while another is a `Boolean`. `Object` is their common superclass. To get the value out for a key, we can use `Map.get` and downcast the returned `Object` to the appropriate type.

The Jackson library is distributed as three JAR files. They have been downloaded and added to your `specs` directory. Right-click on each and add them to Eclipse's build path.

3 Requirements

Specifications do not tell you how to solve a problem—just what pieces may be used. The classes and methods described below will need to be thought about and pieced together using your own good mind. You will likely need to read this section many times.

Your solution is to meet the following specification:

1. Write all code in your fork of the class Bitbucket project, in package `pre2`.
2. Write a class `LatLon` with the following specification:
   (a) Encapsulates a latitude/longitude pair.
   (b) Has a constructor accepting a latitude and longitude as `double` parameters.
   (c) Has getters for both the latitude and longitude.
   (d) Has a `toString` method that returns the latitude and longitude shown to five digits and separated by a comma. For example, `new LatLon(47.5,33.12345123)` → "47.50000,33.12345". Use `String.format` to control the number of decimal places.
   (e) Has a `hashCode` method that returns as the hashcode of this coordinate pair the hashcode of its `String` representation. This method is used by a hashtable to find the index of a reverse geocoded location.
   (f) Has an `equals` method that returns a `boolean` indicating whether this coordinate pair is equal to some other `Object`. If the two references are identical, return true. If the other reference is `null` or not a `LatLon` return false. Otherwise, if the `String` representations of the two coordinate pairs are equal, return true. If they differ, return false. This method is needed because multiple locations might have the same hashcode. The hashtable finds which is correct by comparing the colliding keys for full equality.
3. Write an interface `Fetcher` with the following specification:
   (a) Has two generic parameters: `K` for the key type and `V` for the value type.
   (b) Imposes method `fetch` on its implementers, which fetches a value for a given key. It accepts a key as a parameter and returns a value.
4. Write a class `GeocodeFetcher` with the following specification:
   (a) Conforms to the `Fetcher` interface.
   (b) Has `LatLon` as its key type and `String` as its value type.
   (c) Implements `fetch` such that, given a `LatLon`, it reverse geocodes the location and returns the name of the location. Use Google's web-based Geocode API to retrieve a list of names, parse the JSON result using Jackson, and return the `formatted_address` of the first element in the `results` array.

5. Write a class `Cache`, which has general utility. You will write it so that it can be used for any caching purpose, not just geocoding. This is made possible by using generic parameters and the `Fetcher` interface. It has the following specification:

   (a) Maintains an association between keys and values. The first time a value for a key is looked up, it is fetched, but the result is stored. The second and all subsequent times the value for the key is looked up, the value is retrieved directly from the cache and the fetcher is not invoked.

   (b) Has two generic parameters: `K` for the key type and `V` for the value type.

   (c) Has a constructor accepting a `Fetcher` parameter. The `Fetcher` will be used when a value is looked up for the first time.

   (d) Has a method `get` that accepts a key parameter and returns a value. It first checks if the key-value pair is already cached. If so, the value is returned immediately. Otherwise, the fetcher is used to fetch the value, the pair is cached, and then the value is returned.

   (e) Has a method `has` that accepts a key parameter. It returns true if the key and its values are cached and false otherwise.

6. Write a class `Main` with the following specification:

   (a) Has a `main` method. What it does is not specified, but you are suggested to use it to test your code. Relying exclusively on the SpecChecker to test things will rob your brain of some neurons that are in your best interest to grow.

4 Submission

This homework is a preassignment and is graded with help from the SpecChecker. To submit your work for grading:

1. Put the SpecChecker for this homework in your Build Path.
2. Run the SpecChecker as a Java Application (not a JUnit Test) and fix problems until all tests pass.
3. Commit and push your work to your repository. If you are resubmitting an earlier assignment, email me. The time of your email will determine the submission week.

The SpecChecker cannot check everything. Your assignment is also expected to fully meet the requirements above and the following:

- You must not plagiarize. Write your own code. Talk about code with your classmates. Ask questions of your instructor or TA. Do not look at others' code. Do not ask questions specific to your homework anywhere online but Piazza. (If you find violators of this rule, please let me know.)
- Work must be submitted according to course policies on deadlines. To be eligible for later-week submission, you must have at minimum the skeletons for all specified classes and methods without compilation errors (no red!) in your repository by the homework deadline.