**CSE 274**

**Java Maps, Sets, and other Data Structures**

A datafile - ZipCodes.txt - contains over 85,000 lines of text - one line for each geographic location in the US territories. The lines are tab-separated and look like:

26569 45056 STANDARD **OXFORD OH** PRIMARY **39.5 -84.74** 0.07 -0.76 0.63 NA US "Oxford, OH" NA-US-OH-OXFORD FALSE 7480 12167 254490797

26570 45056 STANDARD **MIAMI UNIV OH** ACCEPTABLE **39.5 -84.74** 0.07 -0.76 0.63 NA US "Miami Univ, OH" NA-US-OH-MIAMI UNIV FALSE 7480 12167 254490797

When writing your code, assume that the file format never changes (i.e., 19 columns, city name is always in column 4th column, etc.) You may not assume that the file contains any particular number of states (i.e., Texas my secede from the Union or new states may be added at a later date).

You are to write the following methods that utilize standard Java data structures. When writing your solutions, strive for clarity as your primary objective. Efficiency should also be an objective but should not significantly override the goal of clarity.

public class Location {

public int zipCode;

public String cityName;

public String state;

public double latitude, longitude;

private static double parseDouble(String str) {

double value = 0;

if (!str.isEmpty()) {

value = Double.parseDouble(str);

}

return value;

}

public Location(String [] toks) {

zipCode = (int)parseDouble(toks[1]);

cityName = toks[3];

state = toks[4];

latitude = parseDouble(toks[6]);

longitude = parseDouble(toks[7]);

}

}

public class Places {

private ArrayList<Location> places;

public Places(String fname) throws FileNotFoundException {

places = new ArrayList<>();

Scanner input = null;

input = new Scanner(new File(fname));

input.nextLine(); // skip first line

while (input.hasNextLine()) {

String [] toks = input.nextLine().split("\t", -1);

Location loc = new Location(toks);

places.add(loc);

}

input.close();

}

/\*\*

\* Returns all the city names that correspond to a particular zipcode. The  
 \* empty set is returned if the zipcode is not valid.

\* @param zipCode target zip code

\* @return set with all city names with the target zip code.

\*/

public Set<String> getCityNameFromZipCode(int zipCode) {

/\*\*

\* Returns all the zipcodes that are contained in a particular city-state. The  
 \* empty set is returned if the city-state pair is illegal.

\* @param cityName target city name

\* @param state target state

\* @return set with all relevant zipcodes.

\*/

public Set<Integer> getZipCodes(String cityName, String state) {

/\*\*

\* Returns a map that is keyed to state name. The values in the map is a set of

\* zip codes that reside in that particular state. The map looks like:

\* "AL" --> { 36863, 35755, ... }

\* "AK" --> { 44256, 44257, ...}

\* ...

\* @return mapping from states to set of zipcodes.

\*/

public Map<String, Set<Integer>> getZipCodesInStates() {

/\*\*

\* Returns all the states that contain a particular city name. The empty

\* set is returned if the city name is not any state.

\* @param cityName target city name

\* @return set of states that contain the target city.

\*/

public Set<String> getStatesThatContainThisCity(String cityName) {

/\*\*

\* Returns the states that contain any of the target cities. The empty

\* set is returned if none of the cities are in any state. This is similar

\* to above but accepts multiple cities instead of single city.

\* @param cityNames target cities

\* @return set of states that contain any of the target cities.

\*/

public Set<String> getStatesThatContainAnyOfTheseCities(Set<String> cityNames){

/\*\*

\* Returns all zipcodes that are within a specified distance from a

\* particular zipcode.

\* @param zipCode target zipcode

\* @param distance maximum distance from target zipcode

\* @return all zipcodes that are within "distance" from the target zipcode

\*/

public Set<Integer> getZipCodesCloseTo(int zipCode, double distance) {

/\*\*

\* Returns a map that is keyed to state name. The values in the map are the set

\* of city names that reside in that particular state. The map looks like:

\* "AL" --> { "MONTGOMERY", "MOBILE", ... }

\* "AK" --> { "ANCHORAGE", "BARROW", ...}

\* ...

\* @return mapping from states to set of city names.

\*/

public Map<String, Set<String>> getCityNames() {

/\*\*

\* Returns all city names that reside within a particular state.

\* @param state target state

\* @return set of city names

\*/

public Set<String> allCityNames(String state) {

/\*\*

\* Returns all city names that reside within any of the

\* given states.

\* @param state target states

\* @return set of city names

\*/

public Set<String> allCityNames(Set<String> states) {

/\*\*

\* Returns all city names in the entire database

\* @return set of city names

\*/

public Set<String> allCityNames() {

/\*\*

\* Returns all state names in the entire database

\* @return set of state names

\*/

public Set<String> allStateNames() {

/\*\*

\* Returns the city names that appear in both of the given

\* states

\* @param state1 first target state

\* @param state2 second target state

\* @return set of city names

\*/

public Set<String> getCommonCityNames(String state1, String state2) {

/\*\*

\* Ranked list of states, where the ranking is ascending order of number

\* of zipcodes.

\* @return

\*/

public ArrayList<String> mostZipCodes() {

/\*\*

\* The city name(s) that appears in the most states. Note, "NEW YORK"

\* is credited with appearing in NY only once. That is, the large number

\* of "NEW YORK" zipcodes does not make it a more common name.

\* @return set of city names

\*/

public Set<String> cityNameInMostStates() {

Scoring (Each method will be equally weighted; 100 points total)

*Complete the methods shown in bold. These are the methods that were not completed in the corresponding lab assignment.*

1. **public Set<String> getCityNameFromZipCode(int zipCode) {**
2. public Set<Integer> getZipCodes(String cityName, String state) {
3. public Map<String, Set<Integer>> getZipCodesInStates() {
4. public Set<String> getStatesThatContainThisCity(String cityName) {
5. **public Set<String> getStatesThatContainAnyOfTheseCities(Set<String> cityNames) {**
6. **public Set<Integer> getZipCodesCloseTo(int zipCode, double distance) {**
7. **public Map<String, Set<String>> getCityNames() {**
8. **public Set<String> allCityNames(String state) {**
9. **public Set<String> allCityNames(Set<String> states) {**
10. **public Set<String> allCityNames() {**
11. public Set<String> allStateNames() {
12. **public Set<String> getCommonCityNames(String state1, String state2) {**
13. **public ArrayList<String> mostZipCodes() {**
14. **public Set<String> cityNameInMostStates() {**