CS106/206 Assignment HC7/BMC8 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.

Total: 30 points

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 - no comment is okay if well-named
   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 (**24 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.

Assignment 8 (hashtable, sorting and complexity)

1. Design of class to hold a single row (**4 pts**)
   a. Implements `Comparable` and `compareTo` implements reasonable equality test - check uniqueness justification in README
2. Hashing (**7 pts**)
   a. `hashLinearDeduplication` (**3pts**)
      i. Check that the correct key is used in hashing - i.e. whatever `compareTo == 0` was using
      ii. Any of these to implement above is acceptable

            1. hash with a string v as key and the object as value
            2. override `hashCode` to return the correct key string
            3. `toString` returns key string and `hashCode` calls `toString`
    b. `hashDoubleDeduplication` (**3pts**)
        i. Secondary hash function reasonable
    c. Hashing stats correctly collected - only need two integers, total probes and total # of items, take off 2 points if they use an arrayList or some other unnecessary DS (**1pt**)
3. Sorting (**3pts+1EC**)
    a. `quickSortDeduplication`
        i. If quickSort is in place, give 1 EC
4. Complexity analysis (**10pts**)
    a. README efficiency discussion correct - you don't check correctness of the accompanied graphics, but glance at them since they might help (**5pts**)
        i. BMC submits Complexity.png, HC should have fibonacci.png and deduplication.png
        ii. Order should be AP > qsort > Collections.sort > linear hash > double hash, however, the two hashing might be indistinguishable. If double hashing performs much worse than linear probing, then something is probably wrong with their implementation of the secondary hash function
    b. Correct hashing statistics discussion in README (**5pts**)
        i. Linear probing should have near O(1) average probes
        ii. Double hashing should have smaller max probe number than linear hashing. Average probes should also be O(1)
        iii. Double hashing should be more efficient than linear probing, unless they made bad choices for their secondary hash function.

Deductions:

5. non-private instance variables -1 each
6. integer/string literals instead of constants -1 each
7. Additional unnecessary data structures (take off 3-5 points depending on how bad it is)
8. More loops/calls than necessary -1 each occurance