CSC 220 Data Structures

Program #9

Dictionaries

Description:

Your objective is to implement your own Dictionary class that uses a hash table as the backing data structure (as explained in the lecture notes). This table will contain an array of KeyValuePair objects (which each contain a key and a value). Your initial table size should be 7 buckets. Specifically you are to do the following:

Create a public **generic** class called Dictionary. This class will contain two generic types, one for the key and one for the value of each KeyValuePair object. Refer to the given KeyValuePair class to see how to do two generic types for a single class. This Dictionary class should have the following:

Fields

- An array of KeyValuePair objects that we will use as our hash table
- A special "deleted" KeyValuePair object to use for marking an entry as being deleted. This can be defined as:

```
private final KeyValuePair<K,V> DELETED =
                                    new KeyValuePair<K,V>(null, null);
```

Methods

`insert(key, value)`	inserts the specified (key, value) pair into the dictionary (as a KeyValuePair object). If key is already found, value is overwritten
`search(key)`	searches for the given key in the dictionary and returns the corresponding value if found, null otherwise
`delete(key)`	deletes the (key, value) pair from the dictionary

You may create other fields and methods if you want, but you must at least have the ones outlined above.

For collision handling, you must use linear probing from the notes.

For hashing use the following:

- The prehash function will be Java's built-in hashCode() function called on the key.
- The hash function will be the division method from the notes.

About Generics:

It is good practice to always declare, initialize and instantiate using the generic parameters for generic classes. Failure to do so tells Java you want to use what's called "raw types". Raw types can cause issues and is best to be avoided. Therefore, **I will be examining your code to make sure you use the generic type parameters in all necessary places** to avoid the unintentional use of raw types. The only exception

is when allocating your hash table. Java has a weird quirk that doesn't allow this unless you allocate it as an array of KeyValuePair objects without the generic parameters and then cast it like so:

```
(KeyValuePair<K,V>[]) new KeyValuePair[7];
```

Failure to allocate the array like that will result in an error in Java. This assumes that your dictionary uses the same K and V type parameters as your KeyValuePair class, but of course you can rename those if you like.

Note that this will create a warning when you compile your code. The warning will look something like this:

```
Note: ./Dictionary.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

You can safely ignore this warning. I will not take off points for this warning popping up.

Entry Point:

Create a public class called Main that contains your entry point. The entry point should create an object of your Dictionary class with String as the datatype for its keys and Integer as the datatype for its values. This dictionary will hold the salary of several employees. The keys are names of the employees and the values are the salaries (in increments of 1,000). After declaration and instantiation, insert the following keys and values using the insert function:

KEY	VALUE
Bob	50
Bill	120
Roger	80
Kevin	350
Jerry	65
Liam	500

Now use the search function to find each employee by name and print out their salary (refer to the sample output below). After printing them out, delete the entries for "Bob", "Roger" and “Jerry”. Also call the insert function with “Liam” as the key and 650 as the value. Remember insert should replace the value in the table if it finds the key is already there. Print out the salaries again and this time, if the search function ever returns null, print out that that employee got fired. You must actually check the results of your search function for null and not just assume Bob, Roger and Jerry were fired. Again refer to the sample output below for clarity.

For Submission:

Put all of your .java files into a zip folder and submit the zip. Do not submit any .class files.

Sample Output:

```
$ javac Main.java && java Main
Bob makes $50k a year
Bill makes $120k a year
Roger makes $80k a year
Kevin makes $350k a year
Jerry makes $65k a year
Liam makes $500k a year

Bob got fired
Bill makes $120k a year
Roger got fired
Kevin makes $350k a year
Jerry got fired
Liam makes $650k a year
```

Bonus:

For an extra 9 points, allow your hash table to resize. Change the initial size to only 2 buckets. When you can't find an empty bucket for inserting a new entry, resize your table by doubling its current size and rehashing all key, value pairs. Then try to insert the new key and value again. You must use the same collision handling, prehash and hash methods as before (albeit with the new table size). Your output should be exactly the same as before.

Rubric:

#	ITEM	POINTS
1	insert method	8
2	search method	8
3	delete method	8
4	correct handling of generics	8
5	output matches	8
	TOTAL	**40**

#	PENALTIES	POINTS
1	Doesn’t compile	-50%
2	Doesn’t execute once compiled (i.e. it crashes)	-25%
3	Late up to 1 day	-25%
4	Late up to 2 days	-50%
5	Late after 2 days	-100%