Assignment #7- Maps

This assignment may seem easy, but it requires strong OO design. You should read through the assignment to get a basic idea of the requirements and then start working through the assignment in the recommended order.

Objectives:

The main objectives of this assignment are:

- Implement MAP using four different ways
- Analyze the running time
- Practice code reusing, testing and debugging

The Assignment:

The Concept:

Implement MAP ADT with Unordered list, Sorted List(table), a Hash Table and a Binary Search Tree

- Unordered Map - Use ArrayList or DoublyLinkedList to store the data. Data in the list is unordered.
- SortedTable - Use ArrayList to store the data. Data in the array is ordered, binary search should be used to search for a key.
- HashTable - Use an array of Unordered Map to store the data. Pretty good at both insert and search. For this assignment, use separate chaining method for collision-handling.
- Binary Search Tree - Use the binary search tree to store entries.

Your Job:

Code up an unsorted map, sorted map, a hash table, and binary search tree. You are provided with some Junit sample test code. Please be reminded that you are responsible for creating test cases and testing the code yourself.

Getting Started (and Getting Finished):

Download and import the following zip file: Assignment5.zip. For Eclipse:

1. Open Eclipse
2. Choose "File → Import"

3. Select the "General" category
4. Pick "Existing Projects into Workspace"
5. Click on "Next"
6. Select the button for "Select Archive File" and browse to/select the zip file.
7. Click on "Finish"
8. A new project Map has been added to the workspace

The major files in the project Map

- AbstractMap.java
  - It is fully implemented.
  - It has an inner class called mapEntry that implements the interface Entry<K,V>
  - It has methods keySet() and values().
  - The four classes (UnorderedMap.java, SortedTable.java, HashMap.java, BinarySearchTree.java) extend AbstractMap.
- UnorderedMap.java
  - You need to implement this MAP ADT. Use equals() method to compare if two keys are same or not. For example: k1.equals(k2)
- HashMap.java
  - You need to implement this MAP ADT. Use equals() method to compare if two keys are same or not. For example: k1.equals(k2).
  - Please note that hashValue(K) method has been implemented for you.
    ```
    private int hashValue(K key) {
       return Math.abs(key.hashCode()) % capacity;
    }
    ```
- SortedTable.java
  - You need to implement this MAP ADT. Please note that generic type K extends Comparable, so you can use compareTo method to compare the order of keys, for example: K1.compareTo(K2) .

  ```
  public class SortedTable<K extends Comparable<K>, V> extends AbstractMap<K,V>
  ```

- LinkedBinaryTree.java
  - This is a generic binary tree that you need to implement, which will be used by BinarySearchTree.java.

- BinarySearchTree.java
  - You need to implement this MAP ADT. Please note that generic type K extends Comparable, so you can use compareTo method to compare the order of keys, for example: K1.compareTo(K2) .

Suggested order:

1. Copy the files for `ArrayList, DoublyLikedList, LinkedBinaryTree` implementation from previous project.

   If you didn't do well with the related previous assignments, please meet with the TA/instructor to get approval and instructions to import library cs2321util at least 3 days before the project due.

2. Complete the `UnorderedMap` using ArrayList or DoublyLinked List.
3. Complete the `SortedTable` using ArrayList
4. Complete the `HashMap` by using array of the fixed size as the hash table. Use Separate chaining to handle the collision. You need to check the load factor after each insert and implement the rehashing feature when the load factor is more than 0.75. The initial hash table size can be specified in the constructor method. If it is not specified, then choose a default size, like 17 as in the text book. As we insert more data into the hashtable, the load factor increases, the collision increases, and the performance degrades. So we will double the table size and rehash it once the load factor is more than 0.75. Finish UnorderedMap first before you start this one.

   a. You will use Division method to get the hash value. The hashValue() function has been implemented.
   b. Use the chaining scheme to handle collision.
   c. Each element in the array is an Unordered map.
5. Complete `BinarySearchTree`

Submission:

First, look at the following checklist:

1. Do not ever `import` from `java.util`. If so, be sure you only import allowed components (like `Iterator`, Exceptions, etc.). Unless the assignment specifically mentions it is permissible, you should never include any of java's native data structures.
2. Does the program meet all required interfaces? Did you test?
3. Time Complexity Analysis:
   - Do all your methods have a `@TimeComplexity` indicator? How about `@TimeComplexityExpected?`
   - Do you provide adequate `TCJ` comments to prove your time complexity?
4. Comments and Style:
   - Is the indentation easily readable? You can have Eclipse correct indentation by highlighting all code and select "Source → Correct Indentation".
   - Are comments well organized and concise?

If you have reviewed your code and all the criteria on the checklist are acceptable, follow the submission procedure.

Grading Criteria:

- An `UnorderedMap`, `SortedTable`, `HashMap`, and`BinarySearchTree` using the code given, correctly implementing all methods and interfaces: 80 (20 points each)
- Correct documentation of time complexity: 10
- Clear concise code with good commenting: 10
- Getting program done early: 0 points, but you feel good about yourself

FAQ

Q: How to get hash code of any object?
A: Use Math.abs(object.hashcode()).

Q: How to get the hashvalue?
A: The hashvalue (index value) can be calculated with the modular function hashcode % hashtablesize

Q:How do we handle collision for HashMap?
A:Use separate chaining.

Q: Why do we need comparator for Map like a comparator for Priority Queue?
A: No. But we do need to compare between keys. The most common comparison we need is equality comparison. But for sorted map, like the SortedTable, beside the equality comparison, the binary search algorithm need to compare where the middle key is smaller or bigger. Since the key extends Comparable, use compareTo() method.

Q: Is the hashtable fixed size?
A: No. Once the load factor is more than 0.75, it needs to be doubled.

Q: How to instantiate an array of UnorderedMap<K,V>?
A:

```
private UnorderedMap<K,V>[ ] table;
table = (UnorderedMap<K,V>[ ]) new UnorderedMap[capacity]
```