# Report : Assignment 1

# Requirements

## Task 1

**FR 1**: You must create a Java class *Member* that implements the interface *IMember*.

Completed

public class Member implements IMember *{*

**FR2:** You must also implement a constructor or constructors for this class.

Completed

4 marks

## Task 2

**FR3**: You must create a Java class called *MemberHash* that implements the interface *IMemberDB*.

Completed

public class MemberHash implements IMemberDB *{*

**DR3:** You must implement *MemberHash* as a hash table

Completed

**DR4:** The constructor for *MemberHash* must print the string “Hash Table” to *System.out*.

Completed

public MemberHash*() {* System.*out*.println*(*"-- Hash Table --"*)*;  
*}*

**DR5:** You **must** use a Java *array* for implementing the hash table (not an array list or other collection class). You **must not** encapsulate existing implementations of collections in your submission. For example, you **must not** create a *HashMap* object and call methods on that object from your class. Failure to comply with this will result in **zero marks** for this part.

You may take a ‘lazy’ approach to deletion, such as marking the item as no longer present.

Take care that you have not used a linear search *O(n)* in the hash table where you should have used hashing, aiming towards *O(1).*

Completed

static class Node *{* final int hash;  
 final String key;  
 String value;  
 Node next;

}

Node*[]* table = new Node*[DEFAULT\_INITIAL\_CAPACITY]*;

6 marks

**DR6**: You must devise your own *hash function* that will work well for surnames (family names)

You may assume that the names are expressed solely in uppercase ISO basic Latin alphabet (ASCII). You may also assume that enough additional parts are appended to names so that no two members have exactly the same name. You need not regard uppercase and lowercase as the same.

You will need to perform some experiments to find a good hash function. Include an explanation of your hash function and your experiments in your report.

Take care to avoid *numeric overflow* when calculating your hash function. This can be done by applying a mod (%) operation at early stages of the calculation, rather than just at the end.

Completed

4 marks

hash function

public int myHashCode*(*String value*) {* int h = 0;  
 if *(*value.length*()* > 0*) {* for *(*int i = 0; i < value.length*()*; i++*) {* h = 31 \* h + value.charAt*(*i*)*;  
 *}  
 }* return h;  
*}*

public final int hashCode*() {* return *myHashCode(*key*)* ^ *myHashCode(*value*)*;  
*}*

用质数31和ASCII码求hash值 并用XOR来尽量减少冲突

**DR7:** You must not use the Java built-in *hashCode* method, though you can experiment with it.

Completed

**FR4**: You must ensure that *collisions* are catered for well and do not lead to excessive clustering. You will need to perform some experiments to find a good collision-resolution strategy. Include an explanation of your collision-resolution strategy and your experiments in your report.

Completed

用链表 及拉链法 解决个别冲突

putVal方法:

public String putVal*(*int hash, String key, String value*) {* int i;  
 Node*[]* tab = table;  
 Node p;  
 int n;  
 if *(*tab == null || *(*n = tab.length*)* == 0*) {* n = *(*tab = resize*())*.length;  
 *}* if *((*p = tab*[*i = *(*hash & *(*n - 1*))])* == null*) {* tab*[*i*]* = new Node*(*hash, key, value, null*)*;  
 *}* else *{* Node e;  
 String k;  
 if *(*p.hash == hash && *(*Objects.*equals(*k = p.key, key*)* || k.equals*(*key*))) {* e = p;  
 *}* else *{* while *(*true*) {* if *((*e = p.next*)* == null*) {* p.next = new Node*(*hash, key, value, null*)*;  
 break;  
 *}* if *(*e.hash == hash && Objects.*equals(*e.key, key*)) {* break;  
 *}* p = e;  
 *}  
 }* if *(*e != null*) {* String oldVal = e.value;  
 e.value = value;  
 System.*out*.println*(*"name: " + key + " hash: " + hash + " threshold: " + threshold + " size: " + size*()* + " loadFactor: " + size*()* \* 1.0 / threshold \* loadFactor *)*;  
 return oldVal;  
 *}  
 }* this.table = tab;  
 ++modCount;  
 if *(*++size > threshold*) {* resize*()*;  
 *}* System.*out*.println*(*"name: " + key + " hash: " + hash + " threshold: " + threshold + " size: " + size*()* + " loadFactor: " + size*()* \* 1.0 / threshold \* loadFactor *)*;  
 return null;  
*}*

**DR8**: You must *chaining* with a linked list implemented devised by you (not using the *LinkedList* or *ArrayList* or any other class from the Java library). You will need to make a linear search within the chain.

Completed

4 marks

## Task 3

**DR9**: You must make appropriate use of assertions (*assert* statements) to protect preconditions of the operations. Remember to enable assertion checking for your project.

Completed

assert oldTab != null;

4 marks

## Task 4

**DR10**: Your class must keep track of the ‘load factor’ of the hash table and display this after each insertion or deletion. Note that with chaining the load factor can exceed 100%.

