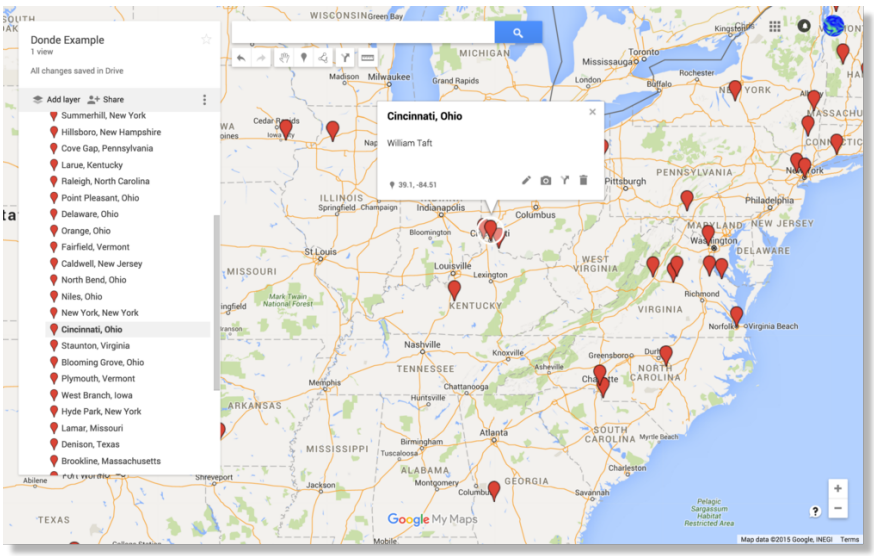
CS 1 Homework 7
Donde

1 Overview

Your task in this homework is to develop an understanding of objects, which unite data and its related code into one structure: a class. You will do so in the context of building a where-are-you-from map that can be viewed in Google Earth or Google Maps, marking people's hometowns with virtual pins. Consider this example of the United States presidents' and their hometowns:



As part of this homework you will invoke a geocoding web service that your instructor has provided for turning a place name into a latitude/longitude. The service receives your place name through a URL. The place name is then passed off to Google's geocoding web service to do the actual work, but because Google's web service blocks requests from computers that issue too many requests too often, your instructor's service includes a delay. You may feel this delay as you test. Be sure to perform your initial tests on small inputs. Please don't abuse the service with denial-of-service attacks.

2 Requirements

Complete the classes described below. Place all code in package `hw7`. All methods should be non-`static` unless specified otherwise.

2.1 Main

Write a class `Main`. It has a `main` method, which you are encouraged to use to test your code. Nothing in particular is required of it, but it must exist.

2.2 NoSuchPlaceException

Write a class `NoSuchPlaceException`, which you will use to throw an error when a place name cannot be geocoded into a latitute-longitude pair. It has the following:

1. Class `Exception` as its supertype. This class must be *throwable*, which is most easily done by making it a subtype of `Exception`.

2. A constructor, which accepts one parameter: a place name of type `String`. It passes off the place name to its superclass `Exception`, which uses it as the error message. Invoke the superclass constructor with a statement like the following:

   ```
   super(placeName);
   ```

Test this class on your own—without the SpecChecker—before moving on. Try throwing an instance of `NoSuchPlaceException`.

2.3 WebUtilities

Write a class `WebUtilities`, which contains a method for retrieving web page data. It has the following:

1. Static method `slurpURL`, which accepts one parameter: a web address of type `URL`. It opens the URL for reading, retrieves all its contents, closes the URL, and returns the contents as a `String`. You can open and close the URL in the following manner:

   ```
   URLConnection conn = url.openConnection();
   InputStream is = conn.getInputStream();
   ```

   The `InputStream` can be fed to a `Scanner` to do the reading. Read all the input with `Scanner`'s `nextLine` method, accumulating it up into one long `String`. Separate lines with an linebreak, but do so in a cross-platform way. `String.format("%n")` is recommended. Let an `IOException` be thrown if the opening or reading fails for any reason. You don't have enough information in this method to recover from the exception, so defer to the calling code.

