CS206 Assignment 5 Style Grading Rubrics

General

5 points are allocated to fairly mechanical rules on naming/comments/indentation - these should be easy to check off. Another 20 points are allocated to more creative practices, as explained below. Consult the formatting guide for details to check for under each category

Print student programs from Emacs, via “postscript print buffer” menu option.

Code formatting (**5 points total**)

1. Naming Conventions: **1 points**
    a. if any of the rules are violated
2. Whitespace: **1 point**
    a. inconsistent spacing (excessively) - - if just one place, point it out but don’t take off
3. Comments: **2 points**
    a. File header missing or malformatted
    b. Uncommented instance variables
    c. Uncommented methods (getters and setters can have no comments, when appropriately named)
    d. Method comments that do not conform to javadoc style
    e. Uncommented complex blocks of code
    f. Unhelpful comments
4. Indentation: **1 point**
    a. inconsistent indentation (excessively) - if just one single line, point it out but don’t take off

Design principles (**20 points total**)
The exact point allocations will change from assignment to assignment. In general, because it is impossible for me to imagine all the ways thing can go wrong, grade somewhat holistically instead of sticking to the rubric strictly.

Below, 1-6 are the same as assignment 2 and are allocated a total of **8 points**. Most students should really have these 8 right already. Minor violations get 1 point off, major 2-3. 7 is on the binary search and new to this assignment and gets another **12 points.**

206 Assignment 5 (binary search)

1. private Instance variables and getters
    a. Any non-private instance variables, including missing modifier
2. `public static final` constants instead of integer/double literals - any literal that has reason to be changed later should be a constant
    a. Cases noted
        i. Using “`00000`” directly in code

    ii. Using `[0]`, `[1]`, `[2]` … directly in code after calling split
3. Constructor must initialize all instance variables
    a. Check `Place, LocatedPlace` and `PopulatedPlace` constructors
    b. `LocatedPlace` and `PopulatedPlace` constructors must call `super` appropriately
4. Reasonable designs for `Place`, `LocatedPlace`, `PopulatedPlace`, `LoopupZip` and no additional classes (besides `Main` of course)
    a. `Place` has zipcode, town and state instance variables (as `String`) and no additional. Has `toString` overridden
    b. `LocatedPlace` has latitude and longitude instance variables as `double`, not `String` and no additional. `toString` appropriately overridden. Preferrably by calling `super.toString()` first (don't take off though, just point it out)
    c. `PopulatedPlace` has population instance variable as `int`, not `String`, and no additional. `toString` appropriately overridden. Preferrably by calling `super.toString()` first (don't take off though, just point it out)
    d. `LookupZip` doesn't have instance variables (constants are not instance variables and they should have them!) and holds the methods `parseLine` (if exists), `readZipCodes` and `lookupZip`
5. Method designs and data weaving
    a. `parseLine`, `readZipCodes` and `lookupZip` should have reasonable designs - any abuse/overcall/redundant use gets -1:
        i. It is acceptable to not have a `parseLine` and merge the functionality into `readZipCodes` directly. Another approach is to write two different versions of `parseLine`, one for each file. `parseLine` (if there is one) should NOT have a loop
        ii. `readZipCodes` should process both files
            1. Both files are read only once
            2. Creates and returns the final ArrayList
        iii. `lookupZip` is called in a while loop in `main`, once per lookup/user input
            1. Scanner for user input is created once outside of the loop, not over and over again. This breaks redirection.
6. Only one correctly-sized ArrayList of `Place` used and created only once
    a. An `ArrayList<Place>` of the appropriate size is created only once after uszipcodes.csv is read. It holds either `Place` or `PopulatedPlace` objects.
    b. When reading ziplocs.csv, replace `Place` with `LocatedPlace` or update `PopulatedPlace` objects in ArrayList with setters
    c. Any additional data structure -1
        i. This includes creating ArrayList in a loop over and over again
7. Binary search **12 points** - it is acceptable whether binary search is implemented imperatively (with a while loop) or recursively
    a. `Place` implements `Comparable`
    b. Binary search uses `Place` object comparison - this is tied into the one above. They should conduct binary search by making a new dummy `Place` object (with the given zip) and search with that as a target.
        i. If they simply used compareTo from String instead - in other words, their `Place` objects are not comparable, take 3 points off.
        ii. If they do other weird things, like converting to integers, compare string to a Place, etc, take more, upto 6 points

iii. If they didn't use compareTo at all, take 6 points off

c. Both searches use binary search, the one in `readZipCodes` and the one in `Main`