CMSC 206: Data Structures
Assignment #4: Aggregating Populations
due on Gradescope by the beginning of class on Thursday, March 22, 2018

This assignment builds on Assignment #2 (*not* Assignment #3), adding a feature where we can look up town information (instead of zip code information), adding up the population in perhaps several zip codes used within one town.

Like Assignment #3, the test cases will *not* test erroneous input.

The tasks have an estimated time burden and how much code it should take, as an indication of how hard a task should be. If you're finding it much harder than this, it's time to speak up.

1. Copy your *LookupZip.java*, *ZipParseException.java*, *LookupZipTest.java*, and *Place.java* files from Assignment #2 into a new project folder. (In order to get JUnit to compile, it may work well to create a new JUnit Test Case in the new project folder and then just delete it.) Also copy the *uszipcodes.csv* file into the new folder. Make sure everything is working before making any changes. (5 mins, no code)

2. Modify your `Place` class to store population information. The constructor should now take *four* parameters (the last one is an `int` parameter denoting the `Place`'s population), and the `toString` method should now return the population as well as the other info. (Put the population at the end, with fields separated by colons as before.) You will have to modify your `parseLine` method to store the population in a `Place`. If a line in the input file doesn't have a population, assume a population of 0. (This is in sharp contrast to the behavior in Assignment #3.) (15 mins, ~10 lines of new code)

3. Create a new `Town` class that represents a municipality, possibly comprised of many zip codes. Each `Town` needs to store the following:

    a. town name
    b. state
    c. total population of the town
    d. one zip code in that town
    e. the total number of zip codes assigned to that town

    Accordingly, it should have a 5-argument constructor that takes all this information. (15 mins, ~25 lines of simple code)

4. Write a `public void printInformation` method in your `Town` class that prints the information about a town. Note that this *prints*, not *returns*. (15 mins, ~10 lines) For example, my `Town` for Philadelphia prints

```
Town: Philadelphia, PA, 19019 (plus 86 other zip codes)
Population: 1526206
```

5. Write a new method in your `LookupZip` class as follows (2-3 hours, ~30 lines):

```
/** Gathers information about a town. The Town object
 *  returned from this method includes the entire
 *  population of the given town, even if the town
 *  comprises many zip codes. The zip code stored in
 *  the returned town is the first listed zip code in
 *  the given array.
 *  @param places All the known places, as read in
 *                from the uszipcodes.csv file
 *  @param townState The town name and state; for
 *                   example, "Bryn Mawr, PA"
 *  @return A Town object with all the information
 *          about the requested town, or null if
 *          the town cannot be found.
 */
public static Town lookupTown(Place[] places,
                              String townState)
```

6. Update your `main` method to lookup towns instead of zip codes. (15 mins, little to no net change in lines of code) Here is an example session:

```
What town should I look up (enter "town, state")? Philadelphia, PA
Town: Philadelphia, PA, 19019 (plus 86 other zip codes)
Population: 1526206
Do you want me to search again? yes
What town should I look up (enter "town, state")? Bryn Mawr, PA
Town: Bryn Mawr, PA, 19010
Population: 21103
Do you want me to search again? yes
What town should I look up (enter "town, state")? Apo, AP
Town: Apo, AP, 96201 (plus 66 other zip codes)
Population: 0
Do you want me to search again? yes
What town should I look up (enter "town, state")? Haverford, PA
Town: Haverford, PA, 19041
Population: 6248
Do you want me to search again? yes
What town should I look up (enter "town, state")? Truth or Consequences, NM
I can't find zip code Truth or Consequences, NM
Do you want me to search again? yes
What town should I look up (enter "town, state")? Truth Or Consequences, NM
Town: Truth Or Consequences, NM, 87901
Population: 7139
Do you want me to search again? yes
What town should I look up (enter "town, state")? Clearfield, UT
Town: Clearfield, UT, 84015 (plus 2 other zip codes)
Population: 62494
Do you want me to search again?
```

7. **Get everything above working before proceeding past this step!**

8. Download the *KWArrayList.java* file from the course syllabus page, putting it in the same Eclipse project. (2 mins)

9. Write a new class `KWALIterator` that is an iterator for the `KWArrayList` collection. Your class must have a type parameter `E`, just like `KWArrayList`. It must also implement the `Iterator<E>` interface, overring the `hasNext`, `next`, and `remove` methods. (Do *not* worry about the `forEachRemaining` method.) These methods must behave as specified by the online documentation for `Iterator`.

   You will need to figure out what data fields are necessary in your `KWALIterator` class and what constructor to write. The code here is short and straightforward; it's figuring out *what* to do that's hard. Work together and post on Piazza to figure this out! (1 hour to understand task, 1 hour to code; ~50 lines)

10. Modify `KWArrayList` to implement the `Iterable<E>` interface. (Look up that interface in the documentation to learn more.) You will have to add a new method to `KWArrayList`. This method should have about one line in it – and it will likely involve the keyword `this`. (15 minutes to understand task, 5 minutes to complete; ~5 lines)

11. Modify your `LookupZip` methods to work with a `KWArrayList` instead of an array. In your `lookupZip` and `lookupTown` methods, use the "foreach" loop (with the `for(Place p : places)` syntax) to iterate through the `KWArrayList`. (10 minutes)

12. After getting everything working, run your program on the input in the example session, above. (Your output does *not* have to match mine character-for-character – but the information presented must match.) Save a screenshot (search online for instructions if you don't know how to take a screenshot – tutorials abound) of this test in your project folder. (5 minutes)

13. Write up your reflections, answering the following questions in a *reflections.txt* file (5-10 minutes):

    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?

14. Submit your work on Gradescope.