Binary Search Tree

Goal: Can write and use a linked implementation of a binary search tree.

Problem Description

In this assignment, you will practice solving a problem using a binary search tree. You will implement a Java application that could be used in a restaurant. You are asked to implement four classes: MenuItem, BSTNode, Order, and RestaurantDriver. Each of these classes is described below. (The method names, parameters, and return types must match the specification exactly. You may add other private helper methods if you wish.)

MenuItem Class

A MenuItem represents one item ordered from a menu. Implement the following:

1. Include exactly three instance variables:
   - name
   - price
   - quantity
2. Implement a constructor that accepts three values and uses those values to initialize the instance variables.

3. Implement getters and setters for all the instance variables.

4. Implement a toString() method that returns a nicely formatted String description of the item. In addition to name, price, and quantity, calculate the total value of menuItem and add it to the end of the String. For example:

   Rice $1.50 3 $4.50

   Use DecimalFormat to display prices with exactly two digits following the decimal place.

5. Implement the equals() method where two MenuItems are considered equal when they have the same name. Note that, the equality of String attributes should be case insensitive.

6. Implement the Comparable interface and the compareTo() method for MenuItem. Compare alphabetically by name.

BSTNode Class

This class represents a node in a binary search tree. This will be a generic implementation. (In other words, this binary search tree node can hold any type of object). You must create your own class. You may not use any Node classes from the Java library. Implement the following:

1. Three private instance variables:
   a. data
   b. left
   c. right
2. A constructor that takes three input parameters and uses them to initialize the three instance variables.
3. Getters and setters for the instance variables.

Order Class

Implement a collection class of MenuItems using a binary search tree. You MUST store the data in order according to the binary search tree principles.

- You must create your own binary search tree and implement the recursive methods you need.
- You may NOT use any tree classes from the Java library.
- Use a linked implementation of the binary search tree (not an array implementation)
- No method in the collection class should print anything except inorder, preorder, and postorder.
- All methods MUST use recursion unless marked as "Not recursive"

Your collection class must have the following:

1. Your class has exactly two instance variables:
    - root of type BSTNode: this represents the root of the binary search tree that stores your collection. This MUST be a PRIVATE variable. You may NOT create a getter for this variable.
    - tableNumber: a string representing the identification of the table

2. Your class should also have one static variable:
    - name: the name of the restaurant (you choose the name)

3. Implement a constructor for your collection class that accepts the table number, uses that to initialize the appropriate variable. This method instantiates an empty binary search tree. Not recursive.

4. **`void insert(MenuItem)`:** a method that takes one input parameter, a MenuItem, and adds it to the binary search tree in the proper spot, preserving the binary search tree property. If the MenuItem is already in the tree, insert() will not add a new item to the tree, but rather will increment the quantity of the existing item by the new quantity. If the MenuItem is not already in the tree, the method inserts the MenuItem in its correct position.

5. **`void preorder()`:** traverses the tree in pre-order (based on the name) and prints the contents of each Node.

6. **`void inorder()`:** traverses the tree in order (based on the name) and prints the contents of each Node.

7. **`void postorder()`:** traverses the tree in post-order (based on the name) and prints the contents of each Node.

8. **`int size()`:** a method that returns the number of MenuItems in the collection. You do NOT have a manyNodes instance variable, so this is going to be more work than it was in the past.

9. **`int depth()`:** returns the height (depth) of the tree. Note: An empty tree has a height of -1. A tree with one node (the root node of the tree) has a height of 0.

10. **`int getTotalQty()`:** returns the sum of the quantities of all the MenuItems in the collection.

11. **`MenuItem search(String):`** traverses the tree and returns a reference to a MenuItem with a search key (name) that matches the parameter. Returns null if not found.

12. **`double getTotalBeforeTax():`** calculates and returns the total value of all items in the tree. Remember to multiply the price by the quantity to get the total for each item.

13. **`double getTax(double):`** a method with one parameter (tax percent). Calculates and returns the amount of tax due on the order. You do not pay tax on the tip. Not recursive.

14. **`double getTip(double):`** a method with one parameter (tip percent). Calculates and returns the amount of tip to be paid on the order's total before tax. Not recursive.

15. **`String toString():`** a method that returns a string representation of all the MenuItems in the tree. The items will be displayed in alphabetical order. The String includes the following:

    - The restaurant name, table number, and some header lines
    - Details of all the MenuItems, one per line
    - The total of all the items in the tree before tax and tip have been added
    - The amount of tax to be added (use 8% when calling getTax())
    - The amount of tip to be added (use 20% when calling getTip())
    - The total after tax and tip have been added

    Sample of String returned from toString():

```
Downtown Café     Table 12
-------------------------------------------------------------------------------
Item                Price           Qty             Total
-------------------------------------------------------------------------------
Injera              $2.50           1               $2.50
Rice                $1.50           3               $4.50
Sambusa             $5.80           4               $23.20
Sushi               $8.25           2               $16.50
-------------------------------------------------------------------------------
Total:          $46.70
Tax:            $3.74
Tip:            $9.34
-------------------------------------------------------------------------------
Grand total: $59.78
```

RestaurantDriver Class

Implement a `Driver` class that includes a test program to:

- Create two orders.
- Add 8-15 menu items to the orders. Change the order of insertion to test different tree topologies. Do not add the items in alphabetical order or your tree will degenerate to a linked list.
- The driver tests all the methods of the collection class. Display the results of each operation after it is performed. Label your output clearly.
- Print both orders in a nicely formatted output.
- Do not use a `Scanner` to read any inputs from the user. Instead, use hard coded values for the different methods' inputs.