GIT Department of Computer Engineering

CSE 222/505 – Spring 2021

Homework5 # Report

Ozan GECKİN

1801042103

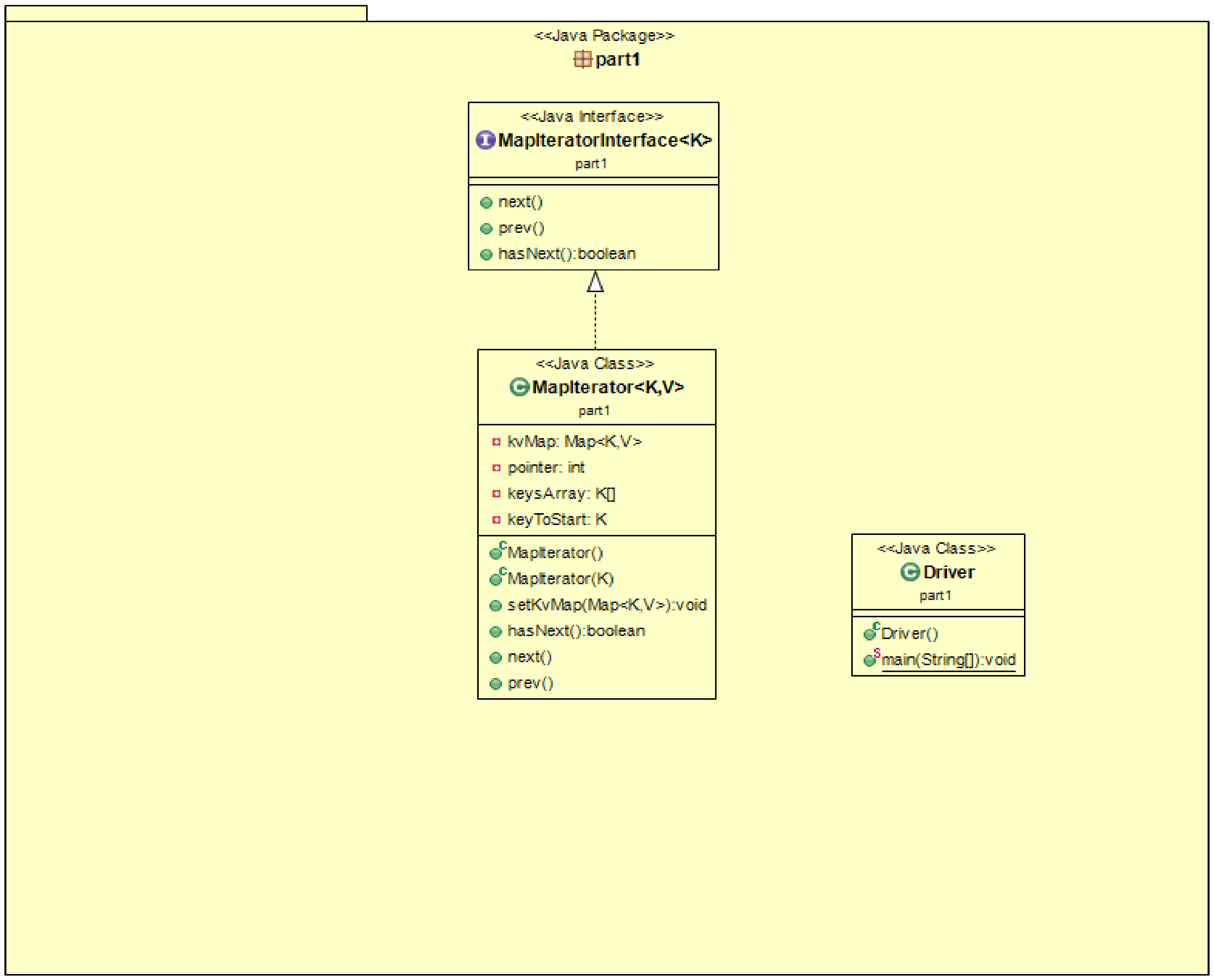
1. **System requirements:**

   I coding my homework in Eclipse 18.04 in java 8. Then I tested ubuntu 14.04 java 8. It works on two of them.

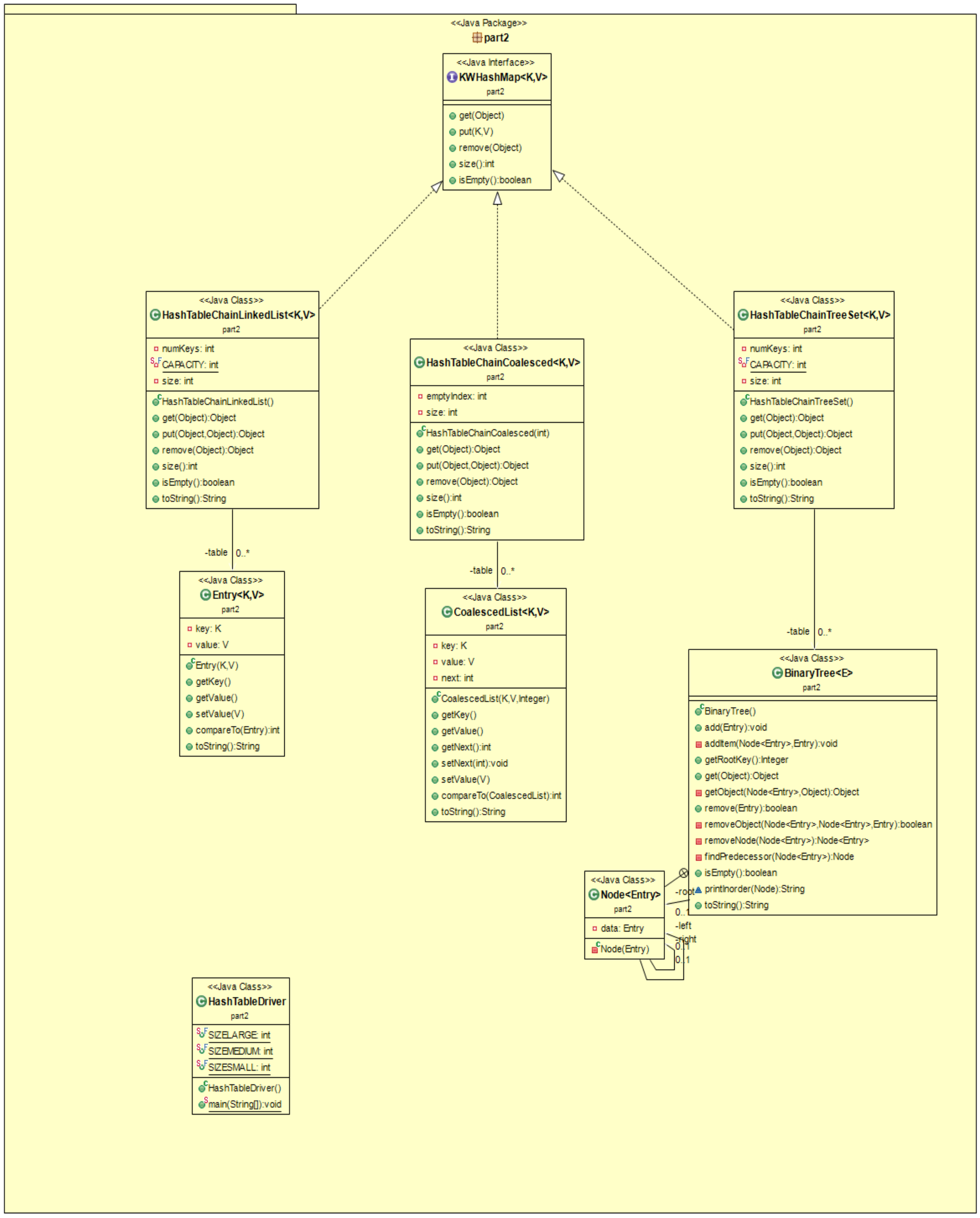
   Complier for Ubuntu Part1: javac Driver.java
   Run command: java Driver
   Complier for Ubuntu part2: javac HashTableDriver.java
   Run command: java HashTableDriver

2. **Class Diagram:**

   Part 1:



Part 2:



<<Java Package>>
part2
<<Java Interface>>
KWHashMap<K,V>
part2
get(Object)
put(K,V)
remove(Object)
size():int
isEmpty():boolean
<<Java Class>>
HashTableChainLinkedList<K,V>
part2
numKeys: int
CAPACITY: int
size: int
HashTableChainLinkedList()
get(Object):Object
put(Object,Object):Object
remove(Object):Object
size():int
isEmpty():boolean
toString():String
<<Java Class>>
HashTableChainCoalesced<K,V>
part2
emptyIndex: int
size: int
HashTableChainCoalesced(int)
get(Object):Object
put(Object,Object):Object
remove(Object):Object
size():int
isEmpty():boolean
toString():String
<<Java Class>>
HashTableChainTreeSet<K,V>
part2
numKeys: int
CAPACITY: int
size: int
HashTableChainTreeSet()
get(Object):Object
put(Object,Object):Object
remove(Object):Object
size():int
isEmpty():boolean
toString():String
-table
0..*
<<Java Class>>
Entry<K,V>
part2
key: K
value: V
Entry(K,V)
getKey()
getValue()
setValue(V)
compareTo(Entry):int
toString():String
-table
0..*
<<Java Class>>
CoalescedList<K,V>
part2
key: K
value: V
next: int
CoalescedList(K,V,Integer)
getKey()
getValue()
getNext():int
setNext(int):void
setValue(V)
compareTo(CoalescedList):int
toString():String
-table
0..*
<<Java Class>>
BinaryTree<E>
part2
BinaryTree()
add(Entry):void
addItem(Node<Entry>,Entry):void
getRootKey():Integer
get(Object):Object
getObject(Node<Entry>,Object):Object
remove(Entry):boolean
removeObject(Node<Entry>,Node<Entry>,Entry):boolean
removeNode(Node<Entry>):Node<Entry>
findPredecessor(Node<Entry>):Node
isEmpty():boolean
printInorder(Node):String
toString():String
<<Java Class>>
Node<Entry>
part2
data: Entry
Node(Entry)
-root
0..1
-left
-right
0..1
0..1
<<Java Class>>
HashTableDriver
part2
SIZELARGE: int
SIZEMEDIUM: int
SIZESMALL: int
HashTableDriver()
main(String[]):void

3.Problem Solution Approach

My Problem solution steps are;

-Specify the problem requirements

-Analyze the problem

-Design an algorithm and Program

-Implement the algorithm

-Test and verify the program

3.1) Specify the problem requirements: I understand the problem.

3.2) Analyze the problem: I identify; input data, output data, Additional requirements and constraints

3.3) Design an algorithm and program: My solution approach is completely based on OOP. I divide the problem into sub-problems. I listed major steps (subproblems). I break down each step into a more detailed list. To do these We have to divide this big project into small pieces. First of all I started with user type we need divide them with seperate classes. And then to handle all data we need to write a class that manipulate the data. And lastly if we combine all these we can complete the project

3.4) Implement the algorithm:

Part 1 :

MapIterator<K, V> implements MapIteratorInterface<K> class: MapIterator class implements MapIterator Interface that iterates through the keys in a HashMap data structure. Map keys are stored in an array and pointer for next and previous methods works on this key array. The iterator starts from the given key and iterate though all the keys in the Map. The iterator starts from any key in the Map when the starting key is not in the Map or not specified.

Methods:

next() : The function returns the next key in the key array. It returns the first key when there is no not-iterated key in the Map.

prev() : The function returns the previous key in the key array. It returns the last key when the iterator is at the first key.

hasNext() : The method returns True if there are still not-iterated key/s in the Map, otherwise returns False.

MapIterator (K key) : The method sets the starting key.

setKvMap(Map<K, V> kvMap): This method sets the key array starting from key to start parameter Iterator will iterate over this key array

Part 2:

Classes :

Entry<K, V> implements Comparable<Entry>: A hash table stores key–value pairs, so we will use an inner class Entry in each hash table implementation with data fields key and value.

