

# ArrayList and Inheritance - Zipcodes with populations and locations

CS 151 - Introduction to Data Structures

Assignment 2 - due Friday 2/3

In this project, we will practice ArrayList and object-oriented design with inheritance. You will build on top of your Assignment 1. There is now a second input file `ziplocs.csv` which gives information about zipcodes and their associated latitudes and longitudes (as decimal numbers), among other things (but not population). Your task this week is to weave these two files together and organize the data into classes stored in an ArrayList that allows us to perform similar lookups as those in Assignment 1.

One of the important goals of this class is to train you to become successful independent programmers who can translate a problem on paper to fully functional code that solves the problem. Thus, each assignment is designed to give you less detailed instructions. You will notice that for example, unlike Assignment 1, this assignment no longer contains exact method signatures or details of any instance variables. Instead, you are asked to think about what instance variables are necessary and reasonable in order to accomplish the tasks given here. In addition, not having exact method signatures doesn't mean anything goes. You are expected to design functional and efficient methods much like those you had before. In other words, Assignment 1 gave you an exact design you just had to implement. This assignment asks you to flesh out a design much in the same style - basic steps are still listed for you - see Sections 2 and 3. I am happy to discuss design questions during lecture, lab or office hours of course.

## 1 Input File Format

The file `ziplocs.csv` contains a header line at the top with column names, but it does not list the number of entries in the file. Thereafter, the lines contain 12 comma-separated fields that look like this:

```
"07677", "STANDARD", "WOODCLIFF LAKE", "NJ", "PRIMARY", 41.02, -74.05, "NA-
```

```
US-NJ-WOODCLIFF LAKE", "false", 2945, 5471, 325436960
```

We will only use three of these fields, the zipcode (#1), the latitude (#6) and the longitude (#7). In the sample line, that is 07677 (in quotes), 41.02 and -74.05.

As in the last assignment, some lines are missing some information, but all lines have the correct number of commas (11).

In your last assignment, you only read the zipcode, town and state fields from `uszipcodes.csv`, but ignored the population information. You will now read the total population field in `uszipcodes.csv` as well, if not missing.

By collating the data between `uszipcodes.csv` and `ziplocs.csv`, we can categorize all zipcodes into one of three categories: zipcodes with a population and location, zipcodes with a location only, and zip codes without either. (Interestingly, the dataset does not contain any zipcodes 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 an inheritance hierarchy, where `PopulatedPlace` is a subclass of `LocatedPlace` (every `PopulatedPlace` is also a `LocatedPlace`) and `LocatedPlace` is a subclass of `Place` (every `LocatedPlace` is also a `Place`).

## 2 Specific Tasks

1. All classes should be `public` in this assignment, and thus each in their own file.
2. Write a new class `LocatedPlace` that is a subclass of `Place`. Include appropriate instance variables, constructor and getters.
3. The `LocatedPlace` class must have an overridden `toString` method that includes the location information in the string returned. For example, a `LocatedPlace` representing Bryn Mawr would return the string `"Bryn Mawr, PA 40.02 -75.31"`.
4. Write a new class `PopulatedPlace` that is a subclass of `LocatedPlace`. Include appropriate instance variables, constructor and getters.
5. The `PopulatedPlace` class must also override the `toString` method to include the place's population in the string. A `PopulatedPlace` representing Bryn Mawr should return the string `"Bryn Mawr, PA 40.02 -75.31 21103"`.
6. Modify the `readZipCodes` method from `LookupZip` to read both data files, constructing an `ArrayList` of `Places` and returns it. 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.

One restriction is that `readZipCodes` should read each file only once: that is, you should create a new `Scanner` for each file only once, not repeatedly. You should also not reset these `Scanners`. You should read in one file first, create objects to accumulate the partial data in that file, and then read the other file, combining the entries appropriately. Note that the zipcodes in the files are not in the same order.

Note that this new version returns an `ArrayList`, not an array. You need to adapt your old `parseLine` or `lookupZip` methods or both.

7. Update the `main` method to work with your new methods. Recall that the appropriate `toString()` will be used when you print your three different objects of the `Place` type hierarchy.

Here's a sample session:

```
zipcode: 19010
Bryn Mawr, PA 40.02 -75.31 21103
```

```
zipcode: 99400
No such zipcode
```

```
zipcode: 91729
Rancho Cucamonga, CA 34.09 -117.56
```

```
zipcode: 97252
Portland, OR
```

```
zipcode: 00000
Good Bye!
```

8. Testing steps are identical to Assignment 1. Obtain the new test files from the appropriate subdirectory in my handouts.

### 3 Design Notes

Here are some design questions I'd like to call your attention to. Make sure you have good answers to these questions BEFORE you start programming!

1. How do you decide whether a zipcode should be a `PopulatedPlace`, a `LocatedPlace`, or just a `Place`?

2. The total population field in `uszipcodes.csv` is not filled out for some lines. How do you detect that it is missing? Hint: research `split` and learn about the overloaded version with a second parameter `limit`.
3. When the population field is not filled for a zipcode in `uszipcodes.csv`, what object do you construct? And if it is, what object do you construct?
4. Read `uszipcodes.csv` first. After you are done reading `uszipcodes.csv`, what data structure do you have and how have you populated it?
5. Then read `ziplocs.csv`. As you read each line, how do you collate the information you have with the data you already collected and stored from `uszipcodes.csv`?
6. The zipcode field in `ziplocs.csv` has double quotes around it. You should remove the quotes (after reading in - don't modify the input file!) or you will not be able to match them with the zipcodes found in `uszipcodes.csv`. I recommend using `replace` from the Java `String` library (use the Java API reference to find out how).

## 4 Electronic Submissions

At this point, you should have one `Main.java`, but also a separate `.java` for every class you created (since they are all `public`).

1. **README:** The usual plain text file `README`

**Your name:**

**How to compile:** Leave empty if it's just `javac Main.java`

**How to run it:** Leave empty if it's just `java Main`

**Known Bugs and Limitations:** List any known bugs, deficiencies, or limitations with respect to the project specifications. Documented bugs will receive less deduction versus uncaught ones.

2. **Source files:** `Main.java` `Place.java` `LocatedPlace.java`  
`PopulatedPlace.java` `LookupZip.java`

3. **Data files used:** `uszipcodes.csv` `ziplocs.csv`

**DO NOT INCLUDE:** Please delete all executable bytecode (`.class`) files prior to submission.

To submit, store everything (README, source files and data files) in a directory called A2. Then follow the directions here:

[https://cs.brynmawr.edu/systems/submit\\_assignments.html](https://cs.brynmawr.edu/systems/submit_assignments.html)