Hash Table Lab

"I've been doing a lot of learning from mistakes, first and foremost, and building off that."

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

Part 2: Modify the *HashTable* class, implementing linear probing to handle collisions.

- Add a *size* field to *HashTable*.
  - Initialize to 0
  - Increment for *put*s decrement for *remove*s.
- Make the *Node* class an inner class (in *HashTable*).
- Add a *removed* field to *Node* to indicate an unused bucket.
- Implement the *remove* method:
  - If the *Node* exists, leave the *Node* in place & mark the *removed* field as *true*.
    - Use *equals* (on the key object) to verify you've found the correct object.
  - Add linear probing when collisions occur
    - Search until object is found or empty bucket encountered
    - Skip *removed* objects.
  - Return the previously stored value if the key is valid; otherwise, return *null*.
  - Decrement *size* if remove succeeded.
- Modify the *get* method:
  - If the *Node* exists, return the *value*.
    - Use *equals* (on the key object) to verify you've found the correct object.
  - If the *key* hashes to a different value, a collision occurred
    - Use linear probing to find the object
    - Search until object is found (verify with *equals*) or empty bucket encountered
    - Skip *removed* objects.
  - Return the value if the *key* is valid; otherwise, return *null*.

Continues on nextpage..

- Modify the *put* method:
    - Check *size* to be sure space is available.
    - If the hashed location is empty, store the value, increment *size*, & return *null*
    - If a collision occurs:
        - Is the object already stored in the table?
            - Use *equals* (on the key object) to verify
            - If duplicate, overwrite the location & return the previously stored value (don't increment *size*)
        - Not at the hashed location? Use linear probing to find an empty table location:
            - Check for duplicates at each location
            - If an empty location is encountered:
                - Save the new object
                - Return *null*
                - Increment *size*
            - If a *removed* location is encountered:
                - Save the new object
                - Return *null*
                - Increment *size*
                - Continue searching for a duplicate object until an empty bucket is encountered
                    - If a duplicate is found, mark it *removed* & decrement *size*
                - Search until an empty bucket is found
- Modify the *toString* method to print "dummy" for deleted locations.
- The driver routine should require no changes; run & validate the program.