BinaryTree<E>: TreeSet implementation for chainin items in the same hash table slot.

CoalescedList<K, V> implements Comparable<CoalescedList>: Hash table entry class for Coalesced hashing technique. Each item in the hash table is stored as key-value pairs. To access next item with same hash, pointer is added.

HashTableChainLinkedList<K, V> implements KWHashMap: In a hash table that implements chaining , we represent the hash table as an array of Linked Lists.

HashTableChainLinkedList class methods:

put() : The method adds new item into hash map. Hash value of key is calculated. If the table slot is not empty key is searched in the list. If it's found value is updated. If not new element is added to linked list.

remove() : The method removes the item with given key from hash table. Hash key value is calculated and searched in the list in the hash table. If it's found removed from the list and value is returned.

get() : The method returns the corresponding value from hash table.

HashTableChainTreeSet<K, V> implements KWHashMap: We represent the hash table as an array of Binary Trees.

HashTableChainTreeSet class methods:

put() : The method adds new item into hash map. Hash value of key is calculated. New Binary Tree is created if the corresponding slot is empty. Item is added as a tree node.

remove() : The method removes the item with given key from hash table. Hash key value is calculated and removed from the tree.

get() : The method returns the corresponding value from hash table.

HashTableChainCoalesced<K, V> implements KWHashMap: In a hash table that implements chaining , we represent the hash table as an array. Each node in the array has three values : key, value and the link to the next colliding element.

HashTableChainCoalesced class methods:

put() : The method inserts the key according to the hash value of that key if that hash value in the table is empty otherwise the key is inserted in first empty place from the bottom of the hash table and the address of this empty place is mapped in NEXT field of the previous pointing node of the chain.

remove() : The key if present is deleted. Also if the node to be deleted contains the address of another node in hash table then this address is mapped in the NEXT field of the node pointing to the node which is to be deleted

get() : The method returns the corresponding value from hash table.

4.Test Case

Part 1:

Test ID	Test Case	Expected Result	Pass/Fail
T1	Test MapIterator class methodu setKvMap() for String type .	Setting the hashMap of the MapIterator object.	Pass
T2	Test MapIterator classı methodu hasNext(), next() ve get() for String type.	Printing the hashMap inside the MapIterator object.	Pass
T3	Test One key parametre MapIterator constructurı . setKv Map for String Type.	Sorting starting from key	Pass
T4	Test MapIterator classı methodu setKvMap() for Integer type .	Setting the hashMap of the MapIterator object.	Pass
T5	Test MapIterator classı methodu hasNext(), next() ve get() for Integer type.	Printing the hashMap inside the MapIterator object.	Pass
T6	Test One key parametre MapIterator constructurı . setKv Map for Integer Type.	Sorting starting from key	Pass

Part 2:

Test ID	Test Case	Expected Result	Pass/Fail
T1	Test put element up to HashTableChainLinkedList sizelarge	Adding all elements to HashTableChainLinkedList	Pass
T2	Test put element up to HashTableChainLinkedList mediumlarge	Adding all elements to HashTableChainLinkedList	Pass

T3	Test put element up to HashTableChainLinkedList smallarge	Adding all elements to HashTableChainLinkedList	Pass
T4	Test size method HashTableChainLinkedList if HashTableChainLinkedList is not null	Return size	Pass
T5	Test size method HashTableChainLinkedList if HashTableChainLinkedList is null	Return null	pass
T6	Test get method HashTableChainLinkedList	Return value by key	Pass
T7	Test get method HashTableChainLinkedList invalid key	Return null	Pass
T8	Test remove method HashTableChainLinkedList	Remove value in key	Pass
T9	Test remove method HashTableChainLinkedList invalid key	Not remove	Pass
T10	Test IsEmpty method HashTableChainLinkedList that is empty	Return true	Pass
T11	Test IsEmpty method HashTableChainLinkedList that is full	Return false	Pass
T12	Test put element String type Key and value HashTableChainLinkedList	Adding all elements to HashTableChainLinkedList	Pass
T13	Test size method String type Key and value HashTableChainLinkedList	Return size	Pass
T14	Test get method String type Key and value HashTableChainLinkedList	Return value by key	Pass
T15	Test get method String type Key and value HashTableChainLinkedList invalid key	Return null	Pass
T16	Test remove method String type Key and value HashTableChainLinkedList	Remove value in key	Pass
T17	Test remove method String type Key and value HashTableChainLinkedList invalid key	Not remove	Pass

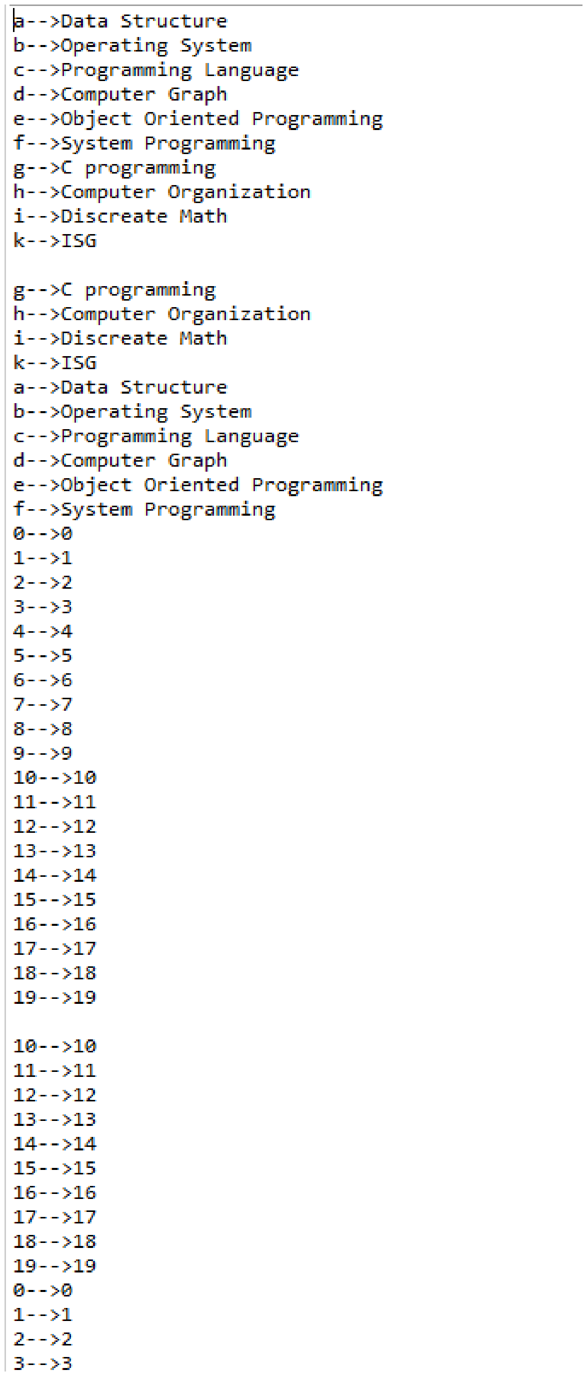
T18	Test IsEmpty method String type Key and value HashTableChainLinkedList that is empty	Return true	Pass
T19	Test IsEmpty method String type Key and value HashTableChainLinkedList that is full	Return false	Pass
T20	Test put element up to HashTableChainTreeSet sizelarge	Adding all elements to HashTableChainTreeSet	Pass
T21	Test put element up to HashTableChainTreeSet mediumlarge	Adding all elements to HashTableChainTreeSet	Pass
T22	Test put element up to HashTableChainTreeSet smallarge	Adding all elements to HashTableChainTreeSet	Pass
T23	Test size method HashTableChainTreeSet if HashTableChainTreeSet is not null	Return size	Pass
T24	Test size method HashTableChainTreeSet if HashTableChainTreeSet is null	Return null	Pass
T25	Test get method HashTableChainTreeSet	Return value by key	Pass
T26	Test get method HashTableChainTreeSet invalid key	Return null	Pass
T27	Test remove method HashTableChainTreeSet	Remove value in key	Pass
T28	Test remove method HashTableChainTreeSet invalid key	Not remove	Pass
T29	Test IsEmpty method HashTableChainTreeSet that is empty	Return true	Pass
T30	Test IsEmpty method HashTableChainTreeSet that is full	Return false	Pass
T31	Test put element String type Key and value HashTableChainTreeSet	Adding all elements to HashTableChainTreeSet	Pass
T32	Test size method String type Key and value HashTableChainTreeSet	Return size	Pass

T33	Test get method String type Key and value HashTableChainTreeSet	Return value by key	Pass
T34	Test get method String type Key and value HashTableChainTreeSet invalid key	Return null	Pass
T35	Test remove method String type Key and value HashTableChainTreeSet	Remove value in key	Pass
T36	Test remove method String type Key and value HashTableChainTreeSet invalid key	Not remove	Pass
T37	Test IsEmpty method String type Key and value HashTableChainTreeSet that is empty	Return true	Pass
T38	Test IsEmpty method String type Key and value HashTableChainTreeSet that is full	Return false	Pass
T39	Test put element up to HashTableChainCoalesced sizelarge	Adding all elements to HashTableChainCoalesced	Pass
T40	Test put element up to HashTableChainCoalesced mediumlarge	Adding all elements to HashTableChainCoalesced	Pass
T41	Test put element up to HashTableChainCoalesced smallarge	Adding all elements to HashTableChainCoalesced	Pass
T42	Test size method HashTableChainCoalesced if HashTableChainCoalesced is not null	Return size	Pass
T43	Test size method HashTableChainCoalesced if HashTableChainCoalesced is null	Return null	pass
T44	Test get method HashTableChainCoalesced	Return value by key	Pass
T45	Test get method HashTableChainCoalesced invalid key	Return null	Pass
T46	Test remove method HashTableChainCoalesced	Remove value in key	Pass

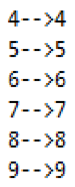
T47	Test remove method HashTableChainCoalesced invalid key	Not remove	Pass
T48	Test IsEmpty method HashTableChainCoalesced that is empty	Return true	Pass
T49	Test IsEmpty method HashTableChainCoalesced that is full	Return false	Pass
T50	Test put element String type Key and value HashTableChainCoalesced	Adding all elements to HashTableChainCoalesced	Pass
T51	Test size method String type Key and value HashTableChainCoalesced	Return size	Pass
T52	Test get method String type Key and value HashTableChainCoalesced	Return value by key	Pass
T53	Test get method String type Key and value HashTableChainCoalesced invalid key	Return null	Pass
T54	Test remove method String type Key and value HashTableChainCoalesced	Remove value in key	Pass
T55	Test remove method String type Key and value HashTableChainCoalesced invalid key	Not remove	Pass
T56	Test IsEmpty method String type Key and value HashTableChainCoalesced that is empty	Return true	Pass
T57	Test IsEmpty method String type Key and value HashTableChainCoalesced that is full	Return false	Pass

4.Running And Result:

Part 1:



```
a-->Data Structure
b-->Operating System
c-->Programming Language
d-->Computer Graph
e-->Object Oriented Programming
f-->System Programming
g-->C programming
h-->Computer Organization
i-->Discreate Math
k-->ISG

g-->C programming
h-->Computer Organization
i-->Discreate Math
k-->ISG
a-->Data Structure
b-->Operating System
c-->Programming Language
d-->Computer Graph
e-->Object Oriented Programming
f-->System Programming
0-->0
1-->1
2-->2
3-->3
4-->4
5-->5
6-->6
7-->7
8-->8
9-->9
10-->10
11-->11
12-->12
13-->13
14-->14
15-->15
16-->16
17-->17
18-->18
19-->19

10-->10
11-->11
12-->12
13-->13
14-->14
15-->15
16-->16
17-->17
18-->18
19-->19
0-->0
1-->1
2-->2
3-->3
```



```
4-->4
5-->5
6-->6
7-->7
8-->8
9-->9
```

For Part 1, T1, T2, T3, T4, T5, T6, T7 test cases were made in driver class as code and this output was taken.

