COP 3540 – Data Structures with OOP

Project 5 – Spring 2011

Due: 22 April 2011 at 2pm – Note: No late assignments accepted after this date.

Hash Tables

Using NetBeans 6.9.1, you are to write a Java program using OOP principles to accommodate the following functionality

Assignment #5

Objectives:

Provide student with exercises in learning UML
Provide student with exercises in Javadoc and its various formats
Provide student with exercises in hashing (randomizing) and collision processing
Provide student with experience in inserting, deleting, and changing node contents in a hash table

Functionality:

Given a sequential file, *team.datafile.txt* on my web page, you are to build a hash table based on team nickname ONLY. This means that the city name, state, etc. are NOT required. Nicknames are, for example, Jaguars and NOT Jacksonville Jaguars; Colts and NOT Indianapolis Colts, okay? Just the nickname.

Task 1: Build Hash Table: You are to build the hash table in the order of appearance of the records in this input data file You are to use the hashing of strings algorithm (see book and lecture notes part 2) for the algorithm as your hashing function. You are to use separate chaining as your collision algorithm (see also textbook and my lecture notes). Your linked list does not have to be sorted.

Task2: Display Hash Table: Once the hash table is built from the input constrained by the hashing and collision algorithms cited above, you are to display the hash table and its links. Your format – always include nice headers, etc. – should appear in tabular format: Hash table index (integer) in the left most column followed by, say, five spaces, followed by the nickname (left justified) stored in this home address. Any items linked to this home address are to appear **left justified** to the **right** of this first entry, with five spaces in between any succeeding collision at that hash table address. If a hash table entry, say item 16 has no occupants, then the 16 is still to appear in your output format, but it will have no entries to its right. A home address having no occupants should contain the letters, null

The size of the hash table is to be 32. Thus output data must have 32 detail lines.

Task 3: Update the Hash Table: Using the *team.transactionfile.new.txt*, you are to update your hash table. Inserts should be simple and use the same hashing and collision algorithms that were used to build the table. A ➔ Add; C ➔ Change; D ➔ Delete.

Deletions: These are to be flagged with a delete byte. Items are to be logically deleted and not physically deleted.

Changes: Here, you will accommodate a delete-add pair. You must find the nickname to be changed, delete that entry (as above) and then hash using the new nickname to the proper index in the hash table and insert this new nickname.

Task 4: Display Updated Hash Table: Using the format described above, you are to display the updated hash table and its links. For those records that are logically deleted, but still physically present, you are to precede the nickname with an asterisk, as in: *Ravens.

As usual, you have **PLENTY of time if you start right away and work slowly and methodically. Don't get caught by some surprise late in the game.**

Deliverable: Your zipped folder to me MUST include copies of ***teamdatfile.txt*** and ***team.transactionfile.new.txt.*** .

You are to zip all files in your P5 folder as expected and Send them to be via Blackboard using our standard naming conventions.

Grade Sheet – Program is worth 80 points.

Source Code – 10 points

I look for: Indentation
Internal comments
Scope terminators
Overall program structure

Javadoc – 10 points -

I look for:

Appropriateness and completeness of comments
ALL methods must have Javadoc comments up front that are meaningful, please.
ALL methods require @params, @returns, and a description of the functionality

UML – 10 points – no excuse to not have this wonderful at this time. We have discussed formats a number of times. I will grade this closely.

Correctness, associations, completeness. This means that the classes you identify are correct, that associations are indicated, and that the attributes and methods are documented within the classes. Methods must show method name, input arguments, and returned item **in UML format – not Java format**.

Outputs – 50 points

Accuracy and **Format. All functionality present!**
Skip lines in between displayed outputs as described above. .
Include headers / descriptors as you may feel appropriate, but they need to be respectable by my standards..

➔ If your program does not contain all the functionality required, you cannot expect a grade above 60, that is a 'passing grade.' Please note that this will be rigidly enforced.

Program must run correctly to receive a passing grade and to receive at least partial credits above. If the program does not successfully run to end of job and include all major functional requirements, all bets are off.

Start early and do this a little at a time.

Good luck and have fun!!!