1. public StringTable(int nBuckets);
   - Parameters:
     nBuckets: represents the number of buckets used in the string table.
   - Return value:
     Does not have a return value.
2. public boolean insert(Record r);
   - Parameters:
     r: type Record, telling if r has already been inserted?
   - Return value:
     Returns true or false.
3. public Record find(String key);
   - Parameters:
     key: type String, is the string that will be looked for.
   - Return value:
     Returns the object record if found, or null if not found.
4. public void remove(String key);
   - Parameters:
     key: type String, removes what every key was from the string table.
   - Return value:
     No return value.
5. private int toIndex(int hashcode);
   - Parameters:
     hashcode: type int, is the hashcode for a single string that can be used to find which bucket that string is in from mostly likely the mod function.
   - Return value:
     Returns an integer, meaning which bucket that string will be in.
6. What is the difference in meaning between the (public) size and the (private) nBuckets data members?

   Public size allows the method to be used in other classes while the nBuckets data members can only be accessed in that one class.

7. Which method(s) must you call, and in what order, to convert a String to an index into the array of hash buckets?

   toIndex(int hashcode) –¿ find(String key) –¿ add it to the array of hashbuckets

8. Where (i.e. in which method) will you allocate the linked list for each hash bucket? Note that "allocating" an object refers to reserving memory for the object and is usually done with the 'new' keyword in Java, as distinct from simply declaring an object.

   You need to allocate the linked list for each hash bucket in the constructor (StringTable(int nBuckets)).

9. Your hash calculations will involve the modulo operator '%'. Hashcodes in Java are signed integers; what does the Java modulo operator return when given a negative number? How might this impact your toIndex() function?

   It returns the remainder but keeps the negative sign, this results in a 'negative' bucket number when using the toIndex() function.