Test this class on your own—without the SpecChecker—before moving on. One of the `URL` constructors accepts a `String` just like what you would type in your browser's location bar.

2.4 Set

Write a class `Set`, which contains methods managing a collection of unique `String`s. Java has a builtin `Set` class. Do not use it. This class has the following:

1. A default constructor (a constructor that accepts no parameters) that initializes any necessary state.

2. Method `has`, which accepts one parameter: a possible element, of type `String`. It returns a `boolean` indicating if the `Set` already contains the `String`.

3. Method `add`, which accepts one parameter: a possibly new element, of type `String`. It adds the `String` to the set only if it hasn't been added before.

4. Method `toString`, which returns a linebreak-separated `String` of the `String`s that have been added to the set—in the order they were first added. Linebreaks appear only *between* each pair of `String`s; no linebreak appears after the final, most recently added `String`. For example, suppose the following `String`s were added:

```
Hazel
Fiver
Bigwig
Hazel
Blackberry
Fiver
```

   Then we expect the following `String` to be returned:

```
Hazel¶
Fiver¶
Bigwig¶
Blackberry
```

   Use a cross-platform linebreak.

Test this class on your own—without the SpecChecker—before moving on.

2.5 Place

Write a class `Place`, which contains methods for managing a named location somewhere on Earth. This class has the following:

1. Getters for a place's name, latitude, and longitude. Use the names `getName`, `getLatitude`, and `getLongitude`.

2. A constructor, which accepts a `String` place name, a `double` latitude, and a `double` longitude. This constructs a `Place` with the given information and without any people currently associated with it.

3. Method `toGeocodeURL`, which returns a `URL` for a web service that will translate the place name to a latitude/longitude pair. For example, if the place is named `Fooville, AR`, then the following URL is returned:

```
http://www.twodee.org/teaching/cs1/2017c/homework/hw7/geocode.php?place=Fooville,%20AR
```

   Because the name might contain spaces or other characters that are not legal in a URL, the name must be encoded. The related `URI` class can help. Construct a `URI` in the following manner:

   (a) scheme → `http`

(b) authority → `www.twodee.org`

(c) path → `/teaching/cs1/2017c/homework/hw7/geocode.php`

(d) query → `place=PLACENAME`, replacing `PLACENAME` with your actual place name

(e) fragment → `null`

`URI`s can be turned into `URL`s. See the documentation.

4. A constructor, which accepts one parameter: a place name of type `String`. The place initially is associated with no people. Retrieve the place's latitude and longitude by slurping the content from the place's geocode URL. If the place is valid, a single line of text containing the latitude and longitude is retrieved. For example, `new Place("Eau Claire, WI")` generates the following result from the web service:

```
44.811349 -91.498494
```

   Parse this text and store the latitude and longitude. If the web service request fails for any reason or if the request yields a result that doesn't contain two `double`s, then throw a `NoSuchPlaceException` exception using the place's name.

   Be sparing on your users' data plans. The web service should only be invoked once for a `Place`. Future calls to `getLatitude` and `getLongitude`, for example, must simply reference the persisted state of the `Place` and not generate more network traffic.

5. Method `addPerson`, which accepts one parameter: a person's name, of type `String`. It associates that person with this place. If the person has already been associated with the place, the add request must have no effect.

6. Method `toKML`, which returns as a `String` the Keyhole Markup Language (KML) representation of this place, which looks like the following:

```
<Placemark>¶
<name>NAME</name>¶
<description>PEOPLE</description>¶
<Point><coordinates>LONGITUDE,LATITUDE</coordinates></Point>¶
</Placemark>
```

   Replace `NAME` with the place name, `PEOPLE` with the linebreak-separated list of people associate with the place, and `LONGITUDE` and `LATITUDE` with the coordinates rounded to 2 decimal places. `String.format` can help with this. Cross-platform linebreaks appear after each line but the last. No line is indented.

