Hash Table Lab

"A picture is worth a thousand words."

After finishing each part of the lab, copy your entire project and work on the copy for the next part!

Part 3: Load Factor.

Do this once at program start up:

- Create an *ArrayList* of *String*s, with an initial size of 500K.
- Open the "Very Large Data Set.txt" – called *store* below (500K records)
- Read a line in the text file & add the string to the *ArrayList*
  - Repeat to the end of the file
- Close the file.
- Build similar *ArrayList* objects for the "Successful Search Records.txt" file and the "Unsuccessful Search Records.txt" file

For each test run, do the following:

1. Given an input file of 500k records, build a table sized for the load factor to be profiled
   - e.g, for an $\alpha = 0.50$, the table size should be ~1M
   - e.g, for an $\alpha = 0.67$, the table size should be ~750K
   - Use the following α values: 0.1, 0.5, 0.8, 0.9, 1.0
   - *[Hard code table sizes for each value of α]*

--- Measure table insertion performance ---

2. Start "Build Table" timer
   - `long start = System.currentTimeMillis();`
3. For each record in the "Very Large Data Set.txt" (large file)
   - Parse the record
   - Build *key* and *value* objects to be inserted in the table
   - Insert it into the hash table using the linear collision resolution scheme.
     - Increment collision counter for each unsuccessful probe
4. Stop "Build Table" timer
   - `long stop = System.currentTimeMillis();`

--- Measure performance for finding items in table ---

5. Start "Successful Search" timer
6. For each record in "Successful Search" data set
   - Parse the record
   - Build a *key* object
   - Search the table – get number of probes needed to find the entry
     - I.e., count number of objects checked before correct object is found
     - One counter for all items searched – get the average probes/search
7. Stop "Successful Search" timer

--- Measure search performance for items NOT in table ---

8. Start "Unsuccessful Search" timer
9. For each record in "Unsuccessful Search" data set
    - Parse the record
    - Build a *key* object
    - Search the table – get number of probes needed to determine the record isn't in the table
        - I.e., count number of objects checked before reaching a null entry
10. Stop "Unsuccessful Search" timer

11. Output a report containing:
    - Type of hashing used – Linear Probing
    - Hash function used – Integer value
    - Number of records added to the table, table size, and load factor
    - Average insertion time
    - Number of table insertion collisions
    - Number of collisions vs. number of insertions (expressed as %)
    - Average time & number of probes needed to find table entry
    - Average time & number of probes needed to determine entry is not in table

Note: it's not required, but you may find it convenient to output test run results to a text file. If you output data in CSV format, you can open the file directly in Excel. The `PrintWriter` *class is the easiest way to output text.*

Results Spreadsheet.

Transfer data from the program output text file to Excel – yes, use Excel!

Make the following charts:

1. 2D line chart: for building the table,
    a. Average number of probes vs. load factor α
    b. Average insertion time vs. load factor α
    c. Do the average time results tell you the same story as the number of probes results?

2. 2D line chart: for *successful* search,
    a. Average number of probes vs. load factor α
    b. Average time vs. load factor α
    c. Do the average time results tell you the same story as the number of probes results?

3. 2D line chart: for *UNsuccessful* search,
    a. Average number of probes vs. load factor α
    b. Average time vs. load factor α
    c. Do the average time results tell you the same story as the number of probes results?