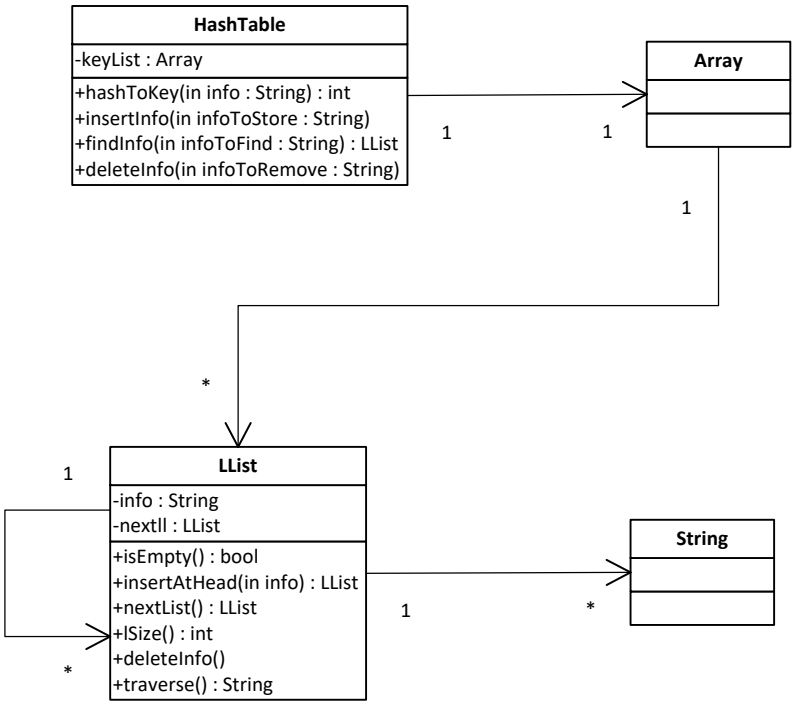
CMPT 270, Assignment #2, Due, May 26, 2017.

Hash Table using Linked Lists

In this assignment I want you to implement a hash table structure using the linked list you developed for assignment #1.

A hash table is a complex object build on top of another object, in this case a linked list. Here is the UML for the hash table:



The hash table contains an array of linked lists. Each linked list is a folded set of info to be stored in the hash table where all info elements in the list will have the same key value. The array is set to 15 elements in capacity. (NOTE: Both Array and String are Java classes – you do not have to implement these.)

Here are the story specs for each behavior:

Story 1: hashToKey(in info : String) : int

The input argument is the string of text to be hashed. The function returns an integer value in the range 0..14 by converting the two first characters of the info string into integers, adding them together and then computing the modulo 15. For example, suppose we have “Canada” as the info string. The key will be computed from ‘C’ and ‘a’ which are ASCII 67 and 97 respectively. 67+97 = 164, 164 mod 15 = 14 so the info string “Canada” will be inserted into the LList in index 14 of the array. (HINT: the following statements give the numeric int equivalent of the ‘C’ in “Canada”:

```
String s2 = new String("Canada"); // creates a String object
char aChar = s2.toCharArray()[0]; // String object creates an array of char with C in pos 0;
 System.out.println((int) aChar);  // output the int cast of the ‘C’ which is 67
```

Story 2: insertInfo(infoToStore) returns nothing.
Insert the infoToStore into the hash table based on its hash function key value.

Story 3: findInfo(infoToFind) returns LList.
This function performs the same computation as the insertInfo() function except returns a reference to the LList object containing the infoToFind or NULL if the infoToFind is not in the hasTable.

Story 4: deleteInfo(infoToRemove) returns nothing.
This function determines the key of the infoToRemove then simply tells the LList at the key location within the array to remove that info from it.

NOTES:
Consider the following design issues:

- The current max size of the LList array is 15 but may be changed in the future. Do not use a constant, such as 15, throughout the code.
- The same information may exist more than once in the information LList. If it does, deleteInfo() removes only the first occurrence, not all occurrences.

Requirements for this assignment:
Submit the class code for the HashTable, the test code for this class, and the LList code you used (yours or mine) so that we have the complete source code. Please document whether you used my code or your own from assignment #1. If you are programming in pilot/copilot configuration, identify both of you by name and NSID within every source code file and every documentation file you upload.

- [5] Correctly manipulate the info string to compute the index key for determining where in the hash table the info should go
- [20] Provide a set of tasks for each story in the specification (i.e. what are the things you have to accomplish to have this class operate as described) (0.5 for each story)
- [5] efficient use of the LList within the array
- [10] correct overall implementation of the hashTable class
- [10] You do not have to provide incremental development TDD examples for this assignment. I do want the final TDD test app however that demonstrates that your hash table class works properly.

And of course the highest priority is HAVE FUN!!

Total 50.