Lab 12: Unordered Tables

In this lab you will create an `UnorderedTable`. You will use it to implement a very basic restaurant program.

The file `menu.tsv` contains the menu of a certain restaurant. Each line represents one item on the menu. A line contains the item's name, followed by a tab, and then the cost of the item. (The cost is in cents. This is to avoid the rounding errors inherent in using doubles.)

Your data structure should be able to automatically resize itself, in order to hold the necessary key-value pairs. It is your choice how to handle this. You could simply make an `ArrayList` of each one, or you could handle the growing and shrinking yourself. (In this case, you'll have two arrays. When the size is about to exceed the capacity, double them. When the size decreases to one quarter the capacity, halve them.)

The first object you make must be the `UnorderedTable` object. This will use generics: `K` representing the key, and `V` representing the value. Your class should be declared like this:

```
public class UnorderedTable<K,V> {
```

It should have nothing in it to suggest that it was made for a specific application (i.e. no restaurant-specific stuff, just unordered-table stuff). Your public methods must be as follows:

- `UnorderedTable()`, the constructor. It will set up the array(s) or `ArrayList`(s).
- `void clear()`, which deletes all key-value pairs from the table.
- `boolean delete(K key)`, which removes a key and its corresponding value. If the key isn't in the table, do nothing and return false. If it is, delete it and return true.
- `V get(K key)`, which returns the value to which the key maps. If the key is not contained in the table, return null.
- `int getSize()`, which returns the number of key-value pairs there are in the table.
- `void put(K key, V value)`, which puts the key-value pair into the table. If key already mapped to something else, the old value should be replaced with the new one.
- A unit-testing `main()` method to test all the above methods, in a variety of situations.

You may have other private methods if you wish.

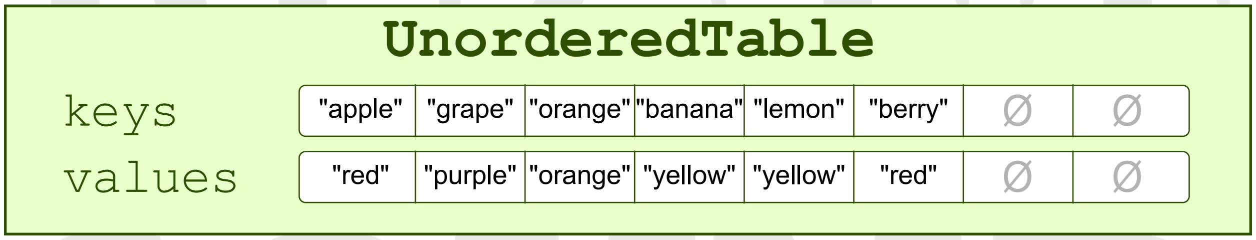
- If the user enters a food item, it will add that item's cost to the user's subtotal, and display the subtotal.
- If the user hits enter, it will calculate tax (10%) and tip (15%) (each rounded to the nearest cent), print them separately, add them to the subtotal, print the total, and exit. All money amounts should be formatted correctly (e.g. print $4.00, not 400).
- If the user enters anything else, it will apologize and say that it didn't understand.

If you're having trouble displaying prices properly, look up the `printf()` method in the Java `PrintWriter` specs. `PrintWriter` is the kind of object that `System.out` and `System.err` are. You will see the familiar `print()` and `println()` methods, as well as `printf()`. The arguments to printf are a `String` containing special formatting symbols (that begin with %), followed by a variable number of arguments that are the numbers to print out. For example, this:

```
System.out.printf("$%d.%02d", dollars, cents);
```

Will print out dollars and cents as you expect—so long as you've defined two ints named dollars and cents. Here `%d` means print out some integer, and `%02d` means print out an integer padded with 0s so that it's two digits long. Because there were two special % codes in the first `String`, there are two more arguments, each holding one of the values to print.

The two classes you turn in should be called `Restaurant` and `UnorderedTable`.


UnorderedTable
keys
"apple"
"grape"
"orange"
"banana"
"lemon"
"berry"
∅
∅
values
"red"
"purple"
"orange"
"yellow"
"yellow"
"red"
∅
∅