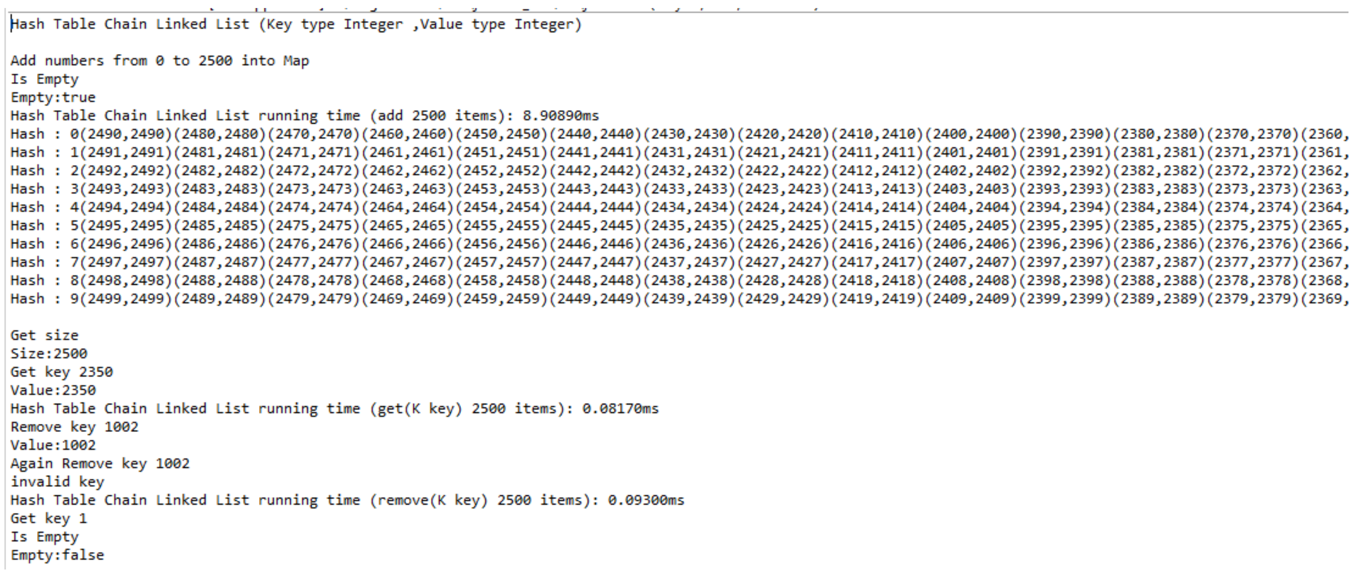
As you can see, all methods of the iterator are working. It is understood in the examples that the generic structure was established. I tried to show it using String and Integer.

When it is re-set according to the key, in the first example, it is sequenced again according to the "g" key. In the second example it is ordered according to the "10" key.

All tests were successful.

Part2:

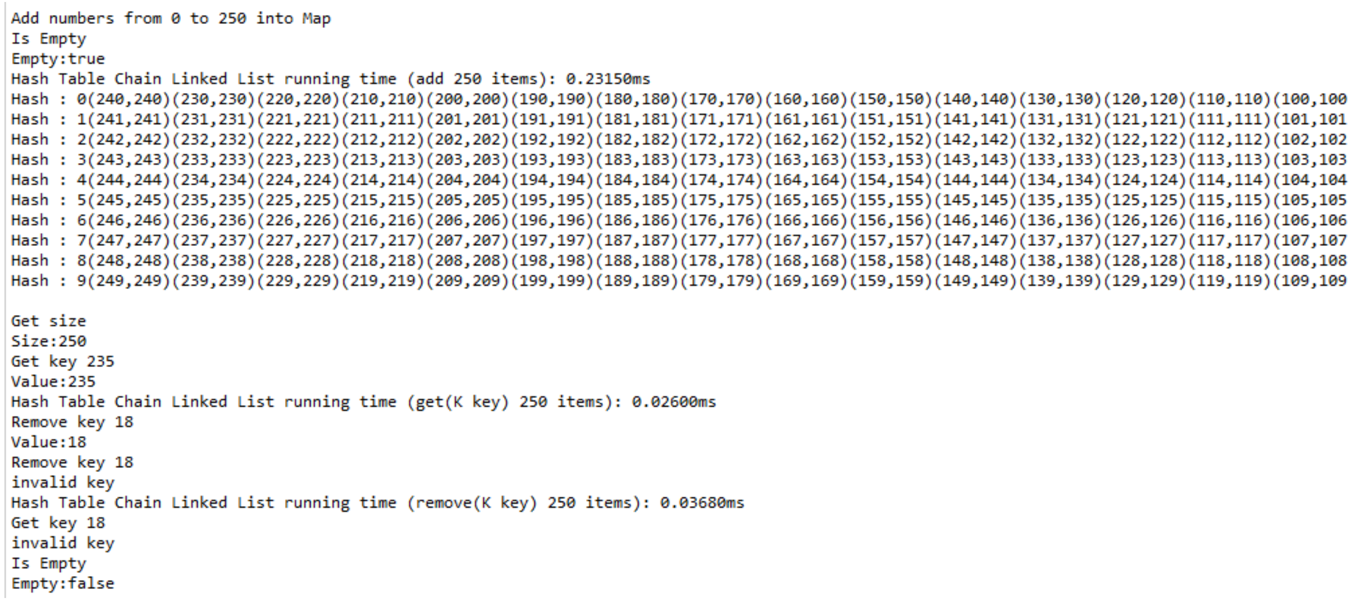
T1,T4,T5,T6,T7,T8,T9,T10,T11:



```
Hash Table Chain Linked List (Key type Integer ,Value type Integer)

Add numbers from 0 to 2500 into Map
Is Empty
Empty:true
Hash Table Chain Linked List running time (add 2500 items): 8.90890ms
Hash : 0(2490,2490)(2480,2480)(2470,2470)(2460,2460)(2450,2450)(2440,2440)(2430,2430)(2420,2420)(2410,2410)(2400,2400)(2390,2390)(2380,2380)(2370,2370)(2360,
Hash : 1(2491,2491)(2481,2481)(2471,2471)(2461,2461)(2451,2451)(2441,2441)(2431,2431)(2421,2421)(2411,2411)(2401,2401)(2391,2391)(2381,2381)(2371,2371)(2361,
Hash : 2(2492,2492)(2482,2482)(2472,2472)(2462,2462)(2452,2452)(2442,2442)(2432,2432)(2422,2422)(2412,2412)(2402,2402)(2392,2392)(2382,2382)(2372,2372)(2362,
Hash : 3(2493,2493)(2483,2483)(2473,2473)(2463,2463)(2453,2453)(2443,2443)(2433,2433)(2423,2423)(2413,2413)(2403,2403)(2393,2393)(2383,2383)(2373,2373)(2363,
Hash : 4(2494,2494)(2484,2484)(2474,2474)(2464,2464)(2454,2454)(2444,2444)(2434,2434)(2424,2424)(2414,2414)(2404,2404)(2394,2394)(2384,2384)(2374,2374)(2364,
Hash : 5(2495,2495)(2485,2485)(2475,2475)(2465,2465)(2455,2455)(2445,2445)(2435,2435)(2425,2425)(2415,2415)(2405,2405)(2395,2395)(2385,2385)(2375,2375)(2365,
Hash : 6(2496,2496)(2486,2486)(2476,2476)(2466,2466)(2456,2456)(2446,2446)(2436,2436)(2426,2426)(2416,2416)(2406,2406)(2396,2396)(2386,2386)(2376,2376)(2366,
Hash : 7(2497,2497)(2487,2487)(2477,2477)(2467,2467)(2457,2457)(2447,2447)(2437,2437)(2427,2427)(2417,2417)(2407,2407)(2397,2397)(2387,2387)(2377,2377)(2367,
Hash : 8(2498,2498)(2488,2488)(2478,2478)(2468,2468)(2458,2458)(2448,2448)(2438,2438)(2428,2428)(2418,2418)(2408,2408)(2398,2398)(2388,2388)(2378,2378)(2368,
Hash : 9(2499,2499)(2489,2489)(2479,2479)(2469,2469)(2459,2459)(2449,2449)(2439,2439)(2429,2429)(2419,2419)(2409,2409)(2399,2399)(2389,2389)(2379,2379)(2369,

Get size
Size:2500
Get key 2350
Value:2350
Hash Table Chain Linked List running time (get(K key) 2500 items): 0.08170ms
Remove key 1002
Value:1002
Again Remove key 1002
invalid key
Hash Table Chain Linked List running time (remove(K key) 2500 items): 0.09300ms
Get key 1
Is Empty
Empty:false
```

I show that I have done tests T1, T4, T5, T6, T7, T8, T9, T10, T11. In addition, I tested the performance of HashTableChainLinked functions. And I had the times printed. I was able to put a certain part of the map because the picture did not fit. But you can see all of them when you run print and run.

