



**T.R.  
GEBZE TECHNICAL UNIVERSITY  
FACULTY of ENGINEERING  
DEPARTMENT of COMPUTER ENGINEERING**

## **MAP ITERATOR**

### **CSE 222 DATA STRUCTURES AND ALGORITHMS HOMEWORK 5 – PART 1REPORT**

**STUDENT**  
**Şeyda Nur DEMİR**  
**12 10 44 042**

**LECTURER**  
**Prof. Dr. Fatih Erdoğan SEVİLGİN**

**TEACHING ASSISTANT**  
**Başak KARAKAŞ**  
**Mehmet Burak KOCA**

**KOCAELİ, 2021**

# 1.DESCRPTION

This homework has 2 parts.

## Part 1 : 40 pts

Write a custom iterator class MapIterator to iterate through the keys in a HashMap data structure in Java. This class should have the following methods:

- next(): The function returns the next key in the Map. It returns the first key when there is no not-iterated key in the Map.
- prev(): The iterator points to the previous key in the Map. 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 iterator should start from the given key and still 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 (zero parameter constructor).

## Part 2 : 60 pts

Implement KWHashMap interface in the book using the following hashing methods to organize hash table:

- Use the chaining technique for hashing by using linked lists (available in the text book) to chain items on the same table slot.
- Use the chaining technique for hashing by using TreeSet (instead of linked list) to chain items on the same table slot.
- Use the Coalesced hashing technique. This technique uses the concept of Open Addressing to find first empty place for colliding element by using the quadratic probing and the concept of Separate Chaining to link the colliding elements to each other through pointers (indices in the table). The deletion of a key is performed by linking its next entry to the entry that points the deleted key by replacing deleted entry by the next entry. See the following illustration as an example:

**Input** = {3, 12, 13, 25, 23, 51, 42}

**Hash Function** = (data % 10)

Test all the three hash table implementations empirically. Use small, medium, and large-sized data and hash tables in suitable sizes for testing. Perform different tasks over the tables to compare their performance results (like accessing existing/non-existing items or adding/removing items).

### **Restrictions**

- Can be only one main class in project
- Don't use any other third part library

### **General Rules**

- For any question firstly use course news forum in Moodle, and then the contact TA.
- You can submit assignment one day late and will be evaluated over sixty percent (%60).

### **Technical Rules**

- You must write a driver function that demonstrates all possible actions in your homework. For example, if you are asked to implement an array list and perform an iterative search on the list then. The driver function should run when the code file is executed.
- Implement clean code standards in your code ;
  - Classes, methods and variables names must be meaningful and related with the functionality.
  - Your functions and classes must be simple, general, reusable and focus on one topic.
  - Use standard java code name conventions.

### **Report Rules**

- Add all javadoc documentations for classes, methods, variables ...etc. All explanation must be meaningful and understandable.
- You should submit your homework code, Javadoc and report to Moodle in a "studentid\_hw5.tar.gz" file.
- Use the given homework format including selected parts from the table below:
  - Problem solutions approach
  - Test cases
  - Running command and results

### **Grading**

- No OOP design: -100
- No interface: -95
- No method overriding: -95
- No error handling: -50
- No inheritance: -95
- No polymorphism: -95
- No javadoc documentation: -50
- No report: -90
- Disobey restrictions: -100
- Cheating: -200
- Your solution is evaluated over 100 as your performance.

## **2.REPORT**

I detailed here what I did in my homework Part 1.

## 2.1.Problem Solutions Approach

**Note :** I googled the way you said in the report to problem solving approach, but I could not any useful article, what I found was either paid or long. After all I found a useful post from medium. I prepared this part according to this post. I hope I got it right.

- **Clearly understanding and/or defining the problem :**  
I understood and defined the problem clearly.
  - We should implement MapIterator
- **Breaking down large problems into smaller problems :**  
I broke down large problem MapIterator implementation to small problems EntryNode, MyList, MapIterator, and also a Test and a Main class.
  - We had a large problem,
  - I have now small problems.
- **Solving the problem at an abstract level first :**  
I solved the problem at an abstract level first.
  - I thought a lot about the problem.
  - I scribbled something about this subject in the ledger.
  - Something started to take shape in my head.
  - I used my knowledge of data structures, and it is.
- **Using notes and pseudo-code :**  
I noted what I thought, and wrote permanently pseudo-codes mixed with java codes.
  - If i understand or find something new, I noted.
  - I wrote pseudo-codes mixed with java codes, not clear.
- **Running code early and often :**  
I wrote real codes and run them often.
  - I turned my pseudo-codes into real java codes.
  - I coded them in ide and run them often.

## 2.2.Test Cases

### Testing Requirements

Test Case	Pres / Posts	Done
MapIterator implementation	<ul style="list-style-type: none"><li>• Implement MapIterator</li><li>• Iterate on a map that HashMap object</li></ul>	Done

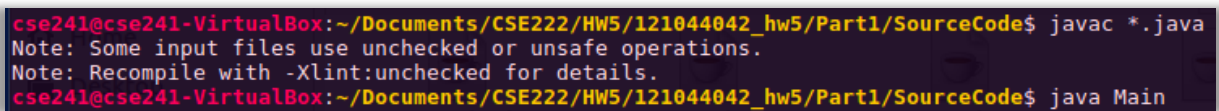
## 2.3. Running Commands and Results

### Compile and Run Commands, Testing Steps

Usage of my program :

1. Compile program with “**javac \*.java**” command
2. Run program with “**java Main**” command

### Simply Follow This Screenshot



```
cse241@cse241-VirtualBox:~/Documents/CSE222/HW5/121044042_hw5/Part1/SourceCode$ javac *.java
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
cse241@cse241-VirtualBox:~/Documents/CSE222/HW5/121044042_hw5/Part1/SourceCode$ java Main
```

## Commands and Results with Screenshots

### Hashing Performance Test

1. Program creates MapIterator objects
2. Adds entries
3. Prints entries as iterating on map

### Test Results are Here

```
cse241@cse241-VirtualBox:~/Documents/CSE222/HW5/121044042_hw5/Part1/SourceCode$ java Main
-----
MAPITERATOR TEST STARTED 1.8.0_275
-----
0:Value1 g ./EntryNode.html...
1:Value2 g ./Main.html...
2:Value3 g ./MapIterator.html...
2:Value4 or.java:76: warning: @return tag has no arguments.
2:Value5 g ./MyList.html...
2:Value6 g ./package-frame.html...
2:Value7 g ./package-summary.html...
2:Value8 g ./package-tree.html...
2:Value9 g ./constant-values.html...
2:Value10 index for all the packages and classes...
-----
MAPITERATOR TEST ENDED .html...
-----
```



## **END OF THE REPORT**

### **LAST UPDATE**

**May 12, 2021 Wednesday 23:30**

### **STUDENT**

**Şeyda Nur DEMİR  
12 10 44 042**

### **LECTURER**

**Prof. Dr. Fatih Erdoğan SEVİLGİN**

### **TEACHING ASSISTANT**

**Başak KARAKAŞ  
Mehmet Burak KOCA**

**KOCAELİ, 2021**