CSC 260-01/Spring 2011 Instructor: Beifang Yi

Challenge Project 8

(**Due by 3/23/Wednesday-Midnight at Moodle**)

Your name:	Score:

22.16 Modify Figs. 22.17 and 22.18 to allow the binary tree to contain duplicates.

22.17 Write a program based on the program of Figs. 22.17 and 22.18 that inputs a line of text, tokenizes the sentence into separate words (you might want to use the StreamTokenizer class from the java.io package), inserts the words in a binary search tree and prints the inorder, preorder and postorder traversals of the tree.

22.19 Modify Figs 22.17 and 22.18 so the Tree class provides a method getDepth that determines how many levels are in the tree. Test the method in an application that inserts 20 random integers in a Tree.

22.20 (*Recursively Print a List Backward*) Modify the List<T> class of Fig. 22.3 to include a method printListBackward that recursively outputs the items in a linked list object in reverse order. Write a test program that creates a sorted list of integers and prints the list in reverse order.

22.21 (*Recursively Search a List*) Modify the List<T> class of Fig. 22.3 to include a method searchList that recursively searches a linked list object for a specified value. Method searchList should return a reference to the value if it is found; otherwise, null should be returned. Use your method in a test program that creates a list of integers.The program should prompt the user for a value to locate in the list.

22.22 (*Binary Tree Delete*) In this exercise, we discuss deleting items from binary search trees. The deletion algorithm is not as straightforward as the insertion algorithm. Three cases are encountered when deleting an item—the item is contained in a leaf node (i.e., it has no children), the item is contained in a node that has one child or the item is contained in a node that has two children.
If the item to be deleted is contained in a leaf node, the node is deleted and the reference in the parent node is set to null.

If the item to be deleted is contained in a leaf node, the node is deleted and the reference in the parent node is set to null.

If the item to be deleted is contained in a node with one child, the reference in the parent node is set to reference the child node and the node containing the data item is deleted. This causes the child node to take the place of the deleted node in the tree.

The last case is the most difficult. When a node with two children is deleted, another node in the tree must take its place. However, the reference in the parent node cannot simply be assigned to reference one of the children of the node to be deleted. In most cases, the resulting binary search tree would not embody the following characteristic of binary search trees (with no duplicate values):
The values in any left subtree are less than the value in the parent node, and the values in any right subtree are greater than the value in the parent node.

Which node is used as a *replacement node* to maintain this characteristic? It is either the node containing the largest value in the tree less than the value in the node being deleted, or the node containing the smallest value in the tree greater than the value in the node being deleted. Let us consider the node with the smaller value. In a binary search tree, the largest value less than a parent's value is located in the left subtree of the parent node and is guaranteed to be contained in the rightmost node of the subtree. This node is located by walking down the left subtree to the right until the reference to the right child of the current node is null. We are now referencing the replacement node, which is either a leaf node or a node with one child to its left. If the replacement node is a leaf node, the steps to perform the deletion are as follows:

a) Store the reference to the node to be deleted in a temporary reference variable.
b) Set the reference in the parent of the node being deleted to reference the replacement node.
c) Set the reference in the parent of the replacement node to null.
d) Set the reference to the right subtree in the replacement node to reference the right subtree of the node to be deleted.
e) Set the reference to the left subtree in the replacement node to reference the left subtree of the node to be deleted.

The deletion steps for a replacement node with a left child are similar to those for a replacement node with no children, but the algorithm also must move the child into the replacement node's position in the tree. If the replacement node is a node with a left child, the steps to perform the deletion are as follows:

a) Store the reference to the node to be deleted in a temporary reference variable.
b) Set the reference in the parent of the node being deleted to reference the replacement node.
c) Set the reference in the parent of the replacement node to reference the left child of the replacement node.
d) Set the reference to the right subtree in the replacement node to reference the right subtree of the node to be deleted.

e) Set the reference to the left subtree in the replacement node to reference the left subtree of the node to be deleted.