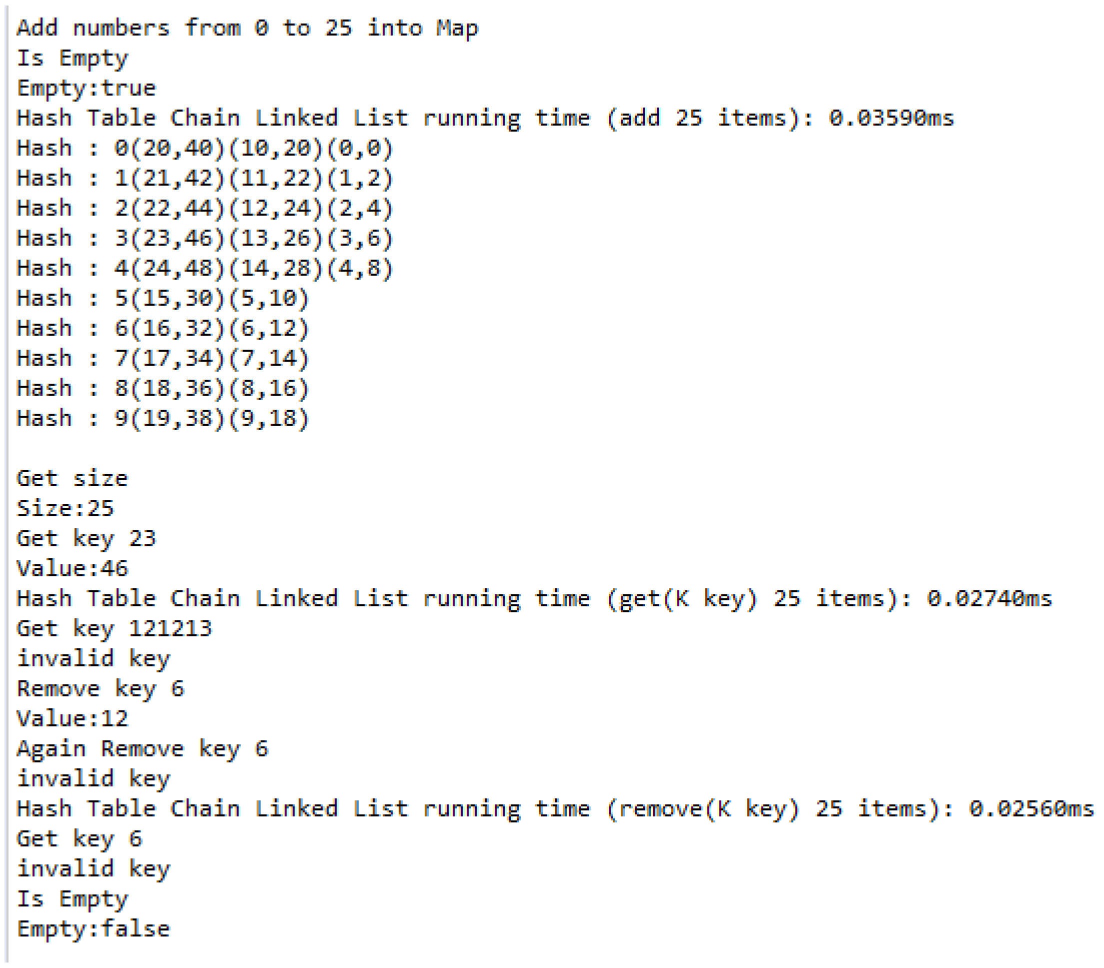
T2,T4,T5,T6,T7,T8,T9,T10,T11



```
Add numbers from 0 to 250 into Map
Is Empty
Empty:true
Hash Table Chain Linked List running time (add 250 items): 0.23150ms
Hash : 0(240,240)(230,230)(220,220)(210,210)(200,200)(190,190)(180,180)(170,170)(160,160)(150,150)(140,140)(130,130)(120,120)(110,110)(100,100
Hash : 1(241,241)(231,231)(221,221)(211,211)(201,201)(191,191)(181,181)(171,171)(161,161)(151,151)(141,141)(131,131)(121,121)(111,111)(101,101
Hash : 2(242,242)(232,232)(222,222)(212,212)(202,202)(192,192)(182,182)(172,172)(162,162)(152,152)(142,142)(132,132)(122,122)(112,112)(102,102
Hash : 3(243,243)(233,233)(223,223)(213,213)(203,203)(193,193)(183,183)(173,173)(163,163)(153,153)(143,143)(133,133)(123,123)(113,113)(103,103
Hash : 4(244,244)(234,234)(224,224)(214,214)(204,204)(194,194)(184,184)(174,174)(164,164)(154,154)(144,144)(134,134)(124,124)(114,114)(104,104
Hash : 5(245,245)(235,235)(225,225)(215,215)(205,205)(195,195)(185,185)(175,175)(165,165)(155,155)(145,145)(135,135)(125,125)(115,115)(105,105
Hash : 6(246,246)(236,236)(226,226)(216,216)(206,206)(196,196)(186,186)(176,176)(166,166)(156,156)(146,146)(136,136)(126,126)(116,116)(106,106
Hash : 7(247,247)(237,237)(227,227)(217,217)(207,207)(197,197)(187,187)(177,177)(167,167)(157,157)(147,147)(137,137)(127,127)(117,117)(107,107
Hash : 8(248,248)(238,238)(228,228)(218,218)(208,208)(198,198)(188,188)(178,178)(168,168)(158,158)(148,148)(138,138)(128,128)(118,118)(108,108
Hash : 9(249,249)(239,239)(229,229)(219,219)(209,209)(199,199)(189,189)(179,179)(169,169)(159,159)(149,149)(139,139)(129,129)(119,119)(109,109

Get size
Size:250
Get key 235
Value:235
Hash Table Chain Linked List running time (get(K key) 250 items): 0.02600ms
Remove key 18
Value:18
Remove key 18
invalid key
Hash Table Chain Linked List running time (remove(K key) 250 items): 0.03680ms
Get key 18
invalid key
Is Empty
Empty:false
```

I show that I have done tests T2, T4, T5, T6, T7, T8, T9, T10, T11. In addition, I tested the performance of HashTableChainLinked functions. And I had the times printed. I was able to put a certain part of the map because the picture did not fit. But you can see all of them when you run print and run.

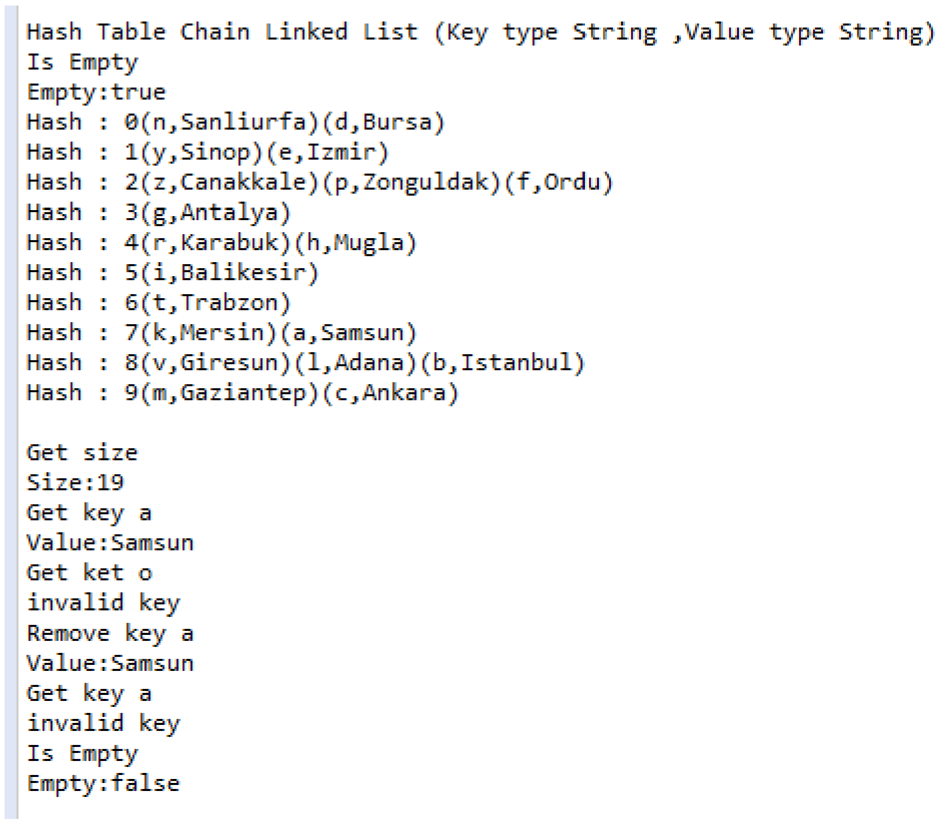
T3,T4,T5,T6,T7,T8,T9,T10,T11



```
Add numbers from 0 to 25 into Map
Is Empty
Empty:true
Hash Table Chain Linked List running time (add 25 items): 0.03590ms
Hash : 0(20,40)(10,20)(0,0)
Hash : 1(21,42)(11,22)(1,2)
Hash : 2(22,44)(12,24)(2,4)
Hash : 3(23,46)(13,26)(3,6)
Hash : 4(24,48)(14,28)(4,8)
Hash : 5(15,30)(5,10)
Hash : 6(16,32)(6,12)
Hash : 7(17,34)(7,14)
Hash : 8(18,36)(8,16)
Hash : 9(19,38)(9,18)

Get size
Size:25
Get key 23
Value:46
Hash Table Chain Linked List running time (get(K key) 25 items): 0.02740ms
Get key 121213
invalid key
Remove key 6
Value:12
Again Remove key 6
invalid key
Hash Table Chain Linked List running time (remove(K key) 25 items): 0.02560ms
Get key 6
invalid key
Is Empty
Empty:false
```

I show that I have done tests T3, T4, T5, T6, T7, T8, T9, T10, T11. In addition, I tested the performance of HashTableChainLinked functions. And I had the times printed.

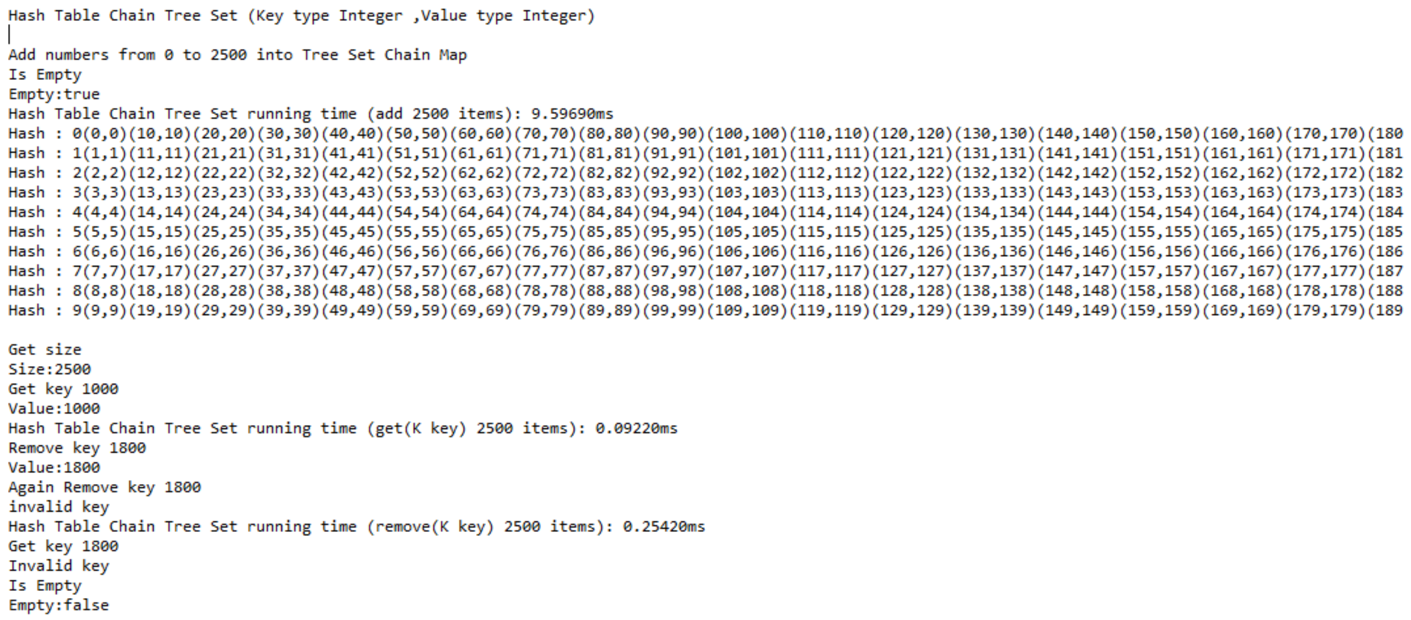
T12,T13,T14,T15,T16,T17,T18,T19



```
Hash Table Chain Linked List (Key type String ,Value type String)
Is Empty
Empty:true
Hash : 0(n,Sanliurfa)(d,Bursa)
Hash : 1(y,Sinop)(e,Izmir)
Hash : 2(z,Canakkale)(p,Zonguldak)(f,Ordu)
Hash : 3(g,Antalya)
Hash : 4(r,Karabuk)(h,Mugla)
Hash : 5(i,Balikesir)
Hash : 6(t,Trabzon)
Hash : 7(k,Mersin)(a,Samsun)
Hash : 8(v,Giresun)(l,Adana)(b,Istanbul)
Hash : 9(m,Gaziantep)(c,Ankara)

Get size
Size:19
Get key a
Value:Samsun
Get ket o
invalid key
Remove key a
Value:Samsun
Get key a
invalid key
Is Empty
Empty:false
```

I show that I have done tests T12, T13, T14, T15, T16, T17, T18, T19.

