CSCI 3901 Winter 2021

Lab 1: Basic Problem Solving

Name: Dhrumil Rakesh Shah

- **No Team Member (Doing Solo)**

- **A list of items from the description that needed clarification:**
  - Which data structure to use to implement the Map.
  - Do we have to implement a finite or an infinite Map?
  - What about Hash Collision?
  - What data type needs to be stored and retrieved from the Map. (homogeneous/heterogeneous)
    - That is whether the <key, value> pair needs to be of the same data type or a combination of data types as Java has Primitive and Non-Primitive data types.
  - Which hash function should be used to map the Key to an Index and is a Compressor required.
  - What would be the most efficient way to implement a Map? (execution & storage wise)
  - What about exceptions?
  - What happens when we try to remove a value in an empty Map?
  - What size is returned if the Map is empty?
  - Does the size go into negative if a value is removed from an empty Map?

- **Your decisions on the items that need clarification:**
  - I used an ArrayList to implement the Map
  - I used Linked List to store <Key,Value> pairs with the same hashed key to make it infinite and handle collision.
  - I used String as Key and int as value.
    - Could even use long instead of int.
  - In Java every Object has its own hash code, so I am using the has code generated by JVM in my program and as a Compressor I have used the modulo (%).
  - There are a lot of factors contributing to the efficiency:
    - The hash code function
    - The use of compressor
    - Load factor to increase the size of the Map/hash table.
  - Handled NullPointerException for both put and remove methods implemented
  - Null is returned if we try to remove from an empty Map.
  - 0 is returned as the size for an empty Map.
  - No the size of the Map implemented does not go to negative when remove is performed on an empty Map.

- **How you showed that your work (so far) is working:**
  - I am able to put() any <Key,Value> pair in the Map.
  - I am able to remove() any <Key,Value> pair in the Map.
  - I am able to get() any Value for a particular Key in the Map.
  - I am able to generate hashCode() for any Key in the program.
  - I am able to get the getSize() of the Map at any given point in time.
  - I am able to check whether the Map isEmpty() at any given point in time.
  - The getBucketIndex() is implemented correctly to get the hash value for any given Key using the hashCode() and using modulo as a compressor.
  - Taken care of NullPointerException that might be caused due to input validation problems.

- Tried to cover all possible cases relating to Map implementation.

- **What you did well in developing the implementation that you could use as an approach to implementing another problem:**
  - I first thoroughly analysed the problem and noted down points that were given in the problem statement.
  - Then I made a list of questions relating to the problem statement based on what the behaviour of a Map should be.
    - There were quite a few questions that I failed to recognize in the initial step, but when I started implementing the problem, I faced challenges which led me to ask better questions leading to better solutions.
  - After finalizing the list of questions, I tried to find a workable solution for as many questions as possible.
  - As I started to code, initially I covered the basic things required for Map implementation, then I tried to make it more robust & cleaner.
    - That includes properly commenting the code
    - Making it readable and easy to follow
  - Performed various tests on the program to cover all possible scenarios where the program could result in an exception and even tried handling exception
  - There are a few things that I was not able to implement to even further improve the quality & coverage of the program:
    - Load factor implementation to improve the efficiency of the program