COP3330

University of Central Florida

**Assignment 4**

In this assigment you will implement a Hash Table in Java. All of the java files will be inside assignment4 package. Unless otherwise stated, all the methods are public and all instance variables are private. There are a few examples where methods should be private.

**Part 1: Record.java file**

Create a Record class. It should have an int *number* and String *value* as instance values. Create a no argument constructor as you like. Create a constructor which will take number and value as parameters. Create getter and setter for those.variables. Instance variables should be private. Also write toString method which returns *number:value* . Number should have a colon immediately after it. Leave no space after or before the colon.

**Part 2: ArrayListAlt.java file**

Create an ArrayListAlt class which will be an alternative Arraylist. That arraylist will store Records in sorted manner (ascending) according to the number in the record object. It will have an ArrayList object as instance variable which will hold Records.

The method descriptions are as follows:

Create a no argument constructor which creates an empty arraylist.

void add(Record *r*) Adds a Record object to the sorted ArrayListAlt. You need to be adding the new record to appropriate place in the arraylist as the arraylist will store the elements sorted in ascending manner. If a record with the same number exists in the ArrayList, change the value of that record using value of *r* parameter. Hint: Use find() method to find the position of the element

boolean remove(int num) Removes a record from the ArrayListAlt whose number matches with the given num. If no such element exists, it will return false and *throw* exception with message “No such element”. Hint: Use find() method to find the position of the element

Override toString so that it returns a string representation of the ArrayListAlt.

int size() returns the size of the ArrayListAlt

get(int number) returns the value of a record ( as String) from the ArrayListAlt based on the provided number. It will return null if no such element exists. Hint: Use find() method to find the position of the element

private find(int number) method finds the position of a record in the ArrayListAlt based on the provided number. Use binary search to search the element. That method will be *private*. If the record does not exist, this method should return -1.

**Part 3 MyHashTable.java file**

Create a MyHashTable class with the following methods inside. The class should have an array of ArrayListAlt objects as an instance variable which will be private.

Create a constructor which will take size of the array as an int parameter.

Create a no argument constructor which will set the size of the array to 11

private int hashFunc(int position) will return the remainder of the position divided by the array size. Note that this method should be private.

void add(Record a) will add the Record object to the hash table appropriately. Note that the instance variable array contains ArrayListAlt objects.

void remove(int number) removes a record from the hash table using the provided number. If no such element exists it will print “No such element” ( without the quotation marks). It should not print the stack trace when it faces with an exception. It will only print the message.

String get(int number) Retrieves a record from the hash table based on the provided number. If no such element, the method will return null.

Override the toString method so that it will print the hash table, showing all the records.

**Part 4 HashTableTester.java**

You will take the name of the data file as a command line parameter. You will create a myHashTable object and do the actions stated in the file. Then you will print the resulting myHashTable.

An example data file is below:

The first line will have a single number showing the size of the array inside the hashTable class.

The other lines will have either add or remove. If “add”, then the number and value in a record will be written. If “remove”, only the number will be written. You can add some more methods to Record to parse the data easier if you prefer. The data file will always exists and will contain no syntax errors/problems. The data file will always contain positive numbers.

Example data file

7

add 12:Test

add 22:Test1

add 26:Test5

add 97:Test2

add 19:Test3

add 22:Test4

remove 9

Output of the program which is parsing the data file is as follows

No such element

0:

1: [22:Test4]

2:

3:

4:

5: [12:Test, 19:Test3, 26:Test5]

6: [97:Test2]

As you see the output has *No such element* at the beginning because record number with 9 does not exist, therefore cannot be removed. Also you can see that the entry with 22 is changed to Test4 once it is added again to the table. You can also see that the 12,19 and 26 record numbers are stored in sorted manner in the 5th position.