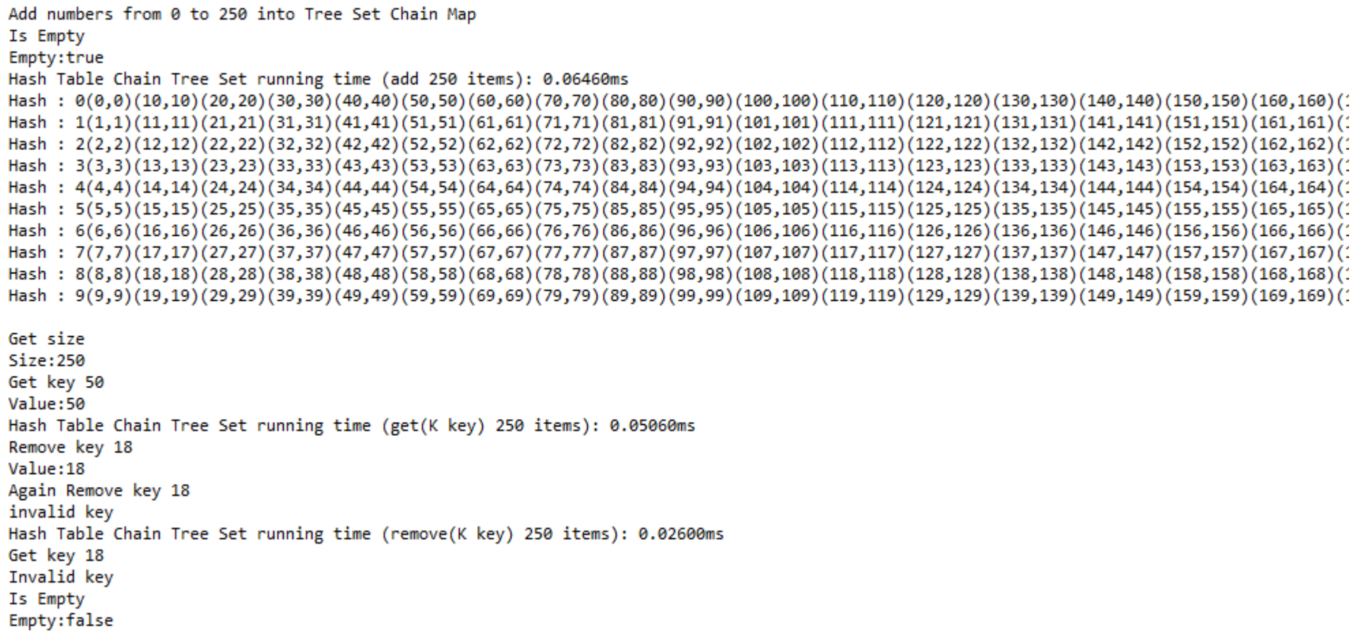
T20,T23,T24,T25,T26,T27,T28,T29,T30



```
Hash Table Chain Tree Set (Key type Integer ,Value type Integer)
|
Add numbers from 0 to 2500 into Tree Set Chain Map
Is Empty
Empty:true
Hash Table Chain Tree Set running time (add 2500 items): 9.59690ms
Hash : 0(0,0)(10,10)(20,20)(30,30)(40,40)(50,50)(60,60)(70,70)(80,80)(90,90)(100,100)(110,110)(120,120)(130,130)(140,140)(150,150)(160,160)(170,170)(180
Hash : 1(1,1)(11,11)(21,21)(31,31)(41,41)(51,51)(61,61)(71,71)(81,81)(91,91)(101,101)(111,111)(121,121)(131,131)(141,141)(151,151)(161,161)(171,171)(181
Hash : 2(2,2)(12,12)(22,22)(32,32)(42,42)(52,52)(62,62)(72,72)(82,82)(92,92)(102,102)(112,112)(122,122)(132,132)(142,142)(152,152)(162,162)(172,172)(182
Hash : 3(3,3)(13,13)(23,23)(33,33)(43,43)(53,53)(63,63)(73,73)(83,83)(93,93)(103,103)(113,113)(123,123)(133,133)(143,143)(153,153)(163,163)(173,173)(183
Hash : 4(4,4)(14,14)(24,24)(34,34)(44,44)(54,54)(64,64)(74,74)(84,84)(94,94)(104,104)(114,114)(124,124)(134,134)(144,144)(154,154)(164,164)(174,174)(184
Hash : 5(5,5)(15,15)(25,25)(35,35)(45,45)(55,55)(65,65)(75,75)(85,85)(95,95)(105,105)(115,115)(125,125)(135,135)(145,145)(155,155)(165,165)(175,175)(185
Hash : 6(6,6)(16,16)(26,26)(36,36)(46,46)(56,56)(66,66)(76,76)(86,86)(96,96)(106,106)(116,116)(126,126)(136,136)(146,146)(156,156)(166,166)(176,176)(186
Hash : 7(7,7)(17,17)(27,27)(37,37)(47,47)(57,57)(67,67)(77,77)(87,87)(97,97)(107,107)(117,117)(127,127)(137,137)(147,147)(157,157)(167,167)(177,177)(187
Hash : 8(8,8)(18,18)(28,28)(38,38)(48,48)(58,58)(68,68)(78,78)(88,88)(98,98)(108,108)(118,118)(128,128)(138,138)(148,148)(158,158)(168,168)(178,178)(188
Hash : 9(9,9)(19,19)(29,29)(39,39)(49,49)(59,59)(69,69)(79,79)(89,89)(99,99)(109,109)(119,119)(129,129)(139,139)(149,149)(159,159)(169,169)(179,179)(189

Get size
Size:2500
Get key 1000
Value:1000
Hash Table Chain Tree Set running time (get(K key) 2500 items): 0.09220ms
Remove key 1800
Value:1800
Again Remove key 1800
invalid key
Hash Table Chain Tree Set running time (remove(K key) 2500 items): 0.25420ms
Get key 1800
Invalid key
Is Empty
Empty:false
```

I show that I have done tests T20, T23, T24, T25, T26, T27, T28, T29, T30. In addition, I tested the performance of HashTableTreeSet functions. And I had the times printed. I was able to put a certain part of the map because the picture did not fit. But you can see all of them when you run print and run.

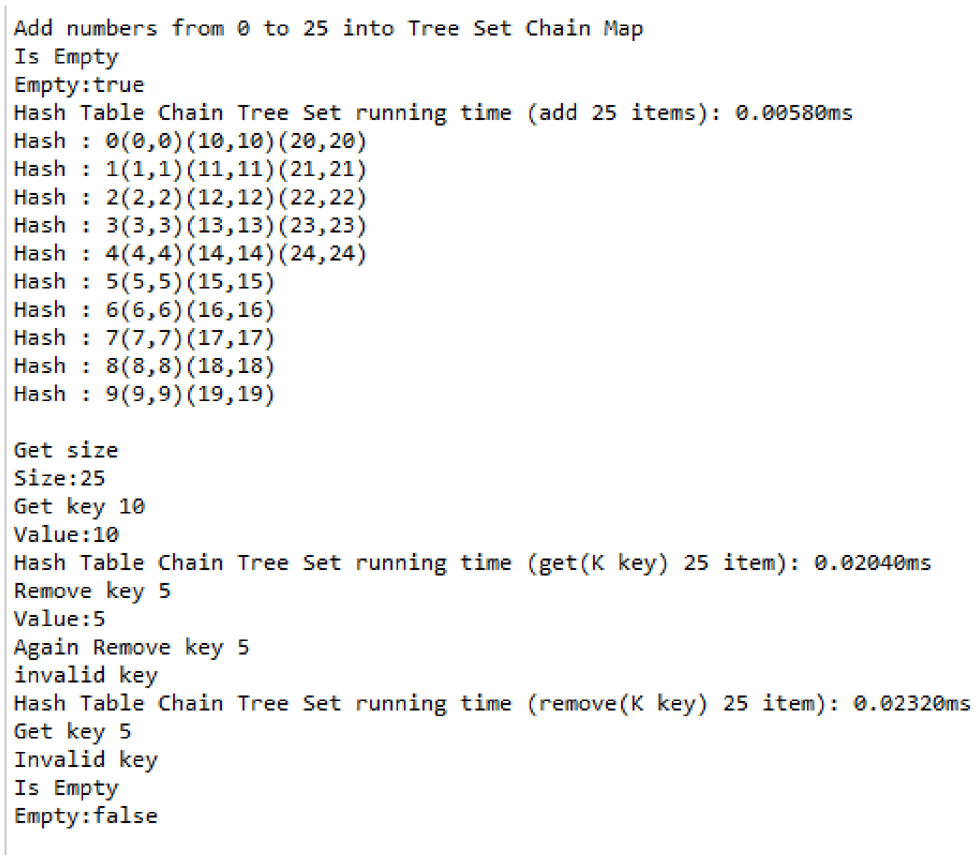
T21,T23,T24,T25,T26,T27,T28,T29,T30



```
Add numbers from 0 to 250 into Tree Set Chain Map
Is Empty
Empty:true
Hash Table Chain Tree Set running time (add 250 items): 0.06460ms
Hash : 0(0,0)(10,10)(20,20)(30,30)(40,40)(50,50)(60,60)(70,70)(80,80)(90,90)(100,100)(110,110)(120,120)(130,130)(140,140)(150,150)(160,160)(
Hash : 1(1,1)(11,11)(21,21)(31,31)(41,41)(51,51)(61,61)(71,71)(81,81)(91,91)(101,101)(111,111)(121,121)(131,131)(141,141)(151,151)(161,161)(
Hash : 2(2,2)(12,12)(22,22)(32,32)(42,42)(52,52)(62,62)(72,72)(82,82)(92,92)(102,102)(112,112)(122,122)(132,132)(142,142)(152,152)(162,162)(
Hash : 3(3,3)(13,13)(23,23)(33,33)(43,43)(53,53)(63,63)(73,73)(83,83)(93,93)(103,103)(113,113)(123,123)(133,133)(143,143)(153,153)(163,163)(
Hash : 4(4,4)(14,14)(24,24)(34,34)(44,44)(54,54)(64,64)(74,74)(84,84)(94,94)(104,104)(114,114)(124,124)(134,134)(144,144)(154,154)(164,164)(
Hash : 5(5,5)(15,15)(25,25)(35,35)(45,45)(55,55)(65,65)(75,75)(85,85)(95,95)(105,105)(115,115)(125,125)(135,135)(145,145)(155,155)(165,165)(
Hash : 6(6,6)(16,16)(26,26)(36,36)(46,46)(56,56)(66,66)(76,76)(86,86)(96,96)(106,106)(116,116)(126,126)(136,136)(146,146)(156,156)(166,166)(
Hash : 7(7,7)(17,17)(27,27)(37,37)(47,47)(57,57)(67,67)(77,77)(87,87)(97,97)(107,107)(117,117)(127,127)(137,137)(147,147)(157,157)(167,167)(
Hash : 8(8,8)(18,18)(28,28)(38,38)(48,48)(58,58)(68,68)(78,78)(88,88)(98,98)(108,108)(118,118)(128,128)(138,138)(148,148)(158,158)(168,168)(
Hash : 9(9,9)(19,19)(29,29)(39,39)(49,49)(59,59)(69,69)(79,79)(89,89)(99,99)(109,109)(119,119)(129,129)(139,139)(149,149)(159,159)(169,169)(

Get size
Size:250
Get key 50
Value:50
Hash Table Chain Tree Set running time (get(K key) 250 items): 0.05060ms
Remove key 18
Value:18
Again Remove key 18
invalid key
Hash Table Chain Tree Set running time (remove(K key) 250 items): 0.02600ms
Get key 18
Invalid key
Is Empty
Empty:false
```

I show that I have done tests T21, T23, T24, T25, T26, T27, T28, T29, T30. In addition, I tested the performance of HashTableTreeSet functions. And I had the times printed. I was able to put a certain part of the map because the picture did not fit. But you can see all of them when you run print and run.