Test this class on your own—without the SpecChecker—before moving on.

2.6 PlacesCache

Write a class `PlacesCache`, which contains methods for managing a collection of named locations somewhere on Earth. This class has the following:

1. A default constructor that initializes the cache to contain no places.

2. Method `isCached`, which accepts one parameter: a place name of type `String`. It returns `true` if the place already exists in the cache and `false` otherwise.

3. Method `getPlace`, which accepts one parameter: a place name of type `String`. It returns a `Place`. If the place with this name is already in the cache, it returns the existing place. Otherwise, it creates a new `Place` for the given name, stores it for later fast retrieval, and returns the new place. Defer any `NoSuchPlaceException`s to the caller.

4. Method `size`, which returns the number of places in the cache as an `int`.

5. Method `get`, which accepts one parameter: an index `i` of type `int`. It returns the `i`th place added to the cache via `getPlace`. If the index is not legal, throw an `IndexOutOfBoundsException`.

Test this class on your own—without the SpecChecker—before moving on.

2.7 DondeUtilities

Write a class `DondeUtilities`, which contains a methods managing a named location somewhere on Earth. This class has the following public interface:

1. Static method `readCSV`, which accepts one parameter: a file of type `File`. It returns a `PlacesCache` for the contents of the file. The file contains a list of people and their associated places in a comma-separated format. Consider this example file:

   ```
   George Washington|Westmoreland County, Virginia
   John Adams|Quincy, Massachusetts
   Thomas Jefferson|Shadwell, Virginia
   James Madison|Port Conway, Virginia
   James Monroe|Westmoreland County, Virginia
   George Washington|Westmoreland County, Virginia
   ```

   Each line can be broken into pieces with `String.split(Pattern.quote("|"), -1)`.

   The `PlacesCache` returned for this example contains the four unique places listed. The entry for Westmoreland County contains both George Washington and James Monroe, while the remaining three places are associated with only one individual. Repeated lines have no observable impact on the cache.

   If a `NoSuchPlaceException` is generated, catch it and print out an appropriate warning message to `System.err`. Defer any `FileNotFoundException` to the caller.

2. Static method `writeKML`, which accepts two parameters in this order:

   (a) a cache of type `PlacesCache`

   (b) a file write to, of type `File`

   It writes the `PlacesCache` out to the given file in the KML format. Follow this template when writing out the file:

```
<?xml version="1.0" encoding="UTF-8"?>¶
<kml xmlns="http://www.opengis.net/kml/2.2">¶
<Document>¶
<name>Donde</name>¶
<Placemark>¶
...
</Placemark>¶
<Placemark>¶
...
</Placemark>¶
</Document>¶
</kml>
```

The first `Placemark` element corresponds to the first place that was added to the cache via `getPlace`, the second element to the second place, and so on.

3 Extra

You can visualize the resulting KML file in a couple of ways: in the desktop application Google Earth or in the browser via Google Maps. To import your KML file in Google Maps, click menu / Your Places / Maps / Create Map / Import.

Anyone who shares on Piazza a CSV file and map image for some meaningful dataset (not just random names and cities) posted under folder `hw7_share` will receive one extra credit participation point.

4 Submission

To submit your work for grading:

1. Put the SpecChecker for this homework in your Build Path. Run the SpecChecker as a Java Application and fix problems until all tests pass.
2. Commit and push your work to your repository. Verify that your solution is on Bitbucket.

A passing SpecChecker does not guarantee you credit. Your grade is conditioned on a few things:

- You must meet the requirements described above. The SpecChecker checks some of them, but not all.
- 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. Your instructor employs a vast repertoire of tools to sniff out academic dishonesty, including: drones, CS 1 moles, and a piece of software called MOSS that rigorously compares your code to every other submission. You don't want to live in a world serviced by those who squeaked by through questionable means. For your future self, career, and family, do your own work.
- Your code must be submitted correctly and on time. Most excuses devolve into, "I started too late." The fix for this problem is not an extension.