Completed

System.*out*.println*(*"name: " + key + " hash: " + hash + " size: " + size*()* + " loadFactor: " + size*()* \* 1.0 / threshold \* loadFactor + " threshold: " + threshold*)*;

分别打印 名字、哈希值、map大小、负载因子、map当前阈值

2 marks

## Task 5

**DR11**: You must make your class *log* monitoring information, either to a text file or by calls of *System.out.println*.

It must log (at least):

* the *size* and *load factor* of the hash table after each addition;
* for every *addition* of a Member (*put*), attempt to *get* the Member or attempt to *remove* the Member:
  + the *Member name;*
  + the *hash value* calculated for it ;
  + the sequence of buckets (array locations and list nodes) visited.
* Paste your log into your report.

Completed

D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? p  
 Name? a  
 Affiliation? ads  
 name: a hash: 97 threshold: 768 size: 501

loadFactor: 0.4892578125  
 a : new member added

如 p(put) 方法输出的blog

4 marks

## Task 6

**DR12:** You must devise a test plan for your implementation. Be sure to check (among many other cases):

* each of the implemented methods is working correctly
* that values added can subsequently be found
* that existing values are not lost when other values are added
* that deleting an item does not disturb the other items

The test plan must include, for each test case:

* Description of case to be tested
* Value(s) to be used
* Expected results
* Actual results (placeholder for Task 6)

Completed

读入测试文件后如下操作:

Loading completed!  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? p  
 Name? a  
 Affiliation? asdasd  
 name: a hash: 97 threshold: 768 size: 501 loadFactor: 0.4892578125  
 a : new member added  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? p  
 Name? a  
 Affiliation? ffffff  
 name: a hash: 97 threshold: 768 size: 501 loadFactor: 0.4892578125  
 a : member overridden  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? g  
 Name? a  
 name: a hash: 97 threshold: 768 size: 501 loadFactor: 0.4892578125  
 affiliation: ffffff  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? c  
 Name? a  
 a found  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? s  
 Size 501  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? r  
 Name? a  
 Before delete:  
 name: a hash: 97 threshold: 768 size: 501 loadFactor: 0.4892578125  
 After delete:  
 name: a hash: 97 threshold: 768 size: 500 loadFactor: 0.48828125  
 a : deleted  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? g  
 Name? a  
 a not found  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? c  
 Name? a  
 a not found  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? p  
 Name? a  
 Affiliation? ssssssssssssssss  
 name: a hash: 97 threshold: 768 size: 501 loadFactor: 0.4892578125  
 a : new member added  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? s  
 Size 501  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? g  
 Name? a  
 name: a hash: 97 threshold: 768 size: 501 loadFactor: 0.4892578125  
 affiliation: ssssssssssssssss  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit? c  
 Name? a  
 a found  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit?

全部符合测试预期

4 marks

## Task 7

**DR13:** By using the supplied main program, or by other means, you must test your *MemberHash*. Set the capacity of the hash table to a small value so that collisions are sure to occur.

(You will use the same main program for Assignment 2 simply by replacing the implementation of *IMemberDB* that uses a hash table by the one that uses a binary search tree.)

Include your test plan, test data used, expected results and actual results in your report. You must show your actual results and the logging information copied from your log file or the output pane of your IDE. Do not simply state “test passed”, or similar – show evidence!

Completed

使用给定csv测试文件测试预计结果与实际一致

输出:

-- Hash Table --  
 Load file? Y/N y  
 name: Butt, James hash: -1677742750 threshold: 12 size: 1 loadFactor: 0.0625  
 name: Darakjy, Josephine hash: -1876323578 threshold: 12 size: 2 loadFactor: 0.125  
 name: Venere, Art hash: 1416090129 threshold: 12 size: 3 loadFactor: 0.1875  
 name: Paprocki, Lenna hash: 1578806101 threshold: 12 size: 4 loadFactor: 0.25  
 ...  
 ...  
 name: Kampa, Raylene hash: -1674648784 threshold: 768 size: 497 loadFactor: 0.4853515625  
 name: Bookamer, Flo hash: -1974379166 threshold: 768 size: 498 loadFactor: 0.486328125  
 name: Biddy, Jani hash: 1637075179 threshold: 768 size: 499 loadFactor: 0.4873046875  
 name: Motley, Chauncey hash: 247862466 threshold: 768 size: 500 loadFactor: 0.48828125  
 Loading completed!  
 D)isplay P)ut G)et C)ontains S)ize R)emove Q)uit?

6 marks

## Task 8

**DR14**: You must state honestly which of the requirements of Assignment 1 you have successfully fulfilled, citing evidence. Also comment on the *time efficiency* and *space efficiency* of your implementation of the hash table.

Completed

已实现全部要求

时间复杂度O(1)

空间复杂度 O(N)

2 marks

total 40 marks