Write method deleteNode, which takes as its argument the value to delete.Method deleteNode should locate in the tree the node containing the value to delete and use the algorithms discussed here to delete the node. If the value is not found in the tree, the method should print a message that indicates whether the value is deleted. Modify the program of Figs. 22.17 and 22.18 to use this method. After deleting an item, call the methods inorderTraversal, preorderTraversal and postorderTraversal to confirm that the delete operation was performed correctly.

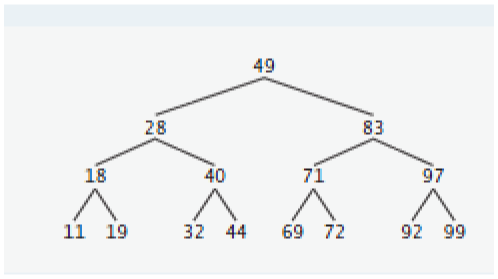
22.23 (*Binary Tree Search*) Modify class Tree of Fig. 22.17 to include a method binaryTreeSearch, which attempts to locate a specified value in a binary-search-tree object. The method should take as an argument a search key to be located. If the node containing the search key is found, the method should return a reference to that node; otherwise, it should return a null reference.

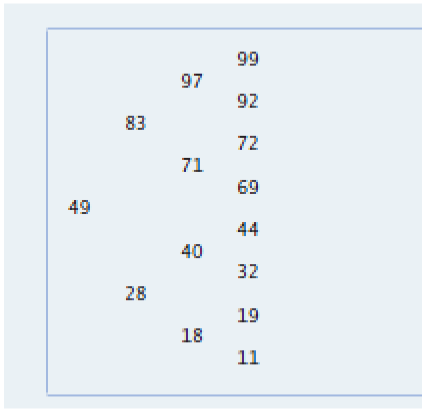
22.24 (*Level-Order Binary Tree Traversal*) The program of Figs. 22.17 and 22.18 illustrated three recursive methods of traversing a binary tree—inorder, preorder and postorder traversals. This exercise presents the *level-order traversal* of a binary tree, in which the node values are printed level by level, starting at the root node level. The nodes on each level are printed from left to right. The levelorder traversal is not a recursive algorithm. It uses a queue object to control the output of the nodes. The algorithm is as follows:

```
a) Insert the root node in the queue.
b) While there are nodes left in the queue, do the following:
    Get the next node in the queue.
    Print the node’s value.
    If the reference to the left child of the node is not null:
        Insert the left child node in the queue.
    If the reference to the right child of the node is not null:
        Insert the right child node in the queue.
```

Write method levelOrder to perform a level-order traversal of a binary tree object. Modify the program of Figs. 22.17 and 22.18 to use this method. [*Note:* You will also need to use queueprocessing methods of Fig. 22.13 in this program.]

22.25 (*Printing Trees*) Modify class Tree of Fig. 22.17 to include a recursive method outputTree to display a binary tree object on the screen. The method should output the tree row by row, with the top of the tree at the left of the screen and the bottom of the tree toward the right of the screen. Each row is output vertically. For example, the binary tree illustrated on the left figure of the following is output as shown on the right below:





```
               99
          97
               92
     83
               72
          71
               69
49
               44
          40
               32
     28
               19
          18
               11
```

The rightmost leaf node appears at the top of the output in the rightmost column and the root node appears at the left of the output. Each column starts five spaces to the right of the preceding column. Method outputTree should receive an argument totalSpaces representing the number of spaces preceding the value to be output. (This variable should start at zero so that the root node is output at the left of the screen.) The method uses a modified inorder traversal to output the tree—it starts at the rightmost node in the tree and works back to the left. The algorithm is as follows:

```
While the reference to the current node is not null, perform the following:
    Recursively call outputTree with the right subtree of the current node
    and totalSpaces + 5.
    Use a for statement to count from 1 to totalSpaces and output spaces.
    Output the value in the current node.
    Set the reference to the current node to refer to the left subtree of
        the current node.
    Increment totalSpaces by 5.
```

Your scores for this project vary (0~N) depending on how much you have completed.—Please create a SUBFOLDER with meaning name for EACH of the programming questions.
(This assignment is based on Deitel's book Version 8 Chapter 22)—sample code is available from the coursewebsite.

***************************Important***************************
Your code must follow Java Coding Convention
****