CS 171 (Spring 2018)
Assignment 8: Hash Tables
Due: 4/25/2018 (11:59pm) via Turnin

Goals

- Complete the implementation of a hash table with separate chaining.
- Write a program to use the hash table.

Requirements:

1. Make the size of the hash table's array dynamic.

   The starter code in the file `HashSeparateChaining.java` contains a functional implementation of a hash table where collisions are resolved by separate chaining. Each key-value pair in the table is contained in an `Entry` object, defined in the `Entry.java` file and has the following properties:

   - The key is a `String`.
   - The value is an `Integer`.

   However, the size of the array that is used to store the entries is static in the starter code. From lecture, we know that this is not ideal because as more and more entries are added to the table, the average length of the chains will become longer and longer. Long chains destroy the efficiency of searching in the hash table, so we would like to avoid this by resizing the hash table's array when appropriate.

   For the first part of the assignment, you will write an implementation of `resize` method. You will also alter the `put` and `delete` methods to call upon the resize method at the approprate times.

2. Write an application which uses the hash table.

   For the second part of the assignment, you will write a program called `HashTableApp.java`. The main method of the program will take in one parameter, the name of a text file. It will read in the contents of the text file one word[1] at a time and maintains an entry in the hash table for each word where:

   - The key is the word from the text
   - The value is the number of times that the word has appeared in the text so far.

   The text words that are included in the table should only include letters and numbers (no punctuation), and the letters should be lower case. You may use any of the methods in Java's library to help you format your text that you like.

   To test your program, you should start by generating a few small text files where you can directly control the frequencies. For a larger test of your code, the file `WarAndPeace.txt` is given in the share directly, which contains the full text of the novel War and Peace.

   Note: There is a `toString` method included in `HashSeparateChaining` that will create a text representation of the hash table. You can use this to print out your hash table and look at its structure. The output will look bad if the chains get too long.

[1] A word does not include punctuation!

Getting Started

1. Get the starter code `HashSeparateChaining.java`, `Entry.java` and `HashTableApp.java` from Canvas directory and understand it. Since the hash table is a class, you may want to start by writing a short application that creates a hash table and adds/deletes a few entries and performs a couple of searches. Definitely read through each method carefully so that you understand exactly what it is doing and how it is doing it.
2. Complete the method `resize` and alter the methods `put` and `delete` to make appropriate use of resize. You should double the size whenever the average length of the chains exceeds 5 nodes.
3. Test your code to make sure that the hash table is still functional and that it makes appropriate use of resizing.
4. Write your application in a file named `HashTableApp.java`.

Honor Code
The assignment is governed by the College Honor Code and Departmental Policy. Please remember to have the following comment included at the top of the file.

```
/*
THIS CODE WAS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING
CODE WRITTEN BY OTHER STUDENTS. _Your_Name_Here_
*/
```

Submission:
Compress all your source code, including `HashSeparateChaining.java`, `Entry.java`, and `HashTableApp.java` files into a single zip file called `solution.zip` and upload the solution by opening the terminal and typing the following command:

```
>cd cs171
>/home/cs171003/turnin solution.zip hw8
```

Grading:

- Correctness and robustness of the hash table: your hash table resizes correctly and at the right times to maintain the average length of chains, the entries are never lost (i.e. If a key is added to a talbe, it should be searchable until it is removed with the `delete` method.) (40pts)
- Correctness and robustness of the application: your application should correctly read from the given text file and parse the words in the file. The hash table should have the correct frequencies for the words in the file, and should never crash. (40pts)
- Code clarity and style (20 pts)