CS106/206 Assignment 3 Style Grading Rubrics

General

6 points are allocated to fairly mechanical rules on naming/comments/indentation - these should be easy to check off. Another 19 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 (**6 points total**)

1. Naming Conventions: **2 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 (**19 points total**)
Note that student README provides a discussion on how their design works. If the README doesn’t have the required discussion, take 5 points off.

Assignment 3 (Linked list)

1. private Instance variables and getters **1 point**
    a. Any non-private instance variables, including missing modifier
2. Use `public static final` constants instead of integer/double/String literals - any literal that has reason to be changed later should be a constant **1 point**
    a. Likely cases
        i. Using `"-f", "-m"` directly in code
        ii. Using `[0], [1], [2] …` directly in code after calling split on an input line
        iii. Character positions to use with substring to parse the year from filename
3. Constructor must initialize all instance variables **1 point**
4. Reasonable designs for `Name` **4 points**

    a. String instance variable for the actual name (probably not worth a point)
    b. An ArrayList or array for the yearly stats - an array is acceptable if they computed the size of the array from the number of files given in `args`.
    c. There should be a custom class that holds the three yearly statistics: the year, the rank and the number. No total here.
    d. Total number of babies with this name (int or double) - double is acceptable because then they won't have to cast to compute the percentages
    e. Percentages should be computed on the fly when reporting and not stored. Going back to compute all the percentages after file reading is done is a waste of loops
5. Storage for totals for percentage calculation **2 point**
    a. Yearly totals as an array or arrayList - again, array is acceptable if they computed the size - this list of totals should not be part of the `Name` class, but as additional data structures in `Main` or another auxiliary class. Type can be just int or an object that stores a year and the yearly total
    b. Grand totals as two integers/doubles or an array of two integers/doubles
6. No additional/redundant structures **3 points**
    a. Only two top-level linked lists of `Name`, two arraylists/arrays of yearly totals and two instance variables for the grand totals (as specified in 4 and 5)
    b. Take off more than 3 points if they really abuse unnecessary data structures:
        i. copying things in an array just so that they can use `Array.sort`
        ii. using a dictionary/map/hashmap for any reason
        iii. anything else I can't imagine
7. Redundant looping/methods **7 points**
    a. Each file is read once and only once
    b. `insertBefore` has no loop
    c. `insertSorted` only one O(n) loop
    d. `findName` only one O(n) loop
    e. For each name in each file, either it exists (via call to `findName`) already in the list, and an updater is called (no loop in the updater), or it doesn't and `insertSorted` is called
    f. Each total (includes all yearly totals and grand total) is computed once and stored.
    g. The yearly totals should be computed WHILE each file is being read with no additional loops (within the Scanner loops)
    h. The grand totals should also be computed WHILE the files are being read and finished as soon as the last file is read
    i. Total rank should be computed via an O(n) method per name, by searching through the linked list and counting all the names with a greater total