CMSC 206: Data Structures
Assignment #3: Zip Codes with Locations
due on Gradescope by the beginning of class on Tuesday, February 27, 2018

Purpose: Write a complete Java program to repeatedly input a zip code from the user. In response, your program will output the name of the town and the state to which it belongs. If we know where the zip code is (its latitude and longitude), it should output that as well. If we know the population, it will output that, too. In case the zip code doesn't exist, it tells the user about it (see example interaction below).

To complete this assignment, you will adapt your code from Assignment 2.

The big new part is that there is now a second data file that contains a list of all zip codes along with their latitude and longitude (but not accurate counts of populations). The central task of this assignment is to weave the information from these files together, so that your `Place` objects can store both a population and a location for a zip code.

The second file is on the syllabus page, called *ziplocs.csv*. It has a header line at the top with column names, but it does not list the number of entries in the file. Thereafter, the lines look like this:

```
"07677","STANDARD","WOODCLIFF LAKE","NJ","PRIMARY",41.02,-74.05,"NA-
US-NJ-WOODCLIFF LAKE","false",2945,5471,325436960
```

(In the file, that would be all one line, but it doesn't fit here.) The comma-separated fields are:

1. Zip code, in quotes
2. Zip code type, in quotes (you will ignore this)
3. Town name, in quotes (you will ignore this)
4. State abbreviation, in quotes (you will ignore this)
5. Location type, in quotes (you will ignore this)
6. Latitude, expressed as a decimal number; positive numbers are North, negative South
7. Longitude, expressed as a decimal number; positive numbers are East, negative West
8. Full location name, in quotes (you will ignore this)
9. Whether or not the zip code has been decommissioned, in quotes (you will ignore this)
10. The number of tax returns filed in a given year from this zip code (you will ignore this)
11. The estimated population (you will ignore this)
12. The total of all the salaries of people who filed taxes in this zip code (you will ignore this)

Some lines are missing some information, but every line has the correct number of commas (11).

By collating the data between *uszipcodes.csv* and *ziplocs.csv*, we can categorize all zip codes into one of three categories: zip codes with a population and location, zip codes with a location only, and zip codes without either. (Interestingly, the dataset does not contain any zip codes with a population but no location.) We'll call the first a `PopulatedPlace`, the second a

`LocatedPlace`, and the third just a `Place`. These types naturally form a subtype hierarchy, where `PopulatedPlace` is a subtype of `LocatedPlace` (every `PopulatedPlace` is also a `LocatedPlace`) and `LocatedPlace` is a subtype of `Place` (every `LocatedPlace` is also a `Place`).

Here are the tasks you must complete for this assignment:

1. Copy the `Place`, `ZipParseException`, and `LookupZip` classes from Assignment #2 into a new project.

2. Add a new method to your `Place` class, as follows:

```
/** Returns a user-friendly description of this Place.
 *  For example, the Place for Bryn Mawr would return
 *  the string "Bryn Mawr, PA".
 *  @return A user-friendly description of this Place
 */
public String placeDescription()
```

   Your `Place` class must have a `toString()` method that works as specified in the previous assignment.

3. Write a new class `LocatedPlace` that is a subtype of `Place`. It must have a constructor as follows:

```
/** Creates a LocatedPlace with the given info
 *  @param zip The 5-digit zip code
 *  @param town The town name
 *  @param state The state abbreviation
 *  @param lat The latitude
 *  @param lon The longitude
 */
public LocatedPlace(String zip, String town, String state,
                    double lat, double lon)
```

   The `LocatedPlace` class must also override the `placeDescription` method from the `Place` class to include the latitude and longitude information in the description string. (The exact format is not prescribed.)

   The `LocatedPlace` class must also have a `toString` method that prints out the latitude and longitude, separated by colons, after the output produced by the `Place` class. So, for Bryn Mawr, it would be `19010:Bryn Mawr:PA:40.02:-75.31`

4. Write a new class `PopulatedPlace` that is a subtype of `LocatedPlace`. It must have a constructor as follows:

```
/** Creates a PopulatedPlace with the given info
 *  @param zip The 5-digit zip code
 *  @param town The town name
 *  @param state The state abbreviation
 *  @param lat The latitude
 *  @param lon The longitude
 *  @param pop The population
 */
public PopulatedPlace(String zip, String town, String state,
                      double lat, double lon, int pop)
```

The `PopulatedPlace` class must also override the `placeDescription` method to include the place's population in the description string.

`PopulatedPlace`'s `toString` method must also append the population. For Bryn Mawr, this would yield `19010:Bryn Mawr:PA:40.02:-75.31:21103`

5. Modify the `readZipCodes` method from `LookupZip`, as follows:

```
/** Reads a zipcodes file and a ziplocs file, creating
 *  an ArrayList of Places. If a place's population is
 *  known, it will be represented by a PopulatedPlace
 *  object; otherwise, if a place's location is known, it
 *  will be represented by a LocatedPlace object; otherwise
 *  it will be represented by a Place object.
 *  @param zips The name of the zipcodes file
 *  @param locs The name of the ziplocs file
 *  @return The array of Places representing all the
 *  data in the files.
 */
public static ArrayList<Place>
  readZipCodes(String zips, String locs)
    throws FileNotFoundException, ZipParseException
```

There are several different approaches to implementing this new `readZipCodes` method. The one restriction is that it should read each file only once: that is, you should create a `new Scanner` for each file only once, not in a loop. You should also not reset these `Scanner`s. Instead, you will have to read in one file first, accumulate the partial data in that file, and then read the other file, combining the entries appropriately. Note that the zip codes in the files are *not* in the same order.

Note that this new version returns an `ArrayList`, not an array. This may be more convenient than an array, given that the *ziplocs.csv* file does not tell you how many entries it has.

You might find that you can no longer use your old `parseLine` or `lookupZip` methods. That's fine.

This assignment will *not* test error handling.

6. Update your `main` method to work with your new `readZipCodes` method. Use the `placeDescription` method in rendering the text to print for the user. Due to Java's polymorphism, you should automatically get customized output based on whether a place has a known location or population.

7. Write up your reflections, answering the following questions in a *reflections.txt* file:

    a. How long did this assignment take you?
    b. What challenged you the most in this assignment?
    c. Whom did you work with on this assignment?
    d. What resources did you consult while doing this assignment?
    e. Do you have any feedback to offer about this assignment?
    f. Any other comments or questions?

8. Submit your work on Gradescope.