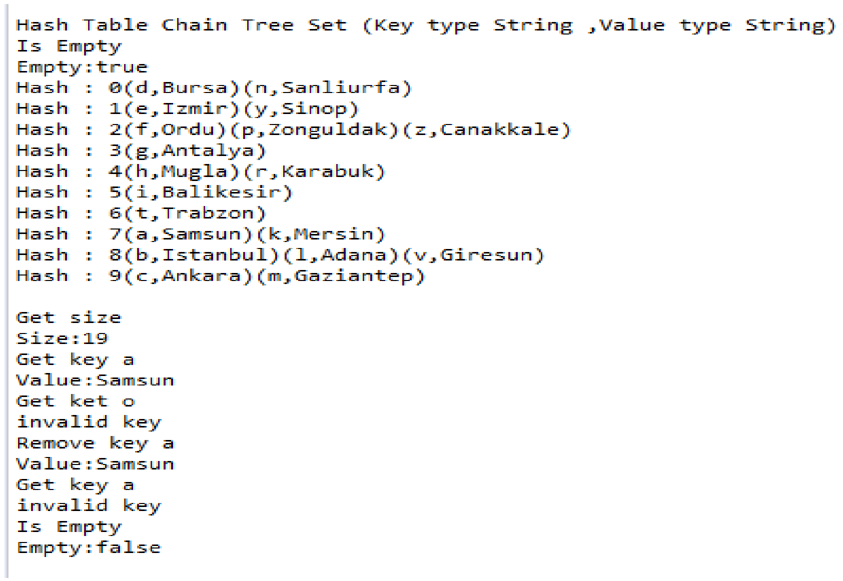
T22,T23,T24,T25,T26,T27,T28,T29,T30



```
Add numbers from 0 to 25 into Tree Set Chain Map
Is Empty
Empty:true
Hash Table Chain Tree Set running time (add 25 items): 0.00580ms
Hash : 0(0,0)(10,10)(20,20)
Hash : 1(1,1)(11,11)(21,21)
Hash : 2(2,2)(12,12)(22,22)
Hash : 3(3,3)(13,13)(23,23)
Hash : 4(4,4)(14,14)(24,24)
Hash : 5(5,5)(15,15)
Hash : 6(6,6)(16,16)
Hash : 7(7,7)(17,17)
Hash : 8(8,8)(18,18)
Hash : 9(9,9)(19,19)

Get size
Size:25
Get key 10
Value:10
Hash Table Chain Tree Set running time (get(K key) 25 item): 0.02040ms
Remove key 5
Value:5
Again Remove key 5
invalid key
Hash Table Chain Tree Set running time (remove(K key) 25 item): 0.02320ms
Get key 5
Invalid key
Is Empty
Empty:false
```

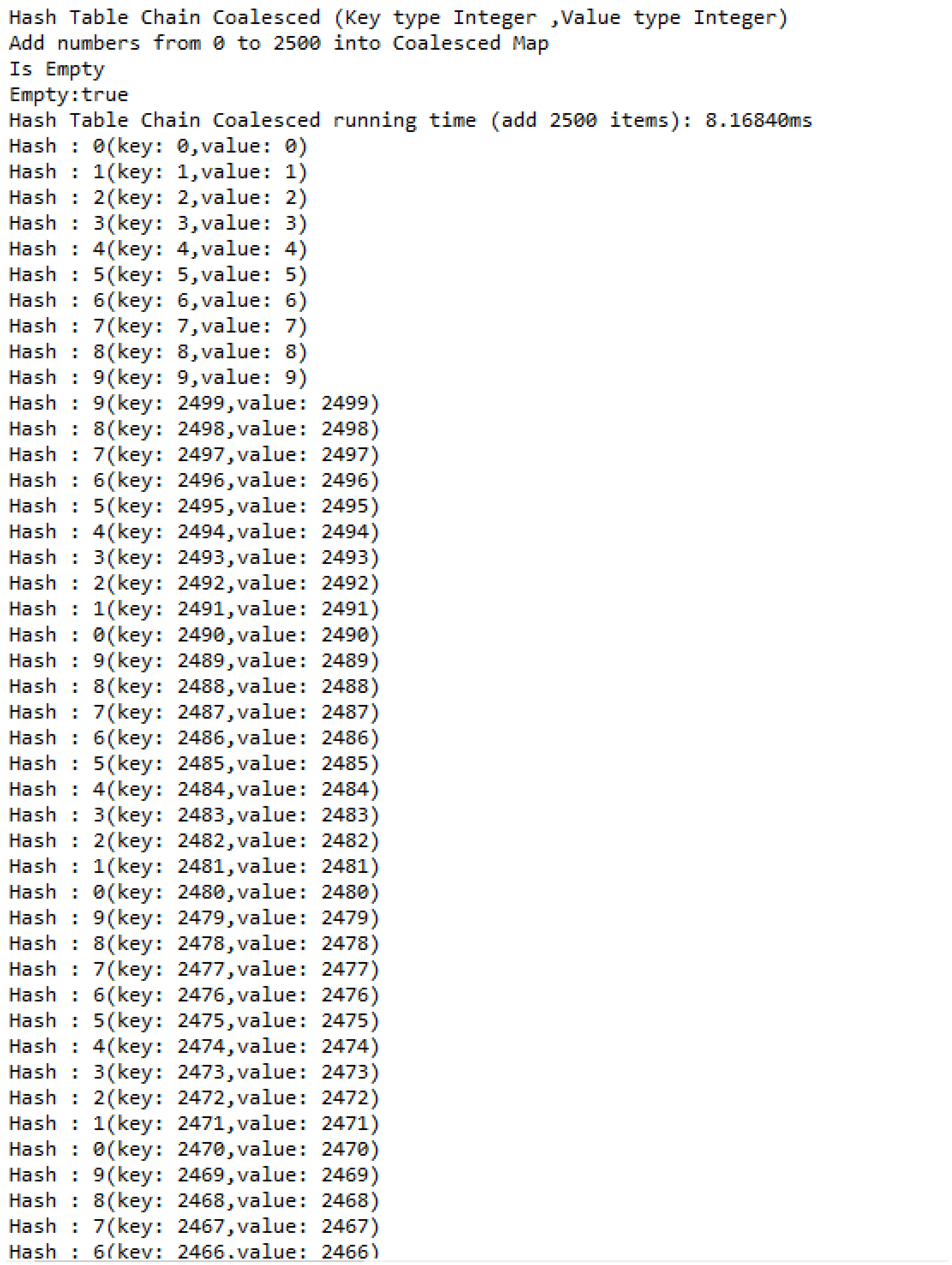
I show that I have done tests T22, T23, T24, T25, T26, T27, T28, T29, T30. In addition, I tested the performance of HashTableTreeSet functions. And I had the times printed.

T31,T32,T33,T34,T35,T36,T37,T38

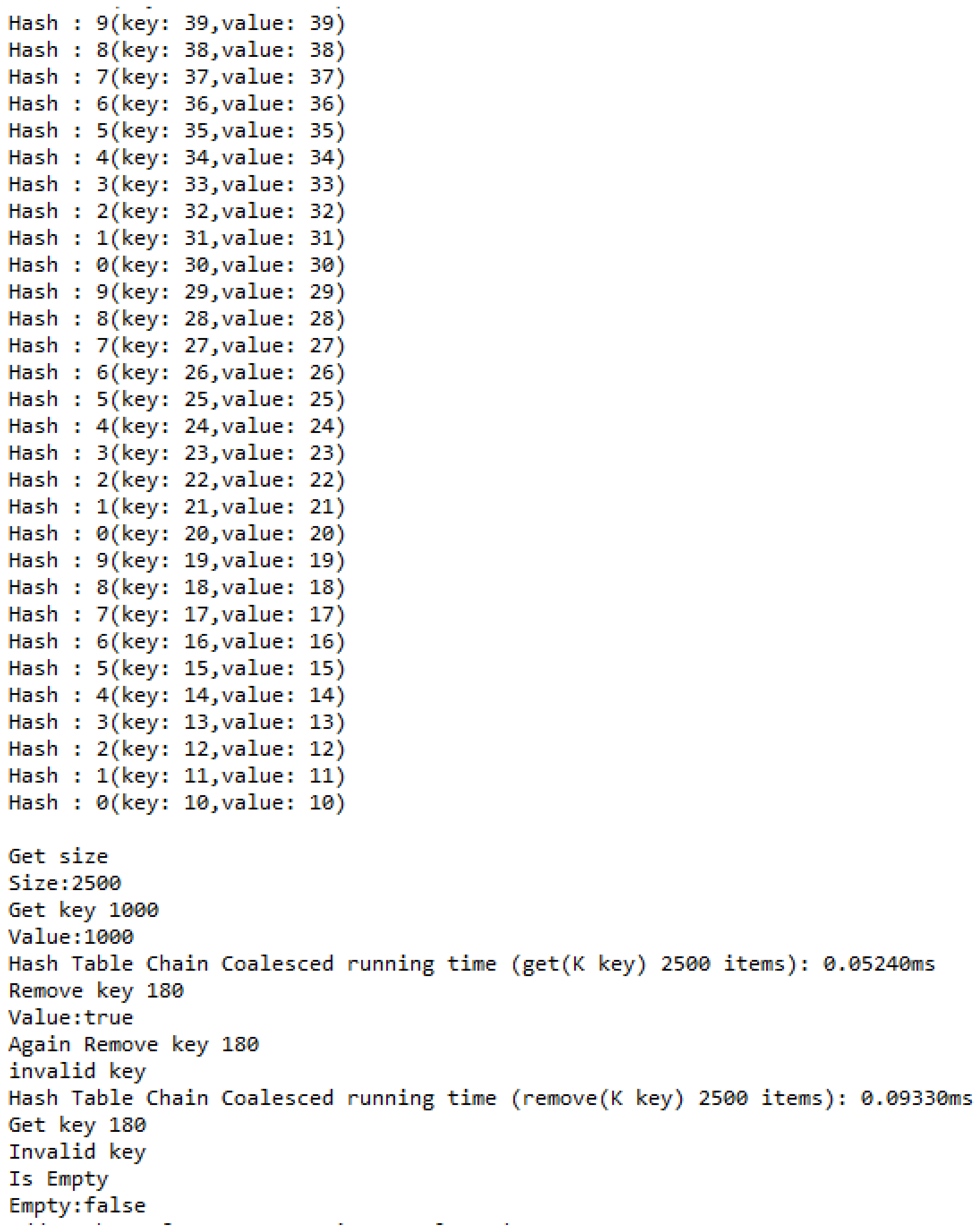


```
Hash Table Chain Tree Set (Key type String ,Value type String)
Is Empty
Empty:true
Hash : 0(d,Bursa)(n,Sanliurfa)
Hash : 1(e,Izmir)(y,Sinop)
Hash : 2(f,Ordu)(p,Zonguldak)(z,Canakkale)
Hash : 3(g,Antalya)
Hash : 4(h,Mugla)(r,Karabuk)
Hash : 5(i,Balikesir)
Hash : 6(t,Trabzon)
Hash : 7(a,Samsun)(k,Mersin)
Hash : 8(b,Istanbul)(l,Adana)(v,Giresun)
Hash : 9(c,Ankara)(m,Gaziantep)

Get size
Size:19
Get key a
Value:Samsun
Get ket o
invalid key
Remove key a
Value:Samsun
Get key a
invalid key
Is Empty
Empty:false
```

T39,T42,T43,T44,T45,T46,T47,T48,T49



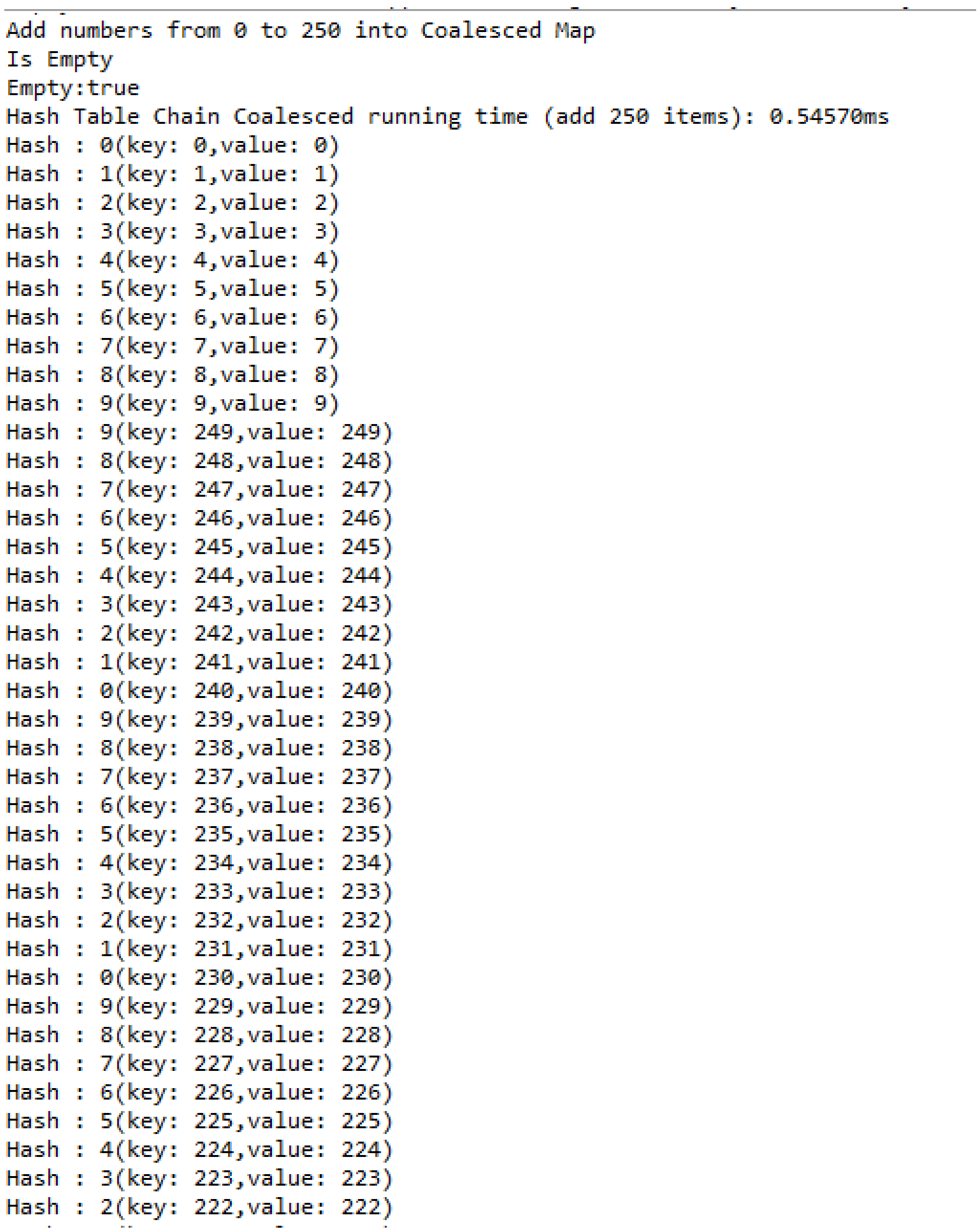
```
Hash Table Chain Coalesced (Key type Integer ,Value type Integer)
Add numbers from 0 to 2500 into Coalesced Map
Is Empty
Empty:true
Hash Table Chain Coalesced running time (add 2500 items): 8.16840ms
Hash : 0(key: 0,value: 0)
Hash : 1(key: 1,value: 1)
Hash : 2(key: 2,value: 2)
Hash : 3(key: 3,value: 3)
Hash : 4(key: 4,value: 4)
Hash : 5(key: 5,value: 5)
Hash : 6(key: 6,value: 6)
Hash : 7(key: 7,value: 7)
Hash : 8(key: 8,value: 8)
Hash : 9(key: 9,value: 9)
Hash : 9(key: 2499,value: 2499)
Hash : 8(key: 2498,value: 2498)
Hash : 7(key: 2497,value: 2497)
Hash : 6(key: 2496,value: 2496)
Hash : 5(key: 2495,value: 2495)
Hash : 4(key: 2494,value: 2494)
Hash : 3(key: 2493,value: 2493)
Hash : 2(key: 2492,value: 2492)
Hash : 1(key: 2491,value: 2491)
Hash : 0(key: 2490,value: 2490)
Hash : 9(key: 2489,value: 2489)
Hash : 8(key: 2488,value: 2488)
Hash : 7(key: 2487,value: 2487)
Hash : 6(key: 2486,value: 2486)
Hash : 5(key: 2485,value: 2485)
Hash : 4(key: 2484,value: 2484)
Hash : 3(key: 2483,value: 2483)
Hash : 2(key: 2482,value: 2482)
Hash : 1(key: 2481,value: 2481)
Hash : 0(key: 2480,value: 2480)
Hash : 9(key: 2479,value: 2479)
Hash : 8(key: 2478,value: 2478)
Hash : 7(key: 2477,value: 2477)
Hash : 6(key: 2476,value: 2476)
Hash : 5(key: 2475,value: 2475)
Hash : 4(key: 2474,value: 2474)
Hash : 3(key: 2473,value: 2473)
Hash : 2(key: 2472,value: 2472)
Hash : 1(key: 2471,value: 2471)
Hash : 0(key: 2470,value: 2470)
Hash : 9(key: 2469,value: 2469)
Hash : 8(key: 2468,value: 2468)
Hash : 7(key: 2467,value: 2467)
Hash : 6(key: 2466,value: 2466)
```



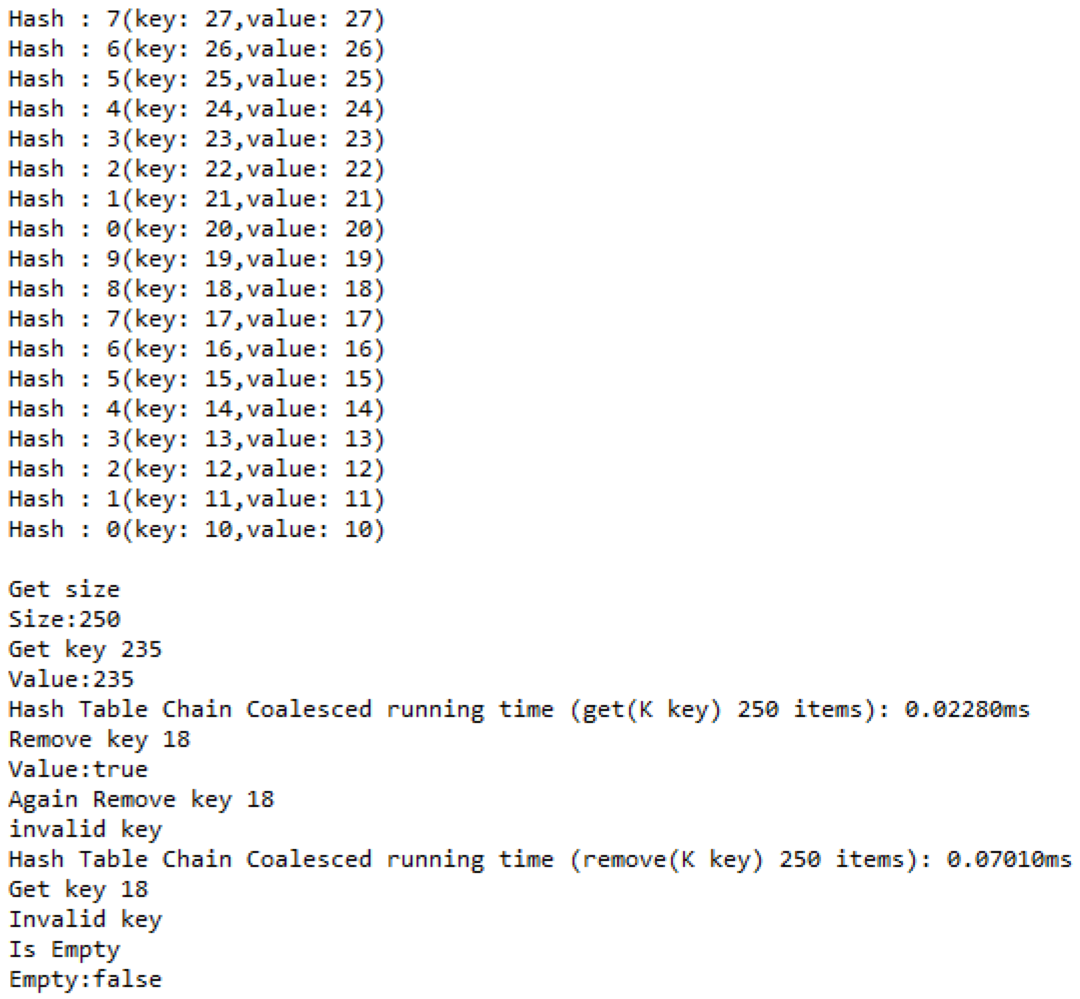
```
Hash : 9(key: 39,value: 39)
Hash : 8(key: 38,value: 38)
Hash : 7(key: 37,value: 37)
Hash : 6(key: 36,value: 36)
Hash : 5(key: 35,value: 35)
Hash : 4(key: 34,value: 34)
Hash : 3(key: 33,value: 33)
Hash : 2(key: 32,value: 32)
Hash : 1(key: 31,value: 31)
Hash : 0(key: 30,value: 30)
Hash : 9(key: 29,value: 29)
Hash : 8(key: 28,value: 28)
Hash : 7(key: 27,value: 27)
Hash : 6(key: 26,value: 26)
Hash : 5(key: 25,value: 25)
Hash : 4(key: 24,value: 24)
Hash : 3(key: 23,value: 23)
Hash : 2(key: 22,value: 22)
Hash : 1(key: 21,value: 21)
Hash : 0(key: 20,value: 20)
Hash : 9(key: 19,value: 19)
Hash : 8(key: 18,value: 18)
Hash : 7(key: 17,value: 17)
Hash : 6(key: 16,value: 16)
Hash : 5(key: 15,value: 15)
Hash : 4(key: 14,value: 14)
Hash : 3(key: 13,value: 13)
Hash : 2(key: 12,value: 12)
Hash : 1(key: 11,value: 11)
Hash : 0(key: 10,value: 10)

Get size
Size:2500
Get key 1000
Value:1000
Hash Table Chain Coalesced running time (get(K key) 2500 items): 0.05240ms
Remove key 180
Value:true
Again Remove key 180
invalid key
Hash Table Chain Coalesced running time (remove(K key) 2500 items): 0.09330ms
Get key 180
Invalid key
Is Empty
Empty:false
```

I show that I have done tests T39, T42, T43, T44, T45, T46, T47, T48, T49. In addition, I tested the performance of HashTableChainLinked functions. And I had the times printed. I was able to put a certain part of the map because the picture did not fit. But you can see all of them when you run print and run.

T40,T42,T43,T44,T45,T46,T47,T48,T49



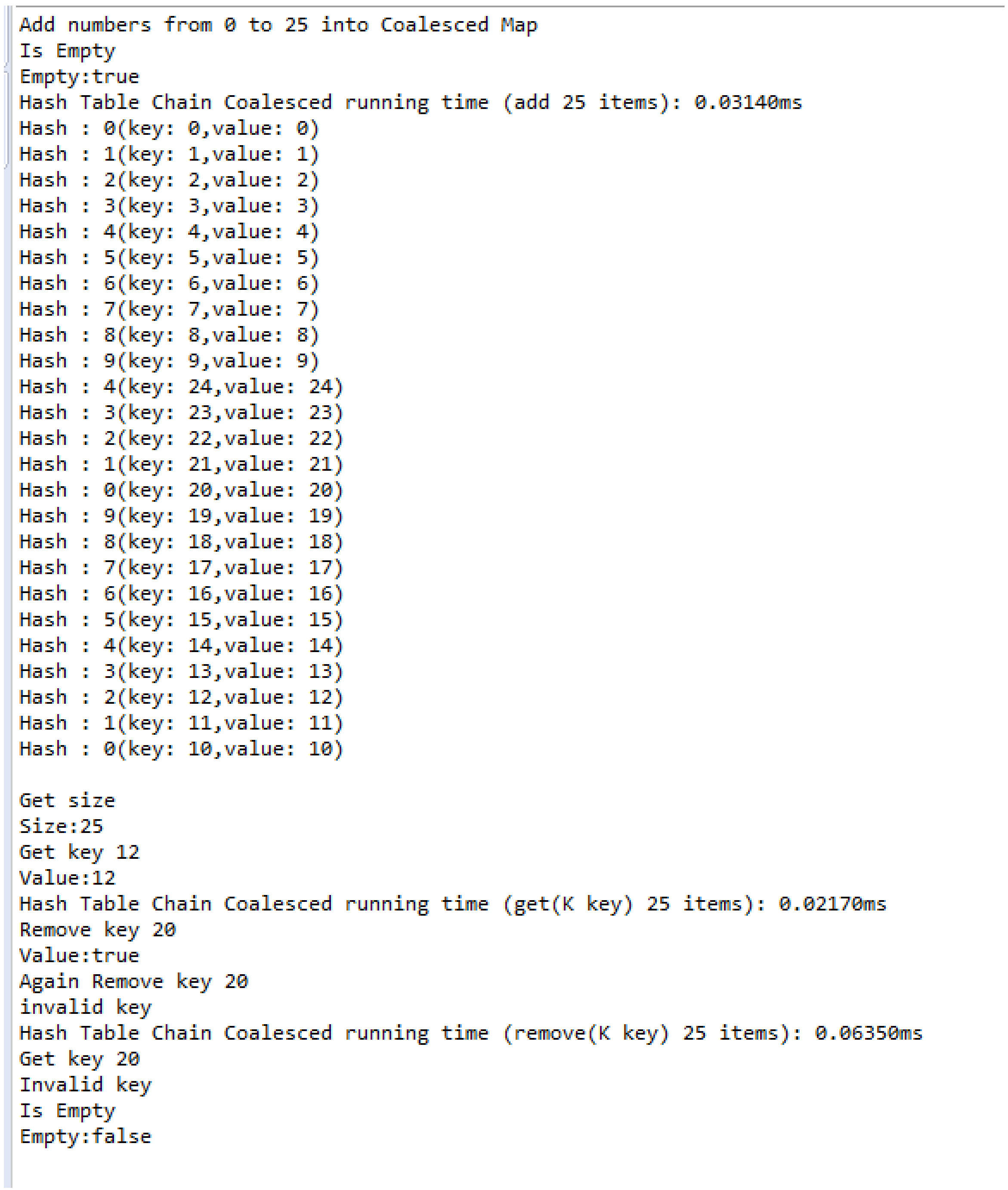
```
Add numbers from 0 to 250 into Coalesced Map
Is Empty
Empty:true
Hash Table Chain Coalesced running time (add 250 items): 0.54570ms
Hash : 0(key: 0,value: 0)
Hash : 1(key: 1,value: 1)
Hash : 2(key: 2,value: 2)
Hash : 3(key: 3,value: 3)
Hash : 4(key: 4,value: 4)
Hash : 5(key: 5,value: 5)
Hash : 6(key: 6,value: 6)
Hash : 7(key: 7,value: 7)
Hash : 8(key: 8,value: 8)
Hash : 9(key: 9,value: 9)
Hash : 9(key: 249,value: 249)
Hash : 8(key: 248,value: 248)
Hash : 7(key: 247,value: 247)
Hash : 6(key: 246,value: 246)
Hash : 5(key: 245,value: 245)
Hash : 4(key: 244,value: 244)
Hash : 3(key: 243,value: 243)
Hash : 2(key: 242,value: 242)
Hash : 1(key: 241,value: 241)
Hash : 0(key: 240,value: 240)
Hash : 9(key: 239,value: 239)
Hash : 8(key: 238,value: 238)
Hash : 7(key: 237,value: 237)
Hash : 6(key: 236,value: 236)
Hash : 5(key: 235,value: 235)
Hash : 4(key: 234,value: 234)
Hash : 3(key: 233,value: 233)
Hash : 2(key: 232,value: 232)
Hash : 1(key: 231,value: 231)
Hash : 0(key: 230,value: 230)
Hash : 9(key: 229,value: 229)
Hash : 8(key: 228,value: 228)
Hash : 7(key: 227,value: 227)
Hash : 6(key: 226,value: 226)
Hash : 5(key: 225,value: 225)
Hash : 4(key: 224,value: 224)
Hash : 3(key: 223,value: 223)
Hash : 2(key: 222,value: 222)
```



```
Hash : 7(key: 27,value: 27)
Hash : 6(key: 26,value: 26)
Hash : 5(key: 25,value: 25)
Hash : 4(key: 24,value: 24)
Hash : 3(key: 23,value: 23)
Hash : 2(key: 22,value: 22)
Hash : 1(key: 21,value: 21)
Hash : 0(key: 20,value: 20)
Hash : 9(key: 19,value: 19)
Hash : 8(key: 18,value: 18)
Hash : 7(key: 17,value: 17)
Hash : 6(key: 16,value: 16)
Hash : 5(key: 15,value: 15)
Hash : 4(key: 14,value: 14)
Hash : 3(key: 13,value: 13)
Hash : 2(key: 12,value: 12)
Hash : 1(key: 11,value: 11)
Hash : 0(key: 10,value: 10)

Get size
Size:250
Get key 235
Value:235
Hash Table Chain Coalesced running time (get(K key) 250 items): 0.02280ms
Remove key 18
Value:true
Again Remove key 18
invalid key
Hash Table Chain Coalesced running time (remove(K key) 250 items): 0.07010ms
Get key 18
Invalid key
Is Empty
Empty:false
```

I show that I have done tests T40, T42, T43, T44, T45, T46, T47, T48, T49. In addition, I tested the performance of HashTableChainLinked functions. And I had the times printed. I was able to put a certain part of the map because the picture did not fit. But you can see all of them when you run them..

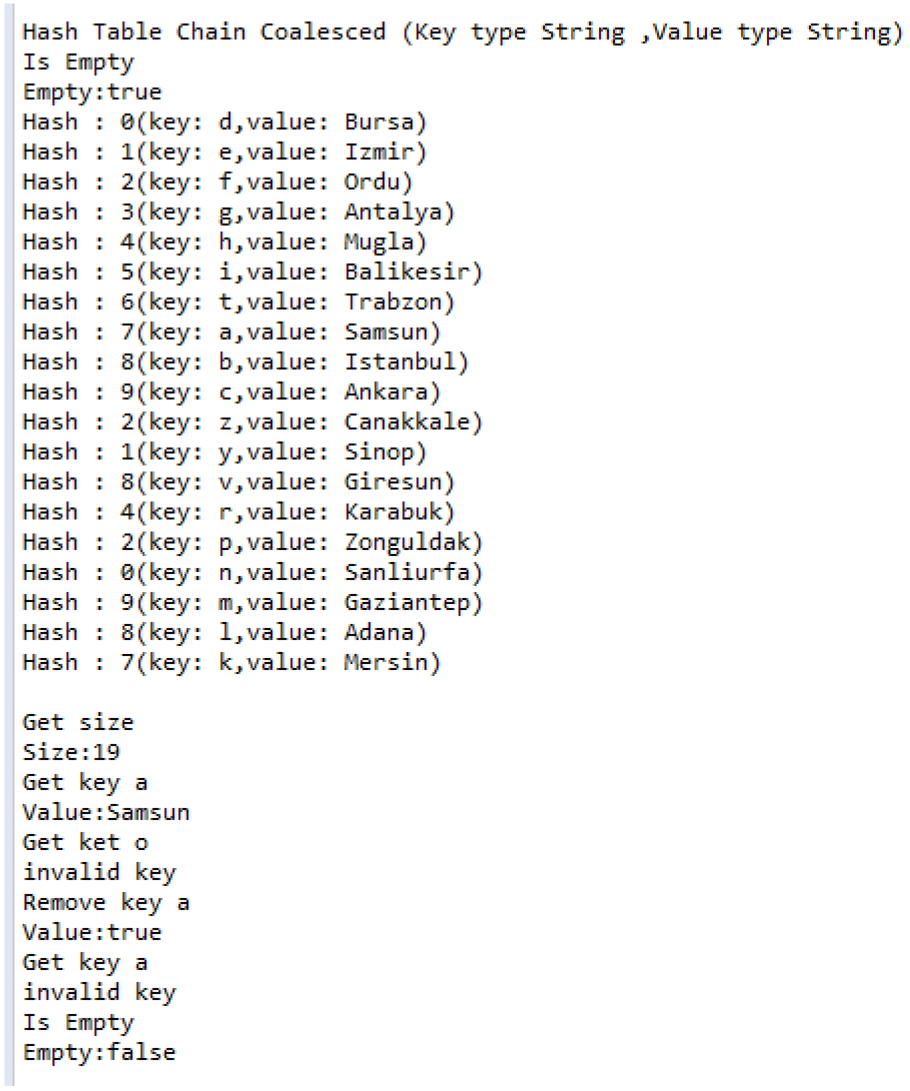
T41,T42,T43,T44,T45,T46,T47,T48,T49



```
Add numbers from 0 to 25 into Coalesced Map
Is Empty
Empty:true
Hash Table Chain Coalesced running time (add 25 items): 0.03140ms
Hash : 0(key: 0,value: 0)
Hash : 1(key: 1,value: 1)
Hash : 2(key: 2,value: 2)
Hash : 3(key: 3,value: 3)
Hash : 4(key: 4,value: 4)
Hash : 5(key: 5,value: 5)
Hash : 6(key: 6,value: 6)
Hash : 7(key: 7,value: 7)
Hash : 8(key: 8,value: 8)
Hash : 9(key: 9,value: 9)
Hash : 4(key: 24,value: 24)
Hash : 3(key: 23,value: 23)
Hash : 2(key: 22,value: 22)
Hash : 1(key: 21,value: 21)
Hash : 0(key: 20,value: 20)
Hash : 9(key: 19,value: 19)
Hash : 8(key: 18,value: 18)
Hash : 7(key: 17,value: 17)
Hash : 6(key: 16,value: 16)
Hash : 5(key: 15,value: 15)
Hash : 4(key: 14,value: 14)
Hash : 3(key: 13,value: 13)
Hash : 2(key: 12,value: 12)
Hash : 1(key: 11,value: 11)
Hash : 0(key: 10,value: 10)

Get size
Size:25
Get key 12
Value:12
Hash Table Chain Coalesced running time (get(K key) 25 items): 0.02170ms
Remove key 20
Value:true
Again Remove key 20
invalid key
Hash Table Chain Coalesced running time (remove(K key) 25 items): 0.06350ms
Get key 20
Invalid key
Is Empty
Empty:false
```

I show that I have done tests T41, T42, T43, T44, T45, T46, T47, T48, T49. In addition, I tested the performance of HashTableChainLinked functions. And I had the times printed.

T50,T51,T52,T53,T54,T55,T56,T57



```
Hash Table Chain Coalesced (Key type String ,Value type String)
Is Empty
Empty:true
Hash : 0(key: d,value: Bursa)
Hash : 1(key: e,value: Izmir)
Hash : 2(key: f,value: Ordu)
Hash : 3(key: g,value: Antalya)
Hash : 4(key: h,value: Mugla)
Hash : 5(key: i,value: Balikesir)
Hash : 6(key: t,value: Trabzon)
Hash : 7(key: a,value: Samsun)
Hash : 8(key: b,value: Istanbul)
Hash : 9(key: c,value: Ankara)
Hash : 2(key: z,value: Canakkale)
Hash : 1(key: y,value: Sinop)
Hash : 8(key: v,value: Giresun)
Hash : 4(key: r,value: Karabuk)
Hash : 2(key: p,value: Zonguldak)
Hash : 0(key: n,value: Sanliurfa)
Hash : 9(key: m,value: Gaziantep)
Hash : 8(key: l,value: Adana)
Hash : 7(key: k,value: Mersin)

Get size
Size:19
Get key a
Value:Samsun
Get ket o
invalid key
Remove key a
Value:true
Get key a
invalid key
Is Empty
Empty:false
```