Computer Science 3A

Practical Assignment 7

18 April 2024

Time: 18 April 2024 13:45 – 17:00 Marks: 50

Practical assignments must be uploaded to `eve.uj.ac.za` **before** 18h00 in the practical session.

Late submissions **will not be accepted**, and will therefore not be marked. You are **not allowed to collaborate** with any other student. You **must** upload your assignment to Eve **before** it will be marked.

Hash Tables are a quick and convenient method of implementing a Map ADT. The basic principle of a Map is that items can be added and removed from the map, the items are associated with Key and Value pairs, and that there is no duplication of keys.

You are required to implement a HashTable that makes use of the hash function defined below. This has been defined for a byte array but your implementation should allow strings and integers to be hashed using this function (the Integer and the String should be converted to a byte array). Objects that are not Integers or Strings should make use of the built in Java hashCode function for the object.

```
Input: a byte array
Output: a hash code

hash <- 5381
for each byte b:
hash <- ((hash << 5) + hash) + b
return hash
```

Your implementation will make use of a bucket array, where each element in the hash table is a bucket of items. The buckets are realised by creating `PositionList<Entry<K,V>>` objects. Each `PositionList` stores `Entry` objects. The most convenient method of implementation is to make use of an array of Objects where each Object is initialised to be a `PositionList<Entry<K,V>>` object. This is a work around as remember Java does not like to create Generic arrays, and nested Generic array are even more difficult to create. The only consideration that you have to make in this case is to convert the bucket to a `PositionList<Entry<K,V>>` object before you use it.

You must complete the classes and methods marked by:

```
//TODO: COMPLETE CODE HERE
```

A test class has been provided to test your implementation. You should not add any extra functions to the provided classes, only complete the methods that have been indicated.

The results of your application should be as follows (your answer will be slightly different as random strings are used for testing):

```
925 -> macPWnbgPo
189 -> WVdEisxw[b
273 -> SrpMG[sTtO
206 -> gEScwMugRG
753 -> LLOrgwUWaF
207 -> afZRTngQbn
53 -> juyvRTqriL
63 -> XscVZIWsaY
306 -> jTTtaqECgm
532 -> o[OrxJyXvJ

Values:
macPWnbgPo
WVdEisxw[b
SrpMG[sTtO
gEScwMugRG
LLOrgwUWaF
afZRTngQbn
juyvRTqriL
XscVZIWsaY
jTTtaqECgm
o[OrxJyXvJ

Removing items:
Removed: macPWnbgPo
Removed: WVdEisxw[b
Removed: SrpMG[sTtO
Removed: gEScwMugRG
Removed: LLOrgwUWaF
Removed: afZRTngQbn
Removed: juyvRTqriL
Removed: XscVZIWsaY
Removed: jTTtaqECgm
Removed: o[OrxJyXvJ

Showing values again:

String test:
```

```
LXXdHYZ[KG -> 861
TOwVfdgwL[ -> 85
hJNmmpZCeZ -> 274
blJddMJ[qR -> 426
KRcwQKDOxd -> 231
PFCbEGNVLk -> 927
WFUwZjTSQb -> 948
R[iVWxUxos -> 762
Yhg[PoFJTl -> 664
NDKvIZgetJ -> 73

Values:
861
85
274
426
231
927
948
762
664
73

Removing items:
Removed: 861
Removed: 85
Removed: 274
Removed: 426
Removed: 231
Removed: 927
Removed: 948
Removed: 762
Removed: 664
Removed: 73

Showing values again:
```

The following files must be submitted to EVE:

1. *studentnumber*_p8.zip

Bonus

For an additional 15 marks make a Linear Probing Based Hashtable which is the same as the *HashTable* class, but uses a linear probing based collision handling approach. So it requires a new *remove*, *get* and *put* methods that do NOT make use of a PositionList at each index, but store a single Entry instead.

Marksheet

1. HashTable: createArray	[4]
2. HashTable: remove	[10]
3. HashTable: get	[10]
4. HashTable: put	[8]
5. HashTable: keys	[8]
6. Compilation and Correct